Université des sciences et de la technologie d'Oran – MB
Faculté des Mathématiques et de l'Informatique - Département d'informatique
2ème Année LMD – S3

# Architecture des ordinateurs

## Cours & exercices corrigés


**Support de cours préparé pour les étudiants inscrits en 2ème Année LMD – S3**

Diplôme préparé : **Licence académique**

Spécialités :
**Systèmes Informatique (SI)**

**Ingénierie des Systèmes d'Information et du Logiciel (ISIL)**

**Année universitaire 2020-2021**

# 1. Introduction

L'architecture d'un système à microprocesseur représente l'organisation de ses différentes unités et de leurs interconnexions. Le choix d'une architecture est toujours le résultat d'un compromis :

- entre performances et coûts
- entre efficacité et facilité de construction
- entre performances d'ensemble et facilité de programmation

Ce cours est assuré pour les étudiants inscrits en 2ème année Licence en Informatique, au niveau de l'université des sciences et de la technologie d'Oran. Il représente une récolte de ce qui a été fait durant le troisième semestre (S3) concernant la matière « Architecture des Ordinateurs » en séances de cours, de travaux dirigés et de travaux pratiques. Il contient les principales notions ainsi qu'un ensemble d'exemples illustratifs.

Un bouquet d'exercices corrigés est, également, offert aux étudiants pour qu'ils puissent avoir une idée sur les différents types d'exercices pouvant être traités pour cette matière.

# 2. Objectif du cours

Ce manuscrit a pour objectif de mettre en clair le principe de fonctionnement de l'ordinateur avec une présentation détaillée de l'architecture de l'ordinateur. Il offre aux étudiants inscrits en 2ème année, Licence en Informatique, la possibilité d'acquérir de nouvelles connaissances de programmation en langage « Assembleur », et ce conformément au programme du CPND.

# 3. Structure du polycopié

Ce document est organisé en deux parties : une première partie pour le cours contenant cinq chapitres relativement au plan de la matière « Architecture des ordinateurs » du programme CPND, où chaque chapitre est enrichi par un ensemble d'exemples et une deuxième partie contenant des exercices corrigés, relativement aux différents chapitres vus en cours.

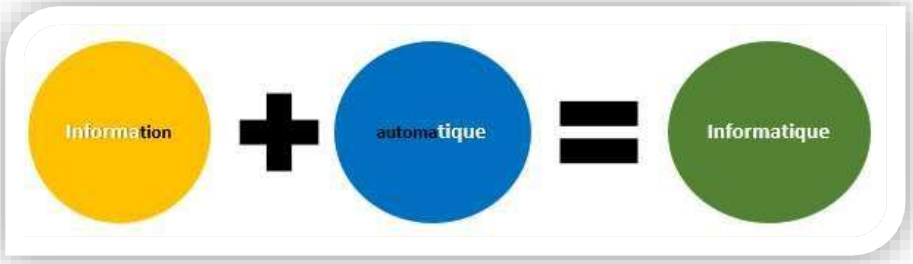
# Partie I – Cours

# Chapitre 1 : Structure de base d'un ordinateur



## 1. Introduction à la notion d'architecture des ordinateurs

Le mot *informatique* vient de la contraction des mots **information** et **automatique.**



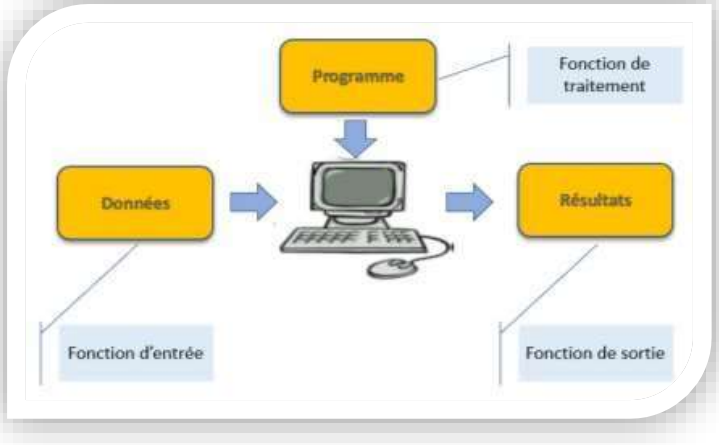

### 1.1. Qu'est-ce qu'un ordinateur ?

« Machine automatique de traitement de l'information obéissant à des programmes formés par des suites d'opérations arithmétiques et logiques » (Larousse 2002)

Un *ordinateur* est une machine de traitement de l'information (texte, image, signal, vidéo).

Il est capable d'**acquérir** de l'information, de la **stocker**, de la **transformer** en effectuant des traitements quelconques, puis de la **restituer** sous une autre forme :

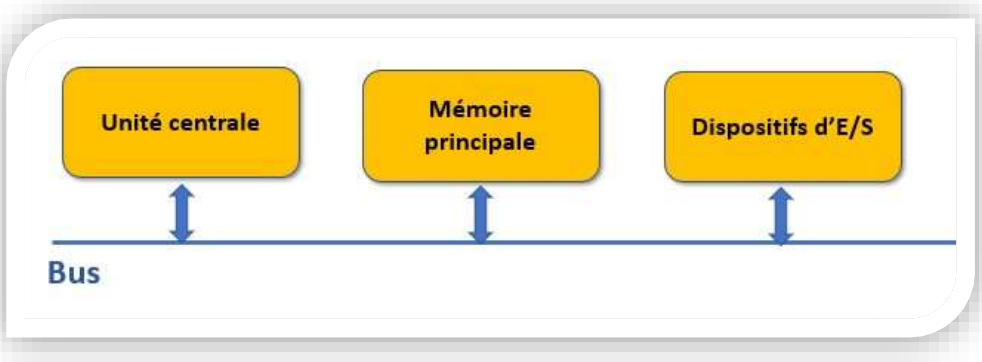
- Il peut recevoir des données en entrée ⇒ «*Fonction d'entrée*»,
- Effectuer sur ces données des opérations en fonction d'un programme ⇒ «*Fonction de traitement*»
- Et enfin fournir des résultats en sortie ⇒ «*Fonction de sortie*»




Un ordinateur est formé de trois composants :

1. Le processeur (ou UC unité centrale, ou *CPU* pour *Central Processing Unit*)
2. Les mémoires
3. Les dispositifs d'entrée-sortie

Ces éléments étant interconnectés entre eux par des bus.




Cela constitue l'architecture de base de tout ordinateur.

### 1.2. Qu'appelle-t-on architecture des ordinateurs ?

***L'Architecture des ordinateurs*** est un domaine de l'informatique centré sur les machines (du point de vue à la fois matériel et logiciel).

**L'*architecture des ordinateurs*** est l'étude et la description du fonctionnement des composants internes d'un ordinateur. Elle traite :

- Le type des informations manipulées et leur codage,
- Le dialogue entre composants,
- Le fonctionnement logique (pas électronique) interne des composants.

## 2. La machine de Von Neumann et la machine Harvard

On peut distinguer deux architectures principales qui sont :

### 2.1. L'architecture de Von Neumann

Cette architecture est appelée ainsi en référence au mathématicien *John Von Neumann* qui a élaboré en **juin 1945** dans le cadre du projet **EDVAC**[1], la première description d'un ordinateur dont le programme est stocké dans sa mémoire.

*L'architecture de Von Neumann* décompose l'ordinateur en quatre parties :

1. **Unité Arithmétique et Logique (UAL)** : effectue les calculs (les opérations de base)
2. **Unité de contrôle (UC) :** commande les autres unités ; qui est chargée du séquençage des opérations
    a. Envoie des signaux de contrôle aux autres unités
    b. Supervise le fonctionnement de l'UAL
    c. Envoie des signaux d'horloge aux autres unités…
3. **Mémoire** : dispositif de stockage de données et programme
4. **Dispositifs d'Entrée-Sortie** : permettent l'échange d'informations avec les dispositifs extérieurs

Les différents organes du système sont reliés par des voies de communication appelées bus (Bus d'adresse et bus de données).

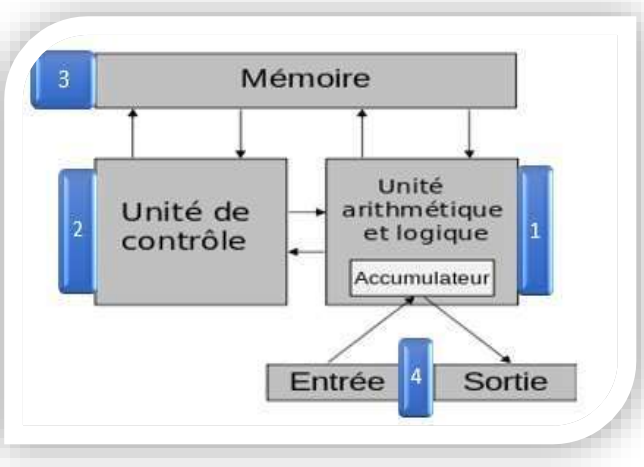



**Schéma de l'architecture de Von Neumann.**

*L'architecture de Von Neumann* est un modèle pour un ordinateur qui utilise une structure de **stockage unique** pour conserver **à la fois les instructions et les données** demandées ou produites par le calcul. De telles machines sont aussi connues sous le nom d'ordinateur à programme enregistré. La séparation entre le *stockage* (Mémoire) et le *processeur* (Unité de contrôle et Unité arithmétique et logique) est implicite dans ce modèle.

## 2.2. L'architecture de Harvard

Le nom de cette structure vient du nom de l'université de *Harvard* où une telle architecture a été mise en pratique pour la première fois avec le **Harvard Mark 1** créé par *Howard Aiken* et fut construit par IBM en **1944**. (Également appelé IBM Automatic Sequence Controlled Calculator (ASCC)).

Avec **deux bus** distincts, *l'architecture dite de Harvard* permet de transférer simultanément les **données** et les **instructions** à exécuter. Ainsi, l'unité de traitement aura

[1] EDVAC (*Electronic Discrete Variable Automatic Computer*) est l'un des tout premiers ordinateurs électroniques. Il opère en mode binaire contrairement à l'ENIAC, qui opère en décimal.

accès simultanément à l'instruction et aux données associées. Cette architecture peut se montrer plus rapide à technologie identique que *l'architecture de Von Neumann* ; le gain en performance s'obtient cependant au prix d'une complexité accrue de structure.

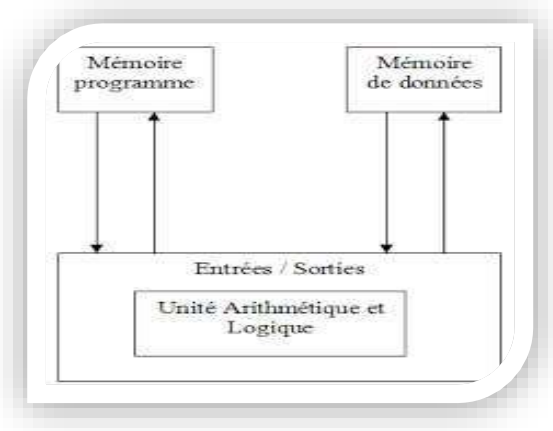



**Schéma de l'architecture Harvard.**

*L'architecture Harvard* est généralement utilisée dans des systèmes spécialisés ou à des usages très spécifiques. Elle est utilisée dans le traitement du signal numérique spécialisé (DSP Digital Signal Processing), typiquement pour le traitement des données vidéo et audio.

Elle est, également, utilisée dans de nombreux petits microcontrôleurs utilisés dans des applications de l'électronique.

## 2.2. La différence entre l'architecture de Von Neumann et l'architecture de Harvard

| Architecture de Von Neumann | Architecture de Harvard |
|---|---|
| Porte le nom du mathématicien et informaticien **John Von Neumann.** | Le nom provient de «**Harvard Mark I**», un ancien ordinateur à relais, projet réalisé à l'université Harvard |
| **Une seule mémoire** pour les instructions et les données. | Elle a besoin de **deux mémoires** pour les instructions et les données. |
| La **conception** est **simple.** | La **conception** est **compliquée.** |
| Ne requiert qu'**un seul bus** pour les instructions et les données. | Nécessite **deux bus séparés**, un pour les instructions et un pour les données. |
| Le processeur a besoin de **deux cycles d'horloge** pour terminer **une instruction**. | Le processeur peut compléter **une instruction** en **un cycle.** |
| **Faible performance** par rapport à l'architecture de Harvard. | Plus facile à canaliser, donc de **hautes performances** peuvent être atteintes. |
| **Coût moins cher.** | **Coût relativement élevé** |

Dans *l'architecture de Von Neumann* le processeur a besoin de **deux cycles d'horloge** pour exécuter une instruction, il lit d'abord l'instruction (mémoire programme) après il accède à la donnée (mémoire donnée) car il n'y a qu'une seule mémoire.
Tandis que dans *l'architecture de Harvard* le processeur prend **un cycle d'horloge** pour compléter une instruction, il peut lire une instruction et accéder la donnée en même temps car les deux mémoires sont séparées.
Actuellement, la plupart des ordinateurs sont des machines de Von Neumann, seules les technologies ont changé.

## 3. Conclusion

Nous avons présenté dans ce chapitre une introduction à la notion d'architecture des ordinateurs et aux principales architectures utilisées pour concevoir des ordinateurs, à savoir l'architecture de Von Neumann et celle de Harvard.

Nous présentons dans le chapitre suivant les principaux composants d'un ordinateur, tels que le processeur, l'UAL, le bus, les mémoires et autres.

# Chapitre 2 : Principaux composants d'un ordinateur



## 1. Introduction aux principaux composants d'un ordinateur

Un ordinateur est une machine programmable universelle de traitement de l'information.

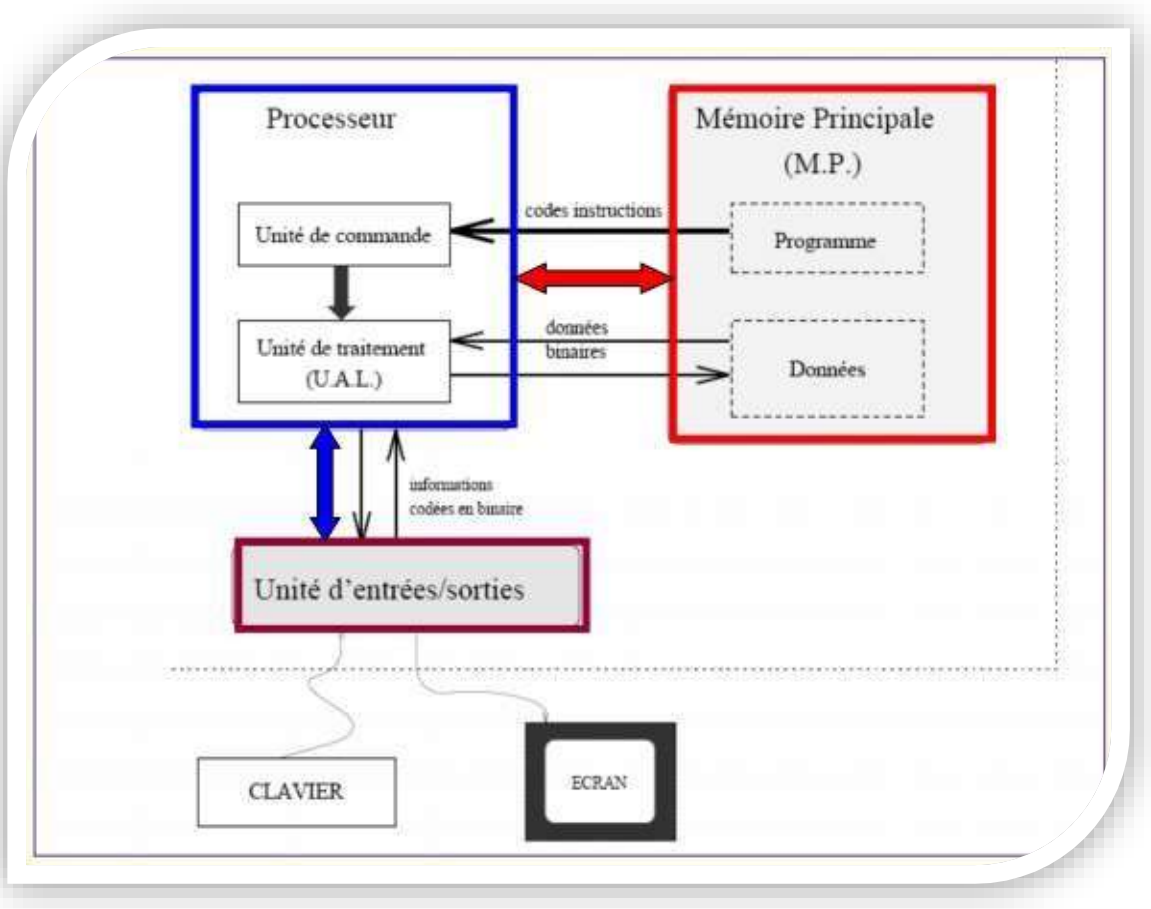
Pour accomplir sa fonction, il doit pouvoir :

- **Acquérir** de l'information de l'extérieur
- **Stocker** en son sein ces informations
- **Combiner** entre elles les informations à sa disposition
- **Restituer** ces informations à l'extérieur

L'ordinateur doit donc posséder :

- **Une ou plusieurs unités de stockage**, pour mémoriser le programme en cours d'exécution ainsi que les données qu'il manipule.
- **Une unité de traitement** permettant l'exécution des instructions du programme et des calculs sur les données qu'elles spécifient.
- **Différents dispositifs** '**périphériques**' servant à interagir avec l'extérieur : clavier, écran, souris, carte graphique, carte réseau, etc.

Les constituants de l'ordinateur sont reliés par un ou plusieurs bus, ensembles de fils parallèles servant à la transmission des adresses, des données, et des signaux de contrôle.




**L'architecture de base d'un ordinateur**

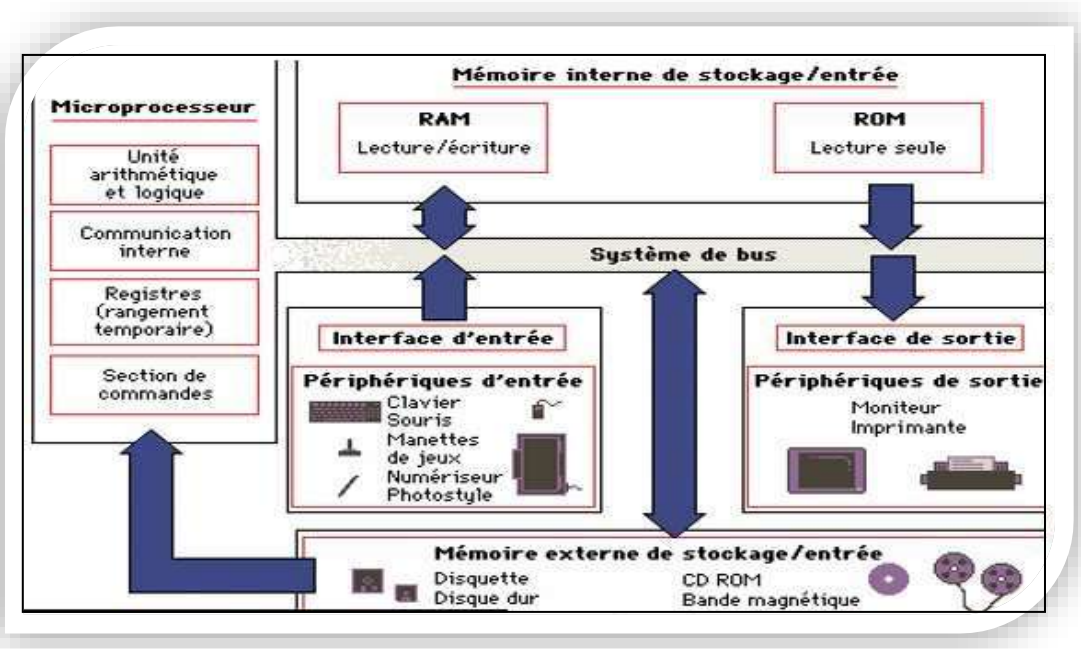
## 1.3. Les composants de l'ordinateur

Le traitement de l'information par l'ordinateur nécessite la coopération de plusieurs éléments. Au niveau de l'ordinateur, on distingue deux parties :

- *Une partie matérielle (Hardware) :* elle renvoie à la construction physique de la machine.
- *Une partie logicielle (Software) :* elle est constituée de l'ensemble des programmes pouvant être un programme d'application ou un programme de pilotage ou de base.

Les composants physique (ou matérielle) de l'ordinateur est composée en général : d'une unité centrale et de différents périphériques.

Dans le boîtier (unité centrale) est monté une *carte-mère* où sont implantés les lignes du *bus* et les principaux circuits électroniques : processeur, chipset, RAM *(Mémoire vive ; en anglais Random Acces Memory)*, ROM *(Mémoire Morte ; en anglais Read Only Memory)*, connecteurs et câble de liaisons, les cartes contrôleurs et sorties des différents périphériques.




**Structure d'un ordinateur**

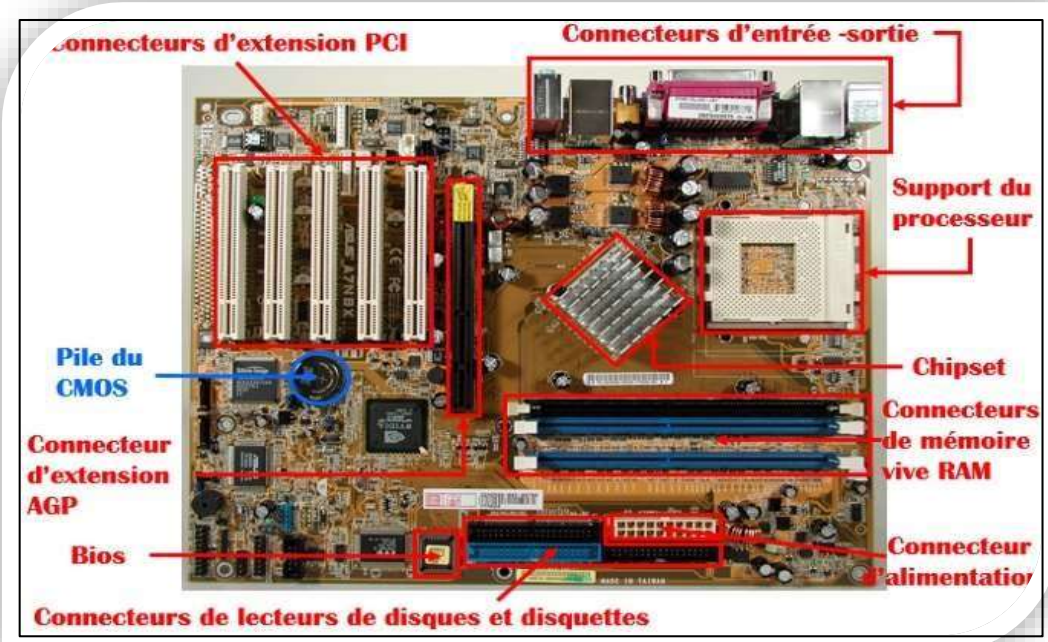
## 1.4. La carte mère

La carte mère *(en anglais « **Mainboard** » ou « **Motherboard** »)* est le socle permettant la connexion de l'ensemble des éléments essentiels de l'ordinateur. La carte mère est une carte maîtresse, prenant la forme d'un grand circuit imprimé possédant notamment des connecteurs pour les cartes d'extension, les barrettes de mémoires, le processeur, etc.

### a. Les caractéristiques d'une carte mère

Il existe plusieurs façons de caractériser une carte mère, notamment selon les caractéristiques suivantes :

- ***Le facteur d'encombrement ou facteur de forme :*** définit la géométrie, les dimensions, l'agencement et les caractéristiques électriques de la carte mère.
- ***Le chipset :*** circuit électronique chargé de coordonner les échanges de données entre les divers composants de l'ordinateur (processeur, mémoire...).
- ***Le type de support de processeur :*** On distingue deux catégories de supports :
  - **Slot** (en français fente) : il s'agit d'un connecteur rectangulaire dans lequel on enfiche le processeur verticalement.
  - **Socket** (en français embase) : il s'agit d'un connecteur carré possédant un grand nombre de petits connecteurs sur lequel le processeur vient directement s'enficher.
- ***Les connecteurs de mémoire vive* (RAM) :** Les connecteurs d'extension sont des réceptacles dans lesquels il est possible d'insérer des cartes d'extension, c'est-à-dire des cartes offrant de nouvelles fonctionnalités ou de meilleures performances à l'ordinateur.

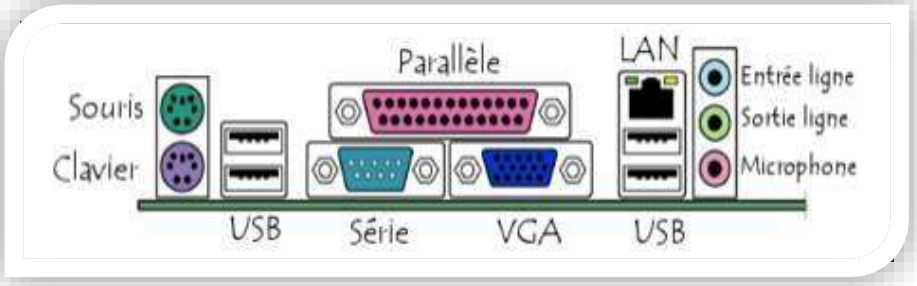



**Modèle de carte mère**

## b. Les types de connecteurs

Il existe plusieurs sortes de connecteurs :

- ***Les connecteurs d'entrée-sortie*** : La carte mère possède un certain nombre de connecteurs d'entrées-sorties regroupés sur le « panneau arrière ».




**Vue arrière du PC - branchement des périphériques**

La plupart des cartes mères proposent les connecteurs suivants :

- **Port série**, permettant de connecter de vieux périphériques ;
- **Port parallèle**, permettant notamment de connecter de vieilles imprimantes ;
- **Ports USB** « Universal Serial Bus » (1.1, bas débit, ou 2.0, haut débit), permettant de connecter des périphériques plus récents ;
- **Connecteur RJ45** « Registered Jack » (appelés LAN ou port Ethernet) permettant de connecter l'ordinateur à un réseau. Il correspond à une carte réseau intégrée à la carte mère.

- **Connecteur VGA** « Video Graphics Array » (appelé SUB-D15), permettant de connecter un écran. Ce connecteur correspond à la carte graphique intégrée.
- **Prises audio** (entrée Line-In, sortie Line-Out et microphone), permettant de connecter des enceintes acoustiques ou une chaîne hi-fi, ainsi qu'un microphone. Ce connecteur correspond à la carte son intégrée

## 2. Le processeur

### 2.1. Définition

Le **processeur** (noté **CPU**, pour Central Processing Unit) est un circuit électronique cadencé au rythme d'une horloge interne, grâce à un cristal de quartz qui, soumis à un courant électrique, envoie des impulsions, appelées « **top** ».

### 2.2. Le rôle du processeur

Le **processeur** (**CPU**, pour Central Processing Unit, soit Unité Centrale de Traitement) est le cerveau de l'ordinateur. Il permet de manipuler des informations numériques, c'est-à-dire des informations codées sous forme binaire, et d'exécuter les instructions stockées en mémoire.

### 2.3. La structure du processeur

Le processeur est constitué d'un ensemble d'unités fonctionnelles reliées entre elles. Les rôles des principaux éléments d'un processeur sont les suivants :

**1)** Une **unité d'instruction** (ou unité de commande, en anglais control unit) qui lit les données arrivantes, les décode puis les envoie à l'unité d'exécution ; L'unité d'instruction est notamment constituée des éléments suivants :

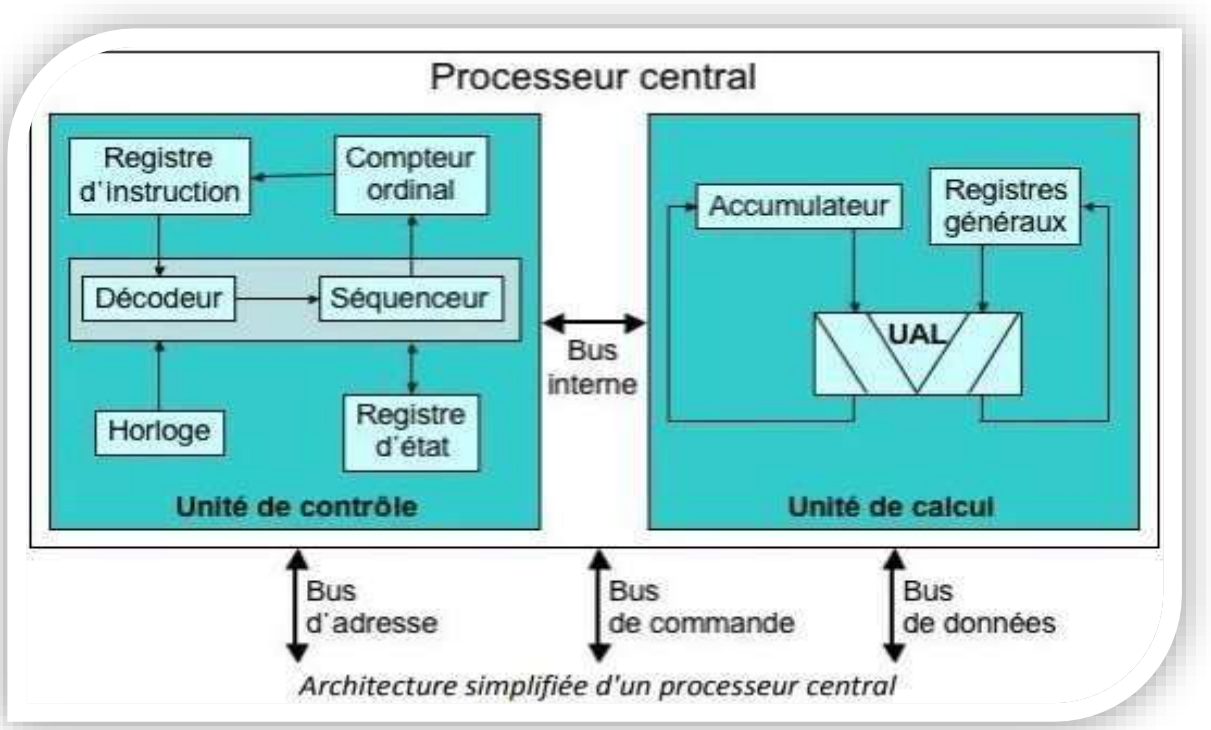
**a)** **Séquenceur** (ou bloc logique de commande) chargé de synchroniser l'exécution des instructions au rythme d'une horloge. Il est ainsi chargé de l'envoi des signaux de commande
**b)** **compteur ordinal** contenant l'adresse de l'instruction en cours
**c)** **registre d'instruction** contenant l'instruction à exécuter.
**d)** **décodeur** identifier l'instruction à exécuter qui se trouve dans le registre RI, puis d'indiquer au séquenceur la nature de cette instruction afin que ce dernier puisse déterminer la séquence des actions à réaliser.

**2)** Une **unité d'exécution** (ou unité de traitement), qui accomplit les tâches que lui a données l'unité d'instruction. L'unité d'exécution est notamment composée des éléments suivants :

**3)** L'**unité arithmétique et logique** (notée **UAL** ou en anglais ALU pour Arithmetical and Logical Unit) pour le traitement des données.

**a)** L'**unité de virgule flottante** (notée **FPU**, pour Floating Point Unit), qui accomplit les calculs complexes non entiers que ne peut réaliser l'unité arithmétique et logique.
**b)** Le **registre d'état**.
**c)** Le **registre accumulateur**.

**4)** Une **unité de gestion des bus** (ou unité d'entrées-sorties), qui gère les flux d'informations entrant et sortant, en interface avec la mémoire vive du système ;




*Architecture simplifiée d'un processeur central*

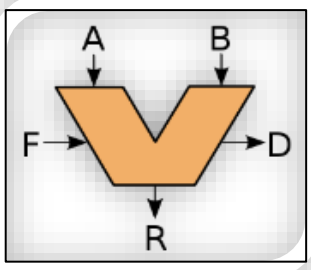
## 3. L'Unité Arithmétique et Logique (UAL)

### 3.1 Définition

L'unité arithmétique et logique, abrégée UAL, est l'organe de l'ordinateur chargé d'effectuer les calculs. Elle est incluse dans le microprocesseur.

Elle effectue les opérations spécifiées par les instructions et exécute les calculs arithmétiques (ex : addition) et logique (ex : comparaison).

C'est un circuit combinatoire qui produit un résultat (R) sur n bits fonction des données présentes sur ses entrées (A et B) et de la fonction à réaliser (F) et met à jour des drapeaux.




**Une unité arithmétique et logique à deux entrées**

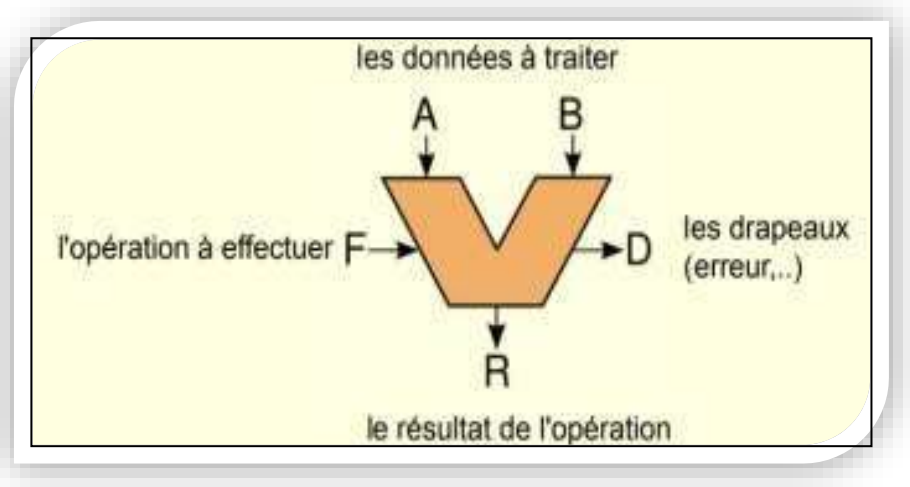
### 3.2 Fonctionnement de l'UAL

L'**UAL** permet de réaliser différents types d'opérations sur des données de la forme R=F(A,B) :

- *Des opérations arithmétiques* : additions, soustractions, ...
- *Des opérations logiques* : ou, et, ou exclusif, ...

Elle met par ailleurs à jour **des indicateurs** (ou drapeaux ou flag) en fonction du résultat de l'opération effectuée :

- **Z** (Zéro) : indicateur mis à 1 si le résultat de l'opération est 0.
- **N** (Négatif) : indicateur mis à 1 pour un résultat négatif (bit le plus à gauche égal à 1). o **C** (Carry-out) : mis à 1 en cas de retenue ou débordement en contexte non signé. o **V** (Overflow): mis à 1 en cas de débordement en contexte signé.




**Fonctionnement de l'UAL**

### 4. Les bus

En informatique, le mot bus désigne l'ensemble des liaisons électrique (nappes, pistes de circuits imprimés, etc.) utilisés par plusieurs éléments matériels afin de communiquer entre eux. Si cette liaison relie deux éléments seulement, elle est appelée *port matériel* (port série, port parallèle, etc.).

Un bus est un ensemble de fils permettant de lier et faire communiquer les composants d'un ordinateur afin d'assurer la transmission du même type d'information (données, adresses ou commandes).

**4.1. Caractéristiques d'un bus** Un bus est caractérisé par :

**a) Sa largeur :** un bus est caractérisé par le volume d'informations transmises simultanément (exprimé en bits). La **largeur** désigne le nombre de bits qu'un bus peut transmettre simultanément.

**1 fil** transmet **un bit, 1bus à n fils = 1 bus n bits.**

**Exemple :** une nappe de 32 fils permet ainsi de transmettre 32 bits en parallèle.

**b) Sa vitesse :** est le nombre de paquets de données envoyés ou reçus par seconde. Elle est également définie par sa **fréquence** (exprimée en Hertz).

On parle de **cycle** pour désigner chaque envoi ou réception de données. Un cycle mémoire assure le transfert d'un mot mémoire :

**Cycle mémoire (s)=1/Fréquence**

**c) Son débit :** Le **débit** maximal du bus (ou le taux de transfert maximal) est la quantité de donnée qu'il peut transfert par unité de temps ; en multipliant sa largeur par sa fréquence.

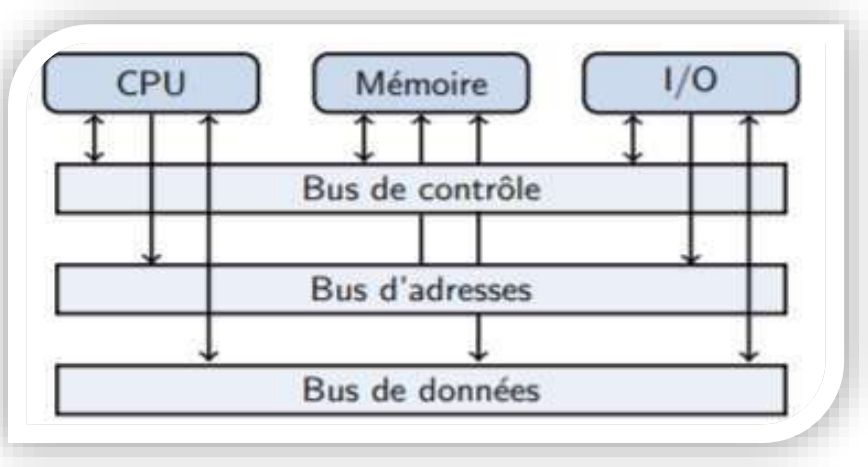
**Débit (octets/s) = (Nombre de transferts par seconde x Largeur) / 8**

**Bande passante (en Mo/s) = Largeur bus (en octets) x Fréquence (en MHz)**

## 4.2. Différents types de bus

On peut distinguer trois types de bus véhiculant des informations en parallèle dans un système de traitement programmé de l'information :

- **Un bus de données** : bidirectionnel qui assure le transfert des informations entre le microprocesseur et son environnement, et inversement. Son nombre de lignes est égal à la capacité de traitement du microprocesseur.
- **Un bus d'adresses :** unidirectionnel qui permet la sélection des informations à traiter dans un espace mémoire (ou espace adressable) qui peut avoir $2^n$ emplacements, avec **n** = nombre de conducteurs du bus d'adresses. L'espace mémoire adressable dépend de la largeur du bus d'adresses.
- **Un bus de commande** : constitué par quelques conducteurs qui assurent la synchronisation des flux d'informations sur les bus des données et des adresses.

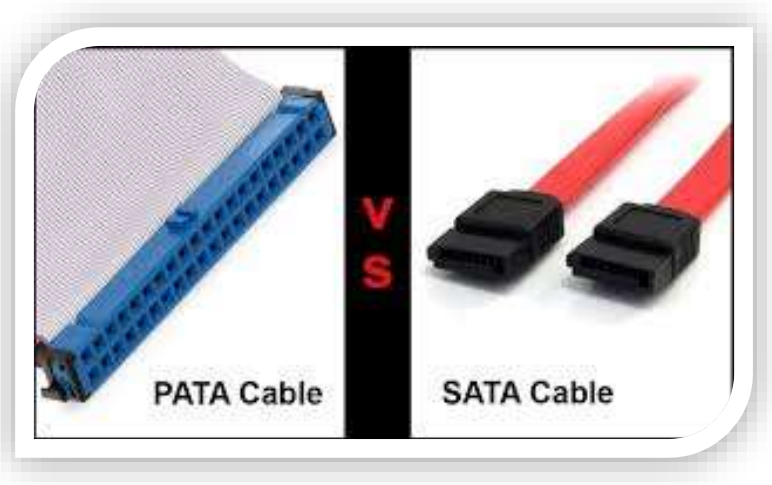



**Différents types de bus selon la nature de l'information à transporter**

## 4.3. Types de bus de données

Il existe deux grands types de bus de données selon le type de transmission :

- **Les bus séries :** ils permettent des transmissions sur de grandes distances. Ils utilisent une seule voie de communication sur laquelle les bits sont envoyés les uns à la suite des autres.
  **Exemples** : USB, SATA.
- **Les bus parallèles :** sur un bus parallèle plusieurs bits sont transmis simultanément. Ils sont utilisés sur des distances courtes ; par exemple ; pour relier le processeur ; la mémoire.
  **Exemple :** Bus PATA.




**Câble PATA et câble SATA**

### 4.4. Les principaux bus

On distingue généralement sur un ordinateur deux principaux bus :

- **Le bus système** (*bus interne, en anglais internal bus ou front-side bus, noté FSB*). Le bus système permet au processeur de communiquer avec la mémoire centrale du système (mémoire vive ou RAM).
- **Le bus d'extension** (*bus d'entrée/sortie*) permet aux divers composants liés à la carte-mère (USB, série, parallèle, cartes branchées sur les connecteurs PCI, disques durs, lecteurs et graveurs de CD-ROM, etc.) de communiquer entre eux. Il permet aussi l'ajout de nouveaux périphériques grâce aux connecteurs d'extension (appelés slots) qui lui y sont raccordées.

## 5. Les registres

### 5.1. Définition

Un registre est un emplacement de mémoire interne à un processeur. Les registres se situent au sommet de la hiérarchie mémoire : il s'agit de la mémoire la plus rapide d'un ordinateur, mais dont le coût de fabrication est le plus élevé, car la place dans un microprocesseur est limitée.

Chaque registre peut stocker une valeur entière distincte, bornée par la taille des registres (nombre de bits)

### 5.2. Les principaux registres

Certains registres sont spécialisés, comme :

- **Le compteur ordinal (« Program Counter » CO ou PC)** qui stocke l'adresse de la prochaine instruction à exécuter
- **Le registre d'instruction (« Instruction Register » RI ou IR),** qui stocke l'instruction en cours d'exécution
- **L'accumulateur (Acc)**, registre résultat de l'UAL, etc.

### 5.3. Les registres de l'UAL

Il y a aussi **les registres de l'UAL** ; qui sont accessibles au programmeur, contrairement aux registres de l'UCC. On dénombre :

- **Registres arithmétiques** : destinés pour les opérations arithmétiques (+, -, *, /, complément à1, ...) ou logiques (NOT, AND, OR, XOR), l'accumulateur (ACC) pour stocker le résultat, ….
- **Registres d'index :** pour stocker l'index d'un tableau de données et ainsi calculer des adresses dans ce tableau
- **Registre d'état** (PSW, Processor Status Word) **:** permettant de stocker des indicateurs sur l'état du système (retenue, dépassement, etc.) ;
- **Registre pointeur :** d'une pile ou de son sommet.
- **Registres généraux :** pour diverses opérations, ex., stocker des résultats intermédiaires
- **Registres spécialisés :** destinés pour certaines opérations comme les registres de décalages, registres des opérations arithmétiques à virgule flottante, …etc.

## 6. La mémoire interne : mémoire RAM (SRAM et DRAM), ROM, temps d'accès, latence,...

### 6.1. La mémoire

- Un ordinateur a deux caractéristiques essentielles qui sont **la vitesse** à laquelle il peut traiter un grand nombre d'informations et **la capacité de mémoriser ces informations**.
- On appelle mémoire tout dispositif capable de contenir, de conserver et de restituer sans les modifier de grandes quantités d'information (instructions + données).

### 6.2. Type de mémoire

Il existe deux types de mémoire dans un système informatique :

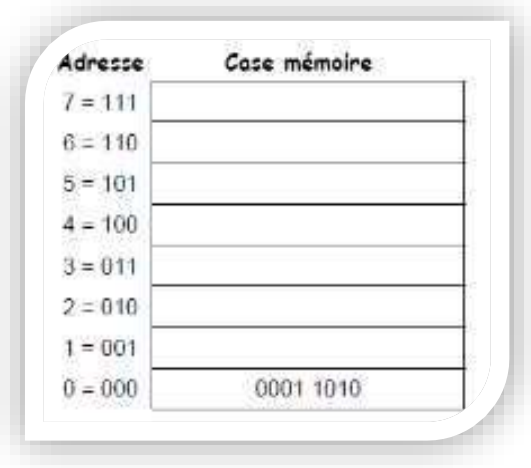
- **La mémoire centrale** qui est très rapide, physiquement peu encombrante mais coûteuse, c'est la mémoire de travail de l'ordinateur,
- **La mémoire de masse** ou mémoire auxiliaire, qui est plus lente, assez encombrante physiquement, mais meilleur marché, c'est la mémoire de « sauvegarde » des informations.

### 6.3. Caractéristiques d'une mémoire

- **La capacité :** c'est le nombre total de bits que contient la mémoire. Elle s'exprime aussi souvent en octet.
- **Le format des données :** c'est le nombre de bits que l'on peut mémoriser par case mémoire. On dit aussi que c'est la largeur du mot mémorisable.
- **Le temps d'accès :** c'est le temps qui s'écoule entre l'instant où a été lancée une opération de lecture/écriture en mémoire et l'instant où la première information est disponible sur le bus de données.
- **Le temps de cycle :** il représente l'intervalle minimum qui doit séparer deux demandes successives de lecture ou d'écriture.
- **Le débit :** c'est le nombre maximum d'information lues ou écrites par seconde.
- **Volatilité :** elle caractérise la permanence des informations dans la mémoire. L'information stockée est volatile si elle risque d'être altérée par un défaut d'alimentation électrique et non volatile dans le cas contraire.

## 6.4. Organisation d'une mémoire

Une mémoire peut être représentée comme une armoire de rangement constituée de différents tiroirs. Chaque tiroir représente alors une case mémoire qui peut contenir un seul élément : des **données**. Le nombre de cases mémoires pouvant être très élevé, il est alors nécessaire de pouvoir les identifier par un numéro. Ce numéro est appelé **adresse**. Chaque donnée devient alors accessible grâce à son adresse.




**Organisation de la mémoire**

- Avec une adresse de ***n* bits** il est possible de référencer au plus $\mathbf{2^n}$ cases mémoire. Chaque case est remplie par un mot de données (sa longueur ***m*** est toujours une puissance de 2). Le nombre de fils d'adresses d'un boîtier mémoire définit donc le nombre de cases mémoire que comprend le boîtier. Le nombre de fils de données définit la taille des données que l'on peut sauvegarder dans chaque case mémoire.

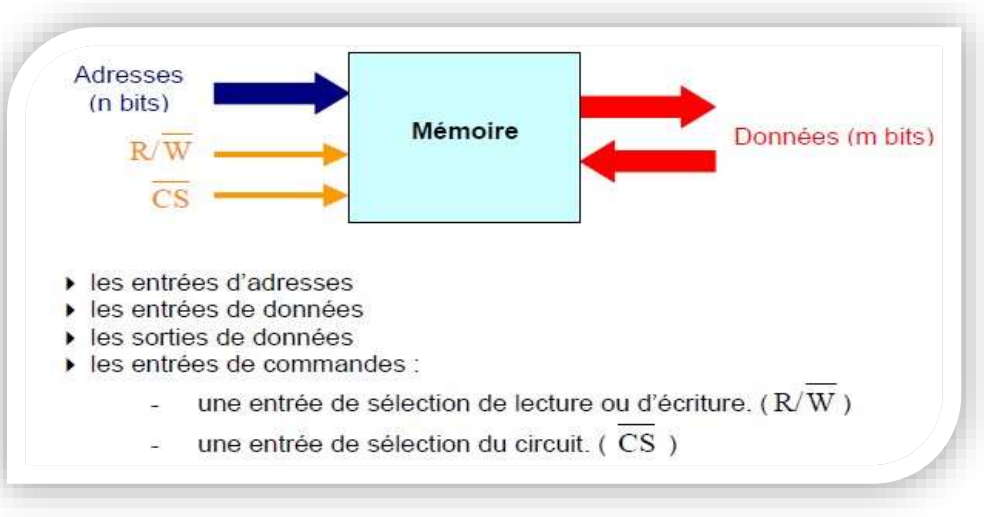
**Capacité (en bits) = 2nombre de lignes d'adresse x nombre de lignes de données**

**Nombre de Mots = Capacité / taille du mot**

**Nombre de mots = $2^{\text{nombre de lignes d'adresse}}$**

**et Taille du mot (en bits) = nombre lignes de données**

- En plus du bus d'adresses et du bus de données, un boîtier mémoire comprend une entrée de commande qui permet de définir le type d'action que l'on effectue avec la mémoire (lecture/écriture) et une entrée de sélection qui permet de mettre les entrées/sorties du boîtier en haute impédance.
- On peut donc schématiser un circuit mémoire par la figure suivante où l'on peut distinguer :




**Circuit mémoire**

- Une opération de lecture ou d'écriture de la mémoire suit toujours le même cycle :
  1) sélection de l'adresse
  2) choix de l'opération à effectuer (R/$\overline{W}$)
  3) sélection de la mémoire ($\overline{CS}$ = 0)
  4) lecture ou écriture la donnée

## 6.5. Classification des mémoires

Les mémoires peuvent être classés en trois catégories selon la technologie utilisée :

- **Mémoire à semi-conducteur** (mémoire centrale, ROM, PROM,…..) : très rapide mais de taille réduit.
- **Mémoire magnétique** (disque dur, disquette,…) : moins rapide mais stock un volume d'informations très grand.
- **Mémoire optique** (DVD, CDROM,..)

## 6.6. Types d'accès à la mémoire

Le mode d'accès à une mémoire dépend surtout de l'utilisation qu'on veut en faire et il existe trois types :

- **Par le contenu** : mémoire adressable par le contenu (ex. mémoire cache). La recherche s'effectue en parallèle sur toutes les cases mémoires via une clé et non via un index numérique. Le temps d'accès est constant.
    - Les opérations associées à ce mode d'accès : écriture (clé, donnée) ; lecture (clé) ; existe (clé) ; retirer (clé)
- **Aléatoire (ex., pour la mémoire vive)** : via une adresse Mémoire à accès aléatoire [Random Access Memory (RAM)] : le temps d'accès est identique car chaque mot mémoire est associé à une adresse unique.
    - Les opérations associées à ce mode d'accès : lecture (adr), écriture (adr, donnée)
- **Direct ou semi séquentiel (ex. les disques durs, CDs, ...)** : accès à un bloc de données ou cylindre (contenant la donnée recherchée) via son adresse puis déplacement séquentiel jusqu'à la donnée recherchée. Le temps d'accès est variable.
    - Les opérations associées à ce mode d'accès : lecture (bloc, déplacement) ; écriture (bloc, déplacement, donnée)

## 6.7. La mémoire interne (centrale ou principale)

La mémoire centrale (MC) représente **l'espace de travail** de l'ordinateur (calculateur).

- C'est l'organe principal de **rangement** des informations utilisées par le processeur.
- Dans une machine (ordinateur / calculateur) pour **exécuter** un programme il faut le charger (copier) dans la mémoire centrale.
- Le **temps d'accès** à la mémoire centrale et **sa capacité** sont deux éléments qui influent sur le **temps d'exécution** d'un programme (performance d'une machine).
- Les mémoires composant la mémoire principale sont des mémoires à base de semi-conducteurs, employant un mode d'accès aléatoire. Elles sont de deux types : volatiles ou non.

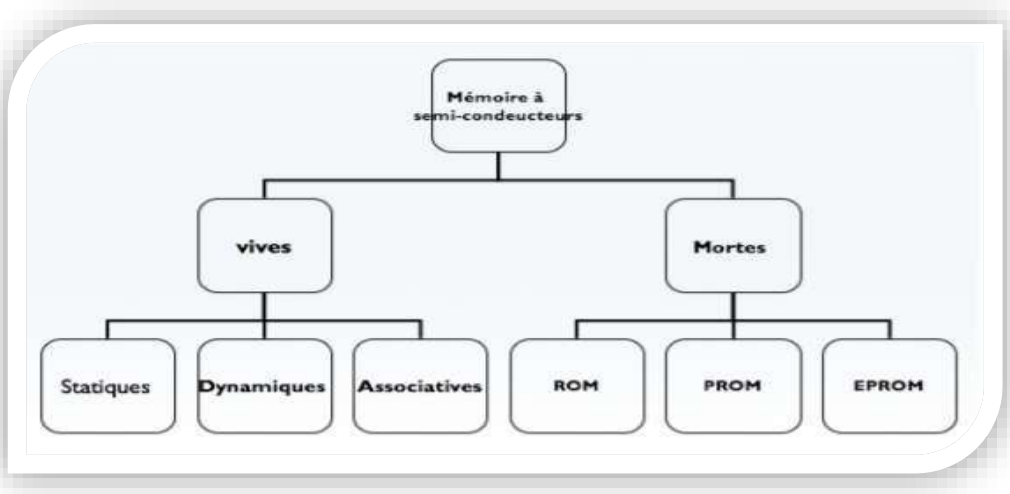



**Schéma des différents types de mémoire**

### Les mémoires vives (RAM)

Une mémoire vive sert au stockage temporaire de données. Elle doit avoir un temps de cycle très court pour ne pas ralentir le microprocesseur. Les mémoires vives sont en général volatiles : elles perdent leurs informations en cas de coupure d'alimentation.

Certaines d'entre elles, ayant une faible consommation, peuvent être rendues non volatiles par l'adjonction d'une batterie. Il existe deux grandes familles de mémoires RAM (Random Acces Memory : mémoire à accès aléatoire) :

a) **Les RAM statiques (SRAM) :** Le bit mémoire d'une RAM statique (SRAM) est composé d'une bascule. Chaque bascule contient entre 4 et 6 transistors.

b) **Les RAM dynamiques (DRAM) :** Dans les RAM dynamiques (DRAM), l'information est mémorisée sous la forme d'une charge électrique stockée dans un condensateur (capacité grille substrat d'un transistor MOS).

a) **Les mémoires statiques**

Dans la mémoire vive statique ou SRAM (Static Random Access Memory), la cellule de base est constituée par une bascule de transistors.

Le terme de statique fait référence à leur fonctionnement interne. Elles ne nécessitent quasiment pas de rafraichissement.

Dans la mesure où ce rafraichissement à un coût en temps, cela explique pourquoi ce type de mémoire est très rapide, entre 6 et 15 ns, mais assez chère.

On utilisera donc essentiellement pour des mémoires de faible capacité comme dans la mémoire cache pour les microprocesseurs.

b) **Les mémoires dynamiques**

- Dans la mémoire vive dynamique ou DRAM (Dynamic Random Access Memory) ; la cellule de base est constituée par un condensateur et un transistor
- Son inconvénient réside dans les courants de fuite des pico-condensateurs : l'information disparaît à moins que la charge des condensateurs ne soit rafraîchie avec une période de quelques millisecondes d'où le terme de dynamique.

**!!!** Il ne faut pas confondre SRAM et SDRAM :

- Une **SRAM** est une mémoire statique (SRAM = **S**tatique RAM) construite avec des bascules.

- Une **SDRAM** est une mémoire dynamique DRAM qui fonctionne à la vitesse du bus mémoire, elle est donc synchrone avec le fonctionnement du processeur le "S" indique la synchronicité (SDRAM = Synchrone DRAM).

- Une **DDR SDRAM** est une SDRAM à double taux de transfert pouvant expédier et recevoir des données deux fois par cycle d'horloge au lieu d'une seule fois. Le sigle DDR signifie **D**ouble **D**ata **R**ate.

- **VRAM [Video RAM] :** si elle a 2 ports pour pouvoir être accédée simultanément en lecture et en écriture

- **Mémoire flash :** mémoire RAM basée sur une technologie EEPROM. Le temps d'écriture est similaire à celui d'un disque dur (ex. mémoire d'appareils photos, téléphone, USB (flash) disk, MemoryStick, ...).
- **Modules mémoire DIMM (RAM) [Dual In-line Memory Module] :** groupe de puces RAM fonctionnant en 64 bits et généralement montées sur un circuit imprimé de forme rectangulaire, appelé barrette, que l'on installe sur la carte mère d'un ordinateur.
- **Modules SIMM [Single In-line Memory Module]** : idem à DIMM mais en 32 bits

Les performances des mémoires s'améliorent régulièrement, le secteur d'activité est très innovant, le lecteur retiendra que les mémoires les plus rapides sont les plus chères et que pour les comparer en ce domaine, il faut utiliser un indicateur qui se nomme le cycle mémoire.

## Les mémoires mortes (ROM)

Les mémoires mortes ou mémoires à lecture seule (ROM : Read Only Memory) sont non volatiles. Ces mémoires, contrairement aux RAM, ne peuvent être que lue. L'inscription en mémoire des données reste possible mais est appelée programmation. Suivant le type de ROM, la méthode de programmation changera. Il existe donc plusieurs types de ROM:

a) **ROM :** Elle est programmée par le fabricant et son contenu ne peut plus être ni modifié, ni effacé par l'utilisateur.

b) **PROM :** C'est une ROM qui peut être programmée une seule fois par l'utilisateur (Programmable ROM). La programmation est réalisée à partir d'un programmateur spécifique.

c) **EPROM *ou UV-EPROM :*** Pour faciliter la mise au point d'un programme ou tout simplement permettre une erreur de programmation, il est intéressant de pouvoir reprogrammer une PROM. La technique de claquage utilisée dans celles-ci ne le permet évidemment pas. L'EPROM *(*Erasable Programmable ROM*)* est une PROM qui peut être effacée.

d) **EEPROM :** pour (Electically EPROM) est une mémoire programmable et effaçable électriquement. Elle répond ainsi à l'inconvénient principal de l'EPROM et peut être programmée in situ.

e) **FLASH EPROM :** La mémoire Flash s'apparente à la technologie de l'EEPROM. Elle est programmable et effaçable électriquement comme les EEPROM.

### Structure physique d'une mémoire centrale

- **RAM** (Registre d'adresse Mémoire) : ce registre stock l'adresse du mot à lire ou à écrire.
- **RIM** (Registre d'information mémoire) : stock l'information lu à partir de la mémoire ou l'information à écrire dans la mémoire.
- **Décodeur** : permet de sélectionner un mot mémoire.
- **R/W** : commande de lecture/écriture, cette commande permet de lire ou d'écrire dans la mémoire (si R/W=1 alors lecture sinon écriture)
- **Bus d'adresses** de taille **k** bits
- **Bus de données** de taille **n** bits

### Sélection d'un mot mémoire

Lorsqu'une adresse est chargée dans le registre RAM, le décodeur va recevoir la même information que celle du RAM.

A la sortie du décodeur nous allons avoir une seule sortie qui est active Cette sortie va nous permettre de sélectionner un seule mot mémoire.

### Lecture et écriture de l'information

- **Comment lire une information ?**

  Pour lire une information en mémoire centrale il faut effectuer les opérations suivantes :

  – Charger dans le registre RAM l'adresse du mot à lire.
  – Lancer la commande de lecture (R/W=1)
  – L'information est disponible dans le registre RIM au bout d'un certain temps (temps d'accès)

- **Comment écrire une information ?**

  Pour écrire une information en MC il faut effectuer les opérations suivantes :

  – Charger dans le RAM l'adresse du mot ou se fera l'écriture.
  – Placer dans le RIM l'information à écrire.
  – Lancer la commande d'écriture pour transférer le contenu du RIM dans la mémoire.

## 7. La mémoire cache : utilité et principe, algorithmes de gestion du cache (notions de base)

### 7.1. Utilité

La mémoire cache ou ***antémémoire*** est une mémoire très rapide d'accès pour le microprocesseur. Elle agit comme un tampon entre le processeur et la mémoire principale. Elle est utilisée pour maintenir les parties de données et programmes qui sont le plus fréquemment utilisé par les CPU. Les parties de données et les programmes sont transférés du disque vers la mémoire cache par le système d'exploitation. Les données stockées dans une mémoire cache pourraient être les résultats d'un calcul plus tôt, ou les doublons de données stockées ailleurs.

**Exemple de mémoire cache**

## 7.2. Organisation en niveau

 Les processeurs récents possèdent plusieurs niveaux de mémoire cache : Niveaux L1 et L2, voire L3 pour certains processeurs.

- La **mémoire cache de premier niveau** (appelée **L1 Cache**, pour Level **1 Cache**) est directement intégrée dans le processeur. Il est généralement scindé en 2 parties (Instructions / Données). Les caches du premier niveau sont très rapides d'accès. Leur délai d'accès tend à s'approcher de celui des registres internes aux processeurs.
- Le **cache L2** est situé entre le cache **L1** et la mémoire vive. Il est moins rapide que le cache **L1**.
- Le **cache L3,** autrefois situé au niveau de la carte mère, est aujourd'hui intégré dans le CPU.

 Le cache Li+1 joue le rôle de cache pour le niveau Li

 Le cache Li+1 plus grand que Li mais moins rapide en temps d'accès aux donnés

## 7.3.Relation entre les niveaux de cache

 **Cache inclusif**

- Le contenu du niveau L1 se trouve aussi dans L2
- Taille globale du cache : celle du cache L2

 **Cache exclusif**

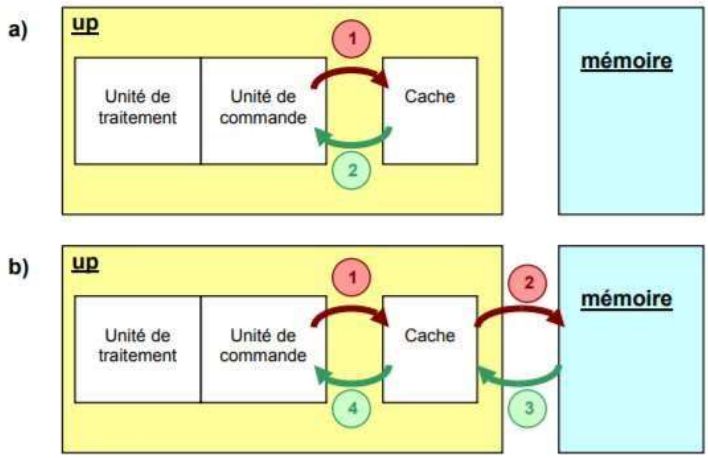
- Le contenu des niveaux L1 et L2 sont différents
- Taille globale du cache : taille L1 + taille L2

## 7.4.Principe

Le principe de cache est très simple : le microprocesseur n'a pas conscience de sa présence et lui envoie toutes ses requêtes comme s'il agissait de la mémoire principale :

 Soit la donnée ou l'instruction requise est présente dans le cache et elle est alors envoyée directement au microprocesseur. On parle de **succès** de cache. **(a)**

 Soit la donnée ou l'instruction n'est pas dans le cache, et le contrôleur de cache envoie alors une requête à la mémoire principale. Une fois l'information récupérée, il la renvoie au microprocesseur tout en la stockant dans le cache. On parle de **défaut** de cache. **(b)**

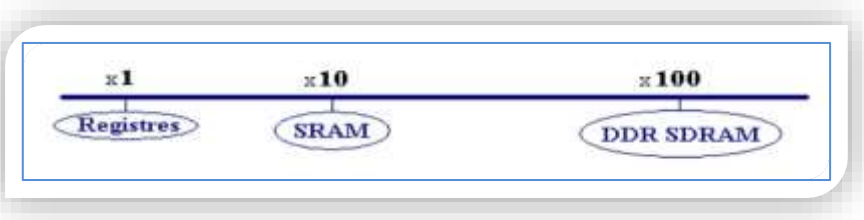



**Succès de cache / défaut de cache**

- Si un étage du processeur cherche une donnée, elle va être d'abord recherchée dans le cache de donnée L1 et rapatriée dans un registre adéquat, si la donnée n'est pas présente dans le cache L1, elle sera recherchée dans le cache L2.
- Si la donnée est présente dans L2, elle est alors **rapatriée** dans un registre adéquat et **recopiée** dans le bloc de donnée du cache L1. Il en va de même lorsque la donnée n'est pas présente dans le cache L2, elle est alors **rapatriée** depuis la mémoire centrale dans le registre adéquat et **recopiée** dans le cache L2.

**Remarque :**

- Le cache de niveau L1 et celui de niveau L2 peuvent être regroupés dans la même puce que le processeur (**cache interne/ on-chip**) ou n'être qu'accessible via un bus externe au processeur (**external cache**).
- Le facteur d'échelle (d'un coefficient de multiplication des temps d'accès à une information) relatif entre les différents composants mémoires du processeur et de la mémoire centrale.




- Les registres, mémoires les plus rapides se voient affecter la valeur de référence 1.
- L'accès par le processeur à une information située dans la DDR SDRAM de la mémoire centrale est 100 fois plus lent qu'un accès à une information contenue dans un registre.

## 7.5.Gestion de la mémoire cache

- **Définitions**
  - **Ligne :** est le plus petit élément de données qui peut être transféré entre la mémoire cache et la mémoire de niveau supérieur. (taille de la ligne = taille du bloc)
  - **Mot :** est le plus petit élément de données qui peut être transféré entre le processeur et la mémoire
- **Localité**

Le principe de localité affirme que les informations auxquelles va accéder le processeur ont une forte probabilité d'être localisées dans une fenêtre spatiale et une fenêtre temporelle.

1) **Localité spatiale** qui indique que l'accès une instruction située à une adresse X va probablement être suivi d'un accès à une zone tout proche de X

   **Exemple :** tableaux, structures.

   La localité spatiale suggère de copier des blocs de mots dans le cache plutôt que des mots isolés.
2) **Localité temporelle** : qui indique que l'accès à une zone mémoire à un instant donné a de fortes chances de se reproduire dans la suite du programme.

   **Exemple** : structures itératives.

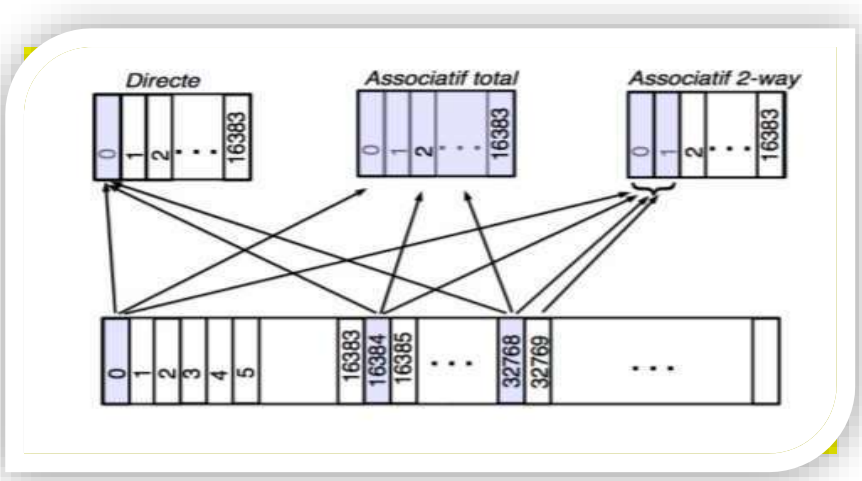
   La localité temporelle suggère de conserver pendant quelque temps dans le cache les informations auxquelles on vient d'accéder.

## 7.6.Correspondance cache et mémoire (le mapping)

La taille du cache est beaucoup plus petite que la taille de la mémoire. Il faut définir une stratégie de copie des blocs de données dans le cache ; Cette méthode s'appelle le « Mapping ». Trois stratégies sont possibles :

1) **Correspondance directe (direct mapped cache) :** le bloc **n** de la mémoire principale peut se retrouver seulement dans le bloc **m = (n modulo sb)** de la mémoire cache, **sb** étant la taille en nombre de blocs de la mémoire cache ;
2) **Correspondance totalement associative (fully associative cache)** : chaque bloc mémoire peut être placé dans n'importe quel bloc du cache
3) **Correspondance associative par ensemble (set associative cache)** : séparation de la mémoire cache en groupes de blocs et associativité complète dans un groupe,

   c.à.d. le bloc n de la mémoire principale peut se retrouver dans n'importe quel bloc du groupe **g = (n modulo sg**) de la mémoire cache, **sg** étant le nombre total de groupes de blocs dans la mémoire cache.




## 7.7.Correspondance cache et mémoire (le « Mapping »)

**Accès à un bloc du cache**

Les adresses mémoires peuvent être construites en fonction de la correspondance entre mémoire principale et cache. Dans ce cas, l'adresse mémoire d'un mot contient des informations sur sa présence dans un bloc et sa présence éventuelle dans le cache. Elle se décompose en deux parties :

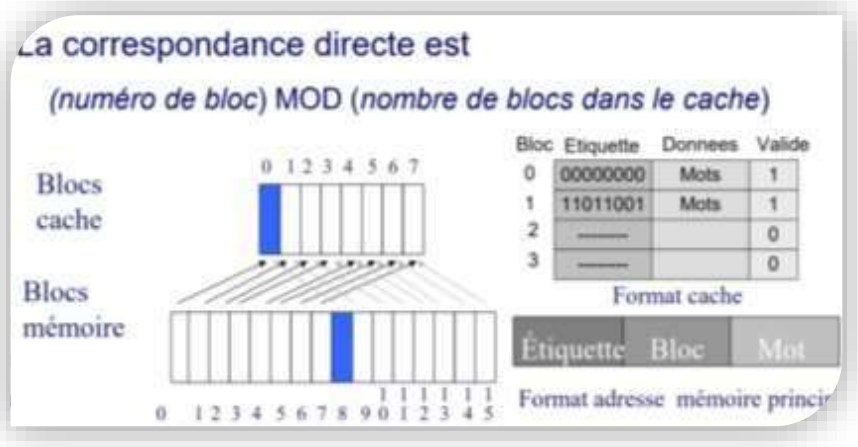
- Un numéro de bloc, qui se décompose en
  - un index, correspondant à l'emplacement de e bloc dans le cache
  - une étiquette permettant d'identifier le bloc mémoire correspondant au bloc placé dans le cache
- Un déplacement dans le bloc (le numéro du mot dans le bloc).

Ainsi, une table d'étiquette est maintenue, qui donne pour chaque bloc du cache l'étiquette du bloc mémoire placé dans ce bloc, ou le fait qu'aucun bloc mémoire n'a été copié dans ce bloc.

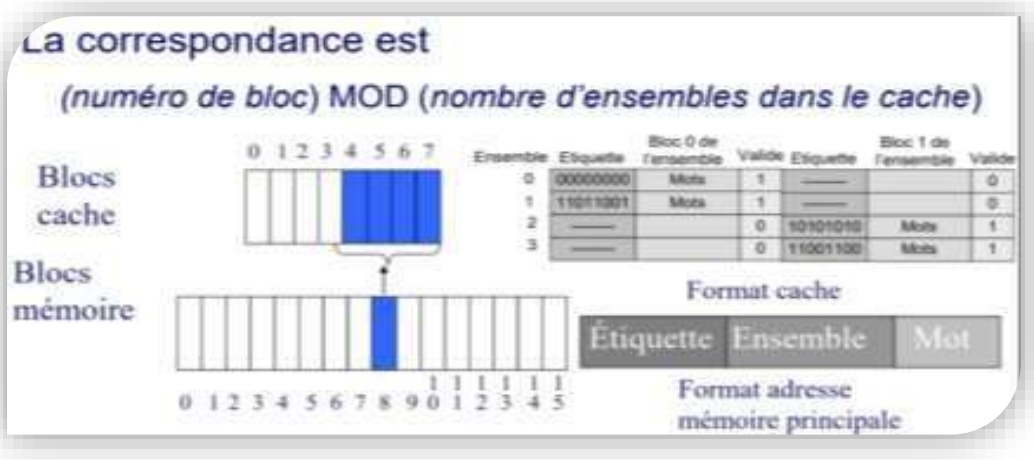
**Exemples :**

1. **Correspondance totalement associative**

## 2. Correspondance directe




## 3. Correspondance associative par ensemble




## 7.8.Algorithmes de remplacement

Si le cache est plein et que le processeur a besoin d'un bloc qui n'est pas dans le cache, il faut remplacer un des blocs du cache. Diverses stratégies sont employées, principalement :

- choisir un bloc candidat de manière aléatoire
- choisir le plus ancien bloc du cache (FIFO, First In First Out)
- choisir le bloc le moins récemment utilisé (LRU Least Recently Used)
- choisir le bloc le moins fréquemment utilisé (LFU Least Frequently Used)

Les stratégies concernant l'utilisation (LFU, LRU) sont les plus efficaces (vient ensuite la stratégie aléatoire). Les stratégies aléatoires et FIFO sont plus faciles à implanter.

## 7.9.Politique d'écriture :

Considérons le cas d'une opération d'écriture. Deux situations se présentent selon que le bloc dans lequel on souhaite écrire se trouve dans le cache ou non. Dans le premier cas, on peut choisir

- **Ecriture immédiate** : écrire à la fois dans le bloc du cache et dans le bloc de la mémoire (écriture simultanée, ou write through)
- **Ecriture remplacement** : écrire uniquement dans le bloc du cache, et différer l'écriture de e bloc en mémoire lorsque l'emplacement qu'il occupe sera désigné pour recevoir un nouveau bloc mémoire (réécriture ou write back).

Dans le deuxième cas, on peut choisir :

- de charger le bloc de la mémoire dans le cache puis effectuer l'opération d'écriture (écriture allouée)
- d'effectuer l'écriture directement dans la mémoire (écriture non allouée).

Une optimisation classique pour diminuer l'attente de la fin d'une écriture consiste à utiliser un tampon d'écriture, permettant au processeur de continuer à travailler dès que la donnée est écrite dans le tampon, sans attendre l'acquittement de la mémoire.


**Politique d'écriture**

## 7.10. Performance

On peut évaluer la performance d'une mémoire utilisant un cache par le calcul du temps d'accès mémoire moyen :

**Temps d'accès mémoire moyen = temps d'accès succès + taux d'échec x Pénalité d'échec**

**Temps d'accès succès = temps d'accès à une donnée résidant dans le cache**

**Taux d'échec = nombre de défaut de cache / nombre d'accès cache ou = 1 – taux de succès**

**Taux de succès = nombre de succès / nombre d'accès cache**

**Exemple :**

Lors de l'exécution d'une instruction, le processeur prend du temps pour la décoder, accéder aux données en mémoire nécessitées par cette instruction, et déclencher les opérations sur les données. Voici le cas suivant :

- durée d'un cycle horloge : T
- pénalité d'échec : 10 cycles
- durée d'une instruction (sans référence mémoire) : 2 cycles
- nombre de références mémoire par instruction : 1,33
- taux d'échec : 2%
- temps d'accès succès : négligeable
- temps d'exécution moyen d'une instruction = (2 + 1; 33 x 2% x10)T = 2; 27T
- et dans le cas où il n'y a pas de cache, e temps passe à : temps d'exécution moyen d'une instruction = (2 + 1; 33 x 10) = 15; 3T

**Remarque :**

- Cas de succès ⇒ **hit** ;
- Cas d'échec ⇒ **miss**

### 7.11. Avantages de la mémoire cache

- Elle est très rapide d'accès plus que la mémoire principale.
- Elle consomme moins de temps d'accès par rapport à .la mémoire
- Elle stocke du programme qui peut être exécuté dans un temps court:...
- – Elle stocke les données pour une utilisation temporaire

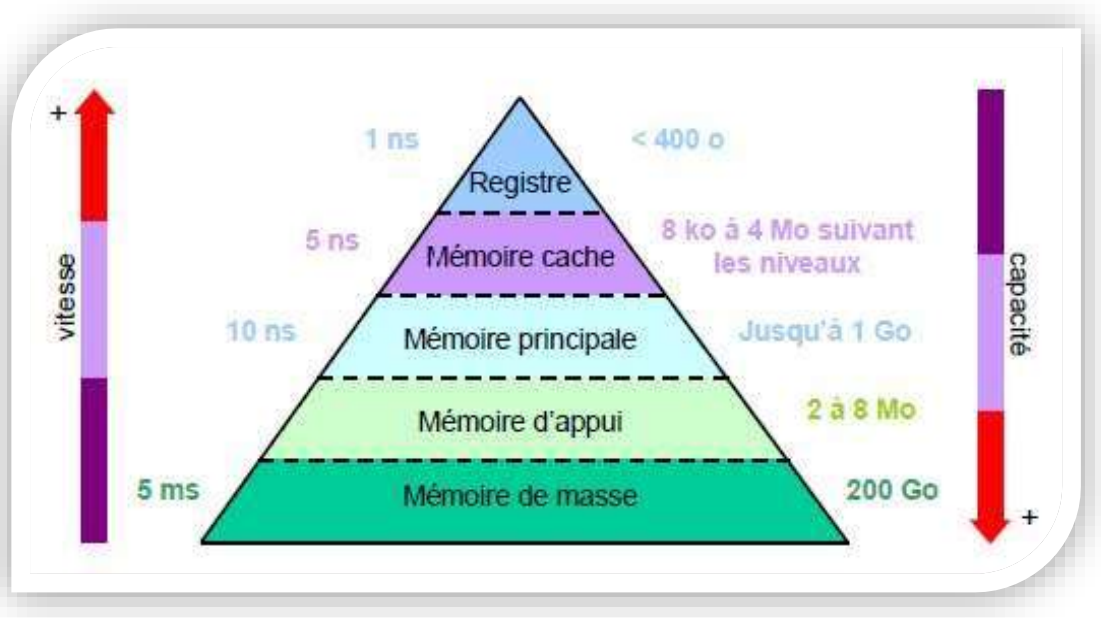
### 7.12. Inconvénients de la mémoire cache

- Elle a une capacité limitée
- Elle est très coûteuse

## 8. Hiérarchie des mémoires

Une mémoire idéale serait une mémoire de grande capacité, capable de stocker un maximum d'information et possédant un temps d'accès très faible afin de pouvoir travailler rapidement sur ces informations. Mais il se trouve que les mémoires de grande capacité sont souvent très lente et que les mémoires rapides sont très chères. Et pourtant, la vitesse d'accès à la mémoire conditionne dans une large mesure les performances d'un système.

En effet, c'est là que se trouve le goulot d'étranglement entre un microprocesseur capable de traiter des informations très rapidement et une mémoire beaucoup plus lente (ex : processeur actuel à 3Ghz et mémoire à 400MHz). Or, on n'a jamais besoin de toutes les informations au même moment. Afin d'obtenir le meilleur compromis coût-performance, on définit donc une hiérarchie mémoire. On utilise des mémoires de faible capacité mais très rapide pour stocker les informations dont le microprocesseur se sert le plus et on utilise des mémoires de capacité importante mais beaucoup plus lente pour stocker les informations dont le microprocesseur se sert le moins. Ainsi, plus on s'éloigne du microprocesseur et plus la capacité et le temps d'accès des mémoires vont augmenter.




**Hiérarchie des mémoires**

- **Les registres** sont les éléments de mémoire les plus rapides. Ils sont situés au niveau du processeur et servent au stockage des opérandes et des résultats intermédiaires.
- **La mémoire cache** est une mémoire rapide de faible capacité destinée à accélérer l'accès à la mémoire centrale en stockant les données les plus utilisées.
- **La mémoire principale** est l'organe principal de rangement des informations. Elle contient les programmes (instructions et données) et est plus lente que les deux mémoires précédentes.
- **La mémoire d'appui** sert de mémoire intermédiaire entre la mémoire centrale et les mémoires de masse. Elle joue le même rôle que la mémoire cache.
- **La mémoire de masse** est une mémoire périphérique de grande capacité utilisée pour le stockage permanent ou la sauvegarde des informations. Elle utilise pour cela des supports magnétiques (disque dur, ZIP) ou optiques (CDROM, DVDROM).

## 9. Conclusion

Nous avons présenté dans ce chapitre les principaux composants d'un ordinateur tels que : le processeur, l'UAL, les bus, les registres, la mémoire interne : mémoire RAM (SRAM et DRAM), ROM, temps d'accès, latence,..., la mémoire cache : utilité et principe, algorithmes de gestion du cache (notions de base) ainsi que la hiérarchie des mémoires.

Le chapitre suivant est consacré à la présentation de quelques notions de base sur les instructions d'un ordinateur telles que celle de langage de haut niveau, assembleur, langage machine, les instructions machines usuelles (arithmétiques, logiques, de comparaison, chargement, rangement, transfert, sauts,…).

Nous allons découvrir, aussi, le principe de compilation et d'assemblage ainsi que le rôle de l'unité de contrôle et de commande et les principales hases d'exécution d'une

instruction (recherche, décodage, exécution, rangement des résultats) avant de présenter le principe de l'UCC pipeline et ceux de l'horloge et du séquenceur.

# Chapitre 3 : Notions sur les instructions d'un ordinateur

**1-** Introduction aux instructions d'un ordinateur
**2-** Langage de haut niveau, assembleur, langage machine
**3-** Les instructions machines usuelles (arithmétiques, logiques, de comparaison, chargement, rangement, transfert, sauts,…)
**4-** Principe de compilation et d'assemblage (notions de base)
**5-** L'unité de contrôle et de commande
**6-** Phases d'exécution d'une instruction (recherche, décodage, exécution, rangement des résultats)
**7-** UCC pipeline
**8-** L'horloge et le séquenceur
**9-** Conclusion

## 1. Introduction aux instructions d'un ordinateur

Les instructions peuvent être classées en plusieurs catégories dont les principales sont :

- **Accès à la mémoire** : des accès à la mémoire ou transferts de données entre registres.
- **Opérations arithmétiques** : telles que l'addition, soustraction, division ou multiplication.
- **Opérations logiques** : opérations ET, OU, NON, NON exclusif, etc.
- **Contrôle** : contrôles de séquence, branchements conditionnels, etc.

Un ensemble d'instructions forme un programme dont l'exécution nécessite l'exécution de toutes les instructions qui le composent. Les différentes étapes suivies lors de l'exécution d'une instruction forment le cycle d'exécution de cette dernière.

Nous allons présenter dans les sections suivantes quelques détails sur les instructions et leurs phases d'exécution, le principe de compilation et d'assemblage, l'UCC et l'UCC pipeline ainsi que l'horloge et le séquenceur.

## 2. Langage de haut niveau, assembleur, langage machine

- **Le langage de haut niveau** tel que C, C++, Java, Python, ... apporte une plus grande facilité de programmation des machines en fournissant par exemple :
  - des éléments de syntaxe : tests conditionnels \if () else if...», boucles \for i in...», \x = 1 +2».
  - une allocation automatique des registres, et des emplacements mémoires \x = 3", \x + y"
  - une optimisation du code par exemple en disposant de manière optimale des portions du programme pour minimiser le nombre de branchement
  - des contraintes sur les opérations applicables sur les variables en les typant

- **Le langage assembleur** est le langage le plus « proche » du langage machine. Il est composé par des instructions en général assez rudimentaires que l'on appelle des mnémoniques. Ce sont essentiellement des opérations de transfert de données entre les registres et l'extérieur du microprocesseur (mémoire ou périphérique), ou des opérations arithmétiques ou logiques. Chaque instruction représente un code machine différent. Chaque microprocesseur peut posséder un assembleur différent.
- **Le langage machine** est le langage compris par le microprocesseur. Ce langage est difficile à maîtriser puisque chaque instruction est codée par une séquence propre de bits. Afin de faciliter la tâche du programmeur, on a créé différents langages plus ou moins évolués.

  Le langage machine est le langage directement interprétable par le processeur. Il est défini par un ensemble d'instructions que le processeur exécute directement.

  Chaque instruction correspond à un nombre (codé selon le cas sur un octet, un mot de 16 bits, ... : le format de l'instruction) et se décompose en :
  - une partie codant l'opération à exécuter appelée CodeOp ou code opération
  - une partie pour les opérandes

Un programme en langage machine est une suite de mots codant opérations et opérandes ; Chaque processeur possède son propre langage machine.

D'un point de vue de la programmation, le processeur offre :

- un certain jeu d'instructions qu'il sait exécuter.
- un certain nombre de registres :
  - utilisables/modifiables directement par le programme : registres de travail - pointeur de segment // il s'agit de registres vus par le jeu d'instructions
  - modifiables indirectement par le programme : compteur ordinal - pointeur de pile - registre d'instruction - registre d'états // ces registres sont manipulés implicitement par le jeu d'instructions
- un certain nombre de manière d'accéder à la mémoire : modes d'adressage

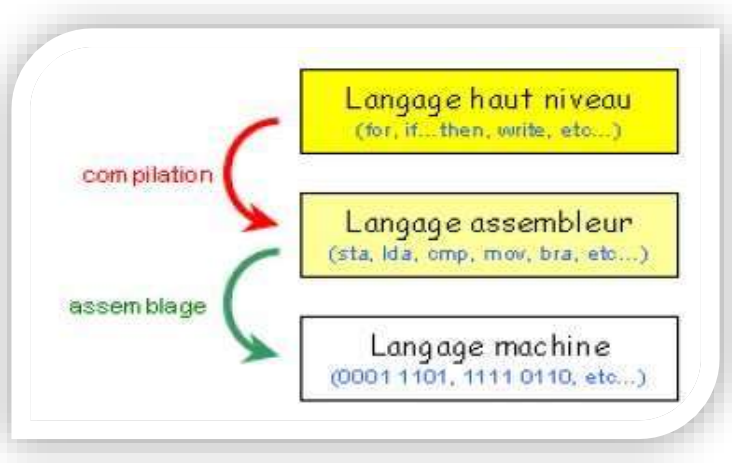
## 3- Les instructions machines usuelles (arithmétiques, logiques, de comparaison, chargement, rangement, transfert, sauts,…)

- Un code d'instruction est un groupe de bits (code binaire) qui ordonne l'ordinateur d'exécuter une s´séquence de micro-opérations.
- Les codes d'instruction et les données (opérandes) sont rangés dans la mémoire de l'ordinateur.
- L'unité de contrôle interprète alors le code binaire de l'instruction, et l'exécute en déployant une séquence de micro-opérations.
- Un code d'instruction est d'habitude divisé en deux parties : un code d'opération et un code d'adresse.
- Le code d'opération d´définit l'opération à être exécutée : addition, soustraction, ET logique, etc.
- Le code d'adresse spécifie d'habitude (mais pas toujours) l'adresse de l'opérande.

- L'unité de contrôle reçoit le code d'instruction de la mémoire, interprète le code d'opération, puis délivre une séquence de signaux de contrôle pour déclencher la séquence de micro opérations nécessaires qu'il faut effectuer sur les opérandes spécifiés.
- On dit que l'ensemble des instructions d'un ordinateur est complet s'il peut être utilisé pour évaluer n'importe quelle fonction qu'il est possible de calculer.
- Pour être complet, un ensemble d'instruction doit contenir assez d'instructions dans chacune des catégories suivantes :
  1) Instructions arithmétiques, logiques, et de décalage.
  2) Instructions pour déplacer de l'information de et vers la mémoire et les registres du processeur.
  3) Instructions de contrôle du programme, et instructions qui vérifient certaines conditions d'état.
  4) Instructions d'entrée et de sortie.
- La largeur en nombre de bits des différents champs constituant l'instruction est également importante, en particulier pour l'op-code. Ce nombre de bits est directement donné par le nombre d'opérations distinctes envisagées : **n** bits autorisent $\mathbf{2^n}$ instructions différentes. Cependant, toutes les instructions ne nécessitent pas forcément le même nombre d'opérandes, ou des champs de même longueur. Ainsi, sur une machine, pour une taille d'instructions donnée, le format des instructions (nombre d'opérande) peut ne pas être fixe, et dépendre du type d'opération, résolution de la mémoire et largeur d'adressage

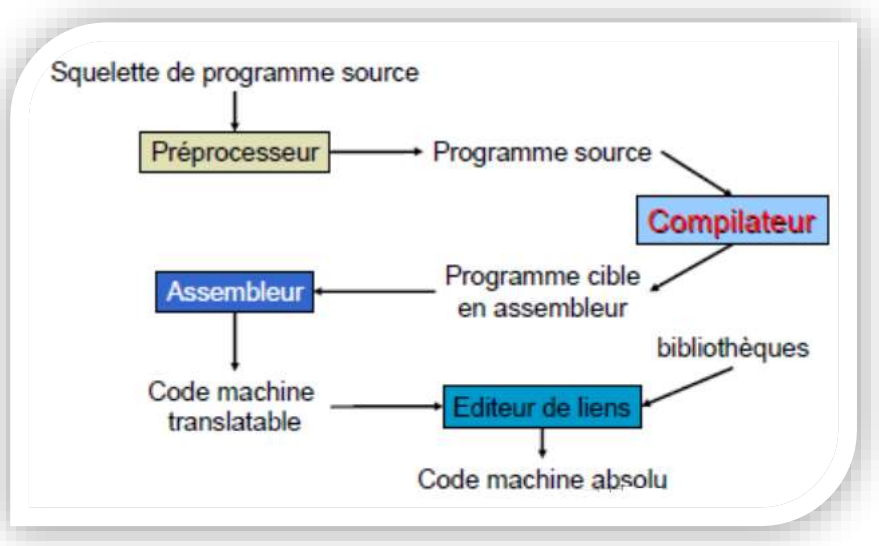
## 4- Principe de compilation et d'assemblage (notions de base)

- La difficulté de mise en œuvre de ce type de langage, et leur forte dépendance avec la machine a nécessité la conception de langages de haut niveau, plus adaptés à l'homme, et aux applications qu'il cherchait à développer.
- Faisant abstraction de toute architecture de machine, ces langages de haut niveau permettent l'expression d'algorithmes sous une forme plus facile à apprendre, et à dominer (C, Pascal, Java, etc…).
- Chaque instruction en langage de haut niveau correspondra à une succession d'instructions en langage assembleur.
- Une fois développé, le programme en langage de haut niveau n'est donc pas compréhensible par le microprocesseur. Il faut le compiler pour le traduire en assembleur puis l'assembler pour le convertir en code machine compréhensible par le microprocesseur. Ces opérations sont réalisées à partir de logiciels spécialisés appelés **compilateur** et **assembleur**.




**Compilation et assemblage**

- Les programmes, suites d'énoncés d'un langage de programmation de haut niveau, sont traduits (**compilés**), en langage de bas niveau (assembleur, code machine), directement interprétable par le matériel.




**Principe de compilation**

**Exemples :**

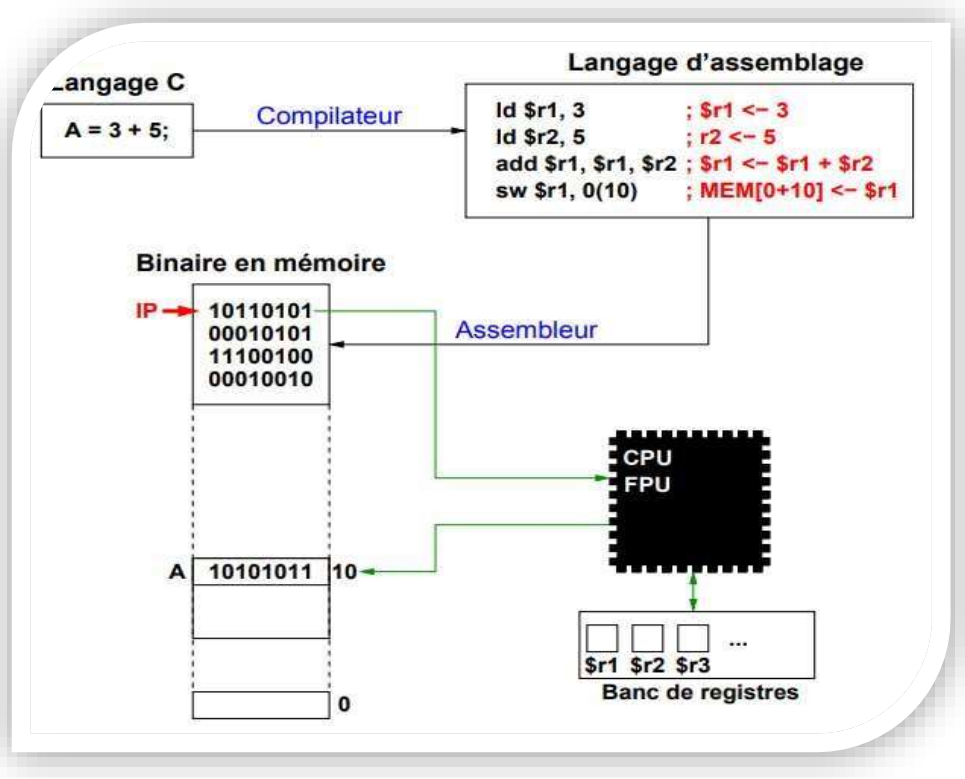
1. L'**expression** $y = x + 1$ est traduite en **assembleur** :

```
LDA X
INC
STA Y
HLT
```

qui est ensuite traduit en **code machine** :

91 00 08 10 39 00 09 64 10 99

## 2. Codage des instructions machine




**Exemple de compilation et d'assemblage**

## 5- L'unité de contrôle et de commande

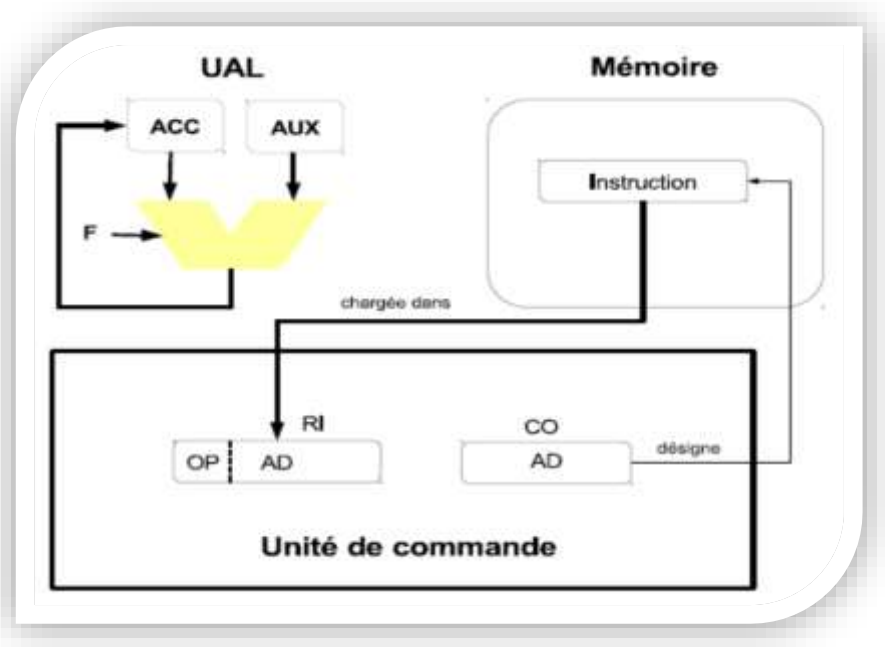
### Principe de fonctionnement de l'UCC

- Cette unité est chargée de **commander** et de **gérer** les différents **constituants de l'ordinateur** (contrôler les échanges, gérer l'enchaînement des différentes instructions, etc...).
- Elle **coordonne le fonctionnement** des autres éléments pour exécuter la séquence d'instructions constituant le programme.
- L'unité de **commande** orchestre l'exécution d'un programme en répétant indéfiniment les trois étapes ci-dessous :

  **Répéter**

  1) RI ← [CO] : Chargement dans le registre d'instruction RI du contenu de la cellule désignée par le compteur ordinal CO.
  2) CO ← CO + 1 : Incrémentation du compteur ordinal
  3) Exécution de l'instruction située dans le registre d'instruction

- Ces étapes représentent le **cycle d'instruction**.




**Fonctionnement de l'UCC**

## La structure de l'UCC :

L'UCC est composée des principaux éléments suivants :

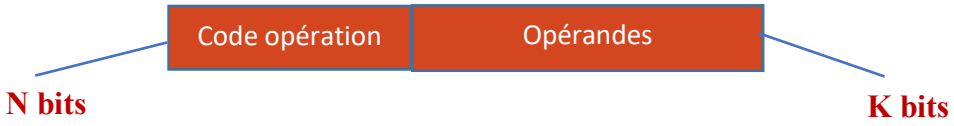
- **un registre instruction RI**, contenant l'instruction qui doit être exécutée
- **un compteur ordinal CO (PC)**, contenant l'adresse de la prochaine instruction à exécuter.
- **un registre adresse RA**, contenant l'adresse de la donnée à lire ou à écrire en mémoire.
- **un registre de données RD** : contenant temporairement la donnée lue ou à écrire en mémoire.
- **un registre d'état RE** : permettant de stocker des indicateurs sur l'état du système après l'exécution d'une instruction. Par exemple :
  - C (pour carry) : vaudra 1 si une retenue est présente.
  - Z (pour Zero) : vaudra 1 si le résultat de la dernière opération réalisée est nul.
  - V (pour Verow) : vaudra 1 en cas de dépassement de capacité N (pour Negative) : vaudra 1 si le résultat est négatif.
  - T (Trap ag) : mis à 1 le processeur fonctionne en mode pas à pas IE (Interrupt Enable) : mis à 1 les interruptions sont prises en compte
- **un registre d'index XR** (utilisé dans le mode d'adressage indexé) : l'adresse est obtenue en ajoutant son contenu à l'adresse contenue dans l'instruction ; peut-être incrémenter/décrémenter automatiquement après son utilisation.
- **un registre de base :** contient l'adresse (le numéro de segment) à ajouter aux adresses (relatives) contenues dans les instructions.
- **une horloge** : qui permet la synchronisation des éléments et des événements.

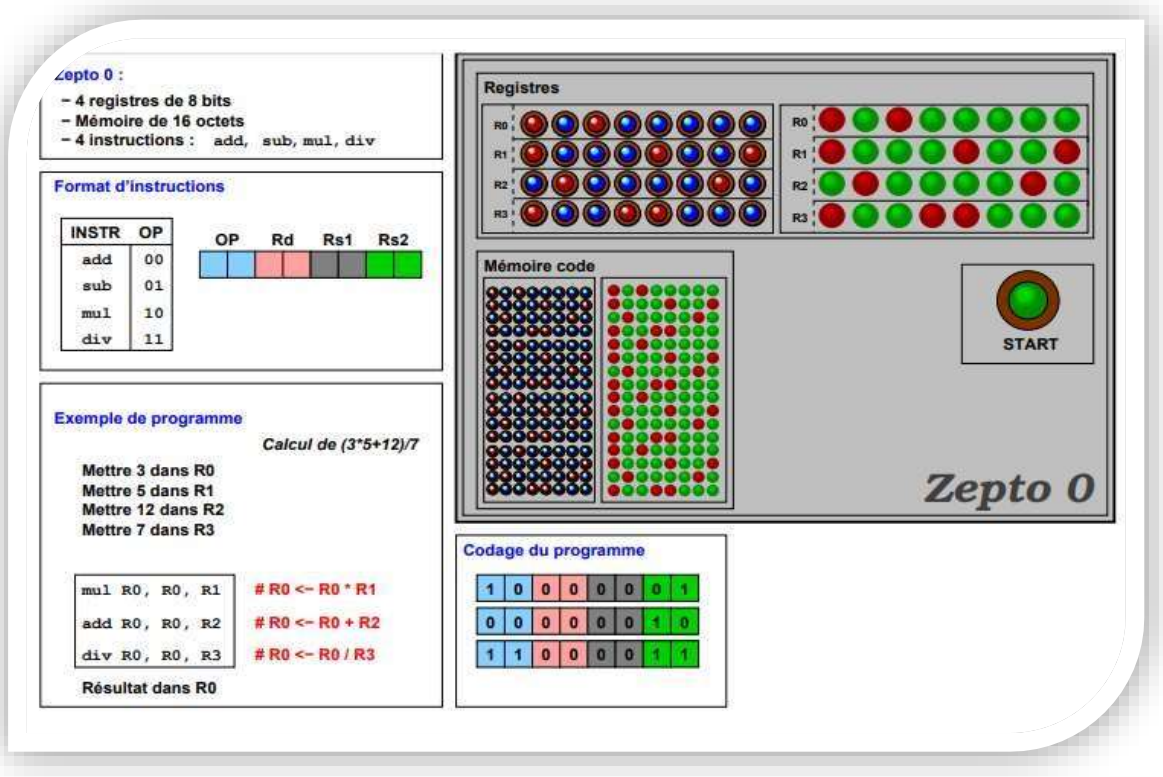
- **un décodeur de fonctions** : qui détermine les opérations à exécuter en fonction du code de l'instruction.
- **un séquenceur :** qui déclenche et coordonne les différentes opérations pour réaliser l'instruction

## Codage d'une instruction

Les instructions et leurs opérandes (données) sont stockés dans la mémoire.

- La taille d'une instruction (nombre de bits nécessaires pour la représenter en mémoire) dépend du type de l'instruction et du type de l'opérande.
- L'instruction machine est une chaîne binaire de **P bits** composée principalement de deux parties :




**Exemple de codage des instructions**

## Classification des machines par le nombre d'opérandes

- **Machine à 3 adresses**

Dans ce type de machine pour chaque instruction il faut préciser :

- l'adresse du premier opérande
- l'adresse du deuxième opérande
- et l'emplacement du résultat


**Exemple :** SUB X, Y, Z (Z←X-Y)

**Remarque :** Dans ce type de machine la taille de l'instruction est grande.

- **Machine à 2 adresses**

Dans ce type de machine, pour chaque instruction, il faut préciser :

- l'adresse du premier opérande
- l'adresse du deuxième opérande,


**Exemple :** ADD X, Y (Y←X +Y)

**Remarque** : l'adresse de résultat est implicitement l'adresse du deuxième opérande.

- **Machine à 1 adresse**

Dans de type de machine pour chaque instruction il faut préciser uniquement l'adresse du deuxième opérande.

- Le premier opérande existe dans le registre accumulateur.
- Le résultat est mis dans le registre accumulateur.


**Exemple :** ADD X (ACC←(ACC) + X)

**Remarque** : Ce type de machine est le plus utilisé.

- **Machine à 0 adresse**

Dans cette architecture, les instructions vont directement agir sur la pile.
Les opérandes sont automatiquement chargés depuis le pointeur de pile (**SP**, Stack Pointer), et le résultat est à son tour empilé.
**Exemple :** L'opération Z = X -Y sera traduite par la séquence suivante :

```
PUSH X          # Empile X
PUSH Y          # Empile Y
SUB             # Soustraction de X et Y (X-Y)
POP Z           # Stocke le sommet de la pile à l'adresse Z et dépile
```

**Remarque** : Avec cette architecture, les instructions arithmétiques et logiques sont vraiment très courtes.

## Les modes d'adressage

Les modes d'adressage sont un aspect de l'architecture des processeurs et de leurs jeux d'instructions.
Les modes d'adressage définis dans une architecture régissent la façon dont les instructions en langage machine identifient leurs opérandes.
Un mode d'adressage spécifie la façon dont est calculée l'adresse mémoire effective d'un opérande à partir de valeurs contenues dans des registres et de constantes contenues dans l'instruction ou ailleurs dans la machine.
Un mode d'adressage définit la manière dont le microprocesseur va accéder à l'opérande.
• Le champ **opérande** contient la **donnée** ou la référence (adresse) à la donnée.
• Le **code opération** de l'instruction comporte un ensemble de **bits** pour indiquer le **mode d'adressage**.
• Les modes d'adressage les plus utilités sont :
**1) Immédiat – 2) Direct – 3) Indirect – 4) Indexé – 5) Relatif**

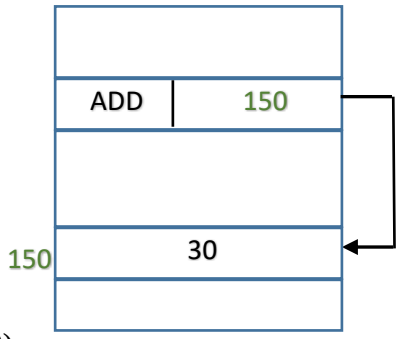
### 1) L'adressage immédiat

L'opérande existe dans le champ adresse de l'instruction.
**Exemple : SUB 20**


L'exécution de cette commande va avoir l'effet suivant : ACC←(ACC) - **20**
Si le registre accumulateur contient la valeur **40** alors après l'exécution son contenu sera égal à **20**.

### 2) L'adressage direct

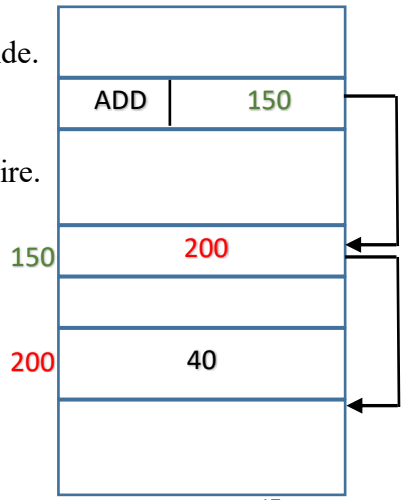
Le champ opérande de l'instruction contient l'adresse de l'opérande (emplacement en mémoire)
Pour réaliser l'opération, il faut le récupérer (lire) l'opérande à partir de la mémoire.
ACC ← (ACC) + (ADR)
**Exemple : ADD 150**
On suppose que l'accumulateur continent la valeur **20**.




A la fin de l'exécution nous allons avoir la valeur **50** (**20**+**30**).

### 3) L'adressage indirect

Le champ adresse contient l'adresse de l'adresse de l'opérande.
Pour réaliser l'opération il faut :
• Récupérer l'adresse de l'opérande à partir de la mémoire.
• Par la suite, il faut chercher l'opérande à partir de la mémoire.
ACC← (ACC) + ((ADR))
**Exemple : ADD 150**
Initialement l'accumulateur contient la valeur **20**
• Il faut récupérer l'adresse de l'adresse (150).
• Récupérer l'adresse de l'opérande à partir de l'adresse **150** (La valeur **200**)



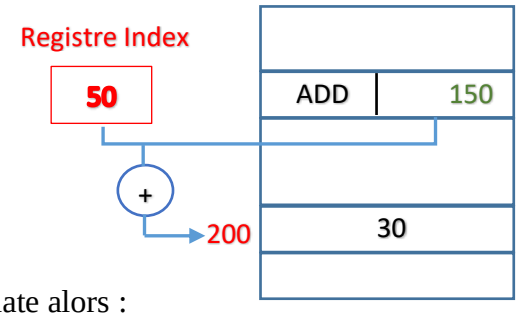
A la fin de l'exécution nous allons avoir la valeur **60** (20+40).

### 4) L'adressage indexé

L'adresse effective de l'opérande est relative à une zone mémoire.

• L'adresse de cette zone se trouve dans un registre spécial (registre index).

• Adresse opérande = ADR + (X)




**Remarque :** si ADR ne contient pas une valeur immédiate alors :
Adresse opérande = (ADR) + (X)

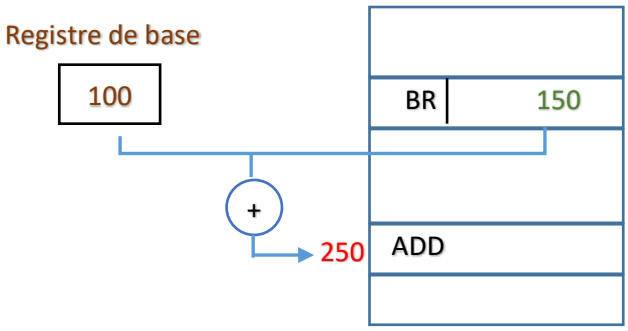
### 5) L'adressage relatif

L'adresse effective de l'opérande est relative à une zone mémoire.

• L'dresse de cette zone se trouve dans un registre spécial (**registre de base**).

• Ce mode d'adressage est utilisé pour les instructions de branchement.

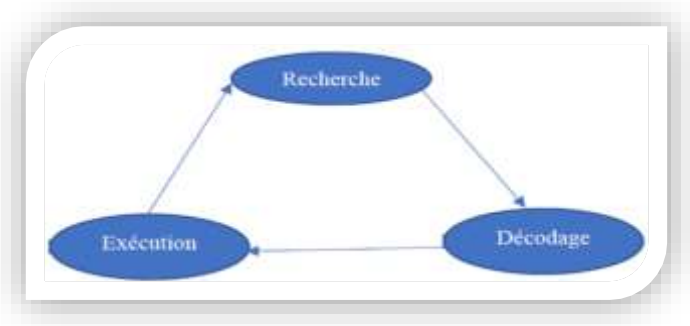
**Adresse = ADR + (base)**




**Remarque :** Relativement au mode d'adressage de la donnée, une instruction peut être codée sur 1 ou plusieurs octets.

### 6- Phases d'exécution d'une instruction (recherche, décodage, exécution, rangement des résultats) (Cycle d'exécution d'une instruction)

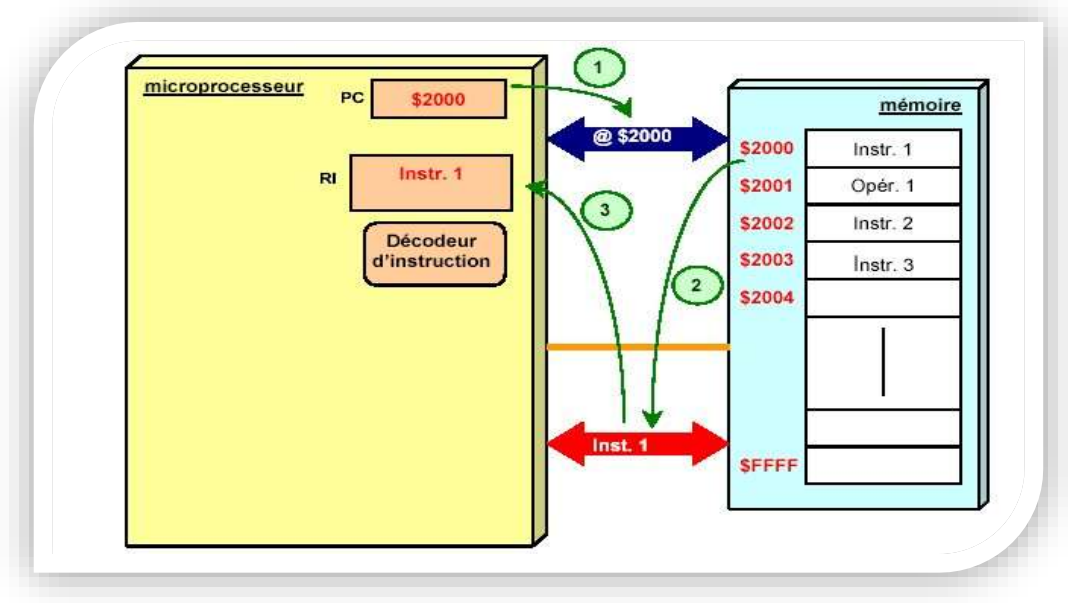
Le microprocesseur ne comprend qu'un certain nombre d'instructions qui sont codées en binaire. Le traitement d'une instruction peut être décomposé en trois phases.




**Cycle d'exécution d'une instruction**

## Phase 1 : Recherche de l'instruction à traiter

1) Le PC contient l'adresse de l'instruction suivante du programme. Cette valeur est placée sur le bus d'adresses par l'unité de commande qui émet un ordre de lecture.
2) Au bout d'un certain temps (temps d'accès à la mémoire), le contenu de la case mémoire sélectionnée est disponible sur le bus des données.
3) L'instruction est stockée dans le registre instruction du processeur.

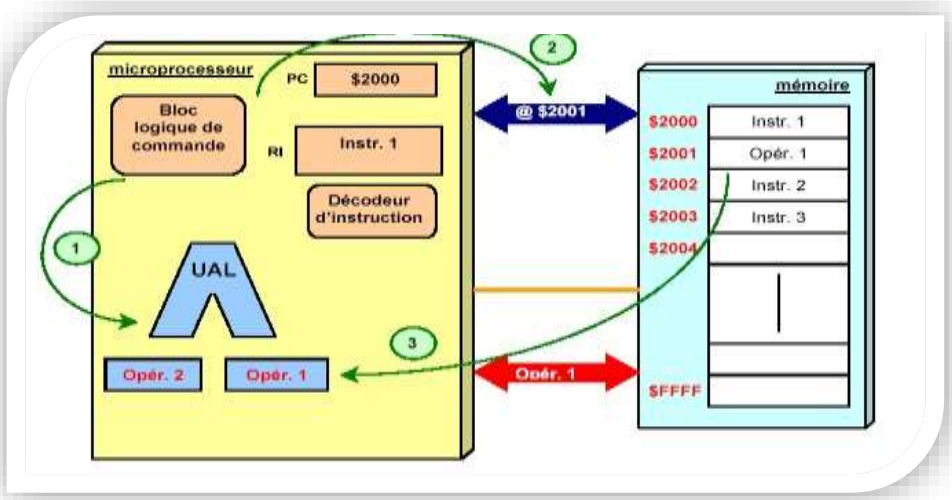



**Phase de recherche de l'instruction à traiter**

## Phase 2 : Décodage de l'instruction et recherche de l'opérande

- Les instructions exécutées par le processeur sont stockées en mémoire. Toutes les instructions possibles sont représentées par des codes.
- Le code d'une instruction est appelé op-code. Les op-code peuvent être de longueur fixe comme dans le LC-3 ou de longueur variable comme dans le Pentium.
- Le registre d'instruction contient maintenant le premier mot de l'instruction qui peut être codée sur plusieurs mots. Ce premier mot contient le code opératoire qui définit la nature de l'opération à effectuer (addition, rotation,...) et le nombre de mots de l'instruction.

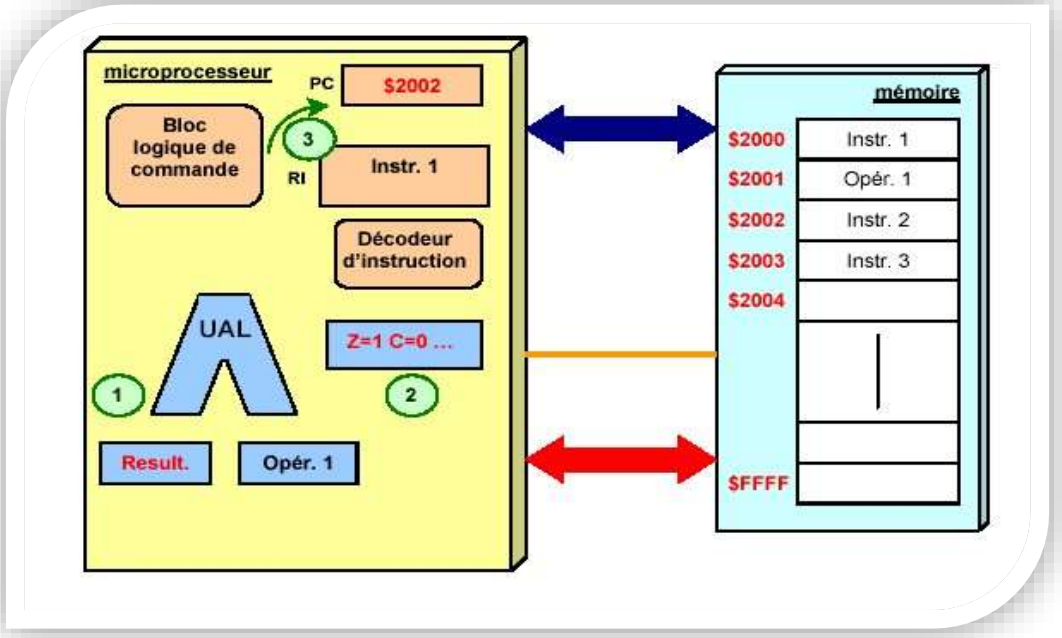
1) L'unité de commande transforme l'instruction en une suite de commandes élémentaires nécessaires au traitement de l'instruction.
2) Si l'instruction nécessite une donnée en provenance de la mémoire, l'unité de commande récupère sa valeur sur le bus de données.
3) L'opérande est stocké dans un registre.




**Phase de décodage de l'instruction et recherche de l'opérande**

## Phase 3 : Exécution de l'instruction

1) Le microprogramme réalisant l'instruction est exécuté.
2) Les drapeaux sont positionnés (*registre d'état*).
3) L'unité de commande positionne le PC pour l'instruction suivante.



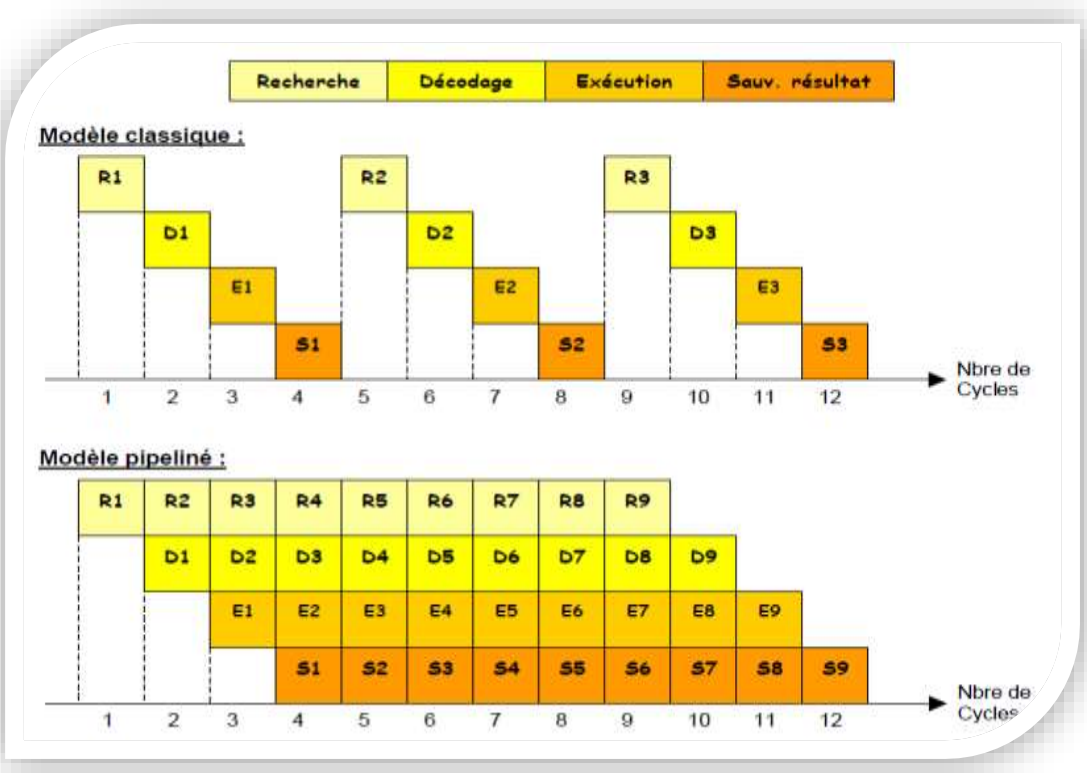

**Phase d'exécution de l'instruction**

## 7- UCC pipeline

### Principe

- L'exécution d'une instruction est décomposée en une succession d'étapes et chaque étape correspond à l'utilisation d'une des fonctions du microprocesseur.
- Lorsqu'une instruction se trouve dans l'une des étapes, les composants associés aux autres étapes ne sont pas utilisés. Le fonctionnement d'un microprocesseur simple n'est donc pas efficace.
- L'architecture pipeline permet d'améliorer l'efficacité du microprocesseur.
- En effet, lorsque la première étape de l'exécution d'une instruction est achevée, l'instruction entre dans la seconde étape de son exécution et la première phase de l'exécution de l'instruction suivante débute.
- Il peut donc y avoir une instruction en cours d'exécution dans chacune des étapes et chacun des composants du microprocesseur peut être utilisé à chaque cycle d'horloge.
- L'efficacité est maximale.
- Le temps d'exécution d'une instruction n'est pas réduit mais le débit d'exécution des instructions est considérablement augmenté.
- Une machine pipeline se caractérise par le nombre d'étapes utilisées pour l'exécution d'une instruction, on appelle aussi ce nombre d'étapes le **nombre d'étages** du pipeline.

**Exemple de l'exécution en 4 phases d'une instruction :**




**Le modèle classique versus le modèle pipeliné**

### Gain de performance

Dans cette structure, la machine débute l'exécution d'une instruction à chaque cycle et le pipeline est pleinement occupé à partir du quatrième cycle. Le gain obtenu dépend donc du nombre d'étages du pipeline :

- En effet, pour exécuter **n** instructions, en supposant que chaque instruction s'exécute en **k** cycles d'horloge, il faut :
  - **n*k cycles d'horloge** pour une **exécution séquentielle**.
  - **k cycles d'horloge** pour exécuter la **première instruction** puis **n-1 cycles** pour les **n-1 instructions** suivantes si on utilise un pipeline de **k étages**
- Le **gain** obtenu est donc de : **G= n*k/k+n-1**
- Donc lorsque le nombre **n** d'instructions à exécuter est grand par rapport à **k**, on peut admettre qu'on divise le temps d'exécution par **k**.

**Remarque :**

Le temps de traitement dans chaque unité doit être à peu près égal sinon les unités rapides doivent attendre les unités lentes.

**Exemples :**

- L'Athlon d'AMD comprend un pipeline de 11 étages.
- Les Pentium 2, 3 et 4 d'Intel comprennent respectivement un pipeline de 12, 10 et 20 étages.

## 8- L'horloge et le séquenceur

### L'horloge :

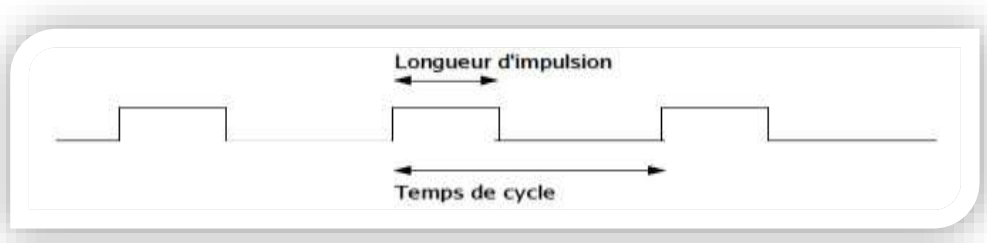
Dans de nombreux circuits numériques, il est essentiel de pouvoir garantir l'ordre dans lequel certains événements se produisent :

- Deux événements doivent absolument avoir lieu en même temps
- Deux événements doivent absolument se produire l'un après l'autre
- 98 % des circuits numériques sont synchrones

⇒ Nécessité de disposer d'une horloge pour synchroniser les événements entre eux.

Une horloge est un circuit qui émet de façon continue une série d'impulsions caractérisées par :

- La longueur de l'impulsion
- L'intervalle entre deux pulsations successives, appelé temps de cycle de l'horloge

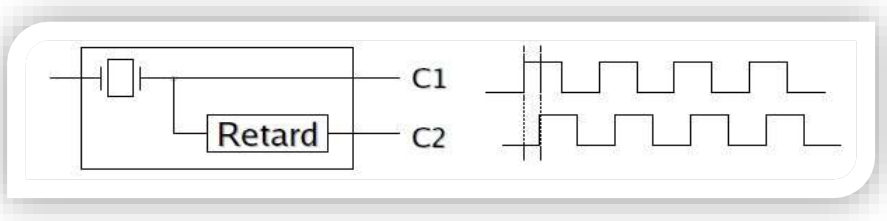



**Circuit d'une horloge**

## Cycles et sous cycles

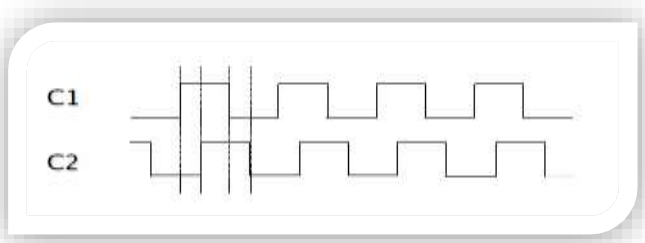
- Dans un ordinateur, de nombreux événements ont à se produire au cours d'un **cycle d'horloge**.
- Si ces événements doivent être séquencés dans un ordre précis, le **cycle d'horloge** doit être décomposé en **sous-cycles.**
- Un moyen classique pour cela consiste à **retarder** la copie d'un signal d'horloge primaire afin d'obtenir un signal secondaire décalé en phase.

**Exemple :**




On dispose alors de quatre bases de temps au lieu de deux :

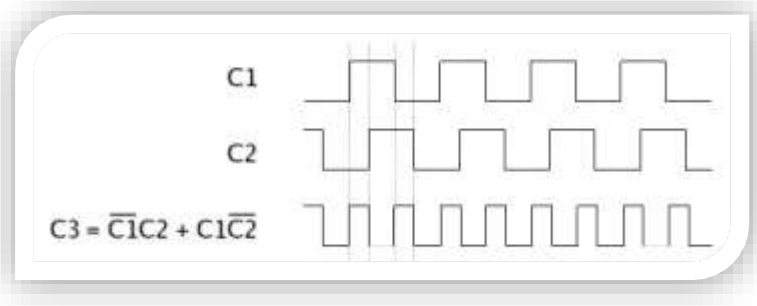
- Fronts montant et descendant de C1
- Fronts montant et descendant de C2




Pour certaines fonctions, on s'intéressera plutôt aux intervalles qu'à des instants précis

- Action possible seulement lorsque C1 est haut

On peut alors construire des sous-intervalles en s'appuyant sur les signaux original et retardé.




- L'horloge définit le cycle de base appelé **cycle machine** ou élémentaire égal à l'inverse de la fréquence. Il est utilisé pour synchroniser chaque étape des cycles de recherche et d'exécution.
- L'exécution du cycle de recherche ou d'exécution prend un certain nombre de cycles de base (dépendant de l'instruction).

- On définit le **cycle CPU** comme le temps d'exécution minimal (le plus court) d'une instruction (Recherche + Exécution)

**Remarque :** la vitesse de fonctionnement d'un ordinateur ne dépend pas seulement de sa fréquence d'horloge mais aussi du cycle mémoire et la vitesse du bus.

**Le séquenceur :**

Le séquenceur est un automate recevant des informations du décodeur et des signaux d'états (entrées) et produisant des signaux de commandes contrôlant les différentes unités. Plusieurs réalisations sont possibles :

1) **séquenceur chablé :**
   - circuit séquentiel (synchrone) réalisé avec des portes logiques
   - Un sous-circuit pour chaque instruction, sous-circuit active selon le code envoyé par le décodeur.
2) **Séquenceur micro-programmé :**
   - Une ROM contient des microprogrammes composés de micro-instructions.
   - Le séquenceur sait exécuter les séquences de micro-instructions
   - Le code de l'opération à exécuter dans l'instruction est utilisé comme étant l'adresse de la 1ère micro instruction du microprogramme.
   - Ce microprogramme est capable de générer une suite de signaux de commande équivalente à celle produite par un séquenceur câblé.

**Remarque :**

Un séquenceur micro-programmé est plus lent qu'un séquenceur câblé

## 9. Conclusion

Nous avons découvert via ce chapitre quelques détails sur les instructions et leurs phases d'exécution, le principe de compilation et d'assemblage, l'UCC et l'UCC pipeline ainsi que l'horloge et le séquenceur.

Le chapitre suivant est consacré au processeur, son rôle, le calcul de CPI (Cycle par Instruction), les processeurs CISC et RISC ainsi que quelques détails sur le microprocesseur MIPS R3000.

# Chapitre 4 : Le processeur



## 1. Introduction

Le **processeur** (noté **CPU**, pour Central Processing Unit) est le cerveau de l'ordinateur. C'est un circuit électronique cadencé au rythme d'une horloge interne, qui envoie des impulsions, appelées « top ».

A chaque top d'horloge le processeur exécute une **instruction** (opération élémentaire) ou une partie d'instruction. La puissance du processeur est ainsi caractérisée par le nombre d'instructions qu'il est capable de traiter par seconde. Les instructions sont stockées dans la mémoire centrale, en vue d'être traitée par le processeur.

Un processeur est défini par :

1) la largeur de ses registres internes de manipulation de données (8, 16, 32, 64, 128 bits);
2) la cadence de son horloge exprimée en MHz ou GHz ;
3) le nombre de noyaux de calcul (core) ;
4) son jeu d'instructions (ISA en anglais, Instructions Set Architecture) dépendant de la famille (CISC, RISC, etc) ;
5) sa finesse de gravure exprimée en nm (nanomètres, $10^{-9}$ mètres, soit un milliardième de mètre).


**Quelques exemples de processeurs**

## 2. Rôle du processeur, calcul de CPI (Cycle per Instruction), les processeurs CISC et RISC.

### Rôle du processeur

- Le rôle fondamental de la plupart des processeurs, indépendamment de la forme physique qu'ils prennent, est d'exécuter une série d'instructions stockées appelée programme.
- Les instructions (parfois décomposées en micro-instructions) et les données transmises au processeur sont exprimées en mots binaires (code machine). Elles sont généralement stockées dans la mémoire.
- Le séquenceur ordonne la lecture du contenu de la mémoire et la constitution des mots présentés à l'UAL qui les interprète.

### Calcul de CPI (Cycle par Instruction)

- A chaque top d'horloge, le processeur exécute une action, correspondant à une instruction ou une partie d'instruction.
- L'indicateur appelé **CPI** (*Cycles Par Instruction*) permet de représenter le nombre moyen de cycles d'horloge nécessaire à l'exécution d'une instruction sur un microprocesseur.
- La puissance du processeur peut ainsi être caractérisée par le nombre d'instructions qu'il est capable de traiter par seconde.
- L'unité utilisée est le **MIPS** (Millions d'Instructions Par Seconde) correspondant à la fréquence du processeur que divise le *CPI*.
- La moyenne des cycles par instruction dans un processus donné est définie comme suit:

$$CPI = \frac{\Sigma_i (IC_i)(CC_i)}{IC}$$

  Où ICi est le nombre d'instructions pour un type d'instruction donné i, CCi est les cycles d'horloge pour ce type d'instruction et IC=Σi(ICi) est le nombre total d'instructions. La sommation résume tous les types d'instructions pour un processus d'analyse comparative donné.
- La puissance du processeur peut ainsi être caractérisée par le nombre d'instructions qu'il est capable de traiter par seconde. L'unité utilisée est le MIPS (Millions d'Instructions Par Seconde) correspondant à la fréquence du processeur que divise le CPI.

**Exemple 1:**

Pour le MIPS multi-cycle, il existe cinq types d'instructions :

1. Load (5 cycles)
2. Store (4 cycles)
3. R-type (4 cycles)
4. Branch (3 cycles)
5. Jump (3 cycles)

Si un programme a:

- 50% load instructions
- 25% store instructions
- 15% R-type instructions
- 8% branch instructions
- 2% jump instructions

Alors le CPI est égale à **4.4**

$$\text{CPI} = \frac{5 \times 50 + 4 \times 25 + 4 \times 15 + 3 \times 8 + 3 \times 2}{100} = 4.4$$

**Exemple 2**

Un processeur à 400 MHz a été utilisé pour exécuter un programme de référence avec le mélange d'instructions et le nombre de cycles d'horloge suivants:

| Instruction TYPE | Instruction count | Clock cycle count |
|---|---|---|
| Integer Arithmetic | 45000 | 1 |
| Data transfer | 32000 | 2 |
| Floating point | 15000 | 2 |
| Control transfer | 8000 | 2 |

Determine the effective CPI, MIPS (Millions of instructions per second)rate, and execution time for this program.

$$\text{CPI} = \frac{45000 \times 1 + 32000 \times 2 + 15000 \times 2 + 8000 \times 2}{100000} = \frac{155000}{100000} = 1.55$$

$400Mhz = 400,000,000Hz$

since: $MIPS \propto 1/CPI$ and $MIPS \propto clockFrequency$

$$\text{Effective processor performance} = \text{MIPS} = \frac{\text{clock frequency}}{\text{CPI}} \times \frac{1}{1\text{ Million}} = \frac{400,000,000}{1.55 \times 1000000} = \frac{400}{1.55} = 258\,\text{MIPS}$$

Therefore:

$$\text{Execution time}(T) = \text{CPI} \times \text{Instruction count} \times \text{clock time} = \frac{\text{CPI} \times \text{Instruction Count}}{\text{frequency}} = \frac{1.55 \times 100000}{400 \times 1000000} = \frac{1.55}{4000} = 0.0003875\,\text{sec} = 0.3875\,\text{ms}$$

## Les processeurs CISC et RISC

Ce qui caractérise principalement un processeur est la famille à laquelle, il appartient :

- **Le processeur à architecture CISC (Complex Instruction Set Code)** est un processeur dont le jeu d'instructions possède les propriétés suivantes :

- Il contient beaucoup de classes d'instructions différentes.
- Il contient beaucoup de type d'instructions différentes complexes et de taille variable.
- Il se sert de beaucoup de registres spécialisés et de peu de registres généraux.

- **Le processeur à architecture RISC (Reduced Instruction Set Code)** est un processeur mis en place par IBM dans les années 70, dont le jeu d'instructions possède les propriétés suivantes :

- Le nombre de classes d'instructions différentes est réduit par rapport à un CISC.
- Les instructions sont de taille fixe.
- Il se sert de beaucoup de registres généraux.
- Il fonctionne avec un pipeline

| Architecture RISC | Architecture CISC |
|---|---|
| Instructions simples ne prenant qu'un seul cycle | Instructions complexes prenant plusieurs cycles |
| Instructions au format fixe | Instructions au format variable |
| Décodeur simple (câblé) | Décodeur complexe (microcode) |
| Beaucoup de registres | Peu de registres |
| Seules les instructions LOAD et STORE ont accès à la mémoire | Toutes les instructions sont susceptibles d'accéder à la mémoire |
| Peu de modes d'adressage | Beaucoup de modes d'adressage |
| Compilateur complexe | Compilateur simple |

**Tableau comparatif entre l'architecture RISC et l'architecture CISC**

## 3. Le microprocesseur MIPS R3000

L'**architecture MIPS** (de l'anglais : ***m****icroprocessor without* ***i****nterlocked* ***p****ipeline* ***s****tages*) est une architecture de processeur de type Reduced instruction set computer (RISC) développée par la société *MIPS Technologies*.

Les premières versions de l'architecture MIPS (R2000, R3000) étaient 32-bits (autant au niveau des registres que des chemins de données), mais par la suite, des versions 64-bits sont apparues telle que le R4000 sorti en 1991(le premier processeur 64 bits).

Le processeur MIPS est capable de terminer l'exécution d'une instruction à chaque cycle d'horloge. Pour obtenir ce type de performances, il est nécessaire d'avoir des instructions de taille constante et de construire un pipeline. Cette structure permet d'exécuter chaque instruction en plusieurs cycles, mais de terminer l'exécution d'une instruction à chaque cycle. En plus d'un banc de registres, le MIPS possède des unités séparées pour l'extraction des instructions, le décodage des instructions, l'exécution et les accès mémoire.

Les processeurs MIPS sont notamment utilisés dans des stations de travail (Silicon Graphics, DEC. . .), de nombreux systèmes embarqués (Palm, modems, imprimantes,…), et également des consoles de jeux (Nintendo 64, Sony PlayStation 2, etc.).

## 4. Structure externe du processeur MIPS R3000

L'architecture externe est le niveau d'abstraction nécessaire à l'écriture de programmes assembleur, de la partie génération de code d'un compilateur, et du programmeur de systèmes d'exploitation multiprocessus (et/ou multitâches). Nous allons traiter, en particulier, dans cette section :

- les registres visibles du logiciel ;
- l'adressage de la mémoire ;
- les appels systèmes.

Les jeux d'instructions seront traités dans une section à part.

Le processeur possède deux modes :

- le mode utilisateur (ou user) pour exécuter les applications, et
- le mode noyau (ou kernel) pour exécuter le système.

Ces 2 modes sont nécessaires à l'exécution sure de plusieurs processus sur un même processeur.

## 1) les registres visibles du logiciel :

Les registres du MIPS R3000 visibles du logiciel, c.-à-d. qui sont manipulés par les instructions implicitement ou explicitement, ont tous une taille de 32 bits.
Le MIPS R3000 visant par construction l'exécution de multiples processus, des mécanismes de protections sont mis en œuvre pour l'accès aux registres relatifs au système. Ces derniers registres appartiennent à un coprocesseur système dit coprocesseur 0 ou *cop0*. Les accès à ce coprocesseur ne peuvent avoir lieu qu'en mode noyau.
Le processeur possède deux modes de fonctionnement (Utilisateur/Superviseur) imposant ainsi d'avoir deux catégories de registres :

**a) Registres non protégés (Registres du processeur):** Le processeur possède **35** registres manipulés par les instructions standards, dont :

- 32 registres de **32 bits** numérotés **$0; : : : ; $31**. Les registres peuvent être accédés soit par leur numéro soit par leur nom.

| Nom | Numéro | Description |
|---|---|---|
| $zero | $0 | constante zéro |
| $at | $1 | réserve pour l'assembleur |
| $v0 | $2 | retour de fonction |
| $v1 | $3 | retour de fonction |
| $a0 | $4 | argument de fonction |
| $a1 | $5 | argument de fonction |
| $a2 | $6 | argument de fonction |
| $a3 | $7 | argument de fonction |
| $t0 | $8 | temporaire |
| $t1 | $9 | temporaire |
| $t2 | $10 | temporaire |
| $t3 | $11 | temporaire |
| $t4 | $12 | temporaire |
| $t5 | $13 | temporaire |
| $t6 | $14 | temporaire |
| $t7 | $15 | temporaire |

| Nom | Numéro | Description |
|---|---|---|
| $s0 | $16 | sauvegardé |
| $s1 | $17 | sauvegardé |
| $s2 | $18 | sauvegardé |
| $s3 | $19 | sauvegardé |
| $s4 | $20 | sauvegardé |
| $s5 | $21 | sauvegardé |
| $s6 | $22 | sauvegardé |
| $s7 | $23 | sauvegardé |
| $t8 | $24 | temporaire |
| $t9 | $25 | temporaire |
| $k0 | $26 | pour noyau système |
| $k1 | $27 | pour noyau système |
| $gp | $28 | pointeur global |
| $sp | $29 | pointeur de pile |
| $fp | $30 | pointeur de frame |
| $ra | $31 | registre d'adresse |

- **pc , *program counter*** ce registre contient l'adresse de l'instruction à exécuter. Sa valeur est modifiée par toutes les instructions ;
- **ir , *instruction register*** ce registre contient l'instruction en cours d'exécution. Il n'est pas directement accessible
- **hi** et **lo** ces registres contiennent le résultat de la multiplication sur 64 bits, ou le résultat de la division euclidienne (quotient dans **lo** et reste dans **hi**).


**Registres non protégés MIPS R3000**

**Exemples :**
Pour les divisions, on peut donner l'exemple suivant :

```
addi $t3, $0, 37
addi $t4, $0, 2
div $t3, $t4
mflo $t0 #équivaut à 37 / 2 équivaut à 18 (le quotient)
mfhi $t1 #équivaut à 37 % 2 équivaut à 1 (le reste)
```

Et pour les multiplications, on peut donner l'exemple suivant :

```
addi $t3, $0, 37
addi $t4, $0, 2
mult $t3, $t4
mflo $t1 #on y trouve le produit mais il faut que le produit soit sur 32 bits
```

**b) Registres protégés (Registres du coprocesseur 0 ) :** Ces registres concernent la gestion des exceptions, interruptions et appels systèmes :

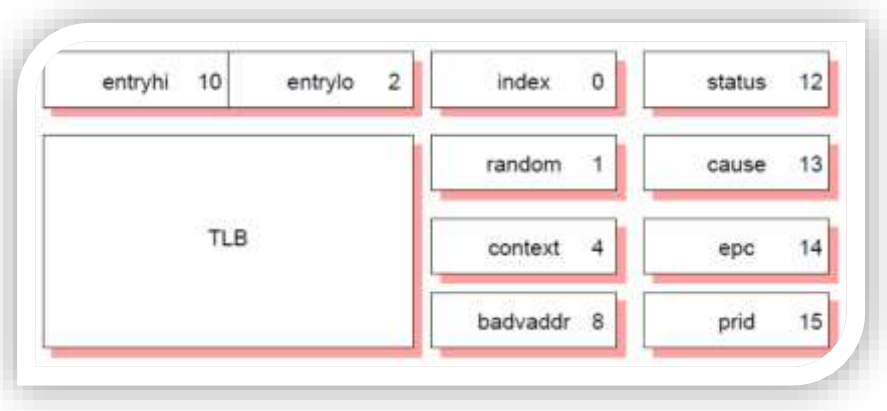
- **badvaddr , *bad virtual address*** ce registre contient l'adresse fautive en cas d'exception de type « adresse illégale » ;
- **SR Registre d'état (Status Register)** c'est le registre d'état. Il contient les masques d'interruption et le mode ;
- **CR Registre de cause** c'est le registre qui contient la cause de l'exception ;
- **EPC Registre d'exception (*Exception Program Counter)*** ce registre contient l'adresse de retour en cas d'interruption et l'adresse de l'instruction fautive en cas d'exception ou d'appel système.

Les registres suivants concernent la gestion de la mémoire virtuelle.

- **TLB, *Transaltion Lookaside Buffer*** c'est la mémoire associative pour la traduction adresse virtuelle vers adresse physique ;
- **index** registre contenant l'index de la **TLB** dans lequel faire les accès ;
- **random** registre contenant un index aléatoire valide pour les accès à la **TLB** ;
- **context** registre utilisé partiellement par le logiciel (une partie sert à pointer sur la structure des pages du système d'exploitation) et par le matériel (l'autre partie

contient les poids forts de l'adresse fautive lors d'une traduction qui échoue) pour faciliter l'écriture en logiciel de la gestion de la mémoire virtuelle ;

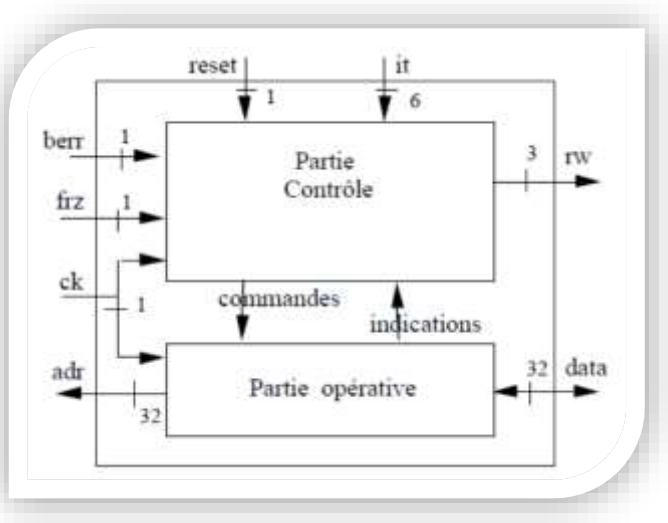
- **entryhi** et **entrylo** valeur à comparer aux entrées de la **TLB** pour savoir s'il y a une erreur de page (***page fault***).




**Registres protégés (du coprocesseur 0) du MIPSR3000**

## 5. Structure interne du processeur MIPS R3000

- L'architecture interne du processeur se décompose en une partie opérative et une partie contrôle.
- La partie opérative (PO) contient les registres et les opérateurs. Elle réalise des transferts élémentaires de données entre un ou plusieurs registres sources et un registre destination.
- Un transfert élémentaire est exécuté en un cycle.
- La partie opérative est commandée par la partie contrôle (PC).
- La partie contrôle est chargée de définir, pour chaque cycle d'horloge, les transferts élémentaires qui doivent être réalisés par la partie opérative.
- La partie contrôle contient principalement un séquenceur décrit comme un automate d'états finis (automate de MOORE).
- Les entrées de cet automate sont les signaux indicateurs envoyés par la partie opérative, et dont les sorties sont les signaux de commande constituant la micro-instruction.
- Cet automate est décrit de façon graphique. Les nœuds (ellipses) représentent les états, les flèches représentent les transitions entre états, qui peuvent dépendre des signaux provenant de la partie opérative.
- Chaque état est identifié par un nom.
- A chaque état de l'automate est associée une micro-instruction, qui porte le nom de cet état.
- Toute instruction du langage machine se décompose en une séquence de microinstructions.
- Toutes les instructions commencent par la même micro-instruction, qui réalise l'incrémentation du compteur ordinal.
- Ensuite le comportement de l'automate dépend de la valeur du code opération (IR [31:26]), qui fait partie des entrées de l'automate.
- Enfin, par convention, la dernière micro-instruction d'une instruction i effectue la lecture de l'instruction i+1.



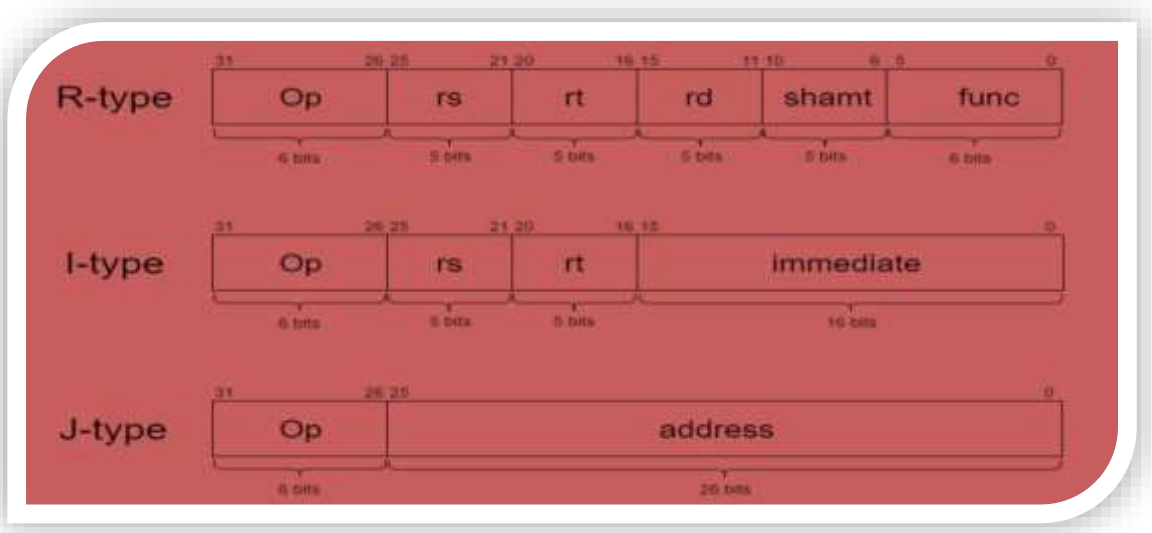

**L'architecture interne du processeur MIPS R3000**

## 6. Jeu d'instructions, Formats et programmation du MIPS R3000

Trois (3) formats d'encodage sont possibles dans MIPS :

Chaque instruction MIPS est contenue sur **32 bits**, l'encodage des **3** types d'instruction possibles est décrit plus haut.

1) **Les instructions de type R (R** pour **R**egistre**)** utilisent **3** registres pour les opérations arithmétiques et logiques, **2** registres sources comme opérandes et un registre destination pour le résultat, **5 bits** sont utilisés pour coder le numéro du registre utilisé.
2) **Les instructions de type** I (**I** pour **i**mmédiate) utilisent aussi **2** opérandes et un registre pour le résultat, la différence avec le type **R** est que l'un de ces opérandes est une immédiate sur **16 bits**, ils sont utilisés pour différents types d'instructions incluant aussi des opérations arithmétiques et logiques.
3) **Les instructions de type J (J** pour **J**ump, ou saut**)** sont réservées pour le saut.




**Types d'instruction MIPS**

MIPS R3000 possède un jeu de 57 instructions très simples qui se répartissent en 4 classes :

1) 32 instructions arithmétiques/logiques entre registres (calcul)
2) 12 instructions de branchement (saut)
3) 8 instructions de lecture/écriture mémoire (transfert)
4) 5 instructions systèmes

## 1) Instructions de calcul

Ces instructions lisent la valeur de 0, 1 ou 2 registres dits **arguments**, effectuent un calcul, puis écrivent le résultat dans un registre dit **destination**.
Un même registre peut figurer plusieurs fois parmi les arguments et destination.

**Exemples :**

**- Les instructions de calcul nullaires**

- **Ecriture d'une constante** (Load Immediate) :

**li** dest, constant
Produit la constante *constant*.
On a également : **la** (load adrdess)**.**

**- Les instructions de calcul unaires**

- **Addition d'une constante** (Add Immediate):

**addi** dest, source, constant
Produit la somme de la constante (de 16 bits) *constant* et du contenu du registre *source*.

- **Déplacement** (Move) :

**move** dest, source
Produit le contenu du registre *source*. Cas particulier de *addi!*

- **Négation** (Negate) :

**neg** dest, source
Produit l'opposé du contenu du registre *source*. Cas particulier de *sub!*

**- Les instructions de calcul binaires**

- **Addition** (Add):

**add** dest, source1, source2
Produit la somme des contenus des registres *source1* et *source2*.
On a également : **sub, mul, div.**

- **Comparaison** (Set On Less Than):

**slt** dest, source1, source2
Produit 1 si le contenu du registre *source1* est inférieur à celui du registre *source2* ; produit 0 sinon.
On a également : **sle, sgt, sge, seq, sne**.

## 2) Instructions de saut

On distingue les instructions de saut selon que :

- Leurs destinations possibles sont au nombre de 1 (saut **inconditionnel)** ou bien 2 (saut **conditionnel**);
- Leur adresse de destination est **constante** ou bien **lue** dans un registre ;
- Une **adresse de retour** est sauvegarde´ e ou non.

**Exemples :**

**- Saut inconditionnel**

- **Saut** (Jump):

**j** address

Saute à l'adresse constante *address*. Celle-ci est en général donnée sous forme symbolique par une *étiquette* que l'assembleur traduira en une constante numérique.

**- Saut conditionnel**

 **Saut conditionnel unaire** (Branch on Greater Than Zero):

**bgtz** source, address

Si le contenu du registre *source* est supérieur à zéro, saute à l'adresse constante *address*.

On a également **bgez, blez, bltz.**

 **Saut conditionnel binaire** (Branch On Equal):

**beq** source1, source2, address

Si les contenus des registres *source1* et *source2* sont égaux, saute à l'adresse constante *address*.

On a également **bne.**

**- Saut avec retour**

 **Saut avec retour** (Jump And Link):

**jal** address

Sauvegarde l'adresse de l'instruction suivante dans le registre *ra*, puis saute à l'adresse constante *address.*

**- Saut vers adresse variable**

 **Saut vers adresse variable** (Jump Register)

**jr** target

Saute à l'adresse contenue dans le registre *target*. L'instruction **jr $ra** est typiquement employée pour rendre la main à l'appelant à la fin d'une fonction ou procédure.

### 3) Instruction de transfert

 **Lecture** (load word):

**lw** dest, offset(base)

On ajoute la constante (de 16 bits) *offset* à l'adresse contenue dans le registre *base* pour obtenir une nouvelle adresse ; le mot stocké à cette adresse est alors transféré vers le registre *dest*.

On a également: **lb** (load byte)**, lbu** (load byte unisgned)**, lh** (load half)**, lhu** (load half unisgned)**.**

 **Ecriture** (store word):

**sw** source , offset(base)

On ajoute la constante (de 16 bits) *offset* à l'adresse contenue dans le registre *base* pour obtenir une nouvelle adresse ; le mot stocké dans le registre *source* est alors transféré vers cette adresse.

On a également: **sb** (store byte)**, sh** (store half)**.**

### 4) Instructions système

 Un appel système se fait par l'instruction **syscall**.

 Les simulateurs fournissent un ensemble de services par l'intermédiaire de l'instruction d'appel **syscall**, dont le comportement dépend de la valeur du registre **$v0** :

- si **$v0 = 1**, alors **syscall** affiche l'entier contenu dans le registre **$a0** ;
- si **$v0 = 4**, alors **syscall** affiche la chaîne de caractère dont l'adresse est contenue dans le registre **$a0** ;
- si **$v0 = 5**, alors **syscall** lit un entier à l'écran et met sa valeur dans le registre **$v0**.

## 7. Programmation du MIPS R3000

### a) Syntaxe d'un programme MIPS:

- Les commentaires commencent par le symbole # et se terminent à la fin de la ligne.
- Un identificateur est une séquence de caractères alphanumériques, de soulignés (_) et de points (.), qui ne commence pas par un chiffre.
- Les codes opération d'instruction sont des mots réservés qui ne peuvent pas être utilisés comme identificateurs.
- Les étiquettes sont déclarées en les plaçant au début d'une ligne et en les faisant suivre du symbole (:).
- Les nombres sont en base **10** par défaut. S'ils sont précédés de **0x** ils sont interprétés comme hexadécimaux.
- Les chaînes de caractères sont encadrées par des doubles apostrophes (").
- Certains caractères spéciaux dans les chaînes de caractères suivent la convention **C**:
- Retour-chariot : **\n**
- Tabulation : **\t**
- Guillemet : **\"**

### b) La structure de base d'un programme MIPS

- section « **.data** », contient les déclarations de données, c.-à-d. les données globales manipulées par le programme utilisateur. Elle est implantée conventionnellement à l'adresse 0x10000000. Sa taille est fixe et calculée lors de l'assemblage. Les valeurs contenues dans cette section peuvent être initialisées grâce à des directives contenues dans le programme source en langage d'assemblage ;
- La section « **.text** », (code du programme) contient le code exécutable en mode utilisateur. Elle est implantée conventionnellement à l'adresse 0x00400000. Sa taille est fixe et calculée lors de l'assemblage. La principale tâche de l'assembleur consiste à générer le code binaire correspondant au programme source décrit en langage d'assemblage, qui sera chargé dans cette section ;
- « **main** », début du programme.
- Pour sortir du programme **MIPS**, on fait un appel system « **li $v0,10 syscall** ».
- Les commentaires permettent de donner plus d'explication sur le code. Ils commencent par un « # » ou un « **;** ».

**Exemple d'un simple programme MIPS**

```
1   # Exemple0
2   .data
3   # Il n'y aura pas de données à déclarer, on utilisera le mode immédiat
4   .text
5   main:
6   li $t0,3 # charger la valeur 3 dans le registre t0
7   li $t1,8 # charger la valeur 8 dans le registre t1
8   add $t2,$t1,$t0 # Faire l'addition de t0+t1 et mettre le résultat dans le registre t2
9   #Fin du programme
10  li $v0,10
11  syscall
```

### c) Quelques directives

| | |
|---|---|
| ***.ascii str*** | Enregistre en mémoire la chaîne de caractères str, mais ne la termine pas par un caractère nul. |
| *.asciiz str* | Enregistre en mémoire la chaîne de caractères **str** et la termine par un caractère nul. |
| *.data<@>* | Les éléments qui suivent sont enregistrés dans le segment de données. Si l'argument optionnel **@** est présent, les éléments qui suivent sont enregistrés à partir de l'adresse **@**. |
| *.byte b1; : : : ;bn* | Enregistre les **n** valeurs dans des octets consécutifs en mémoire. |
| *.word w1; : : : ;wn* | Enregistre les **n** quantités **32 bits** dans des mots consécutifs en mémoire. |
| *.float f1; : : : ;fn* | Enregistre les **n** nombres flottants simples précision dans des emplacements mémoire consécutifs. |
| *.text <@>* | Les éléments qui suivent sont placés dans le segment de texte de l'utilisateur. Dans **SPIM**, ces éléments ne peuvent être que des instructions ou des mots. Si l'argument optionnel **@** est présent, les éléments qui suivent sont enregistrés à partir de l'adresse **@**. |
| *.globl sym* | Déclare que le symbole **sym** est global et que l'on peut y faire référence à partir d'autres fichiers. |

### d) Appel de procédure - Conventions

- Par convention, lors de l'appel de procédure, les registres **$t0,: : :,$t9** sont sauvegardés par l'appelant et peuvent donc être utilisés sans problème par l'appelé. Les registres **$s0,: : :,$s7** doivent quant à eux être sauvegardés et restitués exact par l'appelé.
- La pile croit des adresses hautes vers les adresses basses : on soustrait à **$sp** pour allouer de l'espace dans la pile, on ajoute à **$sp** pour rendre de l'espace dans la pile.
- Les déplacements dans la pile se font sur des mots mémoire entiers (multiples de quatre octets).
- Lors du passage de paramètres : tout paramètre plus petit que **32 bits** est automatiquement promu sur **32 bits**.
- Les quatre premiers paramètres sont passés par les registres **$a0,: : :,$a3**. Les paramètres supplémentaires sont passés dans la pile.
- Toute valeur de format inférieur ou égal à **32 bits** est retournée par le registre **$v0** (sur **64 bits $v1** est utilisé avec **$v0**).

## 8. Conclusion

Nous avons présenté dans ce chapitre le rôle du processeur, comment calculer de CPI (Cycle per Instruction), les architectures CISC et RISC ainsi que quelques détails sur le

microprocesseur MIPS R3000 (Structure interne, structure externe, jeu d'instructions) et autres.
D'autres détails concernant la gestion des exceptions et des interruptions, les entrées-sorties ainsi que les instructions systèmes du MIPS R3000 seront détaillés dans le chapitre suivant.

# Chapitre 5 : Instructions spéciales



## 1. Introduction

Les interruptions et les exceptions sont deux types de rupture de séquence exceptionnelle. Les interruptions surviennent de façon asynchrone. Elles sont commandées par le matériel (interruption externe) ou par le système (interruption système).
Les interruptions n'ont aucune relation avec les instructions en cours d'exécution.
Les exceptions sont déclenchées par des accidents dans l'exécution du programme : débordement arithmétique, erreur de bus, tentative d'utilisation d'instructions réservées, erreur d'adressage, défaut de cache, défaut de page...). Les exceptions provoquent un traitement spécifique : appel à une routine spécialisée. On parle de gestion précise des exceptions si l'état de la machine qui résulterait de l'exécution séquentielle de toutes les instructions antérieures à l'instruction provoquant l'exception peut être reconstruit dans tous les cas. Il est alors possible après le traitement de l'exception de reprendre l'exécution à l'instruction même ayant provoqué l'exception. Dans le cas contraire, les exceptions sont gérées de façon imprécise.

## 2. Notions sur les interruptions

- Certains processeurs possèdent un registre d'état, dont la valeur est mise à jour après l'exécution de chaque instruction, par exemple pour indiquer un fonctionnement normal ou un débordement.
- Le MIPS ne possède pas de registre d'état, mais utilise un système d'exceptions qui, selon les cas, peuvent amener à une interruption de l'exécution du programme.
- La partie contrôle contient neuf bascules 1 bit correspondant aux sept types d'exception définis dans l'architecture du processeur MIPS R3000 (ADEL, ADES, DBE, IBE, OVF, RI, CPU) et aux deux appels systèmes correspondant aux instructions SYSCALL et BREAK (SYS, BRK).
- Ces bascules sont mises à un en cas d'exception lorsque la détection de l'exception correspondante est autorisée par le champ EXCP de la micro-instruction.
- Elles sont toutes remises à zéro si le champ EXCP contient la commande E_CLR, ce qui est normalement fait par la première micro-instruction du microprogramme de branchement au gestionnaire d'exceptions.
- En cas d'exception, le microprogramme de l'instruction fautive se déroule "normalement", jusqu'à la dernière micro-instruction. Cependant, toutes les écritures dans les registres visibles du logiciel et les accès à la mémoire sont inhibées dès qu'une bascule d'exception est à 1.

- La dernière micro-instruction d'une instruction teste s'il n'y a pas une interruption ou une exception pendante.
- Une interruption est pendante si l'une des lignes IT5 à IT0 est active.
- Une exception est pendante si une des neuf bascules d'exception est à 1. Si c'est le cas, au lieu de démarrer l'exécution de l'instruction suivante, l'automate exécute la séquence de micro-instructions permettant de se brancher au gestionnaire d'exceptions.
- Ce microprogramme comporte quatre micro-instructions distinctes et effectue les actions suivantes :
  - passage en mode superviseur et masquage des interruptions dans SR.
  - écriture de la cause du déroutement dans CR et CLR des bascules d'exception.
  - sauvegarde du PC dans EPC.
  - chargement de l'adresse 80000080 dans PC.
  - sauvegarde de AD dans BAR.
  - chargement du registre instruction.
- Dans le cas où une exception et une interruption sont pendantes simultanément, l'exception est traitée en premier.
- Le RESET est traité de la même manière, mais le microprogramme de branchement au gestionnaire d'initialisation est plus simple (trois micro-instructions).
- L'automate complet comporte 106 états.
- Plusieurs exceptions peuvent survenir lors de l'exécution d'un programme : débordements, mémoire non allouée, etc. Par exemple la somme de deux valeurs de 32 bits ne tient pas forcément sur 32 bits. La valeur contenue dans le registre de destination n'est pas alors pas correcte :

**Exemple :**

```
LI $8, 0xA1234567
LI $9, 0xA1234567
ADD $10, $8, $9      # 0xA1234567 + 0xA1234567 = 0x142468ACE
                     # mais $10 ne peut contenir que 0x42468ACE (32 bits)
                     # débordement => la valeur du registre est fausse!
```

## 3. Les entrées/sorties

### Mode appel système

Le mécanisme d'entrée/sortie de données, par exemple dans la console ou sur un périphérique externe, consiste à interrompre l'exécution du programme pour faire exécuter un service par le système.

Le principe général est le suivant :

1. Placer un code de service dans le registre **$v0** ;
2. Appeler l'instruction **syscall** : l'exécution du programme est interrompue et le système réalise la tâche dont le code est présent dans **$v0**. Une fois le service exécuté, l'exécution reprend.

Les paramètres des services sont lus dans les registres **$a0** à **$a3** (par exemple les valeurs à afficher), et les résultats des services sont placés dans **$v0** et **$v1** à la fin de l'exécution (par exemple les valeurs lues). Les services les plus courants, notamment d'entrée/sortie, sont les suivants :

- Pour demander un service, on charge le code du service (voir tableau ci-dessous) dans le registre **$v0** et ses arguments dans les registres **$a0,...,$a3** (ou **$f12** pour les valeurs flottantes).
- Les appels système qui retournent des valeurs placent leurs résultats dans le registre **$v0** (ou **$f0** pour les résultats flottants).

Le tableau suivant illustre les différents codes de **syscall**.

| Service | Code | Arguments | Résultat |
|---|---|---|---|
| **print_int** | 1 | $a0 = entier | |
| **print_float** | 2 | $f12 = flottant simple précision | |
| **print_double** | 3 | $f12 = flottant double précision | |
| **print_string** | 4 | $a0 = chaine de caractères | |
| **read_int** | 5 | | Un entier dans $v0 |
| **read_float** | 6 | | Un flottant simple dans $v0 |
| **read_double** | 7 | | Un flottant double dans $v0 |
| **read_string** | 8 | $a0 = tampon**,** $a1 = longeur | |
| **sbrk** | 9 | $a0 = quantité | Une adresse dans $v0 |
| **exit** | 10 | | |

**Exemple 1 : print_int (Appel système code $v0=1)**

Le code suivant imprime « **La réponse = 5** » dans la console.

```
.data
Str : .asciiz    "La reponse = "
.text
main:
la $a0, Str      # Mettre l'adresse de la chaine de caractère Str dans le registre $a0
li $v0, 4        # Charger le registre $v0 avec le code 4 pour affichage caractère (print_string)
syscall          # appel système pour affichage caractère
li $a0, 5        # Mettre la valeur à afficher dans le registre $a0
li $v0, 1        # Charger le registre $v0 avec le code 1 pour affichage entier (print_int)
syscall          # appel system pour l'opération affichage entier
# Fin du Programme
li $v0, 10
syscall
```

**Exemple2 : print_string (Appel système code $v0 =4)**

Le code suivant affiche le message « **Hello Word** » dans la console.

```
.data
Hello: .asciiz "Hello Word\n" # mettre la chaine de caractère « Hello Word » en mémoire
.text
main:           # section <code
la $a0, Hello   # charger le registre $a0 avec l'adresse de la chaine de caractère
li $v0, 4       # Charger le registre $v0 avec le code 4 pour affichage caractère
                #(print_string).
syscall         # appel system pour l'opération d'affichage
# Fin du Programme
li $v0,10
syscall
```

### Exemple3 : read_int (Appel système code $v0=5)

Le code suivant affiche sur l'écran un entier saisi au le clavier.

```
.data
Str : .asciiz " Saisir un entier "
.text
main:
la $a0,Str  # Mettre l'adresse de la chaine de caractère Str dans le registre $a0
li $v0,4    # Charger le registre $v0 avec le code 4 pour affichage caractère (print_string)
syscall     # Appel système pour affichage caractère
li $v0,5    # Charger le registre $v0 avec le code 5 pour saisir un entier à partir du clavier
syscall     # Appel system pour l'opération lire entier (read_int)
move $a0,$v0 # déplacer la valeur saisie dans le register $a0
li $v0,1    # Charger le registre $v0 avec le code 1 pour affichage entier (print_int)
syscall     # Appel system pour l'opération affichage entier
# Fin du Programme
li $v0,10
syscall
```

### Exemple4 : read_string (Appel système code $v0=8)

Le code suivant affiche le message écrit par l'utilisateur.

```
.data
message: .asciiz "Vous avez écrit "
UserInput: .space 20
.text
main:
# Entrer un text de 20 caractères
li $v0, 8
la $a0, UserInput
li $a1, 20
syscall
# Afficher le message « Vous avez écrit »
li $v0,4
la $a0,message
syscall
# Afficher le message
li $v0,4
la $a0,UserInput
syscall
# Fin du programme
li $v0,10
syscall
```

## Mode interruption

Lorsque le périphérique est prêt à effectuer un échange élémentaire, il envoie un signal au CPU pour "interrompre" l'exécution en cours. Le contrôle du CPU est renvoyé à un programme de traitement de l'évènement stocke en mémoire à partir d'une adresse liée à l'interruption.

- Le CPU n'attends pas ;
- Temps perdu à échanger les programmes et leur contexte.
- Un dispositif, incorpore au niveau du séquenceur, qui enregistre et traite les signaux d'interruption envoyés au CPU

Le Cycle d'interruption peut être défini comme suit :

1) Arrêter le programme en cours ;
2) Sauvegarder l'état de la machine ;
3) Exécuter le programme de service de l'interruption ;
4) Rétablir l'état de la machine ;
5) Reprendre l'exécution du programme interrompu.

## 4. Conclusion

L'objectif de ce chapitre était de présenter quelques instructions spéciales permettant de gérer les exceptions et les interruptions, ainsi que celles permettant la gestion des entrées-sorties et les instructions systèmes dans un processeur MIPS R3000.

Nous avons essayé de décrire ces différentes instructions à l'aide d'exemples qui ont été traités lors des séances de travaux pratiques et ce afin d'expliquer leurs rôles et de faciliter leur compréhensions.

# Partie II – Exercices

# Chapitre 1 : Structure de base d'un ordinateur

## Exercice 1 :

Un *ordinateur* est une machine de traitement de l'information (texte, image, signal, vidéo). Expliquer brièvement son principe de fonctionnement.

## Solution :

L'ordinateur est capable d'**acquérir** de l'information, de la **stocker**, de la **transformer** en effectuant des traitements quelconques, puis de la **restituer** sous une autre forme :

- Il peut recevoir des données en entrée ⇒ «*Fonction d'entrée*»,
- Effectuer sur ces données des opérations en fonction d'un programme ⇒ «*Fonction de traitement*»
- Et enfin fournir des résultats en sortie ⇒ «*Fonction de sortie*»

## Exercice 2 :

Citer les principaux composants d'un ordinateur

## Solution :

Les principaux composants d'un ordinateur sont :

1. Le processeur (ou UC unité centrale, ou *CPU* pour *Central Processing Unit*)
2. Les mémoires
3. Les dispositifs d'entrée-sortie

Ces éléments étant interconnectés entre eux par des bus.

## Exercice 3 :

**1.** Compléter les schémas suivants en indiquant que représente chaque schéma.

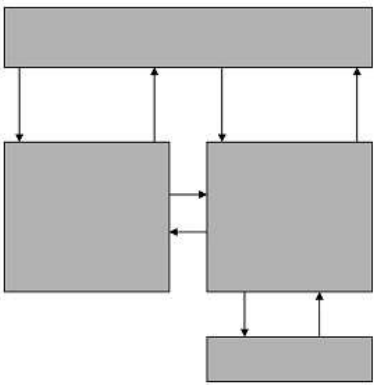


Schéma 1

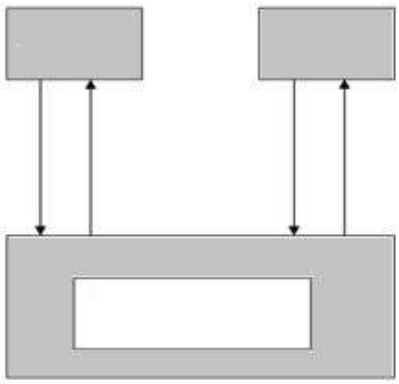


Schéma 2

**2.** Quelle est la différence entre les deux modèles ?

## Solution :

1. Le schéma 1 représente l'architecture de **Von Neumann** et le schéma 2 représente l'architecture de **Harvard**.

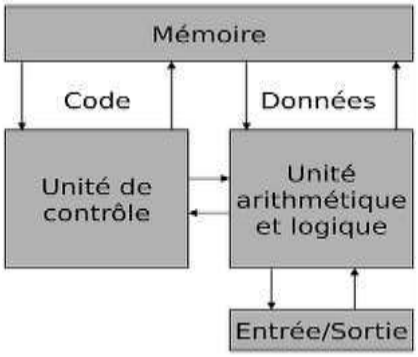



**Schéma 1**

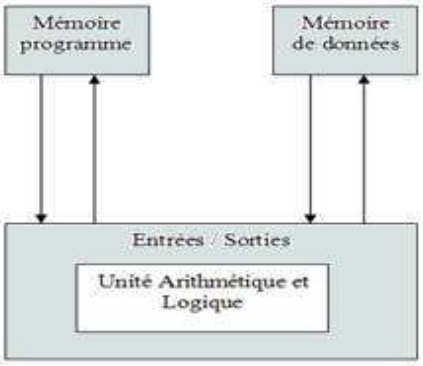



**Schéma 2**

**2.** La différence essentielle entre ces deux architectures est la manière d'accéder aux mémoires.

- Dans une architecture Von Neumann, les programmes et les données sont stockés dans le même mémoire et géré par le même sous-système de traitement de l'information.
- Dans l'architecture de Harvard, les programmes et les données sont stockés et gérés par différents sous-systèmes.

La différence entre l'architecture Von Neumann et l'architecture Harvard peut être résumée dans le tableau suivant :

| Architecture de Von Neumann | Architecture de Harvard |
|---|---|
| Elle porte le nom du mathématicien et informaticien John Von Neumann | Le nom provient de «Harvard Mark I», un ancien ordinateur à relais, projet réalisé à l'université Harvard |
| Elle a besoin d'une seule mémoire pour les instructions et les données. | Elle a besoin de deux mémoires pour les instructions et les données |
| La conception de l'architecture de Von Neumann est simple. | La conception de l'architecture de Harvard est compliquée. |
| Ne requiert qu'un seul bus pour les instructions et les données. | Nécessite un bus séparé pour les instructions et les données. |
| Le processeur a besoin de deux cycles d'horloge pour terminer une instruction. | Le processeur peut compléter une instruction en un cycle |
| Faible performance par rapport à l'architecture de Harvard. | Plus facile à canaliser, donc de hautes performances peuvent être atteintes. |
| Coût moins cher. | Coût relativement élevé. |

## Chapitre 2 : Principaux composants d'un ordinateur

### Exercice 1 :

Associer chaque composant de la liste suivante avec une des définitions proposées :

| Composant | Définition |
|---|---|
| **a. Carte mère** | **1.** Ensemble de fils permettant de lier et faire communiquer les composants d'un ordinateur afin d'assurer la transmission du même type d'information (données, adresses ou commandes). |
| **b. Chipset** | **2.** Permettant l'exécution des instructions du programme et des calculs sur les données. |
| **c. Bus** | **3.** Circuit électronique chargé de coordonner les échanges de données entre les divers composants de l'ordinateur (processeur, mémoire...). |
| **d. Séquenceur** | **4.** Identifier l'instruction à exécuter qui se trouve dans le Registre d'Instruction (RI). |
| **e. Software** | **5.** Permettent l'échange d'informations avec les dispositifs extérieurs. |
| **f. Processeur** | **6.** Dispositif de stockage de données et de programme. |
| **g. Hardware** | **7.** Constituée de l'ensemble des programmes pouvant être un programme d'application ou un programme de pilotage ou de base d'une machine |
| **h. Mémoire** | **8.** Constituée de l'ensemble des composants physique d'une machine. |
| **i. Périphérique d'entrée / de sortie** | **9.** Socle permettant la connexion de l'ensemble des éléments essentiels de l'ordinateur. |
| **g. Décodeur** | **10**. Chargé de synchroniser l'exécution des instructions au rythme d'une horloge. |

### Solution :

| Composant | Définition |
|---|---|
| **a. Carte mère** | **9** |
| **b. Chipset** | **3** |
| **c. Bus** | **1** |
| **d. Séquenceur** | **10** |
| **e. Software** | **7** |
| **f. Processeur** | **2** |
| **g. Hardware** | **8** |
| **h. Mémoire** | **6** |
| **i. Périphérique d'entrée / sortie** | **5** |
| **g. Décodeur** | **4** |

## Exercice 2

Quelle est la signification de chacun des acronymes suivants :

1) UAL : ……...
2) CPU : ……...
3) RAM : ……..
4) ROM : ……..
5) USB : ……
6) VGA : ……...
7) HDMI : ……

## Solution :

1) UAL : **Unité Arithmétique et Logique**
2) CPU : **Central Processing Unit**
3) RAM : **Random Access Memory**
4) ROM : **Read Only Memory**
5) USB : **Universal Serial BUS**
6) VGA : **Video Graphics Array**
7) HDMI : **High Definition Multimedia Interface**

## Exercice 3

Un ordinateur est équipé d'un processeur Pentium 4 à 3,6 GHz fonctionnant à une fréquence de carte mère de 800 MHz.

- Calculer le taux de transfert maximal du bus processeur sachant que la quantité de données pouvant être transférées simultanément est de 64 bits.

## Solution :

Pour calculer le taux de transfert du bus processeur, il faut multiplier la quantité de données pouvant être transférées simultanément (64 bits) par la fréquence d'horloge du bus (identique à la fréquence du processeur avant multiplication).
Pour obtenir le taux de transfert, on doit utiliser la formule suivante :

**Taux de transfert (Mo/s) = Fréquence (en MHz) × Largeur du bus (en octet)**

D'où **Taux de transfert (Mo/s)** = 800 MHz × 8 octets (64 bits) = **6400 Mo/s.**
D**'**où le taux de transfert instantané maximal est de **6400 Mo/s.**

## Exercice 4

Soit un bus PCI 64 bits tournant à 64 Mhz

1. Calculer le taux maximum (théorique) de transfert.
2. Exprimer le résultat obtenu en Méga Byte/s et Méga Octets/s.

## Solution :

**1. Le taux maximum (théorique) de transfert**

**Taux de transmission ou débit (en bits/s) = largeur bus (en bits) x Fréquence (MHz)**

Taux de transmission = 64 X 64 = **4096 M bits/s**

**2. Le résultat obtenu en Méga Byte/s et Méga Octets/s**

Taux de transmission = 4096 M bits/s = 4096/8 =**512 M byte/s** (car 1 byte =8bit)
Taux de transmission = 4096 M bits/s = 4096/8 =**512 M octets/s** (car 1 octet =8 bits)

## Exercice 5

Calculer les taux de transferts pour les bus mémoires suivants :

| | DRAM | SDRAM | SDRAM PC100 | DDR SDRAM PC2100 |
|---|---|---|---|---|
| **Largeur du bus (bits)** | 32 | 64 | 64 | 64 |
| **Fréquence du bus (MHz)** | 66 | 66 | 100 | 133 |
| **Taux de transfert (Mo/s)** | | | | |

## Solution :

Les taux de transferts pour les bus mémoires donnés sont calculés à l'aide de :

**Taux de transmission (en Mo/s) = (largeur bus (en bits) x Fréquence (en MHz)) / 8**

| | DRAM | SDRAM | SDRAM PC100 | DDR SDRAM PC2100 |
|---|---|---|---|---|
| **Largeur du bus (bits)** | 32 | 64 | 64 | 64 |
| **Fréquence du bus (MHz)** | 66 | 66 | 100 | 133 |
| **Taux de transfert (Mo/s)** | **264** | **528** | **800** | **2128 (1064x2 pour PC 2x100)** |

## Exercice 6

Calculer les taux de transfert pour les bus périphériques suivants :

| | ISA | EISA | PCI | AGP | AGP 4x |
|---|---|---|---|---|---|
| **Largeur du bus (bits)** | 16 | 32 | 32 | 32 | 32 |
| **Fréquence du bus (MHz)** | 8.33 | 8.33 | 33.33 | 66.66 | 66.66 |
| **Taux de transfert (Mo/s)** | | | | | |

## Solution :

Les taux de transfert pour les bus périphériques donnés sont calculés à l'aide de :

**Taux de transmission (en Mo/s) = (largeur bus (en bits) x Fréquence (en MHz)) / 8**

| | ISA | EISA | PCI | AGP | AGP 4x |
|---|---|---|---|---|---|
| **Largeur du bus (bits)** | 16 | 32 | 32 | 32 | 32 |
| **Fréquence du bus (MHz)** | 8.33 | 8.33 | 33.33 | 66.66 | 66.66 |
| **Taux de transfert (Mo/s)** | **16.66** | **33.32** | **133.32** | **266.64** | **1066.56** = 266.64 x4 |

**Exercice 7**

A quoi servent les registres suivants du processeur :

1. PC/IP (ou CO/PI)
2. IR (ou RI)
3. SP (ou PP)
4. Accumulateur (Acc)
5. PSW (ou RE)

**Solution :**

1. PC/IP (ou CO/PI) : Program Counter /Instruction Pointer (Compteur Ordinal/Pointeur d'instruction) pointe vers l'instruction à exécuter (suivante)
2. IR (ou RI) : Le registre d'instruction (Instruction Register) contient l'instruction en cours d'exécution
3. SP (ou PP) : Le pointeur de pile (Stack Pointer) pointe le sommet de la pile.
4. Accumulateur (Acc) : L'accumulateur stocke le résultat de l'UAL.
5. PSW (ou RE) : le Processor Status Word (Registre d'état) permettant de stocker des indicateurs sur l'état du système (retenue, dépassement, etc.).

**Exercice 8**

Préciser la fonction de la RAM dans un ordinateur. Quelles sont ses deux caractéristiques principales ?

**Solution :**

La mémoire vive, mémoire système ou mémoire volatile aussi appelée **RAM** de l'anglais Random Access Memory (mémoire à accès aléatoire) est la mémoire informatique dans laquelle un ordinateur place les données lors de leur traitement. Ses principales caractéristiques sont :

1) Sa rapidité d'accès (essentielle pour fournir rapidement les données au processeur)
2) Sa volatilité (cette volatilité implique que les données sont perdues dès que l'ordinateur cesse d'être alimenté en électricité).

**Exercice 9**

Quelles sont les principales différences entre la RAM et la ROM ? Où utilise-t-on la ROM ?

**Solution :**

La RAM est en lecture/écriture mais volatile alors la ROM est en lecture seule et non volatile.

On utilise en général la ROM pour les données du BIOS pour le démarrage de la machine.

**Exercice 10**

Pourquoi utilise-t-on des mémoires caches ? Pour qu'elle raison la mémoire cache est très couteuse ?

**Solution :**

Une mémoire cache en informatique est une mémoire qui enregistre temporairement des copies de données provenant de la RAM, afin de diminuer le temps d'accès (en lecture ou en écriture) d'un matériel informatique (en général, un processeur) à ces données. La

mémoire cache est plus rapide et plus proche du matériel informatique qui demande la donnée, mais plus petite que la mémoire pour laquelle elle sert d'intermédiaire.
La mémoire cache est très couteuse car elle est plus rapide et elle est miniaturisée)

## Exercice 11

Classer les mémoires suivantes par taille et par rapidité : RAM, registres, disques durs, cache L1, cache L2, CD-ROM. Que constatez-vous ?

## Solution :

**Classement par taille :** Registres < L1 < L2 < RAM < CD < DD.
**Classement par vitesse :** Registres > L1 > L2 > RAM > DD > CD.
**Remarque :** Plus la taille de la mémoire est petite plus elle rapide.

## Exercice 12

Soit une machine où tous les registres sont sur 32 bits

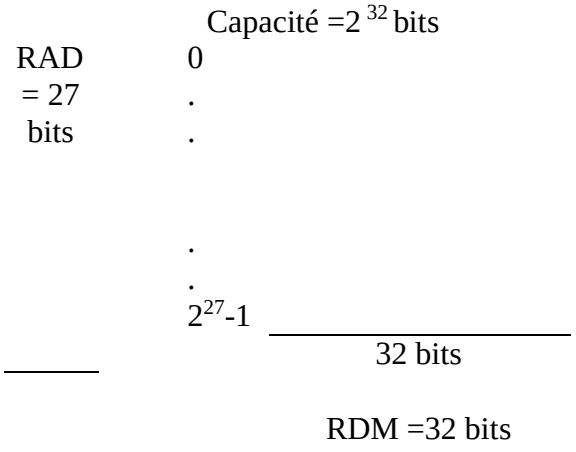
**1-** Calculer la capacité maximale de la mémoire
**2-** Donner le nombre de mot réel de cette mémoire si on suppose que sa capacité = $2^{32}$ bits
**3-** Proposer une architecture réduite pour cette mémoire (nombre de lignes d'adresses ; nombre de lignes de données ; taille du registre de mémoire de données ; taille du registre de mémoire d'adresse)

## Solution :

**1-** La capacité maximale de la mémoire
Capacité = $2^{\text{lignes d'@}}$ X nbre de lignes de données $2^{32}$ X 32 = **$2^{37}$ bits**

**2-** Le nombre de mot réel de cette mémoire si on suppose que sa capacité = $2^{32}$ bits
**Nbre de mots=capacité/taille du mot**
La taille du mot (en bits) =nbre de lignes de données. $\Rightarrow 2^{32}/ 32 =$ **$2^{27}$ mots**

**3-** Une architecture réduite pour cette mémoire (nombre de lignes d'adresses ; nombre de lignes de données ; taille du registre de mémoire de données ; taille du registre de mémoire d'adresse)
Le nbre de mots=$2^{\text{lignes d'@}}$ et la taille du mot (en bits) =nbre de lignes de données.
Nombre de lignes d'adresses =27 = taille RAM
Nombre de lignes de données =32 = taille RDM



## Exercice 13

**I-** une mémoire possède 13 lignes d'adresses et 8 lignes de données :

**1-** Combien de mots binaires peut–on emmagasiner dans cette mémoire et combien de bits possède ce mot ?

**2-** Quelle est la capacité totale de cette mémoire (en bits) ?

**II-** Lesquelles de ces organisations de mémoire sont plausibles ou envisageables ?

**a)** Registre d'adresses de 10 bits, 1024 cellules, 08 bits par cellule

**b)** Registre d'adresses de 10 bits, 1024 cellules, 12 bits par cellule

**c)** Registre d'adresses de 9 bits, 1024 cellules, 10 bits par cellule

**d)** Registre d'adresses de 11 bits, 1024 cellules, 10 bits par cellule

**e)** Registre d'adresses de 10 bits, 10 cellules, 1024 bits par cellule

## Solution :

**I-** une mémoire possède 13 lignes d'adresses et 8 lignes de données :

**1-** Combien de mots binaires peut–on emmagasiner dans cette mémoire et combien de bits possède ce mot ?

**Nbre mots=$2^{13}$, taille du mot =8 bits**

**2-** Quelle est la capacité totale de cette mémoire (en bits) ?

**Capacité =$2^{13}$x8=$2^{16}$bits**

**II-** Lesquelles de ces organisations de mémoire sont plausibles ou envisageables ?

Organisation envisageable si $2^{\text{taille du registre d'@}}$ >= nbre de cellules

**a)** Registre d'adresses de 10 bits, 1024 cellules, 08 bits par cellule ⇒ envisageable

**b)** Registre d'adresses de 10 bits, 1024 cellules, 12 bits par cellule ⇒ envisageable

**c)** Registre d'adresses de 9 bits, 1024 cellules, 10 bits par cellule ⇒ impossible

**d)** Registre d'adresses de 11 bits, 1024 cellules, 10 bits par cellule ⇒ envisageable

**e)** Registre d'adresses de 10 bits, 10 cellules, 1024 bits par cellule ⇒ envisageable

## Exercice 14

Soit une machine dotée d'une mémoire centrale de 1024 K mot de 32 bits.

**1-** Combine de bits, d'octets, de Kilo octets et de Méga octets contient cette mémoire

**2-** Combien de valeur différente peut prendre un mot de cette mémoire ?

**3-** Déterminer la plage d'adressage de cette mémoire (en hexadécimal/ base 16)

**4-** On veut stocker sur cette mémoire des nombres réels et chaque nombre est représenté sur 64 bits. Calculer l'adresse du 9ème nombre sachant que le premier est stocké à l'adresse $FF_{(16)}$

## Solution :

**1-** Combine de bits, d'octets, de Kilo octets et de Méga octets contient cette mémoire

**Capacité (bits)= Nombre de mots * taille du mot**

**Capacité (bits)=** 1024 K * 32=$2^{10}$ * $2^{10}$ * $2^5$ = **$2^{25}$bits**

**Capacité (octets)**=$2^{25}$/ $2^3$ = **$2^{22}$octets**

**Capacité (Koctets)**= $2^{22}$/ $2^{10}$ = **$2^{12}$Koctets**

**Capacité(Moctets)**= $2^{12}$/ $2^{10}$ = **$2^2$Mocts**

**2-** Combien de valeur différente peut prendre un mot de cette mémoire ?

**$2^{32}$ valeurs**

**3-** La plage d'adressage de cette mémoire (en hexadécimal/ base 16)

**Nombre de mots = $2^{\text{nombre de lignes d'adresses}}$**

Nombre de mots = 1024 K = $2^{10}*2^1 = \mathbf{2^{20}}$

⇒ **Nombre de lignes d'adresse =20**

**Adresse minimale** ( 0000000000.. ) sur 20 bits =**$00000_{(16)}$**

et **adresse maximale** **($2^{20}$ -1 )** = 1111111 ….1111 sur 20 bits= **$FFFFF_{(16)}$**

**4-** On veut stocker sur cette mémoire des nombres réels et chaque nombre est représenté sur 64 bits. Calculer l'adresse du $9^{ème}$ nombre sachant que le premier est stocké à l'adresse $FF_{(16)}$

Chaque nombre dans la mémoire prend 2 mots mémoire (taille du nombre réel / taille du mot = 64/32 =2 mots)

**@$n^{ème}$nombre= @$1^{er}$nombre + (nombre de mots de chaque nombre) * (n-1)**

**@$1^{er}$nombre**= $FF_{(16)} = (15*16 + 15)_{(10)} =$ **$255_{(10)}$**

**@$9^{ème}$nombre**=@$1^{er}$nombre + (8 x2) =$255_{(10)} + 16_{(10)} = 271_{(10)}$ =**$10F_{(16)}$**

## Exercice 15

Soit une mémoire cache de niveau L1 ayant les caractéristiques suivantes :

- 32 mots par lignes (mots de 2 octets) - Taille de 32ko - L1 et L2 sont inclusifs - 4-associatifs - Remplacement LRU - Association par poids faible - Taille de bus d'adresse : 32bits

**1-** Combien y-a-t-il de lignes dans cette mémoire cache ?

2- Combien y-a-t-il de blocs associatifs dans cette mémoire cache ?

3- Si la mémoire cache de niveau L2 a une taille de 2 Mo, combien y-a-t-il de blocs de la mémoire cache L2 par bloc de la mémoire cache L1 ?

4- Si la mémoire fait 1Go, combien d'adresses correspondront à un bloc du cache L1 ?

5- Si un bloc n'est pas présent en cache L1, combien de lignes de L1 aura-t-on parcouru ?

## Solution :

Soit une mémoire cache de niveau L1 ayant les caractéristiques suivantes :

- 32 mots par lignes (mots de 2 octets) - Taille de 32ko - L1 et L2 sont inclusifs - 4-associatifs - Remplacement LRU - Association par poids faible - Taille de bus d'adresse : 32bits

**2-** Combien y-a-t-il de lignes dans cette mémoire cache ?

**Nombre de lignes = Taille cache /(Taille mot* nombre de mots par ligne)**

= 32 KOct / (2 Oct * 32) = $2^{10}/2 = 2^9$= **512 lignes**

6- Combien y-a-t-il de blocs associatifs dans cette mémoire cache ?

**Nombre de blocs = Nombre de lignes/ Nombre de lignes par bloc**

**=** 512/4 = $2^9/2^2 = 2^7$ = **128 blocs**

7- Si la mémoire cache de niveau L2 a une taille de 2 Mo, combien y-a-t-il de blocs de la mémoire cache L2 par bloc de la mémoire cache L1 ?

**Taille cache L2 / Taille cache L1 =** 2 MOct/32 KOct = $2^{21}/2^{15} = 2^6$ = **64**

8- Si la mémoire fait 1Go, combien d'adresses correspondront à un bloc du cache L1 ?

**Nbre d'adresse à un bloc =**

**Taille mémoire / (Nbre blocs * Taille mot * Nbre de lignes d'adresse)**

= 1 GOct / (128*2Oct* 32) = $2^{30}/2^{13} = 2^{17}$ =**131072**

Sachant que le nombre de lignes d'adresse = taille du bus d'adresse = 32

9- Si un bloc n'est pas présent en cache L1, combien de lignes de L1 aura-t-on parcouru ?

**4 lignes**

### Exercice 16

**A)** Supposons un programme composé d'une boucle de 10 instructions et que la moitié des instructions se trouvent en mémoire cache et l'autre moitié en mémoire centrale. Si le temps d'accès au cache est de 5 ns et celui de la mémoire centrale est de 20 ns, calculer le temps global d'exécution du programme (sans prise en compte du temps d'exécution des instructions par le processeur).
**B)** Si le temps d'accès au cache est de 5 ns, le temps de pénalité au cache est de 10 ns, et que le temps d'échec du cache est de 20 ns. Calculez le temps de récupération d'une instruction

### Solution :

**A)** Puisque la mémoire cache contient la moitié des instructions du programme, donc nous avons 5 instructions qui se trouvent en cache.
Chaque instruction dans le cache prend 5 ns pour la récupérer.
Au total pour le temps d'accès aux instructions qui se trouvent en mémoire cache, nous avons 5 × 5 = **25 ns**
Nous ajoutons à ce temps, le **temps d'accès aux instructions qui se trouvent en mémoire centrale** et pour lesquelles le **temps d'accès** est de **20 ns.**
Donc, nous avons **5 × 20 = 100 ns**
Le **temps global d'exécution** est de **25 + 100 = 125 ns**
**B)** Le temps de récupération d'une instruction à partir du cache peut être calculé selon la formule de récupération d'une instruction de la façon suivante :

**Temps de récupération = T échec × T pénalité+ (1 − T échec) × T accès**
**Temps de récupération** =20 ×10 + (1 − 20) × 5 = 200 + (-95) **=105 ns**

## Chapitre 3 : Notions sur les instructions d'un ordinateur

### Exercice 1

Donner la trace d'exécution du programme suivant, sachant que : [acc]=50 ; [30] =39 ; [31] =12 ;

```
10 ADD 30
11 DIV 31
12 STA 32
13 Branch si S=0/-4
```

### Solution

Trace d'exécution du programme donné, sachant que : [acc]=50; [30]=39; [31]=12;
10 ADD 30 => [Acc] <- [Acc]+[30]=50+39=89
11 DIV 31 => [Acc] <- [Acc]/[31]=89/12=7.41
12 STA 32 => [32] <- [Acc]=7.41
13 Branch si S=0/-4 => (Branchement vers l'instruction 09 (13-4+1) si le contenu de l'accumulateur est POSITIF)
Dans ce cas Co← 10

Remarque : Le programme boucle à l'infini.

## Exercice 2

**1-** Donner l'expression de X effectuée par le programme suivant en mode immédiat, sachant que

1. LOAD A
2. ADD B
3. MPY C
4. STORE 100
5. LOAD B
6. DIV A
7. SUB C
8. ADD 100
9. STA X

## Solution :

Donner l'expression de X effectuée par le programme suivant en mode **immédiat**.

1. LOAD A => [Acc] <- A
2. ADD B => [Acc] <- [Acc]+B=A+B
3. MPY C => [Acc] <- [Acc]*C=(A+B)*C
4. STORE 100 => [100] <- [Acc] =(A+B)*C
5. LOAD B => [Acc] <- B
6. DIV A => [Acc] <- [Acc]/[A]=B/A
7. SUB C => [Acc] <- [Acc]-[C]=B/A-C
8. ADD 100 => [Acc] <- [Acc]+[100]=(B/A-C)+(A+B)*C
9. STA X => X<- [Acc] =(B/A-C)+(A+B)*C

Donc **X=(B/A-C)+(A+B)*C**

## Exercice 3

Ecrire un programme qui calcule l'expression Z dans une machine possédant **un opérande (machine à une (1) adresses)** et dont le mode d'adressage est **direct**.

Z= (A+B)*(C+D)/(E+F) sachant que : [10]=A, [20]=B, [30]=C, [40]=D, [50]=E, [60]=F, [100]=Z.

## Solution :

**1-** Le programme qui calcule l'expression Z dans une machine possèdent **un opérande (machine à une (1) adresses)** et dont le mode d'adressage est **direct**.

Z= (A+B)*(C+D)/(E+F) sachant que : [10]=A, [20]=B, [30]=C, [40]=D, [50]=E, [60]=F, [100]=Z.

**Machine a 1@ ⇒ Utilisation de l'accumulateur**

```
LOAD 50;     => [Acc] <- [50]=E
ADD 60;      => [Acc] <- [Acc]+[60]=E+F
STORE 200    => [200] <- [Acc] =E+F
LOAD 10;     => [Acc] <- [10]=A
ADD 20;      => [Acc] <- [Acc]+[20]=A+B
STORE 300    => [300] <- [Acc] =A+B
LOAD 30;     => [Acc] <- [30]=C
ADD 40;      => [Acc] <- [Acc]+[40]=C+D
```

MPY 300 => [Acc] <- [Acc] *[300]
= (C+D) * (A+B)
DIV 200 => [Acc] <- [Acc] /[200]
= (C+D) * (A+B) / (E+F)
STA 100 => [100] <- [Acc]
= (C+D) * (A+B) / (E+F)

## Exercice 4

En supposant que la machine est **à pile (machine à zéro (0) adresses)** et le mode d'adressage est **immédiat**, donner le code pour évaluer la même expression de l'exercice 3 et ce en utilisant le minimum d'instruction possible.

## Solution :

En supposant que la machine est **à zéro (0) adresses** et le mode d'adressage est **immédiat**, donner le code pour évaluer la même expression et ce en utilisant le minimum d'instruction possible.

**Machine a 0@ ⇒ Utilisation de la pile**

| | |
|---|---|
| PUSH A; | Pile ={A} |
| PUSH B; | Pile ={A; B} |
| ADD ; | Pile ={A+ B} |
| PUSH C; | Pile ={(A+ B);C} |
| PUSH D; | Pile ={(A+ B);C;D} |
| ADD; | Pile ={(A+ B);(C+D)} |
| MPY; | Pile ={(A+ B)*(C+D)} |
| PUSH E; | Pile ={(A+ B)*(C+D);E} |
| PUSH F; | Pile ={(A+ B)*(C+D);E;F} |
| ADD | Pile ={(A+ B)*(C+D);E+F} |
| DIV | Pile ={(A+ B)*(C+D)/(E+F)} |
| STORE Z; | Pile ={ } |

Soit que

## Exercice 5

l'instruction d'affectation **X= (a+b)*(c+d)** telles a, b, c et d sont des variables préalablement définies et stockées respectivement dans les adresses A, B, C et D.

Le langage d'assemblage dispose des instructions suivantes :

- add pour l'addition
- mul pour la multiplication
- mov pour le transfert de registre à registre ou de registre à mémoire

- load pour le chargement
- store pour le stockage
- push pour l'empilement
- pop pour le dépilement

On vous demande d'écrire les programmes correspondant aux différents cas suivants :

1) Instructions à 3 adresses
2) Instructions à 2 adresses
3) Instructions à 1 adresse
4) Instructions à 0 adresse (pile)

**Solution**

| Instructions à 3 adresses | Instructions à 2 adresses | Instructions à 1 adresse | Instruction à 0 adresse (Pile) |
|---|---|---|---|
| add R1, A, B<br>add R2, C, D<br>mul X, R1, R2 | mov R1, A<br>add R1, B<br>mov R2, C<br>add R2, D<br>mul R1, R2<br>mov X, R1 | load A<br>add B<br>store T<br>load C<br>add D<br>mul T<br>store X<br><br>(T : temporaire) | push A<br>push B<br>add<br>push C<br>push D<br>add<br>mul<br>pop X |

**Exercice 6**

Donner les résultats d'exécution du fragment de programme suivant pour les 03 modes d'adressage suivants : Immédiat – Direct – Indirect, sachant que :
[acc]=100 ; [20] =60 ; [60] =5 ; [5] =20.

10 ADD 20
11 SUB 60
12 MPY 5
13 DIV 20

**Solution :**

Les résultats d'exécution du fragment de programme donné pour les 03 modes d'adressage suivants : Immédiat – Direct – Indirect, sachant que :
[acc]=100 ; [20] =60 ; [60] =5 ; [5] =20 sont :

1) **Immédiat**

10 ADD 20 => [Acc] ← [Acc]+20=100+20=120
11 SUB 60 => [Acc] ← [Acc]-60=120-60=60
12 MPY 5 => [Acc] ← [Acc]*5=60*5=300
13 DIV 20 => [Acc] ← [Acc]/ 20 = 300/20=15

2) **Direct**

10 ADD 20 => [Acc] ← [Acc]+ [20] =100+60=160
11 SUB 60 => [Acc] ← [Acc]- [60] =160-5=155
12 MPY 5 => [Acc] ← [Acc]*[5] =155*20=3100
13 DIV 20 => [Acc] ← [Acc]/ [20] = 3100/60=51.67

3) **Indirect**

10 ADD 20 => [Acc] ← [Acc]+ [[20]] =100+ [60] =100+5=105
11 SUB 60 => [Acc] ← [Acc]- [[60]] =105- [5] =105-20=85
12 MPY 5 => [Acc] ← [Acc]*[[5]] =85*[20] =85*60=5100

13 DIV 20 => [Acc] ← [Acc]/ [[20]] = 5100/ [60] =5100/5=1020

**Exercice 7**

Décrire les différentes étapes d'exécution des instructions suivantes:

10 ADD 25 ;
17 SUB 40 ;
18 STA 25 ;

**Solution :**

Décrire les différentes étapes d'exécution des instructions :

**10 ADD 25 ;**

**Cas : une instruction arithmétique à un seul opérande (mode direct)**

**Phase 1 : *Recherche de l'instruction à traiter***

**1.1** Mettre le contenu du CO dans le registre d'adresse mémoire (R@M).

(CO)→ R@M avec (CO)=10

1.2 Commande de lecture à partir de la mémoire

1.3 La cellule n° 10 est sélectionnée et son contenu est transféré vers le Registre de Données Mémoire (RDM). (10)→ RDM

1.4 Transfert du contenu du RDM dans le registre instruction (RI).

(RDM) → RI avec (RDM)=ADD 25

**Phase 2 : *Décodage de l'instruction et recherche de l'opérande***

2.1 Analyse et décodage du code opération.

2.2 Transfert de l'ADresse de l 'OPérande (ADOP) dans le R@M.

(ADOP) → R@M avec (ADOP)=25

2.3 Commande de lecture.

2.4 La cellule n° 25 est sélectionnée et son contenu est transféré vers le registre de données mémoire (RDM) (25) → RDM

2.5 Transfert du contenu du RDM vers l'UAL. (RDM) → UAL

**Phase 3 : *Exécution de l'instruction et passer à l'instruction suivante***

3.1 Commande de l'exécution de l'opération (Addition). [Acc] ← [Acc]+ [25]

3.2 Les drapeaux sont positionnés (registre d'état).

3.3 L'unité de commande positionne le CO pour l'instruction suivante.

(CO) +1 → CO avec (CO) = 10

**17 SUB 40 ;**

**Cas : une instruction arithmétique à un seul opérande (mode direct)**

**Phase 1 : *Recherche de l'instruction à traiter***

**1.1** Mettre le contenu du CO dans le registre d'adresse mémoire (R@M).

(CO)→ R@M avec (CO)=17

1.2 Commande de lecture à partir de la mémoire

1.3 La cellule n° 17 est sélectionnée et son contenu est transféré vers le Registre de Données Mémoire (RDM). (17)→ RDM

1.4 Transfert du contenu du RDM dans le registre instruction (RI).

(RDM) → RI avec (RDM)=SUB 40

**Phase 2 : *Décodage de l'instruction et recherche de l'opérande***

2.1 Analyse et décodage du code opération.

2.2 Transfert de l'Adresse de l'Opérande (ADOP) dans le R@M.

(ADOP) → R@M avec (ADOP)=40

2.3 Commande de lecture.

2.4 La cellule n° 40 est sélectionnée et son contenu est transféré vers le registre de données mémoire (RDM) (40) → RDM
2.5 Transfert du contenu du RDM vers l'UAL. (RDM) → UAL

**Phase 3 :** ***Exécution de l'instruction et passer à l'instruction suivante***

3.1 Commande de l'exécution de l'opération (Addition). [Acc] ← [Acc]-[40]
3.2 Les drapeaux sont positionnés (registre d'état).
3.3 L'unité de commande positionne le CO pour l'instruction suivante.

(CO) +1 → CO avec (CO) = 17

**18 STA 25 ;**

**Cas : une instruction arithmétique à un seul opérande (mode direct)**

**Phase 1 :** ***Recherche de l'instruction à traiter***

**1.1** Mettre le contenu du CO dans le registre d'adresse mémoire (R@M).

(CO)→ R@M avec (CO)=18

1.2 Commande de lecture à partir de la mémoire
1.3 La cellule n° 18 est sélectionnée et son contenu est transféré vers le Registre de Données Mémoire (RDM). (18)→ RDM
1.4 Transfert du contenu du RDM dans le registre instruction (RI).

(RDM) → RI avec (RDM)=STA 25

**Phase 2 :** ***Décodage de l'instruction et recherche de l'opérande***

2.1 Analyse et décodage du code opération.
2.2 Transfert du contenu de l'accumulateur dans le RDM.

(ACC) → RDM

2.3 Commande d'écriture.
2.4 La cellule n° 25 est sélectionnée, elle est prête à recevoir les données
2.5 Transfert du contenu du RDM vers l'adresse 25. (RDM) →25@

**Phase 3 :** ***Exécution de l'instruction et passer à l'instruction suivante***

3.1 L'unité de commande positionne le CO pour l'instruction suivante.

(CO) +1 → CO avec (CO) = 10

## Exercice 8

Expliquer le principe du pipeline ? Qu'est-ce qu'il nécessite ? Quelles sont ses limites ? Est quel gain en performance existe-t-il ?

## Solution :

Le pipeline est un mécanisme permettant d'accroître la vitesse d'exécution des instructions dans un micro-processeur. L'idée générale est d'appliquer le principe du *travail à la chaîne* à l'exécution des instructions. Dans un micro-processeur sans pipeline, les instructions sont exécutées les unes après les autres. Une nouvelle instruction n'est commencée que lorsque l'instruction précédente est complètement terminée. Avec un pipeline, le micro-processeur commence une nouvelle instruction avant d'avoir fini la précédente. Plusieurs instructions se trouvent donc simultanément en cours d'exécution au cœur du micro-processeur. Le temps d'exécution d'une seule instruction n'est pas réduit. Par contre, le débit du micro-processeur, c'est-à-dire le nombre d'instructions exécutées par unité de temps, est augmenté. Il est multiplié par le nombre d'instructions qui sont exécutées simultanément.

- pour exécuter **n** instructions, en supposant que chaque instruction s'exécute en **k** cycles d'horloge, il faut :
  - **n*k cycles d'horloge** pour une **exécution séquentielle**.

- **k cycles d'horloge** pour exécuter la **première instruction** puis **n-1 cycles** pour les **n-1 instructions** suivantes si on utilise un pipeline de **k étages**

- Le **gain** obtenu est donc de : **G= n*k/k+n-1**

## Chapitre 4 : Le processeur

### Exercice 1

*Si la machine A exécute un programme en 10 secondes et la machine B exécute le même* programme en 15 secondes, de combien A est-elle plus rapide que B ?

### Solution :

Nous savons que A est n fois plus rapide que B si Performances-A / Performances-B = n ou Temps d'exécution-B / temps d'exécution-A = n
Le rapport est donc de **15/10=1,5**

### Exercice 2

Notre programme s'exécute en 10 secondes sur A, qui dispose d'une horloge à 100Mhz. Nous tentons d'aider un concepteur à construire une machine B, qui exécutera ce programme en 6 secondes. Le concepteur établi qu'une augmentation substantielle de la fréquence d'horloge est possible, mais que cette augmentation affectera le reste de la conception de l'UC, imposant à la machine B d'utiliser 1,2 fois plus de cycles d'horloge que la machine A pour ce programme. Quel objectif de la fréquence d'horloge devrons-nous donner au concepteur ?

### Solution :

Déterminons tout d'abord le nombre de cycles d'horloge nécessaires au programme sur A:
Tps UC-A = NB de cycles UC-A / Fréquence d'H-A
10 = NB cycles UC-A / $100*10^6$Cycles/secondes
Nb cycles UC-A = $10*100*10^6 = 1000*10^6$ cycles
Le temps UC pour B peut être obtenu en utilisant l'équation suivante :
Tps UC-B = $1,2*1000*10^6$cycles / Fréquence d'H-B
Fréquence d'H-B = $1,2*1000*10^6$ cycles / 6 secondes = 200Mhz

**Rappel :** NB de cycles UC = NI * CPI. Avec CPI, le nombre de cycles d'horloge par instruction, correspond au nombre moyen de cycles d'horloge qu'il faut à chaque instruction pour s'exécuter.

### Exercice 3

L'architecture des microprocesseurs se compose de deux grandes familles :
1. L'architecture CISC (Complex Instruction Set Computer)
2. L'architecture RISC (Reduced Instruction Set Computer)

Citer quatre (4) différences entre l'architecture CISC et l'architecture RISC.

### Solution :

| Architecture RISC | Architecture CISC |
|---|---|

| | |
|---|---|
| Instructions simples ne prenant qu'un seul cycle | Instructions complexes prenant plusieurs cycles |
| Instructions au format fixe | Instructions au format variable |
| Décodeur simple (câblé) | Décodeur complexe (microcode) |
| Beaucoup de registres | Peu de registres |
| Seules les instructions LOAD et STORE ont accès à la mémoire | Toutes les instructions sont susceptibles d'accéder à la mémoire |
| Peu de modes d'adressage | Beaucoup de modes d'adressage |
| Compilateur complexe | Compilateur simple |

**Tableau comparatif entre l'architecture RISC et l'architecture CISC**

## Exercice 4

Les instructions exécutées sur un processeur peuvent être classées en trois catégories selon le nombre de cycles d'horloge (CCi) nécessaires pour leur exécution.

| Catégorie | Nombre de cycles d'horloge CCi |
|---|---|
| A | 1 |
| B | 2 |
| C | 3 |

Deux compilateurs différents génèrent, pour un même programme, les mélanges d'instructions suivantes Ce nombre d'instructions pour chaque catégorie est indiqué dans le tableau suivant :

| Compilateur | Catégorie A | Catégorie B | Catégorie C |
|---|---|---|---|
| IC(X) | 5.000.000.000 | 1.000.000.000 | 1.000.000.000 |
| IC(Y) | 10.000.000.000 | 1.000.000.000 | 1.000.000.000 |

**1-** En supposant que la fréquence d'horloge du processeur est de 500 MHz, donner la valeur du CPI pour chaque cas.
**2-** Lequel des deux mélanges va s'exécuter le plus rapidement?
***3-*** Quelle est la performance en MIPS (Millions d'Instructions Par Seconde) dans les deux cas?

## Solution :

**1.** Calcul du CPI :

$$\boldsymbol{CPI} = \frac{\sum_i (\boldsymbol{ICi})(\boldsymbol{CCi})}{\boldsymbol{IC}}$$

$$\boldsymbol{CPI(X)} = \frac{\mathbf{5x10^9x1+1x10^9x2+1x10^9x3}}{\mathbf{5x10^9+1x10^9+1x10^9}} = \frac{\mathbf{5+2+3}}{\mathbf{7}} = \frac{\mathbf{10}}{\mathbf{7}} = \mathbf{1,428}$$

$$\boldsymbol{CPI(Y)} = \frac{\mathbf{10x10^9x1+1x10^9x2+1x10^9x3}}{\mathbf{10x10^9+1x10^9+1x10^9}} = \frac{\mathbf{10+2+3}}{\mathbf{12}} = \frac{\mathbf{15}}{\mathbf{12}} = \mathbf{1,25}$$

**2.** Pour donner le mélange qui va s'exécuter plus rapidement, nous devons tout d'abord calculer le temps du cycle horloge pour chaque compilateur.

F= 500Mhz : 500 * $10^6$ op $\rightarrow$ 1 seconde

1 op → T cycle
⇒ T cycle= 1/ 500 * $10^6$ = 2 * $10^{-9}$ secondes $T_X$=
($5*1*2*10^{-9}$) + ($1*2*2*10^{-9}$) + ($1*3*2*10^{-9}$) = 20. $10^{-9}$ secondes
Ty= ($10*1*2*10^{-9}$) + ($1*2*2*10^{-9}$) + ($1*3*2*10^{-9}$) = 30. $10^{-9}$ secondes
C'est le mélange généré par le compilateur X qui va être exécuté plus rapidement.

**3. La performance en MIPS (Millions d'Instructions Par Seconde) dans les deux cas.**
***Cas du compilateur X :*** 7.000.000.000 instructions → 20* $10^{-9}$ secondes
$Per_X$ → 1 seconde
⇒ $Perf_X$= $7*10^3/20*10^{-9}$= 350 MIPS.
***Cas du compilateur Y :*** 12.000.000.000 instructions → 30* $10^{-9}$ secondes
$Per_Y$ → 1 seconde
⇒ $Perf_Y$= $12*10^3/30*10^{-9}$= 400 MIPS

## Exercice 5

Choisir la (ou les) bonne (bonnes) réponse (s) :
1) **Un « appel système » est**
□ Une instruction privilégiée exécutable en mode système pour accéder à certains registres.
□ Une fonction exécutée en mode utilisateur qui provoque l'exécution de code système.
□ Une fonction qui s'exécute en mode système.
□ Un type particulier d'interruption géré par le GIET.
2) **Laquelle de ces actions ne nécessite pas que le processeur soit en mode superviseur**
□ exécuter l'instruction assembleur syscall
□ exécuter l'instruction assembleur reset
□ masquer les interruptions
□ accéder à l'adresse 0x80000000

## Solution :

Choisir la (ou les) bonne (bonnes) réponse (s) :
1) **Un « appel système » est**
□ Une instruction privilégiée exécutable en mode système pour accéder à certains registres.
=> □ Une fonction exécutée en mode utilisateur qui provoque l'exécution de code système.
□ Une fonction qui s'exécute en mode système.
□ Un type particulier d'interruption géré par le GIET.
2) **Laquelle de ces actions ne nécessite pas que le processeur soit en mode superviseur**
=> □ exécuter l'instruction assembleur syscall
□ exécuter l'instruction assembleur reset
□ masquer les interruptions
□ accéder à l'adresse 0x80000000

## Exercice 6

Soit l'instruction conditionnelle suivante :

***if t1 < t2 then t3 := t1 else t3 := t2***

Donner le code MIPS équivalent.

## Solution :

```
blt $t1, $t2, Then  # si t1 < t2 saut à Then
move $t3, $t2       # t3 := t2
j End               # saut à End
Then: move $t3, $t1 # t3 := t1
End:                # suite du programme
Li $v0,10
syscall
```

**Exercice 7 :**

1 Quels sont les registres qui peuvent contenir les paramètres d'entrée d'une fonction lors d'un passage de paramètres d'entrée ?

2 Où est ce que je peux mettre mes paramètres dans le cas où le nombre de ces registres n'est pas suffisant pour tous les paramètres ?

**Solution :**

1 Dans le cas d'un passage de paramètres d'entrée pour une fonction, les registres qui peuvent contenir mes paramètres sont : $a0, $a1, $a2, $a3,

2 Dans le cas où le nombre de ces registres n'est pas suffisant pour tous les paramètres, on peut mettre les paramètres restants dans une pile

## Chapitre 5 : Instructions spéciales (MIPS R3000)

**Exercice1**

Ecrire le code MIPS qui permet de :

1. calculer la somme 5+7 ;
2. afficher le résultat.

**Solution :**

```
1   # Solution Exercice 1
2   # Il n'y aura pas de données à déclarer donc pas de section data, on utilise le mode immédiat
3   .text
4   main:
5   li $t0,5          # charger la valeur 5 dans le registre t0
6   li $t1,7          # charger la valeur 7 dans le registre t1
7   add $t2,$t1,$t0 # Faite l'addition de t0+t1 et mettre le résultat dans le registre t2
8   move $a0,$t2      # le contenu de t2 est copié dans le registre a0
9   li $v0,1          # charger le registre v0 avec le code 1 pour afficher entier print_int
10  syscall           # appel système pour l'opérartion affichage entier
11  #Fin du programme
12  li $v0,10
13  syscall
```

**Exercice 2 :**

Ecrire le code MIPS qui permet de :

1. réserver un espace mémoire pour les entiers A=15, B=35 ;
2. déclarer la chaine de caractères Str=« La somme de A=15 et B=35 est : »
3. calculer la somme A+B ;
4. afficher la chaine de caractère Str sur la console ;
5. afficher le résultat.

## Solution :

```
# Solution Exercice 2
.data           # Section données
A: .word 15     # Déclaration de variable A=15
B: .word 35     # Déclaration de variable B=35
Str: .asciiz "La somme de A=15 et B=35 est: " # Déclaration de chaine de caractère Str
.text
main:           # Section Code
lw $t0,A        # charger la valeur de la variable A dans le registre t0
lw $t1,B        # charger la valeur de la variable B dans le registre t1
add $t2,$t1,$t0 # Faire l'addition de t0+t1 et mettre le résultat dans le registre t2
li $v0,4        # Charger le registre $v0 avec le code 4 pour affichage caractère (print_string)
la $a0,Str      # Mettre l'adresse de la chaine de caractère Str dans le registre $a0
syscall         # Appel système pour l'opérartion affichage chaine de caractère
move $a0,$t2    # Le contenu de t2 est copié dans le registre a0
li $v0,1        # Charger le registre v0 avec le code 1 pour afficher entier (print_int)
syscall         # Appel système pour l'opérartion affichage entier
#Fin du programme
li $v0,10
syscall
```

## Exercice 3 :

Ecrire le code MIPS qui permet de :

1. réserver un espace mémoire pour les réels double A=15.4, B=35.2 ;
2. déclarer une chaine de caractères Str=« La somme de A=15.4 et B=35.2 est : »
3. calculer la somme A+B ;
4. afficher la valeur de la chaine de caractères Str, ensuite le résultat.

## Solution :

```
# Solution Exercice 3
.data                # Section données
A: .double 15.4      # Déclaration de variable A=15.4
B: .double 35.2      # Déclaration de variable B=35.2
Str: .asciiz "La somme de A=15.4 et B=35.2 est: " # Déclaration de chaine de caractère Str
.text
main:                # Section Code
l.d $f0,A            # charger la valeur de la variable A dans le registre t0
l.d $f2,B            # charger la valeur de la variable B dans le registre t1
add.d $f12,$f2,$f0   # Faire l'addition de f0+f2 et mettre le résultat dans le registre f12
li $v0,4             # Charger le registre $v0 avec le code 4 pour affichage caractère (print_string)
la $a0,Str           # Mettre l'adresse de la chaine de caractère Str dans le registre $a0
syscall              # Appel système pour l'opérartion affichage chaine de caractère
li $v0,3             # Charger le registre v0 avec le code 3 pour afficher double (print_double)
syscall              # Appel système pour l'opérartion affichage double
#Fin du programme
li $v0,10
syscall
```

## Exercice 4 :

Ecrire le code MIPS qui permet de :
1. déclarer les chaines de caractères Message1=« Donner un nombre entier : », Message2=«L'entier que vous avez donné est : »
2. afficher la chaine de caractère Message1 sur la console ;
3. lire un nombre entier au clavier ;
4. afficher la chaine de caractère Message2 sur la console suivie de la valeur de l'entier saisi au clavier

## Solution :

```
# Solution Exercice 4
.data           # Section données
Message1: .asciiz "Donner un nombre entier :  " # Déclaration de chaine de caractère Message1
Message2: .asciiz "L'entier que vous avez donné est :  " # Déclaration de chaine de caractère Message2
.text
main:           # Section Code
li $v0,4        # Charger le registre $v0 avec le code 4 pour affichage caractère (print_string)
la $a0,Message1 # Mettre l'adresse de la chaine de caractère Message1 dans le registre $a0
syscall         # Appel système pour l'opérartion affichage chaine de caractère
li $v0,5        # Charger le registre $v0 avec le code 5 pour lire entier (read_int)
syscall         # Appel système pour lecture
move $t0,$v0    # Mettre la valeur de l'entier lu ($v0) dans le registre $t0
li $v0,4        # Charger le registre $v0 avec le code 4 pour affichage caractère (print_string)
la $a0,Message2 # Mettre l'adresse de la chaine de caractère Message2 dans le registre $a0
syscall         # Appel système pour l'opérartion affichage chaine de caractère
move $a0,$t0    # Mettre la valeur du registre $t0 dans le registre $a0
li $v0,1        # Charger le registre $v0 avec le code 1 pour afficher entier (print_int)
syscall         # Appel système pour l'opérartion affichage entier
#Fin du programme
li $v0,10
syscall
```

## Exercice 5 :

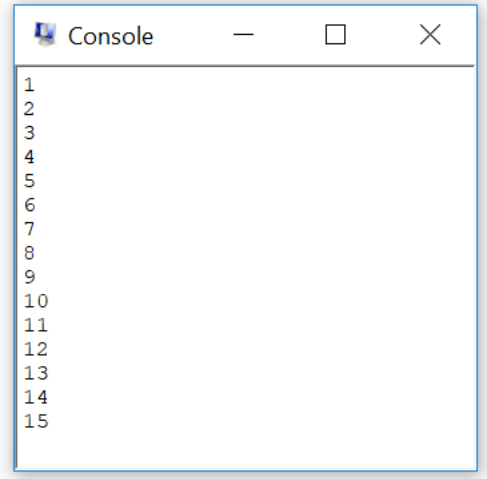
Ecrire le code MIPS qui permet de :
1. déclarer les chaines de caractères Chaine=« Donner un nombre flottant : », Sortie=« La valeur saisie est : »
2. afficher la chaine de caractère Chaine sur la console ;
3. lire un nombre flottant au clavier ;
4. afficher la chaine de caractère Sortie sur la console suivie de la valeur du nombre flottant saisi au clavier.

**Solution :**

```
# Solution exercice5
.data           # Section données
chaine:.asciiz"donner un nombre flottant:"
sortie:.asciiz"la valeur saisie est :"
.text
main:           # Section Code
la $a0,chaine   # Mettre l'adresse de la chaine de caractère chaine dans le registre $a0
li $v0,4        # Charger le registre $v0 avec le code 4 pour affichage chaine de caractère (print_string)
syscall         # Appel système pour l'opérartion affichage chaine de caractère
li $v0,6        # Charger le registre $v0 avec le code 6 pour lire flottant (read_float)
mov.s $f0,$f12  # Mettre la valeur de du flottant lu ($f12) dans le registre $f0
syscall         # Appel système
la $a0,sortie   # Mettre l'adresse de la chaine de caractère sortie dans le registre $a0
li $v0,4        # Charger le registre $v0 avec le code 4 pour affichage chaine de caractère (print_string)
syscall         # Appel système
mov.s $f12,$f0  # Mettre le contenu du registre $f0 dans le registre $f12
li $v0,2        # Charger le registre $v0 avec le code 2 pour afficher flottant (print_float)
syscall         # Appel système pour affichage
#Fin du programme
li $v0,10
syscall
```

## Exercice 6

Ecrire le programme MIPS permettant d'afficher les valeurs entières de 0 à 15 comme indiqué ci-dessous sur la console.

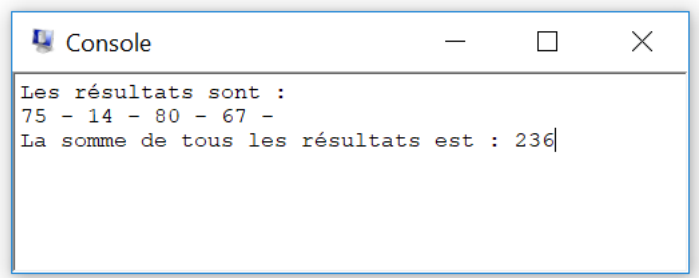



**Solution :**

```
.data
Newln : .asciiz "\n"
.text
main :
li $t0,1
li $t1,15
loop : move $a0,$t0
li $v0,1
syscall
li $v0,4
la $a0,Newln
syscall
addi $t0,$t0,1
ble $t0,$t1,loop #if ($t0<=$t1 aller à loop)
li $v0,10
syscall
```

## Exercice 7

En utilisant l'appel fonction *affiche*, faire un programme MIPS qui calcule et affiche les résultats de : **x+y+z - x+4 - x*8 - x+y+(z/2)** Tels que x, y, z sont des variables déclarées sur **8 bits** et leurs valeurs sont respectivement : **10, 50, 15.**
Pour l'affichage il faut avoir : (Voir l'image de la console ci-dessous)
Message1- **Les résultats sont :** *retour à la ligne*
Résultat des calculs : **résultat1 - résultat2 - résultat 3 - résultat 4 -**
Message 2 - **La somme de tous les résultats est :** ***Résultat de la somme***




## Solution :

```
1   .data
2   x: .byte 10
3   y: .byte 50
4   z: .byte 15
5   message1: .asciiz"Les résultats sont : \n"
6   separateur:.asciiz " - "
7   retour:.asciiz "\n"
8   message2 : .asciiz"La somme de tous les résultats est : "
9   .text
10  main:
11  li$v0,4
12  la $a0,message1
13  syscall
14  lb $t1,x
15  lb $t2,y
16  lb $t3,z
17  add $a0,$t1,$t2
18  add $a0,$a0,$t3
19  move $t0,$a0
20  jal affiche
21  addi $a0,$t1,4
22  move $t5,$a0
23  jal affiche
24  li $t4,8
25  mul $a0,$t1,$t4
26  move $t6,$a0
27  jal affiche
```

# Références bibliographiques :


- Alain Cazes, Joëlle Delacroix, Architecture des machines et des systèmes informatiques 4ème édition, Collection : Informatique, Dunod, 2011.
- Andrew S. Tanenbaum, Todd Austin Structured Computer Organization, Pearson, 2012.
- David Simplot Architecture des Ordinateurs, Licence Informatique – USTL, http://www.lifl.fr/~simplot/ens/archi
- L. Ghalouci, Architecture de l'Ordinateur, Voyage au centre de votre unité centrale, Support de cours, USTO-MB, Département de physique, 2015.
- Laurent Poinsot, Cours "Architecture et Système", Chap. I : Architecture de base d'un ordinateur, UMR 7030 - Université Paris 13 - Institut Galilée, (Université Bordeaux - Architecture de l'Ordinateur, Année 2007-2008
- Olivier Marchetti (CM-TD-TP) Laurent Lambert (TD-TP) Laboratoire d'informatique de Paris 6, Pôle SoC, UPMC Année 2016-2017.
- Paolo Zanella, Yves Ligier, Emmanuel Lazard, Architecture et technologie des ordinateurs : Cours et exercices - Collection : Sciences Sup, Dunod, 6ème édition, 2018.
- Pirouz Bazargan, François Dromard, Alain Greiner, Processeur MIPS R3000.Architecture Interne Microprogrammée, Version 2.5, Septembre 2002.


# Liens vers le microprocesseur MIPS R3000

- ftp://132.227.86.9/pub/mips/mips.asm.pdf
- ftp://asim.lip6.fr/pub/mips/mips.externe.pdf
- ftp://asim.lip6.fr/pub/mips/mips.interne.pdf

# Liste d'abréviations

- ACC : Registre Accumulateur
- AO : Architecture des Ordinateurs
- CO: Compteur Ordinal (or PC for Program Counter)
- CPU : Central Processing Unit
- DDR SDRAM : Double Data Rate Synchronous Dynamic RAM
- DPRAM : Dual Ported RAM
- DRAM : Dynamic RAM
- EDVAC : Electronic Discrete Variable Automatic Computer
- HDMI: High Definition Multimedia Interface
- MC : Mémoire Centrale
- MP : Mémoire Principale
- MRAM : Magnetic RAM
- PP : Pointeur de Pile (or SP for Stack Pointer)
- R@M: Registre d'adresse Mémoire
- RA : Registre d'Adresse (or AR for Adress Register)
- RAM : Random Access Memory
- RD : Registre de Données (or DR for Data Register)
- RDM : Registre de Données Mémoire
- RE : Registre d'Etat (or PSW for Processor Status Word)
- RI : Registre d'Instruction (or IR for Instruction Register)
- ROM : Read Only Memory
- SDRAM : Synchronous Dynamic RAM
- SRAM : Static Random Access Memory
- UAL : Unité Arithmétique et Logique
- UCC : Unité de Contrôle et de Commande
- USB : Universal Serial BUS
- VGA : Video Graphics Array
- VRAM : Video RAM

# Table des matières

## Partie Cours