République Algérienne Démocratique & Populaire
Ministère de l'Enseignement Supérieur & de la Recherche Scientifique
Université Dr Moulay Tahar de Saida
Faculté de Technologie Département d'informatique

# Architecture des Ordinateurs

Deuxième année L.M.D. Informatique

Abdelkrim LATRECHE

Année 2020

# Préface

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 de leur codage, le dialogue entre composants, et le fonctionnement logique (pas électrique) interne des composants.

Ce cours a pour objectif de mettre en clair le principe de fonctionnement de l'ordinateur avec une présentation détaillée de l'architecture de l'ordinateur.

Nous abordons ce cours avec le chapitre 1 sur l'introduction à la notion d'architecture des ordinateurs et l'architecture de la machine de Von Neumann et la machine Harvard.

Le chapitre 2 porte sur les principaux composants d'un ordinateur (Schéma global d'une architecture, L'UAL, les bus, les registres, la mémoire interne : mémoire RAM (SRAM et DRAM), ROM, la mémoire cache, hiérarchie de mémoires, etc.

Le chapitre 3 traite les notions sur les instructions d'un ordinateur : langage de haut niveau, assembleur, langage machine, les instructions machines usuelles, principe de compilation et d'assemblage (notions de base), l'unité de contrôle et de commande, phases d'exécution d'une instruction, UCC pipeline, L'horloge et le séquenceur

Le chapitre 4 porte sur l'architecture externe et interne d'un processeur MIPS R3000. On présente ici successivement : le rôle du processeur, calcul de CPI (Cycle per Instruction), les processeurs CISC et RISC, le microprocesseur MIPS R3000, la structure externe du processeur MIPS R3000, la structure interne du processeur MIPS R3000, le jeu d'instructions, formats et programmation du MIPS R3000 et enfin initier les étudiants à la programmation du MIPS R3000. On traite aussi successivement les notions sur les interruptions, les entrées-sorties et les instructions systèmes (cas du MIPS R3000).

Le recueil regroupe, entre autres, des exercices proposés durant les travaux pratiques, pour permettre à l'étudiant de mieux comprendre les notions théoriques de ce cours.

Ce recueil est un support auxiliaire, sa lecture ne devra pas remplacer la présence active, attentive et intéressée aux cours et aux TD, la recherche bibliographique et la lecture des autres textes de références dans ce domaine restent les meilleurs moyens pour approfondir vos connaissances.

# Table des matières

**Préface** ..... 1

**Chapitre 1 : La machine de Von Neumann et la machine Harvard**

1.1 Introduction à la notion d'architecture des ordinateurs ..... 5
1.2 La machine de Von Neumann ..... 5
1.3 La machine Harvard ..... 6

**Chapitre 2 : Principaux composants d'un ordinateur**

2.1 Schéma global d'une architecture ..... 8
2.2 Unité Arithmétique et Logique (U.A.L.) ..... 11
2.3 Les registres ..... 12
2.4 Les bus ..... 14
2.5 La mémoire ..... 15
2.5.1 Généralités et définitions ..... 15
2.5.1.1 Définitions ..... 15
2.5.1.2 Caractéristiques d'une mémoire ..... 15
2.5.1.3 Technologie des supports ..... 16
2.5.1.4 Méthode d'accès ..... 16
2.5.1.5 Hiérarchies des mémoires ..... 16
2.5.2 Mémoire centrale ..... 17
2.5.2.1 Introduction ..... 17
2.5.2.2 Caractéristiques de la mémoire centrale ..... 18
2.5.2.3 Organisation de la mémoire centrale ..... 19
2.5.2.4 Cycle de lecture/Ecriture ..... 20
2.5.2.5 Ordonnancement des octets en mémoire ..... 21
2.5.2.6 Notion d'adresse mémoire ..... 21
2.5.2.7 Typologie des mémoires ..... 22
2.5.2.8 Conception des MC ..... 24
2.5.3 Mémoire cache ..... 25
2.5.3.1 Introduction ..... 25
2.5.3.2 Principe de fonctionnement et généralités ..... 25
2.5.3.3 Méthodes d'accès aux caches ..... 28
2.5.3.4 Caches de données et caches d'instructions ..... 29
2.5.3.5 Caractéristiques des mémoires caches ..... 29
2.5.3.6 Techniques ou politiques de remplacement dans un cache ..... 30
2.5.3.7 Ecriture en mémoire cache ..... 30
2.5.3.8 Les niveaux de caches ..... 30

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

3.1 Langage de haut niveau, assembleur, langage machine ..... 32
3.2 Principe de compilation d'assemblage et d'éditeur de liens ..... 33

3.3 Structure d'une instruction machine ....33
3.4 Jeu d'instructions et modèles de programmation ....34
3.4.1 Jeu d'instruction ....34
3.4.2 Modèles de programmation ....35
3.5 Modes d'adressage ....35
3.6 L'unité de contrôle et de commande (UCC) ....37
3.6.1 Structure d'une U.C.C ....37
3.6.2 Fonctionnement d'une UCC : ....37
3.7 Le pipeline ....39
3.7.1 Etages d'un pipeline ....39
3.7.2 Performances d'un pipeline ....40
3.7.3 Problèmes liés au pipeline ....41

**Chapitre 4 : Le processeur**

4.1 Le processeur ....42
4.1.1 Rôle du processeur ....42
4.1.2 Caractéristiques d'un processeur ....42
4.1.3 Performances d'un microprocesseur ....43
4.1.4 Notion d'architecture RISC et CISC ....44
4.2 Processeur MIPS R3000 : Architecture externe ....45
4.2.1 Introduction ....45
4.2.2 Registres visibles du logiciel ....45
4.2.2.1 Les registres non protégés ....46
4.2.2.2 Registres protégés ....46
4.2.3 Organisation de la mémoire ....47
4.2.4 Jeu d'instructions ....48
4.2.4.1 Généralités ....48
4.2.4.2 Les instructions arithmétiques ....48
4.2.4.3 Les instructions logiques ....50
4.2.4.4 Les instructions de transfert de données ....50
4.2.4.5 Instruction de chargement rangement dans la MP (Lecture-Ecriture) ....50
4.2.4.6 Instructions de branchement ....51
4.2.5 Programmation en assembleur MIPS ....52
4.2.5.1 Règles syntaxique ....52
4.2.5.2 Traduction des structures de contrôle en MIPS ....53
4.2.5.3 Organisation d'un programme en assembleur ....54
4.2.5.4 Appels système et instructions d'entrée/sortie ....54
4.2.5.5 Traitement des tableaux ....56
4.2.6 Directives supportées par l'assembleur MIPS ....57
4.2.6.1 Déclaration des sections : text, data et stack ....57

4.2.6.2 Déclaration et initialisation de variables ..... 57
4.2.7 Format des instructions et codage en langage machine ..... 58
4.2.7.1 Formats des instructions ..... 58
4.2.7.2 Codage des instructions ..... 59
4.2.7.3 Décodage d'une instruction en assembleur MIPS ..... 60
4.3 Processeur MIPS R3000 : Architecture interne ..... 62
4.3.1 Interface du Processeur ..... 62
4.3.2 Architecture Interne ..... 62
4.3.3 Chemin de Données ..... 63
4.3.4 Langage de la micro-programmation ..... 64
**Travaux dirigés ..... 68**
**Références ..... 83**

# Chapitre 1 : La machine de Von Neumann et la machine Harvard

## 1.1 Introduction à la notion d'architecture des ordinateurs

L'architecture des ordinateurs est l'étude et la description du fonctionnement des composants internes d'un ordinateur, en explicitant leur construction et leurs interactions. Elle traite :

- le type des informations manipulées et de leur codage,
- le dialogue entre composants,
- et le fonctionnement logique (pas électrique) interne des composants.

Un ordinateur est un outil complexe qui peut effectuer des tâches variées, et dont les performances globales dépendent des spécifications de tous ses éléments. Comprendre l'architecture d'un ordinateur permet de savoir dans quelle mesure les caractéristiques propres à chaque composant fonctionnel influencent la réactivité de la machine en fonction de son usage : pourquoi ajouter de la mémoire accélère-t-il l'ordinateur ? Comment les processeurs font-ils pour aller toujours beaucoup plus vite ?

L'utilité de ce domaine de connaissances est encore plus évidente pour les programmeurs qui développent des applications, des étudiants qui s'amusant sur leur matériel personnel au professionnel écrivant des lignes de code au sein d'une équipe. La programmation dans des langages évolués est théoriquement indépendante des contraintes architecturales, mais dans les faits, celles-ci ont un impact sur l'écriture, la correction et les performances des programmes, via, par exemple, l'utilisation des pointeurs, le choix de la taille des variables ou de l'ordre des boucles. Négliger ou ignorer la structure de l'ordinateur amènera tôt ou tard à des déconvenues et à des erreurs.

## 1.2 La machine de Von Neumann

John Von Neumann est à l'origine d'un modèle de machine universelle de traitement programmé de l'information (1945).

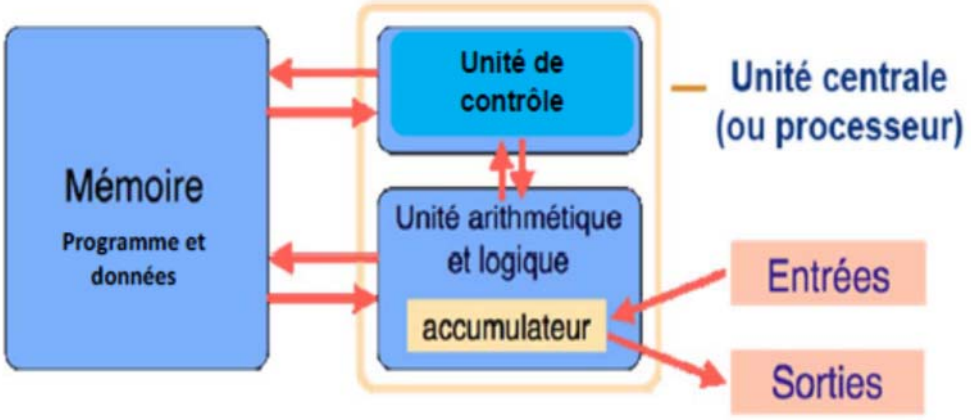



*Figure 1.1. **Architecture de Von Neumann***

L'architecture, dite **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 requises ou générées par le calcul. De telles machines sont aussi connues sous le nom **d'ordinateurs à programme stocké en mémoire**. La séparation entre le stockage et le processeur est implicite dans ce modèle.

Le schéma décrit par Von Neumann reste encore valable pour les ordinateurs actuels. En effet l'architecture, dite architecture de Von Neumann décompose l'ordinateur en quatre parties distinctes :

1. L'unité centrale de traitement (UCT, CPU, Processeur) qui comprend :
   - l'unité arithmétique et logique (UAL ou ALU en anglais).
   - l'unité de contrôle ou de commande (UCC).
2. La mémoire centrale (MC) qui contient à la fois les données et le programme exécuté par l'unité de contrôle ;
3. Les dispositifs d'entrée-sortie, qui permettent de communiquer avec le monde extérieur à travers les unités périphériques.
4. Les bus qui sont chargés de faire circuler des informations entre les différents composants de la machine.

**Cycle d'exécution de Von Neumann**

Le fonctionnement schématique de l'ordinateur est comme suit :

1. Extraction de la prochaine instruction à exécuter de la mémoire ;
2. Analyse et décodage de l'instruction ;
3. Recherche dans la mémoire les données concernées par l'instruction s'il y a lieu ;
4. Exécution de l'opération et rangement du résultat ;
5. Retour à 1

## 1.3 La machine Harvard

Le nom de cette structure vient du nom de l'université Harvard où cette architecture a été mise en pratique pour la première fois en 1944 (avec le Mark I). L'architecture de type Harvard est une conception des processeurs qui sépare physiquement la mémoire de données et la mémoire programme. L'accès à chacune des deux mémoires s'effectue via deux bus distincts, ce qui permet de transférer simultanément les données et les instructions à exécuter. De cette façon, l'unité de traitement aura 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, mais ce gain en performance s'obtient cependant au prix d'une complexité accrue de structure.

L'architecture Harvard est souvent utilisée dans :

- les processeurs numériques de signal (DSP) ;
- les microcontrôleurs, notamment les PIC de Microchip et les AVR d'Atmel.

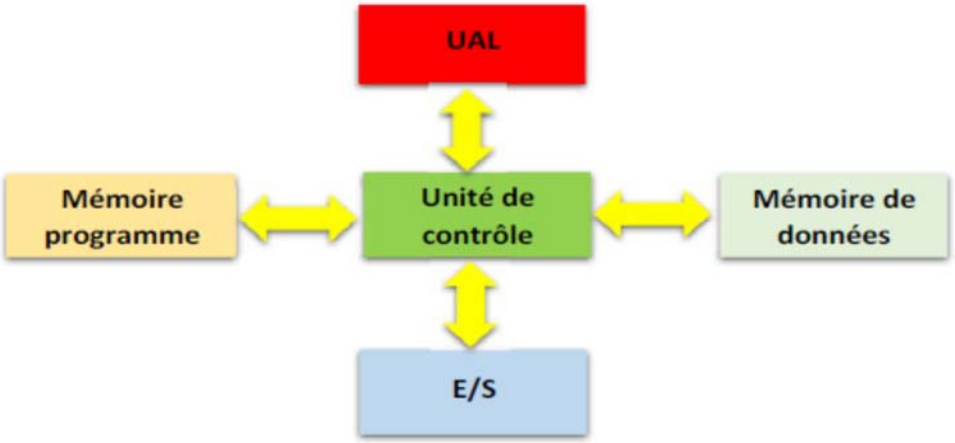



*Figure 1.2. Architecture Harvard*

Le tableau ci-dessous présent une brève comparaison entre l'architecture de Von Neumann et celle de Harvard

| | VON NEUMANN<br>(Motorola, Intel,…) | HARVARD<br>(DSP,…) |
|---|---|---|
| Caractéristiques | – 1 seule mémoire qui contient les instructions et les données<br>– 1 bus d'adresse, 1 bus (bidirectionnel) de donnée<br>– Logique complexe pour lire/écrire tour à tour les données et les instructions<br>– Une instruction contient le code opératoire et l'adresse opérante | – 2 mémoires : 1 pour les instructions, 1 pour les données<br>– 2 bus d'adresse, 2 bus (bidirectionnels) de donnée<br>– Transfert d'instruction et données simultanément |
| Avantages | – Jeu d'instruction riche<br>– Accès facile à la mémoire | – Jeu d'instruction facile à mémoriser<br>– Codage d'instructions facile (chaque instruction est codée sur un mot et dure un cycle machine)<br>– Code plus compact |
| Inconvénients | – Temps d'exécution d'une instruction variable<br>– Codage d'instruction sur plusieurs octets | – Jeu d'instruction très pauvre<br>– Accès aux registres très délicats |

# Chapitre 2 : Principaux composants d'un ordinateur

## 2.1 Schéma global d'une architecture

Le schéma décrit par Von Neumann reste encore valable pour les ordinateurs actuels, l'évolution a permis d'améliorer les caractéristiques des éléments sans altérer le principe général. En effet tout ordinateur doit posséder les unités fonctionnelles suivantes :

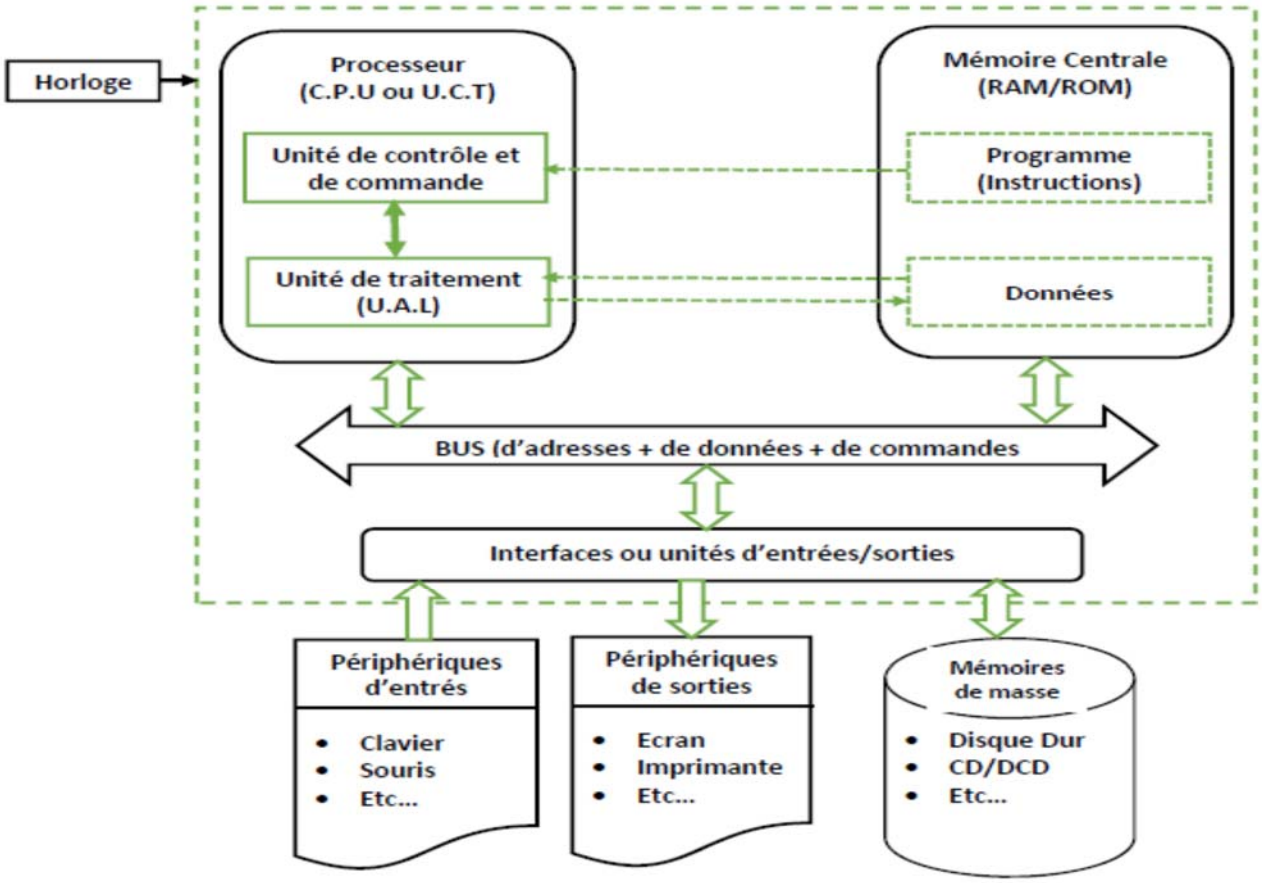



*Figure 2.1. Schéma global simplifié d'un ordinateur*

1. **Unité Centrale de Traitement (**U.C.T) ou Central Processing Unit (C.P.U) ou Processeur est l'élément moteur de l'ordinateur qui interprète et exécute les instructions du programme. Véritable cerveaux de l'ordinateur, le CPU est étroitement associé à la mémoire où sont stockées les instructions et les données à traiter. L'ensemble CPU + mémoire centrale est souvent appelé unité centrale.

   Unité Centrale de Traitement est composée de deux unités fonctionnelles :

   - **l'unité de commande ou de contrôle (U.C.C)** commande l'exécution de toutes les opérations à toutes les niveaux (UAL, mémoire et entrées/sorties) ainsi que le contrôle de leurs déroulement.
   - **L'Unité Arithmétique et Logique (U.A.L.),** exécute les instructions arithmétiques et logiques qu'effectués l'unité de traitement.

2. **Mémoire Centrale (MC) ou Mémoire Principale (MP) :** C'est la partie qui contient les programmes et les données qui seront exécutés par le Processeur. La mémoire est divisée en emplacements de taille fixe (par exemple 8 bits) utilisés pour stocker instructions et données.

Il existe deux types de mémoires internes :

- **Mémoire vive *(R.A.M) :*** La RAM (Random Acess Memory) est une mémoire volatile qui permet la lecture/écriture des données, c'est là où sont stockées les informations (instruction d'un programme et les données) au cours d'exécution. Elle se caractérise par une très grande rapidité de lecture/écriture.
- **Mémoire morte *(R.O.M) :*** La R.O.M (Read Only Memory) est une mémoire non volatile et qui ne peut être que lue, Les programmes sont enregistrés une fois pour toutes dans cette mémoire et ne peuvent être ni modifiés ni effacés, même après coupure de l'alimentation électrique.

3. **Mémoire auxiliaire (Secondaire)** : Elles permettent d'alléger la mémoire centrale qui est volatile en ce sens qu'elles permettent de stocker l'information de façon permanente. Certaines de ces mémoires sont de très grande capacité. Les principaux supports de mémoire de mase sont : le disque dur, les disquettes, les bandes magnétiques, les cassettes magnétiques, les disques optiques numériques (CD ROM), les mémoires flash (clé USB) etc. Le temps d'accès à une donnée est beaucoup plus grand que dans le cas de mémoire principale.
4. **Les unités d'entrée/sortie** : Constituent un moyen d'échange avec l'extérieur, elles relient l'ordinateur au monde extérieur à travers les unités périphériques (écran, clavier, Imprimante, souris, disques, bandes magnétiques, réseaux, etc.). Elles sont en communication avec le processeur et la mémoire centrale grâce à des contrôleurs.
5. **Les Bus** : Sont chargés de faire circuler des informations entre les différents composants de l'ordinateur. Le bus est piloté par le processeur. Chaque bus est une série de lignes électriques parallèles :
    - Bus d'adresse : véhiculer l'adresse mémoire ou d'unité E/S engendrée par l'UC (et dans certains cas par d'autres unités de contrôle) ;
    - Bus de données : véhiculer l'information (instructions, données) entre l'UC, la mémoire et les unités E/S ;
    - Bus de commande : véhiculer les signaux employés par l'UC pour le contrôle du système (Adresse mémoire valide, Adresse E/S valide, Lecture/écriture, Attente, Interruption, etc.).
6. **L'horloge :** Le rôle de l'horloge dans l'UCC est de générer des signaux périodiques qui définissent le cycle machine qui correspond à la durée élémentaire régissant le fonctionnement de la machine, il synchronise toutes les actions du processeur. La fréquence de cette horloge s'exprime en MHz.

Dans cette section nous avons décrit brièvement le rôle et le fonctionnement de chaque unité fonctionnelle. Dans les sections suivantes nous traitons en détaille chaque une de ces unités.

## Principe de fonctionnement d'un ordinateur

Comme on vu précédemment, un ordinateur est constitué d'une mémoire centrale, contenant programmes (suites d'instructions) et données associées, d'une unité centrale de traitement qui exécute un programme chargé en mémoire centrale et d'unités d'entrées/sorties permettant l'échange d'information avec les unités périphériques. L'ordinateur exécute le programme instruction par instruction.

Comment se déroule l'exécution d'une instruction ? et comment inter-réagissent l'ensemble UCC et UAL ?

- L'UCC cherche en MC l'instruction à exécuter en lui envoyant une adresse et une commande.
- L'instruction est transférée vers l'UCC pour décodage pour déterminer l'opération à exécuter.
- Des signaux sont envoyés à l'UAL pour déclencher l'opération.
- Les données nécessaires sont acheminées de la MC vers UA.
- Une **fois** l'exécution terminée, les résultats sont envoyés à la MC.

On récapitule le fonctionnement d'un processeur à accumulateur dans le schéma suivant :

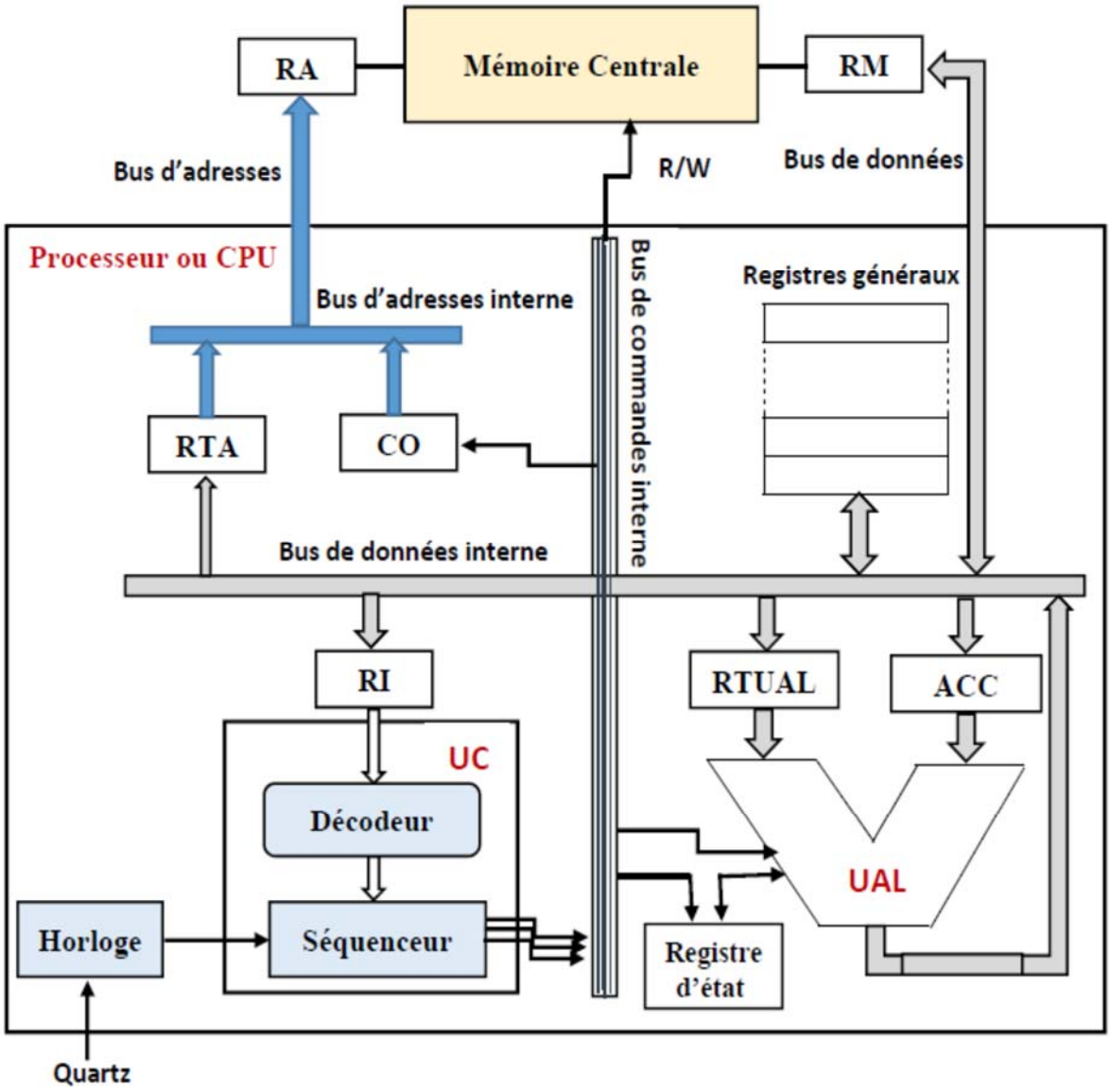



*Figure 2.2 : Schéma fonctionnel d'un CPU*

## 2.2 Unité Arithmétique et Logique (U.A.L.)

L'UAL est un circuit logique combinatoire formé de portes logiques prenant deux nombres en entrée et générant un nombre en sortie en fonction de signaux de commande indiquant l'opération arithmétique (addition, soustraction, division, décalage, test d'infériorité, test de supériorité, etc.) ou l'opération logique (ET, OU, NON, etc.) à effectuer.

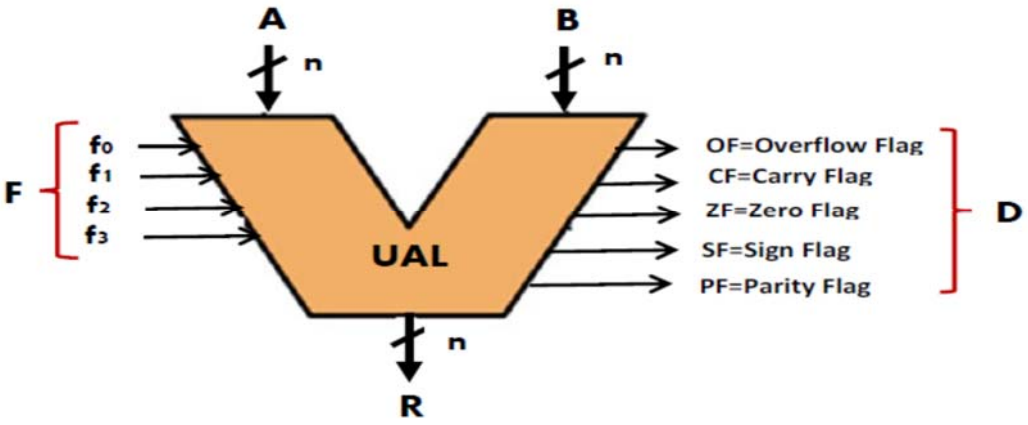



*Figure 2.3. Schéma simplifie d'un UAL*

L'U.A.L, dispose en gros de :

- deux entrées de données **A** et **B** sur lesquelles on présentera les données à traiter ;
- une sortie **R** donnant le résultat de l'opération effectuée ;
- une entrée de sélection **F** permettant le choix de l'opération à effectuer ;
- Un indicateur donnant l'état du résultat après exécution de l'opération. (Drapeau qui indiquera s'il y a eu erreur : division par zéro, résultat négatif, dépassement de capacité …)

Une UAL peut être caractérisée par sa taille et ses possibilités.

Sa **taille**, ou **largeur**, correspond au nombre maximum de bits que peuvent occuper les entrées, autrement dit à la taille maximale des nombres que l'UAL peut traiter. Elle est évidemment fortement liée à la taille des nombres que les registres peuvent stocker puisque ce sont les mêmes qui passent par l'UAL. Il est toujours possible de travailler avec des nombres plus grands, mais il faut alors découper l'opération : on peut additionner de deux nombres de 64 bits dans une UAL de 32 bits de large en ajoutant d'abord les 32 bits de poids faible puis les 32 bits de poids fort, mais cela nécessite au moins deux instructions. Inversement, calculer sur des nombres de 32 bits dans l'UAL ne présente pas d'intérêt si les registres ne peuvent pas stocker plus de 16 bits.

Les **possibilités** de l'UAL correspondent simplement aux différentes commandes qu'elle reconnaît : elle peut certainement faire une addition et une soustraction, mais est-ce qu'elle peut faire par exemple des multiplications et des divisions ? Est-ce que toutes les fonctions logiques sont-elles possibles ?

Le constructeur d'un processeur a toujours la possibilité d'augmenter sa taille (pour permettre la manipulation de nombres plus grands) ou d'étendre les capacités de l'UAL (en autorisant des opérations plus complexes). Mais tout cela implique d'utiliser plus de portes logiques et donc d'occuper plus de place sur la puce, et cela au détriment d'autres circuits.

Un dernier paramètre important de l'UAL est sa vitesse de fonctionnement. Utilisé à chaque instruction de calcul, sa rapidité influence directement la vitesse d'exécution. Il faut donc en premier lieu optimiser ce critère.

**Réalisation d'une UAL 1 bit**

Pour concevoir une UAL à 1 bit qui traite 4 opérations a et b, a ou b, non b, a + b**.** On doit disposer d'un circuit capable de sélectionner l'opération à exécuter, ce circuit ne peut être qu'un décodeur à deux bits et doit avoir deux entrées f0 et f1, à partir desquelles, on peut effectuer 4 choix (00, 01,

10, 11). Pour réaliser l'opération arithmétique a + b, on utilise un circuit appelé additionneur, qui prendre en considération une éventuelle retenue.

La figure ci-dessous montre un exemple d'UAL 1 bit simplifiée. Cette UAL est capable de faire le ET, le OU de 2 bits, le NON du second bit, et la somme des 2 bits avec une retenue d'entrée. Le choix parmi ces quatre opérations se fait via les deux lignes de commandes f0 et f1. Suivant la valeur de ces 2 bits, le décodeur active une des quatre lignes de sortie, sélectionnant soit une des trois fonctions logiques ($a\ et\ b, a\ ou\ b, \overline{b}$), soit la retenue et la sortie de l'additionneur. Ainsi en reliant chacune des trois premières sorties du décodeur à la porte logique appropriée par une porte AND on active une des 3 opérations logiques et en reliant la quatrième sortie à l'additionneur par une porte AND on active l'additionneur et notre schéma complet de l'UAL 1 bit sera le suivant :

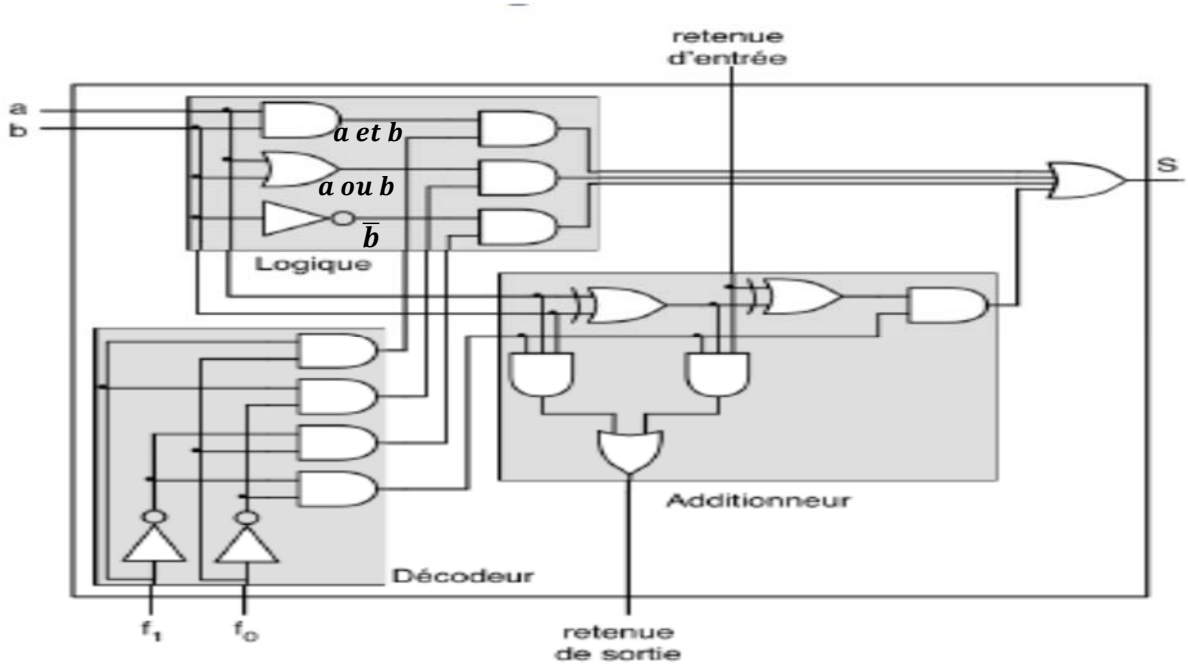



*Figure 2.4. UAL de 1 bit simplifiée*

**Remarque** : pour réaliser une UAL à n bits on a qu'à relier ensemble n circuits d'UAL à 1 bit.

## 2.3 Les registres

Les registres sont une zone de stockage temporaire des données située dans le processeur. Ils sont construits à base de circuits logiques séquentiels pour pouvoir mémoriser des bits. Leur intérêt principal est de pouvoir travailler avec des données localisées directement dans le processeur et donc d'un accès beaucoup plus rapide que celles situées en mémoire principale. Les registres contiennent temporairement les opérandes et les résultats des opérations ou des données comme une instruction ou une adresse. Chaque registre stocke 8, 16, 32 ou 64 bits. Le nombre exact de registres dépend du type de processeur. Il n'est absolument pas nécessaire que les registres soient homogènes bien qu'on puisse souvent grouper les registres en classes, de sorte que les registres d'une classe peuvent être utilisés à la place de n'importe quel autre registre de la même classe. Mais souvent, l'espace de travail comporte des registres ayant des fonctions particulières.

a) **Un accumulateur** : L'accumulateur (ACC) est un registre de l'UAL servant de source implicite à la plupart des instructions et contenant le résultat après leur exécution. Dans certain cas, il peut servir de tampon dans les opérations d'entrées/sorties.

b) **Registres généraux :** Les registres généraux (regroupés dans un banc de registres) sont à la disposition du programmeur en assembleur (ou du compilateur si l'on code en langage évolué), qui les utilise à sa guise dans les instructions pour manipuler des données. Ces

registres permettent de limiter les accès à la mémoire en concevant les informations utilisés fréquemment par le processeur telles que les résultats intermédiaires.

c) **Registre d'instruction (RI) :** Lorsqu'une instruction est récupérée en mémoire pour être exécutée dans le processeur, elle est mémorisée dans un registre spécial : le registre d'instruction (RI ou IR, Instruction Register). Le séquenceur utilise ce lieu de stockage pour disposer de l'instruction et des bits la composant pendant tout le temps de son exécution. Il gère entièrement l'utilisation de ce registre et le programmeur n'y a jamais accès.

d) **Compteur ordinal (CO) :** est un registre spécial, appelé aussi Program Counter (PC), ou encore Instruction Pointer (IP) qui contient l'adresse de l'emplacement mémoire où se situe la prochaine instruction à exécuter. Le processeur incrémente alors PC de la taille de l'instruction pour que le registre pointe sur la suivante (rappelons que chaque adresse mémoire correspond à 1 octet et qu'une instruction est longue de plusieurs octets ; PC doit donc être augmenté du nombre d'octets de l'instruction, et non de 1).

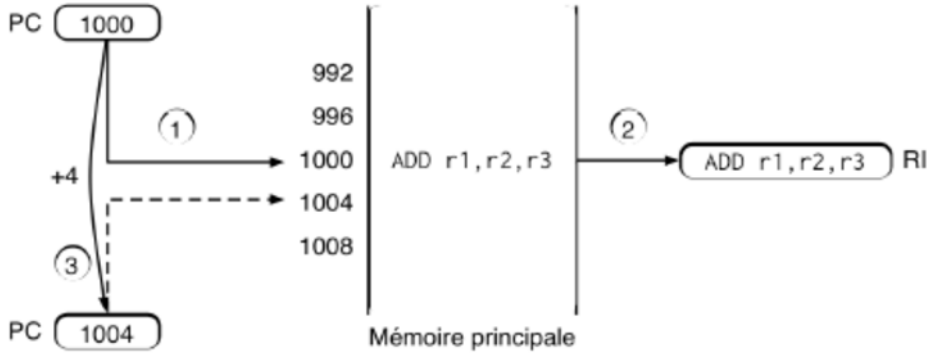



*Figure 2.5. Incrémentation de PC*

La figure 2.5 montre le début de l'exécution d'une instruction : sa récupération à l'adresse contenue dans PC, le stockage dans RI et l'incrémentation de PC.

Le programmeur n'a pas directement accès à ce registre mais peut le modifier via les instructions de branchement.

e) **Registre d'état :** Le registre d'état (State Register ou Program Status Word) est tel que ses bits ne forment pas de valeur numérique mais servent d'indicateurs (aussi appelés « drapeaux » ou flags) sur l'état du processeur. Certains bits peuvent être positionnés par le programmeur pour demander un comportement particulier, d'autres (appelés « bits conditions ») sont automatiquement mis à jour par le séquenceur à la fin de l'exécution de chaque instruction. Certains de ces bits, sont suffisamment universels pour qu'on les retrouve sur tous les modèles de processeurs (mais leur nom peut varier) :

- Le bit Z (Zero). Il est mis à 1 si le résultat de l'instruction est nul, sinon il est mis à 0.
- Le bit C (Carry). Il est mis à 1 si l'instruction génère une retenue finale, sinon est mis à 0.
- Le bit N (Negative). Il est mis à 1 si le résultat de l'instruction est négatif, sinon il est mis à 0 ; c'est la recopie du bit de poids fort du résultat.
- Le bit V (oVerflow). Il est mis à 1 si l'instruction génère un débordement arithmétique, sinon il est mis à 0.

L'utilisation de ces bits conditions se fait toujours de la même manière : d'abord, on effectue un calcul, un test ou une comparaison, positionnant les bits conditions suivant le résultat que l'on veut obtenir, puis l'on effectue un saut conditionnel sur ces bits.

**f) Registre pointeur de pile** : Le pointeur de pile ou Stack Pointer (SP) indiquant l'adresse du sommet d'une pile gérée par le processeur, c'est-à-dire sur la première case mémoire libre à son sommet. Là sont empilées ou dépilées des informations, soit automatiquement par le séquenceur, soit à la demande du programmeur.

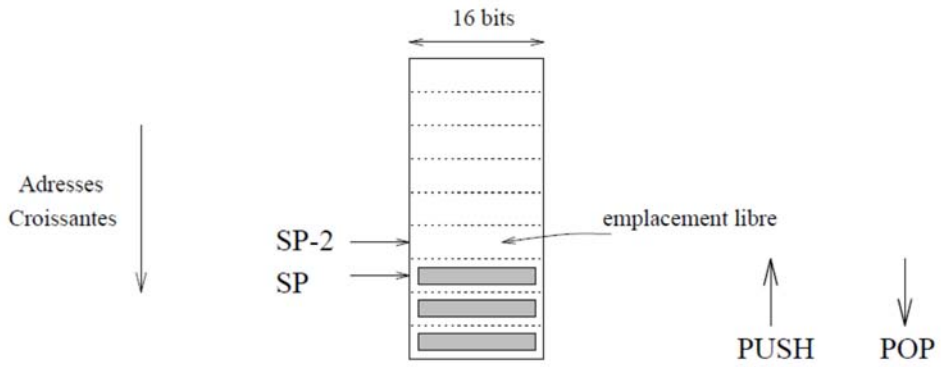



*Figure 2.6. La pile. Les adresses croissent vers le bas. SP pointe sur le sommet.*

**g) Registres de base et d'index** servant au calcul d'adresse par rapport à une base ou index.

## 2.4 Les bus

Lorsque plusieurs composants de l'ordinateur (processeur, cache, mémoire, périphérique d'entrées/sorties…) veulent échanger de l'information, ils utilisent un bus, c'est-à-dire un ensemble de fils reliant plusieurs circuits entre eux, dont la structure et les spécifications fonctionnelles sont bien définies. Un bus assure la transmission du même type d'information.

Les bus sont caractérisés par :

- Le volume d'informations transmises simultanément. Cette quantité (en bits) correspond au nombre de lignes physiques servant à envoyer les données de manière parallèle. Une nappe de 32 fils permet de transmettre 32 bits en parallèle.
- Sa vitesse ou sa fréquence (en Hz) exprimant le nombre de paquets de données envoyés ou reçus par seconde. Il est possible de connaître la quantité de données qu'un bus peut transporter par unité de temps, en multipliant sa longueur de mot par sa fréquence. Un bus de 16 bits, cadencé à une fréquence de 133 MHz possède donc un taux de transfert égal à : 16 * $133.10^6$ = $2128*10^6$ bit/s = $266*10^6$ octets/s = $266*10^3$ Ko/s = 266 Mo/s

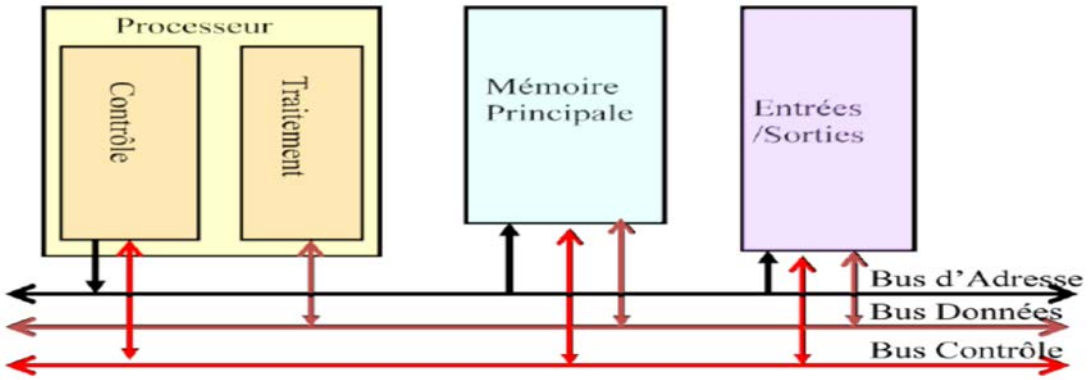



*Figure 2.7. Schéma représentant les trois types de Bus*

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

1. **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 (véhicule les données). Notons que plus ce bus

est large plus le système sera rapide. En effet un système 16 Bits est plus rapide qu'un système 8 bits car il est capable de 'transporter' 2 octets à la fois alors que le système 8 bits n'en transporte qu'un à la fois (1 octet = 8 bits).

On distingue deux types :

- Bus externe : qui est destiné à transporter des données entre la mémoire vers le processeur et vis-versa.
- Bus interne : qui est destiné à transporter des données entre l'UAL vers les registres et vis-versa.

2. **Bus d'adresses :** unidirectionnel qui permet la sélection des informations à traiter dans un espace mémoire ou espace adressable (véhicule les adresses).
   Exemple : bus d'adressage 16 bits = 64Ko adressables, bus 32 bits = 4Go.
   Il permet de transporter l'adresse de l'instruction à exécuter, ainsi que l'adresse des opérandes (données et résultats). L'adresse de l'instruction est donnée par le compteur ordinal (CO). L'adresse de l'opérande est donnée est donnée par le registre d'instruction (RI).

3. **Bus de commande** : constitué par quelques conducteurs qui assurent la synchronisation des flux d'informations sur les bus des données et des adresses. Il permet de véhiculer les signaux de commandes émises par l'unité de commande et de contrôle (UCC) vers les autres unités. On sait que le séquenceur avait pour rôle d'émettre des signaux de commandes qui sans transmis vers, par exemple :
   - L'UAL pour l'exécution de l'opération sélectionnée.
   - La MC pour la lecture des opérandes.
   - La MC pour l'écriture des résultats.
   - La MC pour la lecture de l'instruction suivante (à travers le CO).

## 2.5 La mémoire

### 2.5.1 Généralités et définitions

#### 2.5.1.1 Définitions

Une mémoire est un dispositif capable d'enregistrer, de conserver et de restituer des informations (instructions et données). Il y a écriture lorsqu'on enregistre des informations en mémoire, et lecture lorsqu'on récupère des informations précédemment enregistrées.

La mémoire peut être dans le processeur (des registres), interne (mémoire centrale ou principale) ou externe (mémoire secondaire).

#### 2.5.1.2 Caractéristiques d'une mémoire

- **La capacité** : c'est le nombre total de bits que contient la mémoire et s'exprime aussi souvent en octet.
- **Mode d'accès à l'information (lecture/écriture)** : Sur une mémoire on peut effectuer l'opération de :
  - lecture : récupérer / restituer une information à partir de la mémoire.
  - écriture : enregistrer une nouvelle information ou modifier une information déjà existante dans la mémoire.
  - lecture/écriture : Il existe des mémoires qui offrent les deux modes lecteur et écriture

- **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 sépare deux demandes successives de lecture ou d'écriture.
- **Le débit** : c'est le nombre maximum d'informations lues ou écrites par seconde. Le débit binaire=Longueur du mot/longueur du cycle. À titre d'exemple, considérons une mémoire ayant un cycle de 1.2 ms et des mots de 36 bits. Le taux de transfert (débit binaire) sera 36 bits / $1.2 \times 10^{-6}$ sec = $30 \times 10^{6}$
- **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.

#### 2.5.1.3 Technologie des supports

Il existe trois technologies pour les supports des mémoires :

- **Les mémoires électroniques** à base de circuits appelés semi-conducteurs, exemple : la mémoire centrale,
- **Les mémoires magnétiques** : qui utilisent un enregistrement magnétique, exemple : disques magnétiques,
- **Les mémoires optiques** : qui utilisent un système à base de laser pour l'enregistrement de l'information, exemple : CDROM

#### 2.5.1.4 Méthode d'accès

- **Accès séquentiel :** pour accéder à une information particulière, on est obligé de parcourir toutes celles qui la précèdent (par exemple des bandes magnétiques).
- **Accès direct ou aléatoire :** les informations ont une adresse propre, ce qui permet d'y avoir accès directement (par exemple, la mémoire centrale, les registres).
- **Accès par le contenu (mémoires associative) :** les informations sont identifiées par une clé et la recherche s'effectue sur cette clé de façon simultanée sur toutes les positions de la mémoire (par exemple, l'antémémoire ou mémoire cache).

#### 2.5.1.5 Hiérarchies des mémoires

Une mémoire idéale serait une mémoire de grande capacité et possédant un temps d'accès très faible. 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 pose le problème 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.

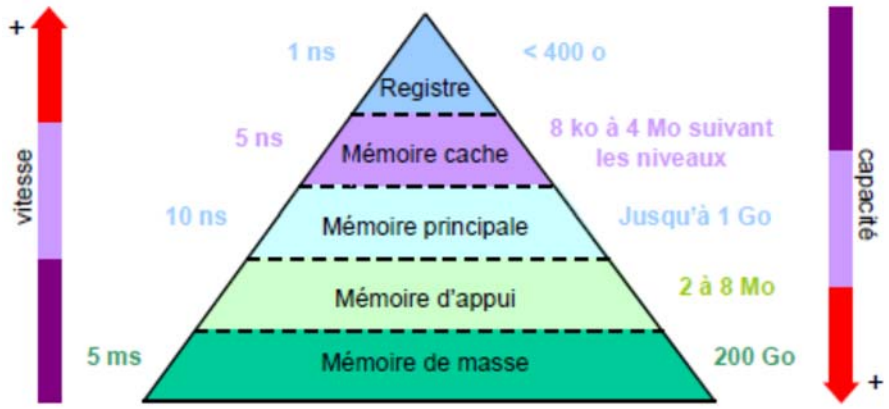



*Figure 2.8. Hiérarchies des mémoires*

- **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 des informations. Elle utilise pour cela des supports magnétiques (disque dur, ZIP) ou optiques (CDROM, DVDROM).

## 2.5.2 Mémoire centrale

### 2.5.2.1 Introduction

La mémoire centrale (principale ou interne) assure la fonction de stockage de l'information (programme machine accompagné de ses données) qui peut être manipulée par le microprocesseur (processeur central). Le processeur et la mémoire centrale sont étroitement liés et aucun d'eux ne peut jouer pleinement son rôle sans l'autre, en effet les données et les instructions du programme circulent entre le processeur et la mémoire centrale. La mémoire centrale est réalisée à base de semi-conducteurs. La capacité et le temps d'accès à la mémoire centrale sont deux éléments qui influent sur le temps d'exécution d'un programme (performance d'une machine).

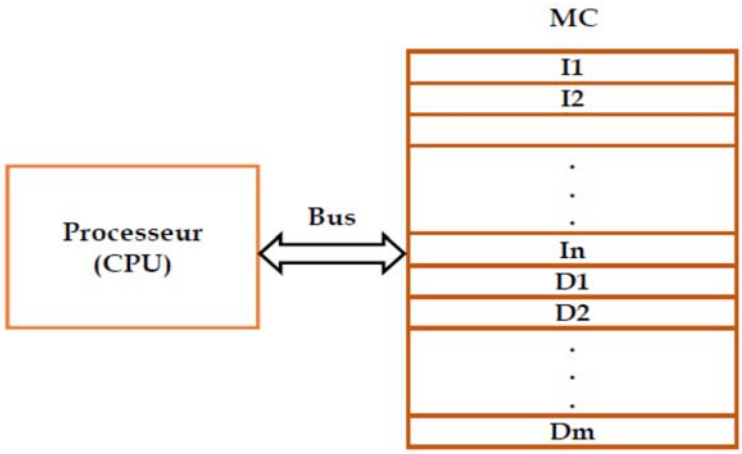



*Figure 2.9. Relation Processeur/Mémoire centrale*

### 2.5.2.2 Caractéristiques de la mémoire centrale

Plusieurs critères importants caractérisent les mémoires :

- ***la capacité de la mémoire*** indique la quantité d'information qu'une mémoire peut stocker. En général cette capacité peut s'exprimer en bits, en octets.
  - Octet : 1 Octet = 8 bits
  - kilo-octet (KO ) = 1024 octets = $2^{10}$ octets
  - Méga-octet ( MO) = 1024 KO = $2^{20}$ octets
  - Géga-octet ( GO) =1024 MO = $2^{30}$ octets
  - Téra-octet (TO) = 1024 Go =$2^{40}$ octets
- ***le temps d'accès de la mémoire*** caractérise le temps nécessaire pour obtenir une information en mémoire. Pour une mémoire électronique qui est une mémoire très rapide (RAM, ROM, registre…) ce temps se mesure en nanosecondes (milliardième de seconde : $10^{-9}$s). Cette vitesse peut également s'exprimer comme une fréquence d'horloge caractéristique de la mémoire, égale à l'inverse du temps d'accès et mesurée en hertz (Hz). Ainsi un temps d'accès égal à 10 nanosecondes correspond à une fréquence de 100 Mhz (1 Mhz = $10^6$ Hz).
- ***(Temps de) Cycle mémoire*** : temps minimal entre deux accès mémoire ; temps d'accès + temps des opérations de maintien, stabilisation, synchronisation, etc.
- ***La bande passante de la mémoire :*** Ce critère s'exprime sous la forme du produit de la largeur du bus de données et de la fréquence de la mémoire. Par exemple pour une mémoire de 64 bits de largeur sur un bus à 100 MHz la bande passante est de 800 Mo/s. Ce critère est très utilisé et tend à remplacer le critère de temps d'accès, il permet, en effet, de mieux évaluer les différents débits à synchroniser entre les modules de l'ordinateur ;
- ***La fréquence*** **:** Il existe un grand nombre de fréquences différentes pour les barrettes de RAM, sans compter ce qu'on peut obtenir avec l'overclocking. Ces fréquences varient environ de 133Mhz à 2000Mhz, mais la fréquence effective varie de 133/2=66.5Mhz à 2000/2=1000Mhz soit 1Ghz.
- ***La latence (CAS & timings)*** : La latence est souvent la donnée technique la moins comprise. Egalement baptisée timings, elle représente le nombre de cycles d'horloge, nécessaire pour accéder à une donnée stockée dans la mémoire. Les timings sont représentés par une série de 4 nombres sous la forme « 9-9-9-24 ». Ces quatre chiffres représentent dans l'ordre :
  - CAS latency (pour Column Address Strobe latency) : nombre de cycles d'horloge entre l'envoi de la commande de lecture et l'arrivée effective de la donnée. Autrement dit, il s'agit du temps d'accès à une colonne.
  - RAS Precharge Time (pour Row Address Strobe, noté parfois tRP,) : nombre de cycles d'horloge entre deux instructions RAS, c'est-à-dire entre deux accès à une ligne.
  - RAS to CAS delay (noté parfois tRCD) : nombre de cycles d'horloge correspondant au temps d'accès d'une ligne à une colonne.
  - RAS active time (noté parfois tRAS) : nombre de cycles d'horloge correspondant au temps d'accès à une ligne.

  Le plus important est le CAS Latency (Column Address Strobe) ou CL, temps d'accès à une colonne.

  Pour faire simple, plus les timings sont agressifs (réduits), et plus la mémoire sera rapide, pour une fréquence donnée. Par exemple, un CL de 7 sera meilleur qu'un CL de 9 pour de la DDR3, pour une fréquence donnée, et pour la DDR4 un CL de 12 sera meilleur qu'un CL de 16.

### 2.5.2.3 Organisation de la mémoire centrale

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 (donnée). La mémoire est donc composée de cellules possédant chacune un numéro qu'on appelle adresse mémoire numérotés de 0 à n-1. Chaque donnée devient alors accessible grâce à son adresse.

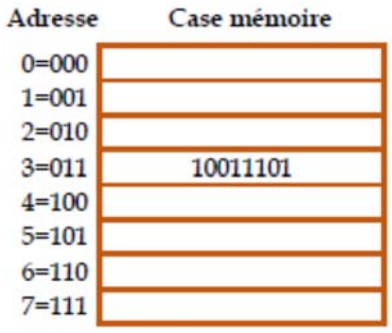
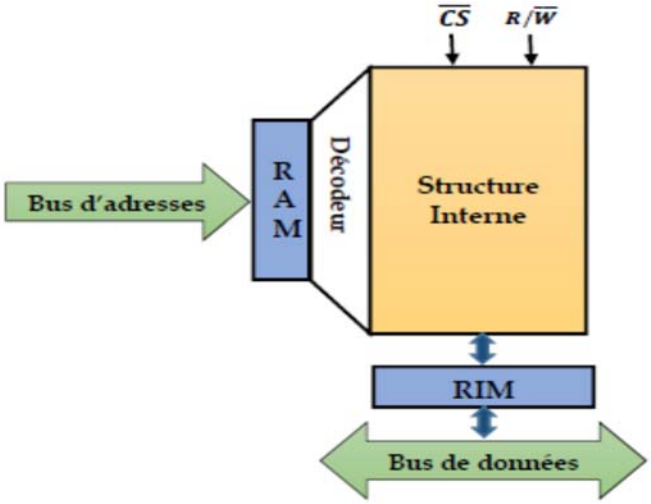



*Figure 2.10. Organisation de la mémoire centrale*

Avec une adresse de n bits il est possible de référencer au plus $2^n$ cases mémoire. Le nombre de fils d'adresses d'un boîtier mémoire définit donc le nombre de cases mémoire que comprend le boîtier. Chaque case est remplie par un mot de données (sa longueur m est toujours une puissance de 2). Le nombre de fils de données définit la taille des données sauvegardés dans chaque case mémoire. En plus du bus de données et du bus d'adresses, un boîtier mémoire comprend une entrée de commande de lecture/écriture et une entrée de sélection du boîtier.

La figure ci-dessous schématise un circuit mémoire où l'on peut distinguer :




- RAM (Registre Adresse Mémoire) : ce registre stock l'adresse du mot à lire ou à écrire.
- RIM (Registre 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.
- $R/\overline{W}$ : commande de lecture/écriture
- Bus d'adresses n bits
- Bus de données m bits
- $\overline{CS}$ : Signal sélection du circuit (Chip Select)

Une mémoire centrale est formée de trois éléments essentiels :

1**- Les circuits de mémorisation (stockage)** : Ce sont des cellules mémoires capables d'emmagasiner l'information et qui sont groupés en mots de m bits. Les mots mémoires sont arrangés à leur tour pour former des blocs mémoires contenant $2^n$ mots de m bits

2- **la logique d'adressage** : C'est l'ensemble des circuits qui permettent d'adresser un mot mémoire. Elle est composée d'un registre appelé registre adresse mémoire (RAM) qui permet d'enregistrer l'adresse du mot mémoire et d'un décodeur d'adresse qui permet de déterminer l'adresse effective du mot en MC.

3- **La logique d'entrée/sortie** : C'est une ensemble de circuits qui permettent la lecture ou l'écriture d'un mot mémoire composé d'un registre appelé registre information mémoire (RIM) qui sert à contenir l'information lue ou écrite et d'une interface d'entrée/sortie.

### 2.5.2.4 Cycle de lecture / écriture

**a-) Cycle de lecture :** Pour lire une information en MC il faut effectuer les opérations suivantes :

1. Etablissement de l'adresse de l'information à lire dans le registre RAM
2. Activer le signal de lecture (R/$\overline{W}$=1)
3. Sélectionner le boitier nécessaire ($\overline{CS}$)
4. Après un certain temps (temps d'accès), l'information apparait sur la sortie dans le registre RIM

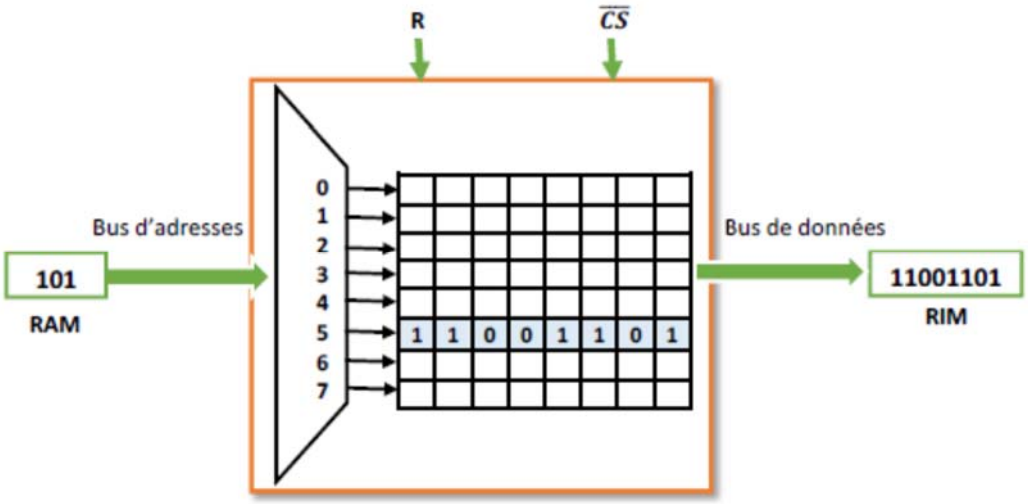



*Figure 2.11. Cycle de lecture en memoire*

La figure ci-dessous montre le chronogramme d'un cycle de lecture

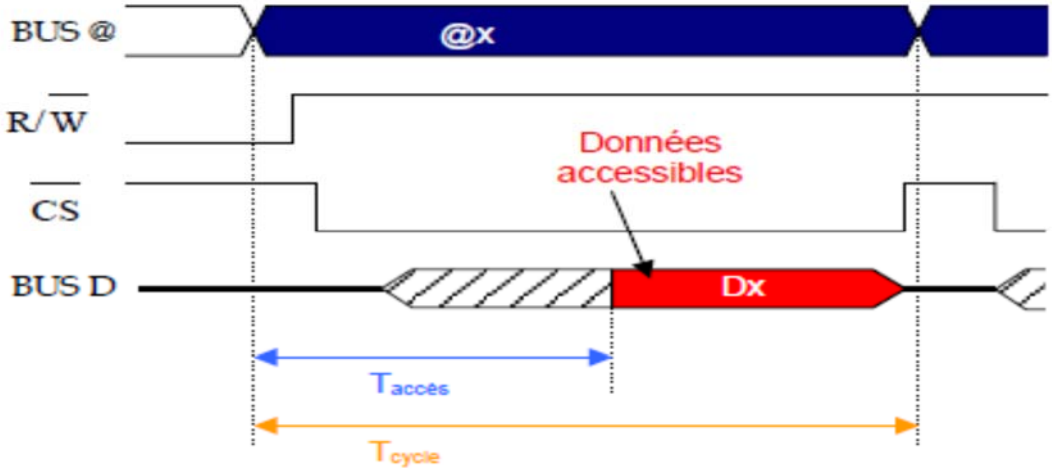



*Figure 2.12. Chronogramme d'un cycle de lecture*

**b-) Cycle d'écriture :** Pour écrire une information en MC il faut effectuer les opérations suivantes :

1. Etablir l'adresse ou se fera l'écriture dans le RAM
2. Sélectionner le boitier nécessaire ($\overline{CS}$)
3. Fournir la donnée sur l'entrée dans le registre RIM
4. Activer le signal d'écriture (R/$\overline{W}$=0)

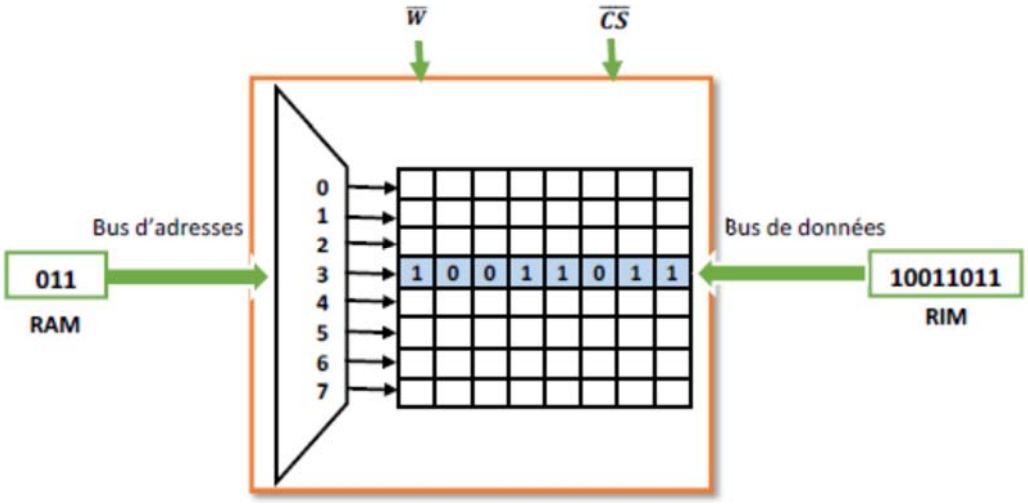



*Figure 2.13. Cycle d'écriture en mémoire*

### 2.5.2.5 Ordonnancement des octets en mémoire

Il existe 2 façons de ranger les octets qui composent un mot mémoire de 32 bits :

- L'octet de poids fort est stocké en premier : big-endian (gros boutiste)

| K | k+1 | k+2 | k+3 |
|---|---|---|---|
| Octet 3 | Octet 2 | Octet 1 | Octet 0 |

- L'octet de poids fort est stocké en dernier : little-endian (petit boutiste)

| K | k+1 | k+2 | k+3 |
|---|---|---|---|
| Octet 0 | Octet 1 | Octet 2 | Octet 3 |

**Exemple 2.1** : La valeur 32 bits 0x01234567 est stockée à l'emplacement 0x100. Donnez le contenu de la mémoire à partir de cette adresse si le système est de type big endian ? et s'il est de type little endian ?

| | 0x100 | 0x101 | 0x102 | 0x103 |
|---|---|---|---|---|
| Big Endian | 01 | 23 | 45 | 67 |

| | 0x100 | 0x101 | 0x102 | 0x103 |
|---|---|---|---|---|
| Little Endian | 67 | 45 | 23 | 01 |

### 2.5.2.6 Notion d'adresse mémoire

La mémoire est constitué d'un ensemble de cellules possédant chacune une adresse numérotés de 0 à n-1. Les adresses sont exprimées en binaire et par convention commencent par 0.

Considérons à titre d'exemple deux organisations possibles d'une mémoire de 96 bits.

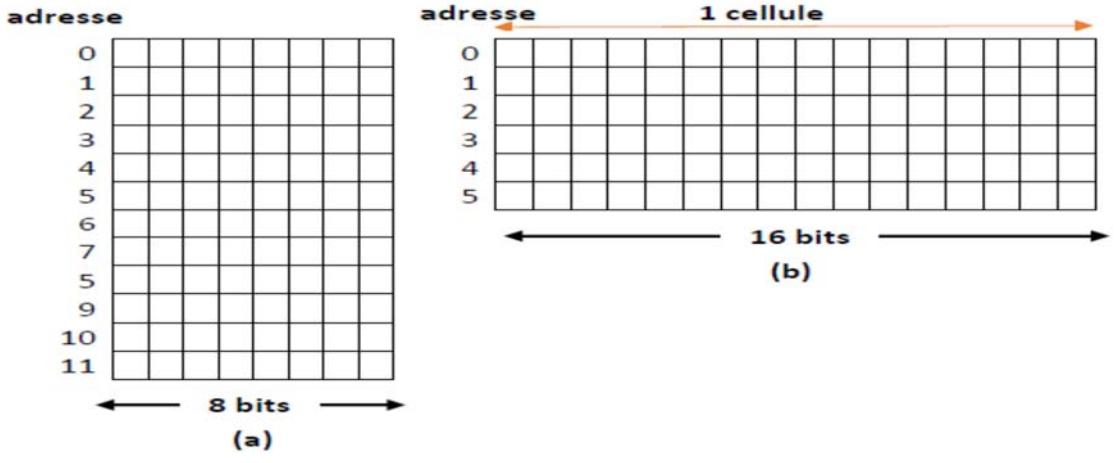



*Figure 2.14. Exemple d'adresse mémoire*

La mémoire de la figure 2.14 (a) nécessite au moins 4 bits pour exprimer les adresses des 12 cellules (de 0 à 11), alors qu'une adresse de 3 bits est suffisante pour la configuration de la figure 2.14 (b).

On peut en déduire que le nombre de bits d'adresse détermine la taille de la mémoire centrale en cellules adressables qui est indépendante de la taille de la cellule.

On conclusion :

Si n = nombre de bits d'adresse Alors la taille de la MC = $2^n$ unités adressables (UA)

Si la taille de la MC = $2^n$ UA Alors la taille du registre RA = n = $log_2 2^n$

**Exemple** 2.2:

Soit une mémoire centrale de 2 G mots de 16 bits. Sachant que l'unité adressable est de 1 octet, quelle est la taille du registre d'adresse (RA) ?

Il faut tout d'abord exprimé la taille de la MC en unités adressables.

Taille de MC = 2G mots de 16 bits = $2*2^{30}*2$ octets = $2^{32}$ octets = $2^{32}$ unités adressables

Alors la taille du registre d'adresse = 32 bits.

### 2.5.2.7 Typologie des mémoires

Il existe deux types de mémoires dans l'ordinateur :

#### 2.5.2.7.1 Les mémoires vives (RAM)

RAM (Random Access Memory, mémoire à accès aléatoire). Il s'agit de la mémoire la plus classique de l'ordinateur, appelée aussi « mémoire vive ». Elle constitue la grande majorité de la mémoire principale et elle est composée de cellules mémoire standard volatiles. La RAM sert à mémoriser tous les programmes (instructions) et les données associées.

Il existe deux grandes familles de mémoires RAM :

- Les RAM statiques
- Les RAM dynamiques

##### 2.5.2.7.1.1 Les RAM statiques (SRAM)

Chaque bit d'une SRAM est formé par une bascule constituée par 4 à 6 transistors. L'information stockée peut être maintenue sans dégradation pendant une centaine d'heures. L'intérêt de ce type de mémoire est sa vitesse (quelques ns) mais son cout est exorbitant. En effet la SRAM est utilisée lorsque le facteur vitesse est important et principalement pour des mémoires de petite taille comme la mémoire cache.

##### 2.5.2.7.1.2 Les RAM dynamiques (DRAM)

Chaque bit d'une DRAM est constitué par un transistor et un condensateur. L'inconvénient des DRAM est que le condensateur a tendance naturelle à se décharger. Pour que l'information reste cohérente, on doit réaliser un rafraîchissement de la mémoire toutes les quelques millisecondes. Ce rafraîchissement consiste à lire et à réécrire la donnée.

Voici quelques DRAM qui ont ou sont utilisées dans les PC : DRAM FPM (Fast Page Mode, 1987), DRAM EDO (Extended Data Out, 1995), DRAM BEDO (Bursted EDO), Synchronous DRAM (SDRAM, 1997), DR-SDRAM (Rambus DRAM), DDR-I ou DDR-SDRAM (Double Data Rate Synchronous DRAM, 2000), DDR2-SDRAM (Double Data Rate two, 2003), DDR3-SDRAM (Double Data Rate 3rd generation, 2007), etc.

#### 2.5.2.7.1.3 Conclusion

Etant donné les caractéristiques des SRAM et DRAM, on peut en déduire que la SRAM est rapide mais chère alors que la DRAM est lente mais bon marché. En général les DRAMs, qui offrent un coût par bit plus faible et une plus grande densité d'information sont utilisées pour la mémoire centrale, alors que les mémoires statiques, plus rapides, sont utilisées pour des mémoires de petite taille comme les caches et les registres où le facteur vitesse est important.

### 2.5.2.7.2 Les mémoires mortes (ROM)

Pour certaines applications, il est nécessaire de pouvoir conserver des informations de façon permanente même lorsque l'alimentation électrique est interrompue. On utilise alors des mémoires mortes ou mémoires à lecture seule (ROM : Read Only Memory). Ces mémoires 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 :

- **ROM** : Elle est programmée par le fabricant et son contenu ne peut plus être ni modifié. ni effacé par l'utilisateur
- **PROM (Programmable ROM) :** Permet une écriture unique mais faite par l'utilisateur au moyen d'une machine spéciale ; ces mémoires comprennent un ensemble de fusibles qui sont détruits lors de la programmation de la mémoire.
- **EPROM (Erasable PROM). PROM effaçable :** Elles peuvent être effacées une fois écrites au moyen d'un faisceau ultraviolet qui force tous les bits à une même valeur. Elles ont la même organisation que les SRAM.
- **EEPROM (Electrically EPROM) :** Elles sont effaçables électriquement sans intervention d'un rayonnement ultraviolet. Elles ne sont pas volatiles, facilement réutilisables.

### 2.5.2.7.3 Usage de ROM et RAM

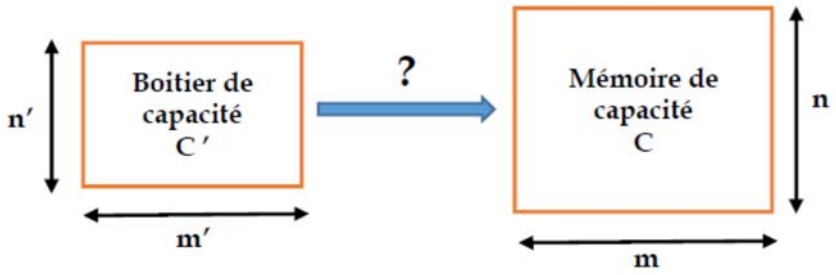
La RAM est la mémoire de travail de l'ordinateur : c'est là où sont stockées les instructions et les données, chaque cellule mémoire pouvant être modifiée des milliards de fois par seconde sans aucun problème.

Les ROM et PROM ne peuvent servir qu'à mémoriser de façon permanente des informations figées. Voici quelques exemples d'utilisation de ces mémoires au sein de l'ordinateur :

- Un séquenceur microprogrammé traite chaque instruction en exécutant des microinstructions qui effectuent chacune un travail élémentaire dans le processeur. Chaque instruction correspond donc à un microprogramme mémorisé dans une mémoire du processeur réalisée lors de sa fabrication. Ces microprogrammes ne doivent évidemment pas disparaître et n'ont pas à être changés ; c'est un cas typique d'utilisation de ROM.
- Lorsque l'ordinateur s'allume, le processeur cherche à exécuter des instructions. Malheureusement, la RAM étant volatile, elle ne contient aucune information viable lors de l'allumage. Là encore, une partie de la mémoire principale est sous forme de mémoire morte contenant un programme de démarrage qui charge depuis un disque dur le reste du système d'exploitation.
- Certaines entrées/sorties sont suffisamment complexes pour nécessiter la présence d'un processeur auxiliaire qui leur est uniquement dédié. Il ne s'agit pas, dans ce cas, d'un programme utilisateur, mais d'instructions réalisant les entrées/sorties. Le fabricant les programme et les stocke sur une mémoire morte pour qu'elles ne disparaissent pas.

### 2.5.2.8 Conception des MC

Supposons qu'on veut réaliser une mémoire de capacité C, mais nous disposons uniquement de boîtiers de taille ou capacité (C') inférieure ?




**Solution**

Soient :

- Soit M une mémoire de capacité C , tel que n est le nombre de mot et m la taille d'un mot.
- Soit M' un boîtier de capacité C' , tel que n' le nombre de mot et m' la taille d'un mot.

  On suppose que C > C' (m >= m', n >=n').

Question : Quel est le nombre de boîtiers M' nécessaire pour réaliser la mémoire M ?

Pour connaître ce nombre, il faut calculer les deux facteurs :

P = n/n' et Q = m/m'

- P : détermine de nombre de boîtiers M' nécessaire pour obtenir le nombre de mots de la mémoire M (extension lignes).
- Q : détermine le nombre de boîtier M' nécessaire pour obtenir la taille de mot de la mémoire M (extension colonnes).
- P x Q donne le nombre total de boîtiers M' nécessaire pour réaliser la mémoire M.

Pour sélectionner les boîtiers on utilise les bits de poids forts d'adresses. Si P est le facteur d'extension lignes alors on prend k bits tel que $P=2^k$. Les autres bits d'adresses restants sont utilisés pour sélectionner un mot dans un boîtier.

**Exemple 2.3 :**

Réaliser une mémoire de 1Ko (la taille d'un mot est de 8 bits) en utilisant des boîtiers de taille 256 mots de 8 bits ?

**Solution** :

(n, m)=(1024, 8) → taille du bus d'adresses est de 10 bits $A_0^9$ $(A_9, \dots, A_0)$, taille du bus de données est de 8 bits $D_0^7$ $(D_7, \dots, D_0)$.

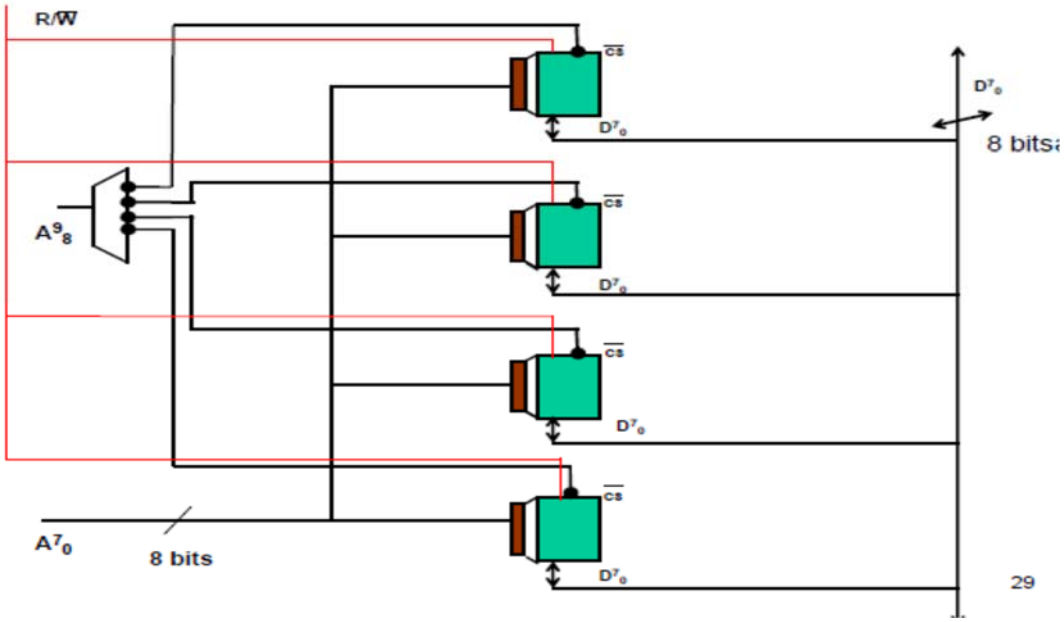
(n', m')=(256, 8) → taille du bus d'adresses est de 8 bits $A'^7_0$ $(A'_7, \dots, A'_0)$, taille du bus de données est de 8 bits $D'^7_0$ $(D'_7, \dots, D'_0)$.

Calculer les deux facteurs d'extension lignes et colonnes :

P = n/n' = 1024/256 = 4 (extension lignes)

Q= m/m' = 8/8 = 1 (extension colonnes)

Le nombre total de boîtiers P x Q = 4




## 2.5.3 Mémoire cache

### 2.5.3.1 Introduction

Les performances (vitesse) des processeurs augmentent sans cesse. Les performances des mémoires augmentent aussi mais surtout en matière de capacité de stockage et beaucoup moins en ce qui concerne les temps d'accès. La fréquence d'un processeur est aujourd'hui de l'ordre de 1 GHz (1 000 MHz) ce qui signifie qu'il peut engendrer des actions $10^9$ fois par seconde. Une mémoire DRAM dont le temps d'accès est de 20 nanosecondes pourra fournir $5 \times 10^7$ informations par seconde. On peut conclure que l'écart est donc considérable entre les deux parties et le processeur ne fonctionne pas au meilleur rythme. Les mémoires SRAM sont plus rapides que les DRAM et elles ont une densité d'intégration trop faible et un coût trop élevé pour constituer des mémoires centrales de grandes capacités.

Pour améliorer les performances, les constructeurs ont intercalé une mémoire rapide, dite «cache» (ou, parfois, appelée « antémémoire ») entre le processeur et la mémoire principale. Elle fournit à celui-ci les données et instructions les plus récentes ou les plus souvent utilisées par le processeur et qui sont nécessaires à l'exécution des programmes, en opérant plus rapidement que la mémoire principale car elle utilise la technologie de mémoire statique (SRAM). En contrepartie, sa taille est limitée et elle ne peut constituer l'intégralité de la mémoire principale, ce qui résoudrait le problème de la différence de vitesse. Au départ cette mémoire était intégrée en dehors du processeur mais elle fait maintenant partie intégrante du processeur et se décline même sur plusieurs niveaux.

### 2.5.3.2 Principe de fonctionnement et généralités

La mémoire cache est une petite mémoire de taille limitée qui est toujours beaucoup plus rapide que la mémoire principale utilisée comme mémoire intermédiaire (tampon) entre le processeur et la mémoire centrale. Son rôle est d'améliorer les performances du système, en réduisant les temps d'accès aux instructions et aux données dont a besoin le processeur dans l'immédiat. Son principe de fonctionnement est le suivant :

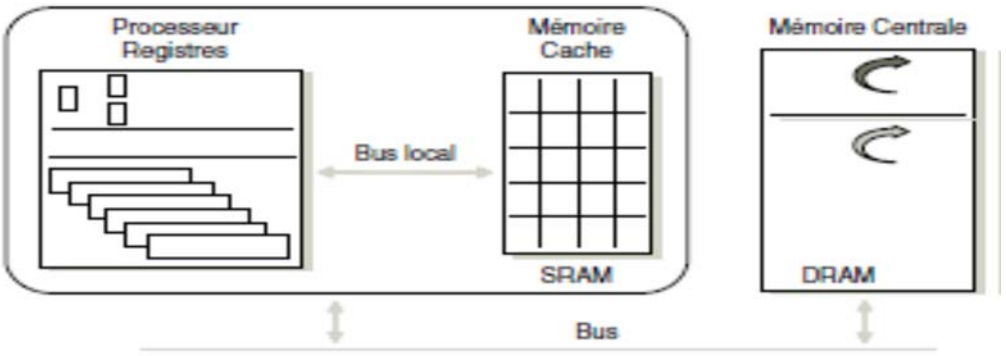



*Figure 2.15. Position du cache dans l'unité centrale*

Le principe est de faire coopérer des mémoires de faible capacité, très rapides et à proximité du processeur avec des mémoires plus lentes et de grandes capacités. Les mots de mémoire centrale les plus fréquemment utilisés sont placés dans le cache. À chaque accès mémoire effectuant une lecture, l'algorithme suivant est exécuté :

- Le microprocesseur fournit une adresse simultanément à la mémoire cache et à la mémoire centrale.
- Soit l'information (donnée ou l'instruction) recherché est présente dans la mémoire cache, le microprocesseur la récupère et poursuit l'exécution. On parle de succès de cache (*cache-hit*). (a)
- Soit l'information n'est pas dans la mémoire cache, alors l'information est recherchée en mémoire centrale puis délivré simultanément au microprocesseur et à la mémoire cache. On parle de défaut de cache (*cache-miss*). (b)

Bien sûr, la mémoire cache n'apporte un gain de performance que dans le premier cas. Sa performance est donc entièrement liée à son taux de succès.

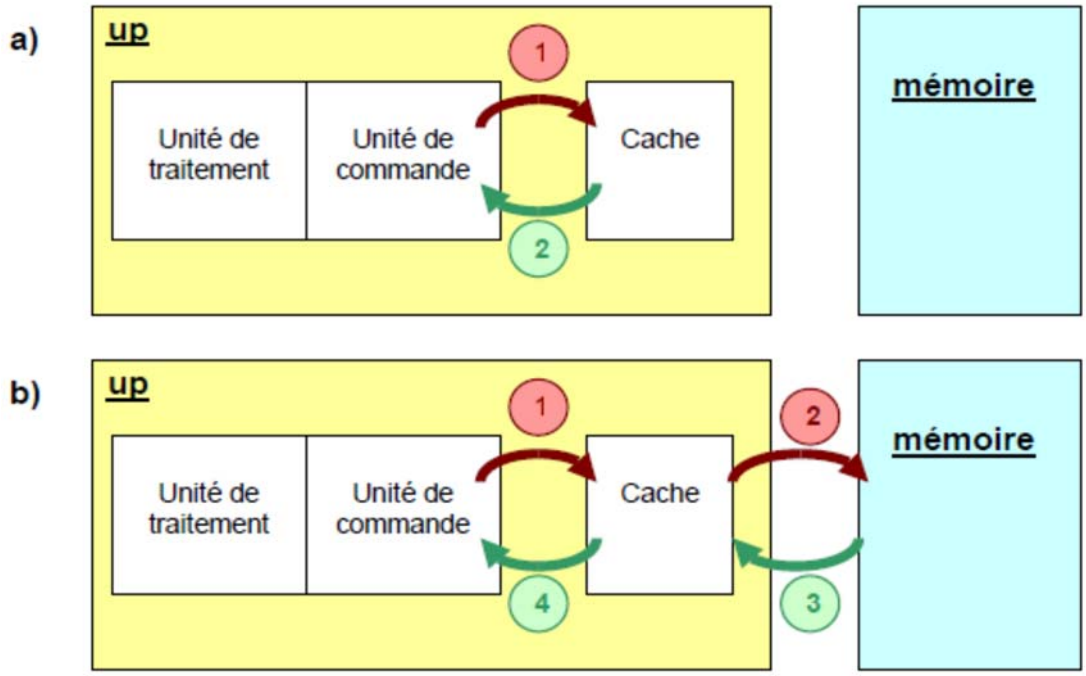



*Figure 2.16. Principe de fonctionnement du cache*

**Remarque** :

- Un cache utilisera une carte pour savoir quels sont les mots de la mémoire principale dont il possède une copie. Cette carte devra avoir une structure simple.
- Il existe dans le système deux copies de la même information : l'originale dans la mémoire principale et la copie dans le cache. Si le microprocesseur modifie la donnée présente dans le cache, il faudra prévoir une mise à jour de la mémoire principale.
- Lorsque le cache doit stocker une donnée, il est amené à en effacer une autre. Il existe donc un contrôleur permettant de savoir quand les données ont été utilisées pour la dernière fois. La plus ancienne non utilisée est alors remplacée par la nouvelle.

Les mémoires caches contiennent généralement un tableau d'étiquettes qui contient les adresses des données contenues dans le cache et un tableau de données qui contient les données elles-mêmes. La mémoire principale contient jusqu'à $2^n$ mots adressables, chacun possédant une adresse unique de n bits. Pour la correspondance avec la mémoire cache, cette mémoire se compose de blocs d'une longueur fixe de K mots chacun. Autrement dit, il y a $M = 2^n/K$ blocs. Le cache se divise en L lignes de K mots.

La figure ci-dessous illustre la structure du système cache/mémoire principale

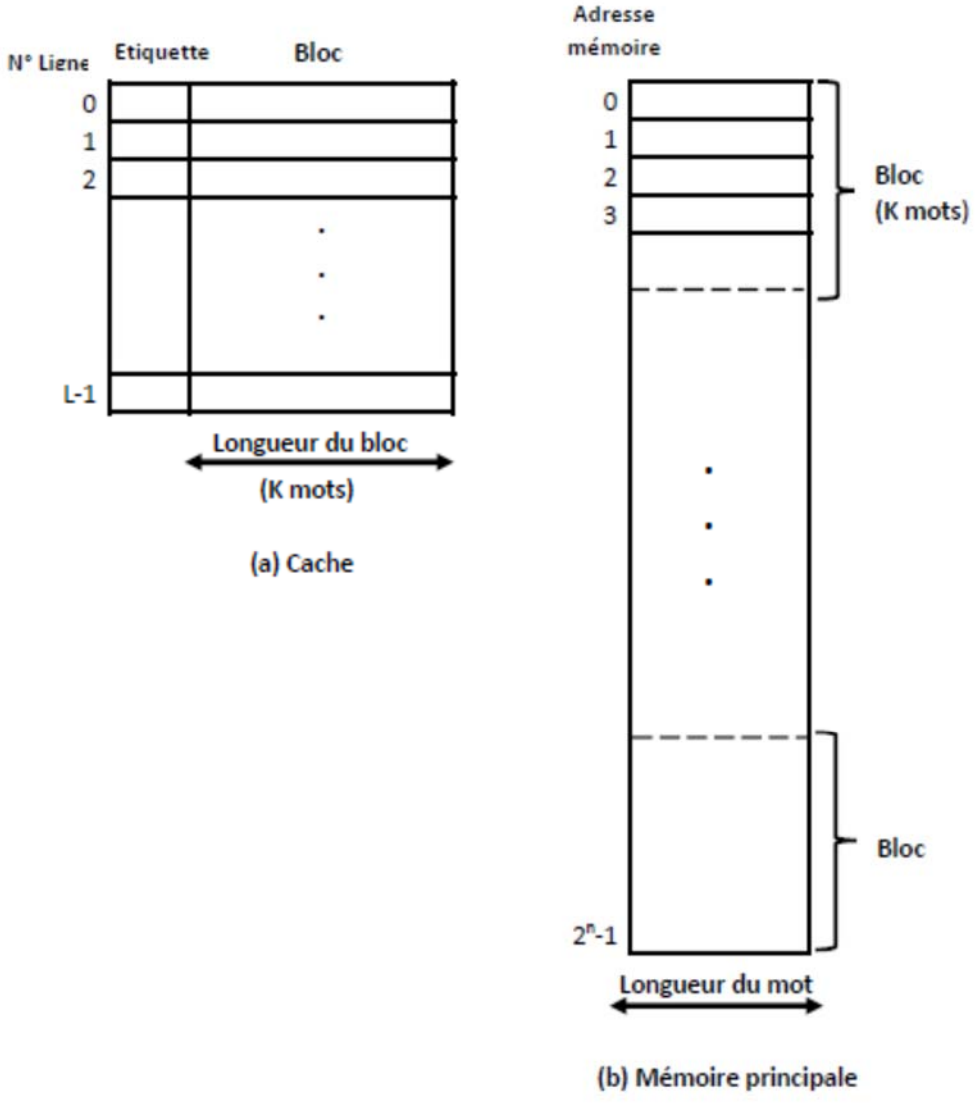



*Figure 2.17. Structure du système cache/mémoire principale*

Si les mémoires cache permettent d'accroître les performances, c'est en partie grâce à deux principes qui ont été découverts à la suite d'études sur le comportement des programmes informatiques :

- ***le principe de localité spatiale*** qui indique que l'accès à une donnée située à une adresse X va probablement être suivi d'un accès à une zone très proche de X. C'est évidemment vrai dans le cas d'instructions exécutées en séquence, et encore pour les boucles courtes.
- ***le principe de 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 immédiate du programme. C'est évidemment vrai dans le cas des boucles de quelques instructions seulement.

### 2.5.3.3 Méthodes d'accès aux caches

D'habitude le CPU utilise une adresse pour accéder à une information en MC, le principe est le même pour une mémoire cache sauf que la correspondance entre adresse de l'information en MC et celle en mémoire cache doit être calculée. Il existe trois techniques de correspondance : associative, directe et associative par ensemble.

#### a-) Caches associatifs

Dans le cas d'une ***cache associatif***, les informations sont adressables par le contenu, chaque case d'une mémoire associative comprend deux champs : Le champ *clé* et le champ *information*.

- Le champ *clé* contient l'adresse en M.C. de l'instruction ou de la donnée.
- Le champ *information* contient l'instruction ou la donnée elle-même.

L'information est stockée dans n'importe quelle ligne de la cache. Lorsque le CPU fournit une adresse, celle-ci est comparée à chaque entrée du cache (clé) pour voir si l'information est disponible ou non. C'est le plus général et le plus simple à comprendre, mais aussi le plus coûteux à mettre en œuvre. Une adresse arbitraire est donc divisée en deux parties : la première indique le numéro de bloc (étiquette) et la seconde indique le numéro d'octet à l'intérieur du bloc (offset).

**Exemple** 2.4 : Pour une taille de bloc de 16 octets et une adresse de 32 bits, la première partie de l'adresse contient donc 28 bits et la deuxième 4 bits. Car 16 octets= $2^4$ donc 4 bit sont nécessaires pour numéroter l'octet, et le reste 28.

#### b-) Caches directs

Dans un cache direct, chaque adresse mémoire ne peut être stockée qu'à un seul emplacement du cache. Etant divisés en bloc (ou en pages), chaque mot d'adresse *i* de chaque bloc en mémoire centrale, ne peut être chargé qu'à l'adresse *j* en mémoire cache. Lorsqu'une opération mémoire est transmise à un cache direct, un premier sous-groupe de bits dans l'adresse est utilisé pour sélectionner la ligne du cache qui peut contenir l'adresse et un second sous-groupe est utilisé pour sélectionner l'octet dans la ligne de cache que pointe l'adresse.

Les n bits de poids faible dans l'adresse sont utilisés pour déterminer la position de l'adresse dans la ligne de cache, où n = $\log_2$ du nombre d'octets dans la ligne. Les m bits suivants de poids supérieur (où m = $\log_2$ du nombre de lignes dans le cache) sont utilisés pour sélectionner la ligne dans laquelle l'adresse peut être stockée.

**Exemple 2.5 :** Dans un cache direct d'une capacité de 16 Ko dont les lignes font 32 octets, combien de bits sont utilisés pour déterminer l'octet qu'une opération mémoire référence dans une ligne de cache et combien de bits sont utilisés pour sélectionner la ligne dans le cache qui pourrait contenir les données ?

**Solution :** $\log_2 32 = 5 \rightarrow 5$ bits sont requis pour déterminer l'octet référencé dans la ligne de cache.
Nombre de lignes=capacité du cache/longueur de la ligne=16 Ko/ 32 = $2^{14}$ / $2^5$ =$2^9$
Alors 9 bits sont requis pour sélectionner la ligne $\log_2 29 = 9$ .

**c-) Caches associatifs par ensemble**

Afin d'éviter le problème du cache direct sans toutefois être pénalisé par le coût élevé du cache associatif, on peut imaginer des solutions mixtes. La mémoire est toujours divisée en blocs et le cache en lignes. Les lignes du cache sont organisées de manière séquentielle comme pour le cache direct. La différence par rapport au cache direct est que chaque ligne peut contenir un nombre de blocs supérieur à 1. En fait, chaque ligne du cache est un mini-cache associatif. Le nombre de blocs de chaque ligne est appelé l'*associativité* du cache.

**Exemple 2.6** : Combien d'ensembles possèdent un cache associatif par ensemble de 2 blocs d'une capacité de 32 Ko dont les lignes font 64 octets ? Combien de bits d'adresse sont utilisés pour sélectionner un ensemble de ce cache ?

**Solution** : nbre lignes=32 Ko / 64 octets = 512 lignes de données.
1 ensemble contient 2 lignes alors le cache contient 256 ensembles.
$\log_2 256 = 8$ alors 8 bits sont utilisés pour sélectionner l'ensemble vers lequel renvoie l'adresse

### 2.5.3.4 Caches de données et caches d'instructions

Dans le cas des caches, les données et les instructions sont souvent stockées dans des caches séparés, contrairement à la mémoire centrale (selon l'architecture de Von Neumann). Cette organisation, appelée cache Harvard est utilisée parce qu'elle permet au processeur d'extraire simultanément les instructions du cache d'instructions et les données du cache des données. La séparation des caches d'instructions et de données évite les conflits entre les blocs d'instructions et les données qui pourraient être renvoyées aux mêmes emplacements mémoire dans le cas d'un cache unifié.

Lorsqu'un cache contient à la fois des instructions et des données, on parle de cache unifié.

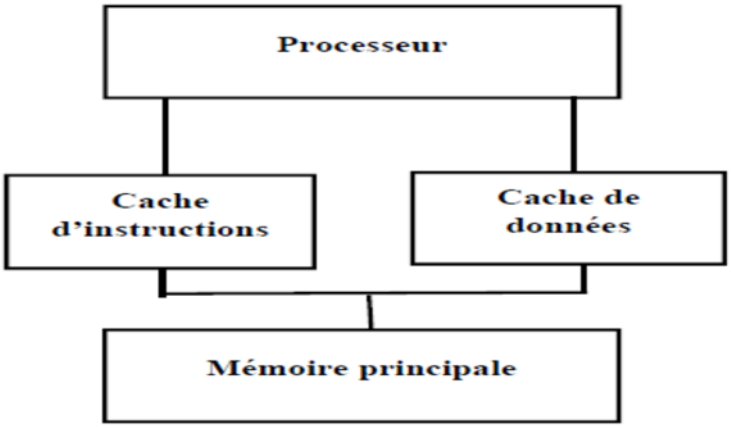



*Figure 2.18. Architecture de cache Harvard*

### 2.5.3.5 Caractéristiques des mémoires caches

Pour pouvoir faire des comparaisons entre les caches, les concepteurs considèrent certaines caractéristiques, tel que : leur capacité, leur longueur de ligne, leur associativité (le nombre d'emplacements qu'une adresse donnée peut occuper), la politique de remplacement et le type du cache.

**Capacité :** La capacité d'un cache est la quantité d'information (données ou instructions) qui peut être stockée dans la mémoire cache. Exemple, une mémoire cache d'une capacité de 32 Ko peut stocker 32 Ko de données ou d'instructions. En réalité, un cache de ce type nécessitera plus de 32 Ko de mémoire, car l'espace de stockage du tableau d'étiquette n'est pas compté dans la capacité.

**Longueur de ligne (ou taille de ligne) :** La longueur de ligne d'un cache correspond à la taille de bloc du cache. Autrement dit c'est le nombre d'octets qui peut être copié à la fois de la mémoire centrale vers la mémoire cache.

#### 2.5.3.6 Techniques ou politiques de remplacement dans un cache

Le problème posé est le suivant :

Quand il y'a échec de lecture (cache-miss) en mémoire cache, le processeur est obligé de chercher l'information en M.C. et de la charger en mémoire cache, cependant le processeur doit alors trouver un emplacement en mémoire cache, et étant donné que ce dernier est plein, il doit choisir quelle ligne de cache il doit remplacer.

La politique de remplacement doit tenir compte de deux facteurs :

- La *longueur de ligne* (taille du bloc à charger).
- Le *choix des lignes* de cache à remplacer.

Nous allons donner ici trois politiques de remplacement :

1. *Le remplacement aléatoire* : elle consiste à remplacer les lignes de manière *aléatoire*, ce qui signifie qu'il n'y a aucun choix sur les lignes à évincer du cache. L'avantage de cette politique est qu'elle est plus simple à réaliser. L'inconvénient est le fait qu'elle présente un risque élevé de voir remplacer des lignes, auxquelles le processeur fera appelle dans la suite de l'exécution du programme.
2. *Le remplacement de la ligne la moins récemment utilisée (Leat Recenrly Used ou LRU)*. Dans le cas de la politique LRU, le cache trie chacune des lignes afin de définir celles qui ont été accédées le plus récemment. La ligne la moins récemment utilisée sera évincée de la cache.
3. *Le remplacement direct* : Cette politique est utilisée pour les caches directs, aucun choix n'est offert, puisque la ligne entrante à un seul endroit dans le cache.
4. *Le remplacement de la ligne la moins fréquemment utilisée* (*Leat Frequently Used ou LFU).*

#### 2.5.3.7 Ecriture en mémoire cache

On veut dire par « écriture » la modification des informations du cache par le processeur.

Dans le cas d'une lecture ou d'un remplacement des lignes de caches, informations restent intègres.

En revanche dans le cas où le processeur écrit sur le cache (modification de données), l'information est modifiée et quand la ligne en question est remplacée, les modifications seront perdues. Alors éviter de perdre les modifications, les lignes concernées doivent être copies en MC, pour cela deux techniques sont utilisées :

- Ecriture immédiate (Write-Through) : La technique Write-Through consiste à écrire simultanément dans le cache et la MC.
- Ecriture différée (Write-Back) : La technique Write-Back consiste à conserver des données modifiées et de ne faire des mises à jour en MC que lorsque la ligne qui subit une modification est évincée du cache.

#### 2.5.3.8 Les niveaux de caches

Dans les ordinateurs actuels, il peut y avoir plusieurs mémoires caches, on parle alors de niveau de caches.

- **La mémoire cache de premier niveau** (appelée L1 Cache, pour Level 1 Cache) est qui est le plus proche du processeur et dans la majorité des cas, le cache est directement intégrée dans le processeur, très rapide d'accès mais petite en capacité de stockage (16 à 64 Ko). Elle se subdivise en 2 parties :
    - La première est le cache d'instructions, qui contient les instructions issues de la mémoire vive décodées lors de passage dans les pipelines.

- La seconde est le cache de données, qui contient des données issues de la mémoire vive et les données récemment utilisées lors des opérations du processeur.

- **La mémoire cache de second niveau** (appelée L2 Cache, pour Level 2 Cache) C'est une extension du cache L1, dont la capacité est plus importante que la L1 (avoisine les 1 Mo) et qui peut être intégrée soit au processeur, soit sur la carte-mère avec une liaison haute vitesse par bus informatique vers le processeur central.
- **La mémoire cache de troisième niveau** (appelée L3 Cache, pour Level 3 Cache) précédemment située au niveau de la carte mère a récemment aussi été intégrée dans la puce, et elle est moins rapide que les précédentes mais dotée d'une capacité plus élevée. En général, elle sert à améliorer les performances des L1 et L2, notamment dans les processeurs à architecture multicœur.

Tous ces niveaux de cache permettent de réduire les temps de latence des différentes mémoires lors du traitement et du transfert des informations.

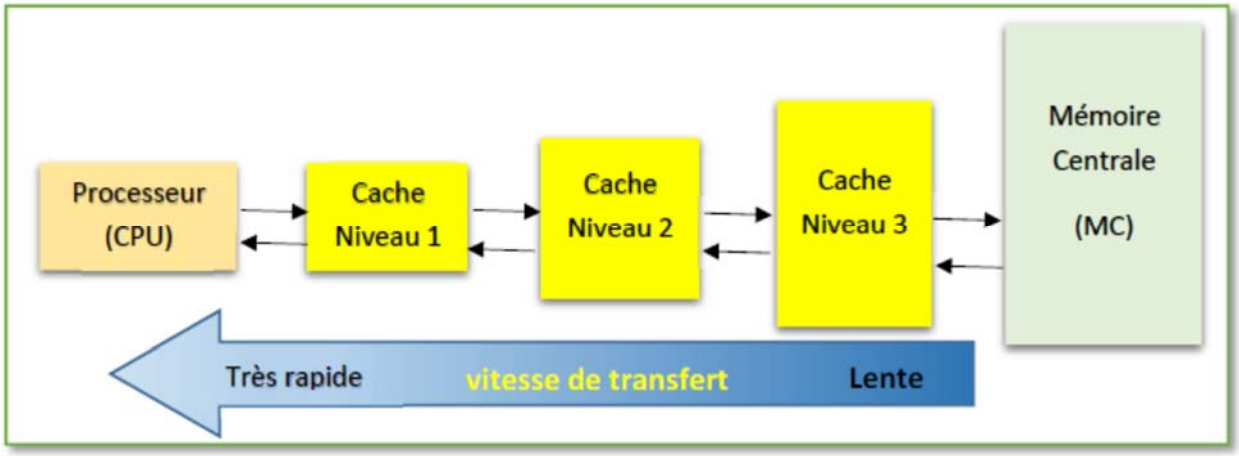



*Figure 2.19. Niveau de caches dans un ordinateur*

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

## 3.1 Langage de haut niveau, assembleur, langage machine

Un programme est une suite d'instructions élémentaires, qui vont être exécutées dans l'ordre par le processeur. Les instructions correspondent à des actions très simples, comme additionner deux nombres, lire ou écrire une case mémoire, etc.

La programmation est l'activité qui consiste à traduire un algorithme en un programme dans un langage assimilable par l'ordinateur et qui peut s'effectuer à différents niveaux, plus ou moins proches et dépendants de l'architecture physique de la machine. On distinguera trois niveaux (figure 3.1) : la programmation de bas niveau en langage machine, la programmation de bas niveau en langage d'assemblage, la programmation de haut niveau à l'aide d'un langage de haut niveau ou langage évolué.

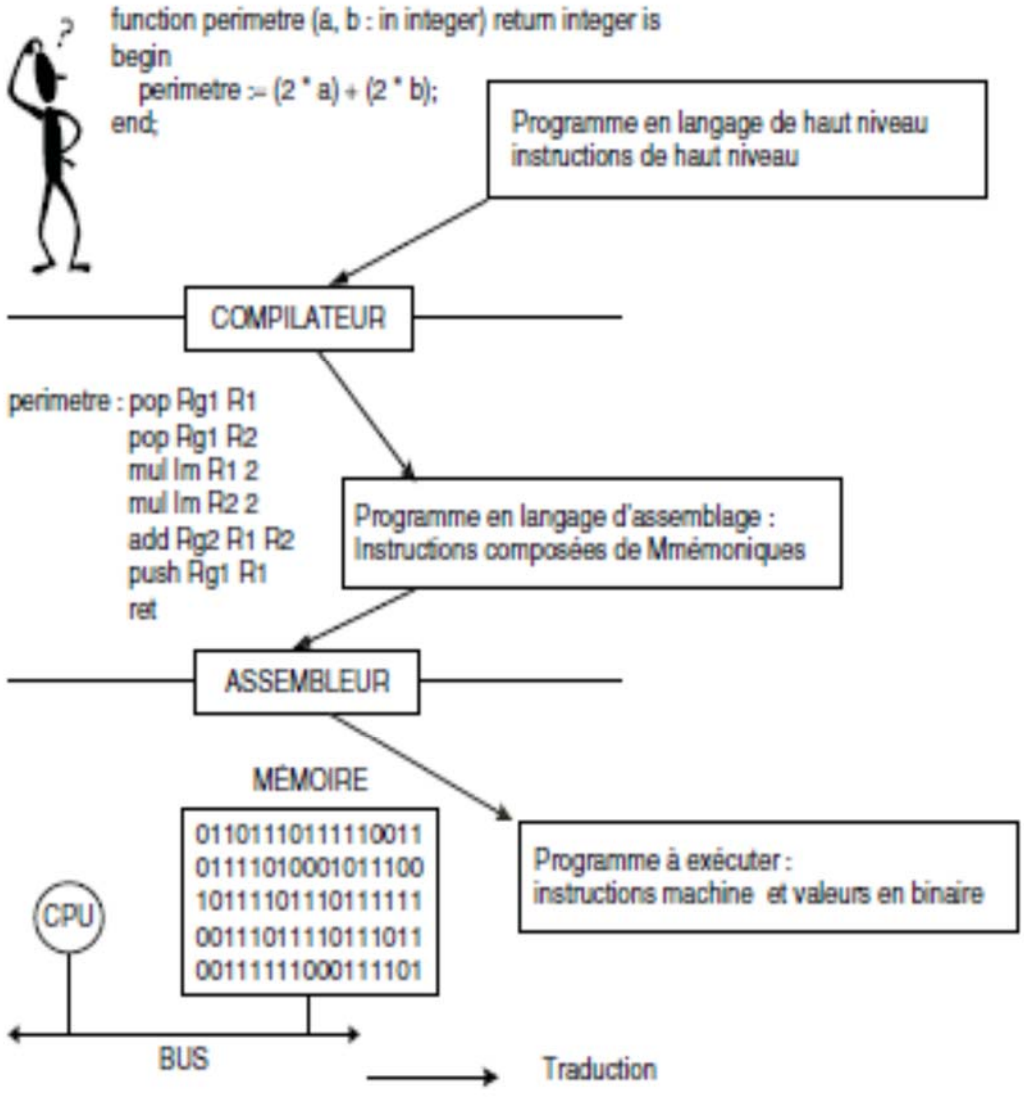



***Figure 3.1*** *Les différents niveaux de programmation.*

Un **langage machine** est un langage binaire ou l'information est exprimé et manipulé sous forme d'une suite de bits Chaque type de processeur est capable d'exécuter un certain ensemble d'instructions, son jeu d'instructions. Le langage machine est très difficile à comprendre par l'humain.

Un **langage Assembleur** c'est le premier langage exprimant les instructions élémentaires de façon symbolique. Exemple : ADD a, 100 (Addition).
Il dépend du quitte de la machine, mais ne sont pas assez efficace pour développer des applications complexes d'où l'apparition des langages évolués.

Un **Langage évolué** est proche du langage humain (Anglais). Grande portabilité (indépendant de la machine). Manipulation des données complexes. Nécessite un traducteur (Compilateur / interpréteur).

## 3.2 Principe de compilation d'assemblage et d'éditeur de liens

La première étape de la chaîne de production de programmes est la *compilation*. La compilation permet la traduction d'un programme dit *programme source* écrit le plus souvent en langage évolué (de haut niveau) vers un programme dit *programme objet* qui est soit directement le langage machine, soit le langage d'assemblage. Le programme objet est stocké sur le disque.

Le compilateur est une application ou un logiciel dépendant de la machine physique vers laquelle il doit produire le langage. Ainsi un programme compilé sur une machine X ne s'exécutera pas forcément sur une machine Y, essentiellement si les deux machines sont différentes physiquement.

Le travail du compilateur se divise en plusieurs phases :

- l'analyse lexicale (reconnaissance des mots du langage, c.-à-d appréhension du vocabulaire) ;
- l'analyse syntaxique (vérification de la syntaxe, c.-à-d. appréhension de la grammaire) ;
- l'analyse sémantique (vérification de la sémantique, c.-à-d. appréhension du sens) ;
- l'optimisation et la génération du code objet.

Le passage du langage d'assemblage au langage machine se fait par l'intermédiaire d'un autre traducteur, l'*assembleur*.

L'*éditeur de liens* est un logiciel qui permet de combiner plusieurs programmes objet en un seul et de résoudre des appels à des modules de librairie. L'éditeur de liens résout les références externes.

## 3.3 Structure d'une instruction machine

Un programme est une suite d'instruction écrites dans un langage de programmation est traduites en langage machine pour qu'elles puissent être traitées par le *processeur*, c'est pour cela qu'on parle *d'instruction machine*. Une instruction machine comporte essentiellement :

- Un champ code opération (obligatoire) qui indique au processeur quelle instruction réaliser ;
- Un ou plusieurs champs d'adresse.

Le format d'une instruction correspond au nombre de champs de l'instruction réservé à ces adresses. Ce format diffère d'une machine à une autre on emploie souvent l'expression Machine à n adresses pour indiquer ce format, avec n (= 0, 1, 2, 3, 4).

a. **Machine à 4 adresses :** Ce format a été utilisé dans les premiers ordinateurs.

| Code Opération | @1er Opérande | @2e Opérande | @résultat | @Inst. Suivante |
|---|---|---|---|---|
| Cod-Op | A1 | A2 | A3 | A4 |

b. **Machine à 3 adresses :** Le champ instruction suivante a été supprimé ; c'est le compteur ordinal (CO) qui sauvegarde cette adresse.

c. **Machine à 2 adresses :** Le champ adresse du résultat a été supprimé et sera placé à la place de celle du 2e opérande.

d. **Machine à 1 adresse :** l'instruction ne contient qu'une seule adresse c'est l'adresse de premier opérande, le deuxième opérande se trouve dans l'accumulateur.

e. **Machine à 0 adresse (à pile**) : Ce type de machine utilise une pile, elle est gérée en utilisant deux instructions à une adresse :
PUSH X : permet de placer X au sommet de la pile (empiler)
POP X : permet de stocker la valeur qui se trouve au sommet de la pile en X (dépiler).

## 3.4 Jeu d'instructions et modèles de programmation

### 3.4.1 Jeu d'instruction

Le terme jeu d'instruction (Instruction Set) est l'ensemble des instructions de base qu'un processeur est capable d'exécuter. Chaque type de processeur possède son propre jeu d'instructions, selon que le nombre d'instructions dans le jeu est grand ou réduit, on distingue deux familles :

- les CPU à architecture CISC (Complex Instruction Set Computer), qui signifie processeurs à jeu d'instructions complexe. Il s'agit d'un plan de conception du CPU basé sur des instructions uniques, capables d'exécuter des opérations en plusieurs étapes.
- les CPU à architecture RISC (Reduced Instruction Set Computer), qui signifie processeurs à nombre d'instructions réduits. Il s'agit d'un plan de conception de CPU basé sur des instructions simples et rapide.

Les instructions que l'on retrouve dans chaque microprocesseur peuvent être classées en 4 catégories :

**1-** Les instructions de transfert de données :
- Entre la mémoire et les registres (ex. LD, ST).
  LD rX,(rY) ; rX ← Mem[rY]
- Entre registre et registre (ex. MOV)
  MOV rX,rY ; rX ← rY

**2-** Les instructions de traitement de données :
- Opérations arithmétiques : ADD, SUB, DIV, MUL, …
  ADD rX,rY,rZ ; rX ← rY + rZ
  ADD rX,rY,#i ; rX ← rY + valeur i
- Opérations logiques : AND, OR, NOT, XOR, …
  AND rX,rY,rZ ; rX ← rY ET rZ
  AND rX,rY,#i ; rX ← rY ET valeur i

**3-** Les instructions de contrôle de séquence **:**
- Branchements conditionnels : JZ, JNZ, JGT, JLT, JGE, JLE.
  JZ rX,Adr ; PC ← *Adr* si $rX = 0$ : Saut conditionnel à l'adr. *Adr* si le registre $rX = 0$.
  JNZ rX,Adr ; PC ← *Adr* si $rX \neq 0$ : Saut conditionnel à l'adr. *Adr* si le registre $rX \neq 0$.
  JGT rX,Adr ; PC ← *Adr* si $rX > 0$ : Saut conditionnel à l'adr. *Adr* si le registre $rX > 0$.
  JLT rX,Adr ; PC ← *Adr* si $rX < 0$ : Saut conditionnel à l'adr. *Adr* si le registre $rX < 0$.
  JGE rX,Adr ; PC ← *Adr* si $rX \geq 0$ ; Saut conditionnel à l'adr. *Adr* si le registre $rX \geq 0$.
  JLE rX,Adr ; PC ← *Adr* si $rX \leq 0$ ; Saut conditionnel à l'adr. *Adr* si le registre $rX \leq 0$.

- Branchements inconditionnels ou saut : JMP
  JMP Adr ; PC ← Adr : Saut inconditionnel à l'adresse Adr.
- Appel de procédure, …

**4-** Les instructions d'entrée/sorties.

### 3.4.2 Modèles de programmation

Sur la plupart des machines, on retrouve deux modèles de programmations :

a) **Les architectures à registres généraux** : les instructions lisent leurs opérandes et écrivent leurs résultats dans des registres. L'accès aux registres est aléatoire et chaque registre possède un identificateur. L'ensemble de registres est appelé fichier de registre généraux

**Exemple** : Ecrire un programme pour un processeur à registre généraux qui calcul l'expression suivant : ( a + b ) * c, avec a, b, c des constantes.

```
Mov R1, #a          ; R1 ← a
Mov R2, #b          ; R2 ← b
Add R3, R1, R2      ; R3 ← R1+R2
Mov R4,#c           ; R1 ← c
Mul R5, R3, R4      ; R5 ← R3*R4
```

b) **Les architectures à pile** : les instructions lisent leurs opérandes et écrivent leurs résultats dans une pile (LIFO : Last In First Out).

Push X : permet de placer X au sommet de la pile (empiler)

Pop X : permet de stocker la valeur qui se trouve au sommet de la pile en X (dépiler).

**Exemple** : On considère que la pile est initialement vide, écrire un programme qui fait le calcul suivant x = a + b * c**.**

```
Push #a
Push #b
Push #c
Mul
Add
Pop x
```

## 3.5 Modes d'adressage

Un mode d'adressage définit la manière dont le microprocesseur va accéder à l'opérande. Les différents modes d'adressage dépendent des microprocesseurs mais on retrouve en général :

**Adressage immédiat** : La valeur de l'opérande est contenue dans le champ adresse.

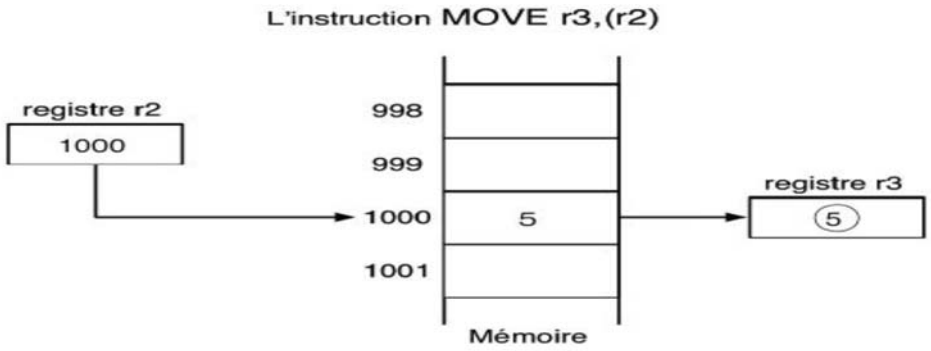
Exemple 3.1 : MOVE R1 , #100 ; R1 | 100 |

**Adressage direct** : Le champ adresse de l'instruction contient l'adresse effective de l'opérande.

Exemple 3.2 : MOVE R1, 100 ; *Après cette instruction le registre R1 contiendra le mot qui se situe à l'adresse 100 en mémoire.*

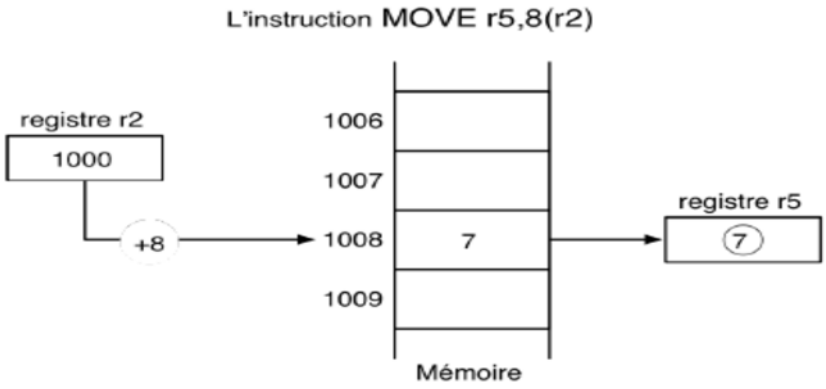
**Adressage indirect :** Le champ adresse contient l'adresse d'un pointeur : mot en mémoire qui contient l'adresse effective de l'opérande.

Exemple 3.3 : MOVE R3, (R2) *; Après cette instruction R3 contiendra la valeur du mot dont l'adresse est contenue dans R2*




**Adressage indirect avec déplacement :** On peut combiner le mode d'adressage indirect avec un décalage. C'est le mode indirect avec déplacement. L'adresse de la donnée est la somme de l'adresse contenue dans un registre (mode indirect) et d'un déplacement immédiat indiqué dans l'instruction.

Exemple 3.4 : MOVE R5, 8(R2) *; met dans R5 la valeur dont l'adresse mémoire est obtenue en ajoutant 8 à celle contenue dans R2.*




**Adressage relatif :** L'adresse effective est obtenue en additionnant le contenu du compteur ordinal au contenu du champ adresse de l'instruction. Ce type d'adressage est utilisé par exemple dans des instructions de branchement.

Exemple 3.5 : BR 50 ; *Si le CO contient la valeur 100, alors l'adr. de l'opérande est : 100+50 =150*

**Adressage indexé :** Dans ce type d'adressage, l'adresse de l'opérande est calculée en additionnant le contenu du champ d'adresse de l'instruction au contenu du registre d'index. Ce mode d'adressage est très utile lorsqu'on travaille, par exemple, sur des tableaux.

## 3.6 L'unité de contrôle et de commande (UCC)

L'UCC son rôle est de commander et contrôler le fonctionnement du système en utilisant des circuits pour décoder les instructions du programme et les transformer en signaux de commande vers toutes ces unîtes.

### 3.6.1 Structure d'une U.C.C.

Les principaux éléments d'une UCC sont :

- **Le compteur ordinal (CO)**
- **Le registre instruction (RI)**
- **Le décodeur :** Il s'agit d'un ensemble de circuits qui a pour rôle d'identifier l'instruction à exécuter, puis d'indiquer au séquenceur la nature de cette instruction. L'instruction se trouve dans le registre RI
- **Le séquenceur :** Le séquenceur est un automate permettant l'exécution effective de l'instruction placée dans le registre RI en générant les signaux de commande nécessaires pour actionner et contrôler les unités qui participent à l'exécution d'une instruction donnée. Le séquenceur est rythmé par l'horloge du microprocesseur et il peut être réalisé de 2 façons : séquenceur câblé ou séquenceur microprogrammé.
  - **Câblé** : circuit séquentiel complexe qui fait correspondre à chaque instruction exécutable un sous-circuit capable de commander son déroulement. Le sous-circuit approprié est activé par un signal provenant du décodeur.
  - **Microprogrammé** : suite de micro-instructions stockées dans une mémoire rapide. Ce microprogramme est capable de générer une suite de signaux de commandes équivalents à celle produites par un séquenceur câblé.
- **L'horloge :** Le rôle de l'horloge dans l'UCC est de synchroniser toutes les actions du processeur, en générant des signaux périodiques qui définissent le cycle machine qui correspond à la durée élémentaire régissant le fonctionnement de la machine.

### 3.6.2 Fonctionnement d'une U.C.C :

Essayons de suivre les étapes effectuées par le processeur pour exécuter une instruction machine. Cette instruction comporte deux champs.

- Code opération (obligatoire).
- un ou plusieurs champs adresse opérandes.

Une instruction est exécuté par le processeur au cours d'un cycle d'instruction (également appelé cycle de recherche et exécution). L'instruction passe donc par 2 cycles.

- Cycle de recherche : cherche l'instruction et décodage.
- Cycle d'exécution : exécution par l'UAL de l'instruction.

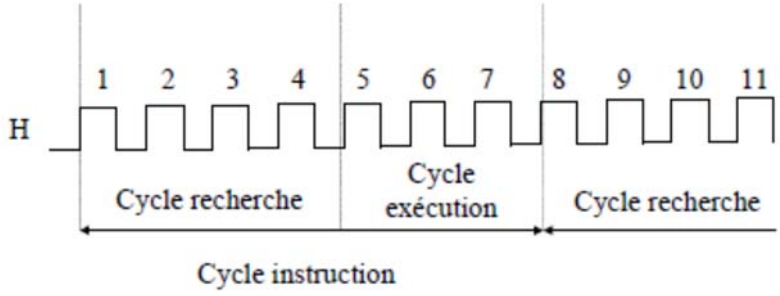



*Figure 3.2. Cycle d'instruction*

- **Le cycle de recherche :** Il comprend les étapes suivantes **:**

  **1.** Transfert de l'adresse de la nouvelle instruction à exécuter du CO vers le registre adresse mémoire RA (en MC)

  **2.** Une commande est générée par l'UCC pour lire l'instruction et l'envoyer vers le registre RM.

  **3.** L'instruction est transférée dans RI (code op + adresse op).

  **4.** Pendant que l'adresse de l'opérande est envoyée vers RA le code opération est transmis au décodeur qui détermine le type de l'opération et le transmet au séquenceur.

  **5.** Le CO est incrémentée pour le suivant le cycle de recherche s'il ne s'agit pas d'une instruction de branchement.

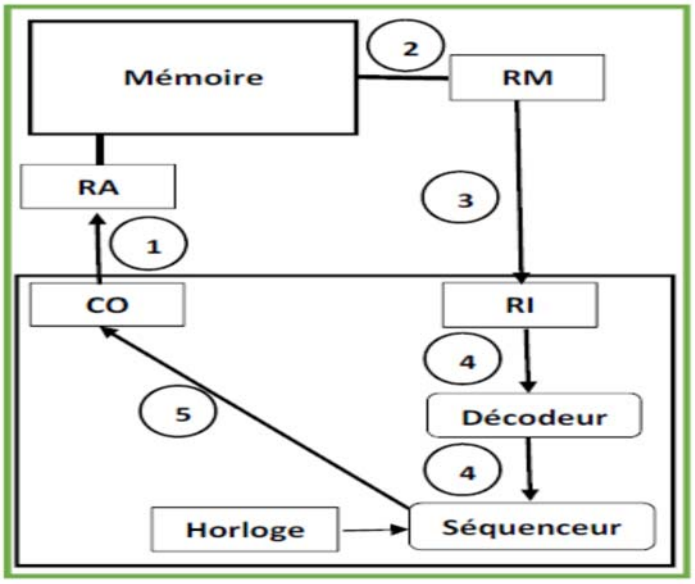



*Figure 3.3. **Cycle de recherche d'une instruction***

- **Le cycle d'exécution :** Il comprend les étapes suivantes **:**

  1- Le séquencer lance des signaux de commande vers le mémoire pour lire l'opérande à l'adresse déjà stockée dans le RA et le transférer dans RM.

  2- Transfert du contenu de RM vers l'UAL, et plus précisément vers l'accumulateur ou tout autre registre affecté à l'opération spécifiée.

  3- L'opération est effectuée sous contrôle du séquenceur.

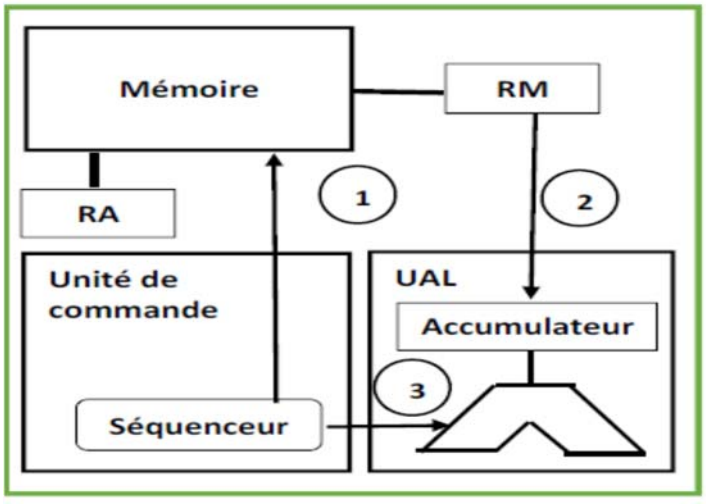



*Figure 3.4 : Cycle d'exécution d'une instruction*

La figure 3.5 présente un exemple de l'exécution en interne de l'instruction ADD r1, r2 , r3.

L'exécution d'une instruction peut se décomposer en un certain nombre d'étapes composant le cycle d'instruction :

- récupération de l'instruction et mise à jour de PC (Fetch 1) ;
- décodage de l'instruction (Decode) ;
- récupération des données (Fetch 2) ;
- exécution de l'instruction (Execute) ;
- écriture du résultat et modification des bits conditions (Write).

Chaque étape est gérée par le séquenceur et transformée en ordres internes.

Chaque instruction demande ainsi de nombreux ordres et transferts internes, entièrement contrôlés par le séquenceur et rythmés par son horloge. Plus les instructions du processeur sont compliquées, plus elles nécessitent d'étapes internes et plus le séquenceur est complexe.

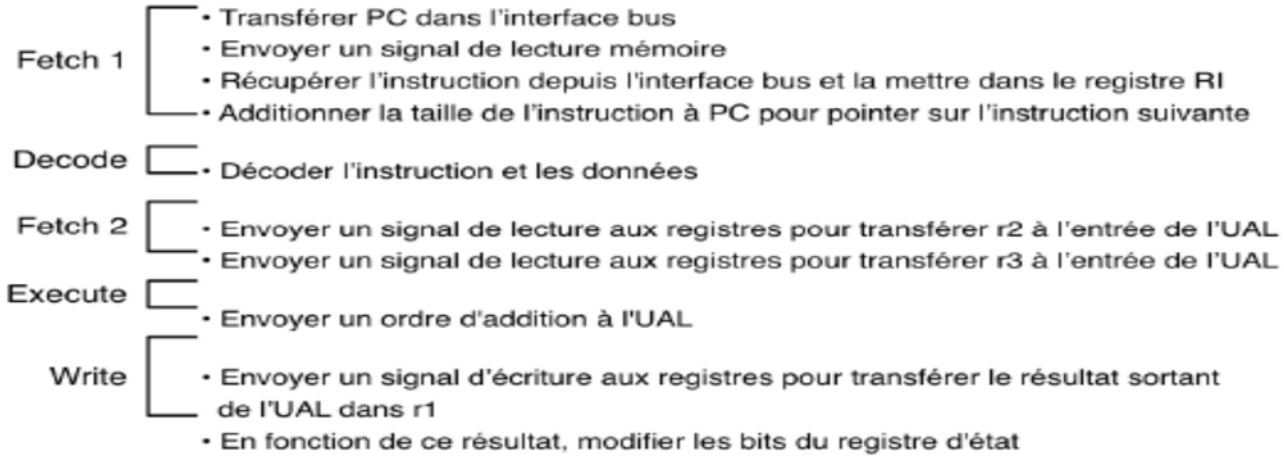



*Figure 3.5. Exécution de ADD r1, r2,r3.*

## 3.7 Le pipeline

### 3.7.1 Etages d'un pipeline

Chaque instruction peut être décomposée en plusieurs étapes (la lecture de l'instruction, son décodage, son exécution, etc.), nécessitant chacune des circuits différents. Ces circuits travaillent pendant une partie du temps d'exécution, mais lorsque l'instruction n'est pas à l'étape concernée ils restent inactifs.

Le principe du traitement pipeline consiste à ne pas attendre la fin de l'exécution d'une instruction pour lancer la suivante, en profitant de l'inactivité des circuits ayant déjà traité l'instruction en cours. C'est l'idée du travail à la chaîne : un circuit effectue une étape d'exécution et enchaîne immédiatement la même étape avec l'instruction suivante pendant que la première avance dans la chaîne de traitement. Chaque maillon de la chaîne est alors un *étage du pipeline*.

Décomposition de l'exécution d'une instruction en étapes élémentaires (pour une architecture simple où tous les opérandes sont dans les registres ou dans l'instruction). Bien évidemment, le traitement de toutes les instructions ne nécessite pas toutes ces étapes.

1. Recherche de l'instruction (IF : Instruction Fetch) : aller chercher la nouvelle instruction et la placer dans le registre d'instruction.
2. Décodage de l'instruction et lecture des registres (ID : Instruction Decode) :
   - en fonction du code de l'instruction, générer les signaux pour l'UAL, la mémoire des registres, etc. ;
   - placer en entrée de la mémoire de registres l'adresse des registres concernés.

3. Exécution et calcul de l'adresse effective (EX : EXecution) : opération effectuée par l'UAL pour le calcul d'une valeur, d'une adresse (pour un saut).
4. Accès à la mémoire et réalisation des branchements (MEM : MEMory) : chargement (load), écriture (store), exécution des branchements et sauts.
5. Mise à jour des registres (WB : Write Back cycle) : écriture des résultats dans les registres ou dépose d'une valeur dans un registre (load).

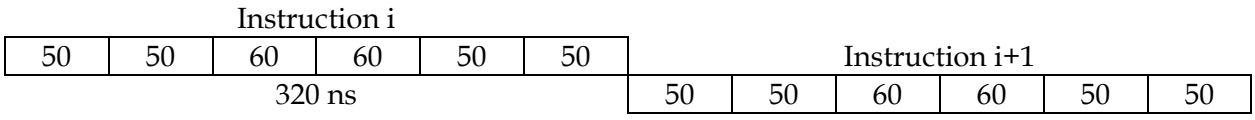
Dans un CPU pipeliné, le CPU exécute les instructions telles qu'elles se suivent en mémoire. Après avoir cherché une instruction, le processeur va chercher tout de suite l'instruction suivante en mémoire pendant qu'il décode la première, etc.

Le tableau ci-dessous illustre ce concept en décomposant une instruction en cinq étapes.

| Instructions / Cycle horloge | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| Instruction 1 | IF | ID | EX | MEM | WB | | | | |
| Instruction 2 | | IF | ID | EX | MEM | WB | | | |
| Instruction 3 | | | IF | ID | EX | MEM | WB | | |
| Instruction 4 | | | | IF | ID | EX | MEM | WB | |
| Instruction 5 | | | | | IF | ID | EX | MEM | WB |

### 3.7.2 Performances d'un pipeline

Considérons un pipeline à 6 étages de temps : 50 ns, 50 ns, 60 ns, 60 ns, 50 ns et 50 ns. Combien de temps prennent l'exécution de 100 instructions ?



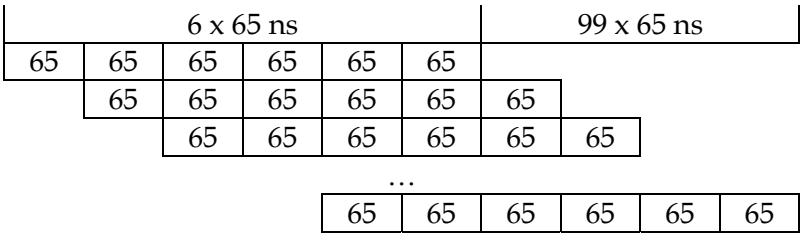

**Sans le pipeline** :

100 instructions : 32 000 ns.

**Utilisons le pipeline** :

– le temps de chaque étage est alors le même (celui du plus lent car les autres étages doivent attendre) ;
– le temps de passage entre deux étages ne peut pas être instantané, il faut attendre une stabilisation des registres de pipeline, prenons ici 5 ns.

Temps d'un étage = MAX (temps des étages) + temps de stabilisation = 60 + 5 = 65 ns.




100 instructions = 65 x 6 + 99 x 65 = 6 825 ns.

Temps moyen traitement d'une instruction non pipeliné = 320 ns.

Temps moyen traitement d'une instruction pipeliné = 65 ns (dans notre exemple : 68,25 ns).

Accélération = 320/65 = 4,92.

Un pipeline n'accélère pas le temps de traitement d'une instruction mais augmente le débit des instructions : dans un intervalle de temps on traite plus d'instructions.

### 3.7.3 Problèmes liés au pipeline

La mise en place d'un pipeline soulève plusieurs problèmes :

- le temps de traitement dans chaque unité doit être à peu près égal, sinon les unités rapides attendent les unités lentes ;
- aléas de branchement : s'il y a saut d'instruction, les instructions qui suivent le saut et qui sont en train d'être traitées dans les étages inférieurs le sont pour rien, il faudra vider le pipeline ; pour atténuer l'effet des branchements, on peut spécifier après le branchement des instructions qui seront toujours exécutées, comme c'est le cas dans l'assembleur SPARC ;
- conflits de dépendance, par exemple entre deux instructions consécutives travaillant sur la même zone mémoire (RAM ou registres) ; solution logicielle : le compilateur génère des instructions nop (ne fait rien) ; solution matérielle : détection des dépendances et mise en attente des instructions suivantes.

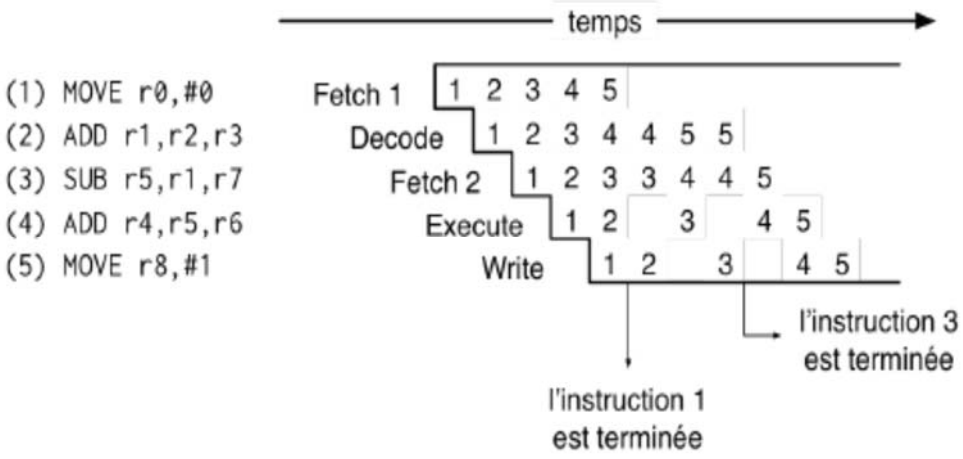



*Figure 3.6. Dépendances dans un pipeline*

L'instruction 2 modifie le registre r1, utilisé (avec sa nouvelle valeur) par l'instruction 3. Celle-ci ne peut donc pas quitter l'étape de récupération des données (Fetch 2) avant que r1 ne soit réécrit par l'instruction 2, c'est-à-dire lorsqu'elle passe l'étape 5 (Write). Lorsque le processeur modifie r1, il envoie en même temps la valeur à l'étape 3 pour l'instruction 3 (mécanisme de bypass), qui peut progresser au prochain top. L'instruction 3 est donc retardée d'un top d'horloge, induisant le retard sur tout le pipeline. Le même phénomène apparaît pour l'instruction 4, qui utilise r5, modifié par l'instruction 3.

Pour remédier à ce problème, le processeur peut essayer de réorganiser le flot d'instructions arrivant dans le pipeline en détectant les dépendances et en exécutant d'abord les instructions qui ne dépendent pas de résultats antérieurs. Cette technique complique fortement le séquenceur, qui doit récupérer plusieurs instructions en même temps, détecter les dépendances, réorganiser l'ordre des instructions et les exécuter, tout cela le plus rapidement possible.

# Chapitre 4 : Le processeur

## 4.1 Le processeur

### 4.1.1 Rôle du processeur

Le processeur est le véritable cerveau de l'ordinateur. Le processeur est un circuit électronique complexe, que toutes les avancées technologiques se concentrent sur ce composant, qui travaille toujours plus vite et effectue des opérations de plus en plus compliquées. Le rôle du processeur est d'exécuter les instructions composant le programme. Il se charge de tous les calculs mathématiques et des transferts de données internes et externes. Il décide (en fonction bien sûr des instructions du programme en cours d'exécution) de ce qui se passe à l'intérieur de l'ordinateur. Donc, Le microprocesseur est chargé d'effectuer les fonctions suivantes :

- Fournir les signaux de synchronisation et de commande à tous les éléments du système,
- Prendre en charge les instructions et les données dans la mémoire,
- Transfert des données entre la mémoire et les dispositifs d'E/S et vice versa,
- Décoder les instructions,
- Effectuer les opérations arithmétiques et logiques commandées par les instructions,
- Réagir aux signaux de commande produits par les entrées/sorties comme le signal "Reset" et les interruptions.

Toute l'activité de l'ordinateur est cadencée par une horloge unique de façon à ce que tous les circuits électroniques travaillent ensembles. La fréquence de cette horloge s'exprime en Hertz (Hz).

### 4.1.2 Caractéristiques d'un processeur

Enfin le choix d'un processeur de nos jours repose sur quelques caractéristiques clés qui sont :

- *La fréquence d'horloge (MHz) :* vitesse de fonctionnement du processeur, c'est le nombre de millions de cycles que le processeur est capable d'effectuer par seconde. La fréquence symbolise la puissance des processeurs. Aujourd'hui les processeurs de PC dépassent les 3.5 GHz (Giga Hertz = Milliards de Hertz), soit plusieurs milliards de de battements par seconde.
- *Largeur des bus (32 ou 64 bits)* : notamment du bus de données, des registres internes et du bus d'adresses. C'est un bon indicateur de la quantité d'information que celui-ci peut gérer en un temps donné et le nombre de mots mémoires maximal qu'il peut adresser.
- *Le jeu d'instructions qu'il peut exécuter : E*nsemble des opérations qu'un processeur eut exécuté, plus ou moins complexes. Par exemple : additionner deux nombres, comparer deux nombres pour déterminer s'ils sont égaux, lequel est le plus grand,...
- *la complexité de son architecture :* Cette complexité se mesure par le nombre de transistors contenus dans le microprocesseur. Plus le microprocesseur contient de transistors, plus il pourra effectuer des opérations complexes, et/ou traiter des chiffres de grande taille.
- *Le nombre de cœurs* : pour tirer profit du parallélisme des tâches (exécution en même temps de plusieurs tâches). Aujourd'hui les ordinateurs sont donc équipés de processeurs qui fonctionnent en parallèle afin d'augmenter leurs puissances. Ces Processeurs contiennent plusieurs cœurs (core) : 2, 4 et même 8 cœurs pour le grand public (1 cœur : équivalent 1 CPU ou deux processeurs logiques). Ainsi, le processeur i7 d'Intel comprend 4 cœurs (8 processeurs logiques).

- *La mémoire cache* : Les processeurs récents comptent à présent trois niveaux de cache.
  - Le cache L1 est petit mais exceptionnellement rapide. Il ne contient que les données et instructions absolument nécessaires aux programmes en cours d'exécution.
  - Le L2 est plus gros et un peu moins véloce et stocke des données et/ou instructions qui ont récemment servi ou risquent de servir.
  - Le L3 est de capacité importante mais relativement « lent ». Outre ses fonctions de cache, il assure la communication entre les différents cœurs.

*Figure 4.1. Exemple de familles de microprocesseurs*

### 4.1.3 Performances d'un microprocesseur

Chaque instruction nécessite un certain nombre de cycles d'horloge pour s'effectuer et qui dépend de la complexité de l'instruction. La durée d'un cycle dépend de la fréquence d'horloge du séquenceur. On peut caractériser la puissance d'un microprocesseur par le nombre d'instructions qu'il est capable de traiter par seconde. Pour cela, on définit :

1. le **CPI** (Cycle Par Instruction) qui représente le nombre moyen de cycles d'horloge nécessaire pour l'exécution d'une instruction pour un microprocesseur donné.

   Le $CPI$ dans un processus donné est définie comme suit :

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

   Où $IC_i$ est le nombre d'instructions pour un type d'instruction donné $i$, $CC_i$ est les cycles d'horloge pour ce type d'instruction et $IC = \sum_i IC_i$ est le nombre total d'instructions.

2. le **MIPS** (Millions d'Instructions Par Seconde) qui représente la puissance de traitement du microprocesseur.

   $$MIPS = \frac{F_H}{CPI} \qquad avec\ F_H\ fréquence\ en\ MHz$$

Pour augmenter les performances d'un microprocesseur, on peut donc soit :

- augmenter la fréquence d'horloge (limitation matérielle),
- diminuer le CPI (choix d'un jeu d'instruction adapté).

Le temps d'exécution d'un programme donné est définit par :

$$Temps\ d"exécution\ (T) = CPI * Nombre\ d'instruction\ (N_I) * Temps\ Cycle\ (T_C) = \frac{CPI * N_I}{F_H}$$

**Exemple 4.1 :** Supposons qu'on a cinq types d'instructions : Load (5 cycles), Store (4 cycles), R-type (4 cycles), Branch (3 cycles), 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 :

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

**Exemple 4.2** : Soit un processeur possédant une horloge de 400 MHz a été utilisé pour exécuter un programme contenant les instructions suivantes :

| Type d'instruction | Nombre d'instruction | nombre de cycle d'horloge |
|---|---|---|
| Integer Arithmetic | 45000 | 1 |
| Data transfer | 32000 | 2 |
| Floating point | 15000 | 2 |
| Control transfer | 8000 | 2 |

Déterminez le *CPI* effectif, le *MIPS* et le temps d'exécution de ce programme.

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

$$F_H = 400\ Mhz = 400\ 000\ 000\ Hz$$

$$Performance\ du\ processeur\ effective = \frac{F_H}{CPI} = \frac{400}{1.55} = 258\ MIPS$$

$$Temps\ d"excution\ (T)\ =\ CPI *\ Nombre\ d'instruction\ (N_I) *\ Temps\ Cycle\ (T_C)\ =\ \frac{CPI *\ N_I}{F_H}$$

$$= \frac{1.55 * 100000}{400 * 1000000} = \frac{1.55}{4000} = 0.0003875\ \text{sec} = 0.3875\ ms$$

### 4.1.4 Notion d'architecture RISC et CISC

Actuellement l'architecture des microprocesseurs se composent de deux grandes familles :

- les microprocesseurs à architecture CISC (Complex Instruction Set Computer) qui signifie processeurs à jeu d'instructions complexe.
- les microprocesseurs à architecture RISC (Reduced Instruction Set Computer), qui signifie processeurs à nombre d'instructions réduits.

Les processeurs de ces deux catégories se distinguent par la conception de leurs jeux d'instructions.

Ainsi, les processeurs CISC possèdent un jeu étendu d'instructions complexes où chacune de ces instructions peut effectuer plusieurs opérations élémentaires comme charger une valeur en mémoire, faire une opération arithmétique et ranger le résultat en mémoire.

Au contraire, les processeurs RISC possèdent un jeu d'instructions réduit où chaque instruction effectue une seule opération élémentaire. Le jeu d'instructions d'un processeur RISC est plus uniforme. Toutes les instructions sont codées sur la même taille et toutes s'exécute dans le même temps (un cycle d'horloge en général).

Le choix dépendra des applications visées. En effet, si on diminue le nombre d'instructions, on crée des instructions complexes (CISC) qui nécessitent plus de cycles pour être décodées et si on diminue le nombre de cycles par instruction, on crée des instructions simples (RISC) mais on augmente alors le nombre d'instructions nécessaires pour réaliser le même traitement.

Dans le tableau suivant on dresse les caractéristiques de chacune.

| 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 | décodeur complexe |
| beaucoup de registres | peu de registres |
| seules les instructions chargement et rangement 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 |

## 4.2 Processeur MIPS R3000 : Architecture externe

### 4.2.1 Introduction

MIPS (de l'anglais Microprocessor without interlocked pipeline stages) est une architecture de microprocesseur de type RISC. Elle fut développée par la compagnie MIPS Computer Systems Inc., basée à Mountain View en Californie. Il existe plusieurs réalisations industrielles de cette architecture (Siemens, Nec, Lsi Logic, Silicon Graphics, etc...). Cette architecture est suffisamment simple pour présenter les principes de base de l'architecture des processeurs, et suffisamment puissante pour supporter un système d'exploitation multi-tâches tel qu'UNIX. Les processeurs fabriqués selon cette architecture sont utilisés dans plusieurs systèmes. On les retrouve dans plusieurs systèmes embarqués (embedded systems en anglais), comme les ordinateurs de poche, les routeurs Cisco et les consoles de jeux vidéo (Nintendo 64 et Sony PlayStation, PlayStation 2 et PSP). Les premières implémentations de l'architecture MIPS étaient de 32 bits (autant au niveau des registres que des chemins de données), mais par la suite, on a développé des implémentations de 64 bits.

Ce cours présente une version simplifiée de l'architecture externe du processeur MIPS R3000.

L'architecture externe est l'interface entre le processeur et le programmeur. L'architecture externe représente ce que doit connaitre un programmeur souhaitant programmer en assembleur, ou la personne souhaitant écrire un compilateur pour ce processeur, elle est composée de :

- registres visibles,
- jeu d'instructions,
- mode d'adressage,
- système d'interruptions / exceptions.

### 4.2.2 Registres visibles du logiciel

Tous les registres visibles du logiciel, c'est à dire ceux dont la valeur peut être lue ou modifiée par les instructions, sont des registres 32 bits. Afin de mettre en œuvre les mécanismes de protection nécessaires pour un système d'exploitation multi-tâches, le processeur possède deux modes de fonctionnement : utilisateur/superviseur.

- **Le mode superviseur :** permet une plus grande sécurité pour le système. Quand le processeur est en mode superviseur, l'utilisateur ne peut pas accéder à certaines zones (ex : mémoire, registres...) ;

- **Le mode utilisateur :** Ce mode permet à l'utilisateur d'accéder uniquement aux zones réservées aux utilisateurs. Le processeur part en exception si l'utilisateur accède à une zone privilégiée.

  **Note** : quand le processeur est en mode superviseur, le système peut accéder aux zone protégées et non protégées.

Ces deux modes de fonctionnement imposent d'avoir deux catégories de registres :

- Les registres **non protégés** (Accessibles en mode utilisateur)
- Les registres **protégés** (Accessibles en mode superviseur)

#### 4.2.2.1 Les registres non protégés

Le processeur MIPS possède 32 registres de travail accessibles au programmeur et chaque registre est connu par son numéro (numéroté de 0 à 31), et est préfixé par un $.

Par exemple :
– Le registre 31 sera noté $31 dans l'assembleur.
– Le registre $0 est un registre particulier. Il contient toujours la valeur constante 0 (l'écriture ne modifie pas son contenu).
– Le registre $29 est le registre pointeur de pile. Il prend la notation $Sp.

En plus des registres présentés, MIPS R300 possède d'autres registres :

- **$PC** Registre compteur de programme (Program Counter : CO) : Ce registre contient l'adresse de l'instruction en cours d'exécution.
- **$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 le reste dans HI).

#### 4.2.2.2 Registres protégés

L'architecture **MIPS** définit 32 registres (numérotés de 0 à 31), qui ne sont accessibles, en lecture comme en écriture, que par les instructions privilégiées (mode superviseur). On dit qu'ils appartiennent au "coprocesseur système".

En pratique, cette version du processeur **MIPS R3000** en utilise 4 pour la gestion des interruptions et des exceptions.

**SR:** Registre d'état (Status Register). Il contient en particulier le bit qui définit le mode : superviseur ou utilisateur, ainsi que les bits de masquage des interruptions. (Ce registre possède le numéro 12).

**CR :** Registre de cause (Cause Register). En cas d'interruption ou d'exception, son contenu définit la cause pour laquelle on fait appel au programme de traitement des interruptions et des exceptions. (Ce registre possède le numéro 13).

**EPC :** Registre d'exception (Exception Program Counter). Il contient l'adresse de retour (PC+4) en cas d'interruption. Il contient l'adresse de l'instruction fautive en cas d'exception (PC). (Ce registre (Ce registre possède le numéro 14)

**BAR:** Registre d'adresse illégale (Bad Address Register). En cas d'exception de type "adresse illégale", il contient la valeur de l'adresse mal formée. (Ce registre possède le numéro 8).

| Registres | Nom | Description |
|---|---|---|
| $0 | $zero | vaut zéro en lecture, non modifié par écriture |
| $1 | $at | Réservé à l'assembleur pour les macros. Ne doit pas être utilisé |
| $2 | $v0 | Utilisé pour les valeurs de retour des fonctions |
| $3 | $v1 | Utilisé. pour les appels systèmes. Contient le numéro d'appel de la fonction système syscall. |
| $4,$5,$6,$7 | $a0 … $a3 | Utilisés par le compilateur pour optimiser les appels de fonctions pour contenir les arguments |
| $8, … , $15 | $t0 … $t7 | Registres de travail utilisable par le code utilisateur |
| $16, … , $23 | $s0 … $s7 | Registres de sauvegardes d'usage universel |
| $24, $25 | $t8, $t9 | Registres de travail utilisable par le code utilisateur |
| $26, $27 | $k0, $k1 | Réservés aux procédures noyau. |
| $28 | $gp | Pointeur global. Constant pour tout processus . |
| $29 | $sp | Pointeur de pile |
| $30 | $s8 | Pointeur sur la zone des variables globales (section data) |
| $31 | $ra | Contient l'adresse de retour d'une fonction |
| | | ***Registres spéciaux*** |
| HI | | Poids fort du résultat d'une multiplication sur 64 bits ou reste de division. |
| LO | | Poids faible du résultat d'une multiplication sur 64 bits ou quotient de division. |
| PC | | Registre compteur de programme (Program Counter). |

### 4.2.3 Organisation de la mémoire

Dans l'architecture MIPS R3000, l'espace adressable est divisé en deux segments : le segment utilisateur, et le segment système (noyau). Un programme utilisateur utilise généralement trois sous-segments (appelés sections) dans le segment utilisateur :

a) la section **text** 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.

b) la section **data** contient les données globales manipul.es par le programme utilisateur. Elle est implantée conventionnellement à l'adresse **0x10000000**. Sa taille est fixe et calculée lors de l'assemblage.

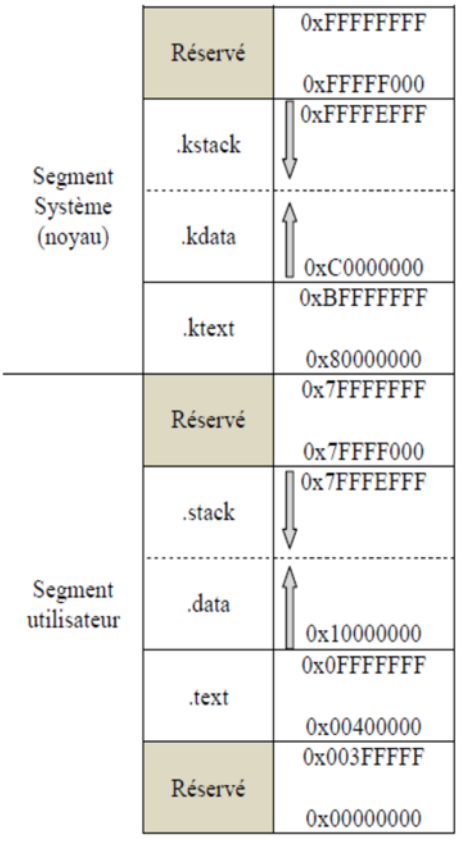
c) la section **stack** contient la pile d'exécution du programme utilisateur. Sa taille varie au cours de l'exécution. Elle est implantée conventionnellement à l'adresse **0x7FFFF000**. La pile s'étend vers les adresses décroissantes.

Trois sections sont également définies dans le segment noyau (kernel):

a) la section **ktext** contient le code exécutable en mode noyau. Elle est implantée conventionnellement à l'adresse **0x80000000**. Sa taille est fixe et calculée lors de l'assemblage.

b) la section **kdata** contient les données globales manipulées par le système d'exploitation en mode noyau. Elle est implantée conventionnellement à l'adresse **0xC0000000**. Sa taille est fixe et calculée lors de l'assemblage.

c) la section **kstack** contient la pile d'exécution du noyau. Sa taille varie au cours de l'exécution. Elle est implantée conventionnellement. l'adresse **0xFFFFF000**. Cette pile s'étend vers les adresses décroissantes.



**Adressage mémoire**

L'unité adressable est l'octet ce qui signifie que la mémoire est vue comme un tableau d'octets, contenant les instructions et données associées.

Les adresses et les instructions ont codées sur 32 bits.

Le processeur respecte la convention « little endian ».

Les mots mémoires sont adressés par des multiples de 4 ; (le CO est incrémenté de 4 à chaque cycle).

Le MIPS ne possède qu'un unique mode d'adressage pour lire ou écrire des données en mémoire : l'adressage indirect registre avec déplacement. L'adresse est obtenue en additionnant le déplacement (positif ou négatif) au contenu du registre. S'il n'y a pas d'entier devant la parenthèse ouvrante, le déplacement est nul. Pour ce qui concerne les sauts, il faut nécessairement utiliser des labels.

**Exemple 4.3 :**

```
lw $t4, 13($t2)          # $t4 ← Mem[$t2 + 13]
sw $s4, -60($s6)         # Mem[$s6 - 60] ← $s4
```

## 4.2.4 Jeu d'instructions

### 4.2.4.1 Généralités

Le jeu d'instructions est "orienté registres". Cela signifie que les instructions arithmétiques et logiques prennent leurs opérandes dans des registres et rangent le résultat dans un registre. La plupart des instructions arithmétiques et logiques se présentent sous les 2 formes registre-registre et registre-immédiat. Les seules instructions permettant de lire ou d'écrire des données en mémoire effectuent un simple transfert entre un registre général et la mémoire. Le processeur possède un ensemble d'instructions qui se répartissent en 4 classes :

- instructions arithmétiques/logiques entre registres
- instructions de branchement
- instructions de lecture/écriture mémoire
- instructions systèmes

### 4.2.4.2 Les instructions arithmétiques

**a-) L'addition**

**add :** Addition registre registre signée

**Syntaxe** : add $rr, $ri, $rj

**Description** : Les contenus des registres **$ri** et **$rj** sont ajoutés pour former un résultat sur 32 bits qui est placé dans le registre **$rr**.

$$rr \leftarrow ri + rj$$

**Exception** : génération d'une exception si dépassement de capacité.

**Addi :** Addition registre immédiat signée

**Syntaxe** : addi $rr, $ri, imm

**Description** : La valeur immédiate sur 16 bits subit une extension de signe, et est ajoutée au contenu du registre **$ri** pour former un résultat sur 32 bits qui est placé dans le registre **$rr**.

$$rr \leftarrow (imm_{15}^{16} \parallel imm_{15\ldots0}) + ri$$

**Exception** : génération d'une exception si dépassement de capacité.

**Addu :** Addition registre registre non-signée

***Syntaxe*** *:* **addu $rr, $ri, $rj**

***Description*** *:* **rr ← ri + rj**

**Addiu** Addition registre immédiat non-signée
*Syntaxe* : **addiu $rr, $ri, imm**
*Exception :* pas d'exception

**b-) Soustraction**

**sub :** Soustraction registre registre signée
**Syntaxe** : sub $rr, $ri, $rj
**Description** : Le contenu du registre **$rj** est soustrait du contenu du registre **$ri** pour former un résultat sur 32 bits qui est placé dans le registre **$rr**.
$rr \leftarrow ri - rj$
**Exception** : génération d'une exception si dépassement de capacité.

**Subu :** Soustraction registre registre non-**signée**
*Syntaxe :* **subu $rr, $ri, $rj**
*Description :* Le contenu du registre $rj est soustrait du contenu du registre $ri pour former un résultat sur 32 bits qui est placé dans le registre $rr. **rr ← ri - rj**

**Exemple 4.4 :** Donner la suite des instructions MIPS pour effectuer les actions suivantes

a) Mettre à zéro le registre R9
**Add $t1, $zero, $zero**

b) Mettre 8000H dans le registre R8
**Addi $t0, $zero, 0x8000**

**Exemple 4.5 :** Programmez en assembleur MIPS les fonctions **:** A : = B + C + D et E : = F – A
Sachant que: A←$s0, B←$s1, C←$s2, D←$s3, E←$s4, F←$s5

```
add $t0, $s1, $s2
add $s0, $t0, $s3
sub $s4, $s5, $s0
```

**c-) La multiplication**

**mult :** Multiplication signé registre registre
**Syntaxe** : mult $ri, $rj
**Description** : Le contenu du registre **$ri** est multiplié par le contenu du registre **$rj**, le contenu des deux registres étant considéré comme des nombres en complément à deux. Les 32 bits de poids fort du résultat sont placés dans le registre **hi** et les 32 bits de poids faible dans le registre **lo**

**multu :** Multiplication non signée
**Syntaxe** : multu $ri, $rj
Meme comportement que mult sauf que les nombres sont non signés

**d-) La division**

**div** : Division entière registre registre et reste signé
**Syntaxe** : div $rs, $rt
**Description** : Le contenu du registre **$rs** est divisé par le contenu du registre **$rt**, le contenu des deux registres étant considéré comme des nombres en complément à deux. Le résultat de la division est placé dans le registre spécial **$lo**, et le reste dans **$hi**.

**Divu** : Division entière et reste non-signé registre registre
**Syntaxe** : Divu $rs, $rt

**Description** : Le contenu du registre **$rs** est divisé par le contenu du registre **$rt**, le contenu des deux registres étant considéré comme des nombres non signés. Le résultat de la division est placé dans le registre spécial **$lo**, et le reste dans **$hi**.

#### 4.2.4.3 Les instructions logiques

**And** : Et logique : Et bit-à-bit registre registre
**Syntaxe** : and $rd, $rs, $rt
**Description** : Un **et** bit-à-bit est effectué entre les contenus des registres **$rs** et **$rt**. Le résultat est placé dans le registre **$rd**.

**Andi** : Et logique immédiat. Et bit-à-bit registre immédiat
**Syntaxe** : andi $rd, $rs, imm
**Description** : La valeur immédiate sur 16 bits subit une extension de zéros. Un et bit-à-bit est effectué entre cette valeur étendue et le contenu du registre **$rs** pour former un résultat placé dans le registre **$rd**.

**Or** : Ou logique. Ou bit-à-bit registre registre
**Syntaxe** : or $rd, $rs, $rt
**Description** : Un **ou** bit-à-bit est effectué entre les contenus des registres **$rs** et **$rt**. Le résultat est placé dans le registre **$rd**.

**Ori** : Ou logique immédiat. Ou bit-à-bit registre, immédiat
**Syntaxe** : ori $rd, $rs, imm
**Description** : La valeur immédiate sur 16 bits subit une extension de zéros. Un **ou** bit-à bite est effectué entre cette valeur étendue et le contenu du registre **$rs** pour former un résultat placé dans le registre **$rd**.

#### 4.2.4.4 Les instructions de transfert de données

**mfhi** : Copie du registre **$hi** dans un registre général
**Syntaxe** : mfhi $rd
**Description** : Le contenu du registre spécialisé **$hi (**qui est mis à jour par l'opération de multiplication ou de division) est recopié dans le registre général **$rd**.

**Mflo** : Copie du registre **$lo** dans un registre général
**Syntaxe** : mflo $rd
**Description** : Le contenu du registre spécialisé **$lo (**qui est mis à jour par l'opération de multiplication ou de division) est recopié dans le registre général **$rd**.

**Move** : transfert registre/registre
**syntaxe** : move $rd, $rs
**Description** : Le mot contenu dans le registre **$rs** est copié dans le registre **$rd**.

#### 4.2.4.5 Instruction de chargement rangement dans la mémoire principale (Lecture-Ecriture)

Les deux instructions lw (load word = lecture) et sw (store word =écriture) permettent les échanges entre la mémoire centrale et les registres.

**Lw :** Lecture d'un mot de la mémoire
*Syntaxe* : **lw $rr, imm($ri)**
*Description* : L'adresse de lecture est la somme de la valeur immédiate sur 16 bits, avec extension de signe, et du contenu du registre $ri. l'adresse doit être multiple de 4. Le mot de 32 bits lu à cette adresse est placé dans le registre $rr. **rr ← Mem[imm + ri]**

**Sw :** écriture d'un mot en mémoire

*Syntaxe* : **sw $rj, imm($ri)**

***Description*** : L'adresse d'écriture est la somme de la valeur immédiate sur 16 bits, avec extension de signe, et du contenu du registre $ri. l'adresse doit être multiple de 4. Le contenu du registre $rj est écrit en mémoire . **Mem[imm + ri] ← rj**

**La :** Chargement d'une adresse dans un registre (c'est une Macro instruction! )

*Syntaxe* : **la $rr, adr**

***Description*** : L'adresse considérée comme une quantité non-signée est chargée dans le registre **$rr**. **rr ← adr**

**Exemple 4.6** :

```
lw $2, 10($3)      # copie dans le registre $2 la valeur située dans la mémoire principale à
                   l'adresse m obtenue en ajoutant 10 au nombre stocké dans la registre $3.
sw $2, 15($1)      # copie la valeur présente dans le registre $2 dans la mémoire principale à
                   l'adresse m obtenue en ajoutant 15 au nombre stocké dans la registre $1.
```

**Exemple 4.7 :** une séquence assembleur MIPS pour calculer **z = w + y ;**

```
la $t10, w         # mettre l'adresse de w dans $t10
lw $s0, 0($t10)    # mettre le contenu de w dans  $s0
la $t1, y          # mettre l'adresse de y dans $t1
lw $s1, 0($t1)     # mettre le contenu de y dans $s1
add $s2, $s0, $s1  # addition w + y, mettre le résultat dans  $s2
la $t2, z          # mettre l'adresse de z dans $t2
sw $s2, 0($t2)     # mettre le contenu de $s2 dans z
```

#### 4.2.4.6 Instructions de branchement

**Beq :** Branchement conditionnel si registre égal registre

**Syntaxe** : beq $rs, $rt, label

**Description** : Les contenus des registres **$rs** et **$rt** sont comparés. S'ils sont égaux, le programme saute à l'adresse correspondant à l'étiquette, calculée par l'assembleur.

Addr ←label

Si rs = rt alors pc ←pc + 4 + addr

Même format pour (**bne, bgt, bge, blt, ble**) respectivement ($\neq, >, \geq, <, \leq$)

**Bgez** : Branchement si registre supérieur ou égal à zéro

**Syntaxe** : bgez $ri, label

**Description** : Si le contenu du registre **$ri** est supérieur ou égal à zéro le programme saute à l'adresse correspondant à l'étiquette, calculée par l'assembleur.

Addr ←label

Si ri > =0 alors pc ←pc + 4 + addr

**J :** Branchement inconditionnel immédiat

**Syntaxe** : j label

**Description** : Le programme saute inconditionnellement à l'adresse correspondant au label, calculé par l'assembleur. Pc ←label.

**Exemple 4.9 :** x←$t1, y←$t2, z←$t3.

| | | |
|---|---|---|
| Si x = y alors | bne $t1, $t2, sinon | beq $t1, $t2, then |
| z := x + y | add $t3, $t1, $t2 | sub $t3, $t1, $t2 |
| sinon | j fsi | j fsi |
| z := x - y | sinon : sub $t3, $t1, $t2 | then : add $t3, $t1, $t2 |
| fsi | fsi : | fsi : |

**Exemple 4.10** : Ecrire un programme en MIPS qui calcul la valeur absolue d'un nombre.

**Programme C**

```
void main()
{
  int x = -10 ;
  int valabs :
  valabs = x;
  if(x <= 0)
    valabs = -x;
  exit(0);
}
```

**Programme assembleur**

```
            .data
x :         .word    -10
valabs :    .word     0
            .text
__start:
            la      $a0, x
            lw      $s0, 0($a0)
            bgtz    $s0, suite
            sub     $s0, $zero, $s0
suite :
            la      $a1, valabs
            sw      $s0, 0($a1)
            ori     $v0, $zero, 10
            syscall
```

## 4.2.5 Programmation en assembleur MIPS

Le langage assembleur est le langage de programmation ; c'est une version lisible par un humain du langage machine, obtenu en remplaçant les valeurs entières du langage machine par des mnémoniques (instruction du langage assembleur).

### 4.2.5.1 Règles syntaxique

On définit ci-dessous les principales règles d'écriture d'un programme source.

**Les commentaires :** ils commencent par un # ou un ; et s'achèvent à la fin de la ligne courante.

**Les entiers :** une valeur entière décimale est notée par exemple 3250, une valeur entière octale est notée 04372 (préfixée par un zéro), et une valeur entière hexadécimale est notée 0xFA (préfixée par zéro suivi de x). En hexadécimal, les lettres de A à F peuvent être écrites en majuscule ou en minuscule.

**Les chaînes de caractères :** elles sont simplement entre guillemets. Exemple : "la valeur de x est"

**Les labels (étiquettes):** Ce sont des mnémoniques correspondant à des adresses. Ces adresses peuvent être soit des adresses de variables, soit des adresses de saut. Ce sont des chaînes de caractères qui commencent par une lettre, majuscule ou minuscule, un $, un _, ou un .. Ensuite, un nombre quelconque de ces mêmes caractères auquel on ajoute les chiffres. Pour la déclaration, le label doit être suffixé par << : >>. les labels ne doivent pas avoir le même nom qu'une mnémonique de l'assembleur ou qu'un nom de registre.

Exemple : Etiq1

**Les immédiats** : ce sont les opérandes (constantes) contenus dans l'instruction. Ce sont soit des entiers, soit des labels.

### 4.2.5.2 Traduction des structures de contrôle en MIPS

Nous montrons par des exemples comment traduire en assembleur MIPS une structure de contrôle (If, While, For) écrite dans un code algorithmique ou dans un langage de haut niveau (tel que le langage C) :

**a-) IF THEN ELSE**

Soit : x → $s1, y → $s2, w → $s3, z → $s4.

```
if (x < 0) then                  bgez   $s1, else
   { y = – x ;                    sub    $s2, $zero, $s1
     w = w + 1}                   addi   $s3, $s3, 1
else                              j      suite
  { y = x ;                else :  add    $s2, $zero, $s1
    z = z + 1}                     addi   $s4, $s4, 1
                             suite :
```

**b-) La boucle WHILE**

Soit : x → $t1, y → $t2

```
while (x < y) do          while :  bge   $t1, $t2, suite
{ x = x + 1;                       addi $t1, $t1 , 1
  y = y - 1}                       addi $t2, $t2, -1
                                   j while
                          suite :
```

**c-) La boucle FOR**

```
$t0 = 0 ;                        add  $t0, $0, $0      #$t0=0
for ($s0 = 5; $s0 > 0; $s0 = $s0 - 1) do   addi $s0, $0, 5      #initialiser le compteur de la boucle a 5
{                          boucle :
  $t0 = $t0 + $s0 ;              add   $t0, $t0, $s0
}                                addi  $s0, $s0, -1    #décrémenter le compteur
                                 bgtz  $s0, boucle     # si ($s0>0) aller a boucle
```

**Exemple 4.11 :** Programme assembleur Mips qui affiche la longueur d'une chaîne.

**Programme C**

```
void main()
{
char *s="Ceci est un exemple ...";
int i=0;
while (*s!=0)
  {
    i++;
    s++;
  }
printf("%d", i);
exit(0);
}
```

**Programme Assembleur**

```
        .data
        str1:    .asciiz  "Ceci est un exemple ..."
        .text
main :
          la      $a1, str1          # pointeur sur la chaine
          addi    $v1, $zero, 0      # compteur de la boucle
loop:
          lb      $a2, 0($a1)        # élement de la chaine
          beq     $a2, $zero, end
          addi    $v1, $v1, 1        # $v1=$v1+1
          addi    $a1, $a1, 1        # $a1=$a1+1
          j       loop
end:
          add     $a0, $v1, $zero
          ori     $v0, $zero, 1      # code de 'print_int'
          syscall                    # affiche le résultat
          ori     $v0, $zero, 10     # terminer le programme
          syscall
```

#### 4.2.5.3 Organisation d'un programme en assembleur

Un programme en assembleur contient obligatoirement une zone d'instructions dont le début est repéré par la directive **.text.** La première instruction à exécuter (point d'entrée du programme) doit être repérée avec l'étiquette **__start:** Si le programme nécessite une zone de données en mémoire, le début de cette zone est repéré par la directive **.data**. D'autres directives sont disponibles pour déclarer et initialiser des variables.

**Exemple 4.12 :** Un programme qui calcule s = a + b (32 bits)

```
        .data                   # début de la section données en mémoire
a :     .word       18
b :     .word       22
s :     .word       0
        .text                   # début de la section instructions (code) en mémoire
__start:
        la      $a1, a          # première instruction du programme
        lw      $t1, 0($a1)     # $t1 ← a
        la      $a1, b
        lw      $t2, 0($a1)     # $t2 ← b
        add     $t4, $t2, $t1   # $t4 ← a + b
        la      $a1, s
        sw      $t4, 0($a1)     # $s ← $t4
        ori     $v0, $zero, 10  # appel système n°10
        syscall                 # pour terminer le programme (exit(0))
```

#### 4.2.5.4 Appels système et instructions d'entrée/sortie

**Syscall** Appel à une fonction du système (en mode noyau).

***Syntaxe*** : syscall

***Description*** : Un appel système est effectué, transférant immédiatement, et inconditionnellement le contrôle au gestionnaire d'exception. ***Note*** : par convention, le numéro de l'appel système, c.-à-d. le code de la fonction système à effectuer, est placé dans le registre $v0. Pc ← 0x80000080

**Break** Arrêt et saut à la routine d'exception

***Syntaxe*** : break imm

***Description*** : Un point d'arrêt est détecté, et le programme saute à l'adresse de la routine de gestion des exceptions. Pc ← 0x80000080

**Appels système**

Pour exécuter certaines fonctions système, typiquement les entrées/sorties (lire ou écrire un nombre, ou un caractère), il faut utiliser des appels système. Par convention, le numéro de l'appel système est contenu dans le registre $2, et son unique argument dans le registre $4.

- **écrire un entier** : Il faut placer la valeur entière à écrire dans le registre **$a0** et ensuite exécuter l'appel système numéro **1**.

```
ori $a0, 537            # met 567 dans l'argument
ori $v0, $zero, 1       # code de 'print_int'
syscall                 # affiche 567
```

- **lire un entier** : Lire un entier consiste à exécuter l'appel système numéro **5** et récupérer le résultat dans le registre **$v0**.

```
ori $v0, $zero, 5       # code de 'read_integer'
syscall                 # $v0 contient la valeur lu
```

- **écrire une chaîne de caractères** : Une chaîne de caractères étant identifiée par un pointeur, il faut passer ce pointeur à l'appel système numéro **4** pour l'afficher.

```
str:
    .asciiz "Chaîne à afficher\n"
    …
    la $a0,  str                # charge le pointeur dans $a0
    ori $v0, $zero, 4           # code de 'print_string'
    syscall                     # affiche la chaine pointée
```

- **lire une chaîne de caractères** : Pour lire une chaîne de caractères, il faut un pointeur et une longueur maximum. Il faut passer ce pointeur, dans **$a0**, et cette longueur, dans **$a1**, et exécuter l'appel système numéro **8**. Le résultat sera mis dans l'espace pointé.

```
read_str:
    .space 256
    la $a0, read_str            # charge le pointeur dans $a0
    ori $a1, $zero, 255         # charge longueur max dans $a1
    ori $v0, $zero, 8           # code de 'read_string'
    syscall                     # lit la chaîne
```

- **quitter** : L'appel système numéro 10 effectue l'**exit** du programme au sens du langage C.

```
ori $v0, $zero, 10              # indique l'appel à exit
syscall                         # quitte pour de bon!
```

**Exemple 4.14** : Programme qui lit et affiche un nombre entier.

**Programme C**

```
main()
{
    int n;
    printf("Donner une valeur entière :\n");
    scanf("%d", &n);
    printf("La valeur est : %d\n", n);
    exit(0);
}
```

**Programme Assembleur**

```
        .data
        str1:   .asciiz  "Donner une valeur entière :"
        str2:   .asciiz  "La valeur est :"
        .text
main :
        la       $a0, str1
        ori      $v0, $zero, 4
        syscall
        ori      $v0, $zero, 5
        syscall
        add      $a1, $v0, $zero
        la       $a0, str2
        ori      $v0, $zero, 4
        syscall
        add      $a0, $a1, $zero
        ori      $v0, $zero, 1
        syscall
        ori      $v0, $zero, 10
        syscall
```

#### 4.2.5.5 Traitement des tableaux

On utilise la directive data pour déclarer un tableau et les instructions de chargement et rangement pour accéder aux éléments.

Déclaration d'un tableau de mots Tab

**. data**

**Tab : .word 15,16,17,18,19,20**

L'instruction **lw $t1, Tab** charge le premier élément du tableau(15) dans le registre $t1. Si $t0 contient l'adresse du tableau, on peut charger le deuxième élément (i.e.,valeur 16) dans le registre $t1 par l'instruction : **lw $t1,4($t0)** car le mot contient 4 octets et l'octet est adressable, de même on peut charger le premier élément par l'instruction : **lw $t1,($t 0)**

**Exemple** 4.15 : Supposons que $a0 pointe vers l'adresse de début du tableau ; la suite d'instructions suivante permute le 2e et le 3e élément du tableau.

```
lw $t0,4($a0)      #chargement du 2e élément dans $t0
lw $t1,8($a0)      #chargement du 3e élément dans $t1
sw $t1,4($a0)      #rangement de $t1 ( 3e ) dans l'adresse du 2e élément
sw $t0,8($a0)      #rangement de $t0 ( 2e ) dans l'adresse du 3e élément
```

**Exemple 4.16 :** Soit un programme qui calcul la somme des éléments d'un tableau

**Programme C**

```
void main()
{  int i=6;
   int table[6]={2, 4, 5, 7, 3, 6};
   int som=0;
   do
      som += table[--i];
   while (i);
   printf(" %d", som);
   exit (0);
}
Soit :
$s0 = som;
$s1 = i;
$s2 = table;
```

**Programme Assembleur**

```
      .data
       table:        .word   2, 4, 5, 7, 3, 6
       .text
main:
        ori       $s0, $zero, 0
        ori       $s1, $zero, 6
        la        $s2, table
boucle :
        lw        $s3, 0($s2)
        add       $s0, $s0, $s3
        addi      $s2, $s2, 4
        addi      $s1, $s1, -1
        bne       $s1, $zero, boucle
        add       $a0, $s0, $zero
        ori       $v0, $zero, 1
        syscall
        ori       $v0, $zero, 10
        syscall
```

### 4.2.6 Directives supportées par l'assembleur MIPS

Les directives ne sont pas des instructions exécutables par la machine, mais permettent de donner des ordres à l'assembleur. Toutes les pseudo-instruction commencent par le caractère << . >> ce qui permet de les différencier clairement des instructions.

#### 4.2.6.1 Déclaration des sections : text, data et stack

Six directives permettent de spécifier quelle section de la mémoire est concernée par les instructions, macro-instructions ou directives qui les suivent. Sur ces six directives, deux sont dynamiquement gérées à l'exécution : ce sont celles qui concernent la pile utilisateur, **stack**, et la pile système, **kstack**. Ceci signifie que l'assembleur gère quatre compteurs d'adresse indépendants correspondants aux quatre sections **text**, **data**, **ktext** et **kdata**.

- **.text** Passage dans la section **text** dans le segment utilisateur.
- **.data** Passage dans la section **data** dans le segment utilisateur.
- **.stack** Passage dans la section **stack** dans le segment utilisateur.
- **.ktext** Passage dans la section **text** dans le segment noyau.
- **.kdata** Passage dans la section **data** dans le segment noyau.
- **.kstack** Passage dans la section **stack** dans le segment noyau.

#### 4.2.6.2 Déclaration et initialisation de variables

Les directives suivantes permettent de d'initialiser certaines zones dans les sections **text** ou **data** de la mémoire.

- **.word :** Positionne des mots successifs aux valeurs des expressions.
  **.word** ***expression, [expression,]*** **: : :**
  La valeur de chaque expression est placée dans des adresses successives de la section active.
  **Exemple 4.17 : entiers:. word -1, -1000, -100000, 1, 1000, 100000**

- **.half** Positionne des demi-mots successifs aux valeurs des expressions.
  **.half** ***expression, [expression,]*** **: : :**
  La valeur de chacune des expressions est tronquée à 16 bits, et les valeurs ainsi obtenues sont placées dans des adresses successives de la section active.
  **Exemple 4.18 : coordonnées: .half 0 , 1024**

- **.byte** Positionne des octets successifs aux valeurs des expressions.
  **.byte** ***expression, [expression,]*** **: : :**
  La valeur de chacune des expressions est tronquée à 8 bits, et les valeurs ainsi obtenues sont placées à des adresses successives de la section active.
  **Exemple 4.19 : table: .byte 1, 2, 4, 8, 16, 32, 64, 32, 16, 8, 4, 2, 1**

- **.align** *expression*
  Aligne le compteur d'adresse courant afin que *expression* bits de poids faible soient à zéro.
  **.align n** Cet opérateur aligne le compteur d'adresse sur une adresse telle que les n bits de poids faible soient à zéro. Cette opération est effectuée implicitement pour aligner correctement les instructions, demi-mots et mots.
  **Exemple 4.20 :**
  ```
  .align 2
  .byte 12
  .align 2
  .byte 24
  ```

- **.ascii** Déclare et initialise une chaîne de caractères
  **.ascii** ***chaîne, [chaîne,]*** **: : :**
  Cet opérateur place à partir de l'adresse du compteur d'adresse correspondant à la section active la suite de caractères entre guillemets. S'il y a plusieurs chaînes, elles sont placées à la

suite. Cette chaîne peut contenir des séquences d'échappement du langage C, et doit être terminée par un zéro binaire si elle est utilisée avec un appel système.
**Exemple 4.21 : message: .ascii "Bonjour, Maître!\n\0"**

- **.asciiz** Déclare et initialise une chaîne de caractères, en ajoutant un zéro binaire à la fin.
  **.asciiz *chaîne, [chaîne,]* : : :**
  Cet opérateur est strictement identique au précédent, la seule différence étant qu'il ajoute un zéro binaire à la fin de chaque chaîne.
  **Exemple 4.22 : message: .asciiz "Bonjour, Maître!\n"**

- **.space** Reserve *expression* octets, et les mets à zéro

  **.space *expression*** Un espace de taille *expression* octets est réservé à partir de l'adresse courante de la section active.

  **Exemple 4.23 :** nuls: .space 1024 # initialise 1 kilo de mémoire à zéro

## 4.2.7 Format des instructions et codage en langage machine

Toutes les instructions ont une longueur de 32 bits et possèdent un des trois formats suivants :

### 4.2.7.1 Formats des instructions

#### a-) Instruction de type R (registre)

Le format R est utilisé par les instructions nécessitant deux registres sources (rs et rt) et un registre résultat (rd).

| codop | Rs | rt | rd | decval | fonction |
|---|---|---|---|---|---|
| 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits |

- **codop** : code opération de l'instruction
- **rs** : premier source
- **rt** : deuxième source
- **rd** : registre destination (reçoit le résultat de l'opération)
- **decval** : valeur du décalage
- **fonction** : variante de l'opération décrite par codop

**Exemple 4.25** : Codez l'instruction MIPS suivante en langage machine : **add $s0, $t1, $t2**

| 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits |
|---|---|---|---|---|---|
| Special | $t1 | $t2 | $s0 | 0 | add |
| 0 | 9 | 10 | 16 | 0 | 32 |
| 000000 | 01001 | 01010 | 10000 | 00000 | 100000 |

$000000\ 01001\ 01010\ 10000\ 00000\ 100000_2$ = **0x012A8020**

#### b-) Instruction de type I (transfert)

Le format I est utilisé par les instructions de lecture/écriture mémoire, par les instructions utilisant un opérande immédiat, ainsi que par les branchements courte distance (conditionnels).

| codop | Rs | rt | Immediat/adresse |
|---|---|---|---|
| 6 bits | 5 bits | 5 bits | 16 bits |

- **rs** : registre source
- **rt** : registre cible / condition de branchement
- **immédiat/adresse** : opérande immédiate ou déplacement d'adresse

**Exemple 4.26 :** Codez l'instruction MIPS suivante en langage machine : **Lw $s1, 0x100($s2)**

| 6 bits | 5 bits | 5 bits | 16 bits |
|---|---|---|---|
| codop | rs | rt | Immediat/adresse |
| 35<br>100011 | 18<br>10010 | 17<br>10001 | 0x100<br>0000000100000000 |

100011 10010 10001 $0000000100000000_2$ = **0x8E510100**

**c-) Instruction de type J (saut)**

Le format J n'est utilisé que pour les branchements à longue distance (inconditionnels).

| Codop | immed |
|---|---|
| 6 bits | 26 bits |

- codop : code opération correspondant à l'instruction
- immed : adresse mémoire

**Exemple 4.27 :** Codez l'instruction MIPS suivante en langage machine : **J 0x1000**

| 6 bits | 26 bits |
|---|---|
| 000010 | 00000000000001000000000000 |

000010 $00000000000001000000000000_2$ = **0x08001000**

### 4.2.7.2 Codage des instructions

Le codage des instructions est principalement défini par les 6 bits du champ **code opération** de l'instruction (**INS 31:26**). Cependant, trois valeurs particulières de ce champ définissent en fait une famille d'instructions : il faut alors analyser d'autres bits de l'instruction pour décoder l'instruction. Ces codes particuliers sont : SPECIAL (valeur "000000"), BCOND (valeur "000001") et COPRO (valeur "010000")

DECODAGE OPCOD

| | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|
| 000 | **SPECIAL** | **BCOND** | J | JAL | BEQ | BNE | BLEZ | BGTZ |
| 001 | ADDI | ADDIU | SLTI | SLTIU | ANDI | ORI | XORI | LUI |
| 010 | **COPRO** | | | | | | | |
| 011 | | | | | | | | |
| 100 | LB | LH | | LW | LBU | *LHU* | | |
| 101 | SB | SH | | SW | | | | |
| 110 | | | | | | | | |
| 111 | | | | | | | | |

Ce tableau exprime que l'instruction LHU (par exemple) possède le code opération "100101". Lorsque le code opération a la valeur SPECIAL ("000000"), il faut analyser les 6 bits de poids faible de l'instruction (INS 5:0).

OPCOD = SPECIAL

| | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|
| 000 | SLL | | SRL | SRA | SLLV | | SRLV | SRAV |
| 001 | JR | JALR | | | SYSCALL | BREAK | | |
| 010 | MFHI | MTHI | MFLO | MTLO | | | | |
| 011 | MULT | MULTU | DIV | DIVU | | | | |
| 100 | ADD | ADDU | SUB | SUBU | AND | OR | XOR | NOR |
| 101 | | | SLT | SLTU | | | | |
| 110 | | | | | | | | |
| 111 | | | | | | | | |

Lorsque le code opération a la valeur BCOND, il faut analyser les bits 20 et 16 de l'instruction. Lorsque le code opération a la valeur COPRO, il faut analyser les bits 25 et 23 de l'instruction. Les trois instructions de cette famille COPRO sont des instructions privilégiées.

OPCOD = BCOND

| INS 20 \ INS 16 | 0 | 1 |
|---|---|---|
| 0 | BLTZ | BGEZ |
| 1 | BLTZAL | BGEZAL |

OPCOD = COPRO

| INS 25 \ INS 23 | 0 | 1 |
|---|---|---|
| 0 | MFC0 | MTC0 |
| 1 | RFE | |

### 4.2.7.3 Décodage d'une instruction en assembleur MIPS

Pour décoder une instruction écrite en langage machine en assembleur MIPS ; il suffit simplement de vérifier les 6 bits de poids fort s'ils sont égaux à zéro alors il s'agit d'une instruction de type R et c'est le champ fonction qui nous renseigne sur le code de l'instruction sinon l'instruction est du type I ou J.

**Exemple 4.28** : Traduire l'instruction suivante écrite en code machine en assembleur MIPS.

**0x02324020** = 000000 10001 10010 01000 00000 $100000_2$

C'est l'équivalent en décimal de : 0 17 18 8 0 32

**Codop = 0** alors il s'agit d'une instruction de transfert (type R)

**Fonction = 32** c'est le code de l'instruction ADD rs=17, rt=18, rd =8

**Add $t0, $s1, $s2**

| 31–25 | 20–15 | 10–5 | 0 | Instr |
|---|---|---|---|---|
| 0 | RT | RD | SA | sll |
| 0 | RT | RD | SA | srl |
| 0 | RT | RD | SA | sra |
| 0 | RS | RT | RD | sllv |
| 0 | RS | RT | RD | srlv |
| 0 | RS | RT | RD | srav |
| 0 | RS | 0 | 8 | jr |
| 0 | RS | 0 | RD | jalr |
| 0 | numéro | | 12 | syscall |
| 0 | numéro | | 13 | break |
| 0 | 0 | RD | 16 | mfhi |
| 0 | RS | 0 | 17 | mthi |
| 0 | 0 | RD | 18 | mflo |
| 0 | RS | 0 | 19 | mtlo |
| 0 | RS | RT | 24 | mult |
| 0 | RS | RT | 25 | multu |
| 0 | RS | RT | 26 | div |
| 0 | RS | RT | 27 | divu |
| 0 | RS | RT | RD | add |
| 0 | RS | RT | RD | addu |
| 0 | RS | RT | RD | sub |
| 0 | RS | RT | RD | subu |
| 0 | RS | RT | RD | and |
| 0 | RS | RT | RD | or |
| 0 | RS | RT | RD | xor |
| 0 | RS | RT | RD | nor |
| 0 | RS | RT | RD | slt |
| 0 | RS | RT | RD | sltu |
| 1 | RS | 0 | offset | bltz |
| 1 | RS | 1 | offset | bgez |
| 1 | RS | 16 | offset | bltzal |
| 1 | RS | 17 | offset | bgezal |
| 2 | Adresse mot | | | j |
| 3 | Adresse mot | | | jal |
| 4 | RS | RT | offset | beq |
| 5 | RS | RT | offset | bne |
| 6 | RS | 0 | offset | blez |
| 7 | RS | 0 | offset | bgtz |

| 31–26 | 25–21 | 20–16 | 15–0 | Instr |
|---|---|---|---|---|
| 8 | RS | RD | Constante signée | addi |
| 9 | RS | RD | Constante signée | addiu |
| 10 | RS | RD | Constante signée | slti |
| 11 | RS | RD | Constante signée | sltiu |
| 12 | RS | RD | Cte non signée | andi |
| 13 | RS | RD | Cte non signée | ori |
| 14 | RS | RD | Cte non signée | xori |
| 15 | | RD | Cte non signée | lui |
| 16 | 0 | RD | CS | mfco |
| 16 | 4 | RS | CD | mtco |
| 16 | 8 | 0 | offset | bcof |
| 16 | 8 | 1 | offset | bcot |
| 16 | 16 | 0 | 1 | tlbr |
| 16 | 16 | 0 | 2 | tlbwi |
| 16 | 16 | 0 | 6 | tlbwr |
| 16 | 16 | 0 | 8 | tlbp |
| 16 | 16 | 0 | 16 | rfe |
| 32 | RS | RD | offset | lb |
| 33 | RS | RD | offset | lh |
| 35 | RS | RD | offset | lw |
| 36 | RS | RD | offset | lbu |
| 37 | RS | RD | offset | lhu |
| 40 | RS | RT | offset | sb |
| 41 | RS | RT | offset | sh |
| 43 | RS | RT | offset | sw |

## 4.3 Processeur MIPS R3000 : Architecture interne

### 4.3.1 Interface du Processeur

L'interface entre le processeur et la mémoire est réalisé par les signaux ADR[31:0], DATA[31:0], RW[2:0], FRZ, BERR. Les requêtes possibles vers la mémoire sont les suivantes :

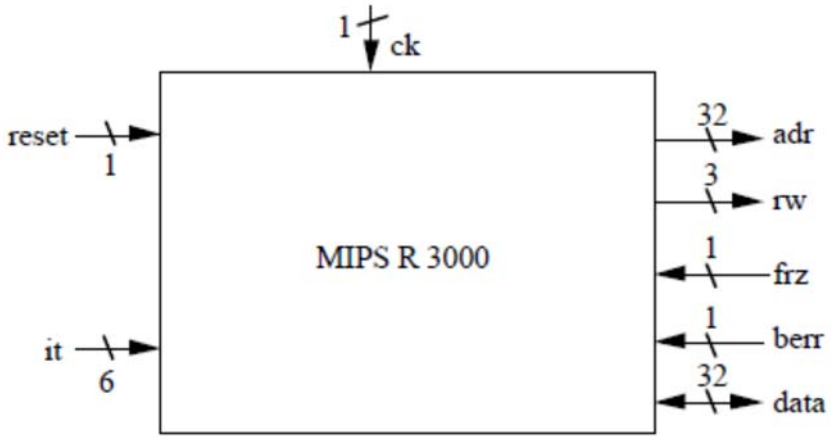
| RW | REQUETE | |
|---|---|---|
| 000 | NO | ni écriture, ni lecture |
| 001 | WW | écriture d'un mot |
| 010 | WH | écriture d'un demi-mot |
| 011 | WB | écriture d'un octet |
| 1** | RW | lecture d'un mot |

Dans le cas WW, les 4 octets du bus DATA sont écrits en mémoire à une adresse alignée sur les mots (les 2 bits de poids faible de ADR ne sont pas pris en compte). Dans le cas WH, les 2 octets de poids faibles du bus DATA sont écrits à une adresse alignée sur les demi-mots (le bit de poids faible de ADR n'est pas pris en compte). Dans le cas WB, l'octet de poids faible du bus DATA est écrit à l'adresse ADR.

Dans le cas RW, les deux bits de poids faible de ADR ne sont pas pris en compte, et la mémoire doit fournir sur le bus DATA un mot aligné. Dans le cas des instructions LBU et LHU, c'est le processeur qui effectue le recadrage et l'extension de signe.

Dans le cas où le système mémoire ne peut pas satisfaire en un cycle la requête d'écriture ou de lecture (par exemple en cas de MISS dans le cache), le signal FRZ doit être activé : le processeur maintient sa requête tant que le signal FRZ est actif.

Dans le cas d'une erreur matérielle lors d'un accès à la mémoire, cette erreur peut être signalée au processeur par le signal BERR.

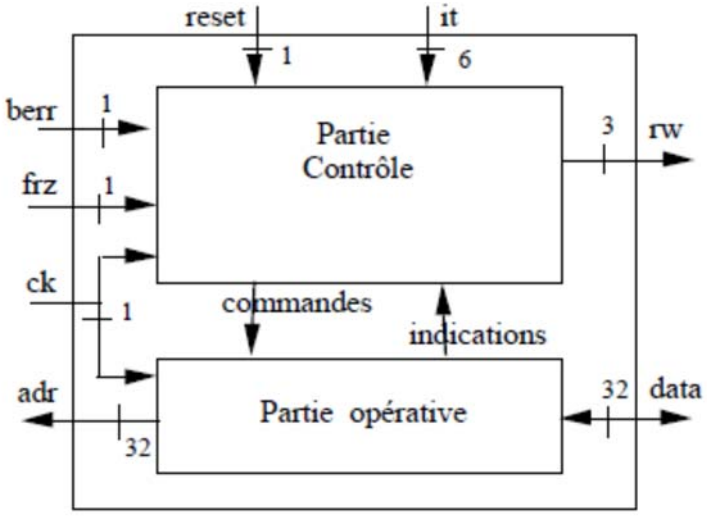



### 4.3.2 Architecture Interne

L'architecture interne du processeur se décompose deux parties :

- 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 (PC) 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).




L'interface entre la partie opérative et la partie contrôle est définie par deux nappes de fils. Dans le sens PC → PO, cette nappe de fils est décomposée en six champs de commande qui définissent de fait le format de la micro-instruction. Ce format ainsi que les valeurs possibles des différents champs de commande constituent le "langage de micro-programmation". Dans le sens PO → PC, cette nappe de fils est constituée de signaux de compte-rendu permettant au séquenceur de prendre des décisions.

### 4.3.3 Chemin de Données

En plus des registres visibles du logiciel, le processeur possède 3 registres internes :

- IR Registre instruction qui mémorise l'instruction lue en mémoire.
- DT Registre data qui reçoit la donnée provenant de la mémoire
- AD Registre d'adresse qui peut être utilisé comme registre temporaire.

Les registres SR et CR sont implantés dans la partie contrôle.

Le processeur possède deux bus opérandes X et Y et une seule "boîte à opérations" capable d'effectuer des opérations logiques, arithmétiques, mais aussi des opérations de comparaison, ou des décalages multi-bits. Toutes les opérations s'exécutent en in cycle, sauf les opérations de multiplication et de division qui s'exécutent en deux cycles. Seuls les registres PC et AD permettent d'adresser la mémoire externe.

Un mot de 32 bits provenant de la mémoire peut être écrit dans les registres DT ou IR.

Un mot de 32 bits provenant de la boîte à opérations peut être écrit dans les registres

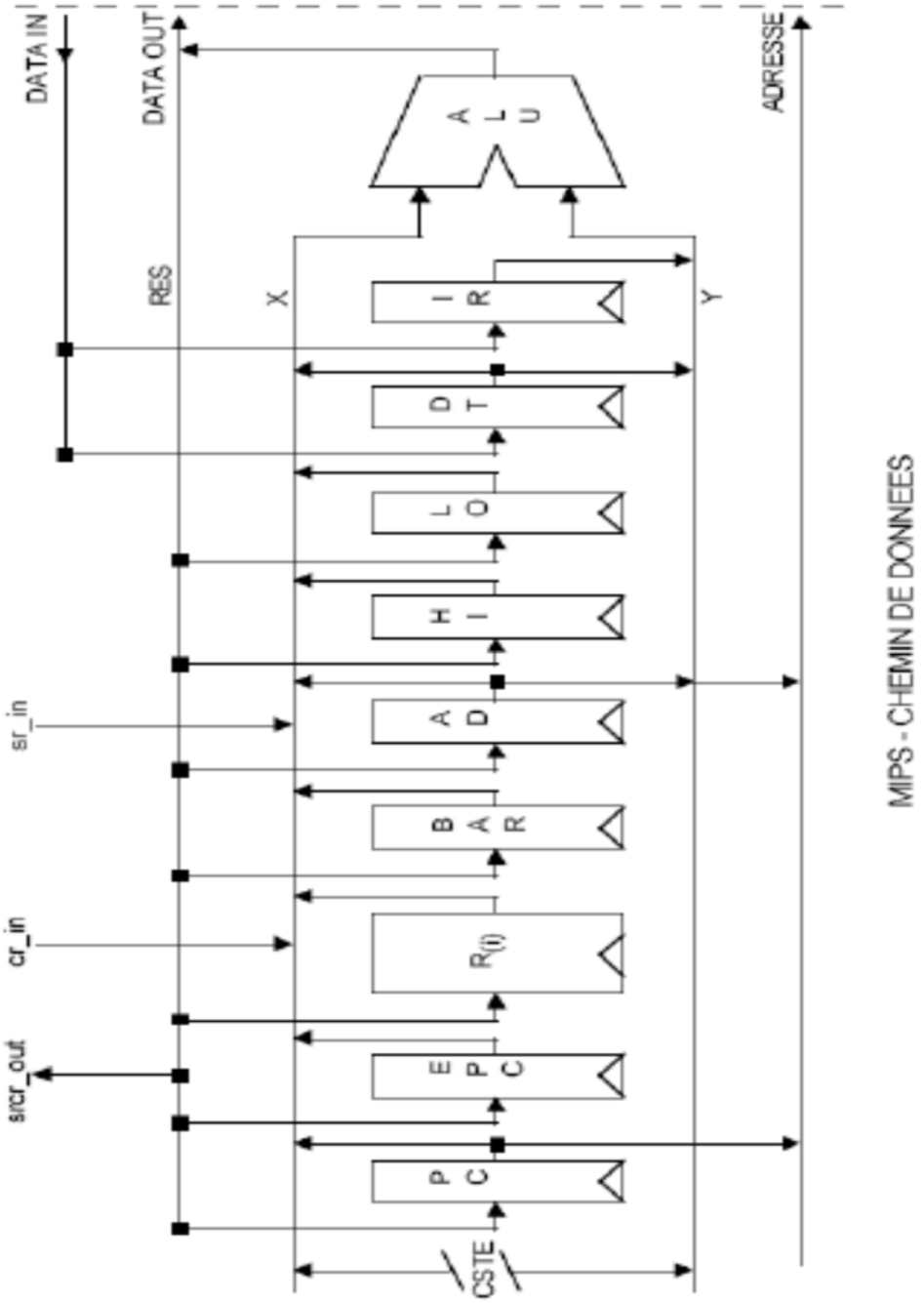
PC, EPC, SR, CR, BAR, HI, LO, AD, ou dans un registre général R(n). Ceci signifie que cette partie opérative est capable d'effectuer deux transferts élémentaires à chaque cycle : un transfert interne (qui utilise deux registres sources, la boîte à opérations et un registre résultat), et un transfert externe (lecture d'un mot en mémoire externe et écriture dans IR ou DT).

Le banc de registre R(n) ne possède que 2 accès simultanés (un en lecture et un en écriture).

Des pseudo-registres permettent de forcer des valeurs constantes sur les bus X et Y.

Les 32 bits du registre d'instruction IR sont envoyés à la partie contrôle pour lui permettre de prendre des décisions.

Par ailleurs, la partie opérative envoie vers la partie contrôle le signal NUL qui vaut 1 quand RES = Ø. Elle renvoie également vers la partie contrôle les bits de signe des deux opérandes X et Y, ainsi que celui du résultat, la retenue de l'alu et les deux bits de poids faibles du résultat. Ces signaux, calculés par la micro-instruction i peuvent être utilisés par la partie contrôle pour choisir la micro-instruction i+1.




### 4.3.4 Langage de la micro-programmation

A chaque cycle, la partie contrôle envoie vers la partie opérative des signaux de commande qui constituent la micro-instruction. La micro-instruction possède 6 champs dont les valeurs possibles définissent le langage de microprogrammation. Ces valeurs sont définies par des mnémoniques dans les tableaux ci-dessous :

- Le champ OPX définit l'opérande X en entrée de la boîte à opérations.

- Le champ OPY définit l'opérande Y en entrée de la boîte à opérations.
- Le champ ALU définit l'opération à exécuter par la boîte à opérations.
- Le champ RES définit la destination du résultat calculé par la boîte à opérations.
- Le champ ADRW définit le type d'échange entre le processeur et la mémoire.
- Le champ EXCP contrôle les mécanismes de détection d'exception et de trappes.

Une micro-instruction est définie par le choix d'une valeur pour chacun des six champs.

| OPX | OPY | ALU | RES | ADRW | EXCP |
|---|---|---|---|---|---|

| Catégorie | Instruction | Exemple | Signification |
|---|---|---|---|
| Arithmétique | addition | add $1,$2,$3 | $1 = $2 + $3 |
| | soustraction | sub $1,$2,$3 | $1 = $2 - $3 |
| | addition immédiate | addi $1,$2,100 | $1 = $2 + 100 |
| | Addition non signée | addu $1,$2,$3 | $1 = $2 + $3 |
| | Soustraction non signée | subu $1,$2,$3 | $1 = $2 - $3 |
| | add immédiat non signée | addiu $1,$2,100 | $1 = $2 + 100 |
| | Multiply signed | Mult $1 , $2 | LO = $(\$1 * \$2)_{31..0}$ HI =$(\$1 * \$2)_{63..32}$ |
| | Multiply unsigned | Multu $1 , $2 | LO = $(\$1 * \$2)_{31..0}$ HI =$(\$1 * \$2)_{63..32}$ |
| | Divide signed | Div $1 , $2 | L0 = Quotient , HI = reste |
| | Divide unsigned | Divu $1 , $2 | L0 = Quotient , HI = reste |
| | Move from coprocessor register | mfc0 $1,$epc | $1 = $epc |
| Logique | and | and $1,$2,$3 | $1 = $2 & $3 |
| | or | or $1,$2,$3 | $1 = $2 \| $3 |
| | and immediate | andi $1,$2,100 | $1 = $2 & 100 |
| | or immediate | ori $1,$2,100 | $1 = $2 \| 100 |
| | xor | xor $1,$2,$3 | $1 = $2 ⊕ $3 |
| | xor immediate | xori $1,$2,100 | $1 = $2 ⊕ 100 |
| | nor | nor $1,$2,$3 | $1 = $2 nor $3 |
| Décalages | shift left logical variable | sllv $1, $2, $3 | $1 = $2 << $3 |
| | shift right logical variable | srlv $1, $2, $3 | $1 = $2 >> $3 |
| | shift right arithmetical variable | srav $1, $2, $3 | $1 = $2 >>* $3 |
| | shift left logical | sll $1,$2,10 | $1 = $2 << 10 |
| | shift right logical | srl $1,$2,10 | $1 = $2 >> 10 |
| | shift right arithmetical | sra $1, $2, 10 | $1 = $2 >>* 10 |

| | | | |
|---|---|---|---|
| transfert | load word | lw $1,(100)$2 | $1 = Memory[$2+100] |
| | store word | sw $1,(100)$2 | Memory[$2+100] = $1 |
| | load half word signed | lh $1,(100)$2 | $1 = Memory[$2+100] |
| | load half word unsigned | lhu $1,(100)$2 | $1 = Memory[$2+100] |
| | store half word | sh $1,(100)$2 | Memory[$2+100] = $1 |
| | load byte signed | lb $1,(100)$2 | $1 = Memory[$2+100] |
| | load byte unsigned | lb $1,(100)$2 | $1 = Memory[$2+100] |
| | store byte | sb $1,(100)$2 | Memory[$2+100] = $1 |
| | Move from HI | mfhi $1 | $1 = HI |
| | Move from LO | mflo $1 | $1 = LO |
| | Move to HI | mthi $1 | HI = $1 |
| | Move to LO | mtlo $1 | LO = $1 |
| | load upper immediate | lui $1,100 | $1 = 100 * $2^{16}$ |
| branchements Conditionnels | branch on equal | beq $1,$2, label | if ($1 == $2) go to label |
| | branch on not equal | bne $1,$2, label | if ($1 != $2) go to label |
| | Branch if greater or equal zero | bgez $1, label | if ($1 >= 0 ) go to label |
| | Branch if greater than zero | bgtz $1, label | if ($1 > 0 ) go to label |
| | Branch if less or equal zero | blez $1, label | if ($1 <= 0 ) go to label |
| | Branch if less than zero | bltz $1, label | if ($1 < 0 ) go to label |
| | Branch if greater or equal zero and link | bgezal $1, label | $ra = PC + 4; if ($1 >= 0 ) go to label |
| | Branch if less than zero and link | gltzal $1, label | $ra = PC + 4; if ($1 < 0 ) go to label |
| branchements inconditionnel | jump | j label | goto lable |
| | jump and link | jal label | $ra = PC + 4; go to label |
| | jump register | jr $ra | PC = $ra (i.e. retum) |
| | set on less than | slt $1,$2,$3 | if ($2 < $3) $1 = 1; else $1 = 0 |
| | set less than immediate | slti $1,$2,100 | if ($2 < 100) $1 = 1; else $1 = 0 |
| | set less than unsigned | sltu $1,$2,$3 | if ($2 < $3) $1 = 1; else $1 = 0 |
| | set less than immediate unsigned | sltiu $1,$2,100 | if ($2 < 100) $1 = 1; else $1 = 0 |

# Travaux dirigés

# T D N°0
## (Rappels sur la représentation de l'information et circuits logiques)

**Exercice 1**
Donner sur 8 bits les représentations des nombres -32, -127 et -128 dans les systèmes de représentation suivants : signe et valeur absolue, complément à 1, complément à 2.

**Exercice 2**
a) Combien d'entiers positifs peut-on coder en binaire sur un octet ? Combien de bits faut-il pour représenter 65563 entiers différents en binaire ?
b) Indiquer la valeur codée par le mot 16 bits 1101100101110101 suivant qu'il représente un entier non signé, ou un entier signé (on utilise le codage « signe et valeur absolue »).
c) Trouver la représentation complément à 2 sur 10 bits de position de $(-276)_{10}$

**Exercice 3**
En utilisant les entiers 8 bits en complément à 2 ; effectuez les opérations suivantes :
1 - 2, 51 + 127, -3 - 127, -127 + 127, -63 - 63.
Préciser, pour chaque opération, la retenue et le débordement.

**Exercice 4**
Donnez la représentation flottante IEEE 754 en simple précision des nombres réels suivants : 10, 10.5, 0.1 et 1.875

**Exercice 5**
On considère une représentation des nombres flottants selon la norme IEEE 754 Simple précision. Quelle est la valeur décimale des représentations internes suivantes : 42EA8000, C6570000, 42EF9100.

**Exercice 6**
a) Donnez en hexadécimal la représentation du plus petit réel représentable en IEEE 754 simple précision.
b) Codez l'information $-78_{10}$ selon les formats suivants :
- valeur signée sur 8 bits;
- complément à 2 sur 8 bits;
- IEEE 754 simple précision.

**Exercice 7**
a) Soit X =40E9999A et Y= BE99999A en virgule flottante IEEE simple précision, effectuer l'opération X+Y
b) Soit X= 40A00000 et Y= C0E00000 en virgule flottante IEEE simple précision, effectuer l'opération X*Y

**Exercice 8**
Soit une transmission utilisant le code de Hamming avec une parité impaire.
a) Trouver le message transmis (représentation octale) sachant que la représentation octale des données à transmettre (composée de 16 bits) est : 116570.
**b)** Retrouver le message initial en ayant soin de corriger une éventuelle erreur si le message reçu (sur 21 bits) est : 6130014.

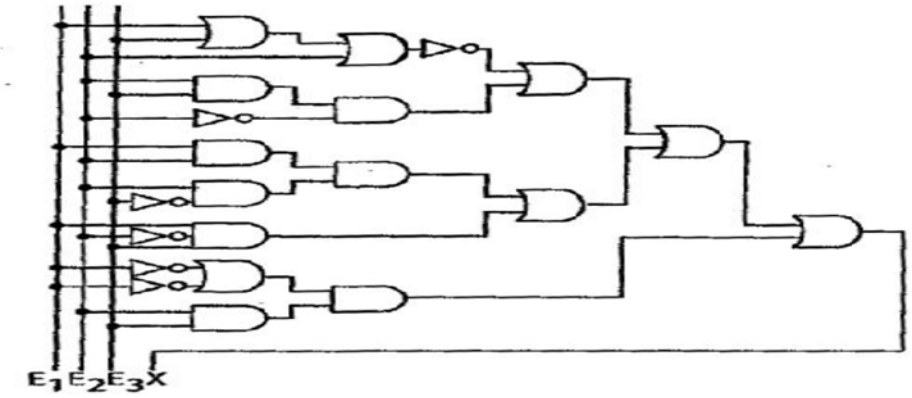
**Exercice 9**
Réaliser le circuit simplifié d'un demi-additionneur binaire qui effectué la somme de deux bits a et b et produit en sortie, le résultat de l'opération, la somme (S) et la retenue générée (R).

**Exercice 10**

Soit à réaliser un circuit M*ajoritaire* qui reçoit en entrée trois bits et qui produit en sortie la valeur 1 si la majorité des bits valent 1 et dans le cas contraire, il produit la valeur 0.

**Exercice 11**

Rappeler les étapes d'analyse d'un circuit logique en analysant le suivant :




***NB***: L'expression simplifiée et le circuit logique correspondant doivent utiliser Uniquement des portes NON et des portes XOR.

**Exercice 12**

Construire un circuit logique comparateur susceptible de comparer entre eux deux nombres binaires. En entrée, on dispose de deux nombres de trois bits chacun : $A_2\ A_1\ A_0$ et $B_2\ B_1\ B_0$ ; En sortie, on aimerait avoir : 1 si $A_2\ A_1\ A_0 = B_2\ B_1\ B_0$

0 sinon

**Exercice 13**

Le transcodeur est un circuit combinatoire qui fait correspondre à un code X en entrée sur n lignes, un code Y en sortie sur m lignes.

Construire un transcodeur qui transforme un code BCD en code excédent 3.

**Exercice 14**

Réalisez un compteur cyclique 2 bits, c'est-a-dire un circuit séquentiel capable de produire en sortie la séquence (00 01 10 11 00 01 10 11 .... ) les transitions a la valeur suivante sont contrôlées par une entrée X (X=O → pas de changement, X=1 → avancer le compteur)

NB : utiliser deux bistables T.

**Exercice 15**

Un générateur de séquence a pour but d'afficher une suite prédéterminée de nombres.

Le principe est Ie suivant :

- écrire la suite des transitions a réalisé
- déterminer les entrées des bascules qui vont réalisés ces transitions
- exprimer ces entrées en fonction des sorties attendues

Réaliser un générateur pour la séquence suivante: 0 → 4 → 1 → 3 → 5 → 2

Utiliser des bascules T.

# T D N°1

**Exercice 1**

1. Décrire l'architecture de Von Neumann et celle de Harvard ?
2. Quelle est la différence entre l'architecture de Von Neumann et celle de Harvard ?
3. Quelles sont les composants principaux d'un ordinateur et quelle est la fonction de chacun ?

**Exercice 2**

1. On considère le programme suivant qui réalise l'addition de deux nombres et affiche le résultat sur l'écran :

   ```
   Begin
     Read (a) ;
     Read (b) ;
     S = a + b ;
     Write (S) ;
   End.
   ```

   On suppose que le programme est en cours d'exécution, les instructions et les données sont stockées en mémoire centrale à leurs adresses bien définies.
   Comment se déroule l'exécution du programme ? et comment fonctionne l'ensemble UCC et UAL ?

2. Donner une relation entre le bus adresse et la taille de la MP.
   Si la taille de d'une mémoire principale est de 1 Go, quelle est la largeur du bus d'adresse ?
   Donner la taille de cette mémoire en mot mémoire si un mot mémoire = (16 bits, 32 bits, 64 bits).

# T D N°2

**Exercice 1**

1. Préciser la fonction de la RAM dans un ordinateur. Quelles sont ses caractéristiques principales ?
2. Quelles sont les principales différences entre la DRAM et la SRAM ? Où utilise-t-on la DRAM ? la SRAM ?
3. Quelles sont les principales différences entre la RAM et ROM ? Où utilise-t-on la ROM ?
4. Décrire le cycle de lecture et écriture de la RAM.
5. Quelle est le rôle des bus en générale? et quelles sont ses caractéristiques ?
6. Pourquoi utilise-t-on des mémoires caches ? Pour qu'elle raison la mémoire cache est très couteuse ?
7. Classez les mémoires suivantes par taille, par rapidité : RAM, registres, disques durs, cache L1, cache L2, cd-rom. Que constatez-vous ?
8. Qu'est-ce qu'un registre ? A quoi servent les registres suivants :
   a. PC/IP (ou CO/PI)
   b. IR (ou RI)
   c. SP (ou PP)
   d. Accumulateur
   e. SR ou PSW

**Exercice 2**

1) Quelle est la taille de la mémoire centrale en octets dont la taille du registre RA est de 16 bits et dont les mots mémoires sont de 16 bits ?
2) Si la taille de la mémoire centrale est égale à 512 KO sachant que l'octet est adressable quelle est la taille du registre d'adresse ?
3) Sachant que le bus d'adresse du processeur est de 16 bits avec un alignement à l'octet, quelle est la taille de l'espace mémoire maximum que celui-ci peut adresser ?

**Exercice 3**

Si le registre d'adresse comporte 32 bits, calculer :
1) le nombre de mots adressables si un mot = 1octet.
2) la taille de la mémoire en MO en GO .
3) le nombre de mots adressables si un mot = 4 octets.
4) la taille de la mémoire pour ce dernier cas.
5) quelle est la plus haute adresse dans chaque cas.

**Exercice 4**

Lesquelles de ces organisations mémoires sont possibles, impossibles ? Lesquelles vous semblent raisonnables, pourquoi ? Calculer la taille de la mémoire correspondante en Ko.
1) 10 bits d'adresse, 1024 mots et 8 bits par mot
2) 10 bits d'adresse, 1024 mots et 16 bits par mot
3) 9 bits d'adresse, 1024 mots et 8 bits par mot
4) 11 bits d'adresse, 1024 mots et 16 bits par mot
5) 10 bits d'adresse, 16 mots et 1024 bits par mot

**Exercice 5**

1) La RAM d'une machine contient 64 mots mémoire et elle peut exécuter 13 opérations (arithmétique et logique)
   a) Donner la taille du CO.
   b) Donner la taille minimale du RI.
   c) Si la taille de cette RAM est égale à 192 octets. Donner la taille réelle de RI.

2) Une machine peut exécuter 11 opérations codées de 0000 à 1010 et le CO contient 00011
   a) Donner la taille du RI.
   b) Donner le nombre de mot mémoire au maximum de la RAM.
   c) Supposons que le RI peut prendre les cas suivants :
   1011110110001010001
   1000111110000010101
   0110110110000100011
   Précisez les cas possibles et les cas impossibles ?
3) Soit une machine dotée d'une mémoire centrale de 512 Kmots de 32bits. Sachant que l'instruction-type occupe un mot mémoire, quelle taille proposeriez-vous pour le registre CO et RI.

**Exercice 6**

1) Réaliser une mémoire de 1Ko (la taille d'un mot est de 8 bits) en utilisant des boîtiers de taille 256 mots de 8 bits ?
**2)** On veut réaliser une mémoire de 1Ko (la taille d'un mot est de 16 bits) en utilisant des boîtiers de taille 1Ko mots de 4 bits ) ?

**Exercice 7**

1. Déterminer le taux de transfert d'un bus de 16 bits, cadencé à une fréquence de 133 MHz.
2. Un ordinateur est équipé d'un processeur Pentium 4 à 3,6 GHz fonctionnant à une fréquence de carte mère de 800 MHz. Déterminer le taux de transfert maximal du bus processeur sachant que la quantité de données pouvant être transférées simultanément est de 64 bits.

**Exercice 8**

Soit une mémoire centrale inconnue (on ne connait pas ses paramètres). Nous souhaitons connaitre les paramètres de cette mémoire en la testant avec un processeur cadencé à 1.4 GHz. Ce processeur nous informe qu'il lui faut attendre 10 cycles d'horloge pour obtenir des données en bloc de 2 octets et patienter 15 nouveaux cycles pour demander une nouvelle donnée. Pour lire l'intégralité de cette mémoire, le processeur met 4.3 secondes.
1) Combien dure un cycle d'horloge de ce processeur ?
2) Quel est le temps d'attente, temps de cycle mémoire et le format de données de cette mémoire
3) Calculez la capacité de cette mémoire et son débit avec la grandeur adéquate.

**Exercice 9**

1) La valeur 32 bits 30A79847 est stockée à l'emplacement 1000. Quelle est la valeur de l'octet à l'adresse 1002 si le système est de type big endian ? et s'il est de type little endian ?
2) Supposez un ordinateur 32 bits big-endian et stocke la valeur 32 bits 0xFFFFFFCA en mémoire à partir de l'adresse 0x1FE. Donnez le contenu de la mémoire à partir de cette adresse.

**Exercice 10**

1) On considère un cache dont les lignes font 32 octets, combien de bits sont utilisés pour déterminer l'octet dans la ligne de cache qui pointe une adresse.
2) On considère un cache de 64 KO, combien de lignes peut contenir la cache si la longueur est de 128 octets ?

**Exercice 11**

On se donne une cache qui possède des lignes de 64 octets, on suppose que la mémoire centrale prend un temps de 10 cycles pour répondre à chaque requête mémoire et retourne 2 octets de données en réponse à chaque requête, combien de temps faut-il pour extraire une ligne de cache ?

**Exercice 12**

On se donne un cache direct d'une capacité de 16 Ko dont les lignes sont de 32 octets.

1) Combien de bits, sont utilisés pour déterminer l'octet qu'une opération mémoire référence dans une ligne de cache ?
2) Combien de bits, sont utilisés pour sélectionner la ligne dans le cache ?

**Exercice 13**

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 1 Go, combien d'adresses correspondront à un bloc du cache L1 ?

**Exercice 14**

Le circuit intégré 74381 est une UAL qui permet de réaliser des opérations logiques ou arithmétiques sur deux nombres A et B de 4 bits.

1- Préciser les fonctions arithmétiques réalisées par ce circuit.
2- Préciser les fonctions logiques réalisées par ce circuit.
3- Compléter le tableau ci-dessous :

| S(S2S1S0) | A(A3A2A1A0) | B(B3B2B1B0) | F'F3F2F1F0) | Opérations réalisées |
|---|---|---|---|---|
| 011 | 1001 | …….. | 1101 | …….. |
| …….. | …….. | 0110 | 1110 | F=A OR B |
| 010 | 1001 | 1010 | …….. | …….. |
| …….. | …….. | 0101 | 0110 | F=A XOR B |
| 111 | 1001 | 1011 | …….. | …….. |

**Exercice 15**

On désire réaliser la somme arithmétique de deux nombres de 8 bits chacun en utilisant le circuit 74181 **A=**A7A6A5A4A3A2A1A0 et **B=**B7B6B5B4B3B2B1B0.

1- Déterminer le nombre de circuits 74181 nécessaires pour réaliser cette addition.
2- Donner les configurations nécessaires des circuits utilisés.
3- Tracer le schéma de montage en précisant les niveaux logiques des différentes entrées et de la sortie si A=(128)10 et B=(64)10
4- Réaliser la fonction $F = (A + \bar{B})\ plus\ 1$ en complétant le tableau suivant :

| Fonction | S(S2S1S0) | M | Cn | A(A3A2A1A0) | B(B3B2B1B0) | F'F3F2F1F0) |
|---|---|---|---|---|---|---|
| Soustraction | | | | 0101 | 0011 | |

# TD N°3

**Exercice 1**

1. Quels sont les différents niveaux de programmation ?
2. Sur la plupart des machines, on retrouve deux modèles de programmations. Lesquels ?
3. Quels sont les différents modes d'adressage qu'on retrouve en général dans un microprocesseur ?
4. Quelle est le rôle de l'UCC ? Quelles ont les principaux éléments d'une UCC ?
5. Décrire le cycle de recherche et le cycle d'exécution d'une instruction.
6. Expliquer le principe du pipeline ? Qu'est-ce qu'il nécessite ? Quelles sont ses limites ? Est quel gain en performance existe-t-il ?

**Exercice 2**

On veut exécuter l'instruction qui effectuée l'opération A + B sachant que:

- L'adresse en mémoire centrale de l'instruction est 100,
- celle du premier opérande est 150,
- celle du deuxième opérande est 200
- celle du résultat 250;

Donnez le contenu du registre d'instruction (RI) et celui de la mémoire centrale pour A=36 et B=33.

1. On considère la machine à quatre adresses.
2. On considère la machine à deux adresses.

**Exercice 3**

On considère l'instruction qui effectue la multiplication des deux nombres A et B. Sachant que le contenu du compteur ordinal avant l'exécution de l'instruction est 100, et que l'adresse du premier opérande est 120 le deuxième opérande se trouve dans l'accumulateur

1. Donnez le contenu du RI.
2. Donnez le contenu de la mémoire centrale pour A=3 et B=2.
3. Donnez le contenu de l'accumulateur avant et après exécution.

**Exercice 4**

Quel est l'état de la pile après l'exécution de chaque instruction de la séquence suivante ?

```
PUSH  #4
PUSH  #7
PUSH  #8
ADD
PUSH  #10
SUB
MUL
POP   X
```

**Exercice 5**

En utilisant le jeu d'instructions de la machine à zéro adresse donnez la suite d'instructions pour l'exécution de l'opération suivante : y = a + (b * c) – d, sachant la pile est vide au départ. Donnez le contenu de la pile pour les valeurs suivantes : a=5, b=3, c=7, d=8.

**Exercice 6**

On considère un processeur à registres généraux et possède des instructions à 3 adresses. Ecrire une suite d'instructions, pour effectuer l'opération (a + b) * (a + c) avec a, b, c des constantes ; Utilisez uniquement trois registres R1, R2, R3.

**Exercice 7**

Un processeur possède les registres R0 à R9.
On veut calculer l'expression : $((R1 * R2 - R3) / (R1 + R4 + R5)) + R6 + R2$ et mettre le résultat dans R0.
1) Le processeur possède des instructions à trois adresses. Écrivez la suite d'instructions correspondant au calcul voulu
2) Le processeur possède des instructions à deux adresses. Écrivez la suite d'instructions correspondant au calcul voulu
3) Le processeur possède un registre accumulateur qui est source et destination de toutes les opérations. Les instructions à une donnée sont :

| | |
|---|---|
| ADD rX | (Acc ←Acc+rX) |
| SUB rX | (Acc ←Acc-rX) |
| MUL rX | (Acc ←Acc*rX) |
| DIV rX | (Acc ←Acc/rX) |
| LOAD rX | (Acc ←rX) |
| STORE rX | (rX ←Acc) |

Écrivez la suite d'instructions correspondant au calcul voulu. On veut toujours avoir le résultat dans R0, qui n'est pas l'accumulateur.

**Exercice 8**

Comparez le style de la programmation sur des ordinateurs ayant respectivement des instructions à zéro (machine à pile), un, deux ou trois opérandes en écrivant quatre programmes qui calculent l'expression: X = (A + B * C) / (D - E * F) . La comparaison portera sur la taille du programme et le trafic mémoire (recherche des opérandes).

**Exercice 9**

Voici la liste des instructions d'un ordinateur fictif :

4 instructions UAL :
- ADD Rd,Rs
- SUB Rd,Rs
- MULT Rd, Rs
- DIV Rd,Rs

2 instructions de chargement :
- LOAD Rd, [Rs]
- LOAD Rd, Immédiat

1 instruction de rangement : STORE [Rd], Rs
1 instruction de branchement : BEQZ Rs, Immédiat
L'ordinateur possède 8 registres 32 bits et les valeurs immédiates sont encodées sur 32 bits.

1. Dans un premier temps, on utilise un seul format d'instruction de longueur fixe et à champ fixe. Enumérez les différents champs d'une instruction ainsi que le nombre de bits accordés à chaque champ. Quelle est la longueur totale d'une instruction ?
2. Afin de réduire la longueur des programmes, on utilise deux formats d'instructions de longueur fixe à champ fixe. Décrivez ces deux formats.

**Exercice 10**

La machine (machine fictive) sur laquelle porte l'exercice comporte les éléments suivants :
- une mémoire principale de 32 octets.
- un registre de travail de 8 bits (nommé ACC) qui contient une des entrées de l'UAL avant exécution de l'opération et le résultat de l'opération ensuite.
- une Unité Arithmétique et Logique (UAL) sachant exécuter les opérations suivantes : Addition, Soustraction.

- un registre de 2 bits (nommé FLAGS) qui sont positionnés en fonction du résultat de l'UAL (Bit0 = 1 si le résultat est nul, Bit1 = 1 si le résultat est négatif)
- un registre compteur d'instruction (IP) qui contient l'adresse de la prochaine instruction à exécuter
- un registre instruction (INST) qui contient le code de l'instruction courante.

1. Combien de bits faut-il pour représenter une adresse ?
2. Supposant que le nombre d'instructions disponible est de 8 et que chaque instruction est représenté sous la forme ***Codop Adresse***, combien de bits faut-il pour coder une instruction ?

La liste des instructions est la suivante : addition, soustraction, chargement du registre ACC, rangement de ACC, branchement, branchement si nul, branchement si négatif, fin.

1. Donner un code binaire (Codop) et un nom symbolique à chacune d'entre elles.
2. Ecrire un programme (en langage symbolique) dont la première ligne sera à l'adresse 10h de la mémoire, permettant d'ajouter deux nombres stockés aux adresses 01h et 02h et rangeant le résultat à l'adresse 03h.
3. Ecrire l'algorithme, puis le programme (commençant en 10h réalisant la multiplication (par additions successives) de deux nombres strictement positifs (N1 et N2) stockés en 01h et 02h et rangeant le résultat en 03h. (on supposera que les constantes 0 et 1 sont rangées respectivement en 05h et 06h). Représenter sous forme de tableau, l'évolution des registres ACC, FLAGS et IP au cours de l'exécution du programme (en prenant N1=2 et N2=3).

**Exercice 11**

1. Donnez le contenu du registre de destination dans chaque cas qui suit :
   MOVE R1, #200 (adressage immédiat)
   MOVE R1, 200 (adressage direct)
   MOVE R2, (R1) (adressage indirect)
   Le contenu de l'adresse 200 est 230, et celui de 230 est 400.

2. Pour accéder à une donnée en mémoire, un processeur ne possède que le mode d'adressage indirect. On souhaite simuler les autres modes d'adressage accédant à la mémoire. En distinguant à chaque fois deux cas (instruction à deux ou trois données), écrivez les instructions correspondant aux exemples suivants utilisant d'autres modes d'adressage :
   a) Adressage indirect avec déplacement : MOVE R1, 8(R2).
   b) Adressage indirect indexé : MOVE R1, (R2, R3).
   **c)** Adressage indirect indexé avec déplacement : MOVE R1, 8(R2, R3)

**Exercice 12**

Un CPU possède des instructions de branchement Inconditionnel : JMP ;
Conditionnel : JE, JNE, JL, JG, JLE, JGE qui compare l'opérande avec 0 et décide du saut à une adresse. Ecrire le programme assembleur qui correspond à :

| while (i==0) { … } | do { … } while (i==0) | for (i=0 ;i<max ;i++) { … } | if(i==0) { … } else { … } |
|---|---|---|---|
| ? | ? | ? | ? |

# T D N°4

**Exercice 1**

1. Quel est le rôle d'un processeur, quelles sont ses caractéristiques ?
2. Quelles est la différence entre l'architecture CISC et RISC ?
3. Expliquer le principe du pipeline ? Qu'est-ce qu'il nécessite ? Quelles sont ses limites ? Est quel gain en performance existe-t-il ?

**Exercice 2**

Soit une architecture de Van Neumann munie d'un processeur 24 bits (adresse) cadencé à 2.4 GHz. La lecture d'un bloc 3 octets de la mémoire vers le processeur se fait en 4 cycles, le traitement des 3 octets par le processeur prend 9 cycles. En considérant que la mémoire centrale a la capacité maximale et en considérant le traitement parfais, sans aucune perturbations ni aléas du système.

1. Quelle est la capacité maximum de la mémoire en octets, Ko, Mo et Go ?
2. Calculer le temps d'un cycle de ce processeur.
3. Que représente le bloc de 3 octets ?
4. Combien de temps faut-il pour traiter toutes les données contenues dans la mémoire ?
5. Soit un autre processeur cadencé à 1.3 GHz qui lit 4 octets en 2 cycles et il les traite en 5 cycles. Combien de temps lui faut-il pour traiter la même quantité de données que le processeur précédent ? Quel est le processeur le plus rapide ?

**Exercice 3**

Soit une architecture munie d'un processeur cadencé à 2 GHz, une mémoire centrale de 1 Go ayant un temps d'accès de 20 ns, un temps de cycles mémoire de 25 ns et un format de données de 32 bits. Le processeur exécute un programme dans lequel il lit 8 octets en 6 cycles d'horloge et traite ces 8 octets en 10 cycles d'horloge.
NB : les 6 cycles de lecture représentent un temps supplémentaire pour organiser la donnée au niveau du processeur.

1. Combien de temps faudra-t-il au processeur pour traiter 1 Go de données ?
2. Combien est le temps d'attente (temps pendant lequel le processeur attend l'arrivée des données) ?
3. Quel est le taux d'attente par rapport au temps total du traitement ?
4. Que conclure de ce résultat ? Comment y remédier ?

**Exercice 4**

Un processeur exécute une instruction de 4 octets toutes les nanosecondes. De plus, 20 % de ces instructions font référence à une donnée de 4 octets en mémoire. Quel est le débit nécessaire entre le processeur et la mémoire ?

**Exercice 5**

Une mémoire RAM, reliée au processeur, a un temps de cycle de 20 ns pour le premier accès et de 10 ns pour les trois accès suivants qui sont accélérés. Chaque accès récupère 8 octets. Quelle est la bande passante du transfert d'informations entre la mémoire et le processeur ?

**Exercice 6**

Supposons qu'on a cinq types d'instructions : Load (5 cycles), Store (4 cycles), R-type (4 cycles), Branch (3 cycles), Jump (3 cycles). Si un programme a : 50% load instructions, 25% store instructions, 15% R-type instructions, 8% branch instructions, 2% jump instructions. Déterminez le CPI effectif ?

**Exercice 7**

Soit un processeur possédant une horloge de 400 MHz a été utilisé pour exécuter un programme contenant les instructions suivantes :

| Type d'instruction | Nombre d'instruction | nombre de cycle d'horloge |
|---|---|---|
| Integer Arithmetic | 45000 | 1 |
| Data transfer | 32000 | 2 |
| Floating point | 15000 | 2 |
| Control transfer | 8000 | 2 |

Déterminer le CPI, Mips (Millions d'instruction par second), et le temps d'exécution de ce programme.

# T D N°5

**Exercice 1**

1. Initialiser le registre $a1 à 0
2. Initialiser le registre $a1 à 0x4567 (valeur 16 bits)
3. Initialiser le registre $a1 à 0x4567ABCD (valeur 32 bits)

**Exercice 2**

1. Programmez en assembleur MIPS la fonction F= 2*X + 2*Y - Z . On suppose que la variable X se trouve dans le registre $t0, la variable Y dans $t1, la variable Z dans $t2 et que le résultat de la fonction F se trouve dans le registre $v0.
2. Programmez en assembleur MIPS les expressions (variables à 32 bits) :
   - **a)** $S = a^* x^2 + b * x + c$
   - **b)** $S = a^2 + b^2 + c^2$

**Exercice 3**

Supposez le code MIPS suivant, qui reçoit deux entrées dans les registres $10 et $11 et produit une sortie dans le registre $20.

```
        add $1,$0,$0
lo:     beq $11,$0,fin
        add $1,$1,$10
        subi $11,$11,1
        j lo
fin:    addi $1,$1,100
        add $20,$1,$0
```

a) Décrivez en une phrase la fonction du programme.
b) Quelle est la valeur de $20 à la fin du programme, si $10=6 et $11=4 au début du programme ?

**Exercice 4**

Supposez que $a0=0x1234 et $a1=0x3. Quelle est la valeur du registre $t0 après l'exécution du programme MIPS suivant ?

```
        add $t0, $0, $0
toto:   add $t0, $t0, $a0
        addi $a1, $a1, -1
        bne $a1, $zero, toto
        srl $t0, $t0, 4
```

**Exercice 5**

Supposez le code assembleur MIPS suivant :

```
        lw $t0,0($a0)
        lw $t1,0($a1)
        bne $t0,$t1,toto
        add $v0,$t0,$zero
        j titi
toto:   add $v0,$t1,$zero
titi:
```

a-) Quelle est la fonction réalisée, exprimée dans un langage de haut niveau ?

**Exercice 6**

Ecrire une séquence d'instructions MIPS pour tester un nombre dans le registre $4 s'il est pair on met 1 dans le registre $5 sinon 0.

**Exercice 7**

**1.** Ecrire une séquence d'instructions assembleur pour permuter le contenu de 2 registres, On utilisera un registre intermédiaire.

**2.** Permuter deux registres sans utiliser un registre intermédiaires (penser à XOR).

**3.** Ecrire une séquence d'instructions assembleur qui permute le contenu de 2 variables 32 bits. Même chose pour deux variables 16 bits, puis 8 bits.

**Exercice 8**

Opérations logiques bit à bit. Pour chaque question, donner la réponse en langage machine MIPS.

1. Multiplier $t0 par 8.
2. Inverser le bit 4 de $t0.
3. Si $t0 est pair, $t0 = 0, sinon $t0 = 1.
4. Mettre à un le bit 7 de $t0.

**Exercice 9**

Ecrire une séquence d'instructions MIPS pour calculer la somme des entiers de 0 à 10.

**Exercice 10**

Supposez un processeur MIPS big-endian qui exécute le programme suivant :

```
        xor $t0, $t0, $t0
        addi $t1, $t0, 0x50
a:      lw $t2, 0($t1)
        beq $t2, $zero, c
        slt $t3, $t0, $t2
        beq $t3, $zero, b
        lw $t0, 0($t1)
b:      addi $t1, $t1, 4
        j a
c:      sw $t0, 0($t1)
```

| | |
|---|---|
| 0x04 | 0x4E |
| 0xFC | 0x4F |
| 0x01 | 0x50 |
| 0x23 | 0x51 |
| 0x45 | 0x52 |
| 0x00 | 0x53 |
| 0x00 | 0x54 |
| 0x00 | 0x55 |
| 0x11 | 0x56 |
| 0x34 | 0x57 |
| 0x00 | 0x58 |
| 0x00 | 0x59 |
| 0x00 | 0x5A |
| 0x00 | 0x5B |
| 0xFC | 0x5C |
| 0x0A | 0x5D |

Figure 1

a) Expliquez quelle est la fonction du programme
b) Si l'état initial de la mémoire est celui de la figure 1, quel est l'état de la même portion de mémoire après l'exécution du programme ?

**Exercice 11**

Ecrire des fonctions en MIPS pour :

1. Vérifier si un nombre est valable dans un système de numération à base B
2. Transformer un nombre entier d'une base B en nombre entier sur 16 bits
3. Ecrire un nombre entier 16 bits dans une base B

Le nombre entier est lu sous forme de chaîne de caractères (pour les bases > 10, on utilise les chiffres A, B, C, D, E, F )

**Exercices 12**

Traduire en assembleur MIPS les programmes C suivants :

**1.**
```
int pgcd(int a, int b )
{
while (a != b )
    if (a < b)
            b = b - a ;
    else
            a = a - b ;
return( a ) ;
}

void main() {
int a , b ;
printf(" entrer a ");
scanf("%d", &a) ;
printf(" entrer b ");
scanf("%d", &b) ;
printf(" pgcd = %d ",pgcd(a,b) );
}
```

**2.**
```
int pgcd(int a , int b)
{
    if ( !b ) return a ;
    return pgcd(b , a mod b ) ;
}
```

**Exercice 13**

a-) Codez les instructions MIPS suivantes en langage machine :

```
add     $s0, $sl, $s2
sub     $t0, $t3, $t5
addi    $t0, $s3, -12
lw      $t2, 32($0)
```

b-) Traduire les instructions suivantes (écrite en code machine) en assembleur MIPS : (0x02324020), (0x00114100) et (0x3908003F).

**Exercice 14**

1. Nous supposerons pour l'exemple suivant que l'instruction à coder se trouve en $pc=0x00400020, et que les étiquettes auxquelles on saute se trouvent en plus = 0x0040003C, et moins = 0x00400018.

```
jalr $ra, $k1
j plus
bne $v0, $v1, plus
bltz $v1, moins
syscall mfc0 $ra, $13
```

2. Traduire les instructions suivantes (écrite en code machine) en assembleur MIPS :

```
0x00031880
0x00021080
0x3C16FFFF
0x00A21026
0x80E30006
0xAE2200C8
```

# Références


1. Emmanuel Lazard, « Architecture de l'ordinateur - Synthèse de cours et exercices corrigés ». Édition : Pearson Education, 1re édition, 2006.
2. Alain Cazes, Joëlle Delacroix, « Architecture des machines et des systèmes informatiques », 4ème édition, Collection : Informatique, Dunod, 2011.
3. Andrew S. Tanenbaum, Todd Austin, « Structured Computer Organization", Pearson, 2012.
4. Paolo Zanella, Yves Ligier, Emmanuel Lazard, « Architecture et technologie des ordinateurs : Cours et exercices », Collection : Sciences Sup, Dunod, 5ème édition, 2013.
5. Robert Strandh, Irene Durand, Architecture de l'ordinateur - Cours condensé avec exercices corrigés, Dunod, 2005.
6. Emmanuel Viennet, Architecture des ordinateurs, IUT de Villetaneuse Département GTR, 1999-2000.
7. Emmanuel Lazard, Architecture de l'ordinateur, Université Paris-Dauphine mars 2011.
8. Mehemmel A., Architecture Des Ordinateurs, Centre Universitaire Bordj Bou Arreridj. 2008/2009.
9. Messain N., Architecture des ordinateurs, Support de cours, Université Ferhat Abbas Sétif. 2012/2013.
10. 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


Attention !

Cette bibliographie n'est pas exhaustive, elle pourra être mise à jour par la suite.