

Chapitre N° 03

Architecture d'un Microprocesseur (CPU).

1. Introduction

Un microprocesseur ou processeur ou encore **CPU** (*Central Processing Unit* pour *Unité Centrale de Traitement* en français) est l'unité intelligente de traitement des informations. C'est un circuit intégré complexe à très grande échelle d'intégration. Il est chargé d'organiser les tâches précisées par le programme, de les décoder et d'assurer leur exécution. Il doit aussi prendre en compte les informations extérieures au système et assurer leur traitement.

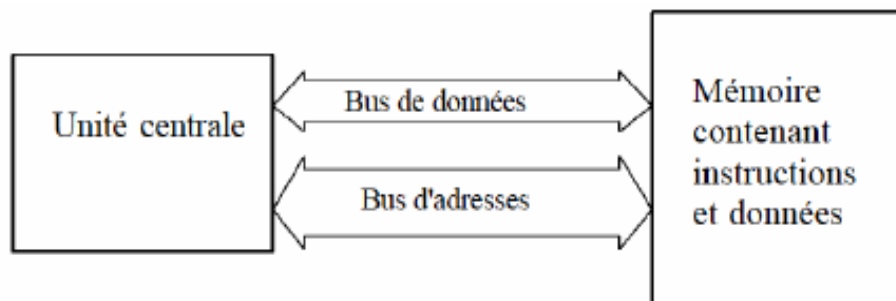
2. Architecture d'un système à microprocesseur

Pour l'organisation des différentes unités d'un système à microprocesseur, il existe deux architectures possibles :

a. Architecture de Von Neuman

La mémoire de programme, la mémoire de données, et les périphériques d'entrées/sorties partagent le même bus s'adresses et de données.

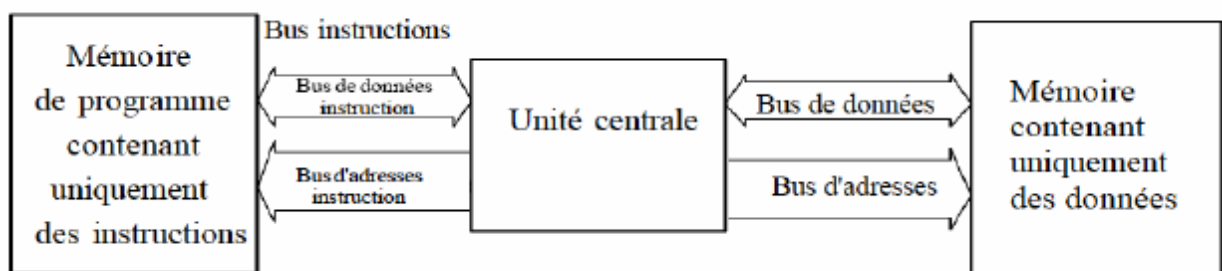
L'exécution d'une instruction nécessite donc plusieurs échanges de données sur le seul et unique bus dévolu à cet usage puisqu'il faut tout d'abord aller chercher le code de l'instruction puis le ou les données qu'elle doit manipuler.



Architecture de Von Neuman

b. Architecture de Harvard

Cette architecture sépare systématiquement la mémoire de programme de la mémoire des données : l'adressage de ces mémoires est indépendant. Ce type d'architecture est utilisé sur des microcontrôleurs qui ont connu un développement important ces dernières années.



Architecture de Harvard

Quoique cette architecture puisse être complexe mais elle est performante : Gain en terme de vitesse d'exécution des programmes :

L'exécution d'une instruction ne fait plus appel qu'à un seul cycle machine puisque l'on peut simultanément, grâce au deux bus, rechercher le code de l'instruction et la ou les données qu'elle manipule.

3. Modèle de base d'une CPU (Architecture interne)

Un microprocesseur est construit autour de deux éléments principaux :

- Une unité de Contrôle (Commande) (*UC*)
- Une Unité Arithmétique et Logique (*UAL*)
- Les deux unités sont associées à des registres chargées de stocker les différentes informations à traiter.
- L'ensemble de ces trois éléments sont reliés entre eux par des bus interne permettant les échanges d'informations.

3.1. Unité de contrôle

L'unité de contrôle est une circuit logique séquentiel chargée de séquencer l'algorithme et de générer les signaux de contrôle pour piloter les éléments du chemin de données. Elle envoie des commandes à l'unité de traitement qui va exécuter les instructions.

L'unité de contrôle est constituée essentiellement de **Bloc logique de commande (ou séquenceur)** qui organise l'exécution des instructions au rythme d'une horloge. Il élabore tous les signaux de synchronisation internes ou externes (bus de commande) du microprocesseur en fonction des divers signaux de commande provenant du décodeur d'instruction ou du registre d'état par exemple. Il s'agit d'un automate réalisé soit de façon câblée (obsolète), soit de façon micro-programmée, on parle alors de micro-microprocesseur.

L'unité de contrôle contient aussi : le **compteur de programme**, le **registre d'instruction** et le **décodeur d'instruction**.

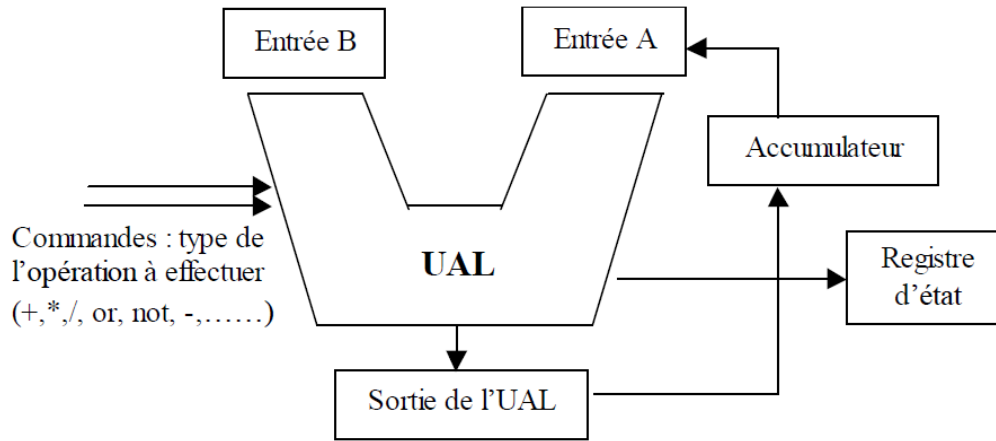
3.2. Unité Arithmétique et Logique

Comme son nom l'indique, cette unité dispose de circuits réalisant deux types d'opérations :

- *Opérations logiques* : tel que ET, OU, NOT OU EXCLUSIF, de même les opérations de rotation et de décalage, et de comparaison.
- *Opérations arithmétiques* : elles incluent l'addition et la soustraction, la multiplication et la division.

L'UAL comporte deux registres : l'**accumulateur** et le **registre d'état (FLAG)**.

En entrée, l'UAL, reçoit ses opérandes (les octets qu'elle manipule) du bus de données. Celles-ci peuvent provenir de registres ou de la mémoire, elle reçoit d'autre part, les commandes permettant d'activer les opérations, venant de l'unité de contrôle. En sortie, on a les résultats des opérations et les conditions qui sont en fait les entrées de l'unité de contrôle. A la fin d'une opération, l'UAL peut aller modifier certains bits du registre d'état (FLAG).



Structure de l'unité arithmétique et logique (UAL).

3.3. Les registres

Un registre est un espace mémoire interne au processeur. On distingue deux types : à usage général qui permettent à l'UAL de manipuler des données et les registres d'adresses qui sont connectés au bus d'adresse.

a) L'accumulateur

Il s'agit d'un registre d'usage général servant à stocker un opérande au début d'une opération arithmétique, les résultats intermédiaires, ou le résultat à la fin de l'opération provenant de l'unité arithmétique et logique.

Ils évitent des appels fréquents à la mémoire, réduisant ainsi les temps de calcul. Donc la plupart des opérations arithmétiques et logiques se font dans l'accumulateur.

b) Le registre d'état (FLAG)

Il est généralement composé de 8 bits ou 16 bits à considérer individuellement. Chacun de ces bits est un indicateur dont l'état dépend du résultat de la dernière opération effectuée par l'UAL. On les appelle indicateur d'état ou flag ou drapeaux (Retenue, débordement, zéro, ...).

Par exemple, quand le résultat d'une opération est trop grand pour être contenu dans le registre cible (celui qui doit contenir le résultat de l'opération), un bit spécifique du registre d'état (le bit **OF**) est mis à 1 pour indiquer le débordement.

On peut citer aussi par exemple les indicateurs de :

- Retenue (carry : **CF**)
- Signe (Sign : **SF**)
- Zéro (**ZF**)

c) Le compteur de programme (PC : *Program Counter*)

Appelé aussi Compteur Ordinal (CO) : Il est constitué par un registre dont le contenu est initialisé avec l'adresse de la première instruction du programme. Il contient toujours l'adresse de la prochaine instruction à exécuter. Autrement dit, il doit indiquer au processeur la prochaine instruction à exécuter.

Ce registre est constamment modifié après l'exécution de chaque instruction afin qu'il pointe sur l'instruction suivante.

d) Le registre d'instruction et le décodeur d'instruction

Chacune des instructions à exécuter est transférée depuis la mémoire dans le registre instruction puis elle est décodée par le décodeur d'instruction.

▪ *Le registre d'instruction :*

Pour exécuter une instruction le microprocesseur transmet l'adresse se trouvant dans le registre compteur de programme à la mémoire, la mémoire retourne au microprocesseur l'octet adressé par ce dernier (le code de l'instruction) celui-ci sera stocker dans un registre appelé registre d'instructions (RI), donc Le RI contient la prochaine instruction à être exécutée par le processeur.

Cette instruction sera acheminée (par un bus de données) au décodeur d'instructions qui sera chargé de l'interpréter.

▪ *Le décodeur d'instruction :*

C'est lui qui va interpréter l'instruction contenue dans le registre d'instruction (RI). C'est-à-dire qu'elle est l'opération à effectuer (Addition, branchement etc...) Et comment aller chercher les opérandes requises pour cette opération (par exemple, les nombres à additionner).

Le décodeur d'instructions communique alors avec l'unité de commandes et de contrôles qui pourra déclencher les événements en conséquence.

e) *Registres d'adresses :*

Ces registres servent à gérer l'adressage de la mémoire. En effet le processeur peut utiliser un registre ou une paire de registres pour accéder à un emplacement mémoire, et puisque les registres peuvent être incrémenter ou décrémenter donc on peut accéder facilement à des données qui se trouvent en mémoire d'une manière adjacente.

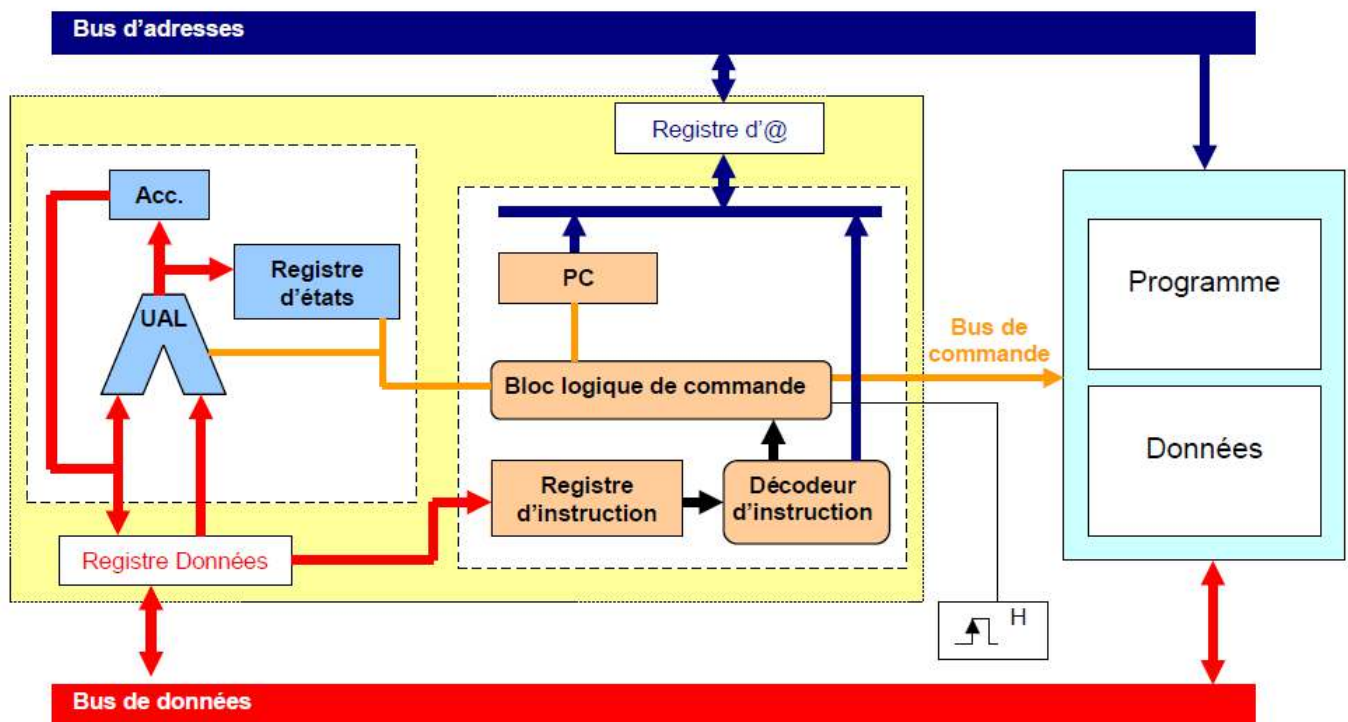
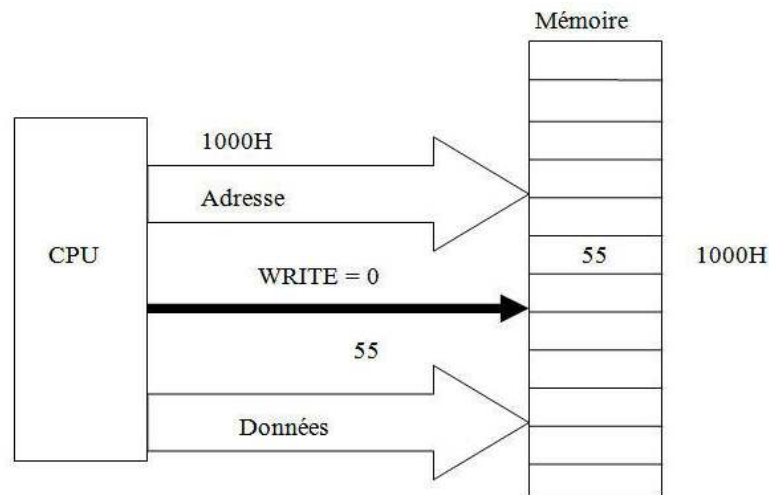


Schéma fonctionnel d'un système à Microprocesseur.

4. Fonctionnement d'un microprocesseur

4.1. L'écriture en mémoire (WRITE)

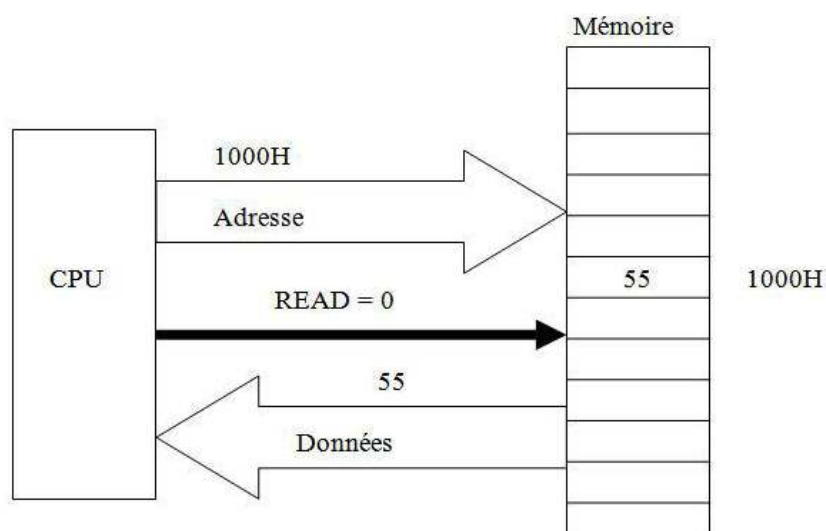
Pour écrire une donnée dans la mémoire le microprocesseur doit placer l'adresse de la donnée sur le bus d'adresses, puis il place la donnée sur le bus de données et enfin génère le signal **WRITE** (ordre d'écriture dans la mémoire).



4.2. La lecture de la mémoire (READ)

Pour lire une donnée de la mémoire le microprocesseur dépose son adresse sur le bus d'adresses puis génère le signal **READ**, alors la donnée sera acheminée vers le microprocesseur à travers le bus de données. La donnée sera stockée dans un registre dans le microprocesseur.

Si la donnée est un code opératoire d'une instruction alors elle sera logée dans le registre d'instructions sinon elle sera logée dans un registre de données (en général l'accumulateur).



4.3. Communication avec les Entrées/Sorties

Les données échangées entre un périphérique et le processeur transitent par l'interface associé à ce périphérique. L'interface possède de la mémoire tampon pour stocker les données échangées et les informations pour gérer la communication avec le périphérique :

- des informations de commande, pour définir le mode de fonctionnement de l'interface : sens de transfert (entrée ou sortie), mode de transfert des données (par scrutation ou interruption), etc. Ces informations de commandes sont communiquées à l'interface lors de la phase d'initialisation de celle-ci, avant le début du transfert.

- des informations d'état, qui mémorisent la manière dont le transfert c'est effectué (erreur de transmission, réception d'informations, ...etc.). Ces informations sont destinées au processeur.

Lors de l'exécution des instructions d'entrées/sorties, le processeur met à 1 sa borne IO/M et présente l'adresse E/S sur le bus d'adresse. Le signal IO/M indique aux circuits de décodage d'adresses qu'il ne s'agit pas d'une adresse en mémoire principale, mais de l'adresse d'une interface d'entrées/sorties.

4.4. Accès direct à la mémoire (DMA)

Lorsqu'un transfert en mémoire est nécessaire de la mémoire RAM à un port d'E/S, la CPU lit le premier octet en mémoire et le charge dans l'un des registres du microprocesseur. La CPU écrit ensuite l'octet rangé précédemment sur le port d'E/S approprié.

Il en résulte que le microprocesseur effectue des opérations de lecture et d'écriture répétées. Ainsi un certain temps est perdu entre le traitement de chaque octet. Pour remédier à ce problème, une procédure est mise au point pour l'accès direct à la mémoire (Direct Memory Access), qui permet de transférer des données de la mémoire RAM au port d'E/S sans passer par le microprocesseur. Pour cela, un contrôleur DMA, qui reprend le rôle de la CPU, c'est à dire qu'il gère les transferts de la RAM aux ports d'E/S.

4.5. Les interruptions

Les interruptions permettent au matériel (périphérique) de communiquer avec le processeur. Dans certains cas, on désire que le processeur réagisse rapidement à un événement extérieur, mais la majorité de ces périphériques n'ont besoin du microprocesseur qu'à certains moments, alors ils génèrent une demande d'interruption.

Une interruption est signalée au processeur par un signal électrique sur une borne spéciale. Lors de la réception de ce signal, le processeur "traite" l'interruption dès la fin de l'instruction qu'il était en train d'exécuter. Le traitement de l'interruption consiste soit :

- à l'ignorer et passer normalement à l'instruction suivante : interruptions masquables.
- à exécuter un traitant d'interruption (interrupt handler). Interruptions non masquables.

Parfois le microprocesseur est sollicité par plusieurs interruptions en même temps, pour répondre à ces appels un ordre de priorité est souvent pris en compte pour leurs traitements.

Les interruptions sont de deux types :

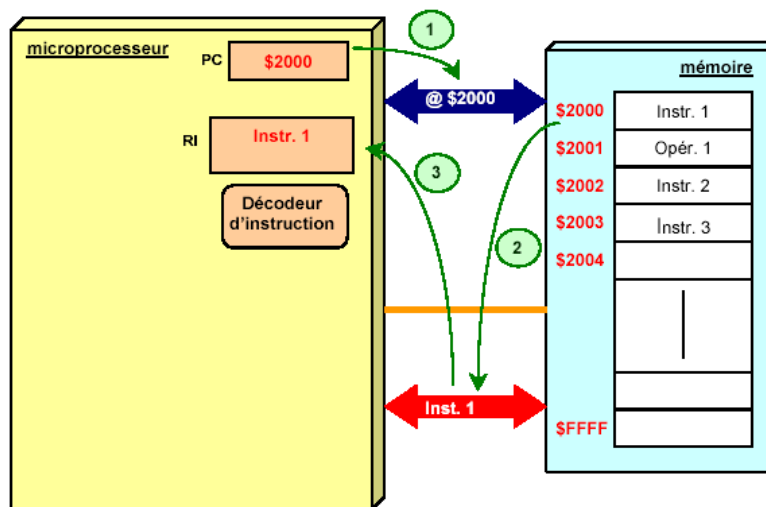
- Interruption matérielle.
- Interruption logicielle.

5. Principe d'exécution de programmes (*Cycle d'exécution d'une instruction*)

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

Phase 1 : Recherche de l'instruction à traiter

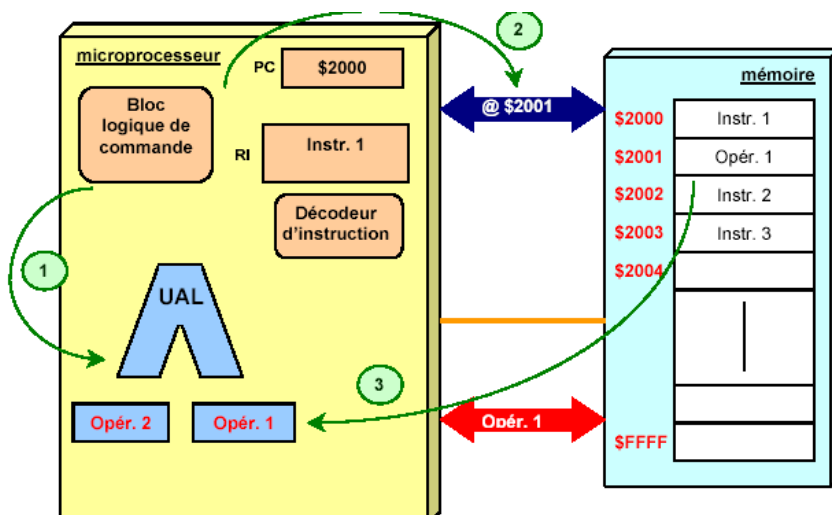
1. Le PC (*Program Counter*) contient l'adresse de l'instruction suivante du programme. Cette valeur est placée sur le bus d'adresses par l'unité de commande qui émet un ordre de lecture.
2. Au bout d'un certain temps (temps d'accès à la mémoire), le contenu de la case mémoire Sélectionnée est disponible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



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

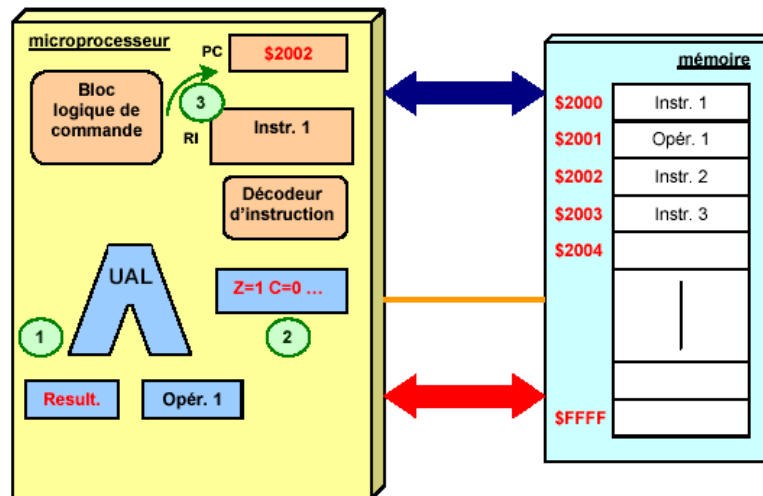
Le registre d'instruction contient maintenant le premier mot de l'instruction qui peut être codée sur plusieurs mots. Ce premier mot contient le code opératoire qui définit la nature de l'opération à effectuer (addition, rotation,...) et le nombre de mots de l'instruction.

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



Phase 3 : Exécution de l'instruction

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

**6. Jeux d'instructions**

Le jeu d'instructions décrit l'ensemble des opérations élémentaires que le microprocesseur pourra exécuter. Il va donc en partie déterminer l'architecture du microprocesseur à réaliser et notamment celle du séquenceur.

6.1. Types d'instructions

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

- *Transfert de données* pour charger ou sauver en mémoire, effectuer des transferts de registre à registre, ...etc.
- *Opérations arithmétiques* : addition, soustraction, division, multiplication
- *Opérations logiques* : ET, OU, NON, NAND, comparaison, test, etc...
- *Contrôle de séquence* : branchement, test, ...etc.

6.2. Codage d'instructions

Les instructions et leurs opérandes (paramètres) sont stockés en mémoire principale. La taille totale d'une instruction (nombre de bits nécessaires pour la représenter en mémoire) dépend du type d'instruction et aussi du type d'opérande. Chaque instruction est toujours codée sur un nombre entier d'octets afin de faciliter son décodage par le processeur. Une instruction est composée de deux champs :

- *Le code instruction*, qui indique au processeur quelle instruction réaliser
- *Le champ opérande* qui contient la donnée, ou la référence à une donnée en mémoire (son adresse).

Exemple :

Code instruction	Code opérande
1001 0011	0011 1110

6.3. Mode 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 :

- *L'adressage de registre* où l'on traite les données contenue dans un registre
- *L'adressage immédiat* où l'on définit immédiatement la valeur de la donnée
- *L'adressage direct* où l'on traite une donnée en mémoire

Selon le mode d'adressage de la donnée, une instruction sera codée par 1 ou plusieurs octets.

6.4. Temps d'exécution

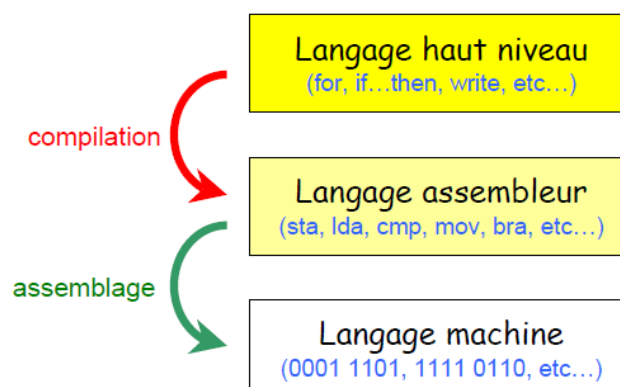
Chaque instruction nécessite un certain nombre de cycles d'horloges pour s'effectuer. Le nombre de cycles dépend de la complexité de l'instruction et aussi du mode d'adressage. Il est plus long d'accéder à la mémoire principale qu'à un registre du processeur. La durée d'un cycle dépend de la fréquence d'horloge du séquenceur.

6.5. Langage de programmation

Le langage machine est le langage compris par le microprocesseur. Ce langage bas niveau est difficile à maîtriser puisque chaque instruction est codée par une séquence propre de bits. Afin de faciliter la tâche du compilation programmeur, on a créé différents langages plus ou moins évolués.

Le langage assembleur est le langage le plus « proche » du langage machine. Il est composé par des instructions en général assez rudimentaires que l'on appelle des mnémoniques. Ce sont essentiellement des opérations de transfert de données entre les registres et l'extérieur du microprocesseur (mémoire ou périphérique), ou des opérations arithmétiques ou logiques. Chaque instruction représente un code machine différent. Chaque microprocesseur peut posséder un assembleur différent.

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



Exemple de programme :

Code machine (68HC11)		Assembleur (68HC11)	Langage C
@00	C6 64	LDAB #100	A=0 ;
@01	B6 00	LDAA #0	for (i=1 ; i<101 ; i++) A=A+i ;
@03	1B	ret ABA	
@04	5A	DECB	
@05	26 03	BNE ret	

7. Notion d'architecture RISC et CISC

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

- L'architecture **CISC** (*Complex Instruction Set Computer*)
- L'architecture **RISC** (*Reduced Instruction Set Computer*)

7.1. L'architecture CISC

Par le passé la conception de machines CISC était la seule envisageable. En effet, vue que la mémoire travaillait très lentement par rapport au processeur, on pensait qu'il était plus intéressant de soumettre au microprocesseur des instructions complexes. Ainsi, plutôt que de coder une opération complexe par plusieurs instructions plus petites (qui demanderaient autant d'accès mémoire très lent), il semblait préférable d'ajouter au jeu d'instructions du microprocesseur une instruction complexe qui se chargerait de réaliser cette opération.

De plus, le développement des langages de haut niveau posa de nombreux problèmes quant à la conception de compilateurs. On a donc eu tendance à incorporer au niveau processeur des instructions plus proches de la structure de ces langages.

C'est donc une architecture avec un grand nombre d'instructions où le microprocesseur doit exécuter des tâches complexes par instruction unique. Pour une tâche donnée, une machine **CISC** exécute ainsi un petit nombre d'instructions mais chacune nécessite un plus grand nombre de cycles d'horloge. Le code machine de ces instructions varie d'une instruction à l'autre et nécessite donc un décodeur complexe (micro-code)

7.2. L'architecture RISC

Des études statistiques menées au cours des années 70 ont clairement montré que les programmes générés par les compilateurs se contentaient le plus souvent d'affectations, d'additions et de multiplications par des constantes. Ainsi, 80% des traitements des langages de haut niveau faisaient appel à seulement 20% des instructions du microprocesseur. D'où l'idée de réduire le jeu d'instructions à celles le plus couramment utilisées et d'en améliorer la vitesse de traitement.

C'est donc une architecture dans laquelle les instructions sont en nombre réduit (chargement, branchement, appel sous-programme). Les architectures **RISC** peuvent donc être réalisées à partir de séquenceur câblé. Leur réalisation libère de la surface permettant d'augmenter le nombre de registres où d'unités de traitement par exemple. Chacune de ces instructions s'exécutent ainsi en un cycle d'horloge. Bien souvent, ces instructions ne disposent que d'un seul mode d'adressage. Les accès à la mémoire s'effectuent seulement à partir de deux instructions (Load et Store). Par contre, les instructions complexes doivent être

réalisées à partir de séquences basées sur les instructions élémentaires, ce qui nécessite un compilateur très évolué dans le cas de programmation en langage de haut niveau.

7.3. Comparaison

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.

Architecture RISC	Architecture CISC
instructions simples ne prenant qu'un seul cycle	instructions complexes prenant plusieurs cycles
instructions au format fixe	instructions au format variable
décodeur simple (câblé)	décodeur complexe (microcode)
beaucoup de registres	peu de registres
seules les instructions LOAD et STORE ont accès à la mémoire	toutes les instructions sont susceptibles d'accéder à la mémoire
peu de modes d'adressage	beaucoup de modes d'adressage
compilateur complexe	compilateur simple