Chapitre 3 : Notions sur les instructions d'un ordinateur

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

1. Introduction aux instructions d'un ordinateur

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

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

Un ensemble d'instructions forme un programme dont l'exécution nécessite l'exécution de toutes les instructions qui le composent. Les différentes étapes suivies lors de l'exécution d'une instruction forment le cycle d'exécution de cette dernière.
Nous allons présenter dans les sections suivantes quelques détails sur les instructions et leurs phases d'exécution, le principe de compilation et d'assemblage, l'UCC et l'UCC pipeline ainsi que l'horloge et le séquenceur.

2. Langage de haut niveau, assembleur, langage machine

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

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

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

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

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

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

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

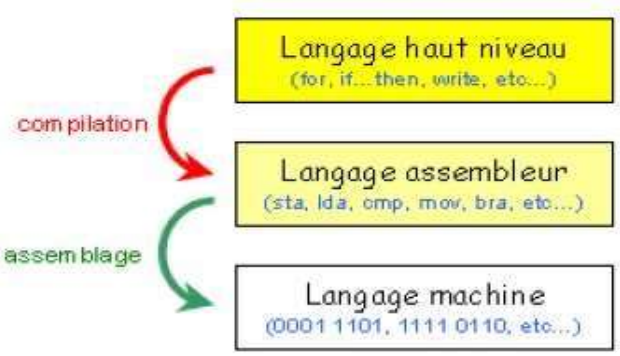
3- Les instructions machines usuelles (arithmétiques, logiques, de comparaison, chargement, rangement, transfert, sauts,…)

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

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

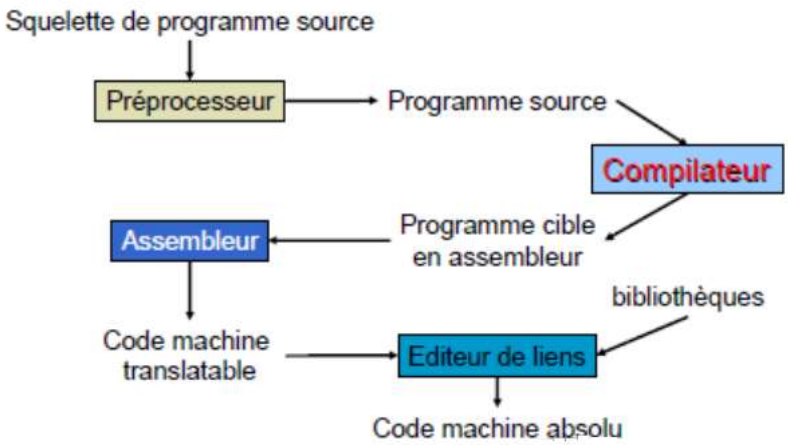
4- Principe de compilation et d'assemblage (notions de base)

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



Compilation et assemblage

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



Principe de compilation

Exemples :

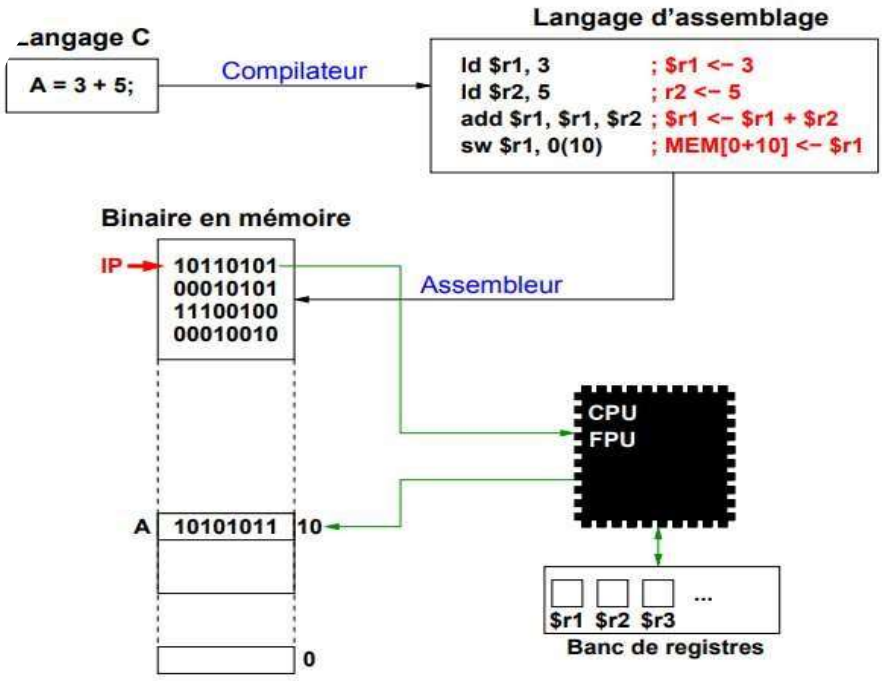
1. L'**expression** $y = x + 1$ est traduite en **assembleur** :

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

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

91 00 08 10 39 00 09 64 10 99

2. Codage des instructions machine



Exemple de compilation et d'assemblage

5- L'unité de contrôle et de commande

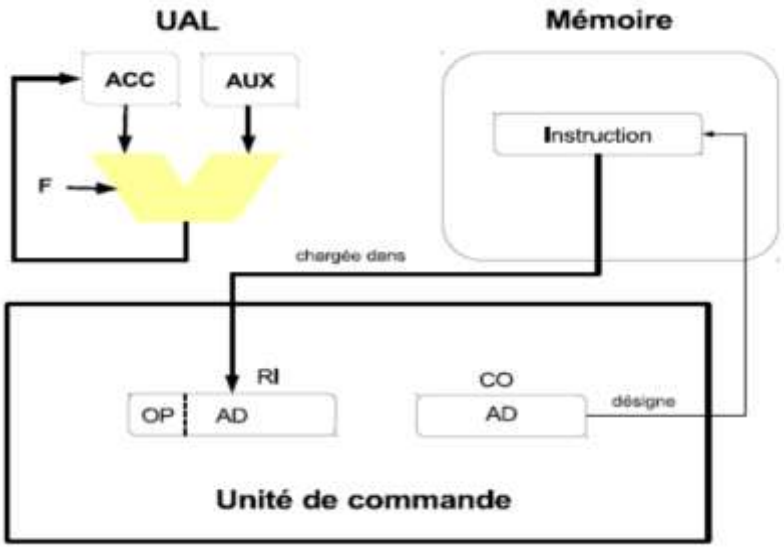
Principe de fonctionnement de l'UCC

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

Répéter

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

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



Fonctionnement de l'UCC

La structure de l'UCC :

L'UCC est composée des principaux éléments suivants :

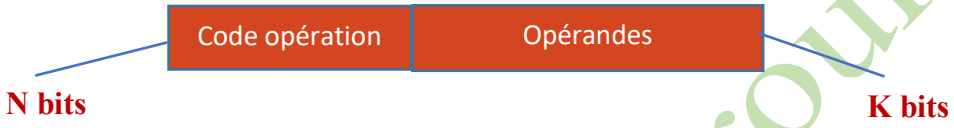
- **un registre instruction RI**, contenant l'instruction qui doit être exécutée
- **un compteur ordinal CO (PC)**, contenant l'adresse de la prochaine instruction à exécuter.
- **un registre adresse RA**, contenant l'adresse de la donnée à lire ou à écrire en mémoire.
- **un registre de données RD**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

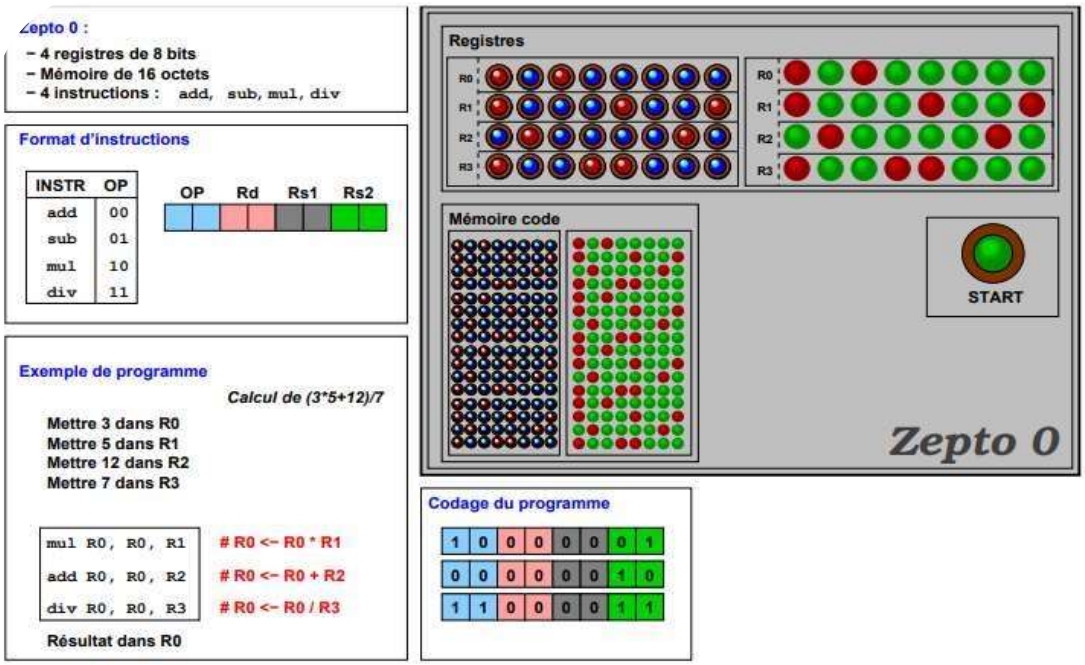
- **un décodeur de fonctions** : qui détermine les opérations à exécuter en fonction du code de l'instruction.
- **un séquenceur :** qui déclenche et coordonne les différentes opérations pour réaliser l'instruction

Codage d'une instruction

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

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





Exemple de codage des instructions

Classification des machines par le nombre d'opérandes

- **Machine à 3 adresses**

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

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

Exemple : SUB X, Y, Z (Z←X-Y)

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

- **Machine à 2 adresses**

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

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

Exemple : ADD X, Y (Y←X +Y)

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

- **Machine à 1 adresse**

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

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

Exemple : ADD X (ACC←(ACC) + X)

Remarque : Ce type de machine est le plus utilisé.

- **Machine à 0 adresse**

Dans cette architecture, les instructions vont directement agir sur la pile.
Les opérandes sont automatiquement chargés depuis le pointeur de pile (**SP**, Stack Pointer), et le résultat est à son tour empilé.

Exemple : L'opération Z = X -Y sera traduite par la séquence suivante :

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

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

Les modes d'adressage

Les modes d'adressage sont un aspect de l'architecture des processeurs et de leurs jeux d'instructions.

Les modes d'adressage définis dans une architecture régissent la façon dont les instructions en langage machine identifient leurs opérandes.

Un mode d'adressage spécifie la façon dont est calculée l'adresse mémoire effective d'un opérande à partir de valeurs contenues dans des registres et de constantes contenues dans l'instruction ou ailleurs dans la machine.

Un mode d'adressage définit la manière dont le microprocesseur va accéder à l'opérande.

• Le champ **opérande** contient la **donnée** ou la référence (adresse) à la donnée.

• Le **code opération** de l'instruction comporte un ensemble de **bits** pour indiquer le **mode d'adressage**.

• Les modes d'adressage les plus utilités sont :

1) Immédiat – 2) Direct – 3) Indirect – 4) Indexé – 5) Relatif

1) L'adressage immédiat

L'opérande existe dans le champ adresse de l'instruction.

Exemple : SUB 20

L'exécution de cette commande va avoir l'effet suivant : ACC←(ACC) - **20**

Si le registre accumulateur contient la valeur **40** alors après l'exécution son contenu sera égal à **20**.

2) L'adressage direct

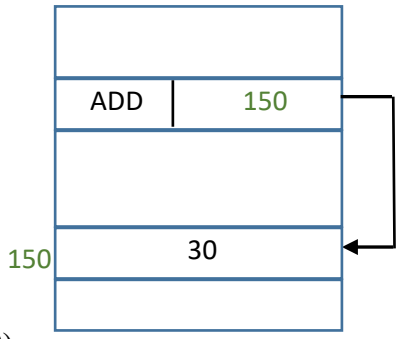
Le champ opérande de l'instruction contient l'adresse de l'opérande (emplacement en mémoire)

Pour réaliser l'opération, il faut le récupérer (lire) l'opérande à partir de la mémoire.

ACC ← (ACC) + (ADR)

Exemple : ADD 150

On suppose que l'accumulateur continent la valeur **20**.



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

3) L'adressage indirect

Le champ adresse contient l'adresse de l'adresse de l'opérande.

Pour réaliser l'opération il faut :

• Récupérer l'adresse de l'opérande à partir de la mémoire.

• Par la suite, il faut chercher l'opérande à partir de la mémoire.

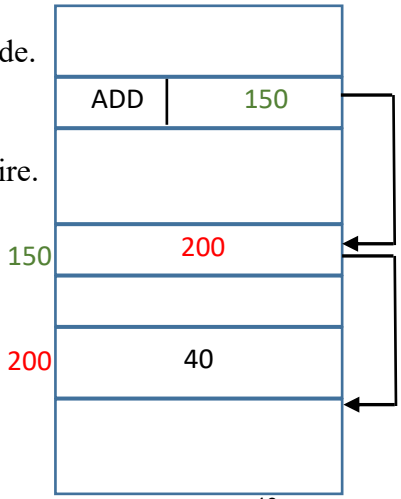
ACC← (ACC) + ((ADR))

Exemple : ADD 150

Initialement l'accumulateur contient la valeur **20**

• Il faut récupérer l'adresse de l'adresse (150).

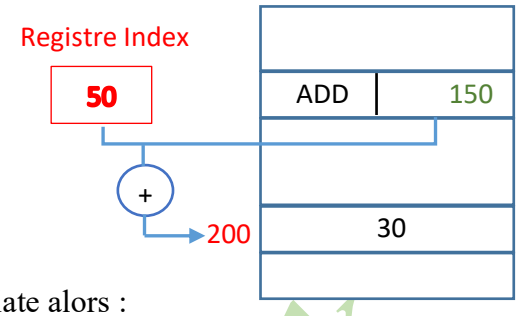
• Récupérer l'adresse de l'opérande à partir de l'adresse **150** (La valeur **200**)



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

4) L'adressage indexé

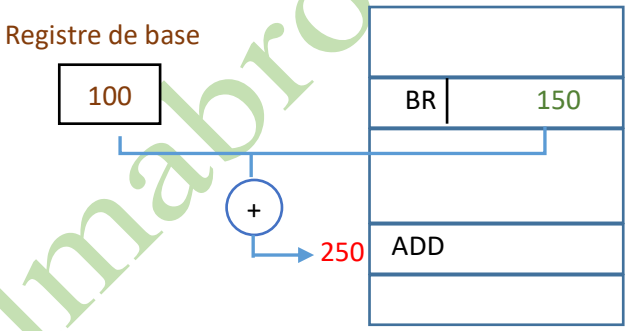
L'adresse effective de l'opérande est relative à une zone mémoire.
• L'adresse de cette zone se trouve dans un registre spécial (registre index).
• Adresse opérande = ADR + (X)



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

5) L'adressage relatif

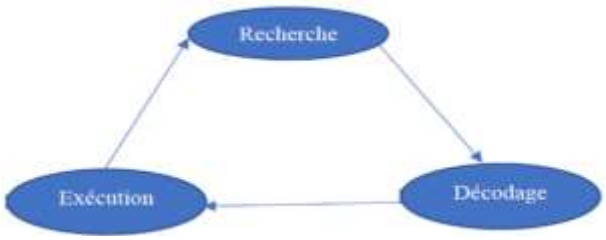
L'adresse effective de l'opérande est relative à une zone mémoire.
• L'dresse de cette zone se trouve dans un registre spécial (**registre de base**).
• Ce mode d'adressage est utilisé pour les instructions de branchement.
Adresse = ADR + (base)



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

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

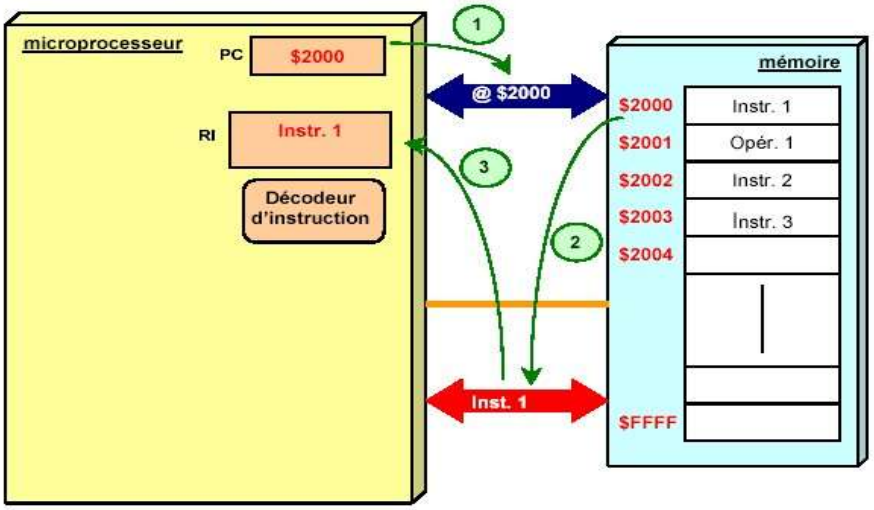
Le microprocesseur ne comprend qu'un certain nombre d'instructions qui sont codées en binaire. Le traitement d'une instruction peut être décomposé en trois phases.



Cycle d'exécution d'une instruction

Phase 1 : Recherche de l'instruction à traiter

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

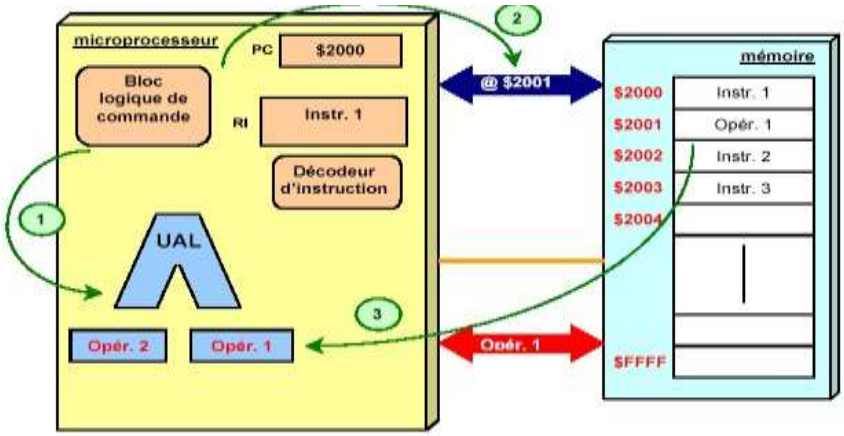


Phase de recherche de l'instruction à traiter

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

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

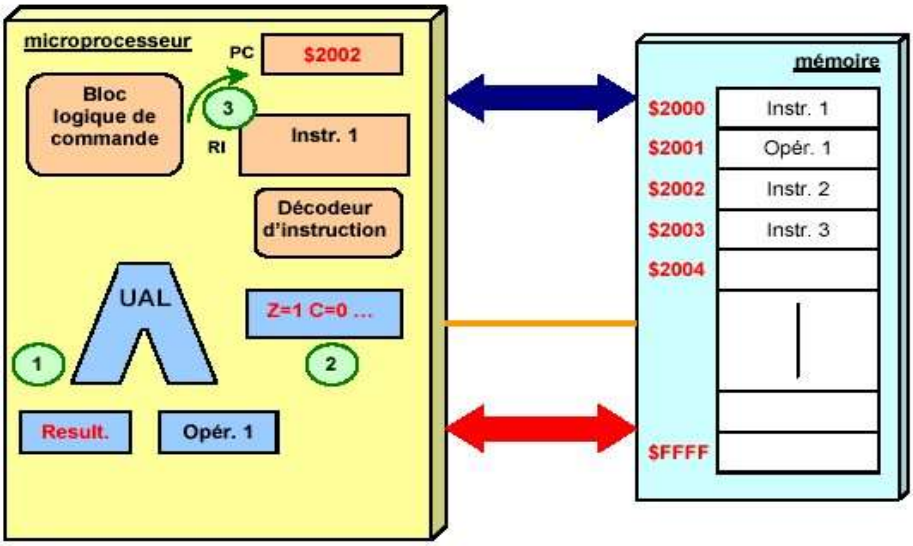
1) L'unité de commande transforme l'instruction en une suite de commandes élémentaires nécessaires au traitement de l'instruction.
2) Si l'instruction nécessite une donnée en provenance de la mémoire, l'unité de commande récupère sa valeur sur le bus de données.
3) L'opérande est stocké dans un registre.



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

Phase 3 : Exécution de l'instruction

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



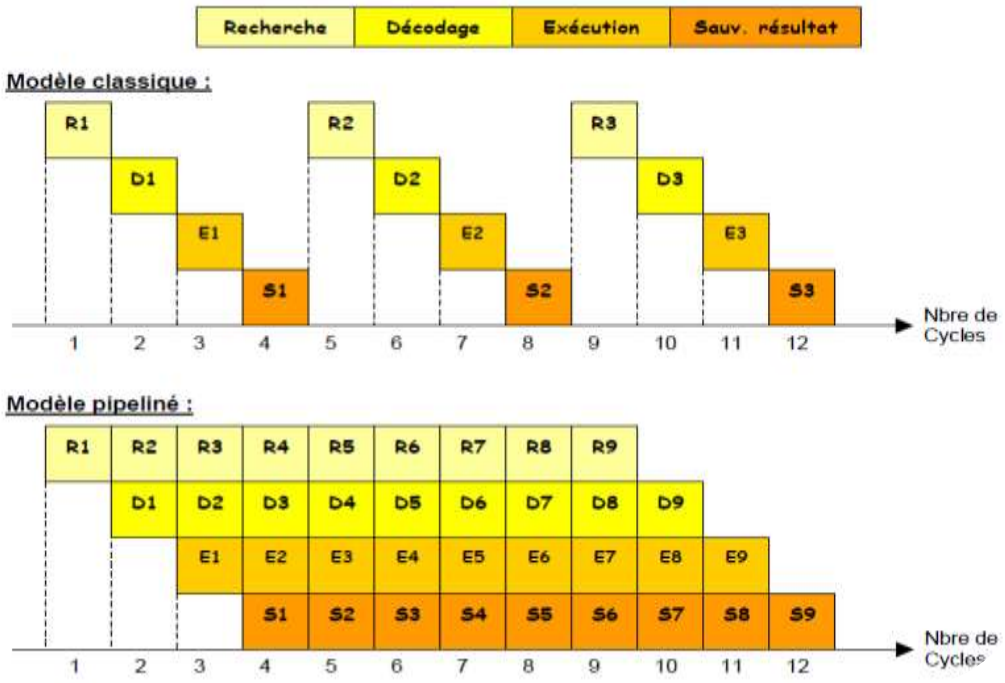
Phase d'exécution de l'instruction

7- UCC pipeline

Principe

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

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



Le modèle classique versus le modèle pipeliné

Gain de performance

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

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

Remarque :

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

Exemples :

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

8- L'horloge et le séquenceur

L'horloge :

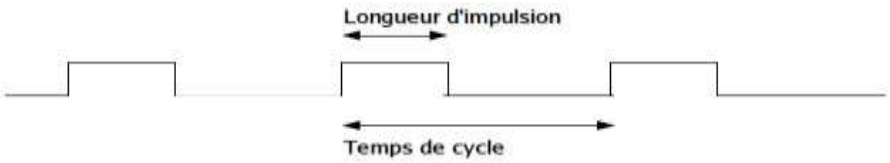
Dans de nombreux circuits numériques, il est essentiel de pouvoir garantir l'ordre dans lequel certains événements se produisent :

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

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

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

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

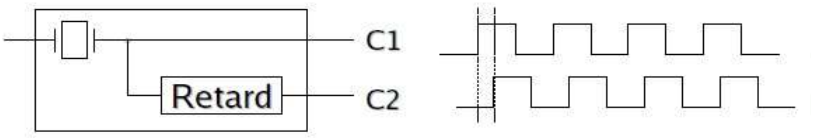


Circuit d'une horloge

Cycles et sous cycles

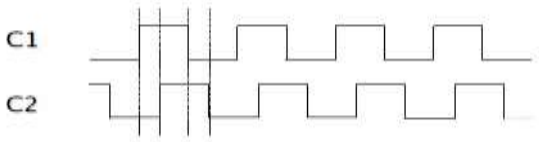
- Dans un ordinateur, de nombreux événements ont à se produire au cours d'un **cycle d'horloge**.
- Si ces événements doivent être séquencés dans un ordre précis, le **cycle d'horloge** doit être décomposé en **sous-cycles.**
- Un moyen classique pour cela consiste à **retarder** la copie d'un signal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

Exemple :



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

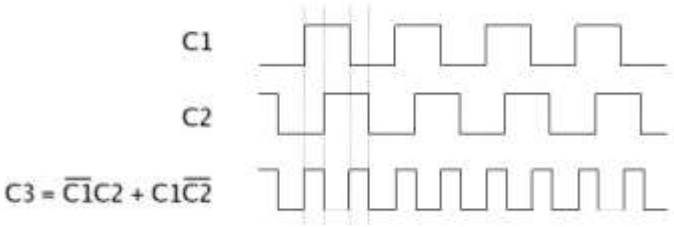
- Fronts montant et descendant de C1
- Fronts montant et descendant de C2



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

- Action possible seulement lorsque C1 est haut

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



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

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

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

Le séquenceur :

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

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

Remarque :

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

9. Conclusion

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

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