Architecture des Ordinateurs

Faculté Polydisciplinaire de Taroudant

GIS4

Contenu du cours

Introduction à l'architecture des ordinateurs

L'architecture d'un ordinateur

- L'architecture des ordinateurs est la discipline qui correspond à la façon dont on conçoit les composants d'un système informatique.
- En informatique, le terme architecture désigne l'organisation des éléments d'un système et les relations entre ces éléments. Il y a :
  - L'architecture matérielle : concerne l'organisation des différents dispositifs physiques que l'on trouve dans un ordinateur. Fonctionnement logique de chaque composant et le dialogue entre les composants.
  - L'architecture logicielle : concerne l'organisation de différents programmes entre eux. Codage de l'information et jeu d'instruction de la machine càd l'ensemble des opérations que la machine peut exécuter

L’architecture d’un ordinateur

- L’architecture d’un ordinateur est la manière dont ses différents composants sont organisés et interconnectés pour permettre son fonctionnement.
- Cette architecture est basée sur un ensemble de circuits électroniques qui peuvent traiter, capter et enregistrer des informations numériques (binaires) via des bus de communication.
- Les bus sont des ensembles de fils électriques qui permettent la circulation de l’information (énergie, données, contrôles) entres les composants.

Ordinateur

- Besoin : Traitement plus complexe et plus rapide.

  ⇒ Automatisation du traitement
- Ordinateur : machine de traitement de l'information.
- Il est capable d'effectuer automatiquement des opérations arithmétiques et logiques à partir de programmes définissant la séquence de ces opérations.
- C'est un ensemble de circuits électroniques permettant de manipuler des données sous forme binaire, ou bits afin d'exécuter des séquences de calculs ou des traitements de tout genre

0rdinateur (peu d'historique)

Claude Shannon: chiffres binaires pour les relations logiques et les calculs logiques et arithmétiques (Tout calcul peut être réalisé avec les 3 opérations logiques de base ET,OU, NON.

Alan Turing: machine universelle ou Machine de Turing décrivant un modèle abstrait du fonctionnement des appareils mécaniques de calcul

$\Rightarrow$ Invente les concepts de programmation et de programme.

JohnVon Neumann: Enregistrer le programme en mémoire

$\Rightarrow$ Architecture de l'ordinateur moderne : l'architecture de Von Neumannm.

Information

- Un ensemble de données ayant un sens précis.
- Des valeurs numériques, textes, images,son, vidéos.
- Des instructions composant un programme.
- Toute information est manipulée sous forme binaire (ou numérique) par un système informatique.

Informatique Système Informatique

- Informatique : Terme provenant des mots ″Information″ et ″automatique″.
- C'est la science du traitement rationnel et automatique de l'information, considérée comme le support des connaissances dans différents domaines.
- Système Informatique : Ensemble des moyens logiciels et matériels nécessaires pour satisfaire les besoins informatiques des utilisateurs.

Programmation

Le fait de réaliser un programme dont l'exécution apporte une solution satisfaisante à un problème donné suivant un algorithme bien précis.

- Elle est effectuée en utilisant un langage de programmation comme le langage machine, l'assembleur ou un langage évolués (traduction de l'algorithme).
- Elle fait partie de l'ingénierie de développement logiciel (implémentation ou code).

Langage de programmation

- C'est l'intermédiaire entre l'humain et la machine, il permet d'écrire, dans un langage proche de la machine mais intelligible par l'humain, toutes les opérations que l'ordinateur doit effectuer.
- Il doit donc respecter une syntaxe stricte.
- Un langage informatique est destiné à décrire l'ensemble des actions consécutives qu'un ordinateur doit exécuter. C'est une façon pratique de donner des instructions à un ordinateur.

Familles de languages de programmation

- Langage fonctionnel: (ou langage procédural) est un langage dans lequel le programme est construit par fonctions, retournant un nouvel état en sortie et prenant en entrée la sortie d'autres fonctions par exemple⇒ diviser un problème complexe en sous-problèmes plus simples.
- Lorsqu'une fonction appelle elle-même, on parle alors de récursivité.
- Langage objet: part du principe que des choses peuvent avoir des points communs, des similarités en elles-mêmes ou en leur façon d'agir. L'idée est regrouper de tels éléments afin d'en simplifier leur utilisation.
  ⇒ Un regroupement est appelé classe, les entités qu'il regroupe sont
- appelées objets.

Programme

- Suite d'instructions dans un langage donné, définissant un des actions spécifiques exécutables par un ordinateur.
  - Programmes systèmes (système d'exploitation gérant différents ressources machine).
  - Programmes d'application (des logiciels de traitements).
- Un programme est composé de deux parties :
  - La partie contenant les données.
  - La partie contenant le code des instructions à exécuter.
- Les instructions sont des opérations de base que l'ordinateur peut traiter comme l'addition, la multiplication la comparaison...

Microprocesseur

- C'est un circuit électronique intégré complexe et miniaturisé contenant plusieurs millions de transistors interconnectés (ex : le Pentium).
- C'est le cœur de l'ordinateur qui permet de traiter et distribuer les informations.
- Il résulte de l'intégration sur une puce de fonctions logiques combinatoires (logiques et/ou arithmétique) et séquentielles (registres, compteur, etc. . .).
- Il exécute les instructions élémentaires au rythme de son horloge interne. (ex : 300 Mhz $\Rightarrow$ 300 millions d'instructions par seconde).

Architecture de Von Neumann-Unités fonctionnelles

Introduction

Il existe deux architectures informatiques, qui diffèrent dans la manière d'accéder aux mémoires :

- L'Architecture de Von Neumann: les programmes et les données sont stockés dans la même mémoire et gérés par le même sous-système de traitement de l'information.
- L'Architecture de Harvard : les programmes et les données sont stockés et gérés par différents sous-systèmes. C'est la différence essentielle entre ces deux architectures.

Dans la suite de ce cours, nous allons détailler l'Architecture de Von Neumann

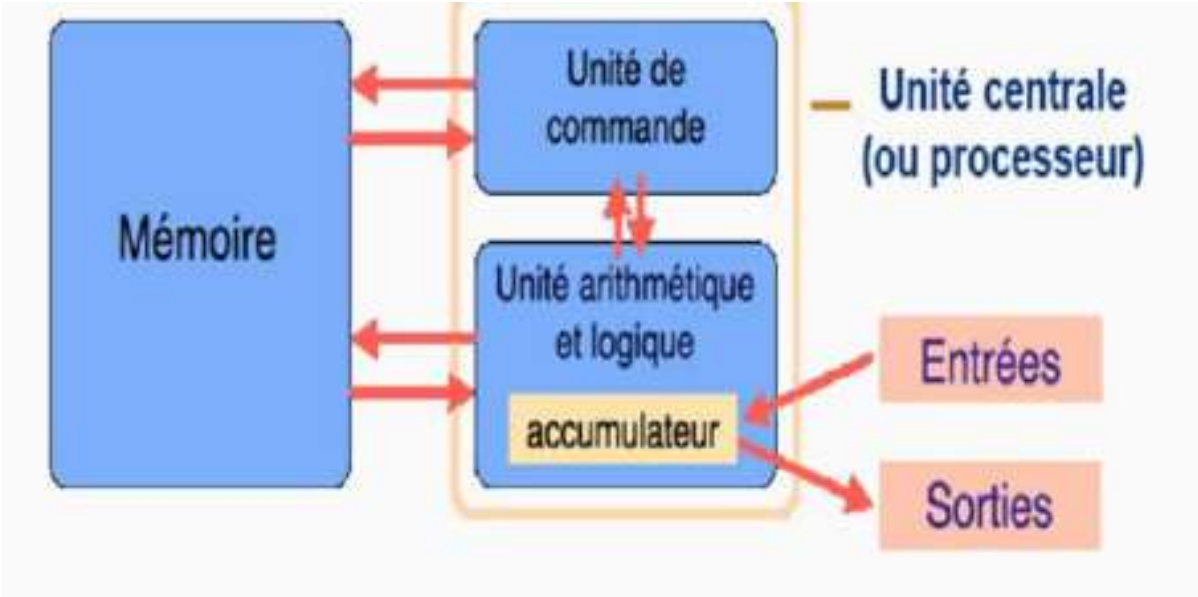
Architecture de Von Neumann

A la fin de 1946 " John Von Neumann " un physicien et mathématicien d'origine Hongroise, propose un modèle d'ordinateur qui fait abstraction du programme et se lance dans la construction d'un EDVAC (Electronic DiscreteVariable Automatic Computer). Il a introduit 2 nouveaux concepts dans le traitement digital de l'information :

- Programme enregistré : Von Neumann, a eu l'idée d'utiliser les mémoires du calculateur pour emmagasiner les programmes : d'où le nom de la machine à programme enregistré donné au nouveau type de calculateur.
- Rupture de séquence : Von Neumann, a eu l'idée de rendre automatique les opérations de décision logique en munissant la machine d'une instruction appelée branchement conditionnelle (ou rupture de séquence conditionnelle).

Architecture de Von Neumann

- L'architecture de Von Neumann est un modèle structurel d'ordinateur dans lequel une unité de stockage (mémoire) unique sert à conserver à la fois les instructions et les données demandées ou produites par le calcul.
- Les ordinateurs actuels sont tous basés sur des versions améliorées de cette architecture.
- Cette architecture peut être représentée sur un schéma simple :



Architecture de Von Neumann

L'architecture de Von Neumann établit la relation entre les différents composants d'un ordinateur. Ces composants sont :

- L'unité centrale de traitement qui contient principalement :
  - l'unité arithmétique et logique (UAL) : elle effectue les opérations de base ;
  - l'unité de commande : s'occupe du séquençage des opérations ;
- La mémoire : stocke les données et le programme ;
- Les entrées/sorties : permettent de communiquer avec l'extérieur.

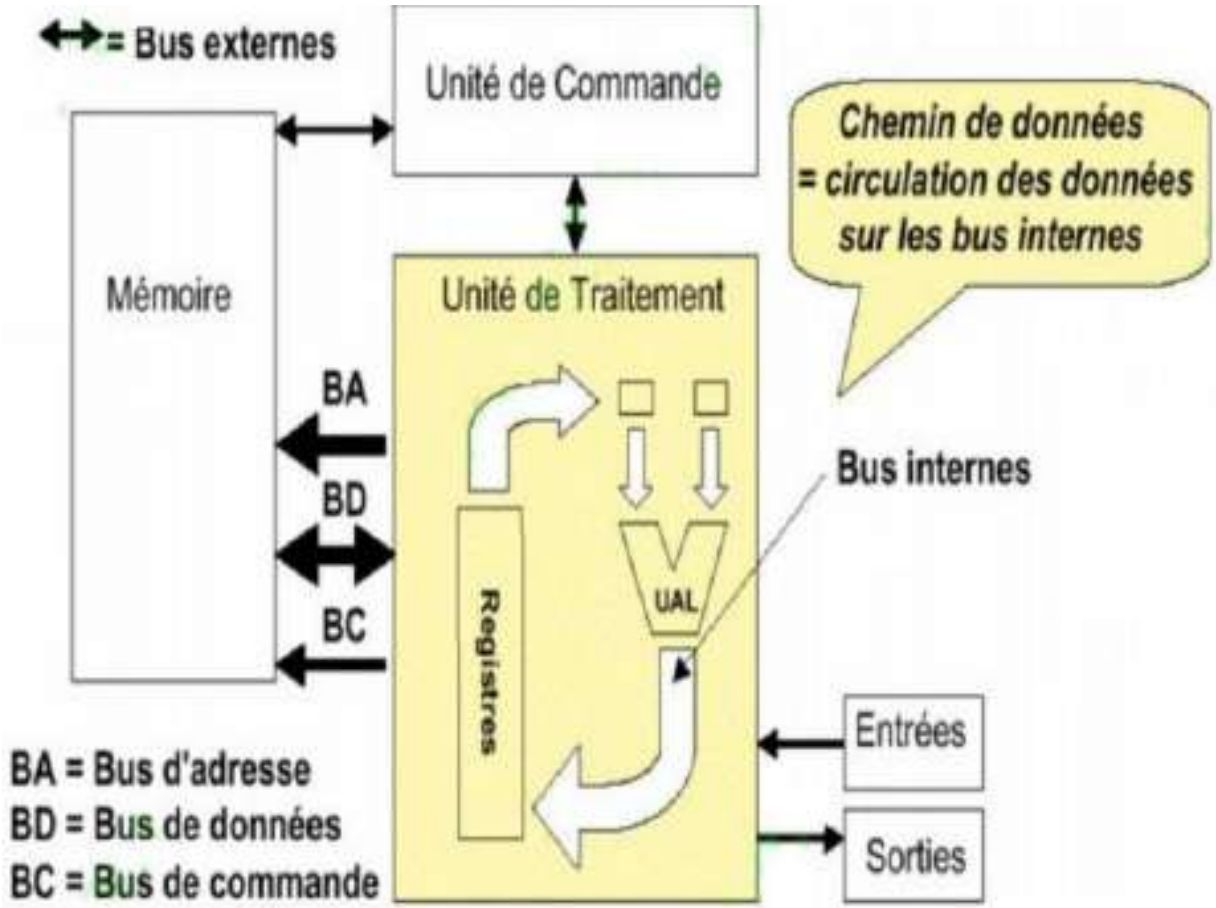
Ces différents éléments sont interconnectés à travers des bus qui constituent le support d'acheminement de l'information entre les différents composants

Unité centrale de traitement

Chargée d'interpréter et d'exécuter les instructions d'un programme, de lire ou de sauvegarder les résultats dans la mémoire et de communiquer avec les unités d'échange. Elle est construite par deux éléments principaux :

- Une unité de commande : permet la lecture en mémoire et le décodage des instructions.
- Une unité de traitement : permet d'exécuter des instructions.

Unité centrale de traitement

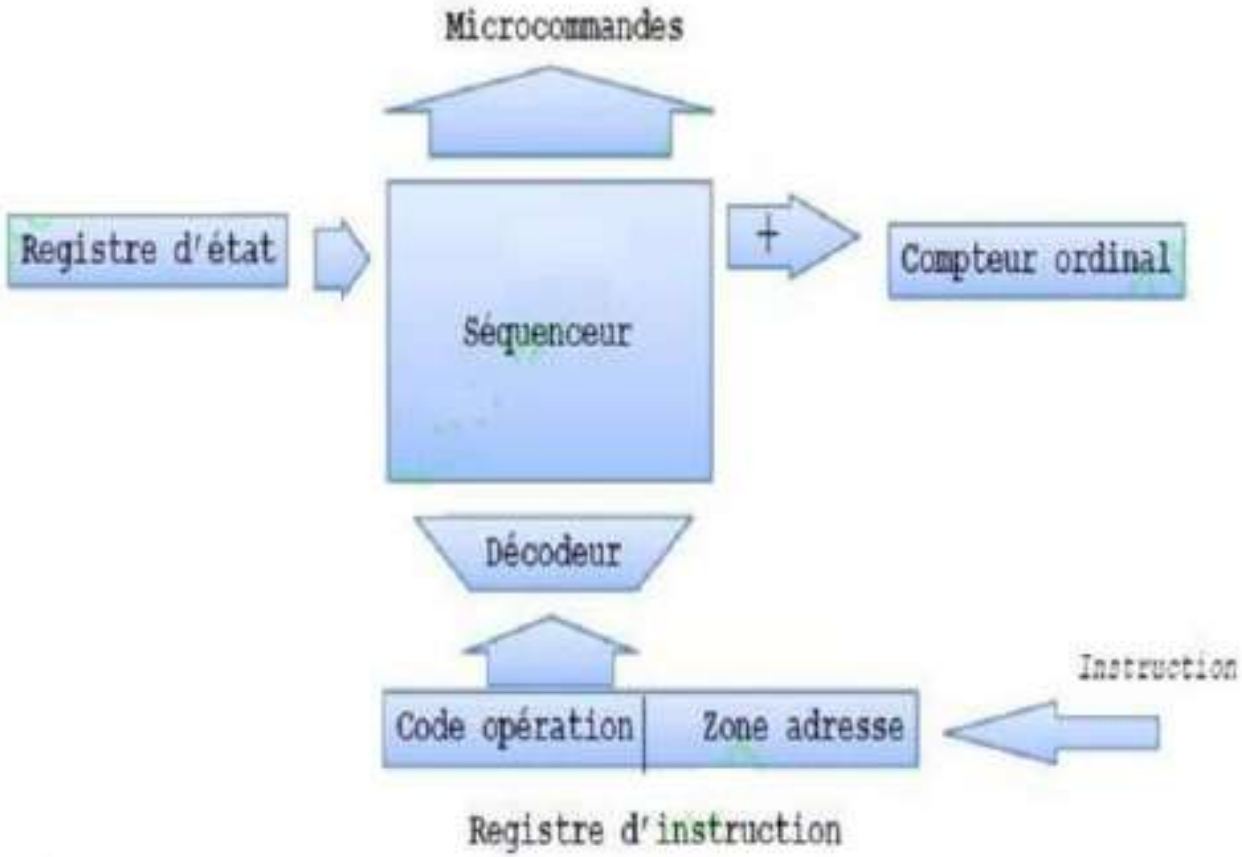


Unité de commande

Description de ”groupes” de choses :

- Elle permet de séquencer le déroulement des instructions.
- Elle effectue la recherche en mémoire de l’instruction.
- Elle assure le décodage de chaque instruction, codée sous forme binaire, pour réaliser son exécution et ensuite préparer l’instruction suivante.
- Elle gère tous les signaux de synchronisation internes ou externes (bus des commandes) au microprocesseur en coordonnant le fonctionnement des autres composants dont les activité sont cadencées par une horloge unique, de fa¸on à ce que tous les circuits électroniques travaillent ensembles.

Unité de commande



Unité de commande

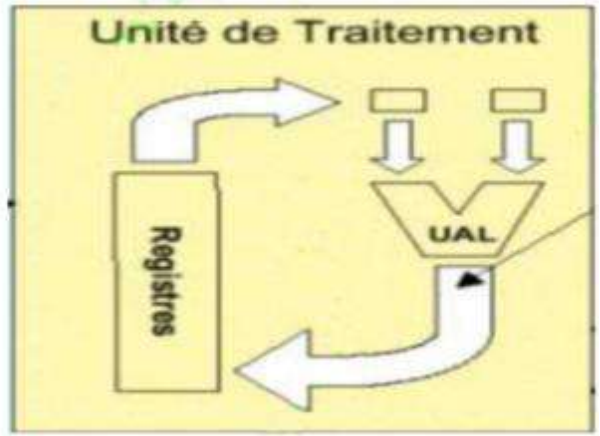
Composants de l'unité de commande

- Le Compteur Ordinal (CO) de programme constitué par un registre dont le contenu est initialisé avec l'adresse de la première instruction du programme. Il contient toujours l'adresse de l'instruction à exécuter.
- Le Registre d'Instruction (RI) et le décodeur d'instruction : chacune des instructions à exécuter est rangée dans le registre d'instruction ensuite elle est décodée par le décodeur d'instruction.
- Bloc logique de commande (ou séquenceur) : Il organise l'exécution des instructions au rythme d'une horloge. Il gère les signaux du microprocesseur

Unité de traitement

Elle regroupe les circuits qui assurent les traitements nécessaires à l'exécution des instructions :

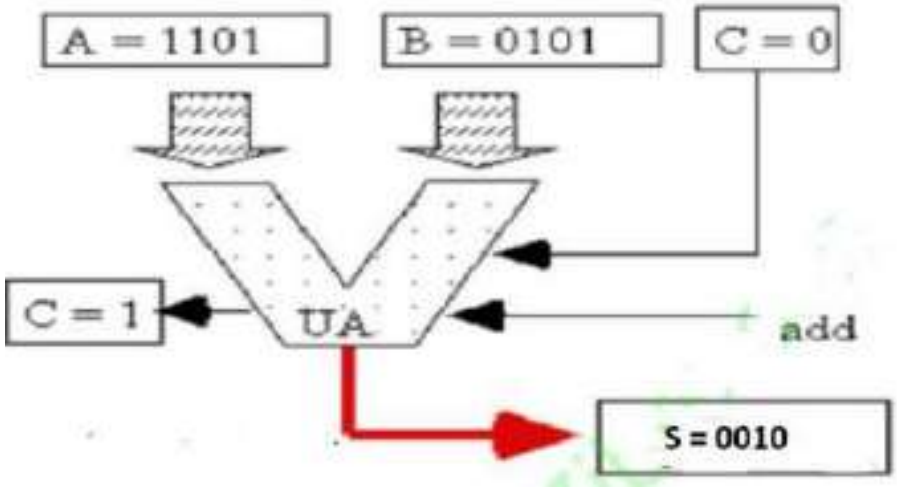
- L'Unité Arithmétique et Logique (UAL).
- L'accumulateur.
- Le registre d'état



Unité de traitement

L'Unité Arithmétique et Logique (UAL).

- Elle assure les fonctions logiques ou arithmétique pour effectuer les calculs et les opérations logiques des différents instructions à exécuter.
- Les données sont prises aux entrées de l'UAL, sont traités, ensuite le résultat est fourni en sortie et généralement stocké dans un registre dit accumulateur.
- Les informations qui concernent l'opération sont envoyées vers le registre d'état.



$$\begin{array}{r} A = 1101 \\ +\ B = 0101 \\ \hline S = 0010 \end{array}$$

Unité de traitement

L'accumulateur

- C'est un registre de travail qui sert à stocker une opérande au début d'une opération arithmétique et le résultat à la fin de l'opération.
- Il permet de stocker les résultats des opérations arithmétiques et logiques en cours d'exécution dans UAL.

Le registre d'´etat

- Il est généralement composé de 8 bits à considérer individuellement.
- Chaque bit est un indicateur dont l'état dépend du résultat de la dernière opération effectuée par l'UAL. On les appelle indicateurs d'état ou flag ou drapeaux.
- Dans un programme le résultat du test de leur état conditionne souvent le déroulement de la suite du programme. Par exemple la retenue, la parité. . . .

Mémoire

- Pour mémoriser une information sur 1 seul bit, on peut utiliser une **bascule**.
- Pour mémoriser une information sur n bits, on peut utiliser un **registre**.

Question : Si on veut mémoriser une information de taille importante, alors comment peut-on le faire ?

Idée : il faut utiliser une mémoire.

Mémoire

Une mémoire est un dispositif capable :

- d'enregistrer une information
- de la conserver (mémoriser)
- de la restituer (possible de la lire ou la récupérer par la suite).

Mémoire

Types de mémoires

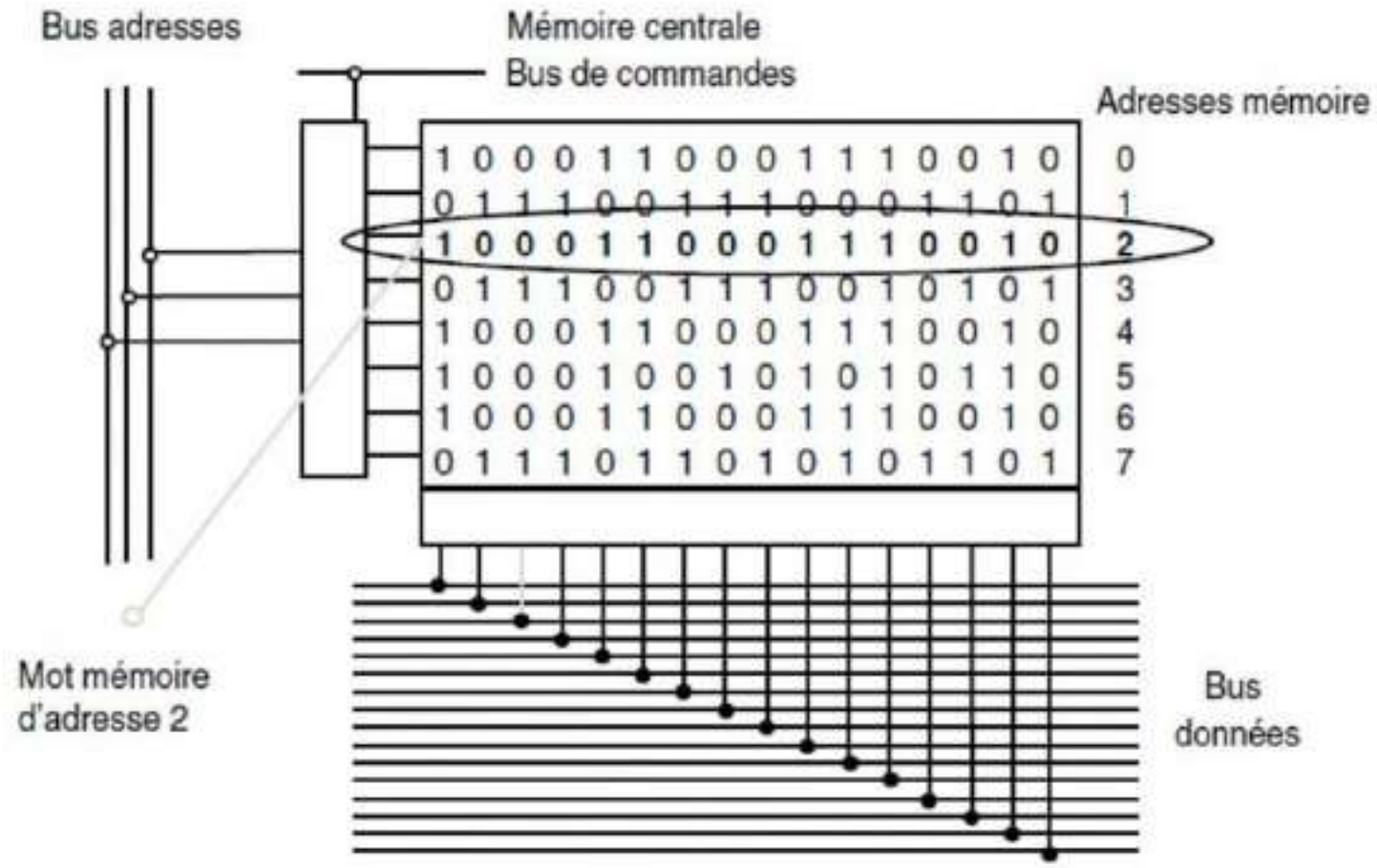
- Les registres : Ce sont les éléments de mémoire les plus rapides. Ils sont situés au niveau du processeur et servent au stockage des opérandes et des résultats intermédiaires.
- La mémoire cache : C'est une mémoire rapide de faible capacité destinée à accélérer l'accès à la mémoire centrale en stockant les données les plus utilisées.
- La mémoire principale : C'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 Elle 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 C'est une mémoire périphérique de grande capacité utilisée pour le stockage permanent ou la sauvegarde des informations. Elle utilise pour cela des supports magnetiqueś (disque dur, ZIP) ou optiques (CDROM, DVDROM).

Types de mémoires

Mémoire

- Pour stocker l'information, la mémoire est découpée en cellules mémoires: les mots mémoires. Donc une mémoire peut être représentée comme une armoire de rangement constituée de différents tiroirs où chaque tiroir représente alors une case mémoire (mots mémoires) qui peut contenir un seul élément .
- Chaque mot est constitué par un certain nombre de bits qui définissent sa taille. On peut ainsi trouver des mots de 1 bit, 4 bits (quartet) ou encore 8 bits (octet ou byte), 16 bits voire 32 ou 64 bits.
- Chaque mot est repéré dans la mémoire par une adresse, un numéro qui identifie le mot mémoire. Ainsi un mot est un contenant accessible par son adresse et la suite de digits binaires composant le mot représente le contenu ou valeur de l'information (Données ou instruction).

Organisation d'une mémoire centrale



Mémoire

- La capacité de stockage de la mémoire est définie comme étant le nombre de mots qui la constituent.
- Avec une adresse de n bits, il est possible de référencer au plus 2^n cases mémoire.
- Chaque case est remplie par un mot de données (sa longueur m est toujours une puissance de 2).

Capacité = 2^n Mots mémoire = $2^n * m$ Bits

Mémoire

- Le nombre de fils d'adresses d'un boitier mémoire définit donc le nombre de cases mémoire que comprend le boitier.
  Nombre de mots = $2^{\text{nombre de lignes d'adresses}}$
- Le nombre de fils de données définit la taille des données que l'on peut sauvegarder dans chaque case mémoire.
  Taille du mot (en bits) = nombre de lignes de données

Exercice 1

Soit une mémoire ayant une capacité de 8 mots de 16 bits chacun.
Exprimer cette capacité en nombre d'octets ou de bits.

Solution

Capacité d'une mémoire = Nombre de mots $*$Taille du mot
La mémoire en question a donc une capacité
Capacité= (8 $*$2 octets) =16 octets = (8 $*$16 bits) 128 bits.

Exercice 2

Dans une mémoire la taille du bus d'adresses K=16 et la taille du bus de données N=8. Calculer la capacité de cette mémoire.

Solution

Capacité d'une mémoire = Nombre de mots $*$Taille du mot

On a Taille de bus d'adresse = Nombre de lignes d'adresses

Donc Nombre de mots = 2 **nombre de lignes d'adresses** Et aussi Taille de bus de données = Taille du mot

Alors

Capacité = 2^{16} mots de 8 bits

= $2^{16} * 2^3$ bits

= 2^{19} bits

= 2^{16} octets

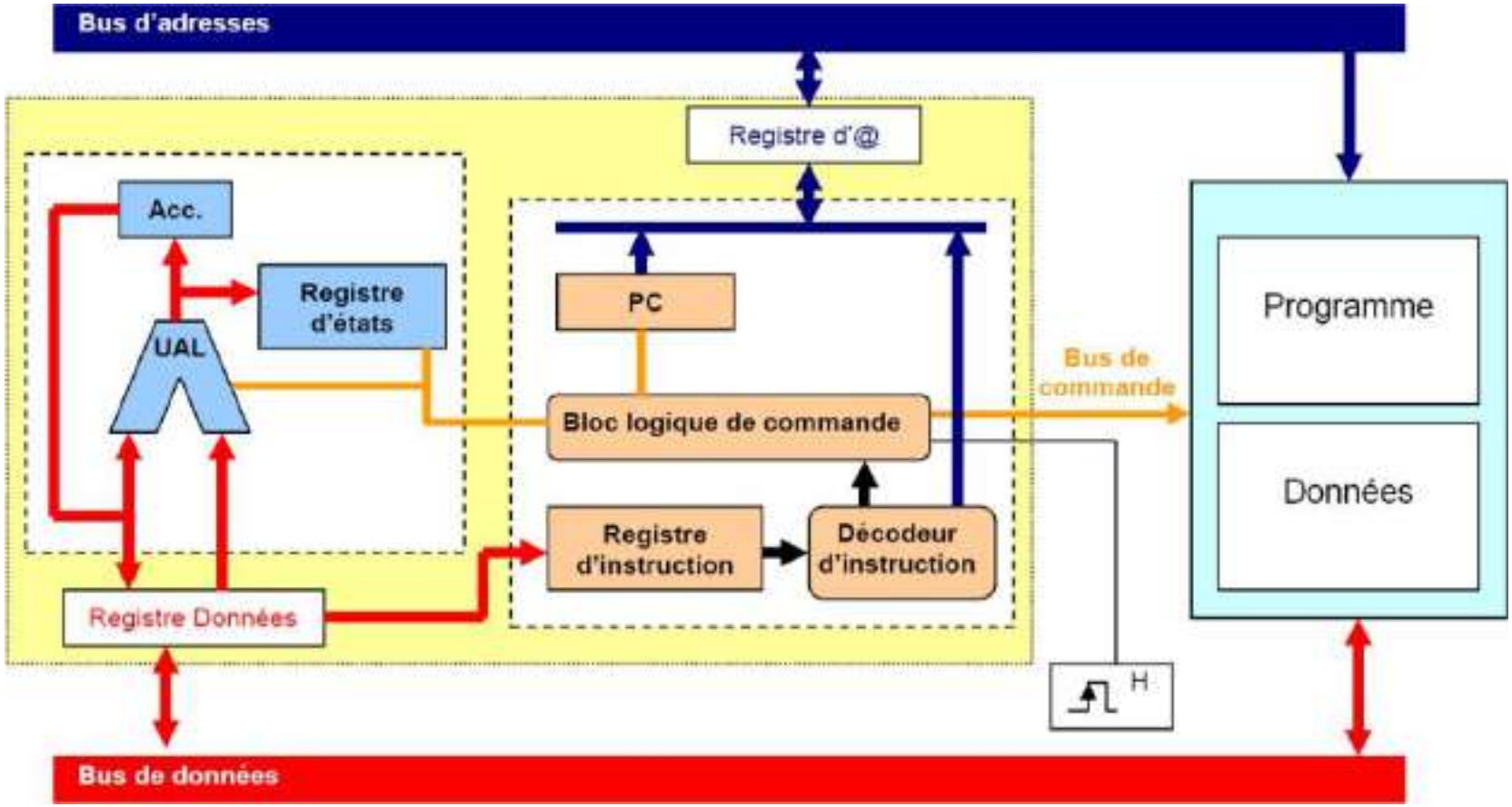
Fonctionnement d'un Microprocesseur

- Le Microprocesseur contient dans son unité de commande (voir la section précédente) deux registres spéciaux :
  - Registre d'Instruction (RI)
  - Compteur Ordinal (CO)
- Utilisation des registres RI et CO par le Microprocesseur :
  - Extraction de l'instruction à l'adresse contenue dans CO
  - Stockage de l'instruction dans RI
  - Décodage de l'instruction et accès à ses opérandes
  - Mise à jour CO
  - Exécution de l'instruction
  - Stockage des résultat

Fonctionnement d'un Microprocesseur

- Extrait une instruction de la mémoire
- Analyse l'instruction
- Recherche dans la mémoire les données concernées
- Assure le chargement des données si nécessaire
- Déclenche l'opération adéquate sur l'UAL
- Range le résultat dans la mémoire
- Passe à l'instruction suivante

Schéma fonctionnel



Fonctionnement d'un Microprocesseur

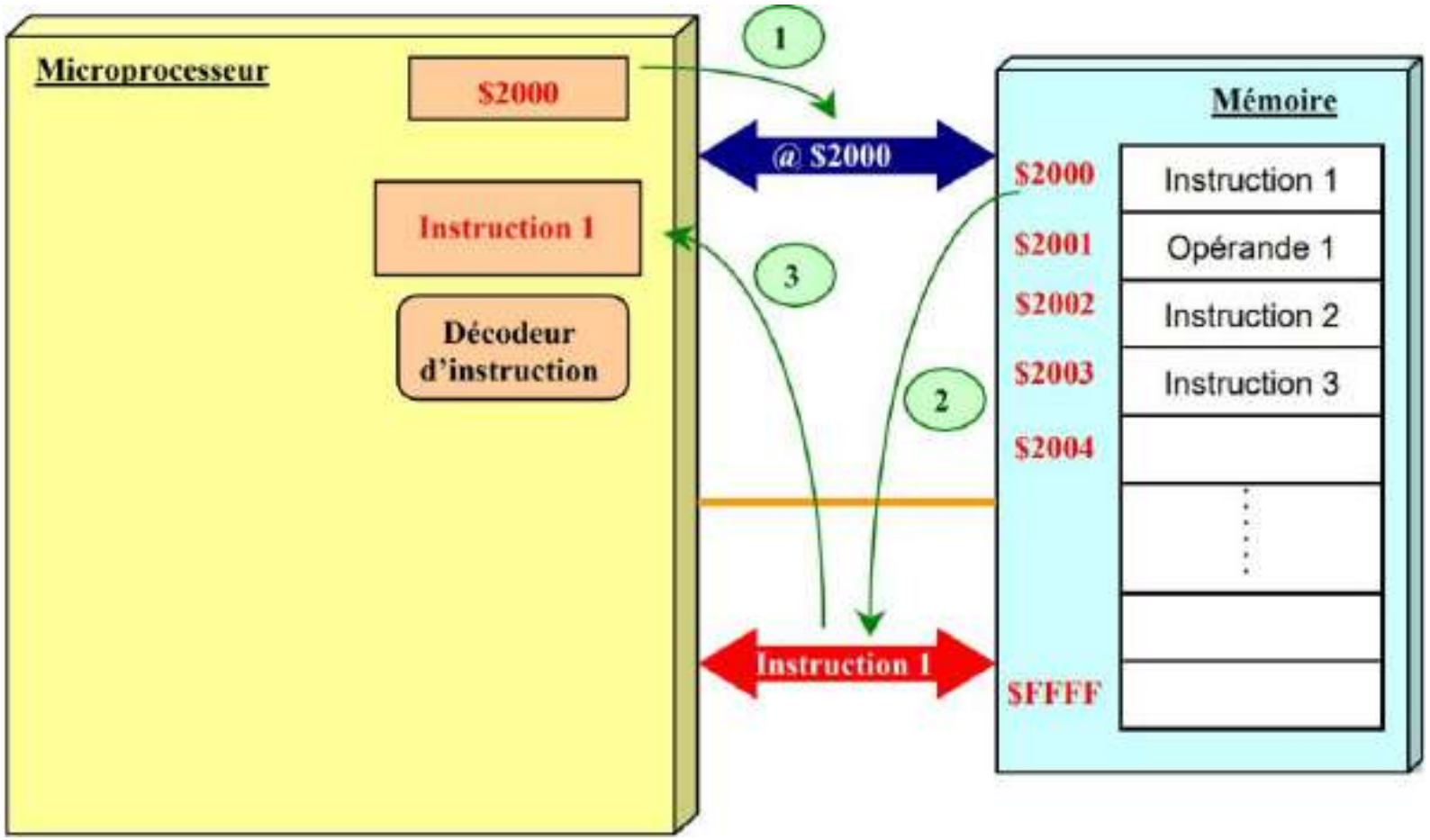
Cycle d'une instruction

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 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.

Schéma explicatif de la phase 1



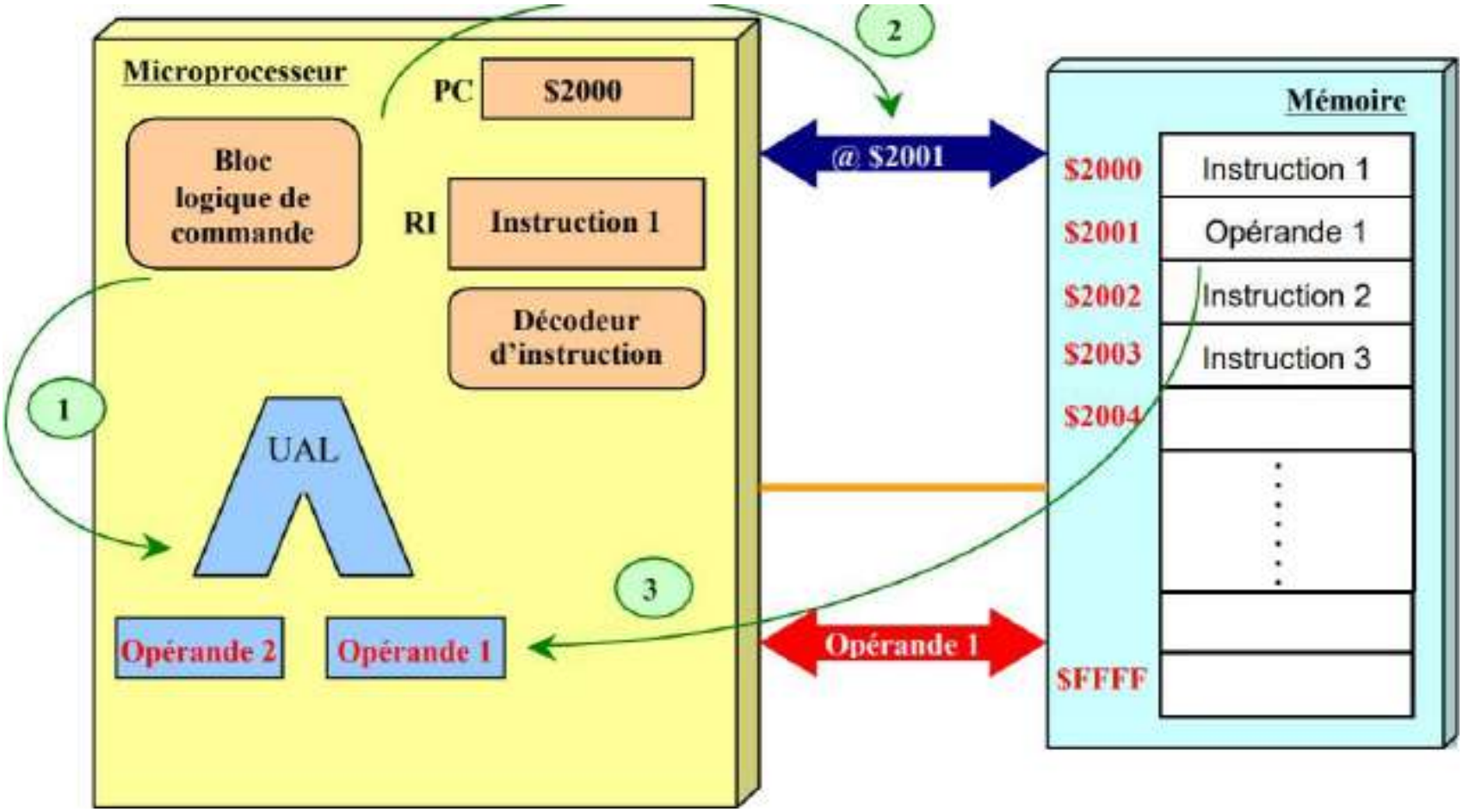
Cycle d'une instruction

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.

Schéma explicatif de la phase 2

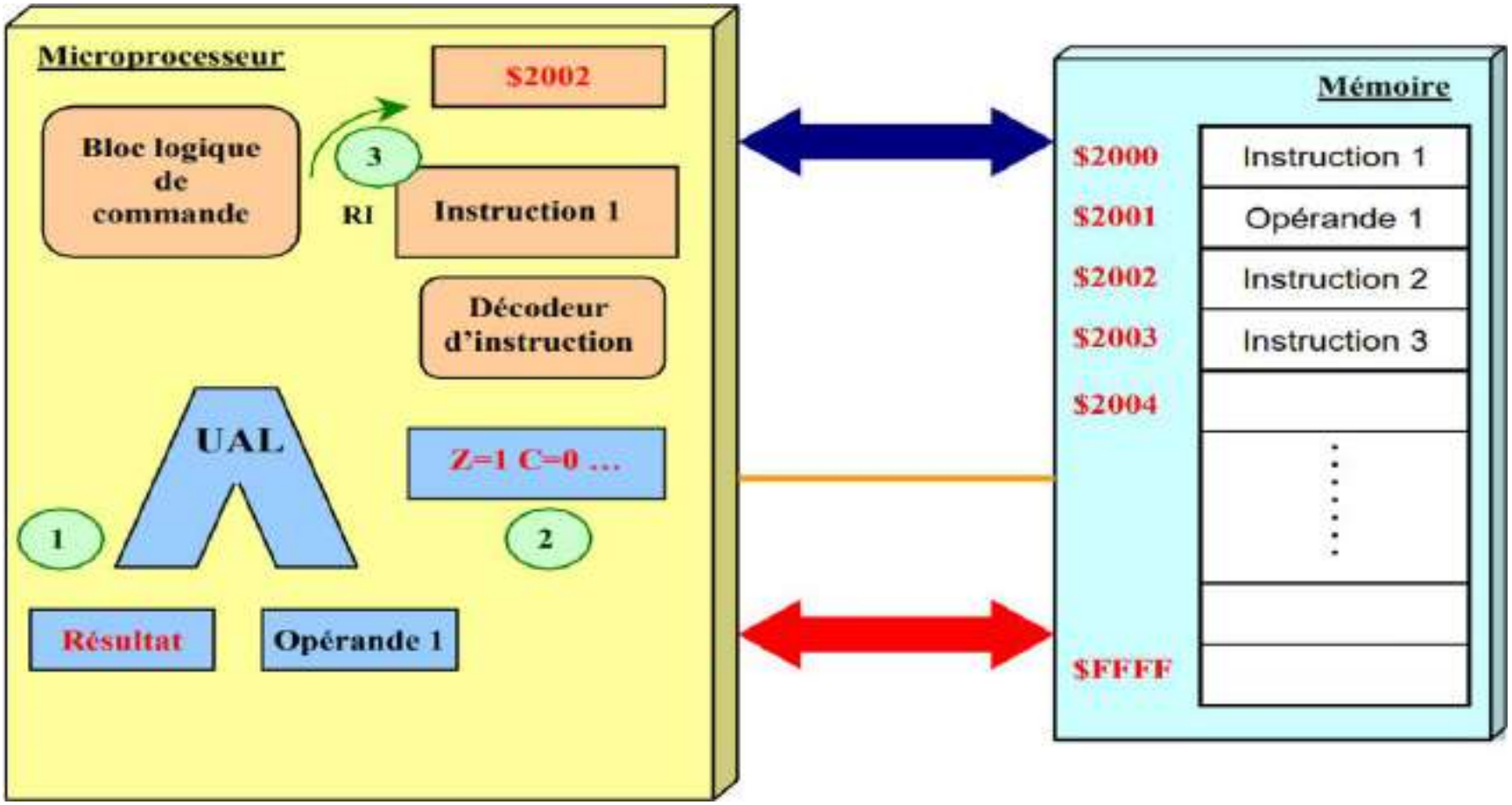


Cycle d'une instruction

Phase 3 : Exécution de l'instruction

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

Schéma explicatif de la phase 3



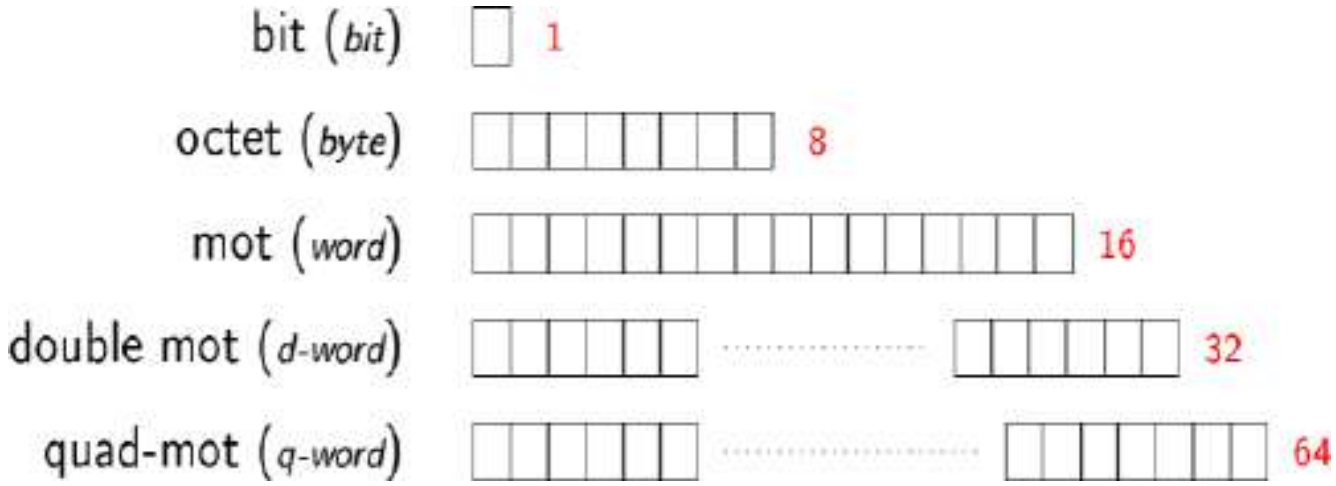
Exercice

Expliquer (via des schémas) les différentes phases permettant d'exécuter le programme suivant par le Microprocesseur :

```
int A=2 ;
int B=4 ;
int C= 17;
int D,E ;
  D=A*B;
  E=C+D ;
```

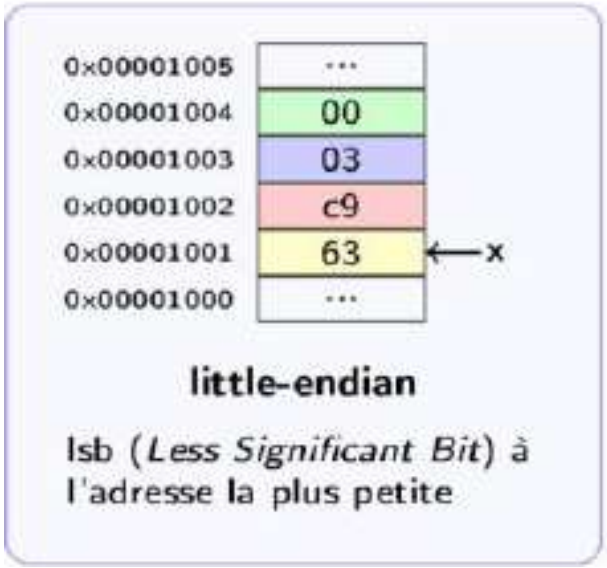
Microprocesseur (CPU) 8086

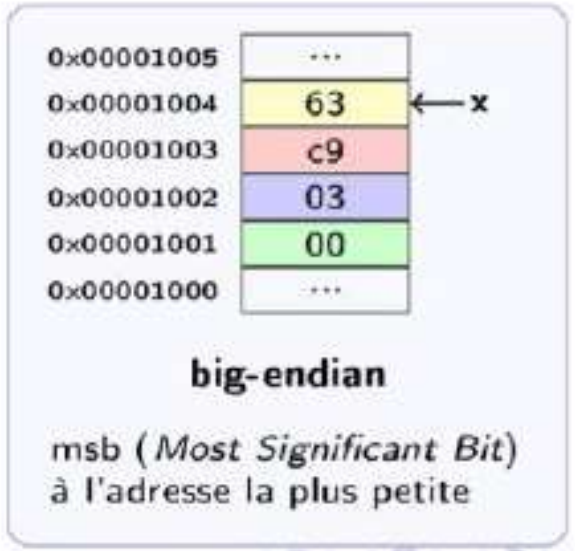
Types de données fondamentaux



Représentation et stockage en mémoire

$$248163_{10} = 111100100101100011_2$$

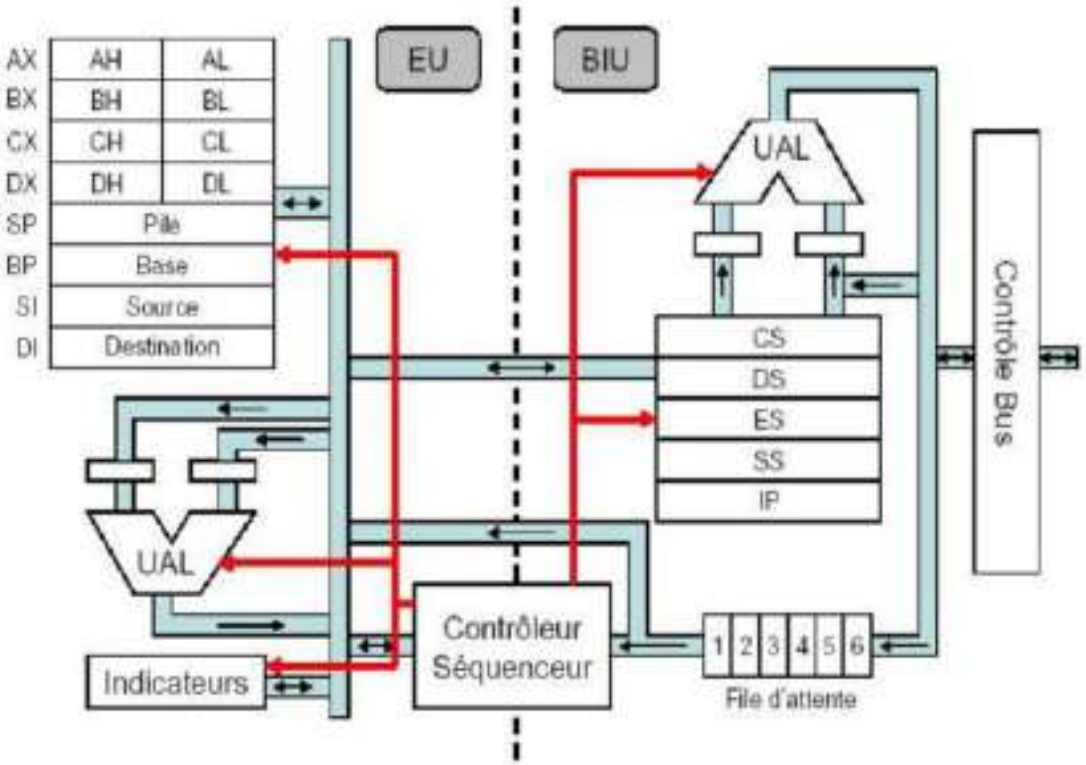




Microprocesseur 8086

- L'architecture de 8086 est little-endian.
- Le microprocesseur 8086 se présente sous la forme d'un boˆıtier contenant des circuits intégrés qui dispose de sorties pour se connecter à un environnement externe.
- Comme tout CPU, le 8086 dispose d'un certain nombre de registres.

Représentations interne d'un 8086



Microprocesseur 8086

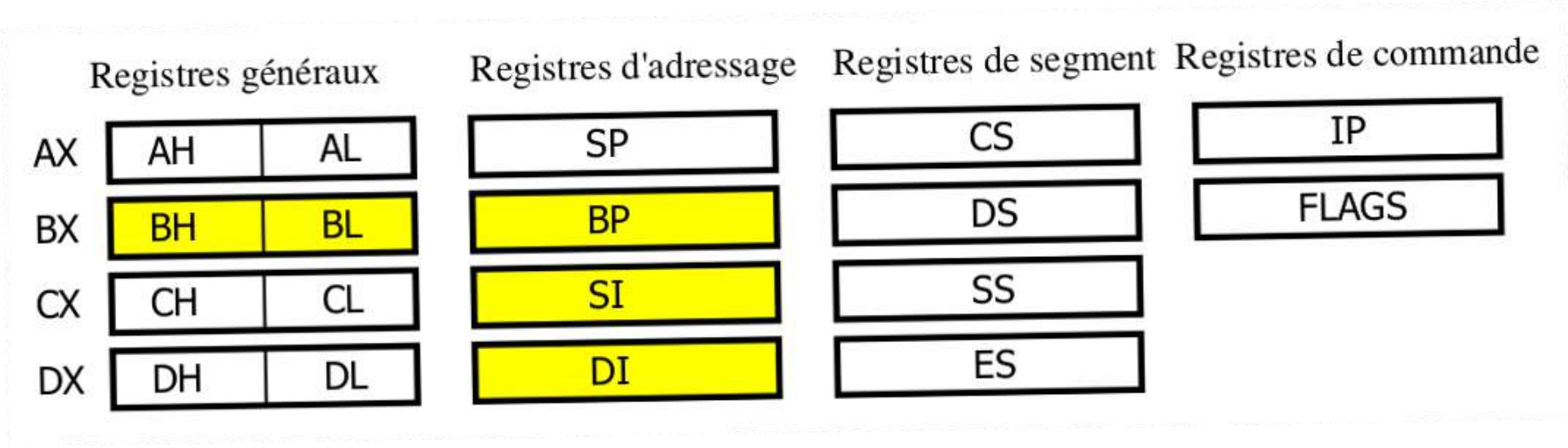
- L'unité d'interface de bus (BIU) recherche les instructions en mémoire et les range dans une file d'attente.
- L'unité d'exécution (EU) exécute les instructions contenues dans la file d'attente.
- Les deux unités fonctionnent simultanément.

Registres du CPU 8086

- Un registre est un composant de mémoire interne au microprocesseur dont sa capacité est calibrée sur une taille des instructions.
- Le rôle du registre est de stocker des adresses ou des résultats intermédiaires des calculs en cours.
- Les commandes de l'assembleur manipulent les registres. Pour les contrôler et les utiliser chaque registre dispose d'un nom.

Registres du CPU 8086

Le 8086 comprend 3 groupes de 4 registres de 16 bits, un registre d'état de 9 bits et un compteur programme de 16 bits non accessible par l'utilisateur.



Les registres généraux

Groupe de données:

- AX : Registre accumulateur
- BX : Registre d'adresses de base
- CX : Registre compteur
- DX : Registre de données

Ces registres sont destinés au traitement des valeurs. Ils permettent d'effectuer des additions, des multiplications, des calculs logiques,

	15 8	7 0
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

Utilités des registres généraux

Si nous manipulons des nombres codés sur 16 bits (1 mot mémoire), nous utilisons 4 registres généraux :

- AX sert à effectuer des calculs arithmétiques.
- BX sert à effectuer des calculs arithmétiques ou bien des calculs sur les adresses.
- CX utilisé généralement comme compteur dans des boucles.
- DX sert à stocker des données destinées à des fonctions.

Il s'agit là de l'utilisation de base de ces registres, mais dans la pratique, ils peuvent être utilisés à d'autres fins (on les verra dans la partie Jeu d'instuction)

Utilités des registres généraux

Si nous manipulons des nombres codés sur 8 bits (1 octet), nous utilisons les 4 registres généraux en mode 8 bits pour en produire 8 registres AH, AL, BH, BL, CH, CL, DH et DL avec comme convention, H est mis pour ”High” et L pour ”Low” :

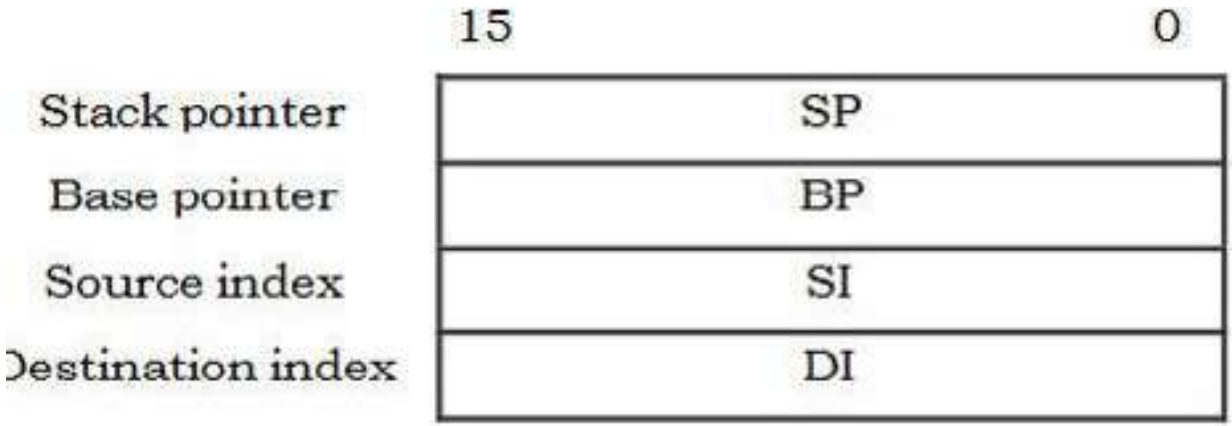
Exemples:
AH contient l’octet de poids fort du registre AX.
BL contient l’octet de poids faible du registre BX.

Les registres d'adressage (d'offset)

Ces registres de 16 bits permettent l'adressage d'un opérande à l'intérieur d'un segment de 64 ko (2^{16} positions mémoires).

Ils forment 2 groupes:

- Registres d'index : SI et DI
- Registres pointeurs : SP et BP



Registres d'index

Dans les instructions de mouvements de chaînes d'octets, on utilise les registres SI et DI:

- SI : indexe les caractères de la chaîne émettrice.
- DI : indexe les caractères de la chaîne réceptrice.

Registres pointeurs

- Une pile est une zone mémoire qui permet de conserver de manière temporaire des données (par exemple, l'état des registres lors d'un appel de procédure). Elle est organisée comme une pile d'assiettes. On pose et on retire les assiettes toujours sur le haut de la pile. On dit que c'est une pile LIFO (Last IN, First Out).
- Empiler une donnée: sauvegarder une donnée sur (le sommet) de la pile.
- Dépiler une donnée: retirer une donnée (du sommet) de la pile

Registres pointeurs

Nous utilisons 2 registres pointeurs :

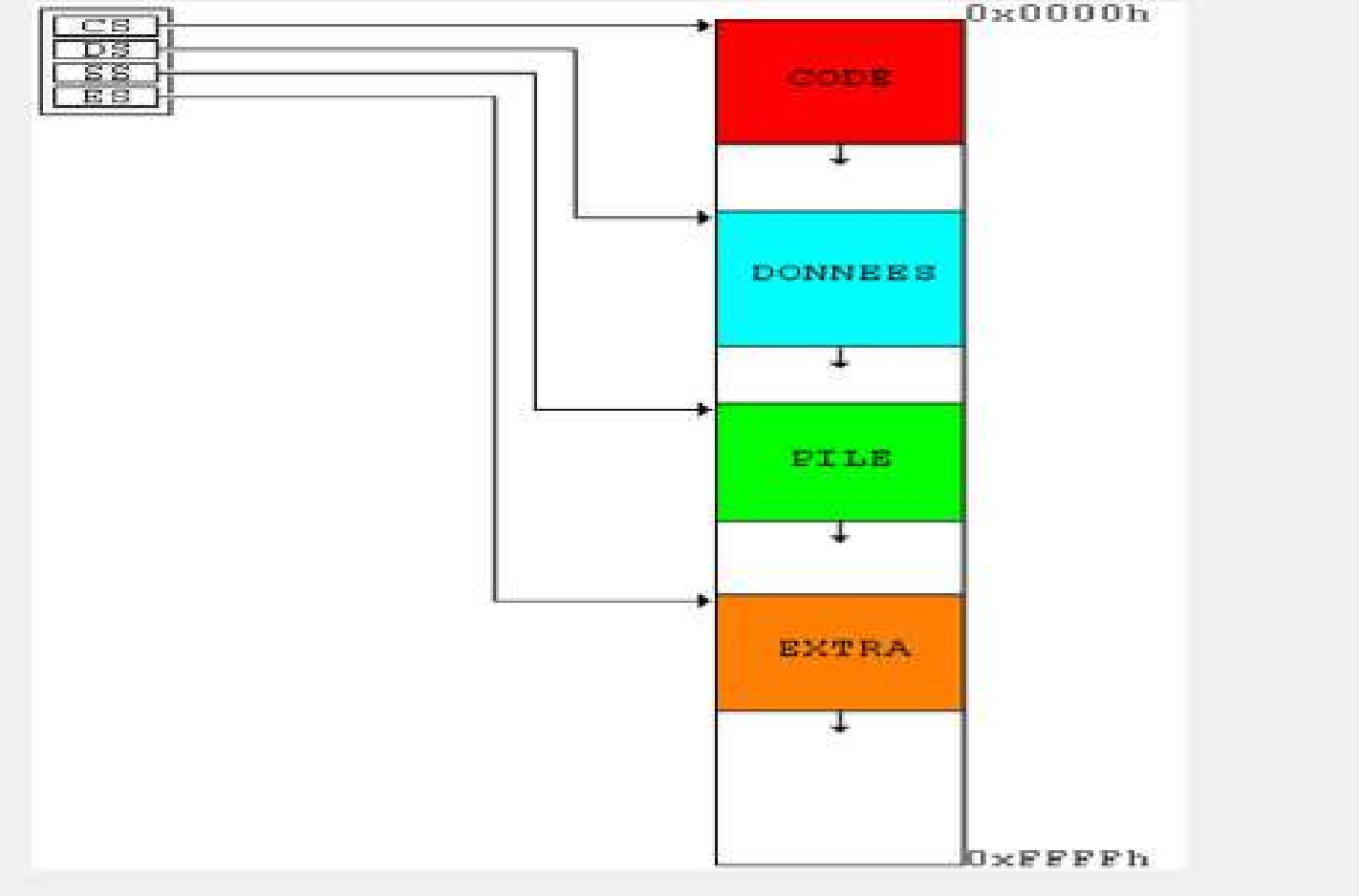
- SP (Stack Pointer) pointe sur le sommet de la pile et se met à jour automatiquement par les instructions d'empilement et de dépilement.
- BP (Base Pointer) pointe la base de la région de la pile contenant les données accessibles (variables locales, paramètres,...) à l'intérieur d'une procédure.

Registres de segment

Pour accéder à sa mémoire centrale, le 80x86 dispose de registres de segment suivants :

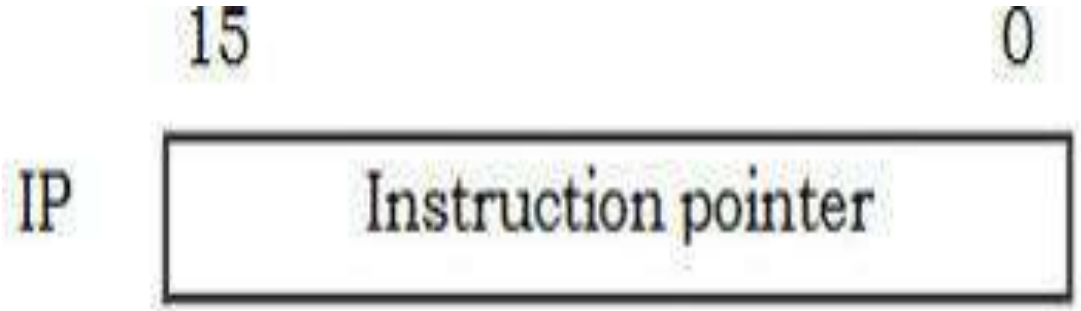
- CS (Code Segment) pointe sur la base du segment qui contient le code (les instructions que doit exécuter le processeur).
- DS (Data Segment) pointe sur la base du segment contenant les données (variables, tableaux ...).
- SS (Stack Segment) pointe sur la base du segment qui contient la pile gérée par les registres SP et BP.
- ES (Extra Segment) pointe sur la base d'un segment supplémentaire qui est généralement utilisé pour compléter le segment de données.

Registres de segment



Le registre IP

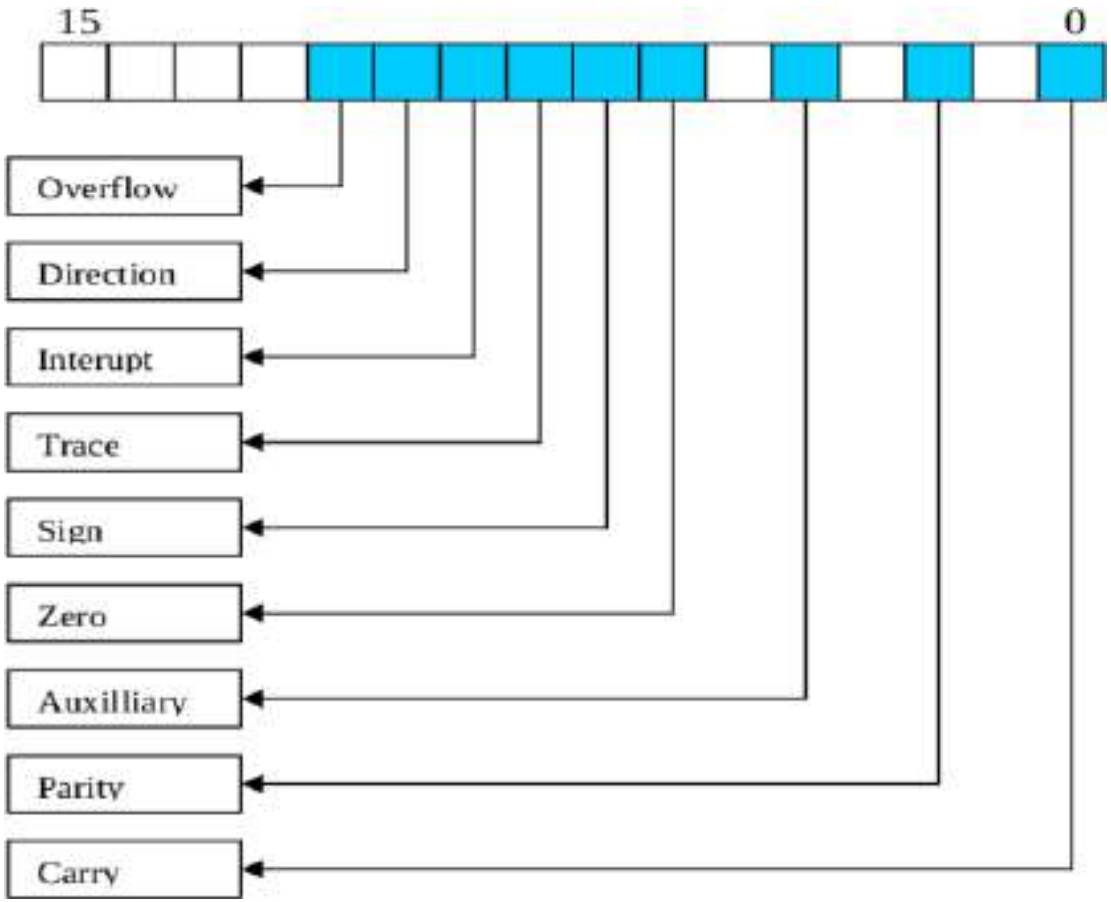
- Le registre IP: pointeur d'instruction (compteur de programme). Il contient l'adresse de l'emplacement mémoire où se situe la prochaine instruction à exécuter. Autrement dit, il doit indiquer au processeur la prochaine instruction à exécuter. Le registre IP est constamment modifié après l'exécution de chaque instruction afin qu'il pointe sur l'instruction suivante.



Le registre Flag

- Le registre d'état (Flag) : indique l'état de CPU.
- C'est un registre de 16 bits, chaque bit s'appelle un flag (ou un indicateur) et peut prendre la valeur 1 ou 0.
- Il sert à contenir l'état de certaines opérations effectuées par le processeur. Par exemple, quand le résultat d'une opération est trop grand pour être contenu dans le registre cible, un bit spécifique du registre d'état (le bit OF) est mis à 1 pour indiquer le débordement.

Le registre Flag



- CARRY Flag : ce flag est positionné à ”1” lorsque on a un débordement (retenue).

	1	0	0	1
+				
	1	0	0	0
=	0	0	0	0

CF
1

- PARITY Flag : si le résultat de l'opération contient un nombre pair de 1 alors PF=1 sinon PF=0.
- AUXILLIARY Flag : ce flag est à "1" lorsqu'un débordement a lieu de demi-mot de 8 bits ou 16 bits.
- ZERO Flag : ce flag est à "1" lorsque tous les bits d'un mot sont à "0", c à d si le résultat d'une opération est égale à 0 (soustraction, comparaison...) alors ZF=1 sinon ZF=0.

- SIGN Flag : SF est positionné à 1 si le bit de poids fort du résultat d'une addition ou soustraction est 1 ; sinon SF=0. SF est utile lorsque l'on manipule des entiers signés, car le bit de poids fort donne alors le signe du résultat. Exemples (sur 8 bits):

```
      10010110
    + 01010100
SF=1  11101010
```

```
      11011001
    + 01010010
SF=0  00101011
```

- OVERFLOW Flag: ce flag est à ”1” lorsqu’on a un débordement signé.

	0	1	1	1	(+7)
+					
	0	1	0	1	(+5)
=	1	1	0	0	(Résultat non signé = 12) (Résultat signé = - 4).

OF
1

- Direction Flag: pour auto incrémenter ou auto décrémenter le SI et le DI. Ce flag est utilisé par quelques instructions pour traiter les chaˆınes de données, lorsque ce drapeau est placé à 0, la chaˆıne est traitée octet par octet en incrémentant, lorsque ce drapeau est placé à 1, la chaˆıne est traitée octet par octet en décrémentant. Autrement dit DF indique le sens de progression des registres d'index SI et DI lors des instructions de traitements de chaˆınes. DF=1 adresses décroissantes, DF=0 adresses croissantes
- TRACE Flag: ce bit est à "1" signifie que l'exécution est en mode pas à pas (STEP BY STEP).

- Interrupt Flag: pour masquer les interruptions venant de l'extérieur, ce bit est mis à 0, dans le cas contraire le microprocesseur reconnaˆit l'interruption de l'extérieur.
- Interruption: un événement extérieur qui demande l'attention de l'ordinateur, par exemple la pression d'une touche du clavier. La machine doit pouvoir réagir immédiatement, sans attendre que le programme en cours d'exécution se termine. Pour cela, elle interrompt ce dernier pendant un bref instant, le temps de traiter l'événement survenu puis rend le contrôle au programme interrompu.

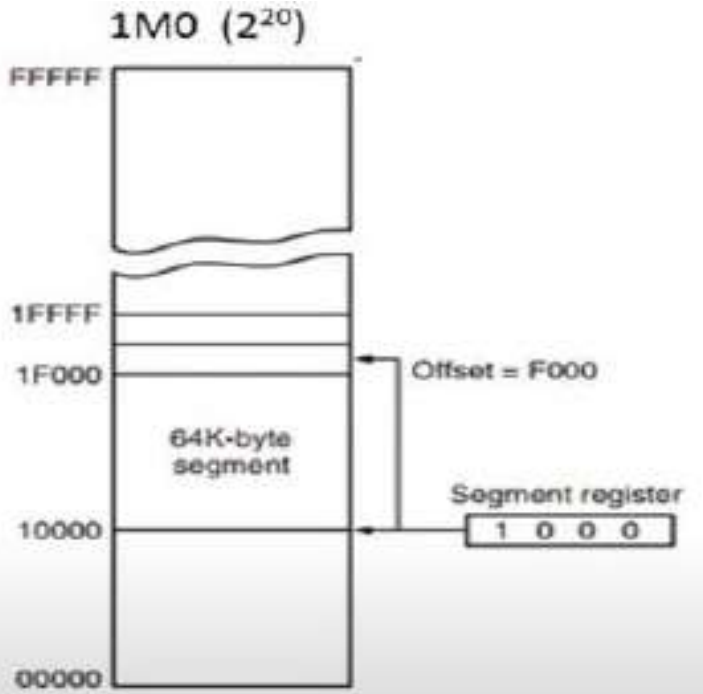
Segmentation de la mémoire

Votre PC est conçu pour gérer 1 Mo (soit 2^{20} octets) de mémoire vive en mode réel. Il faut donc 20 bits au minimum pour adresser toute la mémoire. Or en mode réel les bus d'adresses n'ont que 16 bits. Ils permettent donc d'adresser 2^{16} = 65536 octets = 64 Ko, ce qui est insuffisant!

- L’espace mémoire est divisé en segments. Un segment est une zone mémoire de 64 ko définie par son adresse de départ (doit être un multiple de 16).
- Dans une telle adresse, les 4 bits de poids faible sont à 0.
- Ainsi, on peut représenter l’adresse d’un segment en utilisant seulement ses 16 bits de poids fort (pour désigner une case mémoire dans un segment, il suffit une valeur sur 16 bits).

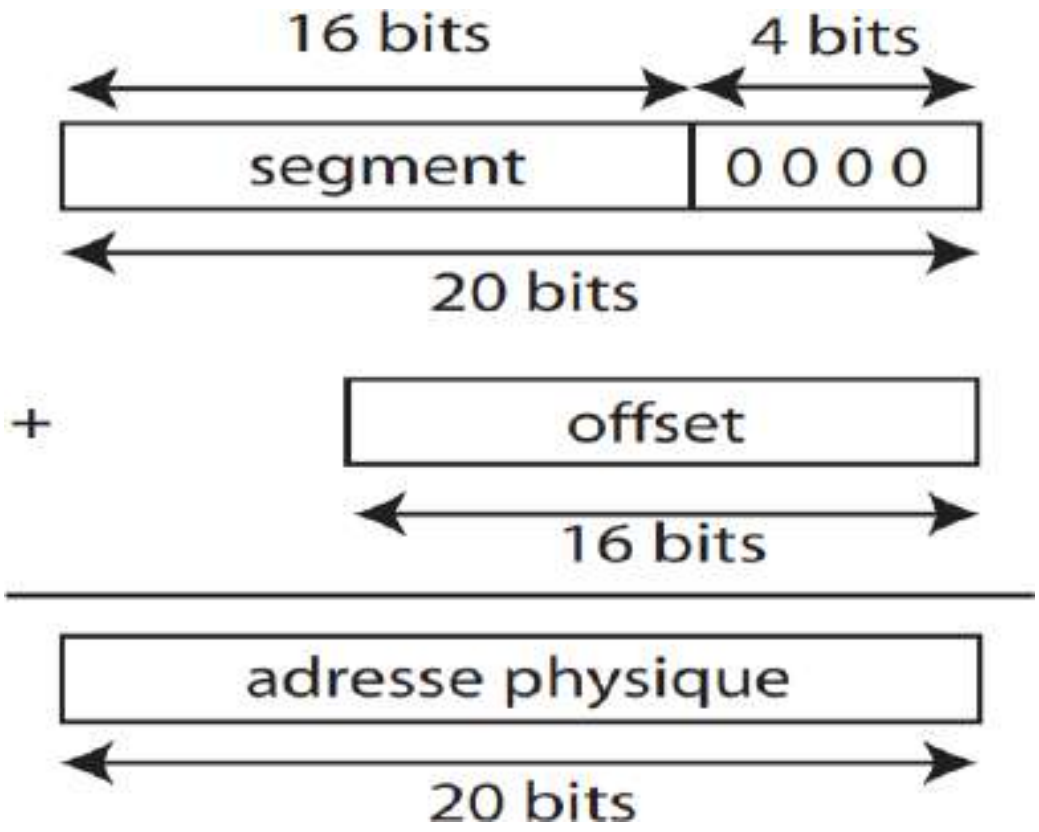
- Pour adresser une case mémoire, on utilise deux nombres. Le premier est appelé adresse de segment, le second adresse d'offset.

Segment	Adresse début	Adresse fin	Pointeur de segment
Segment 0	00000	0FFFF	00000
Segment 1	10000	1FFFF	10000
Segment 2	20000	2FFFF	20000
Segment 14	E0000	EFFFF	E0000
Segment 15	F0000	FFFFF	F0000



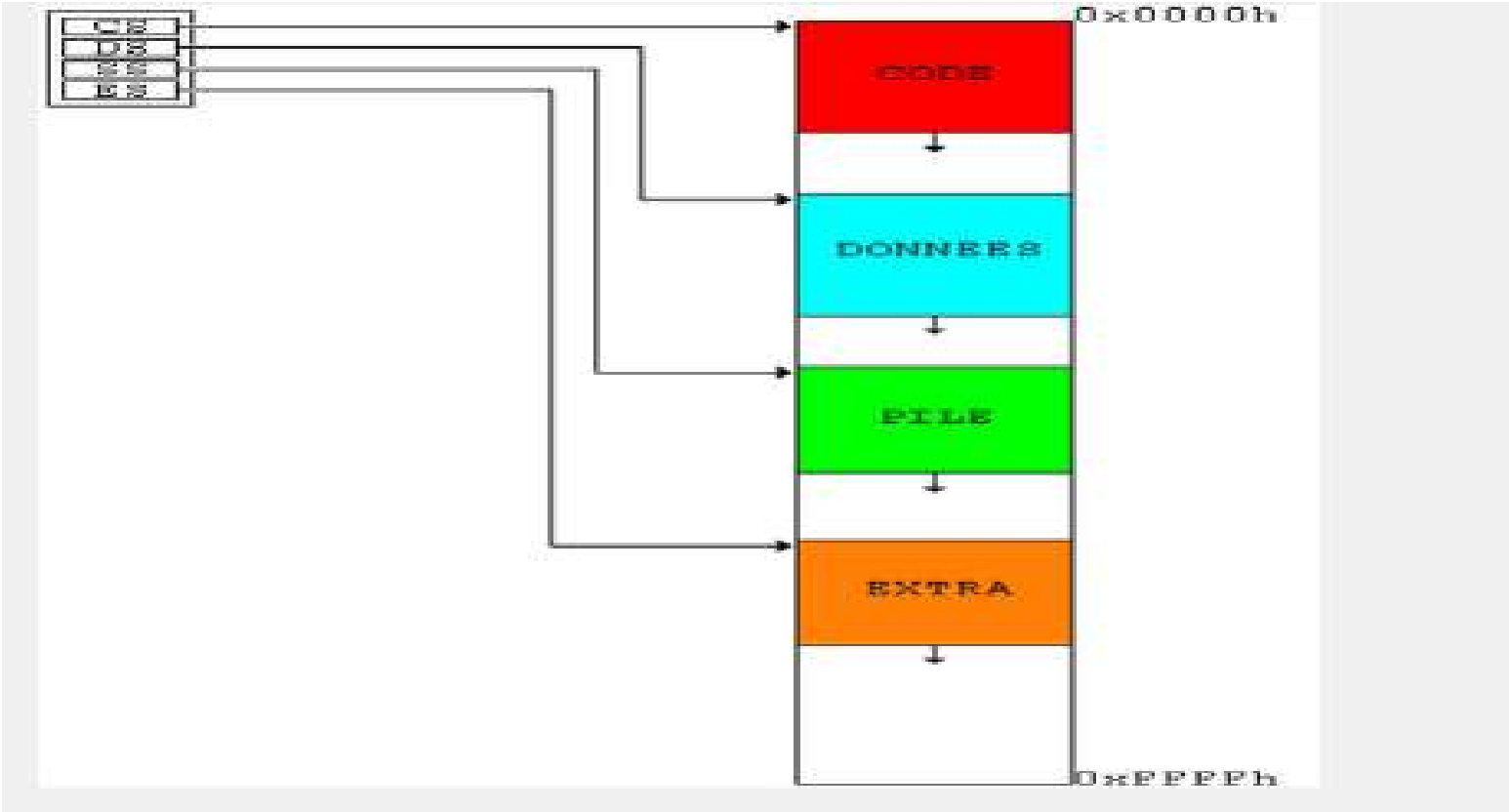
- La donnée d'un couple (segment,offset) définit une adresse logique notée sous la forme segment : offset
- L'adresse d'une case mémoire donnée sous la forme d'une quantité sur 20 bits (5 digits hexa) est appelée adresse physique (elle correspond à la valeur envoyée réellement sur le bus d'adresse).

Correspondance entre adresse logique et adresse physique



Ainsi : Adresse Physique= 16 x segment + offset

A un instant donné, le CPU 8086 a accès à 4 segments dont les adresses sont stockées dans les registres de segment:



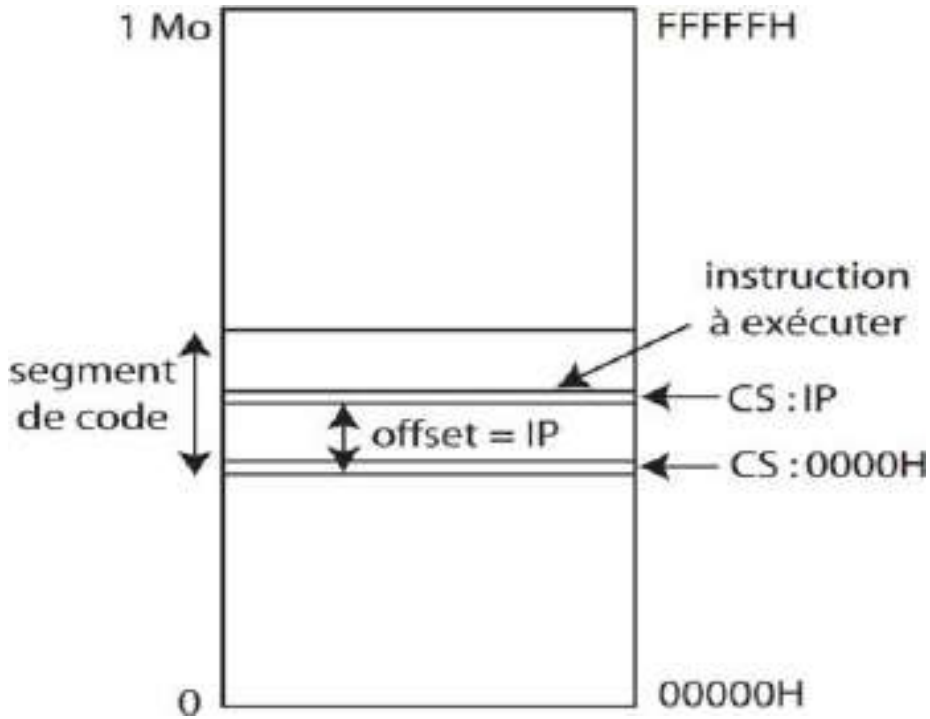
Les registres de segments DS et ES peuvent être associés à un registre d'index.

example
DS : SI
ES : DI.

Le registre de segment de pile peut être associé aux registres de pointeurs
example

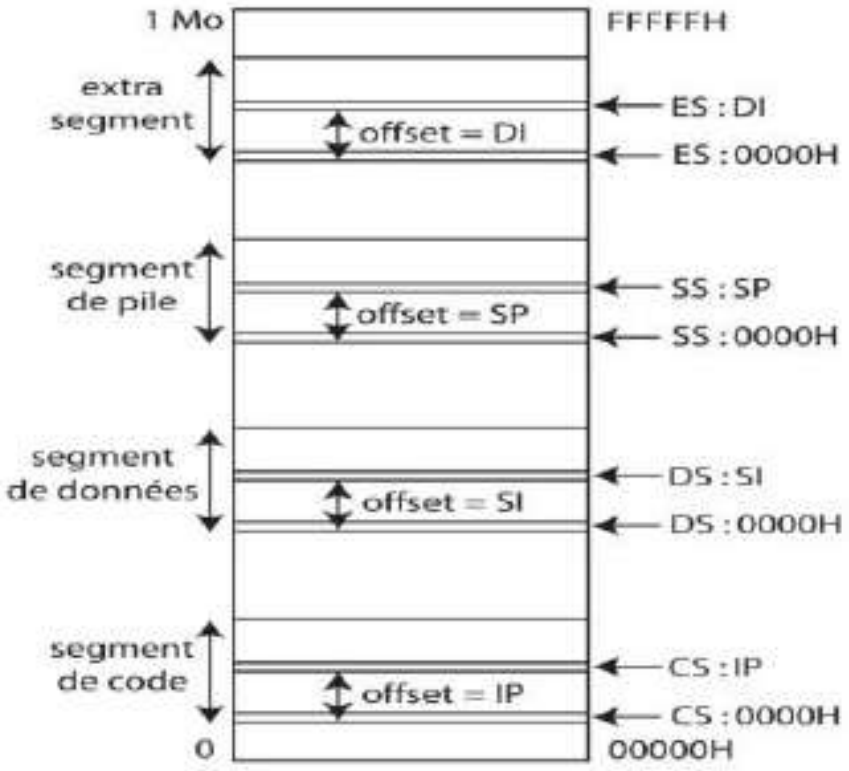
SS : SP ou SS : BP.

Le registre CS est associé au registre IP. Ainsi, la prochaine instruction à exécuter a l'adresse logique CS : IP



Exercice

On vous donne la mémoire accessible par le 8086 à un instant donné:



On suppose que les contenus des registres après un RESET sont:

IP=0000H

CS=FFFFH

DS=0000H

ES=0000H

SS=0000H

Donner l'adresse logique de l'instruction à exécuter.

Déduire son adresse physique.

Solution

Puisque CS contient la valeur FFFFH et IP la valeur 0000H, la première instruction exécutée par le 8086 se trouve donc à l'adresse logique FFFFH:0000H, correspondant à l'adresse physique FFFF0H.

Assembleur

Programmation en Assembleur

- Introduction
- Types de données
- Structure d'un programme (.asm)

Introduction:
Langage machine

- Un langage machine est un langage compris directement par le processeur en vue d'une exécution. C'est un langage binaire : ses seules lettres sont les bits 0 et 1.
- Chaque modèle de processeur possède son propre langage machine. Soient deux modèles de processeurs P1 et P2, on dit que P1 est compatible avec P2 si toute instruction formulée pour P2 peut être comprise et exécutée par P1. L'inverse n'est pas forcément vrai, Pl peut avoir des instructions supplémentaires que P2 ne connaˆıt pas.
- Dans la plupart des langages machine, une instruction commence par un opcode, une suite de bits qui porte la nature de l'instruction. Celui-ci est suivi des suites de bits codant les opérandes de l'instruction.

Langage machine

Exemple

La suite 10101011 00000000 00000010 (AB 00 02 en hexa) est une instruction dont : le opcode est 10101011 et l'opérande est 00000000 00000010.

Ce langage est très difficile à maîtriser puisque chaque instruction est codée par une séquence propre de bits.

Afin de faciliter la tâche du programmeur, on a créé un langage Assembleur.

Langage Assembleur

- Historiquement, le langage assembleur a été le premier langage de programmation non binaire accessible au programmeur. C'est un langage qui permet de décrire les instructions élémentaires qui manipulent dans le processeur les registres, l'accès à la mémoire.
- Le langage Assembleur est la représentation symbolique du langage machine. Autrement dit, le code en langage assembleur est la version lisible du code machine.
- Le langage assembleur est donc un langage de très bas niveau. Cela signifie qu'il est plus proche du langage machine. Cette caractéristique offre des avantages et des inconvénients divers.
- Le langage assembleur est propre à chaque type du processeur. On dit que le langage assembleur est un langage "orienté machine" car il nécessite de penser d'abord à la machine avant de penser au problème à résoudre.

Langage Assembleur

Le langage assembleur permet au programmeur :

- d'utiliser des codes symboliques (MOV, ADD, DIV, etc.) au lieu des codes numériques des opérations.

  Exemple

  Additionner 2 et 3 produit le code suivant en langage Assembleur :

  ```
  MOV AX,2
  ADD AX,3
  ```

  Traduit en langage binaire, il donnera :

  ```
  10101011 00000000 00000010 (AB 00 02 en hexa)
  11011011 00000000 00000011 (DB 00 03 en hexa)
  ```

- d'utiliser les adresses symboliques (étiquettes) au lieu des adresses numériques des emplacements mémoire.

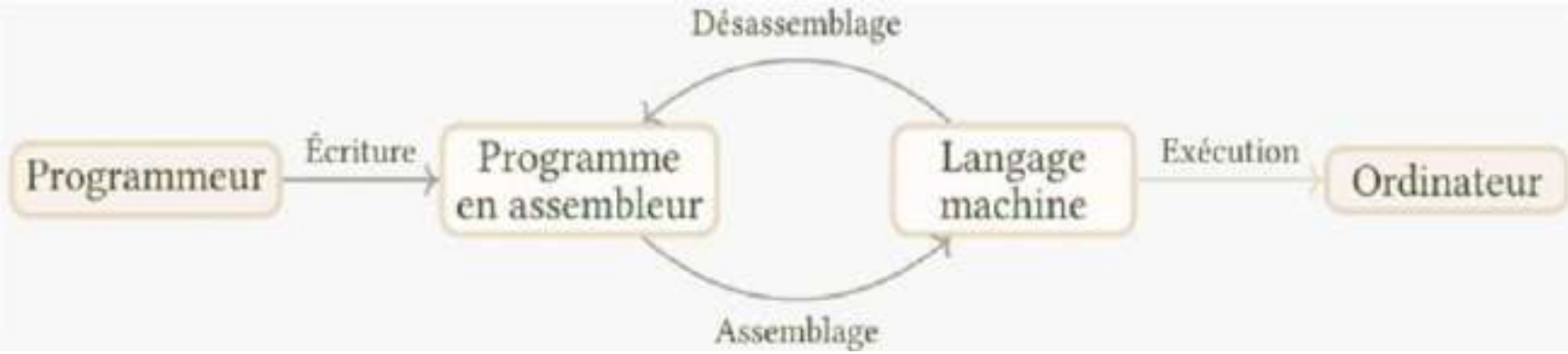
Langage Assembleur

- d'exploiter au maximum les ressources matérielles. Il ne cache rien au programmeur,contrairement aux langages évolués (C++, Java, etc).
- d'avoir un contrôle très fin sur la mémoire ; souvent, de produire du code dont l'exécution est très rapide.
- de programmer rapidement et facilement (le temps de programmation plus long).

En bref, le maˆitrise de la programmation en langage assembleur permet de mieux comprendre le fonctionnement des autres langages et ainsi d'écrire des programmes plus performants.

Langages d'assemblage vs assembleur

- L'assemblage est l'action d'un programme nommé assembleur qui consiste à traduire un programme en assembleur vers du langage machine.
- Le désassemblage, réalisé par un dés assembleur, est l'opération inverse. Elle permet de retrouver, à partir d'un programme en langage machine, un programme en assembleur qui lui correspond.
- Schéma opérationnel liant tout ceci :



De l'écriture du programme à son exécution

L'exécution de programme source se déroule en quatre étapes :

1. L'édition : La saisie du programme source au clavier nécessite un éditeur de texte. Ce programme doit sauvegardé avec un nom suivi de l'extension .asm.
2. La compilation (l'assemblage) : Consiste à traduire chaque instruction du programme source en une instruction machine. Le résultat de l'assemblage est enregistré dans un fichier avec l'extension .obj.
3. L'édition de liens : Combine plusieurs fichiers objets afin de former un seul fichier complet et exécutable, avec l'extension .exe.
4. L'exécution du programme : Un utilitaire spécial du système d'exploitation, le chargeur est responsable de la lecture du fichier exécutable, de son implantation en mémoire principale, puis du lancement du programme.

Le compilateur

Il existe plusieurs compilateurs pour programmer en langage assembleur. Les plus connus sont :

Masm (Microsoft Assembler): Pour MSDOS, Windows.

Nasm (Netwide Assembler): Pour MSDOS, Windows et Linux.

Fasm (Flat Assembler): Pour MSDOS, Windows et Linux.

Tasm (Turbo Assembler): Pour MSDOS et Windows. NBAsm (New Basic Assembler) : Pour MSDOS seulement.

GoAsm: Pour Windows et sa programmation.

Le débogueur

Le débogueur (debugger) est un programme qui facilite la mise au point de programmes et la correction des erreurs. Il permet d'examiner le contenu de la mémoire ainsi que celui des différents registres du processeur. Il permet également de créer des points d'arrêt (break points) lors de l'exécution du programme à mettre au point, et à partir de ces points d'arrêt, d'exécuter le programme pas à pas, i.e. instruction par instruction tout en observant le contenu des registres, des variables et de la mémoire.

Types de données

- Les variables simples
- Les constantes
- Les tableaux
- Les chaînes de caractères
- Les directives
- Les expressions

Les variables en Assembleur

- Les variables sont considérées comme des emplacements mémoire qui peuvent être manipulés par l'intermédiaire de leurs adresses ou de leurs noms.
- Le nom d'une variable peut être une combinaison de chiffres, de lettres et du caractère "_", mais il doit respecter les contraintes suivantes :
  1. Ne pas commencer par un chiffre.
  2. Ne pas correspondre à un mot clé du langage Assembleur.
- Lorsque l'association entre le nom et la variable est créée, ce nom peut être utilisé pour désigner l'emplacement mémoire contenant l'information.
- L'assembleur traduit le nom en une adresse lors du processus d'assemblage.

Les variables numériques simples

L'association entre le nom et l'adresse s'effectue en utilisant l'un des descripteurs de taille suivants (selon la taille du contenu de la variable) :

DB (Define Byte : 1 octet pour 8086)

DW (Define Word : 2 octets pour 8086)

DD (Define Double : 4 octets)

La valeur associée à la variable peut être un nombre (hexadécimal, décimal ou binaire) ou le symbole ? (si la variable n' est pas initialisée).

Exemples

```
var1 DB 56d
var2 DB ?
var3 DW 1210h
```

Les constantes

Une constante numérique est définie par la directive EQU qui indique à l'assembleur que le nom de cette constante doit être remplacée par sa valeur au moment de la création de l'exécutable.

Exemples

var1 EQU 50h MOV AX, var1 EQU construit une constante dont la valeur est immédiate (et normalement invariante).

Remarque

- Il peut être intéressant de connaître l'adresse d'une variable. Celle-ci se décompose en 2 parties :
  - Adresse du segment, obtenue par l'opérateur SEG.
  - Déplacement à l'intérieur du segment, accessible grâce à l'opérateur OFFSET ou l'instruction LEA (Load Effective Address).
- L'instruction `LEA CX, var1` sera traduite en `MOV CX, 00100h` (avec par exemple `00100h` étant la valeur du déplacement pour la variable `var1`).
- Exemples var1 DB 50h MOV AX, SEG var1 MOV BX, OFFSET var1

Les tableaux

- Un tableau peut être vu comme une liste de variables qui sont associées à un même nom.
- Exemple de création d'un tel tableau :

```
tab DB 01h , 02h , 03h
```

- Il est alors possible d'accéder à une case précise du tableau en utilisant un opérateur d'indexation : nous écrivons le nomdu tableau suivi du numéro de la case placé entre crochet (les cases étant numérotées de 0 à N-1 pour un tableau comportant N éléments):

```
MOV AX, tab[1]
```

- Si un tableau est construit en répétant une ou plusieurs valeurs, il peut être utile d'utiliser l'opérateur DUP :

```
tab DB 5 DUP (1,2)
```

L'opérateur DUP nécessite 2 informations: le nombre de répétition (un entier placé après DB) et la liste des valeurs à répéter, que nous plaçons entre parenthèses.

Les chaînes de caractères

- Une chaîne de caractères peut être vue comme un tableau contenant une suite de code ASCII séparé par une virgule (correspondant chacune à une lettre).
- Dans tous les cas, et pour faciliter les parcours et les recherches, les chaînés de caractères seront terminées par le symbole $(pour marquer la fin de la chaîne de caractères).
- Exemple

```
var5 DB 'H','E','L','L','O','$'
var6 DB 'HELLO$'
```

Les directives

LES DIRECTIVES ne sont pas des instructions du 8086, elles sont destinées à l'assembleur qui les utilise pour savoir de quelle manière il doit travailler.

BITS : Cette directive indique à l'assembleur s'il doit produire un code 16 bits ou un code 32 bits.
Elle est utilisée sous la forme : BITS 16 ou BITS 32.

ORG : Cette directive indique à l'assembleur l'adresse à partir de laquelle le programme sera implanté dans la RAM.
Pour les programmes ".com", cette adresse doit être 100h.

SEGMENT : Cette directive précise dans quel segment, les instructions ou les données qui la suivent seront assemblées.

Les expressions

+, −, *: addition, soustraction et multiplication

/ : division non signée

% : modulo non signée

& : ET logique

| : OU logique

^ : XOR logique

Les expressions

Exemples

```
x equ 0F00h
mov ax,(2*x+6)/2
mov ax,(x*8)+(x-4)
```

Attention, les expressions ne peuvent être utilisés qu'avec des constantes.
On ne peut pas avoir des choses du genre : `MOV AX , DX+2`

Structure d'un programme (.asm)

Structure d'un programme (.asm)

```
Pile segment ;d´eclaration d'un segment de Pile
dw 256 dup(0); ici on r´eserve la taille de la pile (Pile)
Ends ; fin de la d´eclaration de la Pile

Data segment ; d´eclaration d'un segment de donn´ees qui va contenir les variables
; Ici on d´eclare les variables
Ends ; fin de la d´eclaration de donn´ees

Code segment ;d´eclaration du segment de Code qui va contenir le Code
; Ici on d´eclare les fonctions locales, globales et externes
Main:  ;Point d'entr´ee du code
Assume CS:Code, DS:Data, SS: Pile
;Assume permettant d'initialiser les registres de segments CS, DS et SS
MOV AX, Data ; Initialisation du segment de donn´ees
MOV DS, AX
.MOV AX, Pile ; Initialisation du segment de Pile
MOV SS, AX
.-------------------------------------------------------------------------------
; A` partir d'ici on peut placer nos lignes de code
.-------------------------------------------------------------------------------
Fin: ;Arrˆet de programme
MOV AH, 4Ch ; l´equivalent du return (0) en C INT 21h ; appel au MS-DOS
Ends ;fin de la d´eclaration du Code

End Main ; fin du point d'entr´ee du Code
```

Programme assembleur affichant ”Bonjour” à l’écran

```
;Segment de donn´ees pour stocker les donn´ees initialis´ees
data segment
message db ’Bonjour $’ ; Stocker le message dans une variable de type cha^ıne de caract`eres
data ends
-------------------------------------------------------------------------------
; Segment de code
code segment
assume cs:code, ds:data ; Assigner les segments de code et de donn´ees aux registres de segment
appropri´es
start:      ; Le point d’entr´ee du programme est ´etiquet´e comme ”start”
mov ax, data      ; Charger l’adresse de la section de donn´ees dans le registre AX
mov ds, ax      ; Charger l’adresse de DS avec la valeur de AX pour acc´eder aux donn´ees
lea dx, message    ; Charger l’adresse du message dans le registre DX
mov ah, 09h      ; Charger la fonction d’affichage de cha^ıne de caract`eres dans AH
int 21h     ; Appeler l’interruption 21h pour afficher la cha^ıne de caract`eres
mov ah, 4ch     ; Charger la fonction de fin de programme dans AH
int 21h     ; Appeler l’interruption 21h pour terminer le programme
code ends     ; Fin du segment de code
end start      ; D´eclarer le point de fin du programme et sp´ecifier le point d’entr´ee (start)
-------------------------------------------------------------------------------
```

Modes d'adressage

- Adressage immédiat
- Adressage registre
- Adressage direct
- Adressage indirect
  - Adressage basé
  - Adressage indexé
  - Adressage basé indexé

Rseg : Registre Segment

Roff : Registre d'offset

Off : Offset de l'adresse

Ri : Registre d'indexe

Modes d'adressage

- Les opérandes des instructions du microprocesseur 8086 peuvent être des registres, des constantes ou le contenu de cases mémoire $\Longrightarrow$ Modes d'adressage.
- Un mode d'adressage est un mécanisme utilisé par un processeur pour accéder à des données et des instructions en mémoire. Les modes d'adressage définissent les méthodes utilisées pour spécifier les adresses des opérandes dans les instructions du processeur.
- Les modes d'adressage les plus utilisés sont : mode immédiat, mode registre, mode direct, mode indirect (basé,indexé, basé-indexé)

Mode d'adressage immédiat

- Dans ce mode, l'opérande est directement spécifié dans l'instruction elle-même.
- Le CPU n'a pas besoin d'accéder à la mémoire pour récupérer la valeur de l'opérande.
- Exemples :

```
MOV AH,12H (donn´ee de 8 bits)
MOV BX,FFFFH (donn´ee de 16 bits)
```

Mode d'adressage registre

- Dans le mode d'adressage registre, l'opérande est spécifié en utilisant le nom d'un registre.
- L'opération se fait sur un ou 2 registres.
- Exemples

```
INC AX ; incr´ementer le registre AX
MOV AX, BX ; Copier le contenu de BX dans AX
```

Mode d'adressage direct

- Un des deux opérandes se trouve en mémoire.
  L'adresse de la case mémoire ou plus précisément son Offset est précisé directement dans l'instruction.
- L'adresse Rseg:Off doit être placée entre [ ], si le segment n'est pas précisé, DS est pris par défaut.
- Exemples

```
MOV AX, [243] ; Copier le contenu de la m´emoire d'adresse DS:243 dans AX
MOV [123], AX ; Copier le contenu de AX dans la m´emoire d'adresse DS:123
MOV AX, [SS:243] ; Copier le contenu de la m´emoire SS:243 dans AX
```

Modes d'adressage indirect

- Un des deux opérandes se trouve en mémoire.
  L'offset de l'adresse n'est pas précisé directement dans l'instruction, il se trouve dans l'un des 4 registres d'offset BX, BP, SI ou DI et c'est le registre qui sera précisé dans l'instruction : **[Rseg : Roff]**. Si **Rseg** n'est pas spécifié, le segment par défaut sera utilisé selon le tableau suivant :

Offset utilisé	Valeur	DI	SI	BX	BP
Registre Segment par défaut qui sera utilisé par le CPU		DS			SS

- Exemples

```
MOV AX, [BX] ;Charger AX par le contenu de la mémoire d'adresse DS:BX
MOV AX, [BP] ;Charger AX par le contenu de la mémoire d'adresse SS:BP
MOV AX, [SI] ;Charger AX par le contenu de la mémoire d'adresse DS:SI
MOV AX, [DI] ;Charger AX par le contenu de la mémoire d'adresse DS:DI
MOV AX,[ES:BP] ;Charger AX par le contenu de la mémoire d'adresse ES:BP
```

Mode d’adressage indirect

Dans le mode d’adressage indirect , selon le registre d’offset utilisé, on distingue :

- l’adressage basé (avec/sans déplacement) ;
- l’adressage indexé (avec/sans déplacement) ;
- l’adressage basé indexé (avec/sans déplacement).

Adressage basé

- L'offset se trouve dans l'un des deux registres de base BX ou BP.
- On peut préciser un déplacement qui sera ajouté au contenu de Roff pour déterminer l'offset.

- Exemples

```
MOV AX, [BX] ; Charger AX par le contenu de la m'emoire d'adresse DS:BX
MOV AX, [BX+3] ; Charger AX par le contenu de la m'emoire d'adresse DS:BX+3
MOV AX, [BP-100]; Charger AX par le contenu de la m'emoire d'adresse SS:BX-100
MOV AX, [ES:BP] ; Charger AX par le contenu de la m'emoire d'adresse ES:BP
```

Adressage indexé

- L'offset se trouve dans l'un des deux registres d'index SI ou DI.
- On peut préciser un déplacement qui sera ajouté au contenu de Ri pour déterminer l'offset.

- Exemples

```
MOV AX, [SI] ; Charger AX par le contenu de la mémoire d'adresse DS:SI
MOV AX, [SI+200]; Charger AX par le contenu de la mémoire d'adresse DS:SI+200
MOV AX, [DI-7] ; Charger AX par le contenu de la mémoire d'adresse DS:ID-7
MOV AX, [ES:SI+4]; Charger AX par le contenu de la mémoire d'adresse ES:SI+4
```

Adressage basé indexé

- L'offset de l'adresse de l'opérande est la somme d'un registre de **base**, d'un registre **d'index** et /ou d'un **déplacement optionnel**.
- Si **Rseg** n'est pas spécifié, le segment par défaut du registre de base est utilisé.
- **Exemples**

```
MOV AX,[BX+SI] ; Charger AX par le contenu de la
                 mémoire d'adresse DS:BX+SI
MOV AX,[BX+DI+7] ; Charger AX par le contenu de la
                   mémoire d'adresse DS:BX+DI+7
MOV AX,[BP+SI-3] ; Charger AX par le contenu de la
                   mémoire d'adresse SS:BP+SI-3
MOV AX,[BP+DI] ; Charger AX par le contenu de la
                 mémoire d'adresse SS:BP+DI
```

Jeu d’instructions

- Instructions de transfert
- Instructions arithmétiques
- Instructions logiques
- Instructions de saut/appel
- Instructions de gestion de la pile

Instruction MOV

- **Syntaxe** : MOV dest, source
- **Description** : *dest* reçoit *source*.
- Exemples :

```
MOV AL, 0     # AL ← 0
MOV n, AL     # n ← (AL)
```

Instruction MOV

Quelques mouvements autoris´es

MOV Registre général, Registre quelconque
MOV Mémoire, Registre quelconque
MOV Registre général, Mémoire
MOV Registre général, Constante
MOV Mémoire, Constante
MOV Registre de segment, Registre général

Instruction MOV

- **Remarque** : Source et Destination doivent avoir la même taille.
- **Exemples**

```
MOV AX, 5 ; Transf´erer 5 vers AX
MOV ES, DX ; Transf´erer le contenu de DX vers ES
MOV AL, BL ; Transf´erer le contenu de BL vers AL
MOV AX, SI ; Transf´erer le contenu de SI vers AX
MOV Beta, CL ; Transf´erer le contenu de CL vers la case
 m´emoire Beta
```

Instruction XCHG

- **Syntaxe** :XCHG Destination, Source
- **Description** : Échange les contenus de Source et de Destination.
- **Quelques mouvements autorisés**: XCHG Registre général, Registre général
  XCHG Registre général, Mémoire
  XCHG Mémoire, Registre général
- Exemples :

  XCHG AL, AH
  XCHG Alpha, BX
  XCHG DX, Beta

Instructions Arithmétiques

- Le 8086 permet d'effectuer plusieurs opérations arithmétiques : l'addition, la soustraction, la multiplication, la division, etc.
- Les opérations peuvent s'effectuer sur des nombres de 8 bits ou de 16 bits signés ou non signés.

Instruction ADD

- **Description :** Addition de deux opérandes.
- **Syntaxe :** ADD Destination, Source
  Destination ← Destination + Source.
  La destination peut être : registre, mémoire.
  La source peut être : registre, mémoire, valeur immédiate.
  Les FLAGS affectés : CF OF AF SF ZF PF
- Exemple

```
MOV AL, 03h
MOV BL, 02h
ADD AL, BL   ;  AL ← AL+BL ←→ AL=3+2=5
Plus la mise à jour des FLAG : CF←0 ; OF←0 ; AF←0 ; SF←0 ; ZF←0 ; PF←1.
```

Instruction INC

- **Description** : Incrémentation d'un opérande.
- **Syntaxe** :
  INC Destination ;
  Destination ← Destination+1.

La Destination peut être : registre, mémoire.
Les FLAGS affectés : OF AF SF ZF PF
CF reste inchangé.

Instruction SUB

- **Description** : Soustraction entre deux opérandes.
- **Syntaxe** : SUB Destination, Source
  Destination ← Destination – Source.
  La Destination peut être : registre, mémoire.
  La Source peut être : registre, mémoire, valeur immédiate.
- Exemples :

```
SUB AL,7 permet de soustraire 7 de la valeur du registre AL
SUB CL,44h permet d'effectuer l'op´eration CL = CL - 44h
SUB AL,BL permet d'effectuer l'op´eration AL = AL - BL
```

Instruction DEC

- **Description** : Décrémentation d'un opérande
- **Syntaxe** :
  DEC Destination ; Destination←Destination – 1
  La Destination peut être : registre, mémoire.
  Les FLAGS affectés : OF AF SF ZF PF

CF reste inchangé.

Instruction CMP

- **Description** : La comparaison entre deux opérandes.
- **Syntaxe** : CMP Op1, Op2 ; Résultat = Op1 - Op2.
  Résultat est un registre virtuel, mais le plus important, c'est la mise à jour des FLAGS. CF OF AF SF ZF PF

  Op1 peut être : registre, mémoire.
  Op2 peut être : registre, mémoire, valeur immédiate.
- Exemples :

```
MOV AL, 40h
MOV AH, 30h
CMP AL, AH
```

La mise à jour des FLAG : CF ← 0 ; OF ← 0 ; AF ← 0 ; SF ←0 ; ZF← 0; PF=0.

On peut dire que AL est plus grand que AH.

Instruction MUL

- **Description** : Multiplication entre l'accumulateur et un opérande.
- **Syntaxe** : MUL Opérande
  Si l'opérande est une valeur sur 8 bits : AX ← AL*Opérande.
  Si l'opérande est une valeur sur 16 bits : DX : AX ← AX*Opérande.
  L'opérande peut être : registre, mémoire.
- Exemple :

```
MOV AL, 20h
MOV CL, 02h
MUL CL; AX ← AL*CL. AX=20h*02h=40h.
```

Instruction DIV

- Description : Division entre l'accumulateur et un opérande (non signé).
- Syntaxe : DIV Opérande.
  Si l'opérande est sur 8 bits : AL← AX / Opérande ; AH ← Reste.
  Si l'opérande est sur 16 bits : AX ← (DX : AX) / Opérande ; DX ← Reste.).
- Exemple :

```
MOV DX, 0000h
MOV AX, 2000h
MOV CX, 0002h
DIV CX
```

AX ←(DX: AX) / CX
AX ← (00002000h)/2=1000h.
DX ← 0000h.

Exemples

a = b + c;

```
MOV AL, b #AL ← (b)
ADD AL, c # AL ← (AL) + (c)
MOV a, AL #a ← (AL)
```

Exemples

d = a – e;

```
MOV AL, a # AL ← (a)
SUB AL, e # AL ← (AL) - (e)
MOV d, AL # d ← (AL)
```

Exemples

f = (g + h) – (i + j);

```
MOV AL, i  # AL ← (i)
ADD AL, j  # al (AL) + (j)
MOV BL, AL # BL← (AL)
MOV AL, g # AL + (g)
ADD AL, h # AL ← (AL) + (h)
SUB AL, BL # AL (AL) - (BL)
MOV f, AL # f← (AL)
```