Architecture de Von Neumann

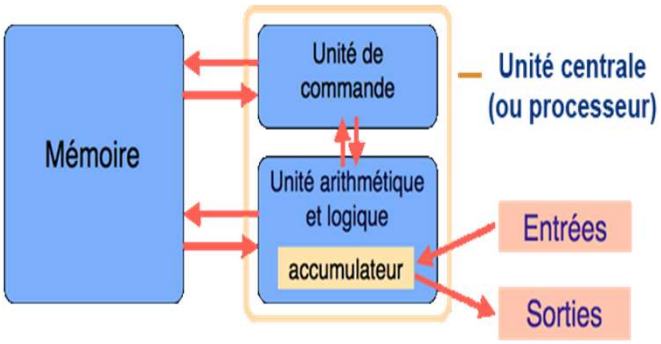
Architecture de Von Neumann

- il faut insérer le microprocesseur dans un système adéquat afin de manipuler l'information.
- **John Von Neumann** a construit en 1946 un modèle de machine universelle de traitement programmé de l'information.
- Cette architecture est la base de la plupart des systèmes à microprocesseur actuel. Elle est composé de:
  - L'unité centrale de traitement (le processeur)
  - La mémoire
  - Les dispositifs d'entrée-sortie
- Ces différents éléments sont interconnectés à travers des **bus** qui constituent le support d'acheminement de l'information entre les différents composants.

258

Architecture de Von Neumann

Architecture de Von Neumann



259

Architecture de Von Neumann

Unité centrale de traitement

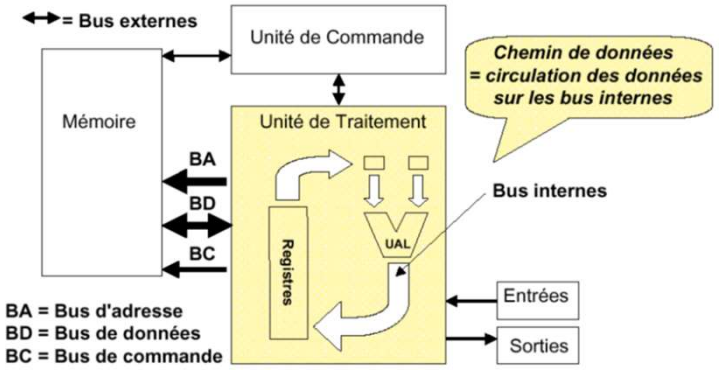
Elle est chargée **d'interpréter** et **d'exécuter** les instructions d'un programme, de **lire** ou de **sauvegarder** les résultats dans la mémoire et de **communiquer** avec les unités d'échange.

- Elle est construite autour de deux éléments principaux :
  - Une **unité de commande** (appelée aussi Unité de commande et de contrôle (UCC)) : elle est responsable de la lecture en mémoire et du décodage des instructions,
  - Une **unité de traitement** : aussi appelée Unité Arithmétique et Logique (UAL) exécute les instructions qui manipulent les données.
- Le microprocesseur est associé à des **registres** chargées de stocker les différentes informations à traiter.

260

Architecture de Von Neumann

Unité centrale de traitement



Architecture de Von Neumann

Unité centrale de traitement

- Pour chaque instruction, le processeur effectue **schématiquement** les opérations suivantes :
  - **Lire** en mémoire principale l'instruction à exécuter
  - **Effectuer** le traitement correspondant
  - **Passer** à l'instruction suivante.

262

Architecture de Von Neumann

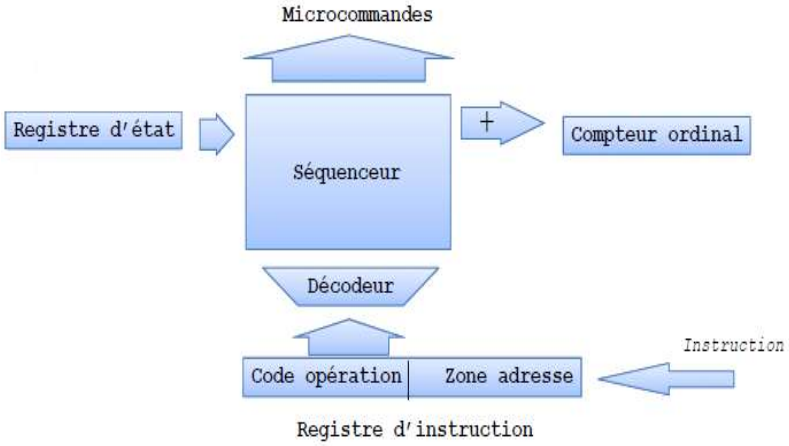
Unité de commande

- Elle **contrôle le transfert** des **instructions** et des **données** entre la mémoire et l'UC et leur exécution
  - 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 façon à ce que tous les circuits électroniques travaillent ensembles.

263

Architecture de Von Neumann

Unité de commande



264

Architecture de Von Neumann

Composants de l'unité de commande

- **Le compteur ordinal 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 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** en fonction des divers **signaux** de **commande** provenant du **décodeur d'instruction** ou du **registre d'état** par exemple.

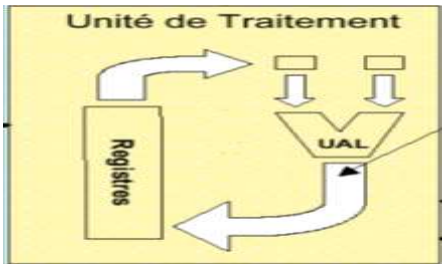
265

Architecture de Von Neumann

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**

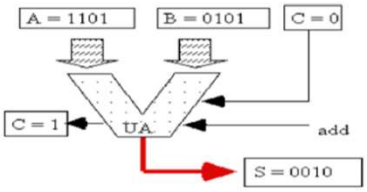


266

Architecture de Von Neumann

L'Unité Arithmétique et Logique (UAL):

- **Elle** assure les **fonctions logiques** ou **arithmétique** pour effectuer les **calculs** et les **opérations logiques** des différentes **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**.



A = 1101
+ B = 0101
S = 0010

267

Architecture de Von Neumann

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 stocker les **résultats** des **opérations arithmétiques** et **logiques** en cours **d'exécution** dans l'UAL.

268

Architecture de Von Neumann

Le registre d'état

- 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 **indicateur 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….

269

Architecture de Von Neumann

Mémoire principale

- Elle contient les **instructions** des **programmes** en cours **d'exécution** et les **données** associées à ce programme.
- Elle se décompose en :
  - Une **mémoire morte** (**ROM** = Read Only Memory) C'est une **mémoire à lecture seule**. Lors d'une coupure du courant électrique, son contenu n'est pas perdu.
  - Une **mémoire vive** (**RAM** = Random Access Memory) chargée de **stocker** les **données intermédiaires ou les résultats de calculs**. On peut lire ou écrire des données dedans, ces données sont **perdues** à la mise hors tension.

270

Architecture de Von Neumann

Interface d'entrée / sortie

- Elles permettent d'assurer la **communication** entre le **microprocesseur** et les **périphériques** : capteur, clavier, moniteur ou afficheur, imprimante, modem, etc…..
- Sert **d'interface** avec les **périphériques**.
- Les **opérations associées** (lecture et/ou écriture) sont **fonctions du périphérique.**

271

Architecture de Von Neumann

BUS

- Un bus est un **ensemble de fils électrique** qui assure la **transmission** des **informations** entre les différentes unités.
- **Largeur du bus** = **nombre de fils constituant le chemin** = nombre **d'impulsions** électriques pouvant être **envoyés** en **parallèle** (en même temps).
- Il existe **trois types de bus** véhiculant des informations :
  - **Bus de données**
  - **Bus d'adresses**
  - **Bus de commande (ou de contrôle)**

272

Architecture de Von Neumann

BUS

- **Bus de données** : **bidirectionnel** qui assure le **transfert** des **informations** entre le **microprocesseur** et son **environnement**, et inversement. Son nombre de lignes est égal à la capacité de traitement du microprocesseur.
- **Bus d'adresses** : **unidirectionnel** qui permet la **sélection** des **informations** à traiter dans un **espace mémoire** (ou espace adressable) qui peut avoir 2^n emplacements, avec n = nombre de conducteurs du bus d'adresses.
- **Bus de commande** : **transporte** les **ordres** et les **signaux** de **synchronisation** en provenance de **l'unité** de **commande** et à destination de l'ensemble des **composants** matériels. Il s'agit d'un bus **bidirectionnel** dans la mesure où il transmet également les **signaux** de **réponse** des éléments **matériels**.

273

Unités fonctionnelles

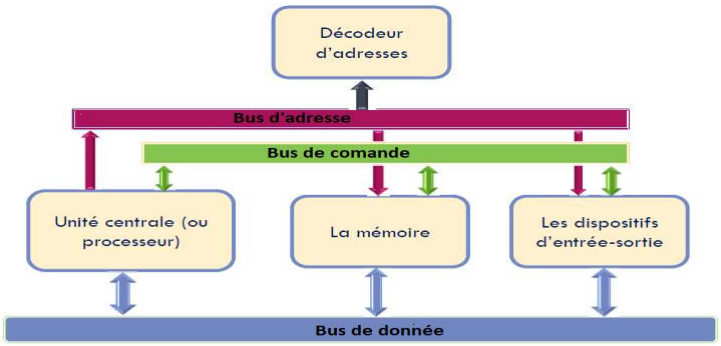
Décodeur d'adresse

- Il est chargé **d'aiguiller** les **données** présentes sur le **bus de données** provenant des multiples **périphériques** .
- Le **microprocesseur** peut **communiquer** avec les différentes **mémoires** et les différents **périphériques**.
- Ces éléments sont tous reliés sur le même **bus de données** et afin d'éviter des **conflits**, un seul composant doit être **sélectionné** à la **fois**.
- Attribue à chaque **périphérique** une **zone d'adresse** et une fonction **«décodage d'adresse »** est donc nécessaire afin de fournir les **signaux** de **sélection** de chacun des **composants**.

274

Architecture de Von Neumann

Décodage d'adresse



275

Architecture de Von Neumann

Mémoire

- La **mémoire** peut être vue comme **un ensemble de cellules** ou cases contenant chacune une information :
- Chaque **case mémoire** est repérée par un **numéro d'ordre unique** : **son adresse**. Elle est exprimée généralement en **Hexadécimale**.
- Chaque case mémoire est **divisée** en **emplacements** de **taille fixe** (par exemple 8 bits)

276

Architecture de Von Neumann

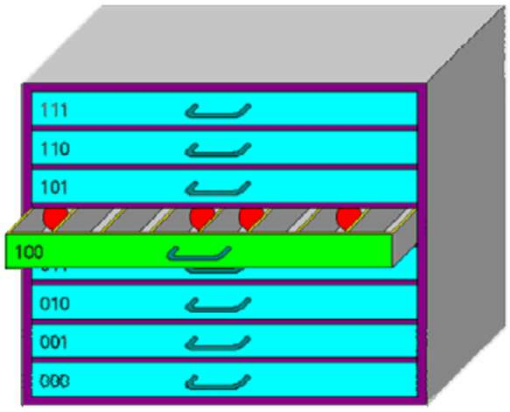
Mémoire

- C'est **un vecteur** dont chaque **composante** est **accessible** par une **adresse**.
- Les **opérations** permises sur la **mémoires** sont les opérations de **lecture** et **d'écriture**.
- L'UC **inscrit l'adresse** d'une cellule dans un **registre d'adresse** (RA) et **demande** une **opération de lecture ou d'écriture**. Les **échanges** se font par **l'intermédiaire** d'un **registre de mot** (RM).
  - **Lecture**: **RA ←adresse; RM←mémoire[RA]**
  - **Écriture**: **RM←valeur; RA←adresse; mémoire[RA]←RM**

277

Architecture de Von Neumann

Mémoire



278

Architecture de Von Neumann

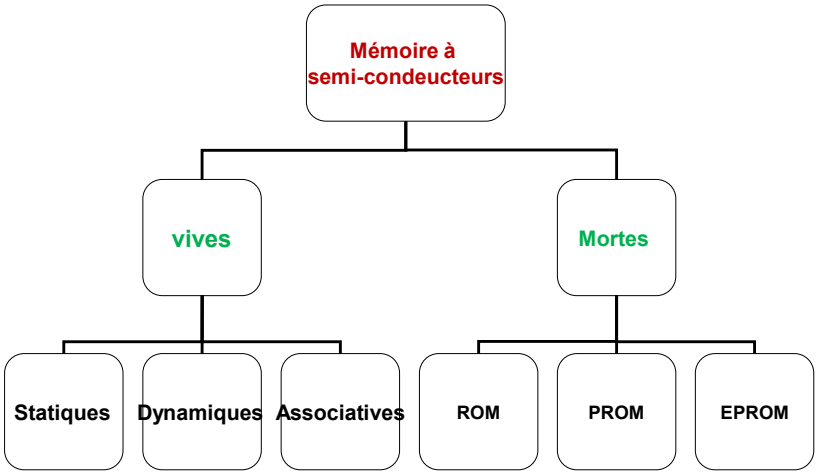
Classification des mémoires

- Les mémoires peuvent être classée en trois catégories selon la technologie utilisée :
  - Mémoire à **semi-conducteur** (mémoire centrale, ROM, PROM,…..) : très rapide mais de taille réduit.
  - Mémoire **magnétique** (disque dur, disquette,…) : moins rapide mais stock un volume d'informations très grand.
  - Mémoire **optique** (DVD, CDROM,..)

279

Architecture de Von Neumann

Mémoire à semi-conducteur



280

Architecture de Von Neumann

Mémoire centrale

- La mémoire centrale (MC) représente l'espace de travail de l'ordinateur (calculateur).
- C'est l'organe principal de rangement des informations utilisées par le processeur.
- Dans une machine (ordinateur / calculateur) pour exécuter un programme il faut le charger (copier) dans la mémoire centrale .
- Le temps d'accès à la mémoire centrale et sa capacité sont deux éléments qui influent sur le temps d'exécution d'un programme (performance d'une machine).

281

Architecture de Von Neumann

Caractéristiques de la mémoire centrale

- La mémoire centrale est réalisée à base de semi-conducteurs.
- La mémoire centrale est une mémoire vive : accès en lecture et écriture.
- La mémoire centrale est dite à accès aléatoire (RAM : Random Acces Memory) c'est-à-dire que le temps d'accès à l'information est indépendant de sa place en mémoire.
- La mémoire centrale est volatile : la conservation de son contenu nécessite la permanence de son alimentation électrique.
- Un temps d'accès à une mémoire centrale est moyen mais plus rapide que les mémoires magnétiques .
- La capacité d'une mémoire centrale est limitée mais il y a toujours une possibilité d'une extension.
- Pour la communication avec les autres organes de l'ordinateur, la mémoire centrale utilise les bus (bus d'adresses et bus de données)

282

Architecture de Von Neumann

Types des mémoires centrales

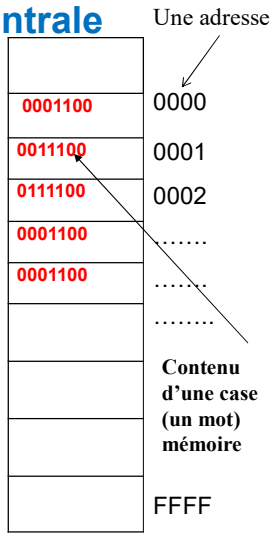
- Il existe deux grandes familles des mémoires centrales : les mémoires statiques (SRAM) et les mémoires dynamiques (DRAM).
  - Les mémoires statiques sont à base de bascules de type D , elles possèdent un faible taux d'intégration mais un temps d'accès rapide (Utilisation pour les mémoires cache).
  - Les mémoires dynamiques à base de condensateurs , ces mémoires possèdent un très grand taux d'intégration, elleS sont plus simples que les mémoires statiques mais avec un temps d'accès plus long .

283

Architecture de Von Neumann

Vue logique de la mémoire centrale

- La mémoire centrale peut être vu comme un large vecteur (tableau) de mots ou octets.
- Un mot mémoire stocke une information sur n bits.
- un mot mémoire contient plusieurs cellules mémoire.
- Une cellule mémoire stock 1 seul bit .
- Chaque mot possède sa propre adresse.
- Une adresse est un numéro unique qui permet d'accéder à un mot mémoire.
- Les adresses sont séquentielles (consécutives)
- La taille de l'adresse (le nombre de bits) dépend de la capacité de la mémoire.



284

Architecture de Von Neumann

Mémoire: BIG/LITTLE ENDIAN

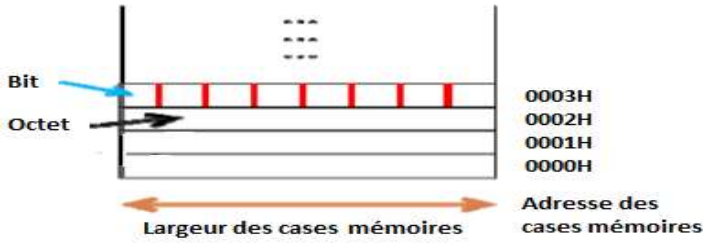
- La **taille minimum** d'une **donnée** stockée sur un disque est habituellement **1 byte.**
- Les données ayant **plusieurs bytes** (integer, long, float) peuvent être **emmagasinée** de 2 façons:
- **Little endian**: le byte le **moins** significatif est placé à la plus **petite adresse** dans la mémoire
- **Big endian**: le byte le **moins** significatif est placé à la plus **haute adresse** dans la mémoire

285

Architecture de Von Neumann

Mémoire

- Le **nombre** de **fils d'adresses** d'une mémoire définit le **nombre de cases mémoires** qu'elle **comprend**.
- Le **nombre** de **fils de données** définit la **taille des données** que l'on peut **sauvegarder** dans chaque **case mémoire**.



286

Architecture de Von Neumann

Fonctionnement de la mémoire

1.Choisir l'adresse en mémoire qui donne **l'accès** à un **emplacement mémoire** pour une **opération de lecture ou d'écriture**

2.Choisir une opération de **lecture** ou **d'écriture** (**R/W**)

3.Sélectionner le boîtier **mémoire** avec le bit **Chip Select** (CS)

4.Valider ou invalider la mémoire : Le **signal** de **sélection** du **boîtier** (CS) **valide** les **fonctions d'écriture et de lecture**.

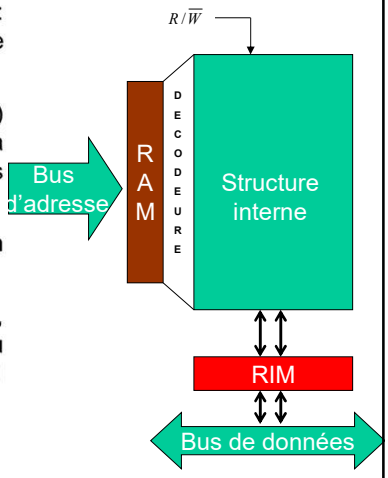
- Le boîtier est bloqué pour **CS = 1**.
- Lorsque **CS = 0**, une fonction **d'écriture** correspond à **$R/\overline{W}=0$** et une fonction de **lecture** à **$R/\overline{W}=1$**.

287

Architecture de Von Neumann

Structure physique d'une mémoire centrale

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



288

Architecture de Von Neumann

Mémoire

- Si **les adresses des cases mémoires** sont codées sur **n bits**, il est possible de référencer au plus **2^n cases mémoire**.
  - **Exemple** : si les adresses des cases mémoires sont codées sur **4 bits**, il est alors possible d'avoir **2^4 = 16 cases mémoire**
- Chaque **case** est remplie par un **mot de données**
  - Un **mot est l'unité d'information** accessible en une **seule opération de lecture** (sa taille varie en fonction de la machine).
  - Un mot **représente** un **regroupement de bits**
  - Sa **longueur** est toujours une **puissance de 2**
- **Octet (byte) = 8 bits et Bit = 0/1**

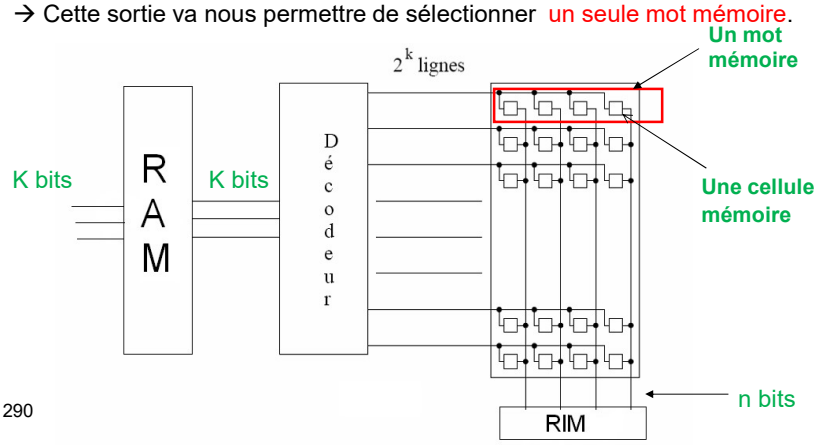
289

Architecture de Von Neumann

Sélectionner un mot mémoire

- Lorsque une adresse est chargée dans le registre RAM , le décodeur va recevoir la même information que celle du RAM.
- A la sortie du décodeur nous allons avoir une seule sortie qui est active

→ Cette sortie va nous permettre de sélectionner un seule mot mémoire.



290

Architecture de Von Neumann

Calculer la capacité d'une MC

- Soit k la taille du bus d'adresses (taille du registre RAM)
- Soit n la taille du bus de données (taille du registre RIM ou la taille d'un mot mémoire)
- On peut exprimer la capacité de la mémoire centrale soit en nombre de mots mémoire ou en bits (octets, kilo-octets,….)
  - La capacité = 2^k Mots mémoire
  - La capacité = 2^k * n Bits
- **Exemple** :

•Dans une mémoire la taille du bus d'adresses K=14 et la taille du bus de données n=4. Calculer la capacité de cette mémoire ?

C=2^{14} = 16384 Mots de 4 bits

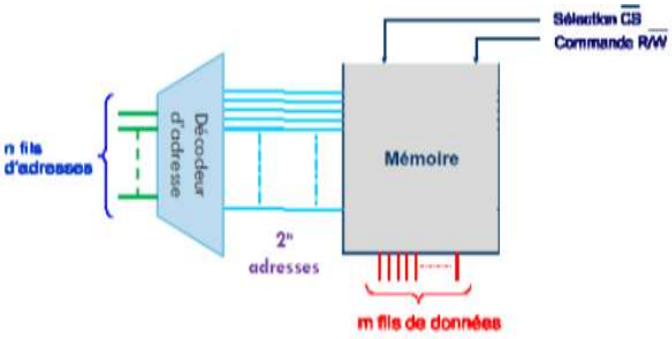
C= 2^{14} * 4 = 65536 Bits = 8192 Octets = 8 Ko

291

Architecture de Von Neumann

Mémoire

- Un **boitier mémoire** comprend en plus des **fils d'adresses et de données**:



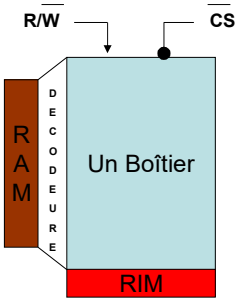
292

Architecture de Von Neumann

Mémoire

- Une **entrée de commande** $R/\overline{W}$ qui permet de définir le **type d'action** que l'on effectue avec la **mémoire** (**lecture/écriture**).
- Une **entrée de sélection $\overline{CS}$ est une commande négative** qui permet de mettre les **entrées/sorties** du boitier en **haute impédance** (pour la **sélection de la mémoire correspondante**)

- $\overline{CS}$ =0 le boîtier est sélectionné
- $\overline{CS}$ =1 le boîtier n'est pas sélectionné



293

Architecture de Von Neumann

Caractéristiques de de la mémoire

- **Capacité** : c'est le **nombre total de bits** que contient la mémoire. Elle s'exprime aussi souvent en **octet** (**byte** en anglais). Si nous avons **m bits** dans chaque **case mémoire** et s'il existe **2^n cases mémoire** alors la capacité est : **$C = m*2^n$**
  - **1 K : Kilo octet = 2^{10} ~10^3**
  - **1 M : Méga octet = 2^{20} ~10^6**
  - **1 G : Giga octet = 2^{30} ~10^9**
  - **1 T : Téra octet = 2^{40} ~10^{12}**
- **Format des données** : c'est le **nombre de bits** que l'on peut **mémoriser par case mémoire**. On dit aussi que c'est la **largeur du mot** mémorisable.

294

Architecture de Von Neumann

Caractéristiques de de la mémoire

- **Temps d'accès** : c'est le temps qui s'écoule entre l'instant où a été lancée une **opération** de **lecture/écriture** en mémoire et l'instant où la **première information est disponible** sur le bus de données.
  - **1 ms : Milli second =10^{-3}s**
  - **1 µs : Micro second =10^{-6}s**
  - **1 ns : Nano second =10^{-9}s**
  - **1 ps : Pico second =10^{-12}s**
- **Temps de cycle** : il représente **l'intervalle minimum** qui doit séparer deux **demandes successives** de **lecture ou d'écriture.**

295

Architecture de Von Neumann

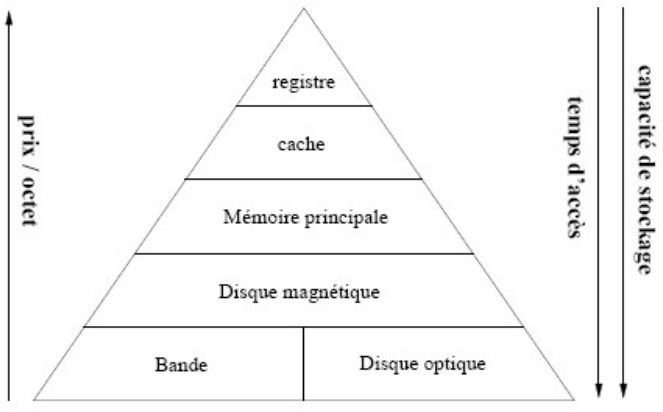
Caractéristiques de de la mémoire

- **Débit** : c'est le **nombre maximum** d'informations **lues ou écrites par seconde.**
- **Volatilité** : elle caractérise la **permanence** des **informations** dans la **mémoire**. L'information stockée est **volatile** si elle risque d'être altérée (effacée) par un défaut d'alimentation électrique et **non volatile** dans le cas contraire

296

Architecture de Von Neumann

Hiérarchie de mémoire



Architecture de Von Neumann

Hiérarchie de mémoire

- **Registres** : **Mémoires très rapides** situées au niveau du **processeur**. Les registres sont utilisés pour assurer le **stockage temporaire** d'informations nécessaires à l'exécution de l'instruction en cours de traitement.
- **Mémoire cache** : une **mémoire rapide** de **faible capacité** destinée à **accélérer l'accès** à la mémoire centrale en stockant les données les plus utilisées.
- **Mémoire principale** (MP) : elle est utilisée pour le **rangement** des informations. Elle contient les **programmes** (**instructions et données**) à utiliser et est plus **lente** que les deux mémoires précédentes.

298

Architecture de Von Neumann

Hiérarchie de mémoire

- **Mémoire d'appui** : **Mémoire tampon** qui se situe **entre** la **MP** et la **mémoire** de **masse**. (Même rôle que la mémoire cache)
- **Mémoire de masse (auxiliaires)** : est une **mémoire périphérique** de **grande capacité** utilisée pour le stockage **permanent** ou la **sauvegarde** des **informations**. Elle utilise des supports **magnétiques** (disque dur) ou optiques (CDROM, DVDROM,…)

299

Architecture de Von Neumann

classification de mémoire

- **Mémoires de travail :**

Elles désignent les **mémoires** qui sont **actives** dans **l'exécution** d'un programme: les **registres** du **processeur**, la mémoire centrale ou **vive** (RAM), la mémoire **cache** et la mémoire **morte** (électroniques).

- **Mémoires de stockages :**

Elles permettent de **conserver** de manière **permanente** de **grandes** quantités **d'informations**. Ces informations **ne participent pas directement** à **l'exécution** d'un programme mais doivent être **chargées** en mémoire centrale pour être **exploitées** par le **processeur** (magnétique ou optique).

300

Architecture de Von Neumann

Mémoire morte

- **ROM** est acronyme de **Read Only Memory** qui veut dire une mémoire à **lecture seule**.
- Ce genre de mémoire est **entièrement** et **définitivement** réalisé au stade de **fabrication** par le constructeur, donc son contenu **ne peut pas être modifié** (effacé) sans un appareil spécial « Programmeur de ROM ».
- Cette mémoire est composée d'une **matrice** dont la programmation s'effectue en reliant les **lignes** aux **colonnes** par des **diodes**. **L'adresse** permet de sélectionner une **ligne** de la **matrice** et les **données** sont alors reçues sur les **colonnes** (le **nombre** de **colonnes** fixant la **taille** des **mots** mémoire).

301

Architecture de Von Neumann

Mémoire vive

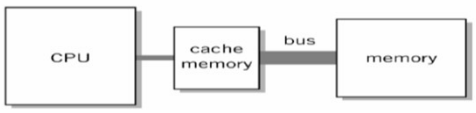
- Mémoire dite **RAM** (**Random Access Memory**), où le **temps d'accès** est **indépendant** de la **place** de **l'information** dans la **mémoire**.
- Elle sert au **stockage temporaire** de **données** et sur lesquelles les **opérations** de **lecture** et **d'écriture** sont possibles.
- Elle est **volatile** : **l'information** est **perdue** en cas de **coupure d'alimentation**. Mais le **temps d'accès** et **consommation électrique** sont **faibles** pour ne pas **ralentir** le **microprocesseur**
- Essentiellement **utilisée** en tant que mémoire **centrale** et mémoire **cache**.

302

Architecture de Von Neumann

Mémoire cache

- **Vitesse** du **processeur** est plus **rapide** que la mémoire **principale**
- La mémoire **cache** permet de **réduire** les **délais d'attente** des **informations** stockées en mémoire **vive**.
- Cette mémoire est utilisée comme **mémoire virtuelle**, **invisible** pour le **système d'exploitation** et à **proximité** du **processeur**. Il y stocker **temporairement** les principales **données** devant être traitées par le processeur **augmentant** ainsi la **vitesse d'accès**.



303

Architecture de Von Neumann

Augmentation de la mémoire

- **Pagination de la mémoire**
  - **Minimise** le nombre de **dépendance** d'accès au mémoire
  - **Augmente** la **vitesse** d'accès
- **Segmentation de la mémoire**:
  - **diviser** la mémoire en **plusieurs parties**
  - **Offre** la possibilité d'accès en **lecture/écriture** au même temps
  - **Augmente** la **vitesse** d'accès

304

Processeur 80x86

Description physique

- Le **microprocesseur Intel 8086** est un **microprocesseur 16 bits**, apparu en **1978**.
- C'est le **premier** microprocesseur de la **famille Intel 80x86** (8086, 80186, 80286, 80386, 80486, Pentium, ...) qui est devenue **l'architecture** de processeur la **plus répandue** dans le monde des **ordinateurs personnels**, **stations** de **travail** et **serveurs informatiques**.
- **Améliorations**
  - **Augmentation** de la **fréquence d'horloge**, de la **largeur** des **bus d'adresses** et de **données**
  - **Ajout de nouvelles instructions** et de **registres**

305

Processeur 80x86

Description physique

- Il est base sur des **registres 16 bits**, Il dispose de
  - Un **bus** externe **de données** de **16 bits** (D0 – D15),
  - Un **bus d'adresse** de **20 bits** (A0 – A19) qui permet d'adresser 1 Mo.
  - Un bus de **Données/Adresses multiplexe**
- Il contient **29 000 transistors gravés** en **3 µm**.
- Sa **puissance de calcul** varie de **0,33 MIPS** (lorsqu'il est cadencé à **4,77 MHz** comme dans **l'IBM PC**) jusqu'à **0,75 MIPS** pour la version **10 MHz**.

306

Processeur 80x86

Caractéristiques

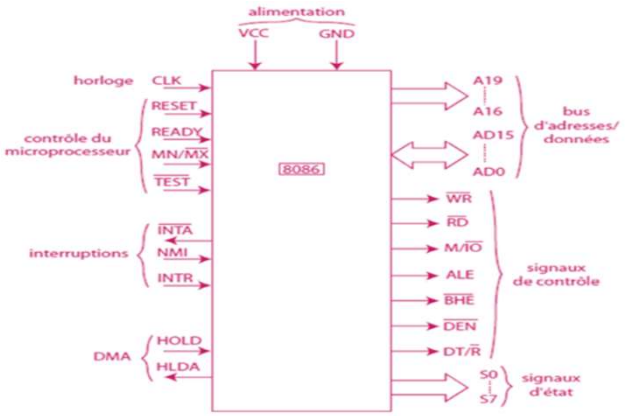
- Le **microprocesseur 8086** se **caractérisé** par :
  - **Structure 16 bits.**
  - **Capacité d'adressage de 1 Mo.**
  - **14 registres internes de 16 bits.**
  - **7 modes d'adressage.**
  - **Operations sur des bits, des octets, des mots et des chaines de caractères.**
  - **Arithmétique signée ou non signée.**
  - **Arithmétique binaire ou BCD, sur 8 ou 16 bits.**

307

Processeur 80x86

Caractéristiques

❑



308

Processeur 80x86

Composition interne

❑ Le **8086** est **constitue** de deux **unités de traitement fonctionnant** en **parallèle** :

❑ **l'unité d'exécution** (**EU** : **Execution Unit**) ;

❑ **l'unité d'interface de bus** (**BIU** : **Bus Interface Unit**).

➔ **fonctionnent simultané**

➔ **une accélération du processus d'exécution**

309

Processeur 80x86

Unité d'interface de bus (BIU) :

- Elle **comporte** une **file d'attente d'instructions** gérée en **FIFO** (First Input First Output), les **registres de segments** (**CS, DS, SS, ES**) et le **pointeur d'instruction** (**IP**).

Rôle

- **Rechercher** les **instructions à exécuter** dans la mémoire et les **ranger** dans la file d'attente **FIFO**.
- **Calculer** les **adresses physiques** sur **20 bits**.
- **Réaliser** le **transfert** des **données** avec la mémoire.

310

Processeur 80x86

Unité d'exécution (EU) :

- Elle **comporte** l'**UAL**, les **registres généraux** (**AX, BX,CX, DX**), les **registres d'adressage** (**SP, BP, SI, DI**), le **registre d'état** (**Flags**) et le **décodeur d'instructions**.

Rôle

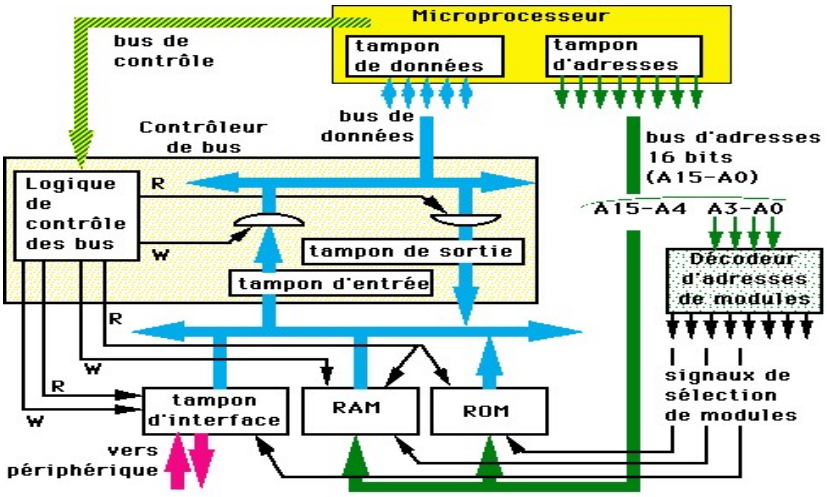
- **Extraire** les **codes des instructions** à partir de la **file d'attente** et les **exécuter**.
- **Fournir** les **adresses des opérandes** à l'BIU en nommant le **segment** concerné et en fournissant le **déplacement** dans ce segment.

311

Processeur 80x86

Exécution

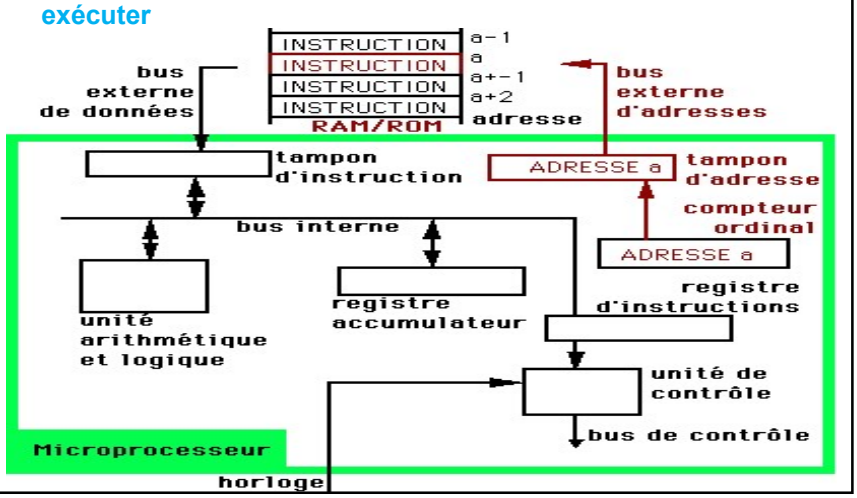
- **Micro-ordinateur** à **mots** de **16 bits** avec **adressage** sur **12 bits**



Processeur 80x86

Exécution

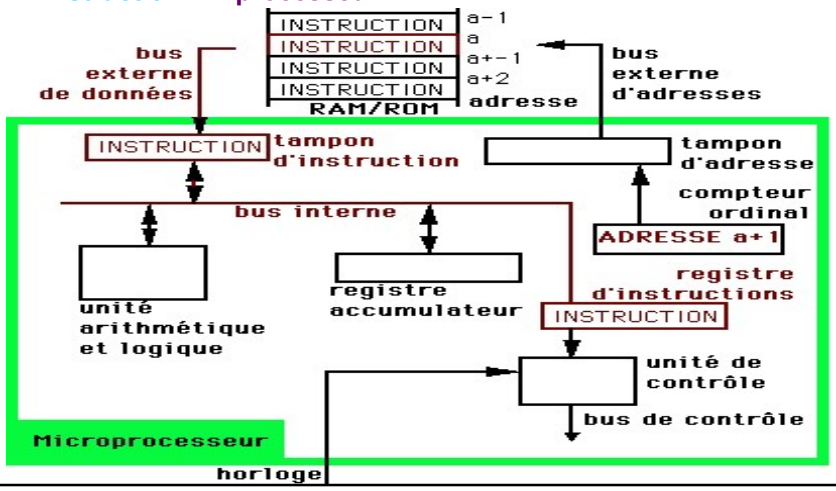
- **Le processeur** va **rechercher** en **mémoire** **l'instruction à exécuter**



Processeur 80x86

Exécution

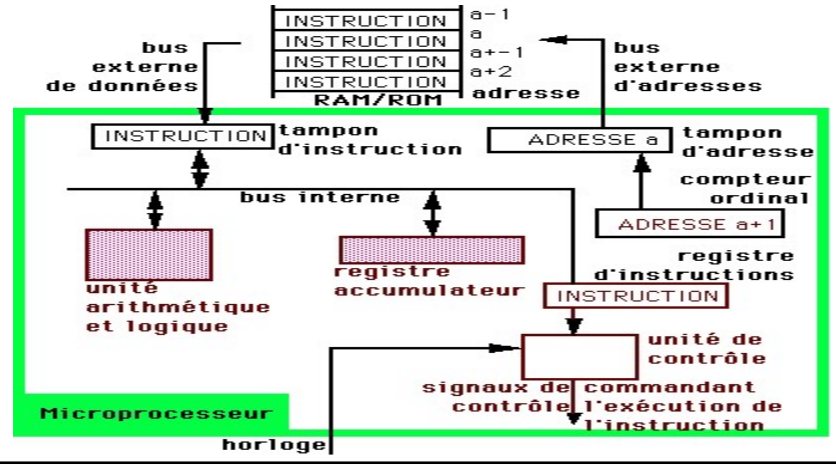
- **l'instruction à exécuter** va être **chargée** dans le **"registre instruction"** du **processeur**



Processeur 80x86

Exécution

- **L'instruction** est **décodée**, pour connaitre son **"code opération"** et ses **"parties adresses",** puis **exécutée**



Processeur 80x86

Registres

- Le **microprocesseur 8086** contient **14 registres** repartis en **4 groupes** :
  - **Registres généraux (AX, BX, CX, DX)**
  - **Registres d'adressage de pointeurs et d'index (SP, BP, SI, DI)**
  - **Registres de segments (CS, DS, SS, ES)**
  - **Registre de Pointeur d'instruction (IP: Le compteur de programme)**
  - **Regnistre d'état (Flag)**

316

Processeur 80x86

Registres généraux (AX, BX, CX, DX) :

- Ils peuvent être **utilisés** dans **toutes** les **opérations arithmétiques** et **logiques** que le **programmeur** insère dans le **code assembleur**.
- Ils **servent** a contenir **temporairement** des **données**.
- Un **registre complet** est **composé** de **16 bits**, divisé en **deux** registres distincts de **8 bits**.
- Ce sont des **registres généraux** mais ils peuvent être **utilisés** pour des **opérations particulières**.

317

Processeur 80x86

Registres généraux (AX, BX, CX, DX) :

- **Accumulateur** (**AX**)
- **Base** (**BX**)
- **Counter** (**CX**)
- **Accumulateur auxiliaire** (**DX**)

AH	AL
BH	BL
CH	CL
DH	DL

318

Processeur 80x86

Le registre AX (Accumulateur) :

- Il permet de **faire** toutes les **opérations** de **transfert** de **données** avec les **mémoires**, le **traitement** des **chaines** de **caractères** et les **opérations arithmétiques** et **logiques**.
- Il permet aussi de **faire** les **conversions** en **BCD** du **résultat** d'une **opération arithmétique** (**addition**, **soustraction**, **multiplication** et **division**)

319

Processeur 80x86

Le registre BX (Registre de base) :

- Il est utilise pour **l'adressage** de **données** dans une **zone mémoire différente de la zone code**
- Il contient, en général, une **adresse** de **décalage** par rapport a une **adresse** de **référence** (**segment de données DS**).
- Il peut aussi servir pour le **stockage intermédiaire** des **données**.

320

Processeur 80x86

Registres de pointeurs

- **Pointeur SP (Stack Pointer) :**
  - **Pointeur de pile** (la pile est une zone de sauvegarde de **données** en **cours d'exécution** d'un programme gérée en LIFO). Il pointe sur la **tête** de la **pile** pour indiquer le **dernier élément**.
  - **Associé** par défaut au **registre de segment SS** (**SS:SP**).
- **Pointeur BP (Base Pointer) :**
  - **Pointeur de base** : utilise pour **accéder** aux **données** de la **pile** lors d'appels de **sous-programmes** (CALL).
  - **Associé** au registre de **segment de la pile SS** (**SS : BP**)

321

Processeur 80x86

Registres d'index

- **Registre d'Index SI (Source Index) :**
  - Il permet de **pointer** sur la **mémoire**, Il forme en général un **décalage** (un **offset**) par rapport a une **base fixe** (le registre **DS**),
  - Il est utilisé aussi dans des **instructions** de **déplacement** de **données** comme **index** de **l'opérande source**.
- **Registre d'Index DI (Destination Index) :**
  - Il permet aussi de **pointer** sur la **mémoire**, il présente un **décalage** par rapport a une **base fixe** (**DS ou ES**),
  - Il sert aussi pour les **instructions** de **chaine de caractères** en **pointant** sur la **destination**.

➔Les **pointeurs** et les **index** contiennent des **adresses** de **cases mémoire**.

322

Processeur 80x86

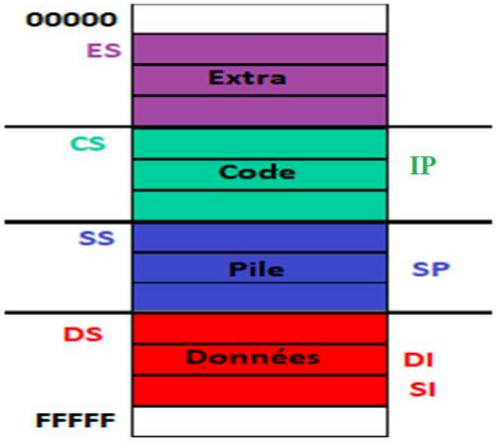
Registres segment

- Ces **registres segment** sont chargés de **sélectionner** les différents **segments** de la **mémoire** en pointant sur le **début** de chacun d'entre eux.
- Chaque **segment de mémoire** ne peut excéder les **2^{16} octets = 64 Ko = 65536 o**
  - **CS : Code Segment** : registre segment de **code** ;
  - **DS : Data Segment** : registre segment de **données** ;
  - **SS : Stack Segment** : registre segment de **pile** ;
  - **ES : Extra Segment** : registre segment **supplémentaire** pour les **données** ;

323

Processeur 80x86

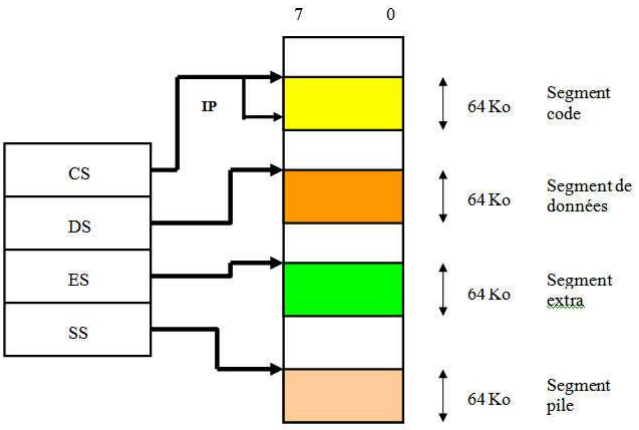
- **Registres segment**



324

Processeur 80x86

- **Registres segment**



325

Processeur 80x86

Registre CS (Code Segment) :

- Il pointe sur le **début** du segment qui contient les **codes** des **instructions du programme en cours**.
- Pour résoudre le **problème des programmes** qui ont une taille **supérieure** à **64 Ko**.
  - **Diviser** le code sur **plusieurs** segments **ne dépassant pas les 64 Ko**
  - **basculer** d'une partie à une autre du programme en **changeant la valeur du registre CS**

326

Processeur 80x86

Registre DS (Data Segment) :

- Le **registre segment de données** DS **pointe** sur le **segment** des **variables globales** du **programme**,
- **Sa taille ne peut excéder 64 K octets** (si on a des données qui **dépassent** cette limite, on utilise la **même astuce** citée précédemment mais dans ce cas on **change la valeur** de **DS**).

327

Processeur 80x86

Registre ES (Extra Segment) :

- Le **registre** de **données supplémentaires ES** est **utilise** par le **microprocesseur** lorsque **l'accès** aux autres **registres** est devenu **difficile** ou **impossible** pour **modifier** des **données**,
- Ce **segment** est **utilisé** aussi pour le **stockage** des **chaines de caractères.**

328

Processeur 80x86

Registre SS (Stack segment) :

- Le **registre SS pointe** sur la **pile**
- La **pile** est une **zone mémoire** où on peut **sauvegarder** les **registres** ou les **adresses** ou les **données** pour pouvoir les **récupérer après l'exécution** d'un **sous-programme** ou d'une **interruption**.

329

Processeur 80x86

Registre IP (Instruction Pointer) :

- Le **pointeur d'instructions** conserve **l'adresse** de **l'emplacement mémoire** de la **prochaine instruction** à **exécuter** pour l'indiquer au **processeur**
- Il est **associé** par défaut au **registre** de **segment CS** (**CS : IP)**
- Le **processeur** effectue les **actions** suivantes pour **chaque instruction** :
  - **Lire et décoder l'instruction à l'adresse IP;**
  - **Incrémenter IP =IP + taille de l'instruction;**
  - **Exécuter l'instruction.**

330

Processeur 80x86

Registre d'état (flags):

- **L'Indicateur d'état** est un **registre** sur **16 bits** qui sert à contenir **l'état** de certaines **opérations** effectuées par le **processeur**.
- Chaque **indicateur** est **manipulé individuellement** par des **instructions spécifiques**.
- Le **registre d'état** du **8086** est **formé** par les **bits** suivants :

331

Processeur 80x86

Registre d'état (Flag)

- **CF (Carry Flag : Retenue) :**
  - Cet indicateur est **positionne à 1** lorsqu'il y a une **retenue** du **resultat** à **8 ou 16 bits**.
  - Il intervient dans les **opérations d'additions** (**retenue**) et de **soustractions** (**emprunt**) sur des **entiers naturels**.
  - Il est **positionne** en particulier par les instructions **ADD**, **SUB** et **CMP** (comparaison entre deux valeurs).
  - **CF = 1** s'il y a une **retenue** après **l'addition** ou la **soustraction** du **bit de poids fort des opérandes**.
- **PF (Parity Flag : Parité) :**
  - Si le **résultat de l'opération** contient un **nombre pair de 1**, cet indicateur est **positionne à 1**, sinon **zéro**.

332

Processeur 80x86

Registre d'état (Flag)

- **AF (Auxiliary Carry : Demie retenue) :**
  - Ce **bit est égal à 1** si on a une **retenue du quarter** (4 bits) de **poids faible** dans le **quarter** de **poids plus fort**.
- **ZF (Zero Flag : Zéro) :**
  - Cet indicateur est **positionné à 1** quand le **résultat** d'une **opération est égal à zéro**.
  - Lorsque une **soustraction** ou une **comparaison** sont effectuées, **ZF = 1** indique que les deux **opérandes** étaient **égaux**. Sinon, ZF est **mis a 0**.

333

Processeur 80x86

Registre d'état (flags)

- **SF (Sign Flag : signe) :**
  - **SF est positionne à 1** si le **bit de poids fort** du résultat d'une **addition** ou **soustraction** est 1 ; sinon **SF = 0**. Il est utile pour manipuler des entiers signés, car **le bit de poids fort donne alors le signe du résultat**.
- **OF (Overflow Flag : Débordement) :**
  - Si on a un **débordement arithmétique**, ce bit est **positionné à 1**, c'est a dire le **résultat** d'une **opération excède la capacité de l'opérande** (registre ou case mémoire), sinon **il est a 0**.

334

Processeur 80x86

Registre d'état (flags):

- **IF (Interrupt Flag : Masque d'interruption) :**
  - Pour **masquer** les **interruptions** venant de **l'extérieur**, ce bit **est mis à 0**, dans le cas contraire (**IF = 1**) le microprocesseur **reconnait l'interruption** de **l'extérieur**.
- **TF (Trap Flag : Piège) :**
  - Indique au **microprocesseur l'exécution pas à pas**.

➔Les bits **DF, IF et TF** sont des **indicateurs de contrôle** qui permettent de **modifier le comportement du microprocesseur**. Ils sont **positionnés** par le **programmeur**.

335

Processeur 80x86

Segmentation de la mémoire par le 8086

- **L'espace mémoire adressable** par le **8086** est de **1 Mo = 2^{20} octets = 1 048 576 octets** (**20 bits** du bus d'adresse).
- Cet **espace** est **divisé** en **segments logiques** avec accès **direct** et **simultané**.
- Un **segment est une zone mémoire** de **64 Ko** (**65 536 octets**) définie par **son adresse de départ** qui doit être un multiple de 16 où les **4 bits de poids faible sont à zéro**.
- **Le compteur programme** (**IP**) est de **16 bits** donc la possibilité **d'adressage** est de **2^{16} = 64 Ko** (ce qui ne couvre pas la **totalité** de la mémoire),

336

Processeur 80x86

Segmentation de la mémoire par le 8086

- **L'adresse** d'un **segment** est représenté avec seulement ses **16 bits** de **poids fort**, les **4 bits de poids faible** étant **implicitement à 0**.
- Une **valeur sur 16 bits** peut designer une **case mémoire** parmi les **2^{16}=65536** contenues dans un **segment**.
- Utilisation de **deux registres** pour indiquer une **adresse** au **processeur**.
- Chaque **segment débute** à **l'endroit spécifié** par le **registre segment**. Le **déplacement** (**offset**) à **l'intérieur** de chaque **segment** se fait par un **registre de décalage** qui permet de trouver une **information** à **l'intérieur du segment**.

337

Processeur 80x86

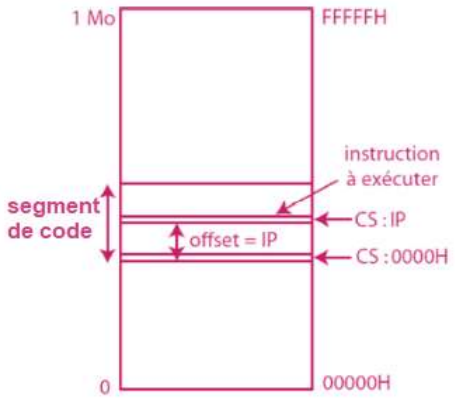
Segmentation de la mémoire par le 8086

- Une **case mémoire** est repérée par le 8086 au moyen de **deux quantités** (registres) sur 16 bits :
  - **l'adresse d'un segment ;**
  - **un déplacement ou offset (appelé aussi adresse effective) dans ce segment.**
- **Exemple** : la paire de registres **CS : IP** : pointe sur le **code d'une instruction** (**CS registre segment** et **IP déplacement**).

338

Processeur 80x86

Segmentation de la mémoire par le 8086



339

Processeur 80x86

Segmentation de la mémoire par le 8086

- Le **couple (segment, offset)** définit une **adresse logique**, notée sous la forme :

  segment : offset

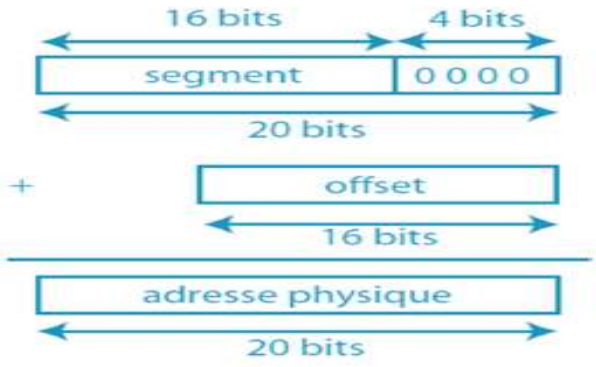
- **L'adresse** d'une **case mémoire** donnée sous la forme d'une quantité sur **20 bits** (5 digits hexa) est appelée **adresse physique** car elle correspond a la **valeur envoyée** réellement sur le **bus d'adresses A0 - A19**.

340

Processeur 80x86

Segmentation de la mémoire par le 8086

- **Correspondance** *entre* **adresse logique** *et* **adresse physique** :



341

Processeur 80x86

Segmentation de la mémoire par le 8086

- **L'adresse physique** se calcule par **l'expression** :

adresse physique = segment × 10(16) + offset

- Le fait **d'injecter 4 zéros** en **poids faible** du **segment** revient a effectuer un **décalage de 4 positions vers la gauche**, c'est-a-dire une **multiplication** par **2^4 = 16 (décimal) = 10 (H)**.

342

Processeur 80x86

Segmentation de la mémoire par le 8086

- Le **registre CS** est **associe** au **pointeur d'instruction IP**, la **prochaine instruction à exécuter** se trouve a **l'adresse logique CS:IP.**
- Les **registres de segments DS et ES** peuvent être **associés** à un **registre d'index DS:SI ; ES:DI**
- Le **registre de segment de pile SS** peut être **associe** aux **registres de pointeurs SS : SP ou SS : BP**

343

Processeur 80x86

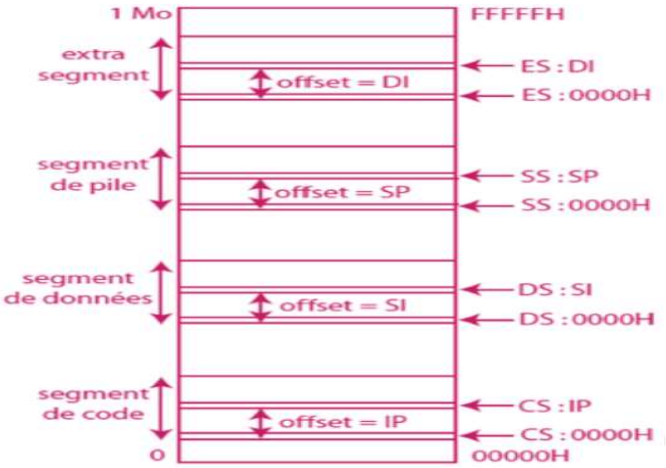
Mémoire accessible par le 8086 :

- Le **programmeur** en **assembleur** doit se **charger** de **l'initialisation de DS**, c'est-a-dire de lui **affecter** l'adresse du **segment** de **données** à utiliser.
- Le **registre CS** sera **automatiquement initialisé** sur le **segment** contenant la **première instruction du code** au moment du chargement en mémoire du **programme**.

344

Processeur 80x86

Mémoire accessible par le 8086 :

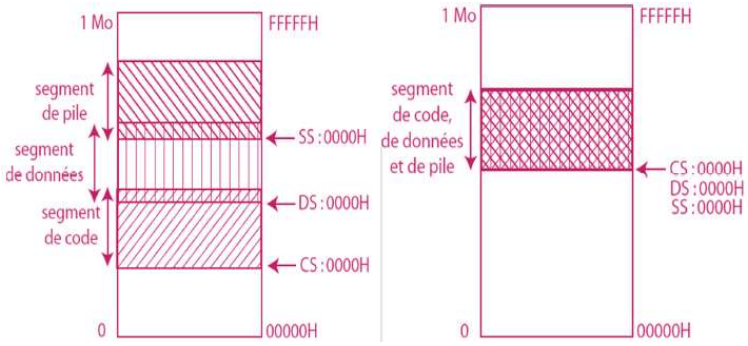


345

Processeur 80x86

Mémoire accessible par le 8086 :

- les **segments** ne sont pas nécessairement **distincts** les **uns des autres**, ils peuvent se **chevaucher** ou se **recouvrir complètement**.



346

Processeur 80x86

Mémoire accessible par le 8086 :

- Le **nombre** de **segments** utilisés définit le **modèle mémoire du programme.**
- Contenu des **registres** après un **RESET** du **microprocesseur** :
  - **IP = 0000H**
  - **CS = FFFFH**
  - **DS = 0000H**
  - **ES = 0000H**
  - **SS = 0000H**
- la **première instruction** exécutée par le 8086 se trouve donc a l'adresse **logique CS:IP = FFFFH** correspondant a l'adresse **physique FFFF0H**

347