UNIVERSITÉ MOHAMED PREMIER - OUJDA

Faculté Des Sciences

Département De Mathématiques Et Informatique

Cours du module

Architecture des Ordinateurs

SMI S4

Prof. TALIBI ALAOUI

Chapitre 1 :

Architecture générale

Petit historique des microprocesseurs Intel

Historique 1945

L'ENIAC, en 1945, comporte :

- ✓ 19000 tubes,
- ✓ pèse 30 tonnes,
- ✓ couvre 72 m^2
- ✓ et permet 330 multiplications par seconde.

La programmation se fait par des fiches à brancher sur un tableau de connections.

APRES ..

A cette belle époque de la construction du premier IBM-PC, les machines disposaient de 64 ko de mémoire vive et d'aucun disque dur !

Intel 8086 : Apparu en 1978, a marqué les débuts de l'architecture Intel moderne.

Avantage par rapport aux précédents : registres 16 bits et un bus de données sur 16 bits.

Intel 8088 : ne proposait que 8 bits pour les bus de données, ce qui le rendait moins coûteux.

La compatibilité descendante :

Il est remarquable de constater que chaque nouvelle génération de processeurs de la famille Intel reste compatible avec les générations antérieures.

Intel 80286 :

Apparu dans l'ordinateur IBM-PC/AT. C'est le premier processeur Intel à pouvoir fonctionner en mode protégé. Le 80286 pouvait adresser jusqu'à 16 Mo de mémoire vive grâce à un bus d'adresse sur 24 bits.

Intel 80386 :

Le successeur du 80286, offrait des registres sur 32 bits et un bus d'adresse sur 32 bits, ainsi qu'un chemin de données externe de la même largeur.

Intel 80486 : successeur du 80386 suivit des différentes générations de Pentium. Tout ces processeurs comprennent un nouveau système d'adressage de la mémoire virtuelle qui permet de gérer un espace plus grand que la mémoire physique.

Pentium :

La famille Pentium a apporté de nombreuses améliorations de performances:

Notamment une conception super-scalaire fondée sur deux pipelines parallèles rendant possibles le décodage et l'exécution de deux instructions en même temps.

Le Pentium offre également un bus d'adresses sur 32 bits avec un chemin de données interne sur 64 bits.

La sous-famille des processeurs P6 :

En 1995, est apparue la sous-famille des processeurs dite P6. Elle permettait une extension de l'architecture de base 32 bits.

Entrant dans cette famille le Pentium Pro, le Pentium II et le Pentium III.

Le Pentium Pro : introduisait des techniques évoluées pour améliorer les méthodes d'exécution des instructions.

Le Pentium II : ajoutait la technologie multimédia MMX.

Le Pentium III : ajoutait le mode SIMD (extension de flux d'exécution) avec des registres spéciaux sur 128 bits permettant de déplacer rapidement de grands blocs de données.

Pentium 4 :

Il a introduit la micro-architecture appelée NetBurst qui permet au processeur de fonctionner à des vitesses bien supérieures à celles des processeurs précédents.

Il est résolument orienté vers les applications multimédia performantes.

Architecture générale

d'un ordinateur

Un ordinateur est une machine capable :

- d'effectuer automatiquement des traitements sur une information fournie par un organe d'entrée suivant un programme.

- de délivrer une information sur un organe de sortie.

L’information est numérisée, c’est à dire représentée en binaire : une suite de 0 et 1

Les détails du codage binaire de l’information seront traités dans le chapitre suivant.

Matériellement, un ordinateur est composé des :

- Cartes : elles-mêmes construite à partir de composants électroniques.

- Périphériques : écran, clavier, disques, etc...

Machine visible par l'utilisateur :

L'utilisateur interagit avec l'ordinateur au travers des périphériques externes (dispositifs externes à l'unité centrale) :

- Clavier
- Écran
- Souris
- Imprimante
- Disquette, etc…

La machine invisible à l'utilisateur

Toute action de l'utilisateur se traduit par l'exécution d'une séquence d'opérations faisant intervenir un ensemble de composants invisible par l'utilisateur.

Cette boite noire réalise généralement les quatre fonctions suivantes :

- Traitement des données.
- Mémorisation des données.
- Transfert des données.
- Contrôle.

Machine de Von Neumann 1948

Pratiquement toutes les structures internes actuelles d'ordinateurs repose sur des concepts développés par *John Von Neumann* à l'institue "Advanced Studies of Princeton".

Cette réalisation appelé l'architecture de **Von Neumann**. En voici les caractéristiques :

- Une mémoire principale contient à la fois les données et les instructions :

- La mémoire est formée d'un ensemble des mots de longueur fixe.
- Chaque mot contenant une information codée en binaire.
- Chaque mot de la mémoire est accessible par l'intermédiaire de l'adresse mémoire.
- Le temps d'accès à un mot est le même quel que soit la place du mot en mémoire.

Architecture de Von Neumann

- Un processeur est capable de :
  - contrôler le fonctionnement d'un ordinateur.
  - d'exécuter les opérations fonctionnant sur des données binaires.
- Une unité d'entrée et de sortie (E/S) sert d'interface avec les périphériques.

Les trois composants sont reliés par des ***bus*** sur lesquels circulent les données, les adresses, et les données de contrôle.

Les instructions sont exécutées dans l'ordre ou en séquentiel par le processeur par défaut. Mais le programme peut en modifier l'ordre d'exécution par création d'instructions pour ruptures de séquences.

Cet architecture est toujours en vigueur de nos jours.

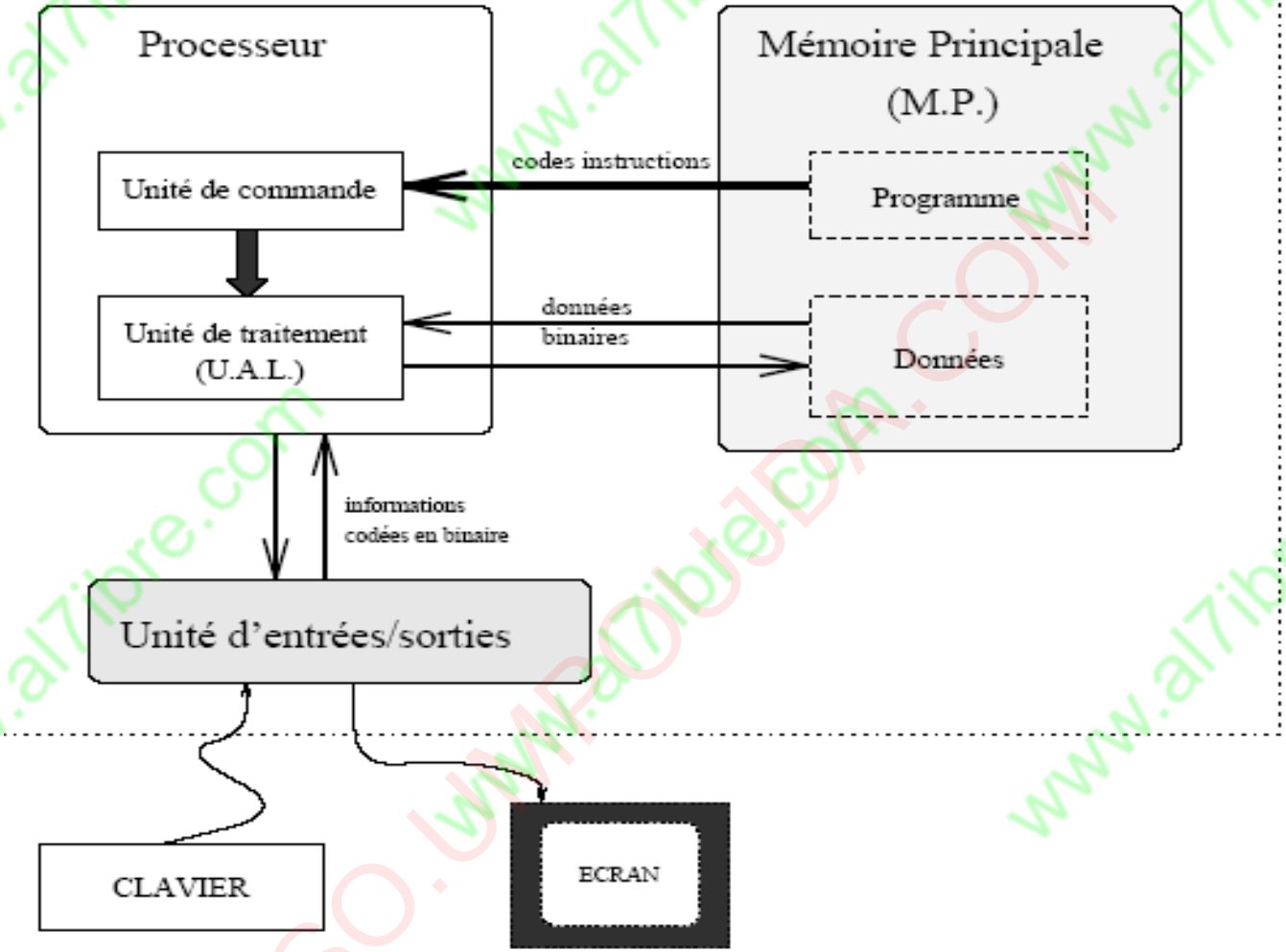


Fig 1. Architecture de Von Neumann

Fonctionnement méthodique du processeur

- extrait une instruction de la mémoire
- analyse l'instruction
- recherche dans la mémoire les données concernées
- chargement des données si nécessaire
- déclenche l'opération adéquate sur l'UAL
- range le résultat dans la mémoire
- passe à l'instruction suivante

Le processeur est relié au reste de la machine grâce à un ensemble de broches qui sortent du boitier du circuit intégré.

Une large série de broches est connectée à un ensemble de fils parallèles constituent le bus de données. Une deuxième série est reliée à un bus de contrôle et une autre encore à un bus d'adresses.

Principe de la conception d'un micro-ordinateur

Le schéma suivant montre les éléments constitutifs d'un micro-ordinateur.

L'unité centrale de traitement, appelée CPU (Central Processor Unit), est le circuit qui réalise tous les calculs et toutes les opérations logiques.

La CPU est dotée d'un espace mémoire limité constitué de *registres*.

Elle comprend également une horloge, une unité de contrôle et une unité de logique arithmétique (ALU).

Figure

Diagramme schématique d’un micro-ordinateur

Unité de stockage :

Mémorise les instructions et les données qui sont utilisées pendant l'exécution d'un programme.

Cette unité reçoit des demandes de données en provenance du processeur. Elle transfère les données depuis un emplacement en mémoire vive (RAM) vers le processeur et réalise l'opération inverse.

UAL + UC :

Bus :

Un bus est un ensemble de liaisons électriques considérées comme un faisceau parallèle. Un bus permet de transférer des données d'un endroit à l'autre de l'ordinateur.

➢ L'horloge :

Chaque opération où interviennent le processeur et le bus système doit être synchronisée par une horloge interne qui pulse à une fréquence fixe.

L'unité de temps élémentaire pour les instructions machine se nomme le cycle machine ou temps de cycle d'horloge.

Chaque instruction machine doit au moins durer un cycle d'horloge, mais la plupart en consomment plus, et certaines instructions complexes durent plus de cinquante cycles d'horloge.

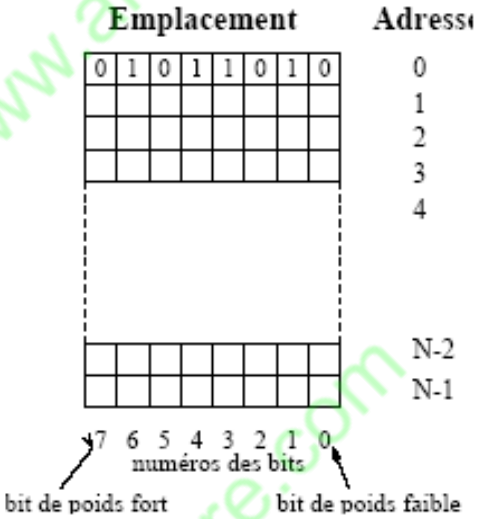
➢ L’horloge :

Dans les ordinateurs actuels, les horloges battent à une fréquence de l’ordre d’un milliard de fois par seconde (1 Ghz), ce qui équivaut à un cycle d’horloge d’une durée d’un milliardième de seconde (une nanoseconde).

Éléments principaux d'un ordinateur

Éléments principaux d'un ordinateur

- Mémoire
  - Stocke les programmes et les données
  - Enregistre les résultats intermédiaires et/ou finaux :
    - Dans une mémoire de taille N,
    - on a N emplacements mémoires, numérotés de 0 à N-1.
    - Chaque emplacement est repéré par son numéro, appelé *adresse.*



Éléments principaux d'un ordinateur

Chaque emplacement mémoire conserve les informations que le processeur y écrit jusqu'à coupure de l'alimentation électrique, où tout le contenu est perdu (contrairement au contenu des mémoires externes comme les disquettes et disques durs).

Éléments principaux d'un ordinateur

Les seules opérations possibles sur la mémoire sont :

- écriture d'un emplacement : le processeur donne une valeur et une adresse, et la mémoire range la valeur à l'emplacement indiqué par l'adresse.

- lecture d'un emplacement : le processeur demande à la mémoire la valeur contenue à l'emplacement dont il indique l'adresse. Le contenu de l'emplacement lu reste inchangé.

Éléments principaux d'un ordinateur

➢ Unité de transfert en mémoire

les opérations de lecture et d'écriture portent en général sur plusieurs octets contigus en mémoire : un *mot mémoire*.
La taille d'un mot mémoire dépend du type de processeur.
Elle est de :

- 1 octet (8 bits) dans les processeurs 8 bits (par exemple Motorola 6502).
- 2 octets dans les processeurs 16 bits (par exemple Intel 8086).
- 4 octets dans les processeurs 32 bits (par ex. Intel 80486 ou Motorola 68030).

Lecture depuis la mémoire

Les opérations d'accès à la mémoire constituent un critère important dans les performances d'exécution d'un programme.

L'horloge du processeur peut fonctionner par exemple à 1, 2 ou 3 Ghz, mais les accès à la mémoire qui passent par le bus système sont beaucoup plus lentes.

Oblige le processeur à attendre 1 ou plusieurs cycles d'horloge le temps que la valeur d'un opérande soit rapatriée depuis la mémoire afin que l'instruction puisse effectuer un traitement sur cette valeur.

Lecture depuis la mémoire

Ces cycles perdus sont appelés les cycles d'attente (Wait State). Il faut en effet compter plusieurs étapes pour lire une instruction ou une donnée depuis la mémoire.

Lecture depuis la mémoire

Description simplifiée de ce qui se produit pendant chaque cycle d'horloge pour lire une valeur en mémoire :

Cycle 1 : les bits de l'adresse à laquelle est stocké l'opérande mémoire sont placés sur bus d'adresses (ADDR).

Cycle 2 : la ligne de lecture Read Line (RD) est forcée à la valeur 0 pour prévenir les circuits mémoire qu'une lecture est demandée.

Cycle 3 : Le processeur attend un cycle pour laisser aux circuits mémoire le temps de réagir. Pendant ce cycle, le contrôleur mémoire place l'opérande sur le bus de données (DATA).

Lecture depuis la mémoire

Cycle 4 : La ligne de lecture Read Line est forcée à nouveau à 1, ce qui permet au processeur de savoir qu'il peut lire la valeur qui a été mise à disposition sur le bus de données.

Lecture depuis la mémoire

Mémoire cache : A cause de la relative lenteur de la mémoire par rapport au processeur, les micro-ordinateurs embarquent une petite zone de mémoire dans la puce du processeur pour y stocker les plus récentes instructions et données utilisées. C'est la *mémoire cache*.

Lecture depuis la mémoire

Fonctionnement d'un programme : Processus de chargement et d'exécution

Ce processus est réalisé par le système d'exploitation

Lecture depuis la mémoire

Les différentes étapes du chargement et du lancement d'exécution d'un processus est réalisé par le SE :

1. L'utilisateur saisit le nom d'une commande correspondant à un programme (sous Dos ou Linux).

2. Le système d'exploitation recherche dans le répertoire en cours un fichier portant le nom indiqué.

3. Une fois le fichier trouvé, le système récupère des informations générales comme sa taille et sa position réelle sur le disque dur.

Le SE détermine alors quel est le prochain bloc d'espace mémoire libre, puis il charge le contenu du fichier à cet endroit.

Lecture depuis la mémoire

Il réserve un bloc mémoire au programme, puis génère des informations décrivant la taille et la position du programme dans une table spécifique, appelée table des descripteurs.

Le système réalise également des ajustements de valeurs pour les pointeurs du programme afin que ceux-çi contiennent les adresses correctes des données associées au programme.

4. Le SE exécute alors une instruction de branchement qui force le processeur à exécuter la première instruction machine trouvée dans le fichier du programme.

Dés que le programme commence son exécution, il prend le statut de processus. Le système lui attribue un numéro d'identification, appelé ID de processus, qui permet de gérer le processus pendant son exécution.

Lecture depuis la mémoire

5. Le processus continue alors de s'exécuter. Le SE se tient prêt à répondre aux demandes d'utilisation de ressources de système. Ces ressources sont la mémoire, les fichiers et les périphériques d'entrée et de sortie.

6. Une fois que le processeur est terminé, son identificateur (appelé handle) est supprimé, et la zone mémoire qui était occupée est libérée pour être utilisée par un autre programme.

Lecture depuis la mémoire

Le mode multitâches :

Un SE qui est capable de lancer l'exécution de plusieurs programmes à la fois est dit multitâches.
La plupart des SE actuels gèrent simultanément plusieurs taches qui doivent pouvoir communiquer avec le matériel, l'interface utilisateur, le système de fichiers, etc.

OR, en réalité, le processeur ne peut exécuter qu'une instruction à la fois. C'est un programme particulier du système d'exécution, appelée le planificateur (scheduler) qui réalise l'arbitrage du temps de traitement CPU.

Lecture depuis la mémoire

Il distribue à chaque tache une tranche de temps (time slice).

Pendant chaque tranche, le processus exécute une série d'instructions appartenant à une tache puis passe à la tache suivante.

La commutation rapide d'une tache à l'autre permet au SE de donner l'illusion de faire progresser l'exécution de plusieurs taches en parallèle.
Un type trés répandu de planification des taches s'appelle la planification Round Robin (tracer figure).

Lecture depuis la mémoire

Un SE multitâches ne peut fonctionner que sur un processeur qui sait réaliser la commutation de taches.

En effet, le processeur doit mémoriser l'état de chaque tache avant de passer à la suivante.

En général, le SE attribue des priorités différentes aux taches, ce qui leur donne des tranches de temps plus ou moins importantes.

Autres éléments principaux d'un ordinateur

- Processeur central (aussi appelé CPU) contient :

  - Unité Arithmétique et Logique (UAL).
  - Unité de commande
  - Registres
  - Mémoire cache :

  Mémoire intermédiaire pour optimiser les performances (cherche les instructions d'un programme en mémoire, les interprète et les exécute).

UAL : On peut résumer les opérations effectuées par l'UAL

- Les opérations arithmétiques comme l'addition, la soustraction, la multiplication, la division, les opérations en point flottant.

- Les opérations d'ordre logique comme la conjonction logique OU, la disjonction ET, la négation NON, les opérations de comparaison (< , > , = , …, etc).

Unité de commande ou unité de contrôle :

est "la patronne" de l'ordinateur et se charge :

- d'appeler les instructions,
- de les décoder,
- de rechercher les opérandes et,
- s'assure que les différentes phases de l'instruction sont exécutées au bon moment par l'unité concernée.

Éléments principaux d'un ordinateur

- Unités d'entrées et de sorties, pour communication avec :
  - En entrée : clavier, souris, disque dur, ...
  - En sortie : écran, disque dur, ...

Bus :

- Les systèmes/éléments sont reliés par bus :
  - Un Bus permet de transmettre en parallèle plusieurs données entre les différentes unités de l'ordinateur.
  - Ensemble de n fils conducteurs : 1 fil = 1 bit
- Largeur du Bus (nombre de fils constituant le chemin):
  - Nombre de bits pouvant être envoyés en même temps.
  - Limite le débit des données, E/S…

- **Bus de données :** sert à transférer des instructions et des données entre processeur et mémoire. Sa largeur est par exemple 64 bits actuels, soit 64 broches de connexion.
- **Bus de contrôle :** véhicule des signaux binaires pour synchroniser les actions des différents équipements qui sont reliés au processeur.

- Bus d'adresses : véhicule les adresses des instructions et des données utilisées par l'instruction en cours d'exécution lors des transferts de données entre processeur et mémoire.

- Dans un PC, plusieurs bus plus ou moins rapides

- bus rapides
  - **Bus adresse (FSB ou Front Side Bus)**
    - **Bus de communication avec le CPU**
  - **Bus données : communication avec la mémoire**
  - **Bus AGP (ou PCI-X) : communication avec la carte graphique**
- bus plus lents
  - **PCI : cartes réseaux, son …**
  - **Connexion périphérique de stockage (DD, CD, DVD…)**
    - **ATA, SATA, SCSI …**
  - **Connexion de périphériques extérieurs**
    - **USB, FireWire …**

Modes de fonctionnement des processeurs

➢ Modes de fonctionnement

Les processeurs offrent 3 principaux modes de fonctionnement :

Mode protégé **:** est le mode normal des processeurs.

Toutes les instructions et toutes les fonctions sont disponibles dans ce mode. A chaque programme est attribuée une zone mémoire distincte (un segment).

Le processeur est capable de détecter toute tentative d'un programme pour accéder à une zone mémoire en dehors de son segment.

Mode virtuel 8086 :

Lorsqu'il est en mode protégé, le processeur peut exécuter directement des logiciels en mode réel (comme anciens programmes MS DOS) tout en restant dans un environnement multitâches. Cette fonction est souvent appelée mode virtuel 8086.

Mode réel :

Le mode réel propose l'environnement d'exécution du processeur ancestral 8086 + possibilité de Basculer vers l'un des deux autres modes.

Tous les processeurs Intel démarrent en mode Réel. Après le démarrage, c'est le système d'exploitation qui réalise la commutation vers le mode protégé.

Mode de gestion système SSM:

Ce mode offre au système d'exploitation un mécanisme pour piloter des fonctions telles que la gestion de l'énergie et la sécurité système.

Ces fonctions sont normalement destinées aux constructeurs des ordinateurs qui ont besoin de personnaliser le processeur pour une configuration système particulière.

Environnement d'exécution de base :

Espace d'adressage :

En mode protégé, un processeur peut disposer jusqu'à 4 Go d'espace mémoire. Cette valeur découle de la largeur des registres (32 bits) qui peuvent donc mémoriser une valeur entre 0 et 2^{32} -1.

En mode réel, la mémoire accessible se limite à 1 Mo.

Environnement d'exécution de base :

Registres d'exécution fondamentaux :

Un registre est un lieu de stockage à très haute vitesse implanté directement sur le circuit du processeur. Par exemple, pour optimiser l'exécution d'une boucle de traitement, il est parfois possible de forcer le stockage des variables dans les registres, et non en mémoire.

En général, nous disposons de :

- 8 registres à usages général :

  EAX, EBX, ECX, EDX, EBP, ESP, ESI et EDI

Environnement d'exécution de base :

- De 6 registres de segment :

CS, SS, DS, ES, FS et GS

- D'un registre de drapeaux (statut flags) : EFLAGS

- Un registre pour le pointeur d'instruction : EIP

Environnement d'exécution de base :

Usages spécifiques :

Certains des registres à usage général satisfont à des besoins spécifiques :

- EAX est utilisé automatiquement par les instructions de multiplication et de division. C'est pourquoi il est souvent appelé le registre *accumulateur étendu*.

- Le processeur utilise automatiquement le registre ECX comme compteur de boucle.

- Le registre ESP sert à gérer les données de la pile (une structure mémoire du système) Ce registre ne doit jamais être utilisé pour les calculs et les transferts de données ordinaires. On l'appelle aussi le registre *pointeur de pile étendu*.

Environnement d'exécution de base :

Usages spécifiques :

- Les registres ESI et EDI servent lors des instructions de transfert mémoire à haute vitesse. Souvent appelés des registres *d'index source étendu* et d'index de destination étendu.

- Le registre EBP est utilisé par les langages de haut niveau pour faire référence à des paramètres de fonctions et à des variables locales sur la pile (lors des appels de fonctions). Il ne faut pas l'utiliser pour les calculs et les transferts de données ordinaires (réservés aux experts). Souvent appelé le registre *pointeur de cadre étendu*

Environnement d'exécution de base :

Registres de Segments :

Un registre de segment contient l'adresse de base d'une zone mémoire délimitée appelée *segment*. Un segment peut contenir soit :

- des instructions du programme (segment de code).
- Des variables (segment de données)
- Les variables des fonctions locales, les paramètres de ces fonctions et l'adresse retour du sous-programme (segment de pile).

Environnement d'exécution de base :

Pointeur d'instruction EIP :

Le registre EIP est le pointeur d'instruction. il contient l'adresse de la prochaine instruction à exécuter. Certaines instructions machine modifient le contenu de ce registre, ce qui permet au programme d'exécuter une instruction autre que celle qui est normalement la suivante dans l'ordre linéaire.

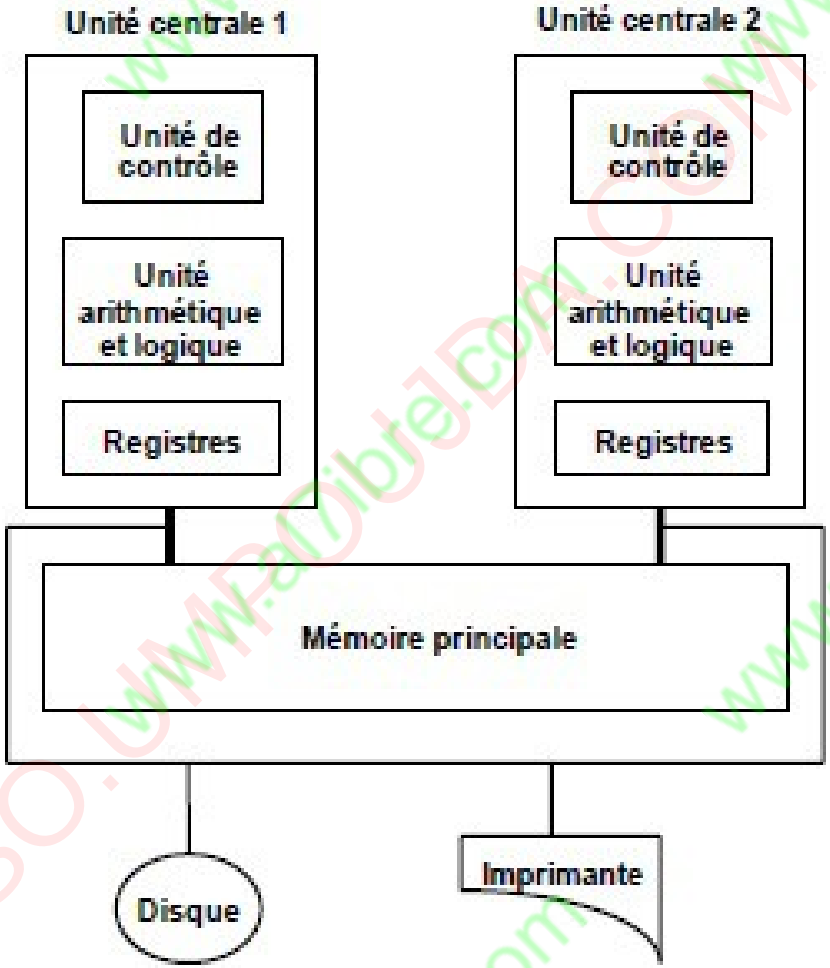
EFLAGS :

Le registre EFLAGS est constitué de bits binaires individuels. Certains d'entre eux servent à contrôler le mode de fonctionnement du processeur, mais la plupart sont seulement destinés à être lu et permettent de connaitre le statut de la dernière opération réalisée.

Différentes architectures des processeurs

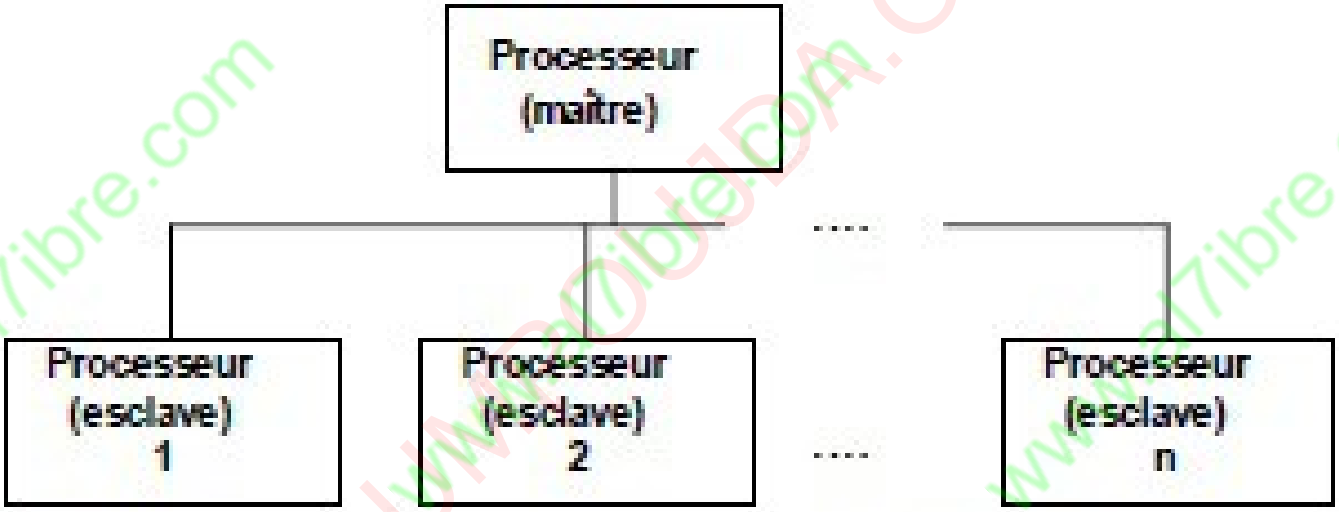
M. TALEBI ALAOUI

Organisation d'un ordinateur à unités centrales multiples



Processeurs multiples maîtres et esclaves

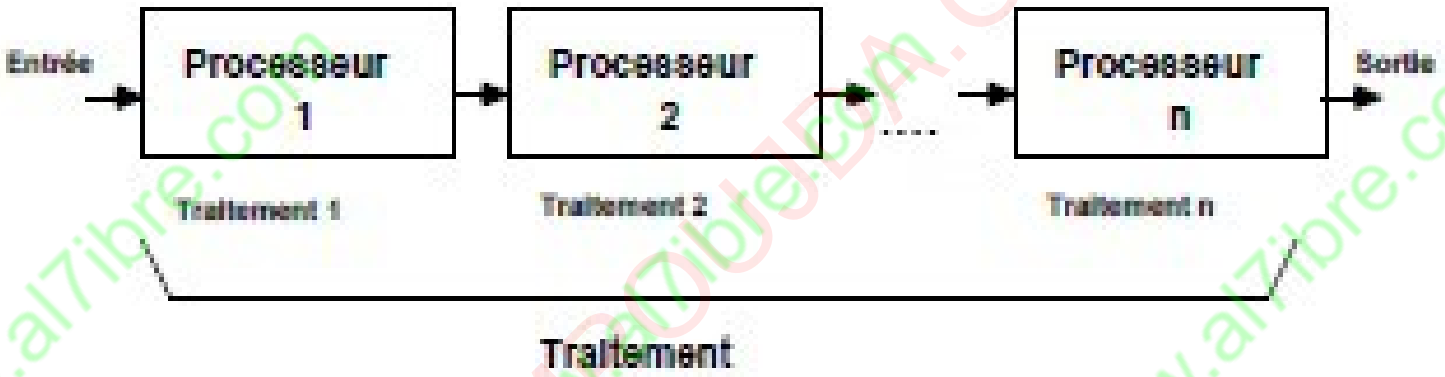
Processeurs multiples maître/esclaves (modèle centralisé)



Le VAX 11/782, par exemple, est constitué de deux VAX 11/780 fonctionnant en mode maître/esclave.

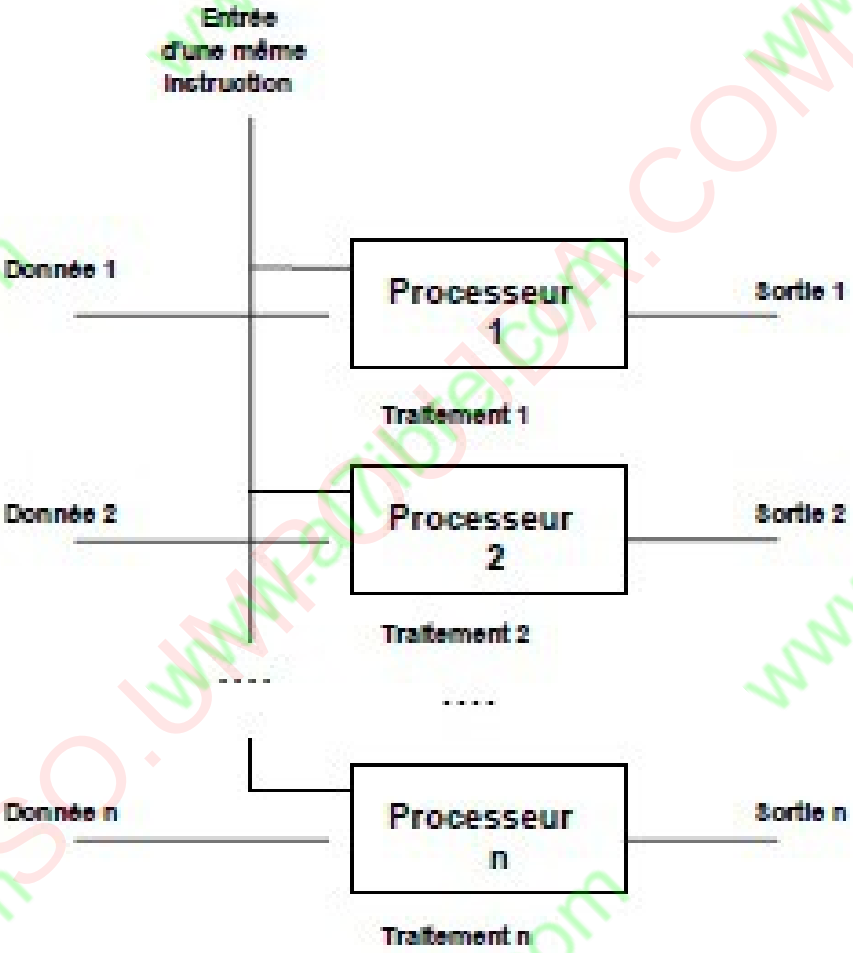
Architecture pipe-line

Architecture pipe-line de multiple s processeurs

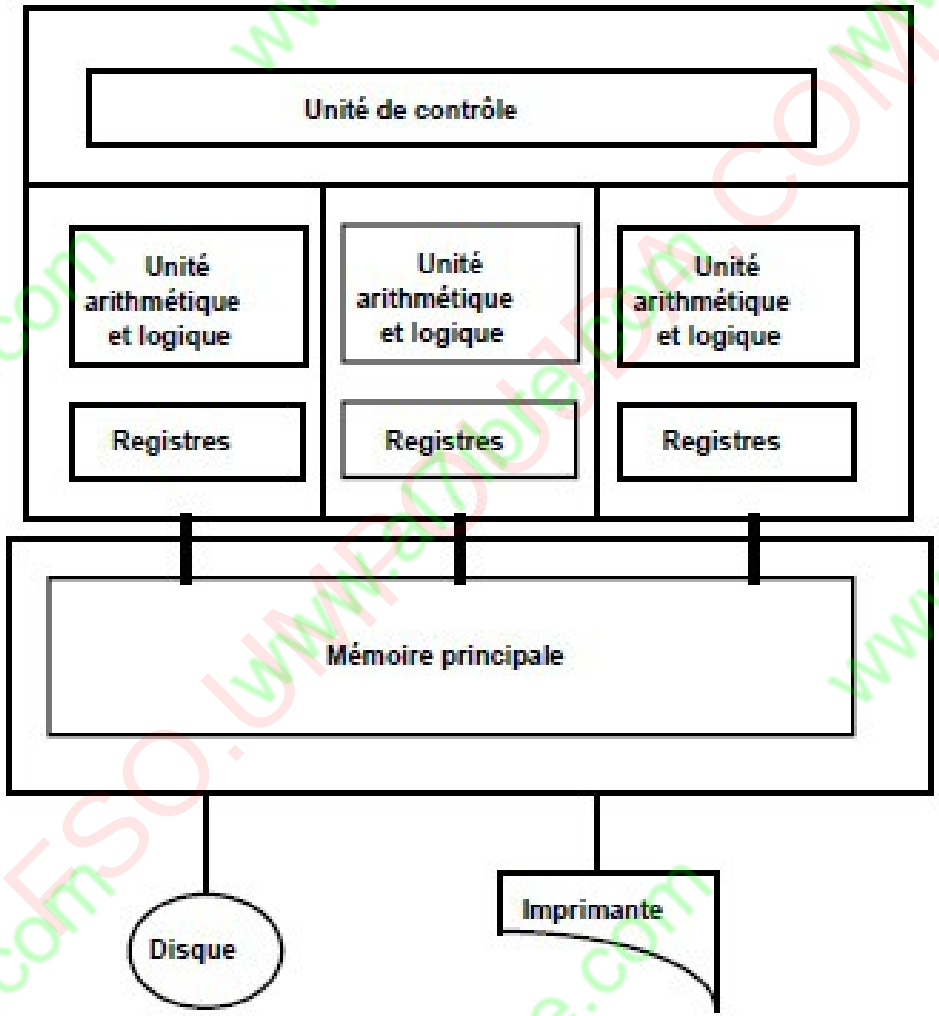


Traitement en parallèle

Traitement en parallèle par multiples CPU



Organisation d'un ordinateur en parallèle



La carte mère

Un micro-ordinateur est d'abord une carte mère.

C'est un grand circuit imprimé sur lequel sont soudés de nombreux circuits intégrés :

- Le processeur
- Les coprocesseurs
- Les circuits de mémoire principale
- Les connecteurs d'entrées / sorties
- Les connecteurs d'alimentations électriques
- Les connecteurs d'extension

Au niveau électrique :

Les différents composants sont connectés les uns aux autres au moyen d'un ou plusieurs bus.

Un bus consistant en un faisceau de plusieurs liaisons électriques parallèles directement gravées sur la carte mère.

Les cartes mères se distinguent sur le marché par leurs performances et leurs possibilités d'extension.

Les cartes mères ont tous un certain nombre d'éléments en commun :

- Un support de processeur (CPU Soket) : le format du boitier du processeur a évolué au cours du temps. Le support destiné à le recevoir est donc d'un format variable selon la génération de processeurs.

- Un connecteur de mémoire cache externe : destiné à des circuits mémoire cache à haute vitesse qui épargnent dans certains cas au processeur d'accéder à la mémoire vive, plus lente.

- Des connecteurs d'extension de mémoire principale : Ce sont des circuits mémoires qui sont déjà soudés sur de tout petits circuits imprimés qui sont insérés dans les connecteurs.

- Un circuit de mémoire morte ROM pour le BIOS (Basic Imput Output System). Cette mémoire figée contient le logiciel qui est exécutée dés le démarrage de l'ordinateur. De nombreux circuits BIOS actuels peuvent être mis à jour, car leur technologie permet de modifier le contenu de la mémoire (EEPROM, mémoire Flash).

- Des connecteurs pour les périphériques IDE :
  Ils permettent de connecter des disques durs, des lecteurs de disquettes et des lecteurs ou graveurs de CD-ROM.

- Des ports parallèle, série, USB, vidéo, clavier, joystick et souris.

- Des connecteurs d'extension PCI pour accueillir une carte son, une carte graphique, une carte d'acquisition de données et d'autres périphériques d'entrées/sorties.

Voici les principaux coprocesseurs d'un système IA-32 :

- L'unité à virgule flottante FPU : Elle gère les calculs à virgule flottante et les calculs entiers étendus. Dans les Pentium, elle est intégrée au processeur principal.

- Le circuit générateur d'horloge de types 8284 ou 82C284 : On l'appelle généralement horloge. Il génère un signal à fréquence constante. Le générateur d'horloge sert à synchroniser le processeur avec le reste de l'ordinateur.

- Un contrôleur d'interruptions programmable 8259 (PIC) :

Il prend en compte les demandes d'interruption en provenance des périphériques matériels tels que le clavier, l'horloge système et les disques durs.

Ces interruptions provoquent une pause dans la tache que le processeur était en train d'exécuter, de sorte qu'il exécute un petit programme correspondant à la demande du périphérique.

- Un Timer/compteur d'intervalles programmables 8254 :

Il provoque dans un PC une interruption du système 18,2 fois par seconde, afin de mettre à jour la date et l'heure du système et de prendre le contrôle du haut parleur.

C’est aussi ce circuit qui assure la rafraichissement de la mémoire.

Il faut savoir que les circuits mémoire principaux sont dynamiques et qu’ils ne peuvent conserver leur contenu que pendant quelques millisecondes. Il faut réécrire leur contenu périodiquement.

Jeu des circuits de la carte mère :

La plupart des cartes mères actuelles sont dotées d'une série de circuits de traitement et de contrôle que l'on appelle souvent un chipset.

Chipset : l'élément chargé d'aiguiller les informations entre les différents bus de l'ordinateur afin de permettre à tous les éléments constituant l'ordinateur de communiquer entre eux.

Il est composé de 2 éléments :

Pont nord (NorthBridge) : appelé également *contrôleur mémoire* est chargé de contrôler les échanges sur les bus rapides.

Pont sud (SouthBridge) : appelé également *contrôleur d'entrée sortie* ou *contrôleur d'extension,* gère les communications avec les périphériques d'entrée-sortie sur les bus lents.

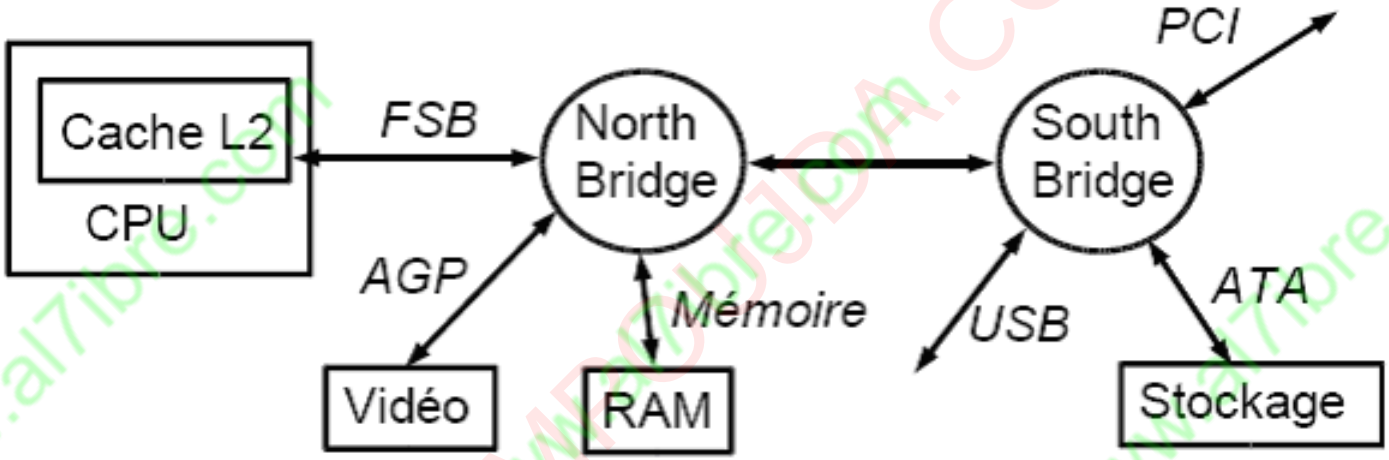
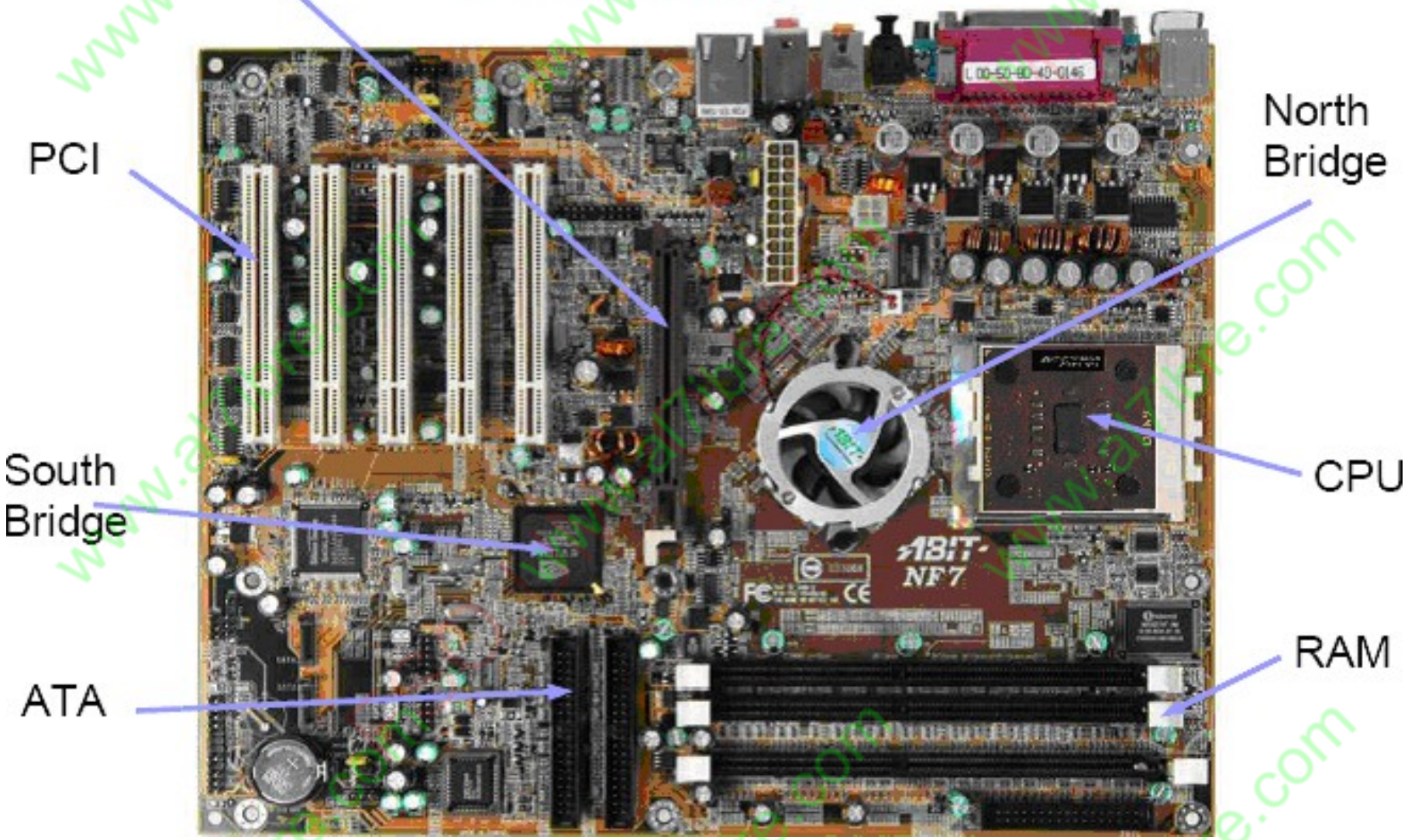


Fig : chipset et bus

Carte mère PC



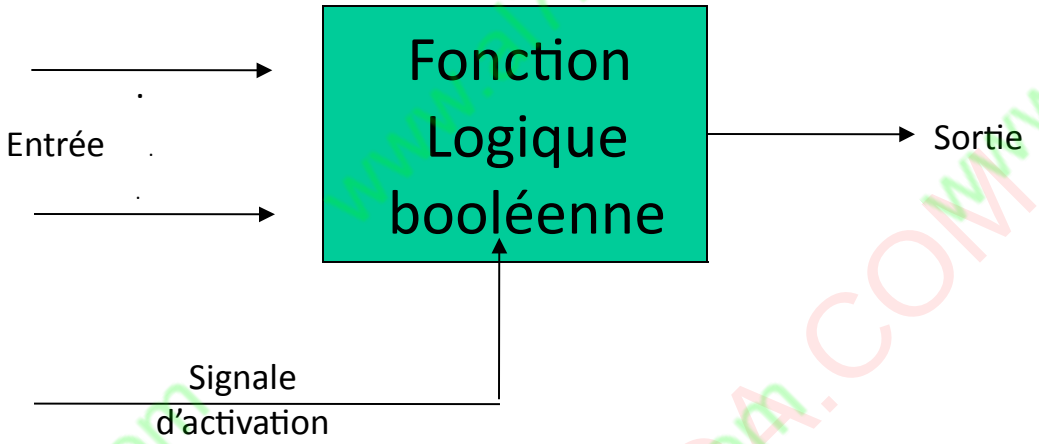
Composants de base d'un ordinateur :

- Les éléments de base d'un ordinateur doivent être capables de réaliser les fonctions de mémorisation, traitements, transfert et contrôle.

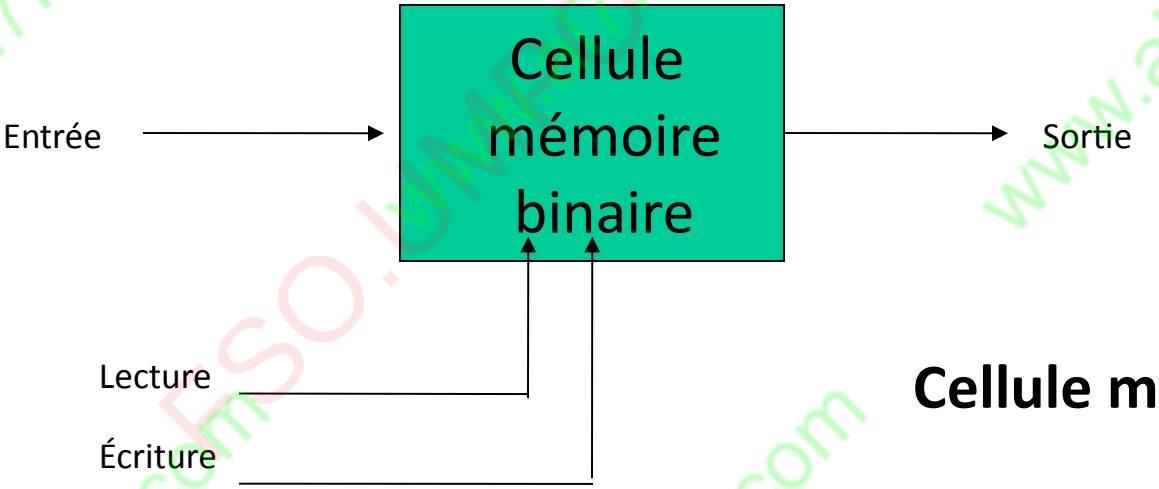
Deux types de composants sont nécessaires pour cela :

Porte : un composant qui met en œuvre une fonction booléenne simple (si a et b sont vrai alors c est vrai).

Cellule mémoire : un dispositif qui peut mémoriser un bit de données. En interconnectant de grandes quantités de ces composants, on peut construire une **mémoire**.



Porte



Cellule mémoire

Composants de base d'un ordinateur

- Lien entre les composants de base et les quatre fonctions de base :
  - **Mémorisation des données :** prise en charge par les cellules de mémoire.
  - **Traitement des données :** pris en charge par les portes.
  - **Transfert des données :** les chemins entre les composants sont empruntés pour transférer des données d'une mémoire vers une autre via des portes.

➢ Contrôle :

Une porte recevra les données en entré plus un signal de contrôle qui va l'activer.

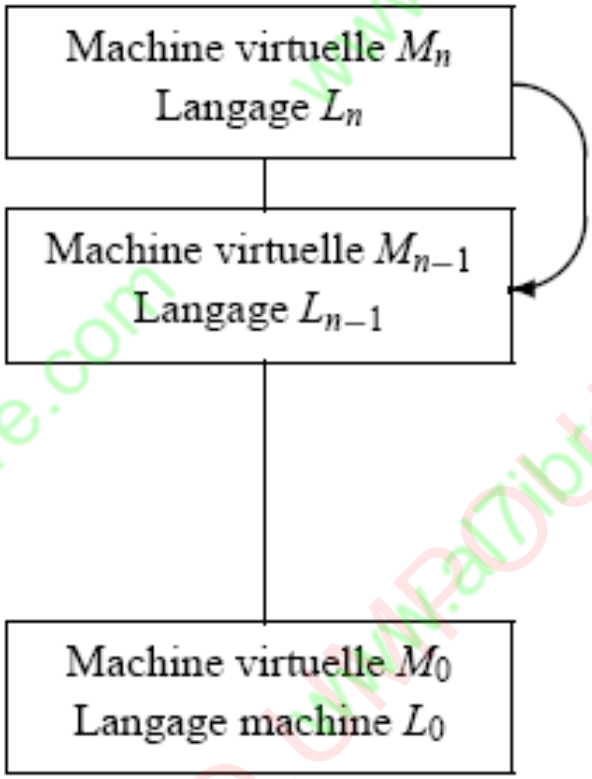
La cellule mémoire mémorise le bit qui se trouve sur son port d'entrée lorsque le signal d'écriture est activé et place le bit de la cellule sur son port de sortie lorsque le signal de lecture est activé.

Architecture en couche :

- Un programme est une suite d'instruction.
- Chaque instruction est une suite de bits contenant le code de l'opération (addition, soustraction, recherche d'un mot en mémoire, etc.) et ses opérandes (arguments de l'opération).

Pour éviter d'avoir à connaître ce langage difficilement manipulable par l'homme, on définit des langages de plus haut niveau d'où une architecture en couches de machines virtuelles (voir Figure).

Architecture en couche :



Le passage d'une machine (ou langage) à une autre se fait par interprétation ou compilation

Fig. Architecture d'une machine multiniveaux

Architecture en couche :

➢ Interprétation :

Chaque instruction du langage L_n est traduite en la séquence adéquate du langage L_{n-1} exécutable, donc, par la machine virtuelle M_{n-1}.

À chaque exécution, il y a ré-interprétation.

- **Compilation** : Un programme de la machine M_n, en entier est traduit une fois pour toute en un programme en langage L_{n-1} exécutable directement par la machine virtuelle M_{n-1}.

Les machines actuelles présentes généralement l'architecture en couche de la figure suivante.

Niveau 5	**Langages d'application**		Pascal, C, Cobol, Basic, Prolog, Caml, Lisp, Java
	↓ ↓	interprétation ou compilation	
Niveau 4	**Langage d'assemblage**		Instructions assembleur, notion de variables, de piles d'exécution, de routines
	↓	traduction assembleur	
Niveau 3	**Système d'exploitation**		Routines programmées
	↓	interprétation	
Niveau 2	**Couche machine**		Langage machine codé en binaire
	↓ ↓	interprétation (microprogramme)	
Niveau 1	**Couche microprogrammée**		Langage interne du microprocesseur
	↓ ↓	exécution directe par le matériel	
Niveau 0	**Couche physique**		

Figure : Couches de la plupart des machines actuelles

Chapitre 2 :

Architecture d'un microprocesseur

Structure d'un processeur

Taches réalisées :

- Lecture d'instruction : Le processeur lit une instruction dans la mémoire.
- Interprétation d'instruction : l'instruction est décodée pour déterminer l'action demandée.

- Lecture des données : lire des données dans la mémoire ou depuis un module d'E/S.
- Traitements des données : exécution d'une opération arithmétique ou logique sur les données.
- Écriture des données : écrire des données dans la mémoire ou dans un module d'E/S.

Structure d'un processeur :

Un processeur se compose de :

- Une unité de contrôle pour contrôler le fonctionnement du processeur.
- Une unité arithmétique et logique pour traiter les données.

- Une mémoire interne pour stocker temporairement les données et les instructions appelée ***registre***.

- Un bus interne pour interconnecter ces différents composants.

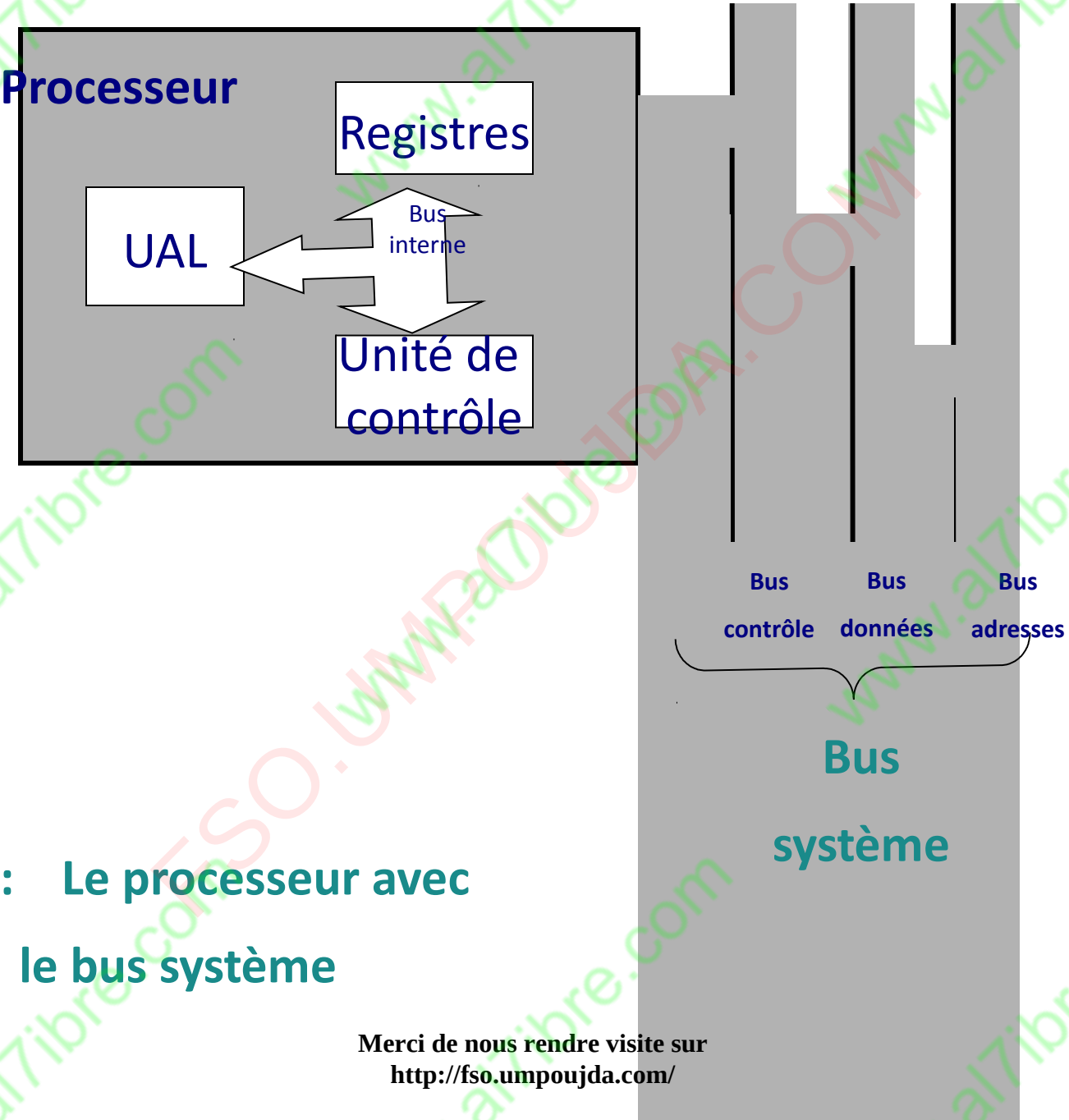


Fig. 1 : Le processeur avec le bus système

Registres :

Dans un processeur on trouve deux types de registres :

- Registres visibles par l'utilisateur : ils permettent au programmeur en langage machine de réduire les références de la mémoire principale en optimisant l'utilisation des registres.
- Registres invisibles de contrôle et d'état :
  - utilisés par l'unité de contrôle pour contrôler le fonctionnement du CPU
  - et par des programmes privilégiés du système d'exploitation pour contrôler l'exécution des programmes.

Registre visible par l'utilisateur :

On peut le classer en plusieurs catégories :

- de données : contiennent uniquement des données et ne peuvent pas être employés pour le calcul d'adresse.
- d'Adresses : consacrés à un mode d'adressage particulier (adresse de base de segment, index, pointeur de pile).

- Codes condition (également appelées drapeaux ou flags) :

sont des bits positionnés par le matériel du CPU qui

résultent d'opérations, et sont en lecture seule.

- Généraux : pas de restriction du contenu.

➢ Registre de contrôle :

En général ces registres ne sont pas visibles par l'utilisateur.

Ils servent à contrôler le fonctionnement du processeur.

Quatre registres sont essentiels à l'exécution d'une instruction. Ils sont utilisés pour l'échange avec la mémoire principale.

- Le compteur ordinal : (PC pour program counter). Contient l'adresse de la prochaine instruction à exécuter.
- Le registre d'instruction (RI) : contient la dernière instruction lue.

- Le registre d’adresse mémoire (MAR) :

Contient l'adresse d'un emplacement mémoire.

Est directement relié au bus d’adresse.

- Le registre tampon mémoire (TR) :

contient un mot de données à écrire en mémoire ou le dernier mot lu.

Il est directement relié au bus de données et fait le lien avec les registres visibles par l’utilisateur.

Les quatre registres que nous venons de mentionner servent à déplacer les données entre le CPU et la mémoire.

Le registre PSW (program status word)

- Contient des informations d'état. Voici les champs et drapeaux les plus courants :

  - Signe : contient le bit de signe du résultat de la dernière opération arithmétique.
  - Zéro : met à 1 quand le résultat est 0.
  - Retenue : met à 1 lorsqu'une opération génère une retenue.

- Égal : mis à 1 quand une comparaison logique entraîne une égalité.
- Débordement : à 1 lorsqu'une opération provoque un débordement.
- Interruption activée/désactivée : utilisé pour activer ou désactiver des interruptions.
- Superviseur : indique si le CPU s'exécute en mode superviseur ou utilisateur. Certaines instructions privilégiées ne s'exécutent qu'en mode superviseur.

Cycle d'instruction : Cycle normal

- Recherche de l'instruction (fetch) : l'instruction est lue à partir de la mémoire.

- Interprétation de l'instruction (décode) : l'instruction est décodée pour déterminer à quelle action elle correspond.

Exécution (execute) :

- Recherche des données : l'exécution d'une instruction peut demander la lecture des données dans la mémoire ou depuis un module d'E/S.

- Traitement des données : l'exécution d'une instruction peut demander des opérations arithmétiques ou logiques sur les données.

- Écriture de données : l'exécution d'une instruction peut demander l'écriture des résultats dans la mémoire ou sur un module d'E/S.

Cycle d'instruction : Cycle normal

- Toutes les machines ont un schéma d'exécution similaire, qui consiste donc à exécuter la boucle suivante :
  - Répéter
    - Recherche de l'instruction
    - Interprétation de l'instruction
    - Exécution

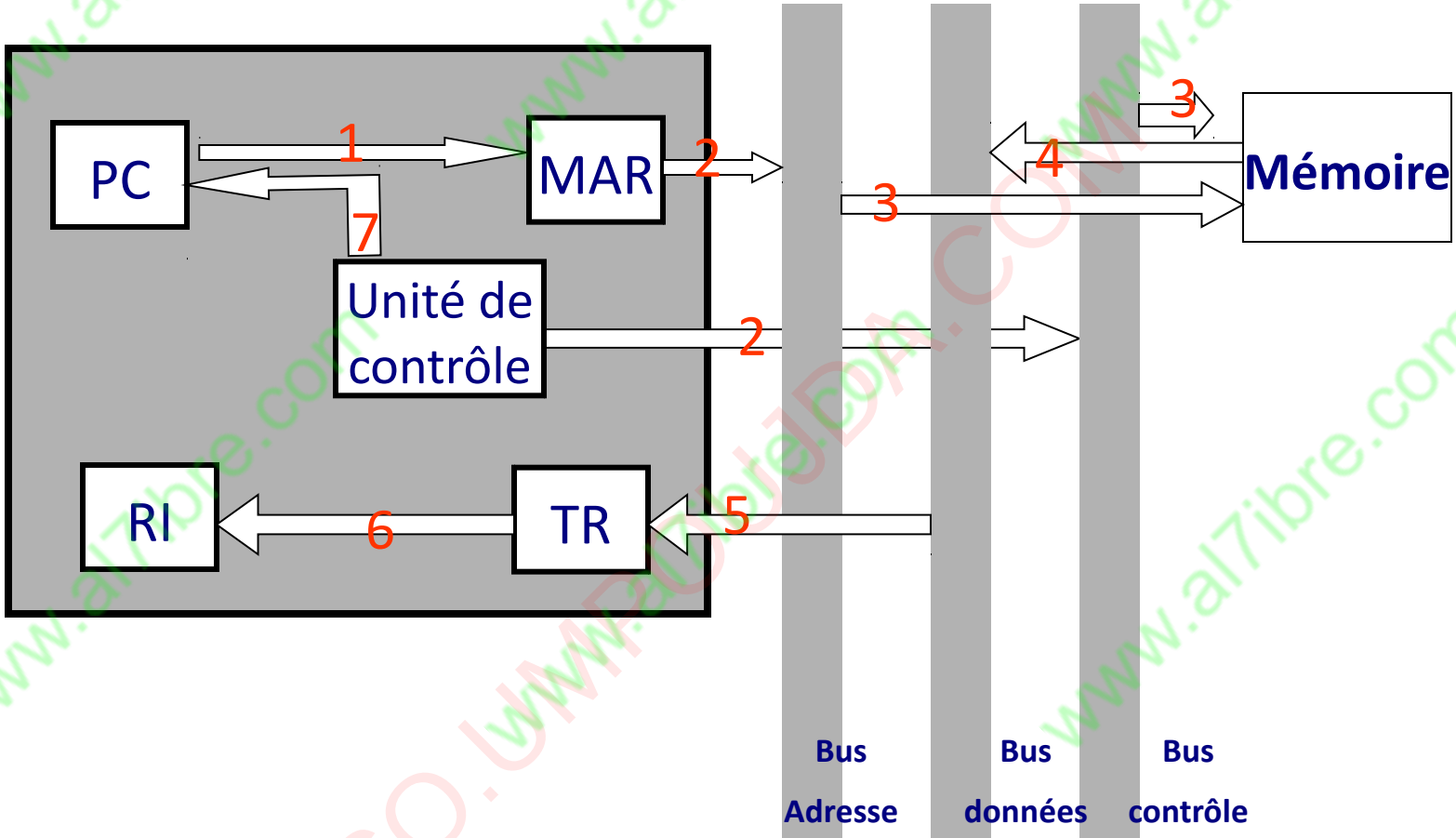


Fig 2 : Cycle de recherche d'une instruction

Cycle d'instruction normale

- Recherche de l'instruction
  - MAR ⟵ PC
  - L'unité de contrôle demande une lecture de la mémoire principale.
  - Le résultat de la lecture est placé sur le bus de données et copié dans TR.

- RI ⟵ TR

- PC ⟵ PC+taille de l'instruction avec

Taille d'instruction = nombre d'unités adressables nécessaires au stockage de l'instruction suivante.

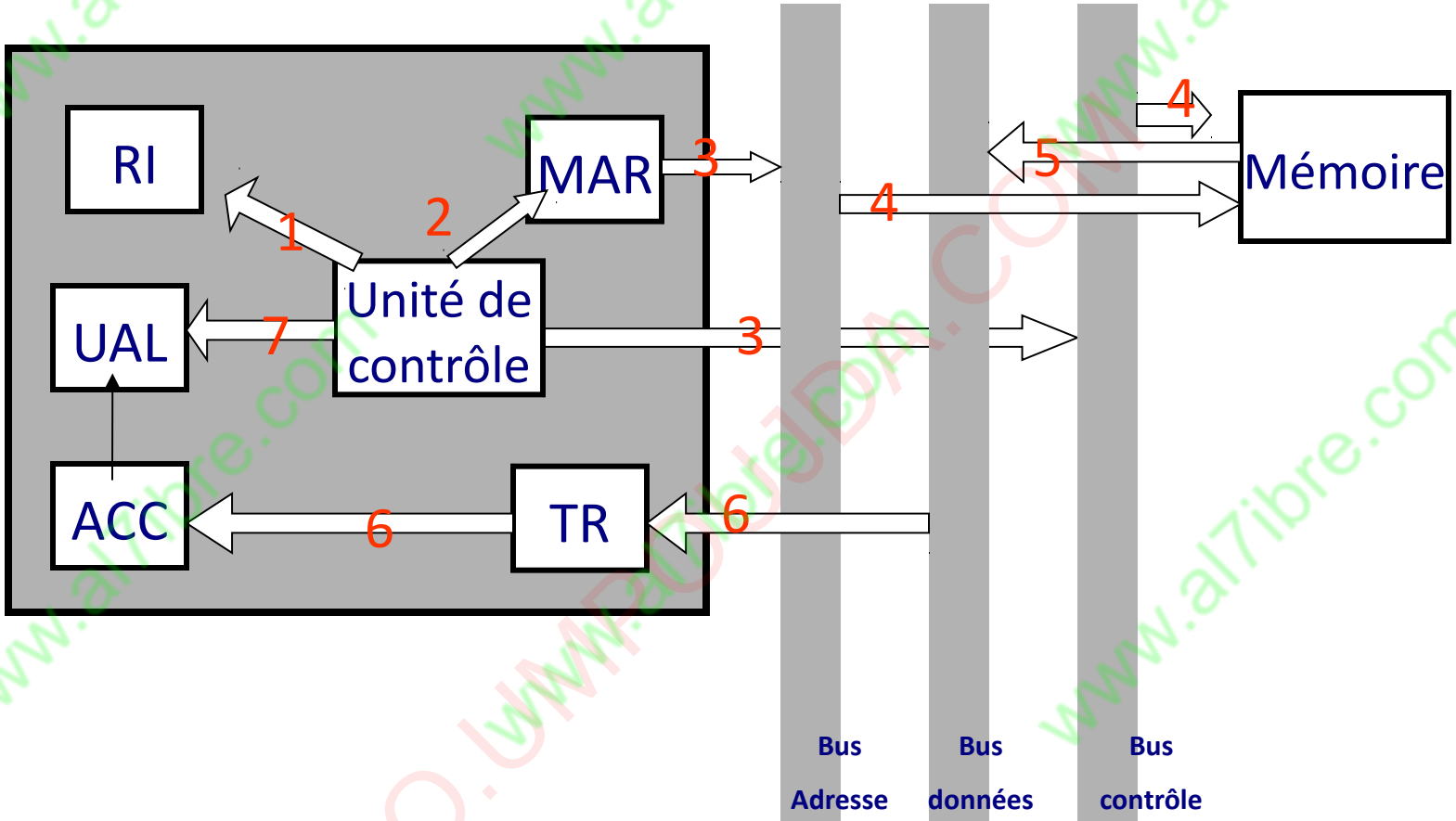


Fig 3 : Cycle d'interprétation et d'exécution

Cycle d'instruction normale

- Interprétation et exécution de l'instruction
- L'unité de contrôle :
  - lit la donnée stockée dans RI et détermine
    - L'adresse en mémoire de l'opérande
    - Le type d'opération qui doit être exécuter
  - place dans MAR l'adresse où est stockée l'opérande en mémoire.
  - demande une lecture de la mémoire principale

- Le résultat de la lecture est placé dans TR.
- Le contenu de TR est placé dans un « accumulateur» ou un registre qui stocke les données en paramètres des opérations.
- L'unité de contrôle demande à l'UAL d'exécuter l'opération.

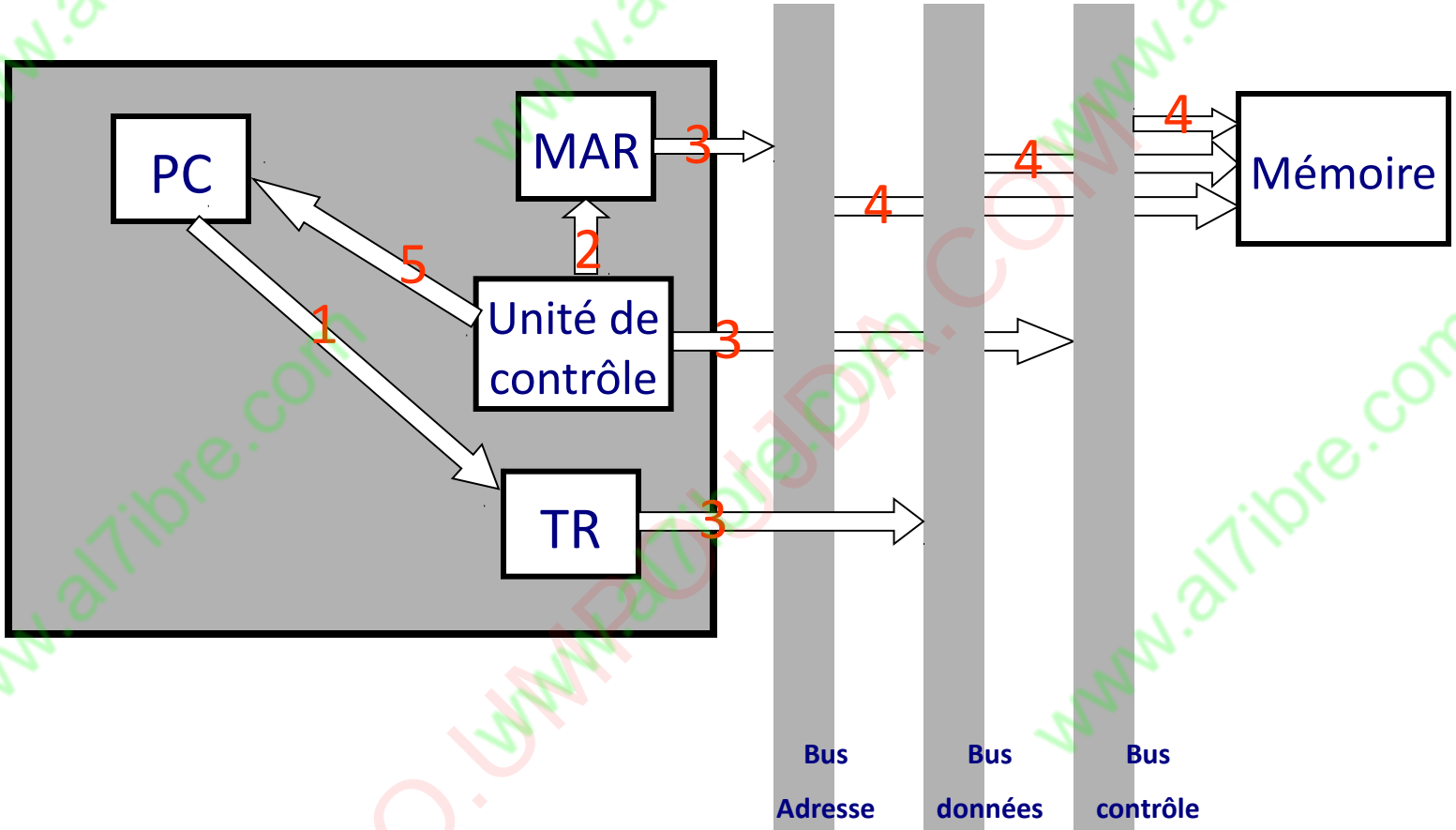


Fig 4 : Flot de données, cycle d'interruption

Interruption :

- Mécanisme selon lequel un module (E/S, mémoire) interrompt un cycle normal
  - Déroutements : Lors de l'exécution des instructions
    - Débordement numérique
    - Division par zéro
    - Non-reconnaissance d'une instruction
    - Accès à la mémoire illégale

- Interruptions matérielles :

- Gestion des périphériques

- Demande des données d'un périphérique

- Alerte provenant du matériel (batterie faible)

- Interruptions logicielles :

- Appels système

- ..

Interruptions :

en cas d'interruption

- sauvegarde de l'état courant
  - PC
  - PSW
  - ...
- exécution d'une suite d'instructions traitant l'interruption (routine d'interruption) et retour à l'état sauvegardé.

Cycle d'instruction : avec interruption

- Il faut sauvegarder le contenu de PC ainsi TR $\leftarrow$ PC pour être écrit dans la mémoire.
- MAR $\leftarrow$ une adresse mémoire ou sauvegarde PC.
- L'unité de contrôle demande une écriture dans la mémoire principale.
- PC $\leftarrow$ adresse de la routine d'interruption.

- Le nouveau cycle commence par la recherche de la première instruction de la routine d'interruption.

Jeu d'instructions :

Du point de vue du programmeur, la meilleure façon de comprendre le fonctionnement d'un processeur

connaître l'ensemble des instructions machine qu'il exécute.

Jeu d'instruction :

- On appelle jeu d'instructions l'ensemble des opérations élémentaires qu'un processeur peut exécuter. Il détermine ainsi son architecture.

- Il y a une correspondance directe entre le jeu d'instructions d'une machine, sa programmation et la conception de son processeur.

- La connaissance du jeu d'instruction par le programmeur offre une vue logique de la machine en terme de registre, de type de données ou de fonctionnement de l'UC et de l'UAL .

Types d'instructions :

- Pour un processeur de type Von Neumann le jeu d'instructions doit permettre de réaliser les tâches suivantes :
  - Traitement des données : instructions arithmétiques et logiques.
    - S'effectue sur les données qui se trouvent dans les registres du processeur.
  - Rangement des données : instruction mémoire
    - Transférer les données entre la mémoire et les registres.

➢ Mouvement de données : instructions d'E/S

- Introduire des programmes et des données en mémoire.
- Remettre à l'utilisateur les résultats du traitement

➢ Contrôle : instruction de test et de branchement

- Effectuer un branchement vers un autre ensemble d'instructions en fonction de la décision prise.

Jeu d'instructions :

- Une instruction doit contenir des informations requises par le processeur pour son exécution :
  - Le code opération : un code (binaire) identifiant l'opération à exécuter, appelé *opcode*.
  - Les références aux opérandes sources : adresses des opérandes (qui peuvent parvenir de la mémoire, d'un registre du processeur ou d'un périphérique).

- La référence à l'opérande résultat : référencer explicitement le résultat produit par l'opération.

- La référence à la prochaine instruction : indique au processeur ou chercher l'instruction suivant l'exécution de l'instruction courante (elle peut être implicite lorsque l'instruction suivante est contiguë à l'instruction courante).

Les opérandes source et résultat peuvent se situer dans trois zones différentes :

- Mémoire principale ou virtuelle : comme pour les références à l'instruction suivante, l'adresse en mémoire principale ou virtuelle doit être fournie.

- Registre de CPU.

- Périphériques d'E/S : l'instruction doit spécifier le module d'E/S.

Représentation de l'instruction :

- Chaque instruction est représentée par une séquence de bits découpée en champs.
- Il y a différents formats d'instructions selon les parties réservées aux opérandes.
- Durant l'exécution d'une instruction, celle-ci est chargée dans le registre d'instruction RI du processeur, qui doit en extraire les différents champs.

Jeu d'instructions :

Types d'instruction :

Instructions d'affectation : Déclenchent un transfert de données entre

l'un des registres du processeur et la mémoire principale.

- Transfert CPU ⟵ Mémoire Principale (MP) (= lecture en MP)

- Transfert CPU ⟶ Mémoire Principale (MP) (= écriture en MP)

Jeu d'instructions :

Types d'instruction :

Instructions arithmétiques et logiques :

Opérations entre une donnée et l'accumulateur AX. Le résultat est placé dans l'accumulateur.

La donnée peut être une constante ou une valeur contenue dans un emplacement mémoire.

Jeu d'instructions :

Types d'instruction :

Exemples :

- addition : AX ⟵ AX + donnée ;
- soustraction : AX ⟵ AX - donnée ;
- incrémentation de AX : AX ⟵ AX + 1 ;
- décrémentation : AX ⟵ AX - 1 ;
- décalages à gauche et à droite ; (Voir plus loin)

Jeu d'instructions :

Types d'instruction :

Instructions de comparaison :

Comparaison du registre AX à une donnée et positionnement des indicateurs.

Jeu d'instructions :

Types d'instruction :

Instructions de branchement :

La prochaine instruction à exécuter est repérée en mémoire par le registre PC.

Les instructions de branchement permettent de modifier la valeur de PC pour exécuter une autre instruction (boucles, tests, etc.).

On distingue deux types de branchements :

- branchements inconditionnels : PC ⟵ adresse d'une instruction ;
- branchements conditionnels : Si une condition est satisfaite, alors branchement, sinon passage simple à l'instruction suivante.

Jeu d'instructions :

Codage des instructions et mode d'adressage :

Les instructions et leurs opérandes (paramètres) sont stockés en mémoire principale.

La taille totale d'une instruction (nombre de bits nécessaires pour la représenter en mémoire) dépend du type d'instruction et aussi du type d'opérande.

Chaque instruction est toujours codée sur un nombre entier d'octets, afin de faciliter son décodage par le processeur.

Jeu d'instructions :

Codage des instructions :

Une instruction est composée de deux champs :

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

Champ Code Opération	Champ Code Opérande

Jeu d'instructions :

Codage des instructions :

Selon la manière dont la donnée est spécifiée, c'est à dire selon le mode d'adressage de la donnée, une instruction sera codée par 1, 2, 3 ou 4 octets.

Nous distinguerons ici quatre modes d'adressage : implicite, immédiat, direct et relatif (nous étudierons plus tard un autre mode, l'adressage indirect).

Jeu d'instructions :

Mode d'adressage :

Adressage implicite :

L'instruction contient seulement le code opération, sur 1 ou 2 octets.

Code opération (1 ou 2 octets)

L'instruction porte sur des registres ou spécifie une opération sans opérande.

Exemple : "incrémenter AX".

Jeu d'instructions :

Mode d'adressage :

Adressage immédiat :

Le champ opérande contient la donnée (une valeur constante sur 1 ou 2 octets).

Code opération (1 ou 2 octets)	Valeur (1 ou 2 octets)

Exemple : "Ajouter la valeur 5 à AX". Ici l'opérande 5 est codée sur 2 octets puisque l'opération porte sur un registre 16 bits (AX).

Jeu d'instructions :

Mode d'adressage :

Adressage direct

Le champ opérande contient l'adresse de la donnée en mémoire principale sur 2 octets.

Code opération (1 ou 2 octets)	Adresse de la donnée (2 octets)

Attention : dans le boîtier, les adresses sont toujours manipulées sur 16 bits, quelle que soit la taille réelle du bus. Nous verrons plus tard comment le processeur fabrique les adresses réelles sur 32 bits.

Exemple : "Placer dans AX la valeur contenue à l'adresse 130H".

Jeu d'instructions :

Mode d'adressage :

Adressage relatif :

Ce mode d'adressage est utilisé pour certaines instructions de branchement. Le champ opérande contient un entier relatif codé sur 1 ou 2 octets, nommé *déplacement*, qui sera ajouté à la valeur courante de IP.

Code opération (1 ou 2 octets)	Déplacement (1 ou 2 octets)

Jeu d'instructions :

Cas d'un adressage en mode réel :

Le processeur peut accéder à 1 Mo en fournissant des adresses sur 20 bits entre 0 et FFFFF (hexadécimal).

Or, le processeur Intel 8086 possède des registres sur 16 bits, ce qui l'empêchait de représenter directement une adresse sur 20 bits.

Solution : - Créer une segmentation de la mémoire.

- Tout l'espace mémoire est divisé en unités de 64 ko, appelées **segments**.

Jeu d'instructions :

Cas d'un adressage en mode réel :

En mode réel, l'adresse linéaire est sur 20 bits. Le programme en mode réel ne peut pas utiliser directement une adresse linéaire.

Solution : Utiliser deux valeurs sur 16 bits, considérées en couple, sont appelées adresse segment-offset.

- La valeur de segment sur 16 bits est stockée dans un des registres de segment (CS, DS, ES et SS).
- Il s'y ajoute une valeur d'offset sur 16 bits (de décalage ou déplacement).

Jeu d'instructions :

Cas d'un adressage en mode réel :

Lorsqu'une adresse est exprimé de cette manière, le processeur applique un calcul arithmétique pour convertir le couple segment-offset en une adresse linéaire sur 20 bits.

Exemple.

Jeu d'instructions :

Temps d'exécution :

Chaque instruction nécessite un certain nombre de cycles d'horloges pour s'effectuer.

Le nombre de cycles dépend de la complexité de l'instruction et aussi du mode d'adressage : il est plus long d'accéder à la mémoire principale qu'à un registre du processeur.

La durée d'un cycle dépend bien sur de la fréquence d'horloge de l'ordinateur. Plus l'horloge bat rapidement, plus un cycle est court et plus on exécute un grand nombre d'instructions par seconde.

Représentation de l'instruction :

Ecriture des instructions en langage symbolique :

Voici un programme en langage machine 80486, implanté à l'adresse 0100H :

A1 01 10 03 06 01 12 A3 01 14

Ce programme additionne le contenu de deux cases mémoire et range le résultat dans une troisième.

Nous avons simplement transcrit en hexadécimal le code du programme.

Il est clair que ce type d'écriture n'est pas très utilisable par un être humain.

Représentation de l'instruction :

Ecriture des instructions en langage symbolique :

A chaque instruction que peut exécuter le processeur correspond une représentation binaire sur un ou plusieurs octets, comme on l'a vu plus haut.

C'est le travail du processeur de décoder cette représentation pour effectuer les opérations correspondantes.

Représentation de l'instruction :

Ecriture des instructions en langage symbolique :

Afin de pouvoir écrire (et relire) des programmes en langage machine, on utilise une notation symbolique, appelée *langage assembleur*. Ainsi, la première instruction du programme ci-dessus (code A1 01 10) sera notée :

MOV AX, [0110]

elle indique que le mot mémoire d'adresse 0110H est chargé dans le registre AX du processeur.

On utilise des programmes spéciaux, appelés *assembleurs*, pour traduire automatiquement le langage symbolique en code machine.

Représentation de l'instruction :

Ecriture des instructions en langage symbolique :

Voici une transcription langage symbolique du programme complet. L'adresse de début de chaque instruction est indiquée à gauche (en hexadécimal).

Adresse	Contenu MP	Langage Symbolique	Explication en français
0100	A1 01 10	MOV AX, [0110]	Charger AX avec le contenu de 0110.
0103	03 06 01 12	ADD AX, [0112]	Ajouter le contenu de 0112 à AX (résultat dans AX).
0107	A3 01 14	MOV [0114], AX	Ranger AX en 0114.

Représentation de l'instruction :

Ecriture des instructions en langage symbolique :

Sens des mouvements de données :

La plupart des instructions spécifient des mouvements de données entre la mémoire principale et le microprocesseur. En langage symbolique, on indique toujours la destination, puis la source. Ainsi l'instruction :

MOV AX, [0110]

transfère le contenu de l'emplacement mémoire 0110H dans l'accumulateur, tandis que : MOV [0112], AX

transfère le contenu de l'accumulateur dans l'emplacement mémoire 0112.

Représentation de l'instruction :

Ecriture des instructions en langage symbolique :

L'instruction MOV (de l'anglais move, déplacer) s'écrit donc toujours :

MOV destination, source

Représentation de l'instruction :

Ecriture des instructions en langage symbolique :

Mode d'adressage :

En adressage immédiat, on indique simplement la valeur de l'opérande en hexadécimal.

Exemple : MOV AX, 12

Représentation de l'instruction :

Ecriture des instructions en langage symbolique :

Mode d'adressage :

En adressage direct, on indique l'adresse d'un emplacement en mémoire principale en hexadécimal entre crochets :

Exemple : MOV AX, [A340]

Représentation de l'instruction :

Ecriture des instructions en langage symbolique :

Mode d'adressage :

En adressage relatif, on indique simplement l'adresse (hexa).

L'assembleur traduit automatiquement cette adresse en un déplacement (relatif sur un octet).

Exemple : JNE 0108

(nous étudierons l'instruction JNE plus loin).

Représentation de l'instruction :

Utilisation du programme *debug :*

debug est un programme qui s'exécute sur PC (sous DOS) et qui permet de manipuler des programmes en langage symbolique.

Il est normalement distribué avec toutes les versions du système MS/DOS.

Représentation de l'instruction :

Utilisation du programme *debug :*

Les fonctionnalités principales de *debug sont les suivantes :*

- Affichage du contenu d'une zone mémoire en hexadécimal ou en ASCII ;
- Modification du contenu d'une case mémoire quelconque ;
- Affichage en langage symbolique d'un programme ;
- Entrée d'un programme en langage symbolique ; *debug* traduit les instructions en langage machine et calcule automatiquement les déplacements en adressage relatif.
- Affichage et modification de la valeur des registres du processeur ;

Autres instructions

Représentation de l'instruction :

- La séquence de bits admet une représentation symbolique, où l'opcode est représenté par une abréviation appelée *mnémonique* :

ADD	Ajouter
SUB	Soustraire
MUL	Multiplier
DIV	Diviser
LOAD	Charger les données depuis la mémoire
STOR	Ranger les données en mémoire

- Une instruction en langage de haut niveau (C, Pascal …) correspond en fait à plusieurs instructions en langage machine.

Représentation de l'instruction :

Prenons une instruction en langage de haut niveau : X = X + Y.

Si X et Y correspondent aux emplacements respectifs 513, 514.

En langage machine on aura trois instructions :

- Charger un registre avec le contenu de l'emplacement mémoire 513.

- Ajouter le contenu de l'emplacement mémoire 514 au registre.

- Ranger le contenu du registre dans l'emplacement mémoire 513.

L'ensemble d'instructions machine doit être suffisamment étendu pour exprimer n'importe laquelle des instructions d'un langage évolué. Cela nous permet de distinguer plusieurs types d'instructions :

- Traitement de données : instructions arithmétiques et logiques.

- Rangement de données : instructions mémoire.

- Mouvement de données : instructions d'E/S.

- Contrôle : instructions de test et de branchement.

Nombre d'adresses :

- En théorie, il faut quatre adresses (pour deux opérandes, un résultat, la prochaine instruction).

En pratique :

- Les formats d'instructions à deux adresses nécessitent que l'une des adresses fasse référence de source et destination.

- Les instructions basées sur une seule adresse emploient systématiquement un registre CPU, appelé accumulateur.

- Les instructions n'employant aucune adresse utilisent des registres implicites.

Conception du jeu d'instructions :

Voici les problèmes de conception les plus importants :

- **Répertoire des opérations :** combien d'opérations prévoir et lesquelles, et leur degré de complexité.

- **Types de données :** les différents types de données sur lesquelles il sera possible d'accomplir des opérations.

- **Format des instructions :** longueurs des instructions (en bits), nombre d'adresses, taille des différents champs, etc..

- **Registres :** nombre de registres du CPU que les instructions peuvent référencés et leur utilisation.

- **Adressage :** modalités de spécification des adresses d'opérandes.

Chapitre 3 :

Langage Assembleur

Généralités

Pourquoi faire de l'assembleur ?

- Écrire un compilateur.
- Environnements embarquées (système de contrôle et d'allumage des automobiles, régulateurs d'air conditionné, modem, ...).

Avantages :

- Meilleure compréhension des langages.
- Meilleure compréhension des machines et des processeurs.

Au niveau du processeur ?

Deux fonctions :

- Calculer et Stocker

Langage spécifique utilisé :

- Un langage par processeur, appelé **jeu d'instructions**.
- Une référence commune : le **binaire**

Au niveau de la mémoire ?

Mémoires de calcul :

- Les calculs se décomposent en **opérations élémentaires**.
- Ces opérations ont besoin de stocker des **résultats intermédiaires**.
- Le processeur contient des petites mémoires appelées **registres**.

- La mémoire se comporte comme un grand **tableau d'octets**.

- Accéder à la mémoire consiste à accéder à une case du tableau en indiquant son indice, appelé **adresse**.

La segmentation de la mémoire en mode réel ou protégé :

Les programmes pour processeur Intel sont structurés autour de la notion de segment.

Un programme comprend en général :

- un segment de code
- un segment de données
- un segment de pile (stack)

Le segment de code : contient toutes les instructions, et au minimum une procédure, qui est dans ce cas la procédure de démarrage (: main).

Le segment de données contient les variables.

Le segment de pile : contient les paramètres de la procédure et les variables locales.

Les quatre registres :

- CS (Code Segment),
- DS (Data Segment)
- SS (Stack Segment)
- ES (Extra Segment)

sont utilisés pour stocker les adresses des segments.

Parties principales d'un programme en machine

Parties principales d'un programme en machine :

- Zones Instructions : positions mémoires contenant la représentation binaire des instructions à exécuter l'une après l'autre

- Zones données : positions mémoires pour la représentation des valeurs des variables des programmes.

- Zone pile : Gestion dynamique des données temporaires

En général, ces zones (segments) sont stockés dans différentes parties de la mémoire, disjointes les unes des autres.

Description de “ la Zone pile ” d’un programme :

- Zone mémoire réservée au stockage temporaire des données

  Concept :

  - Zone de mémoire gérée dynamiquement
  - Actions de base : empiler – des-empiler

- La valeur SS:SP désigne l’adresse courante sur la pile (le pointeur pile : "stack pointer")

Description de la “zone-instructions” d’un programme :

➔ Les premières instructions du programme : servent à définir l’état des différents registres de contrôle du CPU.

➔ Partie “d’initialisation de l’environnement” :

Le début de cette partie : sert à initialiser l’état des variables (tableaux, structures de données) utilisées par le programme.

➔ **La dernière instruction du programme sert à rendre le contrôle au système d'exploitation.**

Remarque : ***"premier" et "dernier" dans l'ordre de l'exécution des instructions (= de la "séquence logique") - qui peut être différent de l'ordre physique du stockage des instructions (= de leur "séquence physique").***

Séquence physique des instructions :

Suite des positions de la mémoire utilisées pour le stockage des instructions.

Séquence logique des instructions :

Séquence déterminée par l'exécution des instructions lorsque le CPU les cherche et les interprète.

Les processeurs

Les microprocesseurs :

Deux grandes familles :

– **Les microprocesseurs CISC (Complex Instruction Set Computer)**

» **Jeu d'instruction étendu : taille « moyenne »**

» **2 grands constructeurs :**

- **INTEL (8088, 8086, 80286... Pentium...)**
- **MOTOROLA (68000... 68040)**

– **Les microprocesseurs RISC (Reduced Instruction Set Computer)**

» **Jeu d'instruction réduit : taille « faible »**

» **cela permet de rajouter plus de registres, de mémoire cache, ... sur le processeur**

- **IBM / MOTOROLA (PowerPC)**
- **SUN (Supersparc)**
- **DIGITAL (Alpha)**

La famille des processeurs 80x86 :

Utilisés dans les premiers PC.

- offrent plusieurs registres 16 bits.
- n'opèrent qu'en mode réel pour un adressage sur 20 bits
  - permet d'adresser jusqu'à 1Mo de mémoire
  - un programme peut accéder à n'importe quelle adresse mémoire, même la mémoire des autres programmes!
  - Cela rend le débogage et la sécurité très difficiles!
  - La mémoire du programme doit être divisée en segments.

La segmentation de la mémoire en mode réel

- **Avec un registre de 16 bits il est impossible d'adresser un espace de 1Mo**
  - **Créer une segmentation de la mémoire**
  - **Un segment est une zone de 64Ko (16 bits),**
  - **Les segments sont espacés à intervalles de 16 octets**
  - **L'adresse d'un octet se note XXXX:YYYY où XXXX est l'adresse de segment et YYYY est le déplacement (tous deux en notation hexadécimale, bien sûr).**
  - **L'adresse physique est calculée par :**

    Adresse physique = xxxx * 16 + yyyy

La conséquence immédiate est qu'*un octet n'a pas une adresse unique.*

Par exemple, l'octet numéro 66 peut être Adressé par 0000:0042, mais aussi par 0001:0032, par 0002:0022, par 0003:0012 ou encore par 0004:0002. Toutes ces adresses sont équivalentes.

La famille des processeurs 80286 :

- **Apporte quelques nouvelles instructions au langage machine de base des 8088/86.**

- **Peut accéder jusqu'à 16 Mo avec un espace d'adressage de 24 bits de registre.**

- Empêcher les programmes d'accéder à la mémoire des uns et des autres.
- Le code n'a le droit d'accéder à tous les segments que sous certaines conditions (3 modes de protection).
- Une vérification de la limite d'adressage est effectuée.
- Cependant, les programmes sont toujours divisés en segments qui ne peuvent pas dépasser les 64 Ko.

La famille des processeurs 80386 :

- **Les registres sont étendus à 32 bits**
- **Ajout de deux nouveaux registres 16 bits**
- **Un espace d'adressage de 32 bits : 4 Go**
- **Ajout également d'un nouveau mode protégé 32 bits.**
  - **Peut accéder jusqu'à 4 Go de mémoire.**
  - **Les programmes sont encore divisés en segments;**
  - **Un nouveau système d'adressage de la mémoire virtuelle qui permet de gérer un espace plus grand que la mémoire physique.**

La famille des processeurs 80486

Pentium/Pentium Pro :

- Apportent très peu de nouvelles fonctionnalités. Ils accélèrent principalement l'exécution des instructions.
- Intègre les cache de niveau 1 et 2
- Support multi-processeurs

Pentium MMX :

- Ce processeur ajoute les instructions MMX (MultiMedia eXentions) au Pentium.
  - Ces instructions peuvent accélérer des opérations graphiques courantes.

2000 : Pentium 4 (42 M transistors)

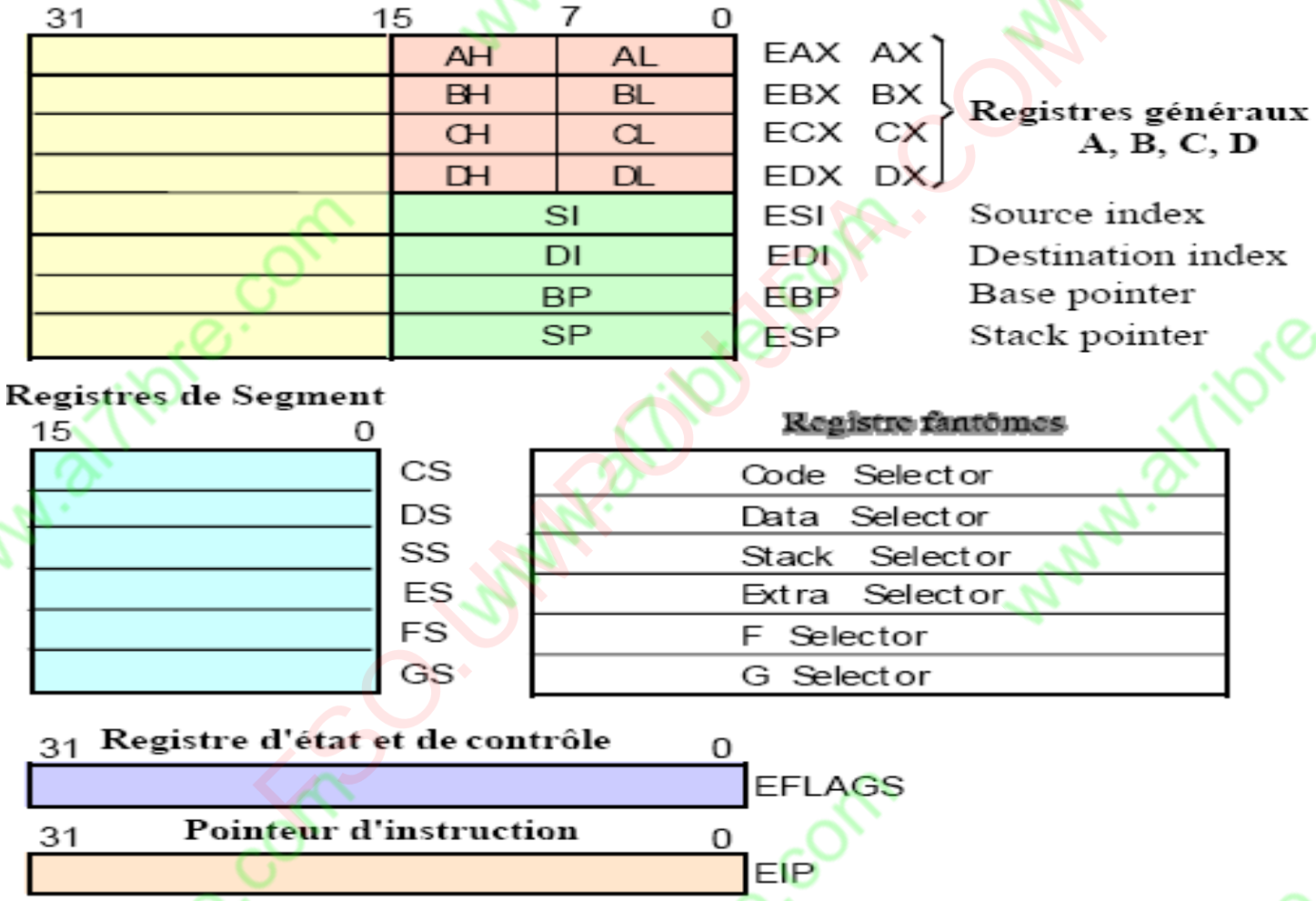
- Hyper-threading : 2 taches de contrôle simultanées

2003 : Pentium M (Centrino)

- Plus petit, plus léger,

- Actuellement 2016 ?

Registres du processeur 80386 et les suivants :



Registres du processeur 80386 et les suivants :

- **Registres généraux 32 bits**
- **Dans les processeurs Intel de la famille i386 et les suivants (dont les Pentium), ainsi que dans les processeurs compatibles (comme les Athlon d'AMD), il y a 8 registres 32 bits général.**

Les registres des générations précédentes ne faisant que 16 bits. Les registres 32 bits sont préfixés par la lettre E qui signifie < extended >.

Ces registres se nomment :

- **EAX : un registre général, utilisé comme registre par défaut, ou accumulateur**
- **EBX : un registre général ;**
- **ECX : un registre général, souvent utilisé pour compter.**

- **EDX : un registre général pour les données**
- **ESI : le registre < Source Index >, qui pointe sur la lettre courante d'une chaîne de caractères source.**
- **EDI : le registre < Destination Index >, qui pointe sur la lettre courante d'une chaîne de caractères destination.**

- **EBP : le registre < Base Pointer >, qui sert a conserver des informations relatives à la pile lors d'appels de fonctions ;**
- **ESP : le registre < Stack Pointer > qui pointe sur le sommet de la pile.**

➢ Registre généraux 16 bits

➢ Les 16 bits de poids faible de ces registres peuvent être accédés, par le même nom que celui utilisé dans les machines 32 bits, c'est-à-dire : AX, BX, CX, DX, SI, DI, BP, SP.

➢ Les registres AX, BX, CX, DX sont eux-mêmes chacun accessibles par deux sous-registres de 8 bits,

AH:AL; BH:BL; CH:CL; DH:DL

- Registres segments :
  - Outre ces registres généraux, le processeur comprend d'autres registres comme CS, DS, SS, ES, FS, GS qui servent pour une gestion dite < segmentée > de la mémoire.
  - CS segment de code.
  - DS segment de données.
  - SS segment de pile.
  - ES segment de données additionnels.
  - FS segment de données additionnels.
  - GS segment de données additionnels.

Registres d'Offset :

IP : Pointeur d'instruction

SP : Pointeur de pile

SI : Index de source

DI : Index de destination

BP : Pointeur de base

Registres des indicateurs : Cf. Voir Chap 4

Il existe à cet effet de petits indicateurs, les ***flags*** qui sont des bits spéciaux ayant une signification très précise.

De manière générale, *les **flags** fournissent des informations sur les résultats des opérations précédentes.*

Rangement des données en mémoire :

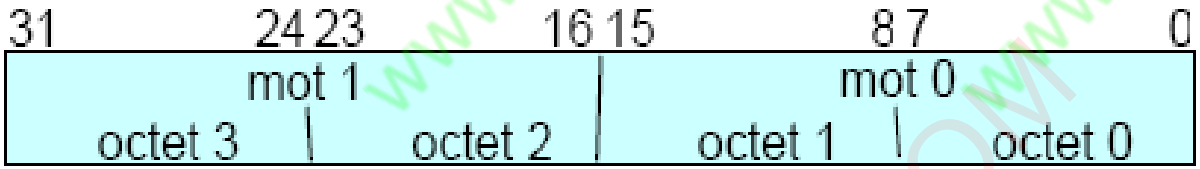
il y a deux façons de ranger les différents octets d'un registre de plus de 8 bits lors de l'écriture de son contenu en mémoire.

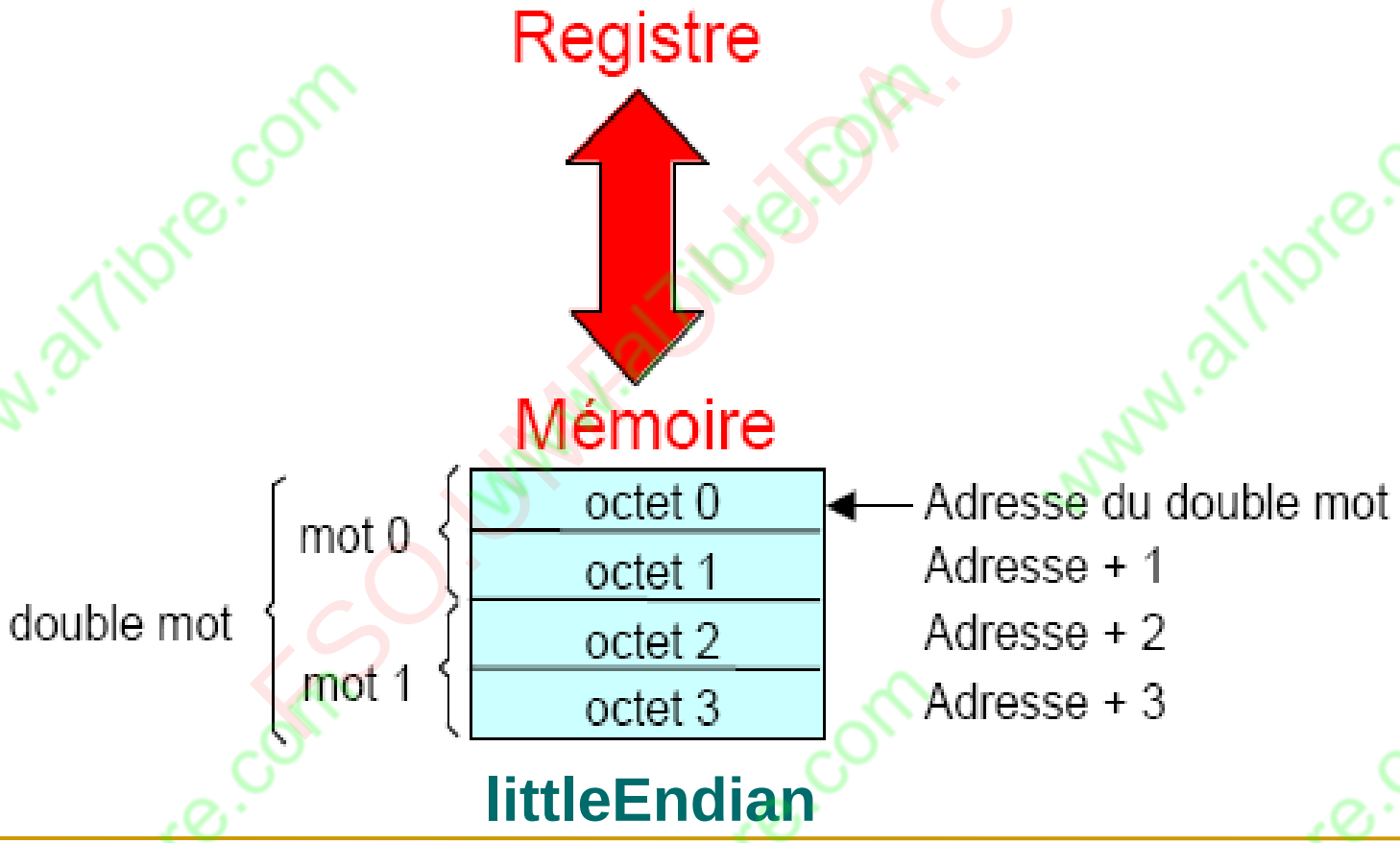
On peut mettre l'octet de poids fort à :

- l'adresse basse (big Endian)
- ou à l'adresse haute (littleEndian).

Intel a choisi la seconde solution :

Rangement des données en mémoire





Assembleur

Quelques définitions :

- Langage Machine = ordres (en binaire) compréhensibles par un processeur donné

- Exécutable = suite d'instructions en langage machine

- Langage Assembleur = mnémoniques associées au langage machine (JUMP, ADD, MOV...)

- Assemblage = utilisation d'un logiciel spécifique (logiciel d'assemblage ou *assembleur*) pour transformer une suite d'instructions écrites en langage assembleur en un exécutable

L’assembleur

Pour un même microprocesseur, plusieurs langages assembleurs sont possibles. Ils se distinguent surtout par les directives (.data?, dss, …) et macros.

Les deux logiciels assembleurs les plus connus pour la famille Intel sont MASM (Microsoft ASseMbler) et TASM (Turbo ASseMbler de borland).

L’outil qu’on va utilisé dans ce cours est **MASM**.

Dans ce chapitre, nous verrons :

- Comment définir et déclarer des variables et des constantes en utilisant la syntaxe MASM
- Comment écrire un programme complet, que nous allons analyser ligne par ligne

Les sections de MASM :

On utilise des sections différentes pour placer le code et les données.

- Trois sections importantes :
  - .code : code assembleur
  - .data : données initialisées
  - .data? : données non initialisées (allocation de mémoire)

```
include Irvine32.inc
;définition de symbole ici
.data
; Déclaration  des données initialisées ici
segment .data?
;Déclaration  des données non initialisées ici
.code
main PROC
;les instructions sont  placées ici
exit ;retour au système d'exploitation
main ENDP
;définition des procédures ici
END main
```

Squelette d'un programme MASM

Éléments constitutifs :

- **Constantes entières :**
  - **[{+|-}]chiffres [base]**
  - **Base peut être l'un des symboles suivants :**

    h : hexadécimal, q/o : octal, d : décimal, b : binaire

Exemple

- **Expression entière :**
  - **Expression mathématique dans laquelle sont mentionnées des valeurs entières et au moins un opérateur arithmétique**
  - **Opérateur arithmétique par ordre de priorité décroissante**

    ( ) +,- unaire *,/ MOD +,- addition, soustraction

Exemple

- **Constante caractère : "A " , 'b'**
- **Constante chaîne : "ABC " , 'bonne'**

Éléments constitutifs :

- **Identificateur : nom choisi par le programmeur**
  - **Permet d'identifier une variable, une constante, une procédure ou un label dans un code.**
  - **Peut être constitué de 1 à 247 caractères.**
  - **Les majuscules ne sont pas distinguées des minuscules.**
  - **Le premier caractère doit être une lettre (A..Z, a..z), @, _ ou $. Les caractères qui suivent peuvent être des chiffres.**
  - **Un identificateur ne peut pas être identique à un mot réservé du langage.**

➢ Directives :

- ➢ Commande reconnue par l'assembleur et provoque une réaction de sa part pendant l'assemblage du code.
- ➢ Font partie de la syntaxe du langage assembleur mais ne dépendent pas du jeu d'instructions Intel.
- ➢ Exemple .DATA , .Code, PROC, BYTE, WORD…

Exemples de directives :

- **.**data : sert à identifier le début de la zone d'un programme source qui contient des variables.
- **.**code : définit le début de la zone d'un programme qui contient des instructions.
- PROC : identifie le début d'une définition de procédure.

Le nom de la procédure doit être un identificateur :

nomproc PROC

Éléments constitutifs :

Instructions :

Format standard d'une instruction :

Label: (fac)	Mnémonique d'instruction (oblig)	Opérande(s)(oblig)	;Commentaires (facult)

Labels :

Un label est un identificateur qui permet de désigner un endroit dans le code source, soit une instruction soit une donnée.

Les labels de code

- Faire référence à l'adresse d'une instruction

```
cible :
        mov ax, bx
        ....
        jmp cible
```

➢ Les labels de données

- ➢ **est défini dans la zone de code d'un programme, et ne se termine pas par le signe (:).**

premiere BYTE 10

Par exemple, le label premiere est un label qui doit être unique au niveau du même fichier.

Mnémoniques des instructions :

Mot court qui identifie une instruction.

MOV Déplace (affecte) une valeur à une autre

ADD Additionne deux valeurs

SUB Soustrait une valeur d'une autre

MUL Multiplie deux valeurs

JMP Saute vers une nouvelle adresse

CALL Appelle une procédure

Opérandes :

- Une instruction assembleur peut avoir entre zéro et trois opérandes. Chaque opérande peut être :
  - Nom d'un registre.
  - Opérande mémoire.
  - Expression constante.
  - Nom d'un port d'entrées/sorties.

- Lorsqu'on choisi le nom d'une variable, on signifie en fait l'adresse de cette variable et on demande à l'ordinateur de lire le contenu de l'adresse mémoire.

Exemple	Type d'opérande
96	Constante (valeur immédiate)
2+4	Expression constante
eax	Registre
compteur	Emplacement mémoire

Exemple : Zéro opérande

Commentaires :

- les commentaires sur une seule ligne doivent commencer par le signe point-virgule (;).

- les commentaires en blocs :

```
COMMENT !
    cette ligne est un commentaire.
    cette ligne est aussi un commentaire.
  !
```

Voici quelques exemples d'instructions assembleur utilisant des valeurs numériques et des opérandes :

L'instruction INC attend un seul opérande :

inc ax ; ajoute 1 au contenu de AX

L'instruction MOV attend deux opérandes :

mov compteur , bx ; Copie le contenu de BX dans la variable compteur

Premier exemple : addition de trois nombres entiers

```
TITLE Addition et soustraction
; addition et soustraction d'entiers sur 32 bits
; last update : 12 / 3 / 15
Include Irvine32.inc

.code
main PROC
mov eax, 10000h    ; EAX = 10000h
add eax, 40000h    ;  EAX = 50000h
sub eax, 20000h     ; EAX = 30000h
call DumpRegs          ; Affichage des registres
exit
main ENDP
END main
```

Définition / Déclaration des données

Types de données intrinsèques :

MASM définit plusieurs types de données intrinséques.

Types de données

Type	Utilisation
BYTE	Entier non signé sur 8 bits
SBYTE	Entier signé sur 8 bits
WORD	Entier non signé sur 16 bits
SWORD	Entier signé sur 16 bits
DWORD	Entier non signé sur 32 bits
SDWORD	Entier singé sur 32 bits
FWORD	Entier sur 48 bits
QWORD	Entier sur 64 bits
TBYTE	Entier sur 80 bits (10 octets)
REAL4	Réel court IEEE sur 32 bits (4 octets)
REAL8	Réel long IEEE sur 64 bits (8 octets)
REAL10	Réel étendu IEEE sur 80 bits(10octets)

Instructions de déclarations de données :

Permet de réserver un emplacement mémoire suffisant pour le stockage de la valeur d'une variable.

Toute déclaration de données possède la syntaxe suivante :

[nom] directive initialiseur [,initialiseur] ...

Exemple :

Les deux directives BYTE et SBYTE

```
valeur1  BYTE ‘A’    ; constante caractère
valeur2  BYTE 0      ; plus petit octet non signé
Valeur3  BYTE 255    ; plus grand octet non signé
Valeur4  SBYTE -128  ; plus petit octet signé
Valeur5  SBYTE +127  ; plus grand octet signé
```

Nom de variable :

Label qui permet de faire référence à l'adresse mémoire à laquelle se trouve la valeur de la variable.

Ex :

```
.data
 valeur1 BYTE 10h
  valeur2 BYTE 20h
```

Initialiseurs multiples :

.data

liste BYTE 10, 20, 30, 40

Exemple : Comment sont stockés les 4 octets de liste avec l'adresse offset ?

Zone de données non initialisées (.data?)

- Réserver de l'espace pour des données sans leur donner de valeur initiale.
- Utile pour préparer la place pour des grandes blocs de données.

A7 BYTE ? ; déclaration d'un octet non initialisé
mot1 WORD ? ; réserve 1 mot de 16 bits
Mot64 WORD 64 DUP(?); réserve 128 octets

L'opérateur de duplication DUP :

- Permet de répéter une réservation d'espace

- Très utile lorsque vous voulez réserver de l'espace pour une chaîne de caractères ou un tableau.

Exemple :

Il peut aussi être utilisé avec des déclarations de données à valeurs initiales :

```
A  BYTE 20 DUP (0)   ; 20 octets, tous égaux à 0
BB  BYTE 20 DUP (?)   ; 20 octets, non initialisés
CC  BYTE   4 DUP ('PILE') ; 16 octets :
                          "PILEPILEPILEPILE"
```

Mélange de code et de données :

Le langage assembleur vous autorise à passer d'un bloc de code à un bloc de données dans votre programme source.

```
. code
 mov eax, ebx
. data
 temp DWORD ?
. code
 mov temp, eax
 ....
```

Zone de données initialisées (.data)

- **A1 BYTE 4** ; octet non signé initialisé avec la valeur 4
- **B1 SBYTE –5** ; octet signé initialisé avec la valeur -5
- **A2 WORD 2000** ; mot initialisé avec la valeur 2000
- **B2 SWORD -2000** ; mot signé initialisé avec la valeur -2000
- **A3 BYTE 111101b** ; octet initialisé à la valeur binaire 111101
- **A5 BYTE 16o** ; octet initialisé à la valeur octale 16
- **A6 WORD 1B91h** ; mot initialisé à la valeur hexa 1B91
- **A8 BYTE "B"** ; octet initialisé avec le code ASCII du B
- **A9 BYTE 0, 1, 2, 3,5** ; définit 5 octets initialisés
- **A10 BYTE "word", 0** ; définit une chaîne = "word"
- **A11 BYTE 20 DUP(0)** ; 20 octets, tous initialisés à zéro

Exemple : Anciennes directives

BYTE et SBYTE :

WORD et SWORD :

DWORD et SDWORD :

QWORD :

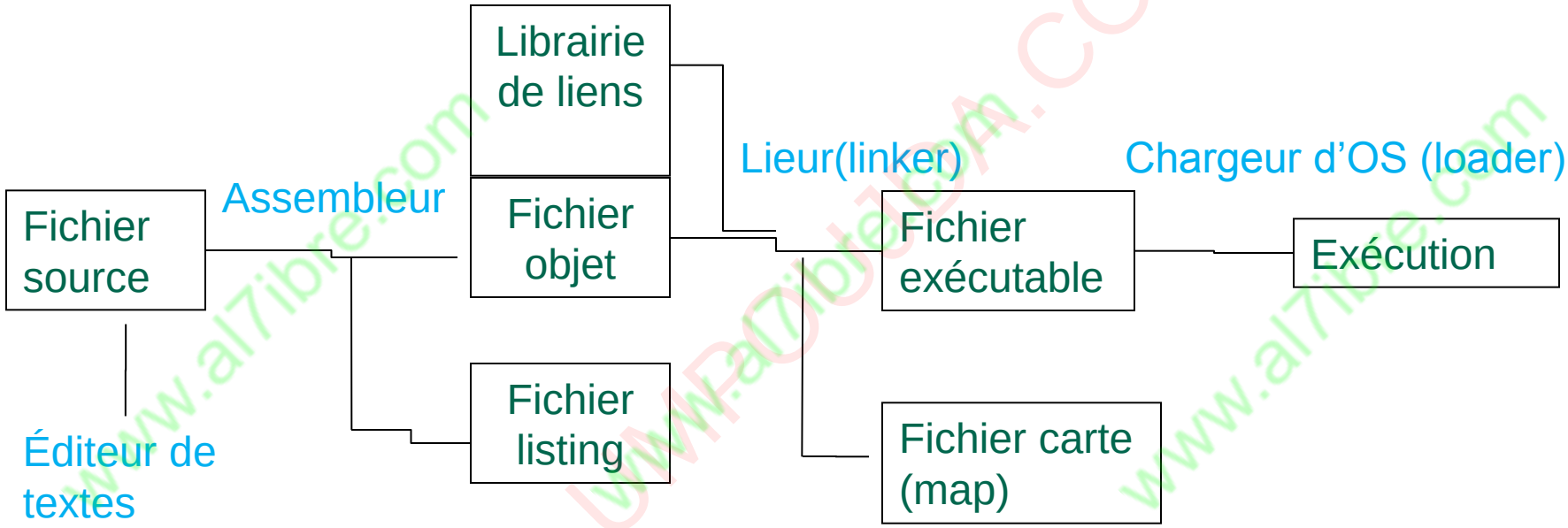
TBYTE :

Version amélioré du premier programme :

En nous servant de ce que nous avons appris entretemps au niveau des directives de déclaration des données.

Nous allons pouvoir définir un segment de données qui déclare plusieurs variables.

Assemblage, liaison et exécution d'un programme :



Le fichier carte (à extension .MAP) donne :

- Une fois la liaison est réalisée, il donne des informations au sujet des segments mémoire exploités par un programme.
- L'adresse du point d'entrée du programme (première instruction à exécuter)
- …

Opérateurs et directives relatifs aux données :

- **OFFSET :** renvoie la distance (l'adresse) d'une variable depuis le début du segment dans lequel il est déclarée.
- **TYPE :** renvoie la taille, en octets, d'une variable complexe ou d'une variable simple.
- **LENGTHOF :** renvoie le nombre d'éléments d'un tableau.
- **SIZEOF :** renvoie une valeur qui équivaut à la multiplication de la valeur de **LENGTHOF** par celle de **TYPE.**

Entrées et Sorties :

- **ReadInt :** lit un entier signé sur 32 bit au clavier et le stocke dans le registre **EAX. ReadInt** met à 1 le drapeau de débordement **OF** si l'entier ne peut pas être stocké sous forme d'un entier signé sur 32 bits**.**

- **ReadChar : L**it un caractère au clavier et stocke son code ASCII dans le registre AL.
- **Crlf : P**lace le curseur au début de la ligne suivante.
- **Clrscr : E**fface ce qui se trouve sur l'écran.

Entrées et Sorties :

- **WriteInt :** affiche à l'écran la valeur d'un entier signé sur 32 bit stocké dans EAX

```
Mov eax , 216543
Call WriteInt  ; affiche : +216543
```

- **WriteChar :** affiche à l'écran le caractère stocké dans AL

```
Mov AL, 'A'
Call writechar ; affiche  : A
```

WriteString : affiche à l'écran le contenu de la chaîne dont l'adresse offset du début est stockée dans EDX.

La chaîne doit être terminée par 0.

Exemple :

```
.data
 Message BYTE  " saisissez votre nom: ", 0
.code
 Mov edx, OFFSET Message
 Call WriteString ; affiche  : saisissez votre nom:
```

Débogage :

dumpRegs : exemple

- affiche les valeurs des registres (en hexadécimal) de l'ordinateur sur stdout (i.e. l'écran).
- affiche les drapeaux **CF, SF, ZF, OF** du registre **FLAGS. Exemple suivant**

dumpMem :

- Affiche les valeurs d'une région de la mémoire (en hexadécimal et également en caractères ASCII).
- Pour l'appeler, placez d'abord l'adresse de départ dans **ESI,** le nombre d'unité dans **ECX** et la taille unitaire dans **EBX (**1=octet, 2=mot,…**)**

```
Include Irvine32.inc
.data
val1   dword  10000h          ; dword 32 bit
val2  dword  40000h
val3   dword  20000h
finalVal   dword  ?
.code
main PROC

mov  eax, val1     ;   Copie de 10000h
Add  eax, val2      ;   Addition de 40000h
sub  eax, val3      ; Soustraction de 20000h
mov  finalVal, eax ; Stockage du résultat (30000h)
Call  DumpRegs     ; Affichage des Registres

exit
main ENDP
END main
```

Exemple :

```
.data
Tableau DWORD 1,2,3,4,5,6,7,8,9,0Ah,0Bh
.code
main PROC
mov esi, OFFSET tableau ; offset de départ
mov ecx, LENGHOF tableau ; nombre d'unités
mov ebx, TYPE tableau ; format double mot
call DumpMem
```

DumpMem affiche les résultats en hexadécimal :

00000001 00000002 00000003 00000004 00000005 00000006

00000007 00000008 00000009 0000000A 0000000B

Directives :

➢ La directive equ

➢ La directive *equ* permet d'associer un nom symbolique à une expression entière ou à du texte. Trois formats sont autorisés :

nom EQU expression

nom EQU symbole

nom EQU <texte>

Exemple :

symbole equ 100

PI EQU <3.1416>

➢ La directive =

➢ Permet d'associer un nom de symbole à une expression numérique entière.

Par exemple, lorsque l'assembleur rencontre les deux lignes suivantes :

compteur = 500

mov al, compteur

Il génère une seule instruction machine :

mov al, 500

➢ La directive %define :

➢ Semblable à la directive #define du C

- %define SIZE 100
- Mov EAX, SIZE

Opérations :

- Opérations sont classées en plusieurs grandes catégories.
- **Arithmétiques**
  - Addition, multiplication ...
- **Logiques**
  - Et, ou, xor ...
- **Transfert de données**
  - Mémoire vers registre, registre vers registre, gestion de la pile...
- **Décalage et rotations de bits**
- **Instructions de contrôle**
  - Sauts conditionnels, appels de procédures ...

Opérations :

- Implicitement, selon l'opération, des registres peuvent être modifiés ou utilisés pour stocker le résultat.

- Exemple :

ADD AX, BX ; AX = AX + BX

; modifie les flags selon résultat

; (débordement, résultat nul ...)

Opérations :

- Opérations de Soustraction SUB :

SUB Dest, Source

Opérations :

- Opérations de Négation NEG :

Inverse le signe d'une valeur en convertissant le nombre dans son complément à deux.

NEG reg

NEG mem

Opérations logiques

- Opérations ET Logique :

AND destination, Source

Les formats d'opérande autorisés sont les suivants :

AND reg, reg
AND reg, mem
AND reg, imm ; valeur immédiate sur 8, 16 ou 32 bits.
AND mem, reg
AND mem, mem

Les opérandes 8,16 ou 32 bits doivent être de même taille.

Opérations logiques

- Opérations OR Logique :

OR destination, Source

Les formats d'opérande autorisés sont les suivants :

OR reg, reg
OR reg, mem
OR reg, imm
OR mem, reg
OR mem, mem

Les opérandes 8,16 ou 32 bits doivent être de même taille.

Opérations logiques

- Opérations XOR Logique :

XOR destination, Source

Les mêmes règles applicables sur AND et OR.

- Opération d'inversion NOT

Inverse la valeur de tous les bits de l'opérande.

NOT reg

NOT mem Exemple

Opérations logiques

- Opérations CMP :

CMP destination, Source

Permet de faire une comparaison en réalisant une soustraction implicite entre l'opérande source et l'opérande destination. Aucun des deux opérandes n'est modifié. **Cette instruction modifie les drapeaux.**

CMP accepte les mêmes combinaisons d'opérande que l'instruction AND.

Opérations :

- Instruction INC et DEC :

L'instruction d'incrémentation INC et de décrémentation DEC permettent respectivement d'ajouter 1 et de soustraire 1 à un opérande unique.

Syntaxe : **INC reg/mem**

DEC reg/mem

Drapeaux affectés par les opérations arithmétiques :

- ***Registre d'états (EFLAGS) :***
  - **Registre sur 16 bits voire 32 sur 80386 et les suivants**
  - **Un bit est un drapeau**
    - **Propriété vérifiée (=1) ou pas (=0) après l'appel d'une opération**
  - **Certains bits n'ont aucune signification**

➢ Principaux drapeaux (flags) :

CF (bit 0) : opération génère une retenue

Débordement si entiers non signés

ZF (bit 6) : résultat de l'opération est 0

SF (bit 7) : signe du résultat de l'opération (0=positif,1=négatif)

OF (bit 11) : si opération génère un débordement de capacité (overflow).

Gère les débordements en entiers signés

Transferts de données :

- **MOV Dest, Source**

Réalise : Dest ← Source

- Une restriction est que les deux opérandes ne peuvent pas être tous deux des opérandes mémoire.
- Les opérandes doivent avoir la même taille
- Les registres CS, EIP et IP ne peuvent pas être des opérandes destination.

Transferts de données :

- Copies de données entre mémoire centrale et registres.
- Plusieurs combinaisons à l'exception de celles qui concernent les registres de segment :
  - MOV registre, mémoire
  - MOV mémoire, registre
  - MOV registre, registre
  - MOV mémoire, immédiate
  - MOV registre, immédiate
- En mode réel seulement (on peut utiliser les registres de segment)
  - Mov registre/mémoire, registre segment
  - Mov registre segment, registre/mémoire

Transfert mémoire vers mémoire :

Vous ne pouvez pas utiliser une seule instruction MOV

pour transférer une donnée d'un point à un autre en mémoire,

Solution : Vous devez passer par un registre.

Exemple :

- **mov al, A1** ; Copie l'octet situe en A1 dans **AL**
- **mov eax, offset A1** ; EAX = adresse de l'octet en A1
- **mov A1, ah** ; Copie AH dans l'octet en A1
- **mov eax, A6** ; copie le double mot en A6 dans **EAX**

Transferts de données :

Movzx (entiers non signés) : copie avec extension de zéro

permet de copier le contenu d'un opérande source dans un opérande de destination en étendant le zéro de la valeur vers 16 ou 32 bits.

0	10001111	Source
00000000	10001111	Destination

Exemple :

Soit la déclaration suivante :

val1 WORD 0A9EBh

- **On veut copier val1 dans EAX**
  - **Solution simple**

    Mov EAX, 0

    Mov AX, val1
  - **Utiliser Movzx**

    Movzx EAX, val1

- **Dans les deux cas le contenu de EAX sera 0000A9EB**

Trois variantes :

Movzx, reg32, reg/m8
Movzx reg32, reg/m16
Movzx reg16, reg/m8

Exemple

Transferts de données :

Movsx(entiers signés) :

Copie le contenu d'un opérande source dans un opérande de destination en faisant une extension de signe de la valeur vers 16 ou 32 bits.

1	10001111	Source
111111111	10001111	Destination

Exemple :

Soit la déclaration suivante :

val2 SWORD 0A6EB

- **On veut copier val2 dans EAX**
  - **Solution simple**

    Mov eax, 0FFFFFFFFh

    Mov ax, val2
  - **Utiliser Movsx**

    Movsx eax, val2

- **Dans les deux cas le contenu de eax sera FFFFA6EB**

Trois variantes :

Movsx reg32, reg/m8
Movsx reg32, reg/m16
Movsx reg16, reg/m8

Transferts de données :

- **XCHG**

Permet de permuter le contenu de deux opérandes.

- **Trois variantes possibles :**
  - XCHG reg, reg
  - XCHG reg, mem
  - XCHG mem, reg **exemple**

- **Permutation de valeurs de deux opérandes donne1 et donne2 de type Word :**
  - Mov ax, donne1
  - Xchg ax, donne2
  - Mov donne1, ax

Multiplication non signée :

MUL source

MUL multiplie un opérande non signée (8, 16, 32 bits) par le contenu de AL, AX ou EAX.

- **Multiplication 8-bits, MUL r/m8**
  - la source et multipliée par le contenu de AL
  - résultat sur 16-bit dans AX.

Multiplication 16-bit, MUL r/m16

- la source et multipliée par le contenu de AX
- résultat 32-bit dans DX:AX.

➢ Multiplication 32-bit, MUL r/m32

- ➢ la source et multipliée par le contenu de EAX
- ➢ résultat 64-bit dans EDX:EAX

Multiplication signée :

- **IMUL**

L'instruction IMUL a les mêmes formats que MUL, la seule différence est la préservation du signe du produit (dans les deux cotés de AX : AH et AL) : (étendre le signe de AX dans DX et de EAX dans EDX).

Division entière non signée :

DIV *source (diviseur)*

Effectue la division entre entiers non signés sur 8, 16 et 32 bits. Elle n'attend que le diviseur (reg / mem)

- **division 8-bits, diviseur r/m8**
  - **AX est divisé par *source*.**
  - **Le quotient est stocké dans AL et le reste dans AH**

Exemple :

Division non signée sur huit bits (83h / 2), qui produit un quotient de 41h et un reste de 1 :

```
mov ax, 0083h     ; dividende
mov bl, 2         ; diviseur
div bl            ; AL = 41h, AH = 01h
```

Division entière non signée :

- **division 16-bits, diviseur r/m16**
  - **DX:AX est divisé par *source*.**
  - **Le quotient est stocké dans AX et le reste dans DX**

Division entière non signée :

- **division 32-bits, diviseur r/m32**
  - **EDX:EAX est divisé par *source*,**
  - **le quotient est stocké dans EAX et le reste dans EDX.**

Division entière signée :

IDIV source

- L'instruction IDIV fonctionne de la même façon que DIV.

- Si le quotient est trop grand pour tenir dans son registre ou que le diviseur vaut 0, le programme est interrompu et se termine.

Une erreur courante est d'oublier d'initialiser DX ou EDX avant la division.

Instructions de contrôle :

- **Par défaut, c'est une exécution séquentielle des instructions**

- **Possibilité de sauter à d'autres endroits du programme de 2 façons :**
  - **Saut à une adresse du programme**
  - **Saut conditionnel à une adresse du programme**

- **Pas de structures de contrôle plus évoluées**
  - **Pas de boucles**
  - **Pas de *if then else* ou de *switch***

Saut non conditionnel à une adresse dans le programme

- **En utilisant des labels de code**
  - On peut placer des labels pour marquer un endroit de la séquence d'instructions.
  - **Instruction de saut : JMP**

JMP label

Sauts conditionnels à 1 adresse du programme:

- Il y a beaucoup d'instructions de branchement conditionnel différentes.
- Les plus simples se contentent de regarder un drapeau dans le registre EFLAGS pour déterminer si elles doivent sauter ou non.

- **D’autres instructions déterminent si elles doivent sauter ou non en fonction d’une comparaison à l’aide de** cmp *vleft vrigtht.*

- **Les deux transparents suivants donnent la liste de ces instructions.**

- **D'autres sauts conditionnels sont fondés sur des comparaisons d'entiers signés et non signés.**

Sauts utilisant les drapeaux

- JZ saut si Zero càd si ZF = 1
- JNZ saut si non Zero càd si ZF = 0
- JO saut si Overflow càd OF = 1
- JNO saut si non Overflow càd OF = 0
- JS saut si Sign càd SF =1
- JNS saut si Non Sign càd SF = 0
- JC saut si Carry càd CF = 1
- JNC saut si Non Carry càd CF = 0
- JP saut si Parity càd PF = 1
- JNP saut si Non Parity càd PF = 0

Saut après une comparaison

- **Comparaison signée**
- JE saute si vleft = vright (G Greater, E Equal, L Less, N Not, B Below, A Above(>))
- JNE saute si vleft < > vright
- JL, JNGE saute si vleft < vright
- JLE, JNG saute si vleft <= vright
- JG, JNLE saute si vleft > vright
- JGE, JNL saute si vleft >= vright
- **Comparaison non signé**
- JE saute si vleft = vright
- JNE saute si vleft < > vright
- JB, JNAE saute si vleft < vright
- JBE, JNA saute si vleft <= vright
- JA, JNBE saute si vleft > vright
- JAE, JNB saute si vleft >= vright

Exemple d'utilisation des Sauts conditionnels :

- On peut avec des sauts conditionnels reproduire des instructions de contrôle de plus haut niveau.

Exemple :

If ... then ... else

if (AX = 3)

then BX = 5 else BX = DX

Exemple d'utilisation des Sauts conditionnels :

```
for (cx=0; cx<5; cx++)

ax = ax + cx
```

instructions de boucle conditionnelles :

- LOOP Décrémente ECX,
  - si ECX <> 0, saute vers l'étiquette

- LOOPE, LOOPZ Décrémente ECX
  - si ECX <> 0 et ZF = 1, saute

- LOOPNE, LOOPNZ Décrémente ECX
  - si ECX <> 0 et ZF = 0, saute

EXEMPLE

Sauts conditionnels

Pour la boucle for précédente, on peut utiliser l'instruction loop comme suit :

for (ecx=5; ecx>0; ecx--)

eax = eax + ecx

Les Structures de Contrôle Standards :

Instructions if : if (condition) then_block;
else else_block;

```
; code pour positionner FLAGS : on peut utiliser jz, jnz, ..

jxx else_block ; choisir xx afin de sauter si la condition est fausse

; code pour then_block

jmp endif

else_block:
; code pour le else_bloc
endif:
```

Exemple :

```
If (op1 == op2)
{
X=1;
Y=2;
}
```

Exemples avec les opérateurs logiques AND et OR.

Les Structures de Contrôle Standards :

Boucles while

```
while( condition ) {
    corps de la boucle;
    }
```

```
while:
; code pour positionner FLAGS selon la condition
jxx endwhile ; choisir xx pour sauter si la condition est fausse
; corps de la boucle
jmp while
endwhile:
```

Exemple :

```
While (val1<val2)
{
val1++;
val2--;
 }
```

Exemple while imbriqué.

Les Structures de Contrôle Standards

Boucles do while

```
do {
    corps de la boucle;
     } while( condition );
```

```
do:
; corps de la boucle
; code pour positionner FLAGS selon la condition
jxx do     ; choisir xx pour se brancher si la condition est vraie
```

Mode d’adressage :

Un mode d’adressage :

est une expression arithmétique conduisant au calcul de l’indice d’une case mémoire logique.

Mode d’adressage :

Nous introduisons ici un nouveau mode d’adressage, *l’adressage indirect*, qui est très utile par exemple pour traiter des tableaux.

L’adressage indirect utilise le registre BX ou autre pour stocker l’adresse d’une donnée.

Mode d'adressage :

En adressage direct, on note l'adresse de la donnée entre crochets :

Mov AX, [130]

De façon similaire, on notera en adressage indirect :

Mov AX, [BX]

Mode d’adressage :

Ici, BX contient l’adressage de la donnée. L’avantage de cette technique est que l’on peut modifier l’adresse en BX, par exemple accéder à la case suivante d’un tableau.

Mode d’adressage :

Avant d’utiliser un adressage indirect, il faut charger BX avec l’adresse d’une donnée. Pour cela, on utilise une nouvelle directive de l’assembleur, offset.

Mode d'adressage :

Utilisation des PTR avec les opérandes indirects :

La taille d'un opérande n'est pas toujours aisée à deviner à partir du contexte d'une instruction.

Par exemple, l'instruction suivante force l'assembleur à générer un message d'erreur :

Inc [esi] ; erreur l'opérande doit avoir une taille définie

Mode d’adressage :

Utilisation des PTR avec les opérandes indirects :

En effet, le programme assembleur ne sait pas si ESI pointe vers un octet, un mot ou un double-mot.

En utilisant l’opérateur PTR, vous levez cette ambiguïté.

Inc BYTE PTR [esi]

Opérandes indexés :

Un opérande indexé **permet** d'ajouter une valeur fixe (constante) à la valeur d'un registre pour obtenir une adresse effective.

Vous pouvez utiliser n'importe lequel des registres à usage général sur 32 bits comme registre d'index.

Vous disposez de 2 formes d'écriture sous MASM :

Contante[reg]
[Constante+reg]

Opérandes indexés :

Dans le premier format, nous combinons le nom d'un élément de donnée à celui d'un registre.

Le nom de cet élément doit correspondre à une constante qui contient l'adresse offset de la variable.

Voici des exemples avec les 2 formats :

Tableau1[esi] [tableau1+esi]
Tableau2[ebx] [tableau1+ebx]

Opérandes indexés :

Exemple :

```
.data

Tableau1 byte 10h,20h,30h

.code

Mov esi, 0

Mov al, [tableau1+esi]; al=10h
```

Opérandes indexés :

Pour accéder au deuxième et au troisième éléments, nous ajoutons un déplacement à la valeur d'un registre.

Exemple :

```
.data
Tableau2 word 1000h,2000h,3000h
.code
Mov esi, offset tableau2
Mov ax,[esi]   ; ax=1000h
Mov ax,[esi+2] ; ax=2000h
Mov ax,[esi+4] ; ax=3000h
```

Opérandes indexés :

Les expressions d'adressage indexé utilisent les registres dits d'index SI et DI (en plus de BX et BP).

Exemple :

- **Indirection registre avec offset**

  mov eax,[ebx + 8] ; adresse = ebx + 8

- **Indirection registre avec offset registre (index)**

  mov [ebx + edi * k],eax ; adresse = ebx + edi * k
  ; k peut être 1, 2, 4 ou 8 seulement.

- Les registres suivants peuvent servir de base :

  eax, ebx, ecx, edx, edi, esi, ebp, esp

Les tableaux :

- Un tableau est un bloc contigu de données en mémoire.
- Tous les éléments de la liste doivent :
  - être du même type
  - et occuper exactement le même nombre d'octets en mémoire.

- **L'adresse de n'importe quel élément peut être calculée en connaissant les trois choses suivantes :**

- **L'adresse du premier élément du tableau**
- **Le nombre d'octets de chaque élément**
- **L'indice de l'élément**

L'indice du premier élément du tableau est 0 (comme en C).

Tableau :

Définition :

- Dans le segment .data
  - utilisez les directives BYTE, WORD, etc..
  - Tab1 WORD 30,10, 23, 44 ;un tableau de 4 mots
  - Tab2 BYTE 30 DUP(0) ; un tableau de 30 octets ; initialisés tous à 0

- Dans le segment .data?
  - Tab3 WORD 20 DUP(?) ; un tableau de 20 mots ;non initialisés

Accès aux éléments de tableaux :

➢ **Pour accéder à un élément d'un tableau, son adresse doit être calculée. Considérons les deux définitions suivantes :**

```
tableau1 BYTE  5, 4, 3, 2, 1  ; tableau d'octets
tableau2 WORD  5, 4, 3, 2, 1 ; tableau de mots
```

➢ Exemple d'accès aux éléments de ces tableaux :

```
mov al, [tableau1]          ; al = tableau1[0]
mov al, [tableau1 + 1]      ; al = tableau1[1]
mov [tableau1 + 2], al      ; tableau1[2] = al
mov ax, [tableau2]          ; ax = tableau2[0]
mov ax, [tableau2 + 2]      ; ax = tableau2[1]
mov [tableau2 + 6], ax      ; tableau2[3] = ax
mov ax, [tableau2 + 1]      ; ax = ??
```

Adressage indirect plus avancé :

- L'adressage indirect est souvent utilisé avec les tableaux. La forme la plus générale d'une référence mémoire indirecte est :

  [reg de base + facteur *reg d'index + constante]

- **reg de base** est un des registres EAX, EBX, ECX, EDX, EBP, ESP, ESI ou EDI.
- **facteur** est 1, 2, 4 ou 8 (s'il vaut 1, le facteur est omis).
- **reg d'index** est un des registres EAX, EBX, ECX, EDX, EBP, ESI, EDI.
- **constante** est une constante 32 bits.

Exemple :

Exemple 1 : Dans cet exemple, le tableau contient 3 octets. En augmentant la valeur ESI, nous pouvons le faire pointer successivement vers chacun des octets dans l'ordre.

Exemple 2 : Copie d'une chaine de caractères.

Exemple 3 : Addition d'entiers sur 32 bits.

Les pointeurs :

Une variable dont la valeur n'est pas utilisée telle quelle mais en tant qu'adresse d'une autre variable est appelée une variable pointeur (ou un pointeur).

Ils sont indispensables pour traiter les tableaux et les structures de données.

Les programmes pour processeur Intel exploitent deux types principaux de pointeurs : NEAR et FAR.

Les pointeurs :

La taille des pointeurs dépend du mode de fonctionnement du processeur :

	Mode 16 bits	Mode 32 bits
Pointeur NEAR	Décalage offset sur 16 bits du début du segment de données	Décalage offset sur 32 bits du début du segment de données
Pointeur FAR	Adresse segment-offset sur 32 bits	Adresse segment-offset sur 48 bits

Les pointeurs :

Les exemples en mode protégé utilisent des pointeurs à accès court (NEAR), ce qui permet de les stocker dans des variables de taille double-mot.

Exemple

L’opérateur Typedef :

Il permet de créer un type spécifique offrant toutes les caractéristiques des types prédéfinis et de définir ainsi de nouvelles variables.

Il est particulièrement approprié à la création de variables pointeurs.

La déclaration suivante crée par exemple un nouveau type de donnée, PBYTE, qui est un pointeur sur un octet :

```
PBYTE TYPEDEF PTR BYTE
```

L’opérateur Typedef :

Normalement, vous placerez ce type de déclaration en début de programme source, avant le segment de données.

Vous pouvez ensuite déclarer des variables du type PBYTE :

```
.data
tabB BYTE 10h, 20h, 30h, 40h
Ptr1 PBYTE ?    ; non initialisé
Ptr2 PBYTE tabB  ; pointe sur tabB
```

L’opérateur Typedef :

Exemple :

Ce programme exploite TYPEDEF pour créer trois types de pointeurs (PBYTE, PWORD, PDWORD). Il crée ensuite des variables pointeurs et les déréférence pendant l’exécution pour lire des données depuis les tableaux correspondants.

Tableaux à Deux Dimensions :

- int tab [3][2];
- Le compilateur C réserverait de la place pour un tableau d'entiers de 6 et placerait les éléments comme suit :

0	1	2	3	4	5
tab[0][0]	tab[0][1]	tab[1][0]	tab[1][1]	tab[2][0]	tab[2][1]

- tab[0][0] est stocké au début du tableau de 6 éléments à une dimension.

➢ En assembleur :

Chaque ligne du tableau à deux dimensions est stockée en mémoire de façon contiguë.

➢ **Le dernier élément d'une ligne est suivi par le premier élément de la suivante.**

Exemple : Pour un tableau 3 lignes 2 colonnes

```
Table byte  10h, 20h
      byte  50h, 40h
      byte  30h, 10h
```

Tableaux à Deux Dimensions :

- Pour l'exemple précédent, tab[i][j] se trouve à l'indice 2i+j.
- Cette formule est généralisée à un tableau de N colonnes pour trouver l'indice de tab[i][j] **N × i + j.**
- Notez que la formule ne dépend pas du nombre de lignes.

La pile :

- La pile système est un tableau à une dimension dont les éléments sont des mots (32 bits).
- Elle est implantée dans la mémoire de l'ordinateur, dans le segment dédié ESP (Segment de la Pile).
- Ce segment est associé au registre de segment SS (Stack Segment).
- Le pointeur de pile ESP contient l'adresse du dernier élément placé sur la pile (empilé).

- **Les opérations de pile sont réalisables avec l'aide d'instructions fournies par la machine. Ce sont :**
  - **PUSH**
  - **POP**

Ces instructions utilisent un pointeur appelé pointeur de pile (ESP) qui contient l'index courant i,

i pointe sur le dernier élément placé sur la pile.

L’opération « empiler » est exécuté par l’instruction PUSH. Sa syntaxe est :

PUSH « source »

Ou « source » peut être :

- Un des registres universels de 32 bits (EAX,EBX, ECX,EDX)
- Un des adresses dédiés à l’adressage (EDI,ESI, EBP, ESP)
- Un registre segment (CS,DS,SS, ES)

- **Un mot-mémoire spécifié à l'aide de différents modes d'adressage**

Le pointeur de pile ESP est décrémenté de 4 avant l'exécution de l'instruction (car on empile 4 octets).

On dit que la pile croit dans le sens des adresses décroissantes.

L’opération « dépiler » est exécutée par l’instruction POP. Sa syntaxe est :

POP « destination »

Ou :

«destination» peut être l’un des registres «source»

L’instruction de dépilage POP commence par copier le contenu de l’ élément de pile actuellement pointé par ESP dans un opérande de destination sur 32 bits.

Puis elle incrémente la valeur de ESP.

Le **MASM** fournit également une instruction **PUSHAD** qui permet de positionner les contenus de tous les registres à usage général sur 32 bits sur la pile pour les sauvegarder dans l'ordre suivant :

EAX, ECX, EDX, EBX, ESP (valeur d'origine), EBP, ESI, et EDI.

L'instruction **POPAD** récupère et restaure les valeurs de tous ces registres dans l'ordre inverse.

PUSHAD et POPAD permettent de sauvegarder

et restaurer simplement l'état des registres

Résumé :

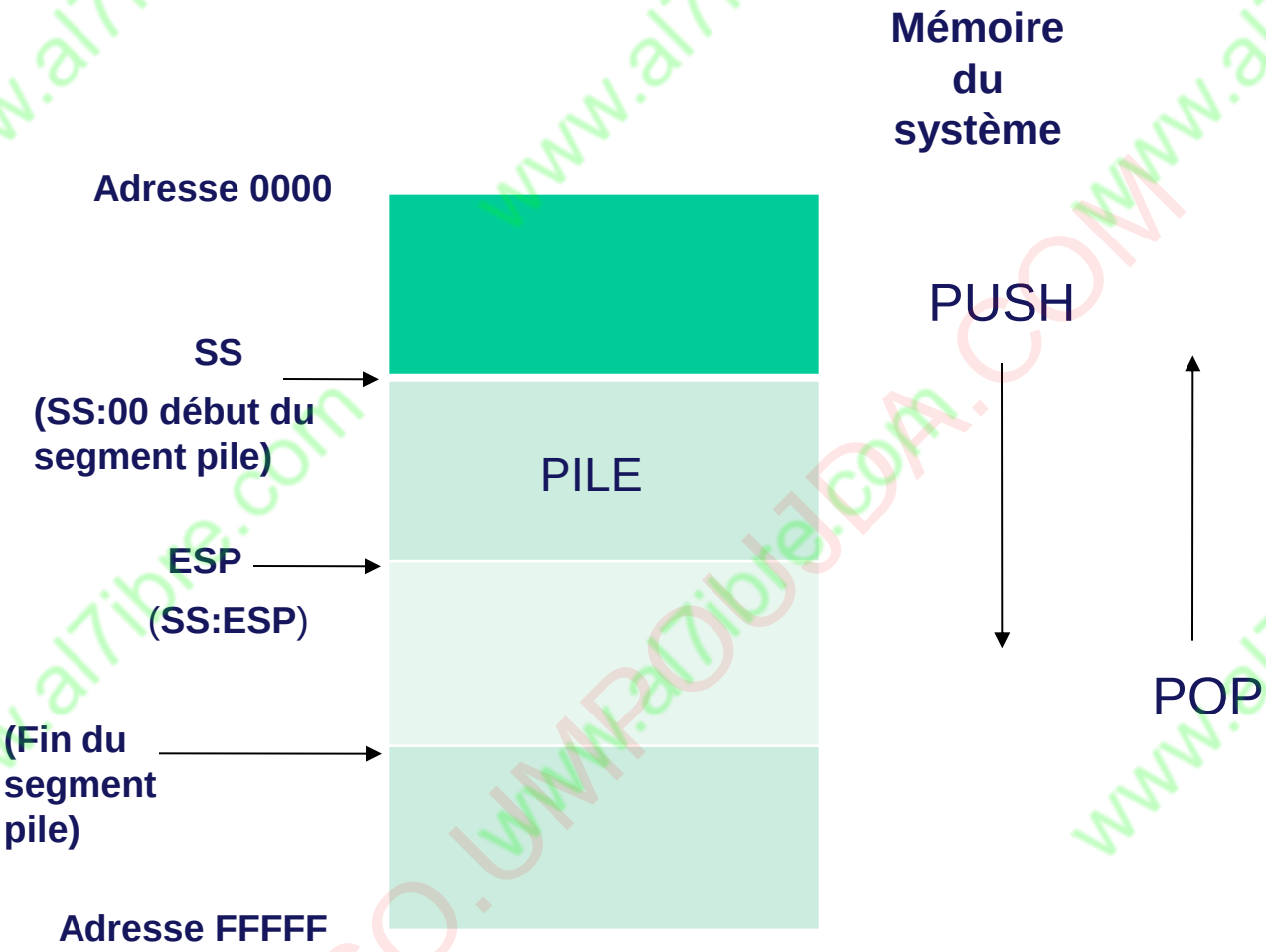
- La pile est une zone de la mémoire organisée sous forme d'une liste Last-In First-Out (LIFO, dernier entré, premier sorti).

- PUSH ajoute des données à la pile et l'instruction POP retire une donnée. La donnée retirée est toujours la dernière donnée ajoutée.

- SS spécifie le segment qui contient la pile (habituellement c'est le même que celui qui contient les données).

- **ESP contient l'adresse de la donnée qui sera retirée de la pile.**

- **Les données ne peuvent être ajoutées que par unités de quatre mots. C'est-à-dire qu'il est impossible de placer deux octets seuls sur la pile.**



Implantation dans la mémoire du segment contenant la pile système

- L'instruction PUSH *source* effectue les opérations suivantes :

  ESP ← ESP - 4

  [ESP] ← valeur de source sur 32 bits.

Source peut être

  - Un des registre généraux(EAX,EBX,ECX,EDX)
  - Un registre d'adressage(EDI,ESI,EBP,ESP)
  - Un mot mémoire spécifié à l'aide des différents modes d'adressage

- L'instruction POP *destination* effectue le travail inverse :

  destination ← [ESP]

  ESP ← ESP + 4

destination peut être

  - Un des registre généraux (EAX,EBX,ECX,EDX)
  - Un registre d'adressage (EDI,ESI,EBP,ESP)
  - Un mot mémoire spécifié à l'aide des différents modes d'adressage

Utilisation de la pile :

La pile est utilisée dans plusieurs domaines très importants de programmation :

- **La pile constitue une zone de stockage temporaire très utile pour les contenus des registres.**

- **Lors de l'exécution d'une instruction d'appel à un sous-programme, le processeur stocke l'adresse de retour d'exécution dans la pile.**

- **Lors de l'appel à une procédure, il est souvent nécessaire de transmettre des valeurs en paramètres d'entrée (des arguments) :**

  Ils peuvent être stockés dans la pile.

- **Enfin, les variables locales d'une procédure sont crées dans la pile et disparaissent en fin de procédure.**

Exemple d'inversion d'une chaine

Les procédures

- Une procédure (sous_programme) est une unité de code indépendante qui peut être utilisée depuis différentes parties du programme.

- Pour les paramètres des procédures et la valeur retournée on utilise deux méthodes:

  - Utiliser des registres

  - Passer par la pile

Les procédures

- **Au niveau élémentaire, une procédure est un bloc contigu d'instructions portant un nom et se terminant par une instruction de retour d'exécution.**

Les procédures :

Déclaration d'une procédure :

Deux directives permettent d'indiquer qu'une séquence d'instructions est une procédure :

- **PROC ;** marque le début de la procédure
- **ENDP ;** marque sa fin

Voici l'utilisation – type :

nom PROC type

.....

RET

nom ENDP

«nom» : est un identificateur accepté par l'assembleur.
Le «type» : informe sur les possibilités d'appel de la routine
L'instruction «RET» indique la fin du traitement effectuée par la procédure.

Appel d'une procédure :

L'instruction d'appel générale est :

CALL Destination

L'exécution de l'instruction d'appel CALL a pour conséquence deux choses :

- *L'adresse de retour à la procédure appelante est mémorisée sur la pile système.*
- *Le contrôle du fonctionnement de la machine est transférée vers la procédure appelée.*

Appel d'une procédure :

Exemple :

Concevez une procédure qui attend deux paramètres d'entrée depuis un programme appelant un pointeur sur un tableau d'entiers sur 32 bits. La routine calcule la somme.

Donnez un programme appelant cette routine.

Appel d'une procédure :

Imbrication d'appels de procédures :

Un appel de procédure imbriqué a lieu lorsqu'une procédure qui fonctionne en tant que sous-programme contient un appel à une autre procédure avant de rendre le contrôle.

Exemple

Communications entre les procédures

En informatique, deux paramètres existent :

Les paramètres formels : servent à la définition de la procédure

Les paramètres actuels : lors de l'appel de la procédure.

Explication : lors de l'appel à la procédure qui calcule la valeur du terme général, la procédure appelante lui communique les valeurs de paramètres actuels.

Remarque :

Le passage des paramètres doit précéder l'appel de la procédure.

Plusieurs méthodes de passage (de communication) de paramètres existent :

- passage par valeur
- passage par référence
- passage par nom

Passage par valeur :

Signifie que les valeurs des paramètres actuels sont communiqués à la procédure appelée.

Passage par référence :

ou passage par *adresse* signifie que les adresses de paramètres actuels de la procédure appelante sont transmises à la procédure Appelée.

Support physique pour le passage de paramètres dans la famille 80x86

Les paramètres actuels peuvent être passés par une zone commune à deux procédures.
La procédure appelante y dépose les paramètres actuels, la procédure appelée y les lit.

Cette zone peut être réalisée à l'aide :

- **de registres**
- **de la mémoire utilisateur**
- **de la pile système**

- Le passage des paramètres par registres est très rapide, mais il permet d'en transmettre peu :

  - Registres peu nombreux et doit toujours utiliser les mêmes à chaque appel donc assez contraignant

- Les autres méthodes (mémoire) ont l'avantage de permettre de transmettre un nombre infini de paramètres, mais sont plus lentes.

- **Pour l'architecture 8086, le mécanisme permet de réaliser facilement le passage de paramètres par la pile :**

- On empile les paramètres de la procédure
- On appelle la procédure
- On dépile les paramètres et résultats
- Plus compliqué mais plus général

Passage de paramètres actuels par la pile – Utilisation du pointeur EBP

A l'exécution d'une procédure, son environnement peut être repéré sur la pile à l'aide du pointeur EBP.

Cette organisation de la pile système permet d'adresser :

- **les paramètres actuels**
- **les variables locales**

Par rapport au contenu du EBP.

En effet,

- L’adresse de la iéme variable locale est :

[EBP – 4 * i];

- L’adresse du iéme paramètre passé est :

[EBP + 4 * i + longueur de l’adresse de retour];

où la longueur de l’adresse retour est **4** octets.

Passage des paramètres par pile

Coté appelé

```
;Empiler ebp
Mov ebp, esp ; sauver esp actuel dans ebp
;empiler les variables locales et les paramètres
;empiler les valeurs des registres que l'appelé va utiliser
; traitement sur les paramètres et les variables
   locales qui sont accessibles via la pile
;dépiler les valeurs des registres utilisés
;libérer la place des variables locales sur la pile
;dépiler ebp
ret
```

Passage des paramètres par pile

Coté appelant

....

push argument 1

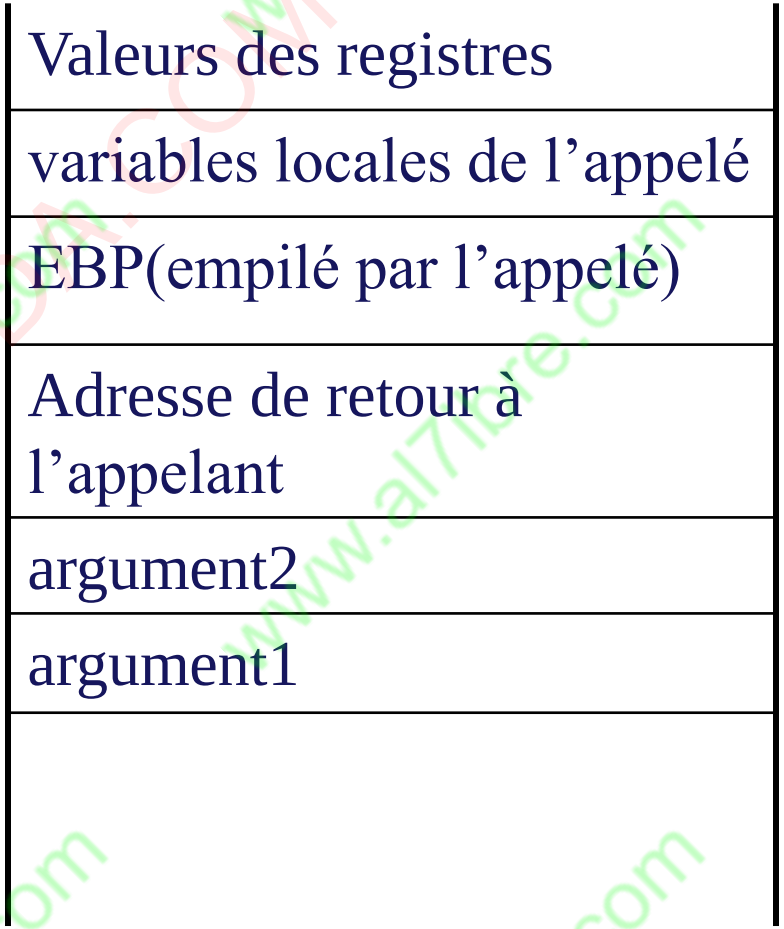
push argument 2

call nom

pop argument 2

pop argument 1

....



```
Include Irvine32.inc
.data
prompt1 byte "donner un élément:",0
prompt2 byte "donner un autre:",0
outmsg3 byte "la somme est : ",0
.code
main proc
   pushad
   mov      edx, offset prompt1
   call     writestring
   call     readint
   mov      ebx, eax
   mov      edx, offset prompt2
   call     writestring
   call     readint
   Call Som
   popad
   exit
main ENDP
```

```
Som PROC
 add eax, ebx
 mov  edx, offset outmsg3
 call  writestring
 call  writeint
 ret
Som ENDP
END main
```

Exemple utilisant les registres

```
include Irvine32.inc
.data
prompt1 byte "donner un element:",0
prompt2 byte "donner un autre:",0
outmsg3 byte "la somme est:",0
.code
main proc
  pushad
  mov     edx, offset prompt1
  call    writestring
  call    readint
  push eax
  mov     edx, offset prompt2
  call    writestring
  call    readint
  push eax
  Call som
  pop eax
  pop eax
  popad  exit; main ENDP
```

```
som PROC
    push ebp
    mov ebp, esp
    push eax
    push edx
    mov eax,[ebp+8]
    add eax,[ebp+12]
    mov  edx,offset outmsg3
    call  writestring
    call  writeint
    pop edx
    pop eax
    pop ebp
    ret
som ENDP
END main
```

Exemple utilisant la pile

Interruptions

Les interruptions permettent au matériel de communiquer avec le processeur.

Les échanges entre le processeur et l'extérieur se faisait toujours à l'initiative du processeur.

Une interruption est signalée au processeur par un signal électrique sur une borne spéciale.

Lors de la réception de ce signal, le processeur traite l'interruption dés la fin de l'instruction qu'il était en train d'exécuter.

Le traitement de l'interruption consiste soit :

- À l'ignorer et passer normalement à l'instruction suivante (interruptions masquables) en cas de traitements urgents. Lorsque le traitement est terminé, le processeur démasque les interruptions et les prend alors en compte.

- à exécuter un traitant d'interruption. Un traitant d'interruption est un programme qui est appelé automatiquement lorsqu'une interruption survient.

Interruptions matérielles

Signaux d'interruption :

Les processeurs de la famille 8086 possèdent trois bornes pour gérer les interruptions : NMI, INTR et INTA.

NMI (Non Masquable Interrupt) : est utilisé pour envoyer au processeur une interruption non masquable. Le processeur ne peut pas ignorer ce signal, et va exécuter le traitant donné par le vecteur 02H. Ce signal est normalement utilisé pour détecter des erreurs matérielles (mémoire principale défaillante par ex).

INTR (Interrupt Request) : Demande d'interruption masquable. Utilisée pour indiquer au MPU l'arrivée d'une interruption. Voir schéma.

Interruptions matérielles

Signaux d'interruption :

INTA (Interrupt Acknowledge) : cette borne est mise à 0 lorsque le processeur traite effectivement l'interruption signalée par INTR (càd qu'elle n'est plus masquable).

Interruptions matérielles

Indicateur IF :

À un instant donné, les interruptions sont soit masquées soit autorisées, suivant l'état d'un indicateur spécial du registre d'état, IF (Interrupt Flag).
-Si IF = 1, le processeur accepte les demandes d'interruptions masquables, càd qu'il les traite immédiatement.

-Si IF=0, le processeur ignore ces interruptions.

L'état de l'indicateur IF peut être modifie à l'aide de deux instructions, CLI (Clear IF, mettre IF à 0), et STI (SeT IF, mettre IF à 1).

Interruptions matérielles

Contrôleurs d'interruption

L'ordinateur est relié a plusieurs périphériques, mais nous venons de voir qu'il n'y avait qu'un seul signal de demande d'interruption, INTR.

Le contrôleur d'interruption est un circuit spécial, extérieur au processeur, dont le rôle est de distribuer et de remettre en attente les demandes d'interruptions provenant des différents périphériques.

Interruptions matérielles

Déroulement d'une interruption externe masquable :

Les différents événements liés à la réception d'une interruption masquable :

1. Un signal INT est émis par un périphérique (ou plutôt par l'interface gérant celui-çi).

2. Le contrôleur d'interruption reçoit ce signal sur une de ses bornes IRQi. Dés que cela est possible (suivant les autres interruptions en attente de traitement), le contrôleur envoie un signal sur sa borne INT.

Interruptions matérielles

Déroulement d'une interruption externe masquable :

3. Le MPU prend en compte le signal sur la borne INTR après avoir achevé l'exécution de l'instruction en cours (ce qui peut prendre qlq cycles d'horloge). Si l'indicateur IF = 0, le signal est ignoré, sinon, la demande d'interruption est acceptée.

4. Si la demande est acceptée, le MPU met sa sortie INTA au niveau 0 pendant 2 cycles d'horloge, pour indiquer au contrôleur qu'il prend en compte sa demande.

Interruptions matérielles

Déroulement d'une interruption externe masquable :

5. En réponse, le contrôleur d'interruption place le numéro de l'interruption associé à la borne IRQi sur le bus de données.

6. Le processeur lit le numéro de l'interruption sur le bus de données et l'utilise pour trouver le vecteur d'interruption. Ensuite, tout se passe comme pour un appel système (interruption logicielle), c'est-à-dire que le processeur effectue les traitements déjà vu en cours.

Interruptions matérielles

Déroulement d'une interruption externe masquable :

7. La procédure traitant l'interruption se déroule. Pendant ce temps, les interruptions sont masquées (IF=0). Si le traitement est long, on peut dans certains cas ré-autoriser les interruptions avec l'instruction STI.

Interruptions matérielles

Déroulement d'une interruption externe masquable :

8. La procédure se termine par l'instruction IRET, qui restaure CS, IP et les indicateurs à partir de la pile, ce qui permet de reprendre le programme qui avait été interrompu.

Interruptions matérielles

Exemple : Gestion de l'heure sur PC

L'horloge d'un PC est un circuit électronique cadencé par un oscillateur à quartz (comme une montre ordinaire).

L'horloge envoie une interruption matérielle au processeur toutes les 0,055 secondes (soit 18,2 fois par seconde). Le vecteur correspondant est le numéro 08H.

Pour gérer l'heure, le BIOS installe un traitant pour l'interruption 08H. Ce traitant incrémente simplement un compteur, nombre entier codé sur 32 bits et toujours rangé à l'adresse 0040 : 006C en mémoire principale.

Ainsi si un programme désire connaitre l'heure, il lui suffit de lire cet emplacement mémoire.

Interruptions logicielles

Une interruption logicielle correspond à un appel fait à une routine du système d'exploitation.

L'instruction Intel **INT** appelle la routine gestionnaire d'interruptions correspondant au numéro fourni en paramètre.

Syntaxe générale : **INT numero**

Le paramètre *numero* est une valeur entière entre 0 et FF en hexadécimal.

Les interruptions les plus utilisées :

Les routines qui correspondent aux gestionnaires d'interruption peuvent faire partie du BIOS ou de DOS. Voici les interruptions les plus utilisées :

- Services vidéo INT 16h : Routine de lecture du clavier et de test de son état.

- Services d'impression INT 17h : Routine d'impression et de test de l'état de l'imprimante

Routine d’horloge INT 1Ah : permettant de connaitre le nombre de top d’horloge depuis le démarrage de la machine,

Exemple :

INT 21h :

Description : écrit un seul caractère vers l'imprimante

Exemple d'appel : mov dl "Z "

INT 21h ; appel MS-DOS

MS-DOS reste en attente jusqu'à ce que l'imprimante soit prête à imprimer le caractère.