HEB
ESI

Techniques des µ-processeurs

Techniques des µ-processeurs et logiciels de contrôle

Version Janvier 2002

Avertissement

Ces notes sont une copie de transparents. Bien que ces transparents soient commentés, cette source d 'information est, sans doute, bien incomplète.

Nous vous conseillons de compléter ces notes des commentaires donnés au cours. C 'est pourquoi nous avons choisi ce format d 'impression.

Remerciements

Ce ' syllabus ' a été rédigé sur base des transparents réalisés par **Jean-Claude Jaumain** et **Jean-Marie Van Loock**. Qu 'ils en soient ici remerciés.

Nous avons augmentés et détaillés les anciens transparents. Merci à **Pandélis Matsos** pour ses remarques et ajouts.

PbT

HEB
ESI μ

Techniques des μ-processeurs

Pierre Bettens (PBT)

pbettens@heb.be
http://esi.pit-it.com

Copyright

E-mail : pbettens@heb.be

HEB
ESI

Techniques des µ-processeurs

Pandélis Matsos (PMA)
pmatsos@hotmail.com
Martin Van Aken (VAK)
vanakenm@hotmail.com
René Pailloucq (RPA)
pailloucq@skynet.be
Jean-Claude Jaumain (JCJ)
jcjaumain@heb.be
Amine Hallal (HAL)
ahallal@ulb.ac.be

HEB ESI µ

Table des matières

Remarque

Ce cours se veut être tout autant une introduction aux µ-processeurs qu'un préalable au cours d 'assembleur du second semestre. Les laboratoires de µ-processeur se dérouleront en langage assembleur. En effet, c 'est la seule manière de voir et comprendre la gestion des registres, l 'organisation de la mémoire …

Voilà pourquoi vous trouverez assez vite une série d 'instructions assembleur.

HEB ESI µ

Table des matières

HEB ESI μ

Table des matières

- Langage Assembleur
  - Les branchements
  - Les instructions de répétitions
  - Les instructions conditionnelles
- La pile du 8086
- Segmentation de la mémoire
- Mode d ’adressage du 8086

HEB ESI µ

Organisation du cours

- MIC : Cours théorique de 12,5 h en groupe de 60
- LMI : Laboratoire de 12,5 h en groupe de 15. Commence en décalage par rapport au cours.
- ASM - LAS : Cours (25h) et labo (25h) de langage assembleur au second semestre.

Techniques des µ-processeurs - 1ière - PBT / PMA

Remarque

Des liens avec d 'autres cours peuvent être faits:

• Le cours d '*Architecture des ordinateurs* explorera dans le détail les notions plus ' électroniques ' des µ-processeurs.

• Les cours de *C* et de *Logique* permettront les parallèles avec les structures de contrôle standard des différents langages de programmation.

HEB
ESI μ

Organisation du cours

Support de cours

ASSEMBLEUR Théorie, pratique et exercices.
Ed Marabout, Bernard Fabrot
isbn 2-501-02186-X

- Transparents ‘ electroniques ’
- Basés sur le syllabus … qui est une copie des transparents papier
- *Conseil : Prendre des notes au vol.*
- Livre

Techniques des µ-processeurs - 1ière - PBT / PMA

Références

Assembleur. Théorie, pratique et exercices. **Ed. Marabout**, *Bernard Fabrot*

Ce livre fait partie d ’une trilogie chez Marabout. L ’un est une simple énumération des instructions assembleur (un mémento) , l ’autre un cours concentré et le dernier, le plus ‘ épais ’ un cours plus complet.

Conseil

Sans vouloir empiéter sur un cours de méthodologie, je vous conseille la prise de notes. Celle-ci permet d ’une part de pouvoir avoir ‘ vos ’ notes et d ’autre part de tenir votre attention (bref de rester éveillé) … ceci même si vous décider de jeter vos feuilles à l ’issue du cours pour ne garder que le syllabus.

HEB
ESI μ

Présentation

Microprocesseurs …

- Nous sommes au cœur de la machine,
- Au niveau ‘ bus - bit - registres ’ … au niveau électronique
- L ’objectif est de comprendre comment cela fonctionne, rien n’est magique. Posez-vous toujours les questions; Pourquoi ? Comment ?

Techniques des µ-processeurs - 1ière - PBT / PMA

Remarque

L ’ordinateur et à fortiori son (ou ses) processeurs sont , en finalité, des circuits électroniques. Il ne faut donc pas les supposer trop intelligents ! Posez-vous donc la question : « Puis-je concevoir qu ’un ou plusieurs circuits électriques puissent faire ça ? »

Présentation

Qu ’est-ce qu ’un µ-processeur ?

- Circuit intégré remplissant les fonctions d’unité centrale de traitement d’un ordinateur
- Né en 1971 : explosion de l’informatique
- Domaines d’application : ordinateurs, robots, automobile, jouets, jeux vidéo, électro-ménager, informatique embarquée, …

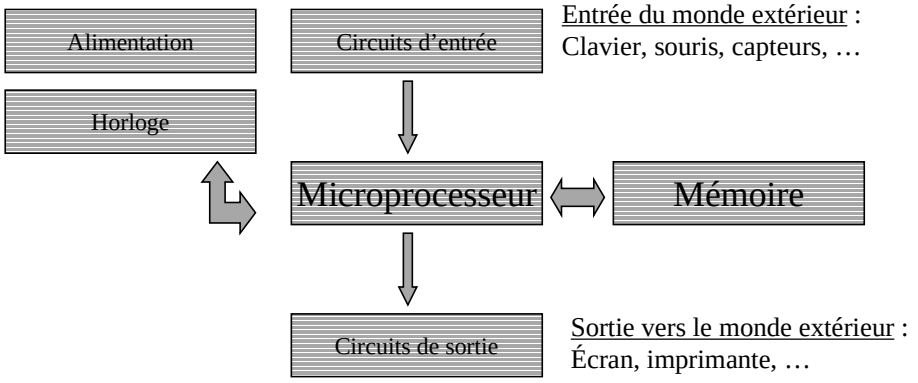
HEB
ESI μ

Présentation

- Le µp est l ’élément central de l ’ordinateur
- Il est implémenté au moyen de circuits logiques intégrés (voir Cours de Structures des ordinateurs)
- µp travaille sur des données, il exécute une série d ’instructions machines

HEB
ESI µ

Présentation



Techniques des µ-processeurs - 1ière - PBT / PMA

Description

Un *µ-processeur* permet de faire des calculs. C 'est un ensemble de circuits électroniques qui doivent être *alimentés*.

A chaque ' top *horloge* ', des portes s 'ouvrent afin de laisser circuler de l 'information … qui se trouve dans la *mémoire*.

De l 'information supplémentaire peut-être fournie via un *circuit d 'entrée* tandis que ce système peut communiquer avec l 'extérieur via un *circuit de sortie*.

HEB
ESI

µ Introduction - Vocabulaire

- Bit : information élémentaire 0 ou 1
- Byte / Octet : 8 bits
- Word : 2 bytes - 16 bits (architecture 16 bits)
- Registre : espace destiné à recevoir de l ’information, sa taille est variable.
- Bus
- Binaire - Octal - Hexadécimal

Techniques des µ-processeurs - 1ière - PBT / PMA

Remarque … pour aller (un peu) plus loin

Au sujet de la notion de **word**.

Un word représente la taille de l ’information pouvant être lue en une seule étape, celle-ci dépend de l ’architecture ! Nous parlons principalement du µ-processeurs Intel 8086 qui a une architecture 16 bits. Dans ce cas un mot à une taille de 16 bits.

Pour information, les processeurs actuels (Pentium) ont une architectures 32 bits, la taille d ’un mot est donc de 32 bits soit 4 bytes.

HEB ESI μ

Introduction - Vocabulaire

- Signaux logiques : 2 niveaux de tension (ex. 0 Volts et 3,5 Volts)
- Circuits électroniques élémentaires : diode, transistor
- Circuits logiques de base : portes logiques
- Circuits intégrés
- Mémoire binaire : bistable

Techniques des µ-processeurs - 1ière - PBT / PMA

Remarques

Référez-vous toujours à votre cours d ’Architecture ...

HEB ESI µ

Introduction - Vocabulaire

Lien entre registre et bus

- Registre : Ensemble ordonné de bits mémorisés. Le nombre de bits dans un registre définit sa longueur.
- Bus : Lignes de communication connectant différents registres.

Le transfert d ’information entre registre se fait via les Bus

HEB
ESI μ

Système à µp - le 8086

Un tel système est constitué des elts suivants :

- MPU : Micro Processor Unit, boîtier de 40 à 132 broches, contient une horloge.
- Bus : bus de données, d ’adresses et de commandes
- Mémoire : ROM et RAM

Techniques des µ-processeurs - 1ière - PBT / PMA

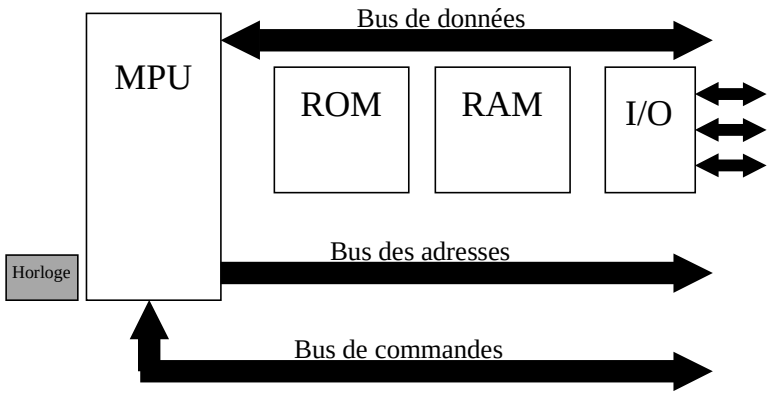
Remarques

Nous étudions ici le µ-processeurs **Intel 8086** … finalement bien loin du Pentium actuel.

Le boitier d ’un pentium est composé d’un nombre de broches de l ’ordre des 400 et ses bus sont des bus de 32 bits voire 64 bits.

HEB
ESI µ

Système à µp - le 8086



Techniques des µ-processeurs - 1ière - PBT / PMA

Remarques

C ’est la taille du **bus de données** (son nombre de fils) qui détermine son architecture c ’est-à-dire la taille d ’un word. Dans le cas du 8086, c ’est 16 bits. Cela veut dire que l ’on peut écrire/lire un mot de 16 bits en une seule opération.

La taille du **bus d ’adresse**, quant à elle, détermine le nombre de case mémoire que l’on peut adresser. Adresser veut dire donner un nom (un numéro) et donc pouvoir y accéder. A l ’époque, un bus de 20 bits semblait être une limite amplement suffisante. Les ingénieurs de l ’époque ne pensait pas que cette limite serait dépassée si vite.

Le **bus de commande** est représenté en une seule ‘ ligne ’ il doit cependant être vu comme une série de fils n ’étant pas spécialement contigus. Chaque fils est branché à un endroit sur la carte en fonction de son rôle.

Système à µp - le 8086

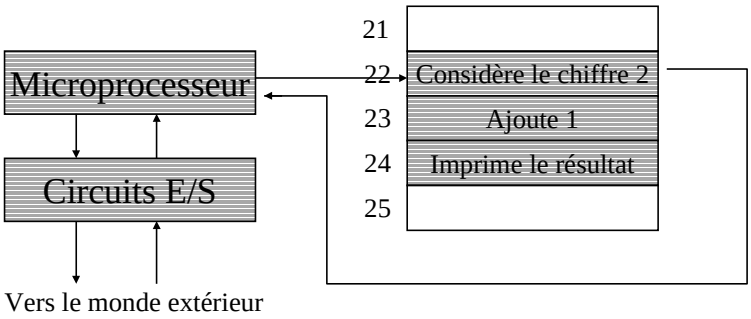
Comment ça fonctionne ? … la base ...

- Le MPU place l ’adresse de l ’instruction à exécuter sur le bus d ’adresse
- Activation d ’un signal sur le bus de commande pour prévenir la mémoire
- La mémoire place le mot contenant l ’instruction sur le bus de données et prévient le µp via le bus de commande
- Le µp lit l ’instruction sur le bus de données et la place dans un registre interne
- …

HEB
ESI μ

Système à µp - le 8086

- Recherche d ’une instruction en mémoire



Techniques des µ-processeurs - 1ière - PBT / PMA

Remarque

Ce schéma est une vue simplifiée permettant simplement de faire un lien entre le µ-processeur et la mémoire.

HEB ESI μ

Système à μp - le 8086

Eléments du 8086

- Registre d ’instruction et compteur d ’instruction
- Additionneur
- Décodeur d ’instruction et séquenceur
- Unité arithmétique et logique
- Jeux d ’instructions
- Pile, accumulateur, registres

HEB ESI µ

Système à µp - le 8086
Registre d ’instruction

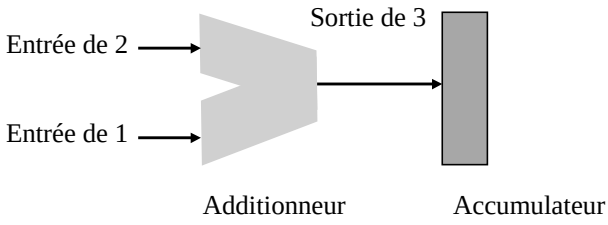
Registre d’instruction et compteur d’instruction : l’instruction à exécuter doit se trouver dans une mémoire du µp

- Compteur d’instruction : adresse mémoire de l’instruction; incrémentation automatique
- Registre d’instruction : reçoit l’instruction depuis la mémoire

HEB
ESI

Système à µp - le 8086
Additionneur - Accumulateur

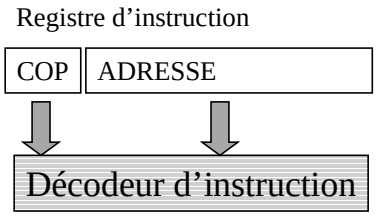
- Additionneur : circuit interne du µp
- Registre Accumulateur : reçoit le résultat de l'additionneur



HEB ESI μ

Système à µp - le 8086
Décodeur d 'instructions

- Registre d'instruction : passe l'instruction au décodeur d 'instruction
- Décodeur d'instruction : dissocie le Code Opératoire de l'adresse des opérandes et active les circuits du processeur chargés de l'exécuter



Techniques des µ-processeurs - 1ière - PBT / PMA

Remarque … pour aller (un peu) plus loin.

Nous ne voulons nullement, dans ce schéma, induire le fait qu'un code opératoire a une taille d 'un byte.

Le décodeur d 'instruction va, dans un premier temps, lire un byte et déterminer si ce byte représente un code opératoire à part entière ou bien s 'il doit lire un byte supplémentaire (le suivant) pour déterminer quelle instruction il doit exécuter.

Ensuite, le décodeur d 'instructions s 'occupera des opérandes éventuels de l 'instruction.

Par exemple

L 'instruction **SUB AX,*op*** peut avoir comme premier byte le nombre 2B ou 2D suivant le type de l 'opérande *op*.

Assembleur	*Code Hexadécimal*
SUB AX,3A91h	2D913Ah
SUB AX,BX	2BC3

Système à µp - le 8086

UAL : Unité Arithmétique et Logique -> exécute des opérations arithmétiques (+, -, *, /) et logiques (ET, OU, NON,…)

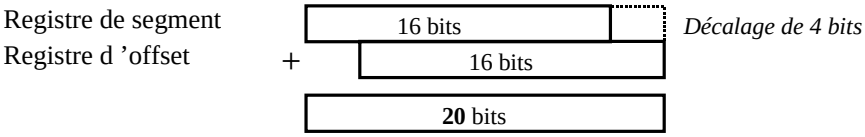
Séquenceur : organe de commande qui ordonne en séquence le fonctionnement du µp au pas de l'horloge

- Recherche d'une instruction en mémoire
- Décodage de cette instruction
- Exécution de ce qu'elle indique

Système à µp - le 8086

Adresse Offset Adder

Sert au calcul des adresses de 20 bits



On obtient une adresse de 20 bits ...

Remarques

Ce système de calcul d 'adresses est propre aux µ-processeurs Intel. D 'autres manières de fonctionner existent ; Motorolla et sa série 68000 en sont un exemple.

HEB
ESI μ

Système à µp - le 8086

Pourquoi des adresses de 20 bits ?

La RAM est adressée par 20 bits

- 2^{20} = 1 048 576 possibilités
- 1 048 576 adresses possibles
- On peut donc adresse une RAM de 1 048 576 bytes.
- 1 048 576 bytes = 1 Mb

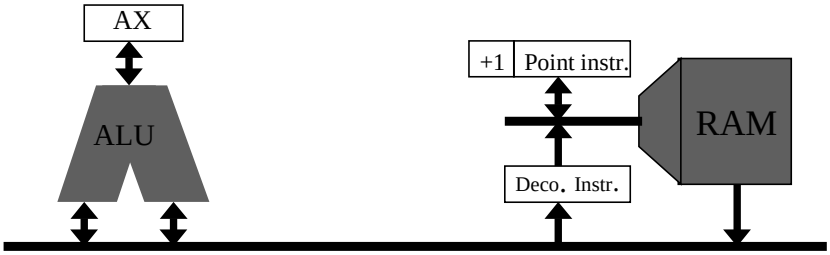
Techniques des µ-processeurs - 1ière - PBT / PMA

Remarque

En fait Intel propose des adresses de 20 bits (dans le cas de notre 8086) car, à l ’époque, une mémoire d ’une taille de 1 Mb semblait irréalisable. Donc les ingénieurs de Intel pensait avoir choisi une limite supérieure suffisante … ce qui ne fut malheureusement pas le cas !

HEB
ESI μ

Machine simplifiée



- AX : registre « accumulateur » de 16 bits
- RAM adressée par bus de 20 bits soit 2^{20} adresses (1 048 576 possibilités)
- ALU : Unité Arithmétique et Logique

Techniques des µ-processeurs - 1ière - PBT / PMA

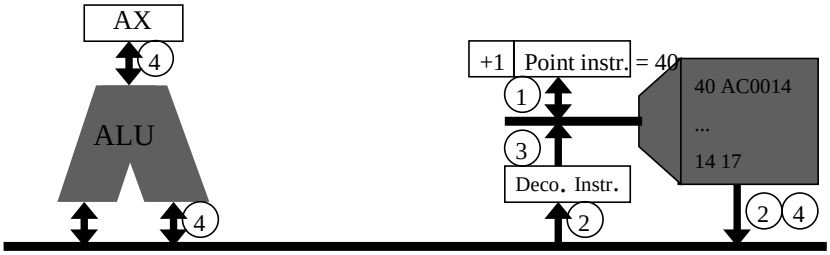
Remarque

Le *+1* renseigné à côté du Pointeur d 'instruction signifie que ce pointeur d 'instruction est incrémenté de une instruction et pas spécialement de la valeur 1. En effet certaines instructions ont une taille supérieure à un octet.

HEB

ESI μ

Machine simplifiée



Instruction MOV AX,[20] AC0014

Porte 'IP' 1 s 'ouvre, l 'adresse 40 circule sur le bus d 'adresse, la RAM fournit la donnée à l 'adresse 40

Les portes 2 s 'ouvrent l 'instruction se retrouve dans le registre d 'instruction

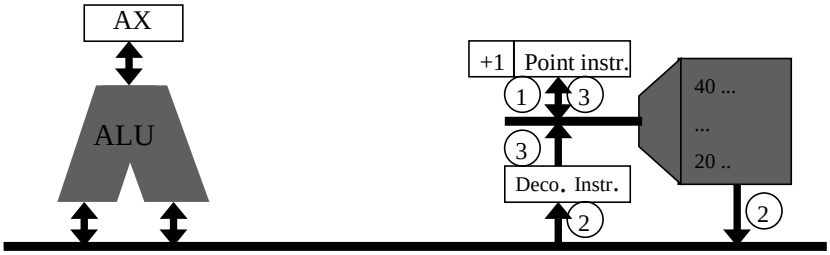
AC = MOV AX donc les portes 3 s 'ouvrent, l 'adresse 20 (0014h) se retrouve sur le bus de d 'adresses, la RAM fournit 17

Les portes 4 s 'ouvrent et la valeur 17 se retrouve dans AX … END

HEB

ESI μ

Machine simplifiée



Instruction JMP [60]

Porte 'IP' 1 s 'ouvre, l 'adresse 40 circule sur le bus d 'adresse, la RAM fournit la donnée à l 'adresse 40

Les portes 2 s 'ouvrent l 'instruction se retrouve dans le registre d 'instruction

Dès la fermeture de la porte 1, le registre d 'instruction est incrémenté afin d 'être prêt à exécuter l 'instruction suivante.

JMP [60] donc les portes 3 s 'ouvrent, l 'adresse 60 est placée dans le pointeur d 'instruction … END

Techniques des µ-processeurs - 1ière - PBT / PMA

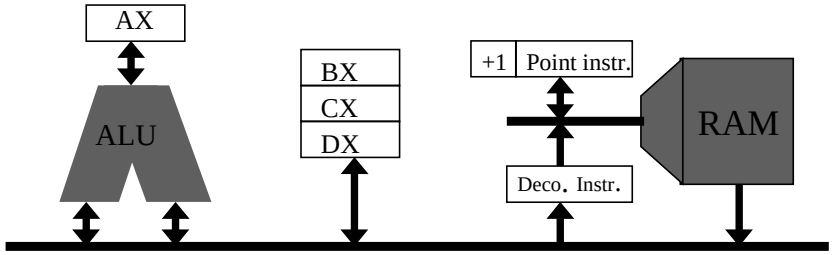
Remarque

Les adresses mémoires sont, ici, écrites en base 10 … par facilité. Pour être tout à fait correct, je devrais 0003Ch écrire au lieu de 60 … ou mieux 0000000000000111100.

HEB

ESI μ

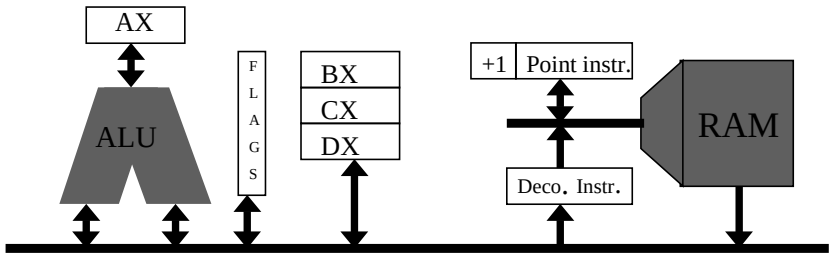
Machine simplifiée



BX : registre de base
CX : registre compteur
DX : registre data

HEB
ESI μ

Machine simplifiée



Les flags : Ensemble de bits (0 ou 1)

- Flag ZERO : se positionne à 1 si le résultat de l 'instruction vaut 0.
- Flag CARRY : se positionne à 1 si le résultat est trop grand pour être représenté.
- ...

Techniques des µ-processeurs - 1ière - PBT / PMA

Remarque

Les flags sont rassemblés dans un registre. Ce registre est particulier en ce sens que c 'est le seul registre pour lequel on accède directement aux bits.

HEB ESI µ

Langage Assembleur - Instructions (1)

- Les instructions ‘ machine ’ sont écrites en hexadécimal …

  0100h : 2D913AB80BAF484BBB66BD

- Le langage assembleur permet :
  - d ’éviter la traduction en hexadécimal
  - d ’utiliser des outils de développement (turbo débugger par exemple)

HEB ESI µ

Langage Assembleur - Instructions (1)

- Le langage assembleur nécessite
  - un compilateur,
  - la connaissance d ’un vocabulaire,
  - le respect d ’une syntaxe précise

MOV AX,0AF0Bh

Remarque : Un nombre hexadécimal dont le premier chiffre est une lettre doit être précédé d ’un 0.

Remarques

En assembleur, les nombres peuvent être représentés en base 10, 2 ou 16.

• *Base 10 :* Ils sont suivis de la lettre *d*

• *Base 2 :* Ils sont suivis de la lettre *b*

• *Base 16 :* Ils sont suivis de la lettre *h* et précédés d ’un **0** si le premier chiffre est une lettre.

HEB
ESI
µ

Langage Assembleur - Instructions (1)

Exemple

0100 : 2D913AB80BAF484BBB66BD

0100 : 2D913A	SUB AX,3A91h
0103 : B80BAF	MOV AX,0AF0Bh
0106 : 48	DEC AX
0107 : 4B	DEC BX
0108 : BB66BD	MOV BX,0BD66h

Remarque

Le code machine, même écrit en hexadécimal (plutôt qu'en binaire) appelle quelques questions :

• Le code écrit en hexadécimal (par facilité de lecture) doit se décomposer en un multiple d 'un octet. Où ?

• Chaque instruction (Taille de 1, 2, 3 ou 4 octets) doit être interprétée. Comment ?

• Les opérandes suivent … sous quelle forme ?

Pour éviter de répondre à ces questions, on utilise un langage ...

HEB ESI µ

Langage Assembleur - Instructions (1)

Au fur et à mesure du cours, nous allons apprendre la syntaxe du langage assembleur.

Quelle est la structure d ’un programme en assembleur ?

HEB
ESI

Langage Assembleur - Instructions (1)

```
.MODEL SMALL
.STACK 100h
.CODE
        MOV AX,45
        ADD AX,12
        MOV BX,6
        MUL BX
        MOV AX,04C00h
        INT 21h
END
```

Directives pour le compilateur
- Environnement de 64k
- Pile d ’une taille de 256bytes

Instructions

Instructions nécessaires
pour signaler la fin du pgm
Signale au compilateur qu ’il a fini.

Techniques des µ-processeurs - 1ière - PBT / PMA

Remarque

Comme pour l ’apprentissage de n ’importe quel langage, il faut commencer par un bout. Il est toujours difficile de choisir entre l ’œuf ou la poule.

Il est donc tout à fait normal que certains passages semblent obscurs ou miraculeux … du moins au début de l ’apprentissage. Car, en effet, au fur et à mesure de l ’accroissement de vos connaissances tout vous paraîtra limpide et clair …

HEB ESI µ

Langage Assembleur - Instructions (1)

Comment exécuter ce programme ?

L ’enregistrer avec un éditeur (edit)

edit monpgm.asm

Le compiler avec le compilateur tasm

tasm monpgm.asm

Le lier avec l ’éditeur de liens tlink

tlink monpgm.obj

L ’exécuter

monpgm.exe

Techniques des µ-processeurs - 1ière - PBT / PMA

Conseil

Procurez-vous assez rapidement un compilateur assembleur, familiarisez-vous avec l ’environnement DOS … et vous serez prêt pour les laboratoires ;-)

HEB
ESI μ

Langage Assembleur - Instructions (1)

Turbo Debugger

Pour débugger un programme ou suivre son fonctionnement.

TD monpgm.exe

TD a besoin d 'une table de commentaires générés à la compilation (tasm) et à l 'édition des liens (tlink) . On utilisera les paramètres **/L/C/ZI** et **/v** respectivement.

Remarque

Nous insisterons beaucoup sur l'utilisation de TD. Tout d 'abord car c 'est lui qui vous permet de voir le fonctionnement interne de la machine (du moins à un certain niveau d 'abstraction). Et deuxièmement, c 'est TD qui vous permettra de déceler vos erreurs lorsque vous écrirez des programmes plus conséquents (dans un cours de Langage Assembleur par exemple).

HEB ESI μ

Langage Assembleur - Instructions (1)

Instruction MOV

- Syntaxe : **MOV dest,src**
- But : copie le contenu de l ’opérande source dans l ’opérande de destination
- Flags : Ne modifie aucun flag

Remarques

Le type précis des opérandes source et destination sera vu plus tard. Sachez d ’ores et déjà que la syntaxe complète des instructions assembleur se trouve dans le syllabus : « Notes techniques PC »

Langage Assembleur - Instructions (1)

Instruction MOV - Exemple

```
MOV AX,1234
MOV BX,1234h
MOV DX,CX
```

```
MOV BX,AX
MOV AX,DX
MOV DX,BX
```

Echange le contenu de AX et DX, BX est perdu.

HEB ESI µ

Langage Assembleur - Instructions (1)

Instruction ADD

- Syntaxe : **ADD dest,src**

- But : ajoute le contenu de l ’opérande source a celui de l ’opérande de destination
- Flags : Modifie les flags zero (ZF) et carry (CF)

Langage Assembleur - Instructions (1)

Instruction ADD - Exemple

```
ADD AX,1234
ADD BX,1234h
ADD DX,CX
ADD CX,CX
```

Double le contenu de CX

Instruction ADD - Exemple (2)

MOV AX,8000h
ADD AX,9000h

Dépassement de capacité, CF = 1

8000 h
\+ 9000 h
= 1 | 1000 h

Dépassement de capacité, CF = 1, ZF = 1

MOV AX,8000h
ADD AX,8000h

8000 h
\+ 8000 h
= 1 | 0000 h

Langage Assembleur - Instructions (1)

Instruction ADC

- Syntaxe : **ADC dest,src**

- But : ajoute le contenu de l 'opérande source et de CF a celui de l 'opérande de destination
- Flags : Modifie les flags zero (ZF) et carry (CF)

Langage Assembleur - Instructions (1)

Instruction ADC - Exemple

MOV DX,1234h	DX-AX <-- 12348765h
MOV AX,8765h	
MOV BX,2468h	BX-CX <-- 24688888h
MOV CX,8888h	
ADD AX,CX	Effectue le calcul
ADC DX,BX	DX-AX <-- DX-AX + BX-CX

Remarque

Cet exemple est purement didactique. Le but est d ’effectuer la somme 12348765h + 2468888h = 369D0FEDh en utilisant des registres qui ne peuvent contenir l ’information en une seule fois.

Langage Assembleur - Instructions (1)

Instruction ADC - Exemple (2)

	8765 h		1234 h	
+	8888 h		2468 h	
=	0FED h et CF set	+	1 h	CF
		=	369D h	

12348765 h + 24688888h = 369D0FED h

HEB
ESI μ

Représentation des nombres en machine

- Comment représenter un ensemble de taille infinie N ou Z en ayant un nombre fini de positions ?
- Un registre étant de taille fixe, il faut se contenter de ne représenter que quelques nombres.

Remarque

Le sujet de la représentation des nombres en machine est très vaste. Comment représenter un ensemble infini en machine ? C 'est impossible, il faut réduire l 'intervalle des nombres représentables et donc se fixer des limites.

Par exemple, nous « déciderons » de ne faire des calculs que dans un intervalle donné [-128,127], ce qui limite notre champ d 'action.

Vous remarquerez, dans la suite, que nous ne traitons que des **nombres entiers**. Cet ensemble est un ensemble *discret* c 'est-à-dire que à un moment donnée, je peux trouver deux nombres entiers tels qu'il n 'existe plus de nombres entiers compris entre eux.

Quid des **nombres réels** ? En effet, je sais que quels que soient deux réels choisis, je peux trouver un nombre réel compris entre les deux … pire, je peux trouver un rationnel et un irrationnel ! Comment faire pour représenter des nombres *pseudo*réels en machine ? … Il faudra attendre le cours de Langage Assembleur ...

HEB ESI µ

Représentation des nombres en machine

Nombres NON signés

Avec 1 bit, on représente 2 nombres : 0 et 1

Avec 2 bits, on représente 4 nombres : 00 01 10 et 11

Avec 8 bits, on représente 2^8 nombres soit l 'intervalle [0;255]

Avec 16 bits, on représente 2^{16} nombres soit l 'intervalle [0;65535]

Remarque

Dans la suite, nous désignerons par *nombre NON signés* le sous-ensemble de N (les naturels) que nous pourrons utiliser et par *nombres signés* le sous-ensemble de Z utilisé.

Représentation des nombres en machine

Nombres signés

Comment représenter les nombres signés ?

On peut utiliser le premier bit pour représenter le signe …
ainsi, 001 représente 1 et 101 représente -1.

Sur 8 et 16 bits, on couvrira alors les intervalles respectifs

[-255;255] et [-32767;32767]

Inconvénient : 00000000 et 10000000 représente 0 !

HEB
ESI μ

Représentation des nombres en machine

Nombres signés - Notation en complément à 2

Représentation d ’un nombre négatif par sa notation en complément à deux.

Rappel : En base 2, pour obtenir le complément à 2, on inverse tous les bits, et on ajoute 1.

Exemple : 5 --> 0101 1010

\+ 1

1011 -- > -5

Remarque

La notion de complément à *b* pour les nombres représentés en base *b* est vue en détail au cours de mathématiques. Nous l ’appliquons ici à la base 2.

HEB
ESI μ

Représentation des nombres en machine

Avantages :

- Une seule représentation du zéro
- Le premier bit représente toujours le signe
- Les opérations binaires d ’addition et de soustraction respectent ses notations

Avec 4 bits, on représente l ’intervalle [-8;7]

Avec 16 bits, l ’intervalle [-32768;32767]

Techniques des µ-processeurs - 1ière - PBT / PMA

Remarque

Les opérations binaires respectent ses notations. L ’effet immédiat étant qu ’un seul circuit électronique est nécessaire pour effectuer les opérations sur des nombres signés ou pas.

HEB
ESI μ

Représentation des nombres en machine

Exercices

- Représenter 5 et -5 sur 4, 8 et 16 bits.
- Que valent ses nombres binaires ?

  01101001

  10110000

  1001101011001001
- Que vaut en base 10 ; CA52h et 711Fh

HEB ESI μ

Représentation des nombres en machine

Exercices - Solutions

- Représenter 5 et -5 sur 4, 8 et 16 bits.

  5 0101 00000101 0000000000000101

  -5 1011 00001011 0000000000001011

- Que valent ses nombres binaires ?

01101001	105
10110000	-80
1001101011001001	-25911

Représentation des nombres en machine

Exercices - Solutions (suite)

- Que vaut en base 10 ; CA52h et 711Fh
  CA52h = $12\ 16^3 + 10\ 16^2 + 5\ 16 + 2 = 51\ 794$ d
  711Fh = $7\ 16^3 = 1\ 16^2 + 1\ 16 + 15 = 28\ 959$ d

HEB ESI μ

Représentation des nombres en machine

Nombres NON signés - Dépassement de capacité

Sur 4 positions, on peut représenter l ’intervalle [0;15] … que se passe-t-il si la somme dépasse 15 ?

--> le flag CARRY est positionné à 1

HEB
ESI μ

Représentation des nombres en machine

Nombres signés - Dépassement de capacité

Sur 4 positions, on peut représenter l ’intervalle [-8;7] … que se passe-t-il si la somme dépasse 7 ou est inférieure à -8 ?

--> le flag OVERFLOW est positionné à 1

HEB ESI µ

Représentation des nombres en machine

Le processeur effectue l ’opération et positionne les flags CF et OF en cas de dépassement de capacité.

CF est positionné si l ’on considère que l ’on travaille sur des nombres NON signés.

OF est positionné si l ’on considère que l ’on travaille sur des nombres signés.

Remarque

Il faut bien comprendre que c ’est tout à fait transparent pour le µ-processeur d ’effectuer une opération sur des nombres signés ou non. Le programmeur « ira », suivant la situation, tester le CF ou le OF.

HEB
ESI μ

Représentation des nombres en machine

Opération	NON Signé	CF	signé	OF
0001 + 0001 = 0010	1 + 1 = 2	0	1 + 1 = 2	0
0110 + 0110 = 1100	6 + 6 = 12	0	6 + 6 = -4	1
0110 + 1110 = 0100	6 + 14 = 4	1	6 + -2 = 4	0
1000 + 1000 = 0000	8 + 8 = 0	1	-8 + -8 = 0	1

Représentation des nombres en machine

Le flag Signe - SF

Le flag de signe est une copie simple du premier bit du nombre … il représente le signe !

HEB ESI µ

Langage Assembleur - Instructions (2)

Instructions ADD et SUB

- Syntaxe : **ADD dest,src**
  SUB dest,src
- But : ajoute/soustrait le contenu de l ’opérande source a celui de l ’opérande de destination
- Flags : Modifie les flags ZF, CF, OF, SF

Techniques des µ-processeurs - 1ière - PBT / PMA

Remarque

Les opérandes *source* et *destination* doivent respecter quelques règles, vous trouverez le détail dans les notes techniques et dans HELPPC. Afin de faciliter votre compréhension, sachez que; les opérandes peuvent être de type **reg** (registre), **mem** (zone mémoire) ou **imm** (valeur immédiate).

Règles

• Jamais deux accès mémoire en même temps

• Les opérandes doivent avoir la même taille … le compilateur doit savoir détecter cette taille.

Exemples

```
ADD  AX,1234
SUB   BX,1234h
ADD  DX,CX
SUB   CX,CX        Met CX à zéro
```

HEB
ESI

Langage Assembleur - Instructions (2)

Instruction MUL

- Syntaxe : **MUL src**
- But : multiplie AX (ou AL) par le contenu de l ’opérande source et place le résultat dans le couple DX-AX (ou AX)
- Flags : Modifie les flags OF, CF=0 si DX=0, ZF et SF indéfinis

Techniques des µ-processeurs - 1ière - PBT / PMA

Type de l ’opérande

L ’opérande *src* peut-être de la taille d ’un byte ou d ’un word.

MUL *byte* - Exécute l ’opération AL * byte -> AX

MUL *word* - Exécute l ’opération AX * word -> DX:AX

Langage Assembleur - Instructions (2)

Instruction MUL - Exemples

MUL CX	DX-AX <-- AX*CX
MUL AX	DX-AX <-- AX^2
MUL DX	DX-AX <-- AX*DX

*Remarque : 16 bits * 16 bits dans 32 bits --> pas de dépassement de capacité.*

HEB ESI μ

Langage Assembleur - Instructions (2)

Instruction DIV

- Syntaxe : **DIV src**
- But : divise le couple DX-AX (ou AX) par le contenu de l 'opérande source. Place le quotient dans AX (ou AL) et le reste dans DX (ou AH)
- Flags : Modifie les flags OF, CF, ZF et SF mais indéfinis

Techniques des µ-processeurs - 1ière - PBT / PMA

Type de l 'opérande

L 'opérande *src* peut-être de la taille d 'un byte ou d 'un word.

DIV *byte* - Exécute l 'opération AX div byte -> AL
AX mod byte -> AH

DIV *word* - Exécute l 'opération DX:AX div word -> AX
DX:AX mod word -> DX

Langage Assembleur - Instructions (2)

Instruction DIV - Exemple

DIV CX AX <-- DX-AX / CX

DX <-- DX-AX mod CX

Remarque : 32 bits / 16 bits dans 16 bits --> un dépassement de capacité est possible ainsi qu 'une division pas 0.

Dans ces cas, le µp stoppe le pgm: « divide by zero »

HEB
ESI

Langage Assembleur - Instructions (2)

Instructions IMUL - IDIV

- Syntaxe : **IDIV src**
  IMUL src
- But : Même rôle que MUL et DIV mais sur des nombres signés.
- Flags : Idem que MUL et DIV

HEB
ESI μ

Représentation des caractères

Convention :

Chaque caractère est représenté par un code.

La taille de ce code est de un byte soit 256 possibilités.

Table ASCII

Quelques exemples de la table ASCII

‘ A ’ = 01000001b = 041h = 65

‘ B ’ = 01000010b = 042h = 66

‘ Z ’ = 01011010b = 05Ah = 90

‘ 0 ’ = 00110000b = 030h = 48

‘ 1 ’ = 00110001b = 031h = 49

‘ 9 ’ = 00111001b = 039h = 57

HEB
ESI μ

Représentation des caractères

Les conventions les plus fréquentes sont :

- ASCII - PC sous DOS
- ANSI - PC sous Windows
- EBCDIC - Main frame IBM

Remarque : Dans certains codes (EBCDIC par ex) les lettres ne se suivent pas toujours.

Code EBCDIC

Voici une partie de la table EBCDIC … pour comprendre comment elle est générée.

Chiffres	0-9	Code	F0h - F9h
Lettres majuscules	A-I	Code	C1h - C9h
	J-R		D1h - D9h
	S-Z		E2h - E9h

Le E1 n 'est pas utilisé car il y a 26 lettres et pas 27 mais aussi car sur une carte perforée la valeur E1h génère deux perforations consécutives[1].

Lettres minuscules	a-i	Code	81h - 89h
	j-r		91h - 99h
	s-z		A2h-A9h

[1] Merci à Jean-Marie Noiret pour l 'information … quid alors du A1h ?

HEB
ESI μ

Historique des µp

Les microprocesseurs apparaissent dans les années 1970 grâce au passage des transistors aux circuits intégrés, avec eux débutent la troisième génération d ’ordinateurs.

Remarques

Suivent ici des slides reprenant l ’historique de l ’évolution des µ-processeurs de la famille Intel uniquement. Pour des informations concernant l ’historique de l ’informatique en général ou d ’autres familles de µ-processeurs, consultez les ouvrages suivants :

Architecture et Technologie des ordinnateurs,

Paollo Zanella et **Yves Ligier**, *Ed. Dunod*

Advanced Microprocessors,

Daniel Tabak, *Ed. Mc Graw - Hill*

HEB
ESI μ

Historique des µp

- **1968** Création d'Intel Corporation par *Gordon Moore* et *Robert Noyce*.
- **1971** Intel 4004, premier microprocesseur de l'histoire.
- **1978** Apparition de l'Intel 8086, premier processeur à architecture x86. µp 16 bits.

Hstorique détaillé ...

1968 Création d'Intel Corporation par *Gordon Moore* et *Robert Noyce*.

1971 Intel 4004, premier microprocesseur de l'histoire, 4 bits, 60.000 opérations par seconde, 108 KHz, 2300 équivalent transistors

1968 Création d'Intel Corporation par *Gordon Moore* et *Robert Noyce*.

1971 Intel 4004, premier microprocesseur de l'histoire, 4 bits, 60.000 opérations par seconde, 108 KHz, 2300 équivalent transistors

1972 Intel 8008, microprocesseur 8 bits, 200 KHz, 3500 équivalent transistors

1974 Intel 8080, µp 8 bits, 2MHz, 6000 transistors.

1978 Apparition de l'Intel 8086, premier processeur à architecture x86. µp 16 bits, bus 16 bits, 5MHz, 29000 transistors

1979 Intel 8088, 16 bits, bus 8 bits

1982 Intel 80286, apparition du mode protégé, 16 bits, bus 16 bits, 12 MHz

1985 Intel 80386DX, 32 bits, bus 32 bits, 16 MHz, 300 000 transistors.

1986 Novembre : Intel 80386, premier microprocesseur 32 bits d'Intel.

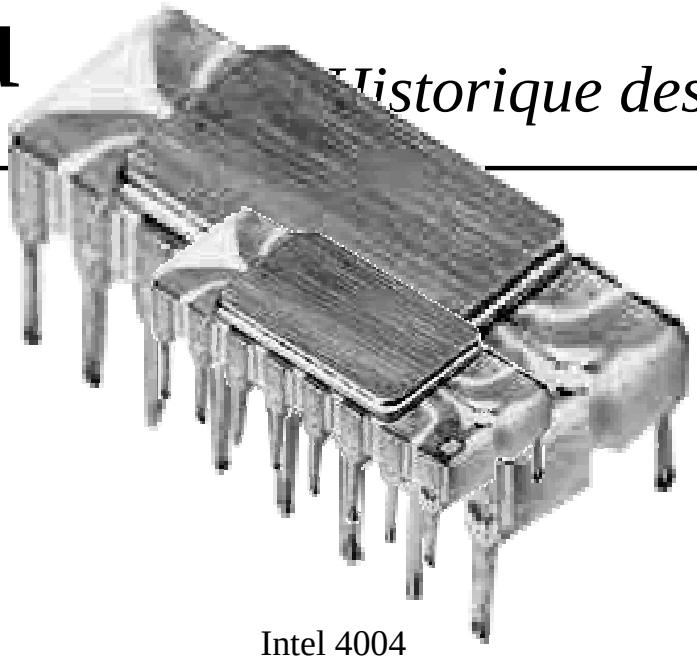
1989 Intel 486DX, intégrant un 80386, un 80387 et de la mémoire cache (8 Ko), 25 MHz, 1 200 000 transistors

1992 Annonce du prochain processeur; Intel 586 baptisé : Pentium.
Edition des spécifications du bus PCI.

HEB

ESI

µ Historique des µp



Intel 4004

Techniques des µ-processeurs - 1ière - PBT / PMA

Intel 4004

1993 Mai Intel annonce la disponibilité du Pentium : deux fois plus puissant qu'un 486, c'est un vrai-faux 64 bits, 32 bits, bus data 64 bits, 60 MHz, 66 MHz.. 3 100 000 transistors,

1994 Février : Intel lance le bus PCI (Peripheral Component Interconnect).

1995 Sortie du Pentium Pro, intégrant 512 Ko de mémoire cache.

1996 Mai : Sortie du Pentium MMX. Modèles : 150MMX, 166MMX et 200MMX.

1996 Juillet : Présentation des spécifications de l'Accelerated Graphic Port (AGP)

1997 Mai : Sortie du Pentium II, intégrant 512 Ko de mémoire cache dans une cartouche,7500 000 transitors

1997 Juin : Sortie du Pentium 233 MMX.

1997 Août : Sortie d'un Pentium Pro avec 1Mo de cache L2.

1997 Septembre : Sortie du Chipset LX pour Pentium II.

1998 Janvier : Sortie du Pentium II 333 Deschutes (gravure en 0,25µ).

1998 Avril : Sortie du Pentium II pour portables (PII 233 et 266).

1998 Avril : Sortie du chipset BX avec bus 100 MHz. Pour Pentium II 350 et 400.

1998 Avril : Sortie du Celeron (Pentium II sans mémoire cache) et de son chipset : le 440EX.

1998 Mai : Sortie du Pentium II Xeon et de ses chipsets : 440 GX et 450 NX.

HEB
ESI

Historique des µp

- **1982** Intel 80286, µp 16 bits
- **1986** Intel 80386, µp 32 bits
- **1989** Intel 80486,
- **1993** Intel Pentium, µp 32 bits, bus data 64 bits
- 1993-2000 Pentium I II III… Mars 2000, Intel livre les premiers 1GHz.
- 2001 Intel Pentium IV

1998 Septembre : Sortie du Celeron avec 128 Ko de mémoire cache L2 intégrée, cadencée à la vitesse du processeur. Modèles : A300 et 333.

1999 Janvier : Sortie du Celeron sur Socket 370. Modèles : 366 et 400.
Egalement disponible sur Slot 1

1999 Fin Février : Sortie du Pentium III équipé des instructions KNI (Katmai New Instructions) et du Processor Serial Number (PSN). Premiers modèles : 450 et 500 MHz.

1999 02 Août : Sortie des Celeron 500 et Pentium *III* 600.

1999 Fin Septembre : Sortie des Pentium *III* B 533 et 600 à FSB 133 MHz.

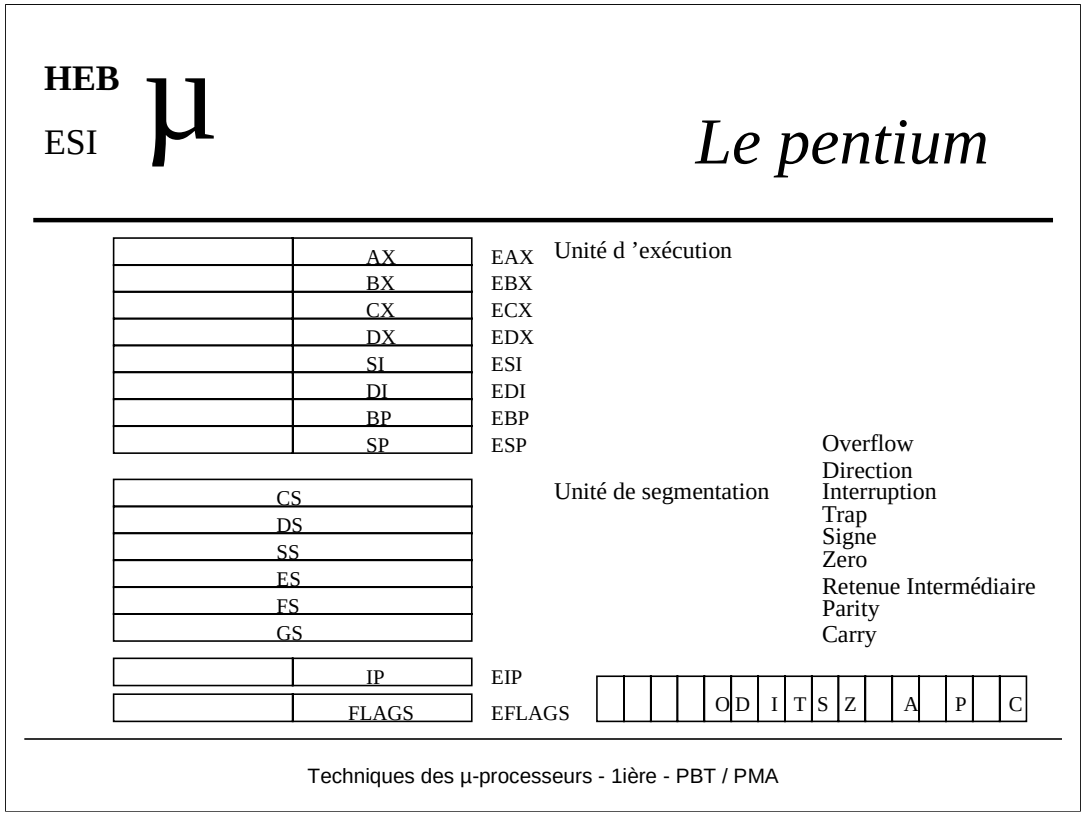
1999 Novembre : Sortie des Pentium *III* E et EB (gravure 0,18µ, cache L2 256Ko "*on die & full speed*").

1999 15 Novembre : Après de nombreux retards, annonce du chipset i820 au Comdex.

2000 Janvier : Pentium *III* pour portable avec technologie SpeedStep.

2000 08 Mars : Intel livre les premiers exemplaires du Pentium *III* 1GHz aux grands intégrateurs.

2001 : Intel livre les premiers exemplaires du Pentium IV 1.6 GHz 1.7 GHz 1.8 GHz 1.9 GHz 2 GHz.

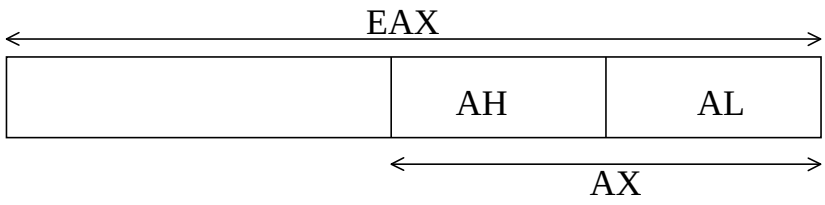


Directive .x86

Afin d 'accéder au set complet d 'instruction d 'un processeur autre que le 8086, il faut ajouter en tête de programme une des directives **.286**, **.386**, ou **.486**.

Les registres généraux

Les quatre registres AX, BX, CX et DX peuvent être considérés comme des registres 8 bits, 16 bits ou 32 bits (à partie du 80386, ils sont précédés d ’un E, extended).



HEB
ESI μ

Langage Assembleur - Les variables

- La mémoire peut contenir une grande quantité d ’informations, c’est un suite de bytes organisés de manière séquentielle.
- Chaque byte est référencié par un numéro

Par exemple,
à l ’adresse [400] se trouve la valeur 2Ah.

HEB
ESI μ

Langage Assembleur - Les variables

Il serait utile d ’avoir la correspondance:

adresse - taille - signification

[400] - 1 byte - age
[401] - 2 bytes - taille

HEB
ESI μ

Langage Assembleur - Les variables

Le langage assembleur se charge de réserver un emplacement mémoire et de lui donner un nom.

Ces emplacements seront réservés de manière séquentielle à l ’adresse DS:[0000h]

Remarque

L ’adresse DS:[0000h] … les notions de segment (ici DS) et d ’offset (ici 0) seront explicitées plus tard. Sachez simplement que des emplacements mémoires peuvent être réservés. Cette réservation se fait de manière séquentielle à partir « d ’un certain point ».

HEB
ESI μ

Langage Assembleur - Les variables

Syntaxe
```
<nom>   DB   <val>
<nom>   DW   <val>
<nom>   DD   <val>
```

<nom> : nom de la variable

- composé de lettres, chiffres et _
- doit commencer par une lettre
- 8 caractères maximun

HEB
ESI μ

Langage Assembleur - Les variables

Syntaxe
<nom> DB <val>
<nom> DW <val>
<nom> DD <val>

- DB : Déclare Byte - Réserve 8 bits
- DW : Déclare Word - Réserve 16 bits
- DD : Déclare Double - Réserve 32 bits

HEB
ESI μ

Langage Assembleur - Les variables

Syntaxe

```
<nom>   DB   <val>
<nom>   DW   <val>
<nom>   DD   <val>
```

<val> : valeur initiale de la variable

- Pas d ’initialisation : ?
- Valeur décimale : 25
- Valeur hexadécimale : 0ABh
- Valeur binaire : 10110111b

HEB
ESI μ

Langage Assembleur - Les variables

Exemple

```
age      DB      25
taille   DW      160
poids    DB      ?
```

```
MOV AL,age      est équivalent à    MOV AL,25
MOV BX,taille   est équivalent à    MOV BX,160
MOV poids,CH    est équivalent à    ...
```

HEB
ESI μ

Langage Assembleur - Les variables

Supposons que l ’adresse mémoire [400] soit libre ainsi que les bytes qui suivent

Assembleur peut réserver [400] pour age et ainsi de suite ...

MOV AL,age	*est équivalent à*	MOV AL,[400]
MOV BX,taille	*est équivalent à*	MOV BX,[401]
MOV poids,CH	*est équivalent à*	MOV [403],CH

HEB
ESI μ

Langage Assembleur - Les variables

Variables de type ‘caractères’

En ASCII, on peut initialiser une variable avec le caractère ‘ A ’ comme suit;

```
lettre      DB    ‘ A ’
lettre      DB    65
lettre      DB    041h
```

Techniques des µ-processeurs - 1ière - PBT / PMA

Remarque

La première initialisation *lettre DB ‘ A ’* est valable quel que soit le code utilisé.

HEB
ESI μ

Langage Assembleur - Les variables

Pour définir une phrase;

phrase DB 'CELA DEFINIT UNE PHRASE'

Ceci déclare 23 bytes en mémoire.

En mémoire, on trouve

67,69,76,65,32,68,69,70,73,78,73…

043h,045h,04Ch,041h,020h,044h,045h,046h...

HEB
ESI μ

Langage Assembleur - Les variables

Si assembleur place la variable *phrase* à l 'adresse [400] et suivantes

Alors, on trouve dans la mémoire [400]=65, [401]=69 …

MOV AL,[405] ≡ MOV AL, 'D'

MOV AL,phrase[6] ≡ MOV AL, 'E'

HEB
ESI
μ

Langage Assembleur - Les Variables

Structure d ’un programme en assembleur

HEB
ESI μ

Langage Assembleur - Les Variables

```
.MODEL SMALL
.STACK 100h
```

```
.DATA
  var1   DB     2
  var2   DW     ?
  var3   DB     ‘ A ’
```

Directives pour le compilateur

- Environnement de 64k
- Pile d ’une taille de 256 bytes

Déclaration des variables

HEB
ESI

Langage Assembleur - Les Variables

```
.CODE
        MAIN   PROC
        MOV    AX,@data
        MOV    DS,AX
```

Début du code

Informe le compilateur du début d ’un bloc

Signale l ’endroit où se trouvent les données.

```
        MOV BL,var1
        MOV AX,var2
        MUL BL
```

Instructions

HEB
ESI

Langage Assembleur - Les Variables

```
        MOV   AX,04C00h
        INT   21h
```

Instructions nécessaires pour signaler la fin du programme.

```
        MAIN  ENDP
```

Informe le compilateur de la fin du bloc

```
END     MAIN
```

Informe au compilateur qu ’il a fini son travail et que le programme commence par le bloc ‘ MAIN ’

HEB
ESI μ

Booter en DOS

Quelles sont les différentes étapes lors de la mise en fonction du DOS.

- Vérifications matérielles (présence et type)
  - CPU
  - Mémoires
  - Périphériques (clavier, lecteur …)

Remarque

Des différences vont, bien évidemment, apparaître suivant le système d ’exploitation que l ’on charge. Les premières étapes concernant le Bios sont, quant à elles, communes car propres à une machine.

HEB ESI µ

Booter en DOS

- Bootstrap Loader

  La séquence de boot commence tjs par l ’exécution du programme se trouvant dans la BIOS ROM.
- Chargement du complément du bios
- Recherche d ’un support bootable (disquette, disque dur …).
- Chargement du SE, ici DOS

HEB
ESI μ

Booter en DOS

– Chargement du code se trouvant sur le MBR. Ce code se termine par AA55h.

– Suivant la manière dont le disque dur est partitionné ou si l ’on boote sur une disquette, le « boot code » charge le code se trouvant sur la partition primaire … bootable.

– Le processus examine la structure du disque.

Booter en DOS

  - Le processus examine le répertoire root à la recherche du SE … pour DOS, ce sont les fichiers :

    IO.SYS, MSDOS.SYS, COMMAND.COM
  - DOS est chargé.
- Lecture des fichiers de configuration
  - config.sys
  - autoexec.bat

HEB ESI μ

Les interruptions

Qu ’est-ce qu ’une interruption ?

- La sonnerie du téléphone
- J ’arrête de jouer sur mon PC car je veux aller manger.
- Maman m ’appelle pour aller manger.

HEB
ESI μ

Les interruptions

Il existe deux types d ’interruptions.

- Interruptions externes ou hardware

  Non prévues par le programmeur,

  Signal provenant de l ’extérieur, ‘ hors ’ du processus en cours. (Plus de papier dans l ’imprimante).

- Interruptions internes ou software

  Appel du programmeur au SE

HEB ESI μ

Les interruptions

Appel du programmeur au SE

Dans son code, le programmeur place une instruction du type

INT nn

Cette instruction rend la main au SE. Le programmeur fait appel à un service existant du DOS ou du BIOS

Les interruptions

Exemple

```
MOV AH,03h
INT 21h
```

Fait appel au DOS. Attend l 'appui sur une touche au clavier …

… place dans AL le code ASCII de la touche enfoncée et rend la main au programme.

HEB
ESI μ

Les interruptions

Comment les interruptions sont-elles traitées par le µp ?

- Fin de l ’instruction en cours
- Sauvegarde du contexte de travail (registres, flags …)
- Détermination de l ’adresse de la routine à exécuter
- Exécution de la routine
- Retour au processus interrompu

Vocabulaire

Vous verrez en détail plus tard la signification exacte du mot **processus**. Pour l ’instant, il suffit de savoir qu’un processus est un programme en cours d ’exécution.

HEB ESI μ

Les interruptions

Une interruption est une routine à exécuter …

Cette routine se trouve en mémoire …
quelque part !

Comment la trouver ?

- Grâce à son adresse (une adresse = 4 octets, un segment et un offset)
- Toutes les adresses se trouvent dans la table des interruptions.

Les interruptions

La table des interruptions

- Table de 1024 octets située à l ’adresse 0000:0000h jusque 0000:03FFh
- Elle contient les adresses des routines d ’interruptions, une adresse se code sur 4 octets;
- 256 interruptions possibles

HEB
ESI μ

Les interruptions

Table des interruptions

	Segment		Offset	
0000:0000h				
0000:0004h				
0000:0008h				
0000:000Ch				
0000:0010h				
0000:0014h				
0000:0018h				
0000:001Ch				
0000:0020h				
0000:0024h				
0000:0028h				
0000:002Ch				

HEB
ESI μ

Les interruptions

- Les interruptions sont identifiées par un numéro
- Ce numéro permet d ’identifier la routine à exécuter dans la table des interruptions
- Une interruption peut être appelée par le programmeur (appel au SE) en utilisant sont numéro INT 10h

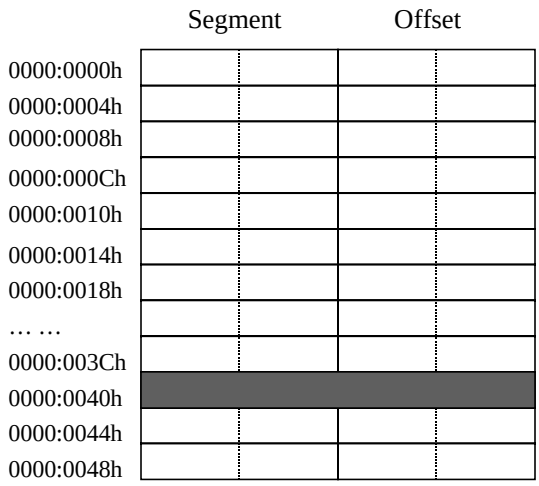
HEB
ESI μ

Les interruptions

Table des interruptions

INT 10h

10 * 4 = 40 h



Techniques des µ-processeurs - 1ière - PBT / PMA

HEB ESI μ

Les interruptions

L ’instruction INT *xx*h

- Appel au SE
- *xx* est le numéro de l ’interruption
- A chaque interruption est associé un numéro de *service* de l ’interruption que l ’on place généralement dans AH.

HEB
ESI μ

Les interruptions

L ’instruction INT *xx*h

Exemple

```
MOV AH,2Ah
INT 21h
```

Les registres reçoivent la date système (jour, mois, année)

HEB ESI µ

Langage Assembleur

Branchements - Conditions - Répétitions

- Dans un programme séquentiel, les instructions se suivent de la première à la dernière.
- Dès qu'une instruction est placée dans le décodeur d 'instruction, le registre d 'instruction est incrémenté pour pointer sur l 'instruction suivante.

HEB
ESI
µ

Langage Assembleur

Branchements - Conditions - Répétitions

- Le langage assembleur permet de modifier le pointeur d ’instruction (CS:IP) pour se placer à une instruction marquée par un label.
- C ’est l ’instruction

JMP *label*

Qu'est -ce qu'un label ?

- Un label (étiquette) se place en début de ligne, c 'est un nom suivi de ' : '

  label:

- Assembleur se charge de remplacer ce label par le numéro de ligne lors de la compilation.

L ’instruction JMP

- Syntaxe : **JMP *label***
- But : Se brancher inconditionnellement à l ’instruction marquée par *label*
- Flags : Aucun flag n ’est modifié

Langage Assembleur

Branchements - Conditions - Répétitions

Remarque

JMP *label* se traduit en langage machine par

EB *byte*	où *byte* contient le déplacement (-128 à 127 bytes)
E9 *word*	où *word* contient le déplacement (-32768 à 32767 bytes)
EA *double*	où *double* contient l 'adresse

Remarque (suite)

C ’est le compilateur assembleur qui choisit, en fonction de la distance qui sépare l ’instruction JMP du label, comment il va traduire l ’instruction.

HEB ESI µ

Langage Assembleur

Branchements - Conditions - Répétitions

- On utilise très rarement les branchements inconditionnels …
- … on leurs préfère les branchements conditionnels
  - les conditions se font sur l ’état des flags
  - une exception avec le registre CX

Les instructions Jf

- Syntaxe : **Jf *label***
- But : Se brancher à l ’instruction marquée par *label* en fonction de l ’état d ’un flag
- Flags : Aucun flag n ’est modifié

Les instructions Jf **(suite)**

J*f label* signifie « se brancher à *label* si f=1 »

JN*f label* signifie « se brancher à *label* si f=0 »

On aura :

JC	C=1	JNC	C=0
JO	O=1	JNO	O=0
JS	S=1	JNS	S=0
JZ	Z=1	JNZ	Z=0

L'instructions JCXZ

- Syntaxe : **JCXZ *label***
- But : Se brancher à l 'instruction marquée par *label* si CX=0
- Flags : Aucun flag n 'est modifié

Exemple

Si AX=BX alors CX=1 sinon CX=0 s ’écrira :

```
        SUB  AX,BX
        JZ   egal
        SUB  CX,CX
        JMP  fin
egal:   MOV  CX,1
fin:
```

HEB
ESI μ

Langage Assembleur

Branchements - Conditions - Répétitions

L'instructions CMP

- Syntaxe : **CMP dest,src**
- But : Comparer le contenu de l 'opérande *src* au contenu de l 'opérande *dest*
- Flags : Les flags ZF, CF, OF, SF sont modifiés.

Techniques des µ-processeurs - 1ière - PBT / PMA

Remarque

L 'instruction CMP fonctionne comme l 'instruction SUB.

Lors de l 'exécution de l 'instruction CMP, une soustraction est effectuée entre l 'opérande *dest* et l 'opérande *src*.

Comme pour l 'instruction SUB, les flags sont positionnées en fonction du résultat de l 'opération.

A l 'inverse de l 'instruction SUB, l 'opérande *dest* n 'est pas modifé.

L'instructions CMP **(suite)**

Une instruction CMP sera généralement suivie d 'une instruction testant un flag.

Exemple

```
        CMP   AX,1234h
        JNZ   different
        …
different:
```

Langage Assembleur

Branchements - Conditions - Répétitions

Instruction J*cc*

- Pour éviter au programmeur de chercher quel flag tester, Assembleur met à disposition une série d ’instruction J*cc*
- où *cc* est une condition du type *plus petit/plus grand*

Instruction J*cc* (suite)

- On distinguera les nombres signés des nombres non signés ... Pq ?

  AL = 10000000b BL=01000000

  CMP AL,BL AL>BL ou AL<BL ?

  Signés : AL=-128 et BL=64 --> AL<BL

  NON Signés : AL=128 et BL=64 --> AL>BL

Instruction J*cc* (suite)

- NON Signés :
  - AL>BL : AL est après BL **ABOVE** *JA*
  - AL<BL : AL est avant BL **BELOW** *JB*
- Signés :
  - AL>BL : AL est plus gd que BL **GREATER** *JG*
  - AL<BL : AL est plus petit **LITTLE** *JL*

L'instructions J*cc* (suite)

- Syntaxe : ***Jcc*** *label*
- But : Se brancher à l 'instruction marquée par *label* en fonction de la condition *cc*
- Flags : Aucun flag n 'est modifié

HEB
ESI

Langage Assembleur

Branchements - Conditions - Répétitions

Table des Jcc

	NON Signés	
>	JA (Above)	JNBE (Not below or equal)
>=	JAE (Above or equal)	JNB (Not below)
<	JB (Below)	JNAE (Not above or equal)
<=	JBE (Below or equal)	JNA (Not above)
=	JE (Equal)	
#	JNE (Not Equal)	

Remarque

Les instructions assembleur d ’une même ligne sont équivalentes … d ’ailleurs, leur code opératoire est identique.

JA	77
JAE	73
JB	72
JBE	76
JE	74
JNE	75

Table des Jcc

Signés		
>	JG (Greater)	JNLE (Not less or equal)
>=	JGE (Greater or equal)	JNL (Not less)
<	JL (Less)	JNGE (Not greater or equal)
<=	JLE (Less or equal)	JNG (Not greater)
=	JE (Equal)	
#	JNE (Not Equal)	

Remarques

•Les instructions assembleur d ’une même ligne sont équivalentes … d ’ailleurs, leur code opératoire est identique.

JG	7F
JGE	7D
JL	7C
JLE	7E
JE	74
JNE	75

•Les instructions JE et JNE sont identiques que l ’on travaille sur des nombres signés ou non.

La directive JUMPS

- Rappel : L ’instruction JMP label se code de trois manières différentes suivant qu’il faille faire un déplacement d ’un, de deux ou encore de quatre bytes.
- Les instructions Jf et Jcc quant à elles n ’acceptent qu’un déplacement codé sur un byte (-128 à +127 bytes).

Langage Assembleur

Branchements - Conditions - Répétitions

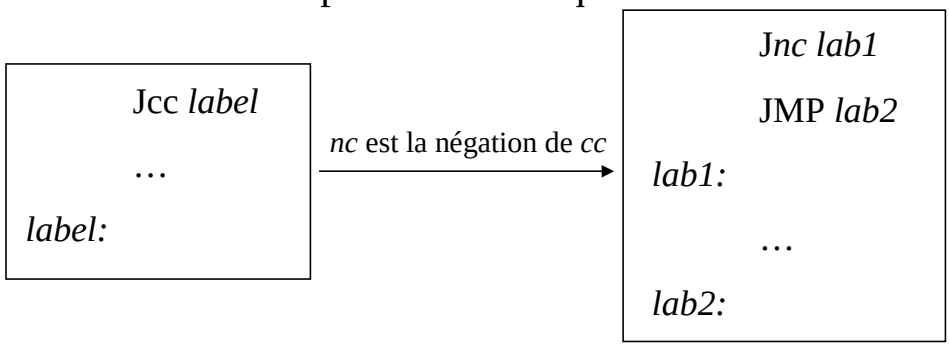
La directive JUMPS (suite)

- La directive JUMPS permet les déplacements longs
- C ’est le préprocesseur qui s ’en charge …

Comment ?

La directive JUMPS (suite)

L ’assembleur remplace automatiquement



Langage Assembleur

Branchements - Conditions - Répétitions

L'instruction LOOP

- Syntaxe : **MOV CX,*n***

  label:

  LOOP *label*
- But : Se brancher à l 'instruction marquée par *label* tant que CX # 0.
- Flags : Aucun flag modifié

L'instruction LOOP (suite)

- Exemple :

```
            MOV   CX,35
label:
            … instructions
            LOOP  label
```

- Les instructions seront exécutées 35 fois ... si l 'on ne modifie pas CX.

Remarque

```
            SUB         CX,CX
label:
            … instructions
            LOOP        label
```

Ce morceau de code aura pour effet d 'exécuter 65536 fois les instructions *instructions*.

En effet, l 'instruction LOOP a pour effet de décrémenter de 1 le registre CX et de tester ensuite s 'il est nul. Ce qui, dans ce cas, fera répéter 65536 fois la séquence d 'instructions.

L'instruction LOOP (suite)

- Une série d 'instructions sont associées à l 'instruction LOOP.

LOOPE = LOOPZ	(equal, ZF=1 ?)
LOOPNE = LOOPNZ	(not equal,ZF=0 ?)
LOOPD	(double, test ECX)
LOOPDE = LOOPDZ	(double equal)
LOOPDNE = LOOPDNZ	(double, not equal)

Remarque.

L 'instruction LOOPZ teste le registre CX et le flag Z. Le branchement se fera au label si CX est différent de 0 et si le flag Z est on

HEB ESI μ

Langage Assembleur

Branchements - Conditions - Répétitions

Lien entre Assembleur et C

- Instruction if
- Instruction for
- Instruction while
- Instruction do while

HEB ESI µ

La pile du 8086

- Les processeurs Intel possède une pile LIFO (Last In First Out)
- La taille de la pile doit être signalée au compilateur par la directive

  .STACK 100h
- 100h signifie que cette taille sera de 256 bytes.

HEB
ESI μ

La pile du 8086

Deux instructions permettent de gérer cette pile.

PUSH *op*

POP *op*

où *op* est par exemple un registre

Attention ce registre doit être de minimum un word.

Remarque

Consultez les notes techniques pour savoir exactement ce que l ’on peut empiler.

HEB
ESI μ

Segmentation de la mémoire

- Dans le 8086, la mémoire est adressée par des adresses de 20 bits,
- On dispose donc de 2^{20} bytes soit 1 048 576 cases mémoires pouvant contenir de l 'information,
- La mémoire est découpée en segments de 64 K

Définition

Un segment est une zone mémoire attribuée par le système d 'exploitation (SE) lorsqu'il charge le programme en mémoire.

Pourquoi la taille d 'un segment est-elle fixée à 64 K ?

Une adresse mémoire est déterminée par une valeur de segment et une valeur d 'offset. Toute deux sont d 'une taille d 'un word (16 bits). Je peux donc me déplacer dans mon segment grâce à la valeur de l 'offset … soit 2^{16} possibilités … 64 K.

HEB ESI µ

Segmentation de la mémoire

- Lors de l ’exécution d ’un programme plusieurs segments nous sont attribués.
- Chaque segment est repéré grâce à un registre de segment (CS par exemple).
- Un registre de segment ne pointe que vers une adresse multiple de 16.

CS:0000h

Techniques des µ-processeurs - 1ière - PBT / PMA

Remarques

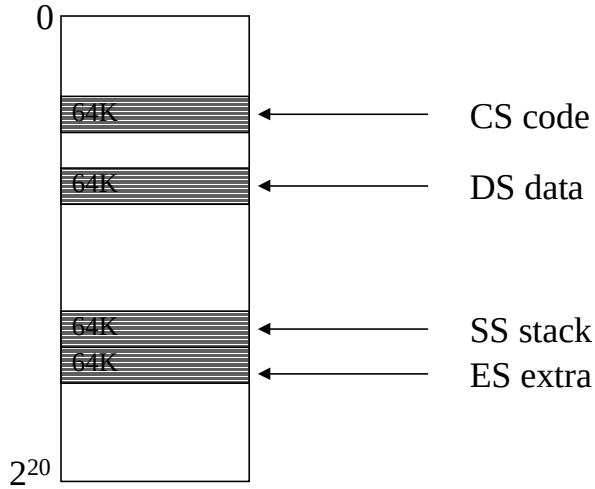
• Si l ’on travaille en assembleur dans un model small, la situation n ’est pas tout à fait semblable … ceci sera vu en détail dans le cours de **Langage Assembleur**.

• Pourquoi une adresse multiple de 16 ?

En début de segment, l ’offset de l ’adresse est nul, on a donc d ’office une adresse de la forme xxxx:0000h qui après ‘ Adresse Offset Adder ’ me donne une adresse de 20 bits de la forme yyyy0h … qui est bien un nombre multiple de 16.

HEB
ESI μ

Segmentation de la mémoire



Techniques des µ-processeurs - 1ière - PBT / PMA

Remarque

• Cette vue a pour seul but d ’illustrer l ’état de la mémoire à un moment donné. Il ne faut pas perdre de vue que d ’autres informations peuvent être stockées dans la mémoire au même moment (SE, autres processus, …).

L ’ordre dans lequel les segments sont placés n ’a pas d ’importance …

Ces détails seront vus dans le cours de Système d ’exploitation de seconde.

• Le code segment a une taille de 64K cela veut-il dire que mon code ne peut contenir que 65 536 caractères ?

HEB ESI μ

Segmentation de la mémoire

Le Code Segment CS

- Associé au registre offset **IP**
- CS:IP forme l ’adresse de l ’instruction à exécuter
- Le Code Segment est considéré par le 8086 comme celui dans lequel se trouve le code du programme.

HEB ESI µ

Segmentation de la mémoire

Le Data Segment DS

- L ’offset sera défini en fonction du type d ’adressage.
- Le Data Segment contient l ’adresse du début du segment des données. Pour toute instruction se référant à des données, le µp utilisera DS pour former l ’adresse physique de la donnée.

HEB
ESI μ

Segmentation de la mémoire

Le Stack Segment SS

- Associé au registre offset **SP**
- SS:SP forme l ’adresse du dernier élément empilé sur la pile.
- Ce registre désigne l ’origine du segment utilisé pour la pile LIFO du µp.
- Cette pile est utilisée par assembleur pour stocker l ’adresse de retour d ’une procédure par exemple.

Techniques des µ-processeurs - 1ière - PBT / PMA

Remarque

Contrairement à d ’autres registres d ’offset, SP à pour valeur initiale la taille de la pile ! En effet, le µp remplit sa pile « *à l ’envers* »

Dans le cas d ’une pile de 256 bytes : **.STACK 100h**

Lors de l ’initialisation du programme, avant l ’exécution de la première instruction. SS:SP pointe sur le sommet de la pile qui à pour valeur **SP=100h.**

Lorsque j ’empile un premier élément d ’un word, **SP=0FEh**.

Segmentation de la mémoire

L ’Extra Segment ES

- Associé généralement à un registre **SI** ou **DI**
- Ce registre permet également d ’accéder à des données ...

Mode d 'adressage

Adressage implicite

– Certaines instructions ne nécessitent aucun opérande.

– L 'instruction est complètement définie par le mnémonique.

– DAA (*decimal adjust after addition*)

Mode d 'adressage

Adressage registres

– Tous les opérandes sont des registres

– MOV AX,BX
– SHL EAX,CL
– MUL AX

Mode d 'adressage

Adressage immédiat

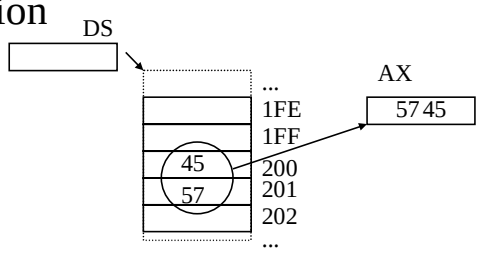
– Dans ce mode d 'adressage, la source est une constante qui sera codée directement dans l 'instruction.

– MOV EAX,12F4h
– SHL BX,2

Mode d 'adressage

Adressage direct

– Dans ce mode d 'adressage, le programmeur donne l 'adresse mémoire de la valeur qui participe à l 'opération



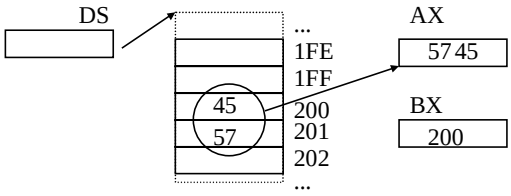
– MOV AX,[200]

Mode d 'adressage

Adressage indirect

– Dans ce mode d 'adressage, le programmeur donne l 'adresse mémoire par l 'intermédiaire d 'un ou de plusieurs registres (BX, SI et DI)

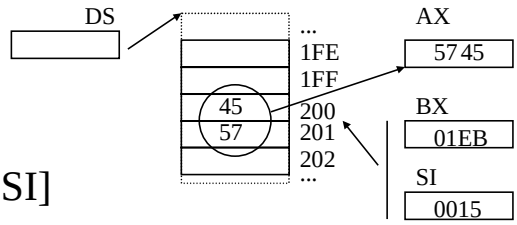
– MOV AX,[BX]



Mode d 'adressage

Adressage indirect

– Autre exemple



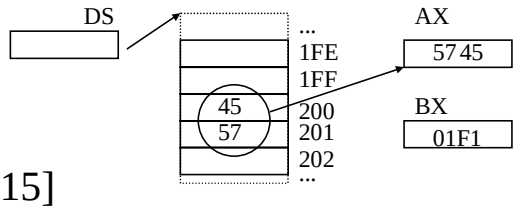
– MOV AX,[BX+SI]

HEB ESI µ

Mode d 'adressage

Adressage indirect

– Autre exemple



– MOV AX,[BX+15]

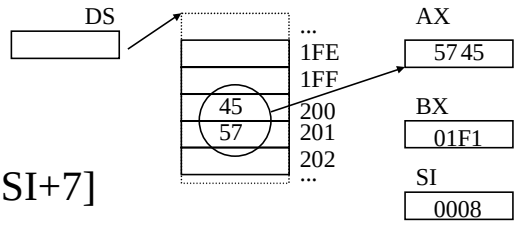
Attention

Dans la notation en assembleur, le 15 signifie 15 en base dix et non 15h … d 'où la modification de BX ...

Mode d 'adressage

Adressage indirect

– Autre exemple



– MOV AX,[BX+SI+7]

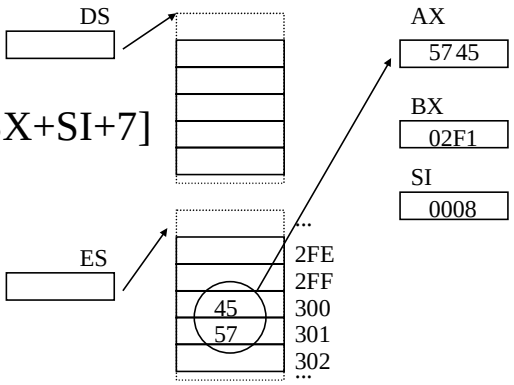
HEB
ESI μ

Mode d 'adressage

Adressage indirect

– Autre exemple

– MOV AX,ES:[BX+SI+7]



Techniques des µ-processeurs - 1ière - PBT / PMA

HEB ESI μ

Questions ?

Merci de votre attention ...