Faculté des Sciences de Tétouan - 2022/2023
SMI – S4

Architecture des ordinateurs

Assembleur

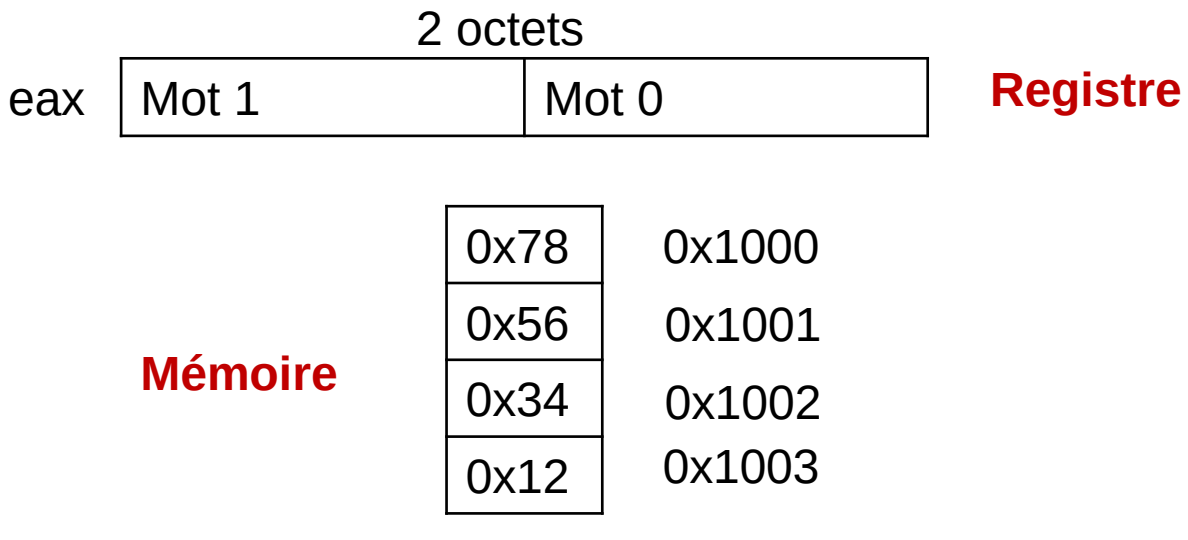
Pr. Zaynab EL KHATTABI

Rangement des données en mémoire

- Comme la mémoire est organisée en octets, 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**)
  - On peut mettre **l'octet de poids fort à l'adresse haute** (**little Endian**).
- **Intel** fonctionne avec le mode **Little Endian.**

Exemple: le registre eax contient 0x12345678

mov 0x1000, eax



Jeu d’instructions:

Charger en provenance de la mémoire, sauver en mémoire, additionner, etc., sont des instructions pour le processeur, l’ensemble des instructions compréhensibles forme le jeu d’instructions. Les instructions que l’on trouve dans le jeu de chaque processeur peuvent être classées en 6 groupes :

- **transfert de données :** pour charger de la mémoire, sauver en mémoire, effectuer des transferts de registre à registre, ou à mémoire , etc. ;
- **opérations arithmétiques :** les quatre opérations de base en entier, signé ou non, court ou long, ou en réel en virgule flottante, simple ou double précision ;
- **opérations logiques**
- **contrôle de séquence :** branchement impératifs et conditionnels, boucles, appels de procédures, etc. ;
- **entrées/sorties** (read, print, etc.) ;
- **manipulation diverses :** décalages, conversions de format, incrémentation de registres, etc.

Instructions de transfert de données

L’instruction mov :

- Elle sert à enregistrer une valeur dans un espace de stockage.

mov destination, source

  - **Destination** : l'espace de stockage où l'on veut mettre notre valeur (registre / mémoire).
    - La destination peut être un registre, une adresse mémoire ou un registre de segment
  - **Source:** la valeur à enregistrer.
    - La source peut être un registre, une adresse mémoire, un registre de segment ou une valeur immédiate.

- Il existe différentes modes d’adressage pour spécifier l’adresse d’une case mémoire dans une instruction.

Instructions de transfert de données

Taille des données transférés:

- Pour spécifier la taille de donnée à transférer à partir ou vers une adresse mémoire, on peut utiliser les mots clés suivantes:
  - **BYTE PTR pour un octet.**
  - **WORD PTR pour un mot de 16 bits.**
  - **DWORD PTR pour un double mot (32 bits)**

Exemple:

- **mov byte ptr [1100H],65H** : transfère la valeur 65H (sur 1 octet) dans la case mémoire d'offset 1100H
- **mov word ptr [1100H],65H** : transfère la valeur 0065H (sur 2 octets) dans les cases mémoire d'offset 1100H et 1101H.

Instructions de transfert de données
Mode d’adressage

- On appelle « mode d'adressage » la manière dont la donnée est spécifiée dans une instruction. Selon le mode d'adressage la taille de l'instruction peut varier de 1 à 4 octets.
- L'architecture Intel supporte plusieurs modes d'adressage :
  - le mode d'adressage immédiat
  - Le mode d’adressage registre
  - Le mode d’adressage direct
  - Indirection registre
  - Indirection registre avec offset
  - Indirection registre avec offset registre (index)
  - Indirection registre avec index + offset

Instructions de transfert de données
Mode d'adressage

Mode d'adressage immédiat

- Permet de transférer une valeur dans un registre par la syntaxe suivante:

  mov registre, valeur

- La donnée est spécifiée immédiatement après l'instruction. Elle est donc située dans le segment de code.
- Exemples:

mov eax,0122Bh	la valeur 0x122B est placée dans eax
mov AX, 1E25H	la valeur 0x1E25 est placée dans AX

Instructions de transfert de données
Mode d'adressage

Mode d'adressage registre

- Permet de transférer une valeur dans un registre par la syntaxe:

  mov registre1, registre2

- Le transfert se fait de registre à registre.
- Les données doivent être de même type
- Exemples:

  mov ebx,eax le contenu de eax est copié dans ebx.

  mov BX, AX le contenu du registre AX est copié dans le registre BX

Instructions de transfert de données
Mode d'adressage

Mode d'adressage direct

- Permet de transférer le contenu d'une case mémoire définie par une adresse effective (offset) vers un registre par la syntaxe:

  mov registre, variable

- La variable est interprété comme une adresse
- L'instruction comporte l'adresse de la case mémoire où se trouve la donnée.
- Ce mode d'adressage provoque un temps d'exécution de l'instruction plus long car l'accès à la mémoire principale est plus long que l'accès à un registre.
- L'adresse effective représente l'offset de la case mémoire dans le segment de données (DS, segment par défaut)
- Exemples:

  mov DL, [1894H]

  mov AX, [1B67H]

Instructions de transfert de données
Mode d'adressage

Mode d'adressage direct

- Permet de changer le segment lors d'un adressage direct en ajoutant un préfixe de segment par la syntaxe:

mov registre, registre de segment :[adresse]

Exemples:

mov BL, ES: [1200H]

Permet de copier la valeur qui se trouve à l'offset 1200H dans le segment ES

Instructions de transfert de données
Mode d'adressage

Mode d'adressage avec registre de base

- Permet de transférer une donnée dont l'offset est contenu dans un registre de base BX ou BP vers un registre par la syntaxe:

mov registre, [BX ou BP]

- Le segment associé, par défaut, au registre BX est le segment de données DS.
- Le segment par défaut associé au registre de base BP est le segment de pile SS.

Instructions de transfert de données
Mode d'adressage

Mode d'adressage avec registre d'index

- Permet de transférer une donnée dont l'offset est contenu dans un registre d'index SI ou DI (associés par défaut au segment de données) vers un registre (ou l'inverse) par les syntaxes:

mov registre, [SI ou DI]

mov [SI ou DI], registre

- Exemples:

mov AX, [SI]

➢ Charger le registre AX avec le contenue de la case mémoire dont l'offset est contenue dans SI

mov [DI] , AL

➢ Charger la case mémoire dont l'offset est contenue dans DI avec le contenu de AL

Instructions de transfert de données
Mode d’adressage

Mode d’adressage registre avec index + offset

- Pour obtenir l’offset, des valeurs constantes peuvent être ajoutées aux registres de base ou d’index.
- Pour charger le registre AX avec le contenue de la case mémoire dont l’offset est contenue dans SI plus un déplacement de 100H, on peut utiliser une des instructions suivantes:

mov AX, [SI+100H]

mov AX, [SI][100H]

mov AX, 100H[SI]

Instructions de transfert de données
Mode d'adressage

Mode d'adressage registre avec index + offset

- Les modes d'adressage basés (avec registre de base) ou indexés permettent la manipulation de tableaux rangés en mémoire .

- **Exemple:** Charger la valeur 1234H et 5678H dans les deux cases (0 et 2) du tableau table. (short tableau[ ] = {50, 75, 342, 9, … })

```
mov SI,0
mov word ptr table[SI],1234H
mov SI,2
mov word ptr table[SI],5678H
```

Instructions de transfert de données
Mode d’adressage

Mode d’adressage registre avec index + offset

Permet d’obtenir l’offset en faisant la somme d’un registre de base, d’un registre d’index et d’une valeur constante par la syntaxe suivante:

mov registre, [SI+BX+valeur]

Exemple: Charger le registre AX avec le contenu de la case mémoire dont l’offset est contenu dans BX+SI+100H

mov AX, [BX+SI+100H]

mov AX, [BX][SI] 100H

Instructions de transfert de données
Mode d'adressage

Mode d'adressage registre avec index + offset

- Ce mode d'adressage permet l'adressage de structures de données complexes(matrices, enregistrements,..)
- **Exemple:**

```
mov BX,10
mov SI,15
mov byte ptr matrice[BX][SI],12H
```

  - Dans cet exemple, BX et SI jouent respectivement le rôle d'indices de ligne et de colonne dans la matrice matrice.

Structure d’un programme en assembleur

```
data segment
                    ; add your data here!
data ends
code segment
Assume Cs:Code, Ds: data
start:
   ....                          ; add your code here
    mov ax, 4c00h                ; exit to operating system.
    int 21h
code ends
end start ;
```

Exemple d’un programme en MASM

```
.586                        ; processeur = Pentium
.model flat, stdcall        ; un seul segment de 4Go, appel standard
option casemap: none        ; l'assembleur respecte les majuscules et minuscules
;-----------------------------------------------------------------------------
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib          ; librairie où se trouve ExitProcess
;-----------------------------------------------------------------------------
maFonction PROC: a:DWORD     ; prototype de maFonction
;-----------------------------------------------------------------------------
.data                                        ; variables globales initialisées
…
.data?                                       ; variables globales non initialisées
…
```

Exemple d’un programme en MASM

```
.code
;-----------------------------------------------------------------------------------
start:                  ; code du programme principal
...
invoke ExitProcess, 0           ; retour à Windows
;-----------------------------------------------------------------------------------
maFonction proc a:DWORD                 ; déclaration de maFonction(a)
LOCAL b:WORD                            ; déclaration de variables locales
LOCAL c[10]:BYTE                        ; c est un tableau de 10 octets
…
maFonction endp
;-----------------------------------------------------------------------------------
end start                               ; fin du programme
```

Directives de base

- Pour programmer en assembleur, on doit utiliser, en plus des instructions assembleur, des directives ou pseudo-instructions, (exp: pour créer de l'espace mémoire pour des variables, pour définir des constantes, etc.)

Définition de variables globales

```
.data
    db 0                                ; définit un octet initialisé à 0
    db "msg", 0                         ; définit une chaîne de caractères terminée par un NULL
    dw 100                              ; définit un mot initialisé à 100 (0x64)
    dw 1, 2, 3                          ; définit un tableau de trois mots initialisés à 1, 2, 3
    dd 0F70ABCDh                        ; définit un mot double initialisé à 0xF70ABCD
    dd 10 dup(0)                        ; définit un tableau de 10 valeurs initialisées à 0
.data?
    db ? ; définit un octet non initialisé
    dw 10 dup(?)                        ; définit un tableau de 10 mots non initialisés
```

Directives de base

Définition de constantes

.const

```
dix equ 10
```

Type de processeur

.386, .486, .586

Début du programme

.code

- Appel d'une fonction ou d'une procédure ou d'un sous-programme :

```
invoke fonction a, b, c                ; appelle fonction(a, b, c)
```

- Le résultat d'une fonction est toujours dans al, ax ou eax, selon que la taille du résultat est 8, 16 ou 32 bits.

Directives de base

Inclusion de fonctions de librairie

include \masm32\include\kernel32.inc

includelib \masm32\lib\kernel32.lib

Fin du programme

end

Exemple d’un programme en MASM

- Le programme principal doit commencer à start et se terminer par **ExitProcess(0)**.
- Les sous-programmes appelés par ce programme sont définis entre **ExitProcess** et **end start**.
- Ces fonctions doivent avoir un prototype avant les déclarations de données .data et .data?.
- On peut déclarer des variables locales dans les fonctions comme c'est le cas pour b et c. b est un mot, c est une chaîne de 10 octets.
- Au besoin, on pourra inclure d'autres fichiers d'en-tête, telles que windows.inc, user32.inc, gdi32.inc et masm32.inc et les librairies correspondantes user32.lib, gdi32.lib et masm32.lib.

Instructions arithmétiques et Logiques

- Les instructions arithmétiques et logiques sont effectuées par l'unité arithmétique et logique. Il s'agit d'opération directement effectuées sur les bits de la donnée que l'on traite.
  - Les instructions d'addition
  - Les instructions de soustraction
  - Les instructions de mutiplication
  - Les instructions de division
  - Les instructions d'ajustement de données
  - Les instructions de conversion de type
  - Les instructions logiques (ET, OU, ...)
- Les opérations arithmétiques et logiques modifient l'état des indicateurs.

Instructions arithmétiques

Instructions d'addition :

ADD : Cette instruction effectue une addition, le résultat est placé dans le premier opérande.(L'opération effectuée est : opérande1 ← opérande1 + opérande2)

ADD reg, imm	ADD BX, 1 ADD AX, [BX+2]
ADD mem, imm	
ADD reg, mem	
ADD mem, reg	
ADD reg, reg	

Exemple:

MOV AX, 45h

ADD AX, 7h

- AX vaut maintenant 4Ch

Instructions arithmétiques

Instructions d'addition : Exemples

- **ADD AH, [1100H]** : ajoute le contenu de la case mémoire d'offset 1100H à l'accumulateur AH

- **ADD AH, [BX]** : ajoute le contenu de la case mémoire pointée par BX à l'accumulateur AH

- **ADD byte ptr [1200H], 05H** : ajoute la valeur 05H au contenu de la case mémoire d'offset 1200H

Instructions arithmétiques

Instructions d'addition :

- **INC (INCrement) :** Cette instruction est en fait l'équivalent de "ADD ?, 1", cette instruction incrémente de 1 l'opérande.

Exemple: AX contient 18h (24 en décimal)

INC AX

- AX contient 19h

Instructions arithmétiques

Instruction de soustraction:

SUB (SUBstract) : Cette instruction soustrait le deuxième opérande au premier (L'opération effectuée est : opérande1 ← opérande1 - opérande2)

- Exemple :

AX contient 18h (24 en décimal)

SUB AX, 5h

AX = Dh

Instructions arithmétiques

Instruction de soustraction:

DEC (DECrement) : Cette instruction fait l'équivalent de "SUB ?, 1", cette instruction décrémente donc de 1 l'opérande.

- Exemple :

AX contient 18h (24 en décimal)

DEC AX

AX contient 17h

Instructions arithmétiques

Instruction de multiplication

MUL (MULtiply) : Cette instruction sert à multiplier un nombre Non Signé par un nombre source. Il y a 3 cas différents :

- Si l'opérande source à une taille d'un octet, AL est multiplié par le source, et le résultat dans AX,
- Si l'opérande source est un mot (16 bits), le registre AX est multiplié par celui-ci, et le résultat placé dans le registre 32 bits : DX:AX,
- Si l'opérande source est un double, le registre EAX est multiplié par l'oprande source, le résultat placé dans le registre 64 bits : EDX:EAX.

Exemple :

- AX contient 9, BL contient 3

MUL BL

- AX contient 27
- **IMUL** (signed IntegerMULtiply) : Cette instruction a la même fonction que l'instruction MUL ci-dessus mais elle support les nombres signés.

Instructions arithmétiques

Instruction de multiplication (Exemples)

MOV AL, 51h

MOV BL,32h

MUL BL

Résultat: **AX=51hx32h**

MOV EAX, 12345873h

MOV EBX, 12453216h

MUL EBX

Résultat: **EDX:EAX=12345873hx12453216h**

Instructions arithmétiques

Instruction de division

DIV (DIVise) : Cette instruction sert à diviser un nombre Non Signé par un nombre source; il y a 3 cas différents :

- Si l'opérande source a une taille d'un octet AX est divisé par celui-ci, le quotient dans AL et le reste dans AH,
- Si l'opérande source est un mot, le nombre 32 bits (DX:AX) est divisé par celui-ci, le quotient dans AX et le reste dans DX,
- Si l'opérande source est un double, le nombre 64 bits (EDX:EAX) est divisé par l'opérande source, le quotient dans EAX et le reste dans EDX,
- Pour les opérandes signés, on utilise **IDIV.**

Exemple :

- AX contient 18, BL contient 3

  DIV BL

- AX contient 9

Instructions logiques

Les instructions logiques sont au nombre de quatre (ET, OU, OU Exclusif, NON). Elles permettent de faire des opérations bit-à-bit sur des nombres binaires (c'est-à-dire en considérant chacun des bits indépendamment des autres, sans se soucier de la retenue).

Instruction AND (ET logique)

- équivalante d'une porte ET logique, elle réalise un ET entre 2 opérandes. Le résultat est placé dans le premier opérande.
- Exemple : On effectue un ET entre AH (9Dh) et BH (6Dh)

AH=1001 1101

BH=0110 1101

AND AH, BH

AH=0000 1101 (0Dh)

Instructions logiques

Instructions OR (OU logique) :

- Encore une autre porte logique, cette porte effectue un OU entre deux opérandes.

- Exemple : On effectue un OU entre AH (9Dh) et BH (6Dh)

AH=1001 1101

BH=0110 1101

OR AH, BH

AH=1111 1101 (FDh)

Instructions logiques

Instructions XOR (OU eXclusif)

- Cette instruction effectue un OU exclusif.
- Exemple : On effectue un XOR entre AH (9Dh) et BH (6Dh)

AH=1001 1101

BH=0110 1101

XOR AH, BH

AH=1111 0000 (F0h)

Instructions logiques

Instructions NOT : (Complément à 1)

- C'est l'équivalent de la porte logique NON, le principe de cette porte est d'inverser tous les bits.
- Exemple :

AH=1001 1101 1110 0110 (9DE6h)

NOT AX

AX=0110 0010 0001 1001 (6219h)

Instructions logiques

Instruction NEG: (Complément à 2)

- L'opération effectuée est : OP $\leftarrow \overline{OP}$+1
- Exemple:

```
mov AL,25          ; (11001 en binaire)
mov BL, 12         ; (1100 en binaire)
neg BL
add AL, BL
```

➔AL= 25+(-12)=13

Instructions logiques

Décalage logique vers la droite (Shift Right):

- Cette instruction décale l'opérande de n positions vers la droite.

SHR opérande,n

- Entrée d'un 0 à la place du bit de poids fort; le bit sortant passe à travers l'indicateur de retenue CF.
- Si le nombre de bits a décaler est supérieur a 1, ce nombre doit être placé dans le registre CL ou CX.

Exemple 1: décalage de AL de trois positions vers la droite :

```
mov CL,3
shr AL,CL
```

Exemple 2:

```
mov AL,11001011B
shr AL,1
```

Instructions logiques

Décalage logique vers la gauche (Shift Left):

- Cette instruction décale l'opérande de n positions vers la gauche.

SHL operande,n

- Entrée d'un 0 à la place du bit de poids faible; le bit sortant passe à travers l'indicateur de retenue CF.
- Si le nombre de bits a décaler est supérieur a 1, ce nombre doit être place dans le registre CL ou CX.

Exemple 1: décalage de AL de trois positions vers la droite :

```
mov CL,3
shl AL,CL
```

Exemple 2:

```
mov AL,11001011B
shl AL,1
```

L’instruction lea

Une instruction est fréquemment utilisée est **lea**, qui calcule l'adresse effective de l'opérande source et place le résultat dans l'opérande destination.

lea reg, mem

C'est la façon de mettre dans un registre l'adresse d'une variable. Par exemple, l'instruction :

lea esi, mavar

place dans esi l'adresse mémoire de la variable mavar. On pourrait également utiliser l'instruction:

mov esi, offset mavar

Exemple d’un premier programme

```
Data Segment
monmsg db "Hello, World! "
Data ends
Code segment
Assume Cs:Code, Ds: Data
start:
mov Ah, 09h      ; permet la sortie d'une chaîne de caractères sur  le périphérique de sortie standard
mov Dx, Offset monmsg    ;equivalante à lea Dx, monmsg
Int 21h
mov Ah, 4Ch      ; permet de mettre fin au programme et retourne un code de fin
Int 21h
Code ends
end start
```