# Arhitectura sistemelor de calcul

# Curs 11 - Instrucțiuni

MOV d, s — d <- s

PUSH s — depune s în stivă

POP d — extrage d din stivă

XCHG — d <-> s (swap), d și s L VALUES (registrii) (registru + memorie)

PUSHA — depune în stivă toți regiștrii

POPA — extrage din stivă toți regiștrii

PUSH F — depune în stivă EFLAGS

POP F — extrage din stivă EFLAGS

[registru segment] XLAT — AL <- DS:[EBX + AL]

cum ajung de la 178 val '178'

x va lua val fiecărei cifre din 178

x + '0'

cum ajung de la 2ab7h la '2ab7'

x va lua val fiecărei cifre din 2ab7h

x < 10 => x = x + '0'

x ≥ 10 => x = x + 'a' - 10

```
tabhex db '0123456789ABCDEF'
mov ebx tabhex
```

Vrem de exemplu cifra C

```
mov al, 12
xlat ; AL ← [EBX + AL] adică 12
```

## Instrucțiunea LEA

LEA REG, conținutul operand memorie

Ea transferă adresa operandului din memorie în reg

MOV eax, v = lea eax, [v]

```
lea eax, [ebx + v - 6] ✓
mov eax, ebx + v - 6   syntax error
```

## Instrucțiuni de setare a flagurilor

```
CLC
STC
CMC
CLD
STD
CLI
STI
```

} 7 instrucțiuni de setare a flagurilor

Inițial Data segment începe la offset-ul 00401000, linkeditorul ia decizia de acest tip și aceasta a fost o convenție.

## Exemple conținut de memorie

```
a1  db  0, 1, 2, 'xyz'        00 01 02 'x' 'y' 'z'
    db  300, "F" + 3          2C 'I'
```

300h = 12Ch

```
a2  times 3 db 44h            44 44 44
a3  times 11 db 5, 1, 3       05 01 03 ... 11 ori
a4  dw  a2 + 1, 'bc'          09 10 'b' 'c'
```

a2 + 1 = 00401008 + 1 = 00401009

```
a5  dd  a2 + 1, 'bcd'         09 10 40 00 'b' 'c' 'd'
a6  times 4 db '13'           '1' '3' '1' '3' '1' '3'
    dw '13'                   '1' '3' '1' '3' '1' '3'
a7  db  a2                    Syntax error adresa, pot să o pun doar pe 1/0
a8  dw  a2                    08 10
a9  dd  a2                    09 10 40 00
a10 dq  a2                    09 10 40 00 00 00 00 00
```

a11 db [a2] — syntax error valoarea lui a2 nu este cunoscută la momentul asamblării

a12 dw [a2] — syntax error ——11—— ✓ offset pe 16 biți

mov ax, v

Asamblorul va determina la momentul asamblării doar distanța variabilelor față de începutul segmentului.

Start:

Jmp Real_Start (instrucțiunea jmp ocupă 2 octeți) 00502000

a db 17 00502002

b dw 1234h 00502003

c dd 12345678h 00502005

Real_start:

offset de a la asamblare este 2

mov ax, c ✓ 00502005

mov dx, [c] ✓ dar se pune DS:[00502005]

mov edx, [CS : C] ✓ edx = 12345678

mov eax, '2345' ✓ eax = 5432

Valoarea asociată unui constante de tip string 'abcd' este de fapt 'dcba', adică este de fapt fix inversa.

## Constante de tip string

Ordinea de umplere a unei zone de memorie cu constante de tip string este aceea în care acestea apar deoarece valoarea asociată unei constante este în little-endian și reprezentând în little-endian se "anulează", directivele mai întâi generează spațiul

a7 dd '2345' '2' '3' '4' '5'

a7 dd '12345', 'abc' '1' '2' '3' '4' '5' 00

00 00 'a' 'b' 'c' 00

a7 dw '23', '45' '2' '3' '4' '5'

a7 dw '2345c' '2' '3' '4' '5' '6' 00

a8 db 'x' 'x'

a8 db "x" 'x'

ag dw '1', '2', '3' '1' 00 '2' 00 '3' 00

ag dw '123' '1' '2' '3' 00

mov dword [a], '2345' va apărea în OLLY DBG

mov dword ptr [DS:401000], 35343332

a7 dd '2345' '2' '3' '4' '5'

mov eax, '2345' eax = '5432' = 35343332

mov eax, 2345h eax = 2345h

mov eax, '12345678' eax = '1234'

↓

în memorie este '8' '7' ... '1'

a times 4 db '13' '1' '3' '1' '3' '1' '3' '1' '3'

a times 4 dw '13' '1' '3' '1' '3' '1' '3' '1' '3'

a times 2 dw '1', '3' '1' 00 '3' 00 '1' 00 '3' 00

a times 2 dd '1', '3' '1' 00 00 00 '3' 00 00 00

'1' 00 00 00 '3' 00 00 00

# Contorul de locații și aritmetica de pointeri

## segment data

```
a  db  1, 2, 3, 4        01 02 03 04
lg db  $ - a             04
lg db  $ - data          syntax error numele de seg ca ud de offset nu, relocată ca variabilele normale
lg db  a - data          syntax error —11—
lg dw  data - a          link error
db  a - $                FB      0 - 5
c equ a - $              0 - 6 = -6 = FA în tabela de constante

db  lg - a               04
db  a - lg               - 4 = FC
db  [$ - a]              syntax error

a   dd  eax              syntax error
a   dd  [eax]            syntax error
```

lg1 equ lg1 lg1 = 0 nasm bug

lg1 equ lg1 - a 0 - 0 = 0 nasm bug

a dw C-2 -8 F 8

b dd a - start syntax error

dd start - a merge nasm, pointer ptr scădere se face între adrese fan

dd start - start1 merge același sunt def în același segment scalar ptr adrese mean

segment code use 32

start:

mov ah, lg1 AH = 0

mov ah, C ah = -6

mov ch, lg syntax error offset nu începe

mov ch, lg - a ch = 4

mov ch, [lg - a] ✓ ul mai prob memory access violation

mov cx, lg - a cx = 4

mov cx, [lg - a] ✓

mov cx, $ - a syntax error dolar stu

conținut de locații a lui code segment și a lui altundeva $ generat aici și a altundeva

```
mov cx, a-$
mov ch, a-$
```

merge

syntax error offset cu număr pe byte

```
mov cx, start-$
```
✓

```
mov cx, $-start
```
✓

```
mov ch, $-start
```
✓

```
mov ch, start-$
```
✓

```
mov cx, a-start
```
✓ ptr a definit altundeva start aici

```
mov cx, start-a
start1:
```
syntax error

```
mov ah, a+b
```
✓ adunare de scalari

```
mov ax, b+a
```
✓ a+b = (a-$$) + (b-$$) scalari

```
mov ax, [b+a]
```
syntax error formula de calc offset

Concluzii

et1 - et2 sunt acceptate dacă:

sunt definite în același segment

et1 aparține unui segment diferit față de cel unde apare et2 și este o scădere de adrese FAR => rezultatul este pointer

Numele unui segment este asociat cu adresa segmentului de memorie dar aceasta nu este valabilă în momentul asamblării

Dacă avem mai mulți operanzi pointeri

mov bx, [V2 - V2 - V] Syntax error

scalar - pointer

mov bx, V3 - V2 - V1 - V = mov bx, (V3 - V2) - (V1 + V)

scalar scalar

A[7] = *(A+7) = *(7+A) = 7[A]