# Arhitectura sistemelor de calcul

# Instrucțiuni de conversie (distructive)

CBW
CWD
CWDE
CDQ
(extensie de semn)

| | | nu poate fi const |
|---|---|---|
| | REG | REG/MEM |
| MOVZX | d | , s |
| MOVSX | d | , s |

MOVZX d, s — zeroorizare fără semn

MOVSX d, s — zeroorizare cu semn

MOV ah, c8h ✓ AH = C8h

MOVSX ebx, ah ✓ EBX = FF FF FF C8

MOVZX ebx, ah ✓ EBX = 00 00 00 C8

MOVSX ax, [v] ✓ dim didusă din ax

MOVZX eax, [v] syntax error operation size not specified, 2 pos. W/D

MOVSX eax, word/byte [v] ✓

MOVSX eax, dword [v] syntax error size-uri egale

MOV bh, 1 ✓

MOVSX ax, bh ✓ 00 01

# Deplasări de biți

| Instrucțiuni de depl logică | Instrucțiuni de depl arit |
|---|---|
| SHL | SAL |
| SHR | SAR |

| Instrucțiuni de rotire fără carry | Instrucțiuni de rotire cu carry |
|---|---|
| ROL | RCL |
| ROR | RCR |

x = abcdefgh

shl x → x = bcdefgh0 CF = a

shr x → x = 0abcdefg CF = h

sal x → x = bcdefgh0 CF = a

sar x → x = aabcdefg CF = h - ține cont de semn

rol x, 1 → x = bcdefgha CF = a

ror x, 1 → x = habcdefg CF = h

rcl x, 1 → x = bcdefgh0 CF = a

CF = a rcr x, 1 → x = aabcdefg CF = h

# Salturi condiționate

JMP operand

CALL operand = salvarea adresei de ret + jump

RET [n] = extragem adresa de ret + jump la acea adresă de ret și adaugă n la stivă

```
mov eax, etich
.
jmp eax
etich:
```

```
salt dd dest
.
jmp [salt]
dest:
```

CMP d,s face comparare fictivă (scădere)

TEST d,s face and fictiv

Unsigned

JL - jump less

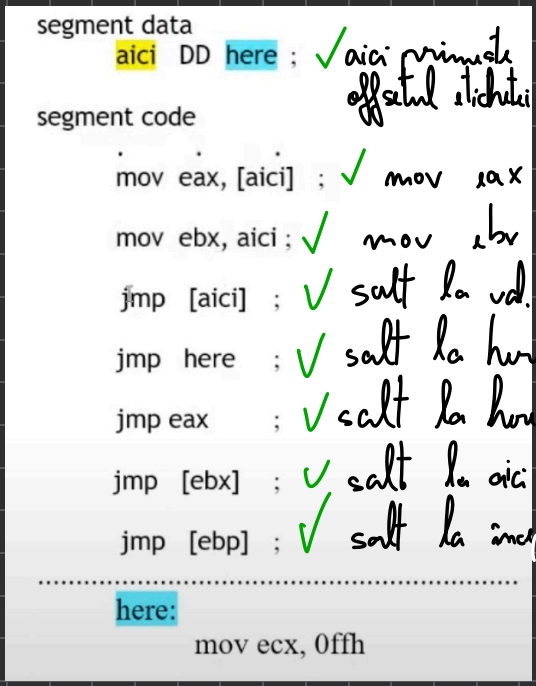
JA - jump above

signed

JB - jump below

JG - jump greater

Distanța între loop și label trb să fie <= 127 octeți pt că avem un jump short pt loop




jmp aici  CS:EIP, EIP = offset

jmp [var_mem]  jmp dword ptr DS:[00401000]

jmp [EBX]  jmp dword ptr DS:[EBX]

jmp [EBP]  jmp dword ptr SS:[EBP]

Un salt near jmp [SS:ebx+12] se va efectua tot în același segment de cod adică la CS:EIP, salt near.

Dacă facem un salt far facem urm. instr.
jmp far [SS: ebx + 12] și înseamnă că și
CS este populate cu SS și offset cu ebx + 12
deci jmp [SS: ebx + 12] <=> mov EIP, [SS: ebx + 12]
jmp far [SS: ebx + 12] <=> mov EIP dword, [SS: ebx + 12]
mov CS word, [SS: ebx + 12 + 4]

mov CS: EIP [memory] → EIP = 56 A4 8C 7B

CS = 47 D4

memorie: 56 A4 8C 7B 47 D4

```
segment data use32 class=DATA

    a db 1,2,3,4
    start3:
      mov edx, eax

segment code use32 class=code
    start:
          mov edx, eax
          jmp start2  - ok – salt NEAR  - JMP  00403000 (offset code segment = 00402000)
          jmp start3 – ok – salt NEAR –  JMP  00401004 (offset data segment = 00401000)
          jmp far start2  - Segment selector relocations are not supported in PE file
          jmp far start3  - Segment selector relocations are not supported in PE file
          add eax,1
    final:
      push    dword 0
      call    [exit]

segment code1 use32 class=code

    start2:
       mov eax, ebx

    push dword 0
    call [exit]
```

Salturi far direct nu merg, doar indirect, adică cu forma offset deoarece nu putem direct pt o adresă pe 32 biți și la fel și un registru, dar folosim forma offset accesăm 48 biți