Reprezentarea little-endian a numerelor în memorie. Salt necondiționat, condiționat. Operații pe șiruri.

* Reprezentarea numerelor in memorie (!!! Doar in memorie nu si in registrii procesorului)
  + Numerele reprezentate pe mai mult de un octet se reprezinta folosind principiul little endian: cel mai putin semnificativ octet se afla la adresa cea mai mica

A dd 12345678h

B dq 1122334455667788h

A B

| |

| 78h | 56h | 34h | 12h | 88h | 77h | 66h | 55h | 44h | 33h | 22h | 11h |

A+1 A+2 A+3 B+1 B+2 B+3 B+4 B+5 B+6 B+7

Octetul low al cuvantului high din A se afla la adresa: A+2

Octetul low al cuvantului high al dublucuvantului low din B se afla la adresa: B+2

Ex. X = a\*b+c\*d, a,b,c,d – cuvinte si interpretarea fara semn

|  |  |
| --- | --- |
| Data segment | Code segment |
| A dw 10; 000Ah  B dw 1234h  C dw 0fa23h  D dw 80; 0050h  X dd 0; 0000 0000h | Mov ax, [a]; ax = 000ah  Mul word [b]; dx:ax = a\*b = …  Mov [x], ax  Mov [x+2], dx  Mov ax, [c]; ax = 0fa23h  Mul word [d]; dx:ax = c\*d  Add [x], ax  Adc [x+2], dx |

A B C D X

| | | | |

| 0Ah | 00h | 34h | 12h | 23h | 0fah | 50h | 00h | 00h | 00h | 00h | 00h

**Instructiuni conditionale de salt**

* Sunt similare cu instructiunea IF
* O instructiune “IF” este compusa din 2 parti:
  + O instructiune de comparatie prin care se determina ordinea dintre operanzi

TEST dest, sursa; dest AND sursa fictiv si afecteaza flag-urile in mod corespunzator

CMP dest, sursa; dest – sursa fictiv si afecteaza flag-urile in mod corespunzator

* + O instructiune de salt

Prin instructiunea de salt se verifica valorile flag-urilor si in functie de valorile lor se efectueaza un salt in cadrul programului la un offset ( o adresa) definit de o eticheta (se modifica valoarea registrului EIP. EIP va indica adresa etichetei si nu adresa urmatoarei instructiuni aflate imediat dupa instructiunea de salt)

Instructiuni conditionale de salt care interpreteaza numerele fara semn

J = jump; B = below; A = above ; E = equal; N = not

JB eticheta; jump if below la eticheta (daca dest<sursa atunci se sare la eticheta)

JBE eticheta ; dest <= sursa

JNB eticheta ; dest >= sursa

JNBE eticheta ; dest > sursa

JA eticheta ; jump if above la eticheta (daca dest > sursa atunci se sare la eticheta)

JAE eticheta ; dest >= sursa

JNA eticheta ; dest <= sursa

JNAE eticheta; dest < sursa

Instructiuni conditionale de salt care interpreteaza numerele cu semn

J = jump; L = less than; G = greater than; E = equal; N = not

JL eticheta; jump if less than la eticheta (daca dest < sursa atunci se sare la eticheta)

JLE eticheta; dest <= sursa

JNL eticheta ; dest >= sursa

JNLE eticheta ; dest > sursa

JG eticheta; dest > sursa

JGE eticheta; dest >= sursa

JNG eticheta; dest <= sursa

JNGE eticheta; dest < sursa

Instructiuni conditionale de salt care se uita la un singur flag

JC eticheta; jump if CF = 1

JNC eticheta

JO eticheta; jump if OF=1

JNO eticheta

JP eticheta; jump if PF = 1

JNP eticheta

JS eticheta; SF=1?

JNS eticheta

JZ eticheta; ZF=1

JNZ eticheta

Instructiuni conditionale de salt care iau in considerare valori din registrii

JCXZ eticheta; jump if CX=0

JECXZ eticheta; jump if ECX=0

Instructiune de salt neconditionat

JMP eticheta

**Siruri de octeti / cuvinte / dublucuvinte**

* Pentru a lucra cu siruri de octeti / cuvinte / dublucuvinte sunt necesare urmatoarele:

Data segment

a dw 5

b dw 1,2,3,4,5,6,7,8,9,10

c db 1,2,3,4,5,6,7,8

d dd 1,2,3,4,5

Code segment:

Mov ax, [a]

Mov bx, [b]

Mov bx, [b+2]

Mov bx, [b+4]

…

Mov al,[c]

Mov al,[c+1]

Mov al,[c+2]

….

Mov eax, [d]

Mov eax, [d+4]

Mov eax, [d+8]

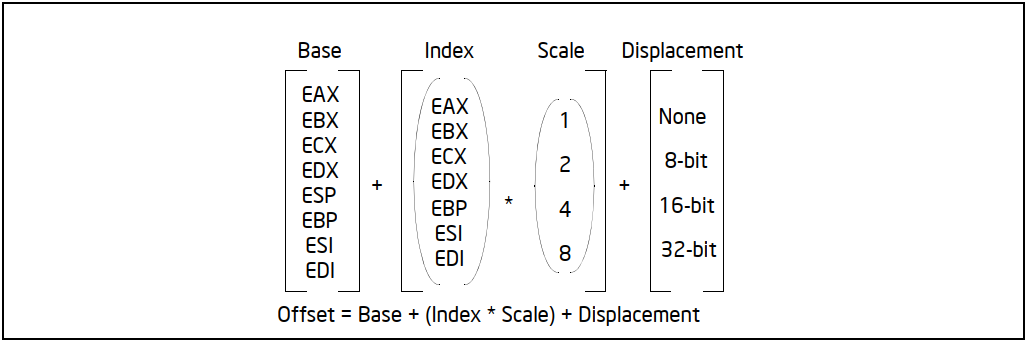
…

* Numele unei variabile este o adresa constanta = adresa variabilei in memorie (OFFSET)
* O specificare completa a adresei se face folosind doua numere (selectorul de segment : offset )
* Selectorul de segment este un numar pe 16 biti
* Offset este un numar pe 32 de biti

Mov eax, a; muta in eax adresa lui a

Mov eax, [a]; muta in eax dublucuvantul aflat la adresa lui a

Definirea unui offset se face folosind 4 valori: baza, index, scala si constanta



Mov ax, [a]; doar constanta

Mov ax, [eax]; doar baza sau index

Mov ax, [a+eax+ebx]; constanta, baza si index

Mov ax, [a+eax+ebx\*2]; constanta, baza eax si indexul ebx cu scala 2

Mov ax, [a+esp+eax]; constanta, baza esp si index eax cu scala 1

Ex. Se da un sir de octeti care contine litere mici. Sa se construiasca un sir nou de octeti care sa contina litere mari coresp literelor mici din sirul initial:

Varianta 1:

|  |  |
| --- | --- |
| Data segment | Code segment |
| A db ‘a’,’b’,’c’,’d’,’e’,’f’ ; sau a db ‘abcdef’  L equ $ - A; $ = contor de locatii  B TIMES L db 0 | Start:  Mov esi,0  Transforma:  Mov al, [A+ESI]  Sub al, ‘a’ - ‘A’  Mov [B+ESI], al  INC ESI  CMP ESI, L  JB Transforma  Push dword 0  Call [exit] |

A B

| |

| ‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | 00h | 00h | 00h | 00h | 00h | 00h |

Varianta 2:

|  |  |
| --- | --- |
| Data segment | Code segment |
| A db ‘a’,’b’,’c’,’d’,’e’,’f’ ; sau a db ‘abcdef’  L equ $ - A; $ = contor de locatii  B TIMES L db 0 | Start:  Mov esi, a  Mov edi, b  Mov ecx, L  Transforma:  Mov al,[esi]  Sub al, ’a’-’A’  Mov [edi], al  Inc esi  Inc edi  Dec ecx  Cmp ecx,0  Ja Transforma  Push dword 0  Call [exit] |

**Instructiuni de ciclare**

LOOP, LOOPE / LOOPZ si LOOPNE / LOOPNZ

LOOP – reia executia blocului de comenzi atata timp cat ECX != 0

- se decrementeaza ECX + se face testul (ECX != 0) + se sare la eticheta daca este cazul

Varianta 3:

|  |  |
| --- | --- |
| Data segment | Code segment |
| A db ‘a’,’b’,’c’,’d’,’e’,’f’ ; sau a db ‘abcdef’  L equ $ - A; $ = contor de locatii  B TIMES L db 0 | Start:  Mov esi, a  Mov edi, b  Mov ecx, L  JECXZ Final  Transforma:  Mov al,[esi]  Sub al, ’a’-’A’  Mov [edi], alx  Inc esi  Inc edi  LOOP Transforma  Final:  Push dword 0  Call [exit] |

Mov ECX, 5

Bucla:

Loop Bucla

=> ecx =4 ,3, 2, 1, 0 => 5 pasi

Mov ecx, 5

Bucla:

Dec ecx

Loop bucla

=> P0: ecx = 5

P1: ecx = 4, 3

P2: ecx =2,1

P3: ecx = 0, -1 (2^32-1)

…. => ciclu infinit

Mov ecx, 6

Bucla:

Dec ecx

Loop bucla

P0 ecx = 6

P1 ecx = 4

P2 = ecx = 2

P3 ecx = 0 => 3 pasi

Mov ecx, 12h

Bucla:

Shr ecx,1

Loop bucla

De cate ori se executa?

p0 ecx = 000000010010b

p1 ecx = 000000001001b, 000000001000b

p2 ecx = 000000000100b, 000000000011b

p3 ecx = 000000000001b, 0 => 3 pasi