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

Reprezentarea little-endian a numerelor in memorie: in memoria calculatorului numerele care sunt reprezentate pe mai mult de un octet sunt reprezentate respectand formatul little-endian (cel mai putin semnificativ octet se afla la cea mai mica adresa)

A db 12h

B dw 1234h

C dd 12345678h

D dq 1122334455667788h

A B C D

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

La ce adresa este octetul high al cuvantului high din dublucuvantul low al lui D?

Quadword-ul D : adresa D

Dublucuvantul low din D: adresa D

Cuvantul high din dublucuvantul low din D: adresa D+2

Octetul high al cuvantului high din dublucuvantul low al lui D: D+3

MOV AL, [D+3]; octetul high ….

Ex 1. x = a\*b+c\*d, a,b,c,d sunt cuvinte + I.f.s

|  |  |
| --- | --- |
| Data segment | Code segment |
| A dw 1234h  B dw 5678h  C dw 4321h  D dw 8765h  X resd 1 | Mov AX, [a]  Mul word [b] ; DX:AX  Mov [x], AX  Mov [x+2], DX  Mov AX, [c]  Mul word [d]  Add [x], AX  Adc [x+2], DX |

A B C D X

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 34h | 12h | 78h | 56h | 21h | 43h | 65h | 87h |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

INSTRUCTIUNI DE SALT CONDITIONAT

* Similaritate cu instructiunea “IF”
  + Instructiune de comparare

TEST dest, sursa; dest AND sursa fictiv si seteaza flag-urile in mod coresp.

CMP dest, sursa; dest – sursa fictiv si seteaza flag-urile in mod coresp.

* + Instructiune de salt conditionat

**Instructiuni de salt conditionat care interpreteaza numerele fara semn**

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

JA eticheta; > daca dest > sursa atunci se sare la eticheta

JAE eticheta; >=

JNA eticheta; <=

JNAE eticheta; <

JB eticheta; <

JBE eticheta; <=

JNB eticheta; >=

JNBE eticheta; >

**Instructiuni de salt conditionat care interpreteaza numerele cu semn**

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

JG eticheta; >

JGE eticheta; >=

JNG eticheta; <=

JNGE eticheta; <

JL eticheta; <

JLE eticheta; <=

JNL eticheta; >=

JNLE eticheta; >

**Instructiuni de salt conditionat care iau in considerare un singur flag**

JC eticheta; jump if CF=1

JNC eticheta

JE eticheta sau JZ eticheta

JNE eticheta sau JNZ eticheta

JS eticheta

JNS eticheta

JP eticheta

JNP eticheta

JO eticheta

JNO eticheta

**Instructiuni care verifica valoarea din CX / ECX**

JCXZ eticheta; jump if CX == 0 la eticheta

JECXZ eticheta; jump if ECXZ ==0 la eticheta

**Instructiuni de salt neconditionat**

JMP eticheta

O specificare completa de adresa se face folosind selectorul segmentului (o adresa pe 16 biti) si un offset (o adresa pe 32 biti)

A db 12h

….

MOV AL, [A]

**Operatii pe siruri**

B db 1,2,3,4,5,6

C dw 1,2,3,4,5,6

D dd 1,2,3,4,5,6

….

MOV AL, [B]

MOV AL, [B+1]

MOV AL, [B+2]

…

MOV AX, [C]

MOV AX, [C+2]

MOV AX, [C+4]

…

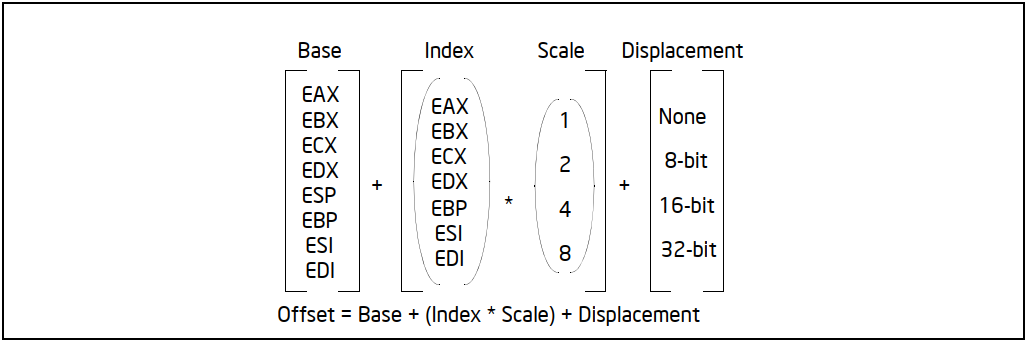
MOV EAX, [D]

MOV EAX, [D+4]

MOV EAX, [D+8]

…

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



MOV AX, [D] ; doar constanta

MOV AX, [EAX]; folosesc baza sau index-ul EAX

MOV AX, [EAX\*2]; folosesc index-ul EAX cu scala 2

MOV AX, [a + ESP]; folosesc constanta a si baza ESP

MOV AX, [a+4+EBX+EBX\*2]; folosesc constanta, baza, index, scala

Ex 2. Se da un sir de octeti care contin litere mici. Construiti sirul de octeti care se obtine din sirul initial, prin convertirea literelor mici in litere mari

Varianta 1

|  |  |
| --- | --- |
| Data segment | Code segment |
| S db ‘a’, ‘b’, ‘c’, ‘d’, ‘e’ ; s db ‘abcde’  L equ $-S  ;D db 0,0,0,0,0,0  ;D resb L  D times L db 0 | Start:  MOV ESI, 0  Transforma:  MOV AL, [S + ESI]  SUB AL, ‘a’-’A’; ‘a’-’a’+’A’ = ‘A’  ; ‘b’-’a’+’A’=1+’A’=’B’  MOV [D + ESI], AL  INC ESI  CMP ESI, L  JB Transforma  Push dword 0  Call [exit] |

S D

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

Varianta 2

|  |  |
| --- | --- |
| Data segment | Code segment |
| S db ‘a’, ‘b’, ‘c’, ‘d’, ‘e’ ; s db ‘abcde’  L equ $-S  ;D db 0,0,0,0,0,0  ;D resb L  D times L db 0 | Start:  MOV ESI, S  MOV EDI, D  MOV ECX, L  Transforma:  MOV AL, [ESI]  SUB AL, ‘a’ - ‘A’  MOV [EDI], AL  INC ESI  INC EDI  DEC ECX  CMP ECX, 0  JNZ Transforma  PUSH dword 0  CALL [exit] |

**Instructiuni de ciclare**

LOOP eticheta <=> DEC ECX + JNZ eticheta <=> se reia blocul de instructiuni si ciclul continua atata timp cat ECX este diferit de 0

LOOPE eticheta sau LOOPZ eticheta

LOOPNE eticheta sau LOOPNZ eticheta

Varianta 3

|  |  |
| --- | --- |
| Data segment | Code segment |
| S db ‘a’, ‘b’, ‘c’, ‘d’, ‘e’ ; s db ‘abcde’  L equ $-S  ;D db 0,0,0,0,0,0  ;D resb L  D times L db 0 | Start:  MOV ESI, S  MOV EDI, D  MOV ECX, L  JECXZ Final  Transforma:  MOV AL, [ESI]  SUB AL, ‘a’ - ‘A’  MOV [EDI], AL  INC ESI  INC EDI  LOOP Transforma  Final:  PUSH dword 0  CALL [exit] |

MOV ECX, 5

Bucla:

LOOP Bucla

De cate ori se executa bucla?

P0: ecx=5

P1: ecx=4 ? ECX == 0 NU

P2: ecx=3 ? ECX == 0 NU

P3: ecx=2 ? ECX == 0 NU

P4: ecx=1 ? ECX ==0 NU

P5: ecx=0 ? ECX ==0 DA => 5 pasi

MOV ECX, 5

Bucla:

DEC ECX

Loop Bucla

De cate ori se executa bucla?

P0: ECX = 5

P1: ecx = 4 , ecx = 3 ? ECX == 0 NU

P2: ecx = 2 , ecx = 1 ? ECX == 0 NU

P3: ecx = 0 , ecx = -1? ECX == 0 NU

… => ciclu infinit

MOV ECX, 12h

Bucla:

SHR ECX,1

Loop Bucla

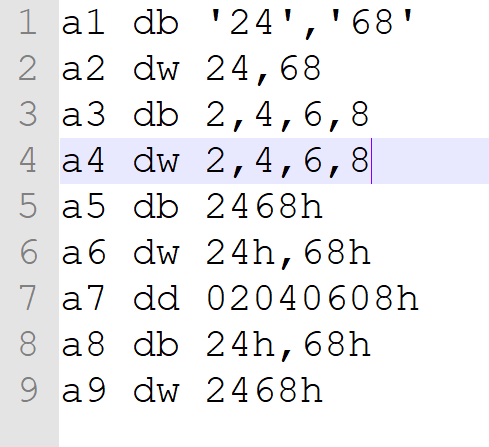
De cate ori se executa bucla?

P0: ECX = 0001 0010b

P1: ECX = 0000 1001b; ECX = 0000 1000b; ECX == 0 ? NU

P2: ECX = 0000 0100b; ECX = 0000 0011b; ECX == 0 ? NU

P3: ECX = 0000 0001b; ECX = 0000 0000b; ECX == 0 ? DA



Reprezentarea in memorie a data segment-ului de mai sus

a1 db '24','68' ; |'2'|'4'|'6'|'8'|

a2 dw 24,68 ; |18h|00h|44h|00h|

a3 db 2,4,6,8 ; |02h|04h| 06h|08h|

a4 dw 2,4,6,8 ;|02h|00h|04h|00h|06h|00h|08h|00h

a5 db 2468h ; eroare deoarece nu incape pe un byte

a6 dw 24h,68h; |24h|00h|68h|00h

a7 dd 02040608h; |08h|06h|04h|02h

a8 db 24h,68h; |24h|68h|

a9 dw 2468h ;|68h|24h