**Seminar 4:**

**Instructiuni de salt neconditionat/conditionat. Instructiuni repetitve. Siruri.**

Intr-un program, codul se executa de sus in jos.

Daca dorim sa schimbam ordinea in care se executa instructiunile dintr-un program, se folosesc instrunctiuni de salt:

1. Instructiuni de salt necoditionat

JMP eticheta

|  |  |
| --- | --- |
| mov al, 10  jmp scadere  adunare:  add al, 2  scadere:  sub al, 5  jmp adunare  final: ... | mov al, 10  jmp scadere  adunare:  add al, 2  jmp final  scadere:  sub al, 5  jmp adunare  final: ... |

2. Instructiuni de salt conditionat

Instructiuni similare cu instructiunea “IF”

In asamblare o instructiune “IF” este compusa din:

O instructiune de comparare

CMP dest, sursa; dest – sursa fictiv => seteaza flag-urile in mod coresp, ZF, CF,SF, OF

O instructiune de salt conditionat

**2.a. Instructiuni care interpreteaza numerele fara semn:**

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

CMP dest, sursa

JB eticheta; <

adica: if dest below source then jump la eticheta <=> daca dest < sursa atunci sari la eticheta

JBE eticheta; <=

JNB eticheta; >=

JNBE eticheta; >

JA eticheta; >

JAE eticheta; >=

JNA eticheta; <=

JNAE eticheta; <

Exemplu:

|  |  |
| --- | --- |
| **if a>b then rez=a-b**  **else rez=b-a** | mov al, [a]  mov bl, [b]  **cmp al, bl**  **JA et\_then ;jump if al is above bl**  **JBE et\_else ; jump if al is below or equal than bl**  et\_then:  sub al, bl  mov rez, al  jmp final  et\_else:  sub bl, al  mov rez, bl  final: |

**2.b. Instructiuni care interpreteaza numerele cu semn**

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

CMP dest, sursa

JL eticheta; <

 adica if dest less sursa then jump la eticheta <=> daca dest < sursa atunci sari la eticheta

JLE eticheta; <=

JNL eticheta; >=

JNLE eticheta; >

JG eticheta; >

JGE eticheta; >=

JNG eticheta; <=

JNGE eticheta; <

Exemplu:

|  |  |
| --- | --- |
| **if a>b then rez=a-b**  **else rez=b-a** | mov al, a  mov bl, b  **cmp al, bl**  **JG et\_then ;jump if al is greater bl**  **JLE et\_else ; jump if al is less or equal than bl**  et\_then:  sub al, bl  mov rez, al  jmp final  et\_else:  sub bl, al  mov rez, bl  final: |

**Instructiuni care verifica continutul din flag-uri**

JC eticheta - adica jump la eticheta daca CF este setat (adica daca CF=1)

JNC eticheta - adica jump la eticheta daca CF NU este setat (adica daca CF=0)

JP eticheta

JNP eticheta

JO eticheta

JNO eticheta

JS eticheta

JNS eticheta

JZ eticheta

JNZ eticheta

**Exemplu:**

**daca vrem sa aduna continutul din combinatia de registri:**

**dx:ax+**

**cx:bx**

jc curatacf

jnc farasteregere

curatacf:

CLC ; clear CF , pune 0 in CF

jmp adunare:

farasteregere:

adunare:

add ax, bx ; ax=ax+bx si se seteaza CF daca avem trans

JC adunacucarry

JNC adunafaracarry

adunacucarry:

adc dx, cx

jmp final

adunafaracarry:

add dx, cx

final:

**Instructiuni care verifica valoarea din registrul ECX**

**JECXZ eticheta** adica jump la eticheta daca valoarea din ECX este zero

**Instructiuni repetitive** – se foloseste instructiunea **Loop**

mov ECX, nr\_de\_repetari

eticheta:

Secventa de instructiuni de repetat

LOOP eticheta

**Se efectueaza mai intai DEC ECX** si apoi se verifica daca ECX <> 0

Cand valoarea din ECX ajunge la 0 se termina executia repetata

 De exemplu:

|  |  |  |
| --- | --- | --- |
| Mov ax, 0  MOV ECX, 5  Repeta:  add ax, 1  LOOP Repeta | Mov ax, 0  MOV ECX, 5  Repeta:  add ax, 1  sub Ecx, 1 (;sau dec Ecx)  cmp Ecx, 0  JNE Repeta | Mov ax, 0  MOV ECX, 5  Mov bx, 0  Repeta:  add ax, 1  add bx, 1  cmp bx, ecx  JNE Repeta |
| La final valoarea din ax va fi: 5 | | |

Operatii pe siruri:

Variabile in memorie:

B db -1

B in memorie:

|  |  |
| --- | --- |
| FF | Valoarea lui B |
| B+0 | Adresa lui B |

A dd 12345678h ; a in data segment si a in registrii

A in memorie; conform little-endian

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 78 | 56 | 34 | 12 | Bytes lui A |
| A+0 | A+1 | A+2 | A+3 | Adresele lui A |

mov ax, word [a+0]; ax =5678

mov bx, word [ a+1] ; bx = 3456

**Siruri:** succesiune de elemente de acelasi tip (bytes, words, doublwords)

Exemplu:

1. Siruri de bytes:

Variabila sirb definita in data segment:

sirb db 12h, 1Ah, 2bh

sirb in memorie:

|  |  |  |  |
| --- | --- | --- | --- |
| 12 | 1A | 2B | Bytes care compun sirul sirb |
| Sirb+0 | Sirb+1 | Sirb+2 | Adresele lui sirb |

**Lungimea unui sir** se determina folsind directiva equ si contorul de locatii.

Contorul de locatii ($) este o constanta cu valoare calculata la momentul asamblarii si ea ne determina cati bytes s-au generat in memorie.

lungime\_sirb EQU $-sirb

**Definirea unui sir rezultat:** alocare spatiu in memorie pentru sirul rezultat:

rez resb lungime\_sirb

sau folosind directiva times

rez times lungime\_sirb db 0 ; rez times lungime\_sirb db -1

sau

rez db 0, 0, 0; rez db -1, -1, -1

**Accesarea unui element din sir:**

Este necesar utilizarea unui reg index, cu rol de index in parcurgerea elementelor din sir:

mov esi, 0

mov al, [s+esi]

inc esi

**Salvarea unui element in sir**

Este necesar utilizarea unui reg index, cu rol de index in parcurgerea pozitilor din sir:

mov edi, 0

mov [rez+edi], al

inc edi

**Problema:**

**Se da un sir S de octeti/bytes (numere intregi). Sa se salveze in sirul D, toate numerele din sirul S care sunt pozitive si divizibile cu 7 din sirul initial.**

**Etape:**

**parcurgem**

**pentru fiecare elem: >0 si apoi este div cu 7**

*-data segment*

s db 7, 14, -7, -13, 10, 2

len\_sir equ $-s ; len\_sir = 6 ; calcul lungime

rez times lungime\_sirb db 0 ; definire sir rezultat

*code segment*

; pregatire lungime sir in reg contor pentru instr repetitiva: loop

mov ecx, len\_sir

mov esi, 0 ; pentru a parcurge sirul initial s

mov edi, 0; pentru parcurgere sir rezultat rez

Repeta:

mov al, [s+esi] ; in primul pas, al=[s+0]=7

cmp al, 0 ; compratie intre al si 0

JGE pozitive

JL negative

pozitive:

cbw ; ax = al extins la word pentru impartire

`mov bl, 7

idiv bl ; ax/bl = al – cat si in ah – rest

cmp ah, 0

JE adauga

JNE nu\_adauga

adauga:

~~mov [rez+edi], [rez+esi] nu putem din cauza lui mov~~

mov al, [s+esi]

mov [rez+edi], al

inc esi

inc edi

jmp final

nu\_adauga:

inc esi

jmp final

negative:

inc esi

final:

Loop Repeta

2. Siruri de worduri:

sirw dw 1234h, 5678h, 1A2Bh

sirw in memorie: element cu element cf little-endian

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 34 | 12 | 78 | 56 | 2b | 1a | Bytes care compun sirul sirw |
| Sirw+0 | Sirw+1 | Sirw+2 | Sirw+3 | Sirw+4 | Sirw+5 | Adresele lui sirw |

**Lungimea unui sir** se determina folsind directiva equ si contorul de locatii.

Contorul de locatii ($) este o constanta cu valoare calculata la momentul asamblarii si ea ne determina cati bytes s-au generat in memorie.

lungime\_sirw EQU ($-sirw)/2 ; lungime\_sirw = 3

**Definirea unui sir rezultat:** alocare spatiu in memorie pentru sirul rezultat:

rezw resw lungime\_sirw

sau folosind directiva times

rezw times lungime\_sirw dw 0

**Accesarea unui element din sir:**

Este necesar utilizarea unui reg index, cu rol de index in parcurgerea elementelor din sir:

mov esi, 0

mov ax, [s+esi]

add esi, 2

**Salvarea unui element in sir**

Este necesar utilizarea unui reg index, cu rol de index in parcurgerea pozitilor din sir:

mov edi, 0

mov [rezw+edi], ax

add edi, 2

**Problema: Se da un sir de worduri, initializat cu valori in baza 10. Sa se parcurga sirul in ordine inversa si sa se determina suma wordurilor care au ultima cifra egala cu cifra 8.**

etape:

initializare esi, lung-1,

accesare word

word mod 10, rest contine ultima cifra, verificare ultima cifra = 8

daca este indeplinita conditia, il punem in suma

decrementare esi

data segment

sw dw 18, 28, 38, 40, 50

lungime\_sw EQU ($-sw)/2 ; LUNGIME\_SW = 5

suma dw 0

**b0 b1 b2 b3 b4 b5 b6b7 b8b9**

**w0 w1 w2 w3 w4**

**18 28 38 40 50**

code segment

mov ecx, lungime\_sw

mov esi, lungime\_sw\*2-2

Repeta:

mov ax, [sw+esi]

mov bl, 10

div bl ; al–cat ah-rest

cmp ah, 8

JE adaugaSuma

JNE nuadaugaSuma

adaugaSuma:

mov ax, [sw+esi]

add [suma], ax

; trebuie sa mergem la urm word, de la dr la st

sub esi, 2

JMP FINAL

nuadaugaSuma:

sub esi, 2

FINAL:

LOOP Repeta

3. Siruri de doubleworduri:

sird dd 12345678h, 1A2B3C4Dh

sird in memorie: element cu element cf little-endian

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 78 | 56 | 34 | 12 | 4d | 3c | 2b | 1a | Bytes care compun sird |
| Sird+0 | +1 | +2 | +3 | +4 | +5 | +6 | Sird+7 | Adresele lui sird |

**Lungimea unui sir** se determina folsind directiva equ si contorul de locatii.

Contorul de locatii ($) este o constanta cu valoare calculata la momentul asamblarii si ea ne determina cati bytes s-au generat in memorie.

lungime\_sird EQU ($-sird)/4

**Definirea unui sir rezultat:** alocare spatiu in memorie pentru sirul rezultat:

rezd resd lungime\_sird

sau folosind directiva times

rezd times lungime\_sird dd 0

**Accesarea unui element din sir:**

Este necesar utilizarea unui reg index, cu rol de index in parcurgerea elementelor din sir:

mov esi, 0

mov eax, [s+esi]

add esi, 4

**Salvarea unui element in sir**

Este necesar utilizarea unui reg index, cu rol de index in parcurgerea pozitilor din sir:

mov edi, 0

mov [rez+edi], eax

add edi, 4

Problema: Se dau doua siruri de doubleworduri de aceasi lungime.

Sa se determine intr-un nou sir rezultat diferenta dintre elementele de pe pozitiile similare din cele doua siruri.

data segment

s1 dd 10, 20, 30

lung\_s1 equ ($-s1)/4 ; 1 dd = 4 bytes

s2 dd 1, 2, 3

lung\_s2 equ ($-s2)/4 ; 1 dd = 4 bytes

rez rezd times lung\_s1 dd 0 ; 9, 18, 27

code segment

mov ecx, lung\_s1

mov esi, 0; pentru parcurgere s1 si s2

mov edi, 0 ; pentru parcugere rez

repeta:

mov eax, [s1+esi]

mov ebx, [s2+esi]

sub eax, ebx

mov [rez+edi], eax

add esi, 4

add edi, 4

loop repeta