8. Setul de instrucţiuni 8086 (III)

8.1 Instrucţiuni pentru şiruri

Operaţii primitive

Reprezintă un set de operaţii pe octet sau pe cuvânt asupra unor locaţii escries e în memorie. Deci operanzii sunt locaţii de memorie pe 8 sau 16 biţi. Distincţia între instrucţiunile pe octet sau pe cuvânt se face prin adaugarea sufixelor B (byte) sau W (word).
Instrucţiunile pentru şiruri nu au operanzi, regiştrii DS:SI sunt folosiţi drept adresă sursă iar ES:DI adresă destinaţie. Parcurgerea şirului se poate face atât înainte cât şi înapoi, deci regiştrii SI, DI sunt actualizaţi fie prin incrementare fie prin decrementare. Sensul de parcurgere al şirului (în ordine crescătoare sau descrescătoare) este determinat de starea flagului D din PSW. Dacă D=0 adresele sunt incrementate cu 1, dacă operaţia este la nivel de octet şi cu 2, dacă e la nivel de cuvânt. Dacă D=1 adresele sunt decrementate în mod similar. Flagul D poate fi setat, respectiv şters prin utilizarea instrucţiunilor fără operanzi STD (Set Direction) respectiv CLD (Clear Direction).

Instrucţiunile de copiere sau transfer (Move String) **MOVSB, MOVSW –** transferă în ((ES :DI)) conţinutul locaţiei de memorie ((DS :SI)) urmată de actualizarea adreselor.

((ES:DI))← ((DS:SI))
şi (SI) = (SI) ± (d), unde d = 1/ 2 (B / W)
(DI) = (DI) ± (d)

Instrucţiunile de comparare şir (Compare String) **CMPSB, CMPSW**— testează egalitatea şirurilor. Se execută o scădere fictivă între octeţii (cuvintele) de la adresele (DS:SI) şi (ES:DI), fără modificarea operanzilor dar cu poziţionarea tuturor flagurilor.

((DS:SI)) – ((ES:DI)) → FLAGS
şi (SI) = (SI) ± (d)
(DI) = (DI) ± (d)

Instrucţiunile de încărcare a elementelor din şir (**Lo**a**d S**tring) **LODSB, LODSW** — se încarcă în AL, respectiv AX octetul, respectiv cuvântul de la adresa (DS:SI), apoi se actualizează adresa.

(AL/AX) ←((DS:SI))
şi (SI) = (SI) ± (d)

Instrucţiunile de memorare şir (**Sto**re **S**tring) **STOSB, STOSW** —se încarcă AL, respectiv AX în octetul, respectiv cuvântul de la adresa (ES:DI), apoi se actualizează adresa.

(AL/AX) →((ES:DI))
şi (DI) = (DI) ± (d)

Instrucţiunile de scanare (**Sca**n **S**tring) **SCASB, SCASW** — testează/caută un anumit octet, cuvânt într-un şir. Se execută diferenţa fictivă dintre AL, respectiv AX şi octetul, respectiv cuvântul de la adresa (ES:DI), fără modificarea operanzilor dar cu poziţionarea tuturor flagurilor.

(AL/AX) – ((ES:DI)) → FLAGS
şi (DI) = (DI) ± (d)

OBS: Începând cu 386 instrucţiunile anterioare se pot realiza şi pe dublu-cuvânt (sufixul este D iar d=4)

Instrucţiunile pentru şiruri realizează inclusiv actualizarea adreselor, însă nu repetă operaţia de un număr de ori egal cu numărul de elemente ale şirului de prelucrat. De aceea, instrucţiunile pentru lucrul cu şiruri sunt combinate cu algoritmi de repetare (gen prefixe de repetare sau bucle cu salt condiţionat).

Prefixe de repetare

Permit execuţia repetată a unor operaţii primitive cu şiruri în funcţie de un contor sau un contor şi o condiţie logică. Formează instrucţiuni compuse alături de operaţiile primitive anterior descrise. Nu sunt instrucţiuni în sine.

REP/ REPE/ REPZ—operaţia primitivă se execută de un număr maxim de ori dat de conţinutul registrului contor CX.

REP/ REPE/ REPZ operaţie_primitivă;

REP este utilizat cu instrucţiuni de tip MOVS, STOS, LODS.
REPE şi REPZ este utilizat cu instrucţiuni de tip CMPS, SCAS.

REPNE/ REPNZ—se iese din buclă dacă rezultatul primitivei este zero. Deci bucla este executată atât timp cât rezultatul este nenul dar nu mai mult de valoarea conţinută de registrul CX.

REP/ REPE/ REPZ operaţie_primitivă;

8.2 Instrucţiuni de salt

Un program se execută prin extragerea din memorie a câte unui octet (fetch). Următoarea instrucţiune de executat se află în segmentul de cod CS, cu un offset dat de către registrul IP (Instruction Pointer). Derularea secvenţială de la o instrucţiune la alta se obţine prin incrementarea registrului IP numit şi Program Counter.
Instrucţiunile de salt ne permit să modificăm derularea secvenţială a unui program. Acestea pot fi clasificate după mai multe criterii.

Salturi – *scurte (SHORT)* sau relative
- *intrasegment (NEAR)* →saltul se face în interiorul segmentului de cod, se modifică IP;
- *intersegment (FAR)* →saltul se poate face oriunde în memorie; se modifică IP şi CS;

Salturi – *condiţionate* →funcţie de valoarea unui anumit bit din PSW;
– *necondiţionate* →derularea e alterată întotdeauna.

8.2.1 Instrucţiunea de salt necondiţionat

JMP—determină întotdeauna un salt în spaţiul de 1 Moctet. Există trei moduri de adresare:
1. Modul relativ: saltul se face la o etichetă a cărei adresă este în domeniul [-128,127] faţă de adresa instrucţiunii JMP;

2. Modul direct: este specificată efectiv adresa unde se sare; mod folosit pentru salturile intra şi inter segment;
NEAR — adresa ţintă este pe 2 octeţi IP_L şi IP_H.
FAR — adresa ţintă este pe 4 octeţi IP_L, IP_H, CS_L şi CS_H.

3. Modul indirect: adresa de salt este rezultatul unor calcule şi se poate modifica dinamic; mod folosit pentru salturile intra şi inter segment
JMP target;

unde target poate fi un registru care conţine offsetul IP-ului, o variabilă care ne dă offsetul sau o adresă de memorie.
Mecanismul de apelare al subrutinelor precum şi instrucţiunile corespunzătoare **CALL** şi **RET** vor fi tratate în lucrarea 9.

8.2.2 Instrucţiuni de salt condiţionat

Condiţiile de salt sunt determinate de starea anumitor flaguri din PSW, afectate de operaţiile aritmetice, logice sau de control.
Modul de adresare este întotdeauna de tip SHORT.

Pentru testarea aceleiaşi condiţii se pot utiliza mnemonici diferite.
Bistabilii de condiţie nu se modifică.

Mnemonica	Condiţie de salt	Interpretare
JE JZ	Z = 1	Equal , Zero
JL JNGE	S ≠ O	Less, Not Greater or Equal
JLE JNG	S ≠ O sau Z = 1	Less or Equal, Not Greater
JB JNAE JC	C = 1	Below, Not Above or Equal, Carry
JBE JNA	C = 1 sau Z = 1	Below or Equal, Not Above
JP JPE	P = 1	Parity, Parity Even
JO	O = 1	Overflow
JS	S = 1	Sign
JNE JNZ	Z = 0	Not Zero, Not Equal
JNL JGE	S = O	Not Less, Greater or Equal
JNLE JG	S = O şi Z = 0	Not Less or Equal, Greater
JNB JAE JNC	C = 0	Not Below, Above or Equal, Not Carry
JNBE JA	C = 0 şi Z = 0	Not Below or Equal, Above
JNP JPO	P = 0	Not Parity, Parity Odd
JNO	O = 0	Not Overflow
JNS	S = 0	Not Sign
JCXZ	Reg CX = 0	CX register is zero

Instrucţiunea JCXZ nu testează indicatori de condiţie ci conţinutul registrului CX.

8.2.3 Instrucţiuni de buclare

În programe apare necesitatea execuţiei unei secvenţe de instrucţiuni, în mod repetat. Secvenţa care se repetă se numeşte buclă (loop) sau iteraţie. Instrucţiunile de control al buclelor sunt prezentate în următorul tabel:
Toate aceste instrucţiuni utilizează registrul CX ca număr de iteraţii, adică se decrementează CX şi dacă acesta e diferit de zero se sare la eticheta specificată. Instrucţiunile de LOOP condiţionat înainte de a testa dacă

registrul contor CX a ajuns la zero, verifică şi indicatorul Z. Eticheta unde se face saltul se află în domeniul [-128, 127] faţă de instrucţiunea curentă.

LOOPxx etichetă;

Mnemonică	Interpretare
LOOP	CX = CX – 1 If (CX ≠ 0) then jump Else continue
LOOPE LOOPZ	CX = CX – 1 If (CX ≠ 0 and Z = 1) then jump Else continue
LOOPNE LOOPNZ	CX = CX – 1 If (CX ≠ 0 and Z = 0) then jump Else continue

8.3 Instrucţiuni pentru controlul procesorului

Aceste instrucţiuni nu au operanzi. O categorie de instrucţiuni se referă la controlul explicit al unor bistabili de condiţie din PSW:

Mnemonică	Acţiune
CLC (Clear carry flag)	C = 0
CLD (Clear direction flag)	D = 0
CLI (Clear interrupt flag)	I = 0
CMC (Complement carry flag)	C = not (C)
STC (Set carry flag)	C = 1
STD (Set direction flag)	D = 1
STI (Set interrupt flag)	I = 1

Dacă dorim ca o anumită secvenţă de program să nu fie întreruptă, aceasta se protejează printr-o instrucţiune CLI înainte şi una STI după.

Cea de-a doua categorie de instrucţiuni realizează operaţii speciale asupra procesorului:

HALT — Oprire procesor; se poate ieşi doar prin întreruperi externe sau reset general.

LOCK — Blocare magistrală; se foloseşte înaintea oricărei instrucţiuni iar pe durata acesteia accesul unui alt dispozitiv la magistrala sistemului hardware este blocat.
WAIT — Aşteaptă; folosită la sincronizarea procesorului cu un alt dispozitiv, de obicei coprocesor matematic.

Instrucţiunile specifice întreruperilor **INT** şi **IRET** vor fi tratate în lucrarea 9.

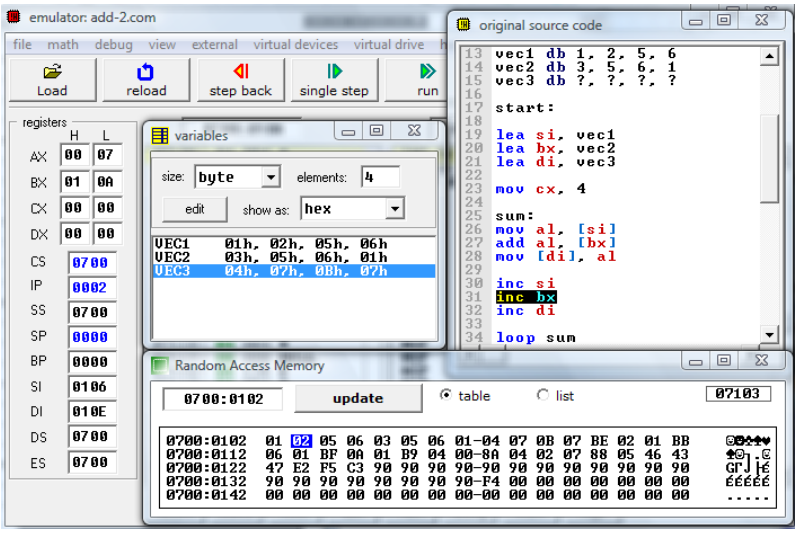
8.4 Exerciţii şi teme

1. Instrucţiuni de salt condiţionate: Calculaţi expresia *r=x-y+z*, numerele fiind reprezentate pe 16 biţi fără semn, testând eventualele depăşiri. In registrul BX puneţi 0 dacă nu sunt depăşiri şi 1 dacă sunt.

```
        mov ax, x
        sub ax, y
        jc et
        add ax, z
        jc et
        mov bx, 0
        jmp gata
et:     mov bx, 1
gata:   ....
```

2. Urmatorul exemplu calculeaza suma a doi vectori iar rezultatul il depune in alt vector

```
        org 100h
        jmp start
        vec1 db 1, 2, 5, 6
        vec2 db 3, 5, 6, 1
        vec3 db ?, ?, ?, ?
        start:
        lea si, vec1
        lea bx, vec2
        lea di, vec3
        mov cx, 4
        sum:
            mov al, [si]
            add al, [bx]
            mov [di], al
            inc si
            inc bx
            inc di
            loop sum
        ret
```



3. Sa se copieze un şir sursa (SIRs) format din 3 elemente definite pe octet într-un alt şir destinatie (SIRd).

```
        ;regiştrii index SI respectiv DI trebuie să pointeze
        ;spre primul element din fiecare şir

        ;pentru fiecare element se executa instrucţiunea MOVSB


      SIRs DB 1,2,3     ; şirul destinaţie cu 3 elemente pe octet
      SIRd DB 3 DUP(0) ;şirul sursă cu 3 elemente, neiniţilizat
        ...
      lea SI, SIRs      ; SI=adr de început a SIRs
      lea DI, SIRd      ; DI=adr de început a SIRd
      mov CX,3          ; CX=3 numărul de elemente
      cld               ; DF=0, şiruri parcurse în sens crescător
et:   movsb             ;eticheta eti se foloseşte pentru a
                        ;asigura revenirea în acest punct
      dec cx            ; după execuţia mosvb (mutare element şi
                        ; actualizare adrese), CX=CX-1
        jnz et          ; dacă încă CX nu a ajuns în zero(jump if
                        ;not ZF->dacă ZF=0,face salt), reia bucla
```

Obs1: Secvenţa de buclare: se poate înlocui cu:

et: movsb dec cx jnz et	et: movsb loop et	rep movsb

Obs2: Programul anterior poate fi rescris folosind instr. LODSB şi STOSB (elementele şirului sursă trebuie preluate în acumulator şi apoi depuse în şirul destinaţie).

```
et: lodsb   ;AL = element curent din SIRs
    stosb   ;din AL se depune elementul curent în SIRd
    dec cx  ;CX=CX-1
    jnz et  ;dacă CX diferit de zero, se reia bucla.
```

4. Sa se testeze egalitatea a 2 şiruri de dimensiuni egale.

```
; se compară sirurile element cu element, iar dacă înainte de a
;ajunge la sfârşit se întâlneşte o nepotrivire, concluzionăm că
;şirurile nu sunt egale(DI contine poziţia primei nepotriviri.

        SIRs DB 0,1,2,3,2,4,2,5,2,6 ;
        SIRd DB 0,1,2,5,2,4,2,5,2,6 ;
        ...
        lea SI, SIRs            ; SI=adr de început a SIRs
        lea DI, SIRd            ; DI=adr de început a SIRd
        mov CX, 10              ; CX= numarul de elemente
        cld                     ; DF=0
        xor bx,bx               ; BX=0
        et1: inc BX
        cmpsb                   ;  verifica  egalitatea  elementelor
        loope et1
```

5. Rezolvaţi problemele de mai jos şi verificaţi-le cu debugger-ul.

a. Scrieţi o secvenţă în limbaj de asamblare care conţine o funcţie care adună conţinutul regiştrilor AX şi BX şi pune rezultatul în registrul CX.

b. Scrieţi o secvenţă care încarcă valorile 00h, 01h, ... FEh, FFh în memorie începând de la adresa DS:0018h. O idee de bază este dată mai jos:

```
        mov al,00h
        mov bx,18h
et:     .....
        jle et
```

c. Scrieţi o secvenţă care încarcă valorile FFh, FEh, FDh, ... 01h, 00h în memorie începând de la adresa DS:0400h.

d. Scrieţi o secvenţă care mută un bloc din memorie de la DS:0220h până la 0300h la adresele începând de la 0400h.

e. Scrieţi o secvenţă care determină cel mai mare octet din memorie între locaţiile DS:0000h şi DS:0050.