

LUCRAREA Nr. 5

INSTRUCȚIUNI DE CONTROL AL PROGRAMULUI (CU EXCEPȚIA SALTURILOR PROPRIU-ZISE) PENTRU MICROPROCESOARELE INTEL

1. Scopul lucrării

Lucrarea de față își propune familiarizarea cu instrucțiunile de control al programului, exceptând salturile propriu-zise, precum și cu câteva tehnici de programare în asamblor specifice microprocesoarelor Intel funcționând “în modul real” (cu particularizări pentru 8086).

2. Memoriu de instrucțiuni

Convențiile folosite sunt cele arătate în lucrarea de laborator nr. 3 pentru operanzi și pentru starea fanioanelor.

2.1. Instrucțiuni de apelare și revenire de/din proceduri

CALL adr	Apelarea unui subprogram	OF DF IF TF SF ZF AF PF CF
-----------------	--------------------------	----------------------------

Descrierea formală a semanticii, în funcție de modul de adresare folosit:

a) Apel de subprogram cu adresare absolută (directă) **intersegment**:

CALL adr32 ;

$$(SP) \leftarrow (SP) - 2$$

$$((SS)\uparrow 0H + (SP) + 1) \uparrow ((SS)\uparrow 0H + (SP)) \leftarrow (CS)$$

$$(CS) \leftarrow ((CS)\uparrow 0H + (IP) + 4) \uparrow ((CS)\uparrow 0H + (IP) + 3)$$

$$(SP) \leftarrow (SP) - 2$$

$$((SS)\uparrow 0H + (SP) + 1) \uparrow ((SS)\uparrow 0H + (SP)) \leftarrow (IP)$$

$$(IP) \leftarrow ((CS)\uparrow 0H + (IP) + 2) \uparrow ((CS)\uparrow 0H + (IP) + 1).$$

În descrierea formală a semanticii am ținut seama că adresa completă **adr32**, care este o *adresă logică*, face parte din formatul instrucțiunii.

b) Apel de subprogram cu adresare relativă:

CALL disp16 ;

$$(SP) \leftarrow (SP) - 2$$

$$((SS)\uparrow 0H + (SP) + 1) \uparrow ((SS)\uparrow 0H + (SP)) \leftarrow (IP)$$

$$(IP) \leftarrow (IP) + \text{disp16} .$$

Este un salt intrasegment, dar se observă că, spre deosebire de salturile propriu-zise, deplasamentul este admis numai pe 16 biți (deci el va ocupa doi octeți în formatul instrucțiunii curente de apel).

c) Apel de subprogram cu adresare indirectă în memorie, **intersegment**:

CALL mem32 ;

$$(SP) \leftarrow (SP) - 2$$

$$((SS)\uparrow 0H + (SP) + 1) \uparrow ((SS)\uparrow 0H + (SP)) \leftarrow (CS)$$

$$(CS) \leftarrow (\text{mem32})_h$$

$$(SP) \leftarrow (SP) - 2$$

$$((SS)\uparrow 0H + (SP) + 1) \uparrow ((SS)\uparrow 0H + (SP)) \leftarrow (IP)$$

$$(IP) \leftarrow (\text{mem32})_l .$$

d) Apel de subprogram cu adresare în registru sau indirectă în memorie, **intra-segment**:

CALL r16|mem16 ;

$$(SP) \leftarrow (SP) - 2$$

$$((SS)\uparrow 0H + (SP) + 1) \uparrow ((SS)\uparrow 0H + (SP)) \leftarrow (IP)$$

$$(IP) \leftarrow (r16) | (\text{mem16}) .$$

Operanzi	Exemple	Descrierea formală a semanticii
adr32	CALL PROCSEGM	$(SP) \leftarrow (SP) - 2$ $((SS)\uparrow 0H + (SP) + 1) \uparrow ((SS)\uparrow 0H + (SP)) \leftarrow (CS)$ $(CS) \leftarrow ((CS)\uparrow 0H + (IP) + 4) \uparrow ((CS)\uparrow 0H + (IP) + 3)$ $(SP) \leftarrow (SP) - 2$ $((SS)\uparrow 0H + (SP) + 1) \uparrow ((SS)\uparrow 0H + (SP)) \leftarrow (IP)$ $(IP) \leftarrow ((CS)\uparrow 0H + (IP) + 2) \uparrow ((CS)\uparrow 0H + (IP) + 1)$
disp16	CALL SORTARE	$(SP) \leftarrow (SP) - 2$ $((SS)\uparrow 0H + (SP) + 1) \uparrow ((SS)\uparrow 0H + (SP)) \leftarrow (IP)$ $(IP) \leftarrow (IP) +$ $\quad + ((CS)\uparrow 0H + (IP) + 2) \uparrow ((CS)\uparrow 0H + (IP) + 1)$
r16	CALL BX	$(SP) \leftarrow (SP) - 2$ $((SS)\uparrow 0H + (SP) + 1) \uparrow ((SS)\uparrow 0H + (SP)) \leftarrow (IP)$ $(IP) \leftarrow (BX)$
mem*	CALL [BX]	$(SP) \leftarrow (SP) - 2$ $((SS)\uparrow 0H + (SP) + 1) \uparrow ((SS)\uparrow 0H + (SP)) \leftarrow (IP)$ $(IP) \leftarrow ((DS)\uparrow 0H + (BX) + 1) \uparrow ((DS)\uparrow 0H + (BX))$
mem**	CALL [DI]	$(SP) \leftarrow (SP) - 2$ $((SS)\uparrow 0H + (SP) + 1) \uparrow ((SS)\uparrow 0H + (SP)) \leftarrow (CS)$ $(CS) \leftarrow ((DS)\uparrow 0H + (DI) + 3) \uparrow ((DS)\uparrow 0H + (DI) + 2)$ $(SP) \leftarrow (SP) - 2$ $((SS)\uparrow 0H + (SP) + 1) \uparrow ((SS)\uparrow 0H + (SP)) \leftarrow (IP)$ $(IP) \leftarrow ((DS)\uparrow 0H + (DI) + 1) \uparrow ((DS)\uparrow 0H + (DI))$

- * apelare de subprogram cu adresare indirectă definită cu directivă de asamblare ca apelare intra-segment;
- ** apelare de subprogram cu adresare indirectă definită cu directivă de asamblare ca apelare inter-segment.

RET [data16]	Reîntoarcere din subprogram	OF DF IF TF SF ZF AF PF CF

Descrierea formală a semanticii, în funcție de modul de adresare folosit:

```

RET [data16] ;
  (IP) ← ((SS)↑0H +(SP)+1) ↑ ((SS)↑0H +(SP))
  (SP) ← (SP) + 2
  [(CS) ← ((SS)↑0H +(SP)+1) ↑ ((SS)↑0H +(SP))
  (SP) ← (SP) + 2]
  [(SP) ← (SP) + data16] .

```

Se observă că revenirea se poate face fie **intersegment** fie **intra-segment**; această decizie, transparentă programatorului, este determinată de tipul apelului de subprogram folosit în prealabil. În plus, există posibilitatea de a se rezerva, opțional, un spațiu în stivă prin folosirea operandului **data16**.

Operanzi	Exemple	Descrierea formală a semanticii
	RET ;intra-segment	(IP) ← ((SS)↑0H +(SP)+1) ↑ ((SS)↑0H +(SP)) (SP) ← (SP) + 2
data16	RET 4 ;intra-segment	(IP) ← ((SS)↑0H +(SP)+1) ↑ ((SS)↑0H +(SP)) (SP) ← (SP) + 2 (SP) ← (SP) + 04H
	RET ;inter-segment	(IP) ← ((SS)↑0H +(SP)+1) ↑ ((SS)↑0H +(SP)) (SP) ← (SP) + 2 (CS) ← ((SS)↑0H +(SP)+1) ↑ ((SS)↑0H +(SP)) (SP) ← (SP) + 2
data16	RET 2 ;inter-segment	(IP) ← ((SS)↑0H +(SP)+1) ↑ ((SS)↑0H +(SP)) (SP) ← (SP) + 2 (CS) ← ((SS)↑0H +(SP)+1) ↑ ((SS)↑0H +(SP)) (SP) ← (SP) + 2 (SP) ← (SP) + 02H

2.2. Controlul iterațiilor

LOOP disp8	Ciclează necondiționat	OF DF IF TF SF ZF AF PF CF

Descrierea formală a semanticii:

$$(\mathbf{CX}) \leftarrow (\mathbf{CX}) - 1$$

if $(\mathbf{CX}) \neq 0$ then $(\mathbf{IP}) \leftarrow (\mathbf{IP}) + \mathbf{disp8}$.

Operanzi	Exemple
disp8	LOOP BUCLA

LOOPE LOOPZ disp8	Ciclează cât timp "egal" sau "zero"	OF DF IF TF SF ZF AF PF CF

Descrierea formală a semanticii:

$$(\mathbf{CX}) \leftarrow (\mathbf{CX}) - 1$$

if $(\mathbf{ZF})=1$ and $(\mathbf{CX}) \neq 0$ then $(\mathbf{IP}) \leftarrow (\mathbf{IP}) + \mathbf{disp8}$.

Această instrucțiune permite un ciclu condiționat; ciclarea se realizează până la epuizarea contorului numai dacă în tot acest timp $(\mathbf{ZF})=1$ (ciclează "cât timp este zero" sau "cât timp este egal", ceea ce înseamnă că se poate ține seama de egalitatea a doi operanzi sau de orice altă operație care are drept urmare setarea fanionului \mathbf{ZF}). Orice resetare a fanionului \mathbf{ZF} duce la ieșirea forțată din ciclu.

Operanzi	Exemple
disp8	LOOPE CICLU

LOOPNE LOOPNZ disp8	Ciclează cât timp "ne-egal" sau "non-zero"	OF DF IF TF SF ZF AF PF CF

Descrierea formală a semanticii:

$$(\mathbf{CX}) \leftarrow (\mathbf{CX}) - 1$$

if $(\mathbf{ZF})=0$ and $(\mathbf{CX}) \neq 0$ then $(\mathbf{IP}) \leftarrow (\mathbf{IP}) + \mathbf{disp8}$.

Condiția de menținere a ciclului este inversă față de instrucțiunea precedentă: ciclează "cât timp diferit de zero" sau "cât timp nu e egal".

Operanzi	Exemple
disp8	LOOPNZ REPETA

JCXZ disp8	Salt dac' (CX) = 0	OF DF IF TF SF ZF AF PF CF
------------	--------------------	----------------------------

Descrierea formală a semanticii:

if **(CX) = 0** then **(IP) ← (IP) + disp8** .

Operanzi	Exemple
disp8	JCXZ STOP

3. Câteva elemente privind tehnica de utilizare a atributelor de arhitectură ale microprocesorului

Când se proiectează un program în limbaj de asamblare, pentru a controla exact evoluția programului sau pentru a-l optimiza, trebuie avute în vedere atributele de arhitectură ale microprocesorului, câteva elemente care țin de modul său de lucru, caracteristicile asamblorului sau sistemului de operare.

Astfel:

- Pe prima linie a programului se introduce directiva **ORG 100h** necesară deoarece sistemul de operare DOS încarcă și execută programele executabile de tip **COM** de la adresa 100h.

- Se recomandă ca la începutul programului să se facă suprapunerea segmentului de date (definit cu **DS**) cu segmentul de cod (definit cu **CS**), de exemplu prin secvența de instrucțiuni:

```
mov ax,cs
mov ds,ax .
```

- Ultima instrucțiune pe care o execută programul trebuie să fie **INT 20h**, dacă se dorește revenirea în sistemul de operare DOS (sau în programul apelant).

- Se recomandă ca datele declarate prin directive **DB** (define byte), **DW** (define word), **EQU** (equality), să fie plasate grupat la începutul sau sfârșitul programului; dacă declararea se face la începutul programului, se va plasa înaintea zonei de date un salt necondiționat la prima instrucțiune a programului, iar dacă declararea se face la sfârșitul programului, zona de date se va plasa după instrucțiunea de reîntoarcere în sistemul de operare, **INT 20h**.

- Când se folosesc subprograme (proceduri), trebuie avută în vedere problema conservării unor registre prin plasarea de instrucțiuni **PUSH** la începutul subprogramului pentru salvarea în stivă a registrelor dorite, iar la sfârșitul subprogramului, instrucțiuni **POP** pentru refacerea acestora.

- În corpul unui subprogram se pot face apelări de alte subprograme, însă trebuie observat că pentru fiecare apel se face automat o salvare în stivă (poate apărea depășirea spațiului alocat pentru stivă).

Transferul de date către subprograme se poate face:

- prin registre, dacă numărul datelor este mic;
- în memorie, prin intermediul stivei; în programul apelant se plasează instrucțiuni **PUSH** iar în subrutină instrucțiuni **POP**;
- în memorie, sub forma unor tabele de date, prin transferarea adreselor acestora folosind registre.

Transferul de date de la subprograme se poate face:

- prin registre;
- prin stivă;
- prin tabele de date;
- prin intermediul fanioanelor.

4. Modul de lucru recomandat

În lucrare sunt prezentate trei programe de aplicații care rezolvă probleme reale, pentru care s-a specificat atât problema de rezolvat cât și algoritmul folosit.

Programele se vor edita, omițându-se comentariile, și se vor rula pas cu pas, urmărindu-se modificările în registre și memorie, conform comentariilor.

5. Desfășurarea lucrării

5.1. Se lansează asamblorul integrat TASMB și se editează textul programului nr.1, fără a scrie și comentariile.

5.2. Se salvează programul sursă pe disc (**W**) cu un nume oarecare, apoi se selectează (**O**) opțiunea **F8-COM FILE** și se assemblează (**A**).

5.3. Se vizualizează lista de simboluri (**S**) și se notează adresele simbolurilor folosite.

5.4. Se părăsește (**Q**) asamblorul integrat și se lansează depanatorul AFD. Se încarcă programul salvat anterior cu extensia **COM (L)**.

5.5. Se setează zona 2 de afișare a memoriei la adresa simbolului **SIR** .

5.6. Se execută programul pas cu pas (**F2**), urmărindu-se în paralel registrele și memoria, conform comentariilor.

5.7. Se repetă punctele 5.1. - 5.6. pentru programele 2 și 3.

Programul 1

Problema:

Să se scrie o procedură de găsim a maximului într-un șir de numere întregi cu semn, reprezentate pe câte un octet.

Se impun:

Date de intrare:

- registrul **BX** va conține adresa de început a șirului;
- registrul **CX** va conține lungimea șirului.

Date de ieșire:

- registrul **AL** va conține octetul maxim din șir;
- registrul **DX** va conține adresa octetului maxim.

Registre afectate: **AL** și **DX**.

Pentru a verifica procedura, se va scrie un program în care se vor defini datele, se vor inițializa registrele și se va apela procedura de găsim a maximului.

Algoritm:

Se inițializează registrul **AL** cu cel mai mic întreg cu semn reprezentabil pe un octet (**80h**). Începând cu primul octet din șir, se face o comparație cu conținutul registrului **AL**. Dacă octetul din șir este mai mare decât conținutul registrului **AL**, se face transferul octetului în **AL** și a adresei octetului în **DX**. Se continuă parcurgerea șirului cu elementul următor, până la epuizarea tuturor octeților din șir.

Textul programului:

```
org 100h
mov ax,cs
mov ds,ax
jmp start ;salt la prima instructiune

sir db 0f0h,0f3h,05h,76h,89h,76h,85h,93h

start mov bx,offset sir ;incarca in BX adresa sirului.
      mov cx,offset start-offset sir
      ;incarca in CX lungimea
      ;sirului

      call getmax ;apelare rutina de maxim.
      int 20h ;revenire in DOS.

getmax push bx ;conserva registrele
       push cx ;BX, CX si fanioanele.
       pushf
       mov al,80h ;incarca in AL cel mai mic numar
              ;cu semn de un octet

adr1 cmp al,[bx] ;compara AL cu un octet din sir
     jl adr3 ;daca AL<octet salt la 'adr3'
adr2 inc bx ;trece la octetul urmator
```

```

                                ;al sirului.
                                ;repetă de la adresa 'adr1'
                                ;pana la sfirsitul sirului.
                                ;reface registrele BX, CX
                                ;si fanioanele.
                                ;revenire in programul
                                ;apelant.
                                ;inacarca in AL octetul cu
                                ;adresa in BX.
                                ;incarca in DX adresa octetului.
                                ;salt la 'adr2'.
loop   adr1
popf
pop    cx
pop    bx
ret
adr3   mov    al, [bx]
      mov    dx, bx
      jmp   adr2

```

Programul 2

Problema:

Să se ordoneze în ordine descrescătoare sau crescătoare un șir de cuvinte (de câte 2 octeți).

Algoritm:

Se parcurge șirul în ordine, comparându-se elementul **i** cu elementul **i+1**. Dacă cele două elemente nu sunt în ordinea dorită, se vor inversa. Dacă a avut loc o inversare de elemente, se reia parcurgerea șirului cu primul element, altfel se continuă parcurgerea șirului cu elementul următor.

Descriere program:

Se folosesc două proceduri **ORDON** și **GETORD**.

Procedura **ORDON** ordonează șirul în ordinea dorită.

Date de intrare:

- registrul **BX = 0** pentru ordonare descrescătoare;
- BX = FF** pentru ordonare crescătoare;
- registrul **BP = adresa de început a șirului**.

Date de ieșire:

- zona de memorie de la adresa șirului.

Procedura **GETORD** găsește relația dintre două elemente succesive ale șirului.

Date de intrare:

- registrul **BX** indică tipul ordonării;
- registrul **BP** conține adresa de început a șirului;
- registrul **SI** conține indexul elementului curent.

Date de ieșire:

- fanionul **CF = 0** ordinea este bună;
- CF = 1** ordinea este inversă.

Textul programului

```

                org     100h
                mov     ax,cs
                mov     ds,ax
                jmp     start           ;salt la prima instructiune

sir             dw     2032h,2037h,2031h,2033h,2035h,2036h,2034h
lungsir        dw     ?               ;lungimea sirului

start          mov     bp,offset sir
                mov     si,0

                mov     [lungsir],(offset lungsir)-(offset sir)-2
                ror     lungsir,1

desc           mov     bl,00           ;ordoneaza sirul in
                call    ordon          ;ordine descrescatoare

cresc mov      bl,0ffh                ;ordoneaza sirul in
                call    ordon          ;ordine crescatoare
                int     20h            ;revenire in programul apelant

ordon          mov     si,0            ;seteaza SI la inceputul sirului
                mov     cx,lungsir     ;incarca in CX nr. de iteratii

bucla call     getord                 ;cheama procedura GETORD
                ;(seteaza sau nu CF)
                jc     invers          ;daca ordinea nu este buna, salt
                inc     si             ;seteaza SI pentru elementul
                inc     si             ;urmator

                loop   bucla           ;repete de la BUCLA
                ret

invers         mov     ax,[bp+si]      ;interschimb intre locatiile
                xchg   ax,[bp+si+2]    ;de memorie BP+SI si BP+SI+2
                xchg   ax,[bp+si]      ;folosind registrul AX ca
                ;registru temporar.
                mov     si,00          ;seteaza SI la inceputul
                ;sirului
                mov     cx,lungsir     ;incarca CX cu nr. de iteratii
                jmp     bucla

getord         mov     ax,[bp+si]      ;compara elementele succesive
                ;de index SI si SI+2
                cmp    ax,[bp+si+2]
                lahf                    ;incarca fanioanele in
                ;registrul AH si seteaza
                xor    ah,bl           ;conform lui BL
                rol    ah,1            ;pregateste continutul lui
                ;AH pentru tranfer
                sahf                    ;in fanioane
                ret

```

Programul 3

Problema:

Dându-se 10 șiruri de caractere, fiecare de maximum 255 octeți, având ultimul octet 0, să se ordoneze în ordine alfabetică, în aceeași zonă de memorie.

Algoritm:

Se consideră șirurile pe rând, începând cu primul, și se compară cu șirul următor. Dacă ordinea în care se află nu este bună, șirul curent este mutat la sfârșitul zonei de date și se reia procesul de la primul șir. Altfel, se trece la următorul șir și se face o nouă comparație.

Descriere program:

Se folosesc trei proceduri: **CALCAD**, **COMP** și **SCHIMB**.

Procedura **CALCAD** calculează adresa șirului cu numărul dat de conținutul registrului **AL**.

Date de intrare:

- registrul **AL** conține numărul șirului pentru care se calculează adresa;
- zona de memorie de la adresa **SIR**.

Date de ieșire:

- registrul **BX** adresa irului specificat;

Registre afectate: **BX**.

Procedura **COMP** compar două șiruri.

Date de intrare:

- registrele **BX** și **DX** conțin adresele celor două șiruri;
- zona de memorie de la adresele date de registrele **BX** și **DX**.

Date de ieșire:

- fanionul **SF = 1** dacă ordinea nu corespunde.

Registre afectate: nici unul.

Procedura **SCHIMB** plasează șirul care începe la adresa dată de registrul **BX** la sfârșitul zonei de date.

Date de intrare:

- registrul **BX** conține adresa șirului de transferat;
- zona de memorie de la adresa dată de registrul **BX** până la sfârșitul zonei de date.

date.

Date de ieșire:

- zona de memorie de la adresa specificată până la sfârșitul zonei de date.

Registre afectate: **SI** și **DI**.

Textul programului

```
org 100h
mov ax,cs
mov ds,ax
jmp start ;salt la prima instructiune

nrsir dw 9 ;cuvant care contine numarul de
;siruri, incepand cu 0
sir db 'program',0 ;definire siruri alfanumerice
;terminate cu octet zero
db 'de',0
db 'ordonare',0
db 'pentru',0
db 'siruri',0
db 'alfanumerice',0
db 'terminate',0
db 'cu',0
db 'octet',0
db 'zero',0

start mov cx,nrsir ;aici incepe programul cu
;incarcarea in CX a numarului de
; siruri (numarul de iteratii).

bucla mov al,byte ptr nrsir
sub al,cl ;incarca in AL numarul sirului
; curent AL=NRSIR- CL.
call calcad ;calculeaza adresa de inceput
;a sirului curent (adresa in BX)
;si stocheaza adresa in DX
mov dx,bx
inc al
call calcad ;calculeaza adresa de inceput a
;sirului curent+1 (adresa in BX)
call comp ;compara sirurile de la adresele
;continute in BX si DX
jg adr1 ;daca ordinea este buna
;face salt la 'adr1'
call schimb ;altfel apeleaza SCHIMB
mov cx,nrsir ;si pregateste CX pentru
;reluare iteratii

adr1 inc cx ;de la inceput.
loop bucla ;repetă programul pînă la
;ultimul sir
int 20h ;revenire in DOS

calcad push cx ;conserva registrele CX
push ax ;si AX.

mov bx,offset sir ;incarca in BX adresa
;primului sir
and al,al ;daca AL este 0
```

```

                jz     adr4             ;salt la 'adr4' (in BX este
                ;adresa sirului 0).
                mov    cl,al           ;pune in CX numarul sirului
                mov    ch,0            ;din AL
                xor    al,al           ;face AL=0 (referinta de
                ;comparatie).
adr2            cmp    [bx],al        ;compara cu 0
                ;continutul locatiei de
                ;memorie cu adresa data de BX.
                je     adr3           ;daca egalitate salt la
                ; 'adr3'
                inc    bx             ;altfel trece la urmatorul
                jmp    adr2           ;element al sirului
adr3            inc    bx             ;in BX adresa de inceput a
                ;unui sir
                loop   adr2           ;daca nu este cel dorit
                ;repete de la 'adr2'
adr4            pop    ax             ;reface registrele AX si CX
                pop    cx
                ret                    ;revine in programul apelant

comp           push   ax             ;conserva registrele AX si BX
                push   bx
                mov    si,dx          ;pune in SI adresa de inceput
                ;a unui sir de comparat
reia           lodsb                ;pune in AL un octet al sirului
                cmp    [bx],al        ;compara un octet al celui
                ;de-al doilea sir de comparat
                ;cu AL setand fanioanele
                je     egal           ;daca egalitate salt la
                ; 'egal'
                pop    bx            ;reface registrele BX si AX
                pop    ax
                ret                    ;revine in programul apelant

egal           inc    bx             ;trece la elementul urmator
                jmp    reia           ;al sirurilor.
schimb         push   ax             ;conserva registrele AX si CX
                push   cx
reit           mov    si,dx          ;incarca in SI si DI adresa
                mov    di,dx          ;sirului de transferat
                lodsb                ;incarca in AL octetul
                ;de la adresa SI
                mov    cx,offset start-1
                sub    cx,dx          ;incarca pe CX cu diferenta
                ;dintre inceputul sirului
                ;si sfarsitul zonei de date
repsz          movsb                ;transfera toti octetii
                ;din aceasta zona
                stosb                ;pune in ultima locatie a zonei
                ;de date octetul din AL
                cmp    al,0          ;compara daca s-a ajuns la
                ;sfarsitul sirului.

```

```
        je    ies        ;daca da, salt la 'ies'  
        jmp   reit      ;altfel salt la 'reit' (reluare).  
  
ies     pop    cx        ;reface registrele CX si AX.  
        pop    ax  
        ret                ;revenire in programul apelant.
```