UNIUNEA EUROPEANĂ
GUVERNUL ROMÂNIEI
Instrumente Structurale 2007-2013

Platformă de e-learning și curriculă e-content pentru învățământul superior tehnic

Programare în limbaj de asamblare

46. Operare pe şiruri, de transfer al controlului programului, de control procesor şi de sistem.

Instrucțiuni de operare pe şiruri

Există cinci operații de bază ce operează pe şiruride lungime de până la un segment (64K la 286, respectiv 4G la 386/486), şi permit: mutare, comparare de şiruri, scanarea unui şir pentru o valoare şi transfer la/de la acumulator. Aceste operații de bază pot fi precedate de un prefix special de un octet care are ca efect repetarea instrucțiunii prin hardware. Repetiția se poate termina printr-o varietate de condiții, iar o operație repetată poate fi întreruptă sau suspendată.

O instrucțiune pe şir poate avea un operand sursă, un operand destinație sau pe amândoi. Şirul sursă este în (**DS**) sau într-un alt segment, dacă este prefixat peste această presupunere. Şirul destinație se află, întotdeauna, în segmentul dat de registrul (**ES**).

SI este ca offset pentru elementului curent al şirului sursă, iar **DI** este offset pentru şirul destinație. Instrucțiunile pe şir actualizează automat SI şi/sau DI. **DF** determină dacă registrele index sunt auto-incrementate-decrementate (DF=0-1), iar pas de actualizare depinde de tipul şirurilor: 1 - octet, 2 - cuvânt, 4 - dublu cuvânt.

Dacă este prezent un prefix de repetare reg. **CX** este decrementat la fiecare repetiție a instrucțiunii pe şir; repetarea va lua sfârşit când CX=0, sau pentru unele prefixe şi în funcție de ZF.

Sunt cinci mnemonici pentru două forme de prefix octet, care controlează repetarea unei instrucțiuni pe şir(uri). Prefixele de repetare nu afectează indicatorii.

REP / REPE / REPZ / REPNE / REPNZ

REP - REPeat - este utilizat împreună cu MOVS- STOS, "repetă cât timp nu este sfârşitul şirului, adică (CX<>0)";

REPE / REPZ - REPeat while Equal / Zero - care operează la fel şi au fizic acelaşi prefix octet ca REP. Sunt utilizate pentru CMPS şi SCAS, şi necesită, în plus față de condiția anterioară, ca ZF să fie setat înainte de efectuarea următoarei repetiții;

REPNE / REPNZ - REPeat while Not Equal / Zero - este asemănător cu cele anterioare, cu deosebirea că ZF trebuie pus pe 0, sau repetiția se termină.

MOVS <sir_dest>, <sir_sursa> (MOVe data from String to string)
MOVSB / MOVSW / MOVSD
Se transferă un element (octet/cuvânt/dublu cuvânt) de la sursă (SI) la destinație (DI), şi actualizează SI/DI
Utilizată împreună cu prefixul REP realizează un transfer de bloc memorie-memorie.

CMPS <sir_dest>, <sir_sursa> (COMPare String operands)
CMPSB / CMPSW / CMPSD
Această instrucțiune scade operandul destinație (referit de DI) din cel sursă (referit de SI), modifică indicatorii de stare, dar nu alterează nici unul dintre operanzi, după care actualizează SI/DI.
Trebuie reținut că spre deosebire de instr. CMP care scade sursa din destinație, CMPS realizează scăderea invers: sursă - destinație.

Dacă CMPS are prefixul REPE sau REPZ operația este interpretată "compară cât timp nu este sfârşit şirul (CX<>0) şi şirurile sunt egale (ZF=1)"; determina prima pereche de elemente diferite.
Dacă CMPS este precedată de REPNE/REPNZ "compară cât timp nu este sfârşit de şir (CX<>0), şi elementele şirurilor nu sunt egale (ZF=0)"; determina prima pereche de elemente egale.

SCAS <sir_dest> (SCAn String data)
SCASB / SCASW / SCASD
Instrucțiunea scade elem. şir dest. (octet/ cuvânt/ dublu cuvânt) adresat de DI, din AL, AX sau EAX, şi actualizează indicatorii, dar fără a modifica şirul destinație sau acumulatorul.
Dacă SCAS este prefixată de REPE/REPZ "scanează cât timp nu este sfârşitul şirului (CX<>0) şi valoarea scanată este egală cu elementele şirului (ZF=1)"; determina abaterea față de o valoare.
Dacă SCAS este prefixată de REPNE/REPNZ "scanează cât timp nu este sfârşitul şirului (CX<>0) şi valoarea scanată este diferită de elementele şirului (ZF=0)"; localizează o valoare într-un şir.

LODS <sir_sursa> (LOaD String operand)
LODSB / LODSW / LODSD
Transferă elementul şirului sursă adresat de SI, în AL, AX sau EAX, şi actualizează SI. Instrucțiunea se utilizează în bucle soft.

STOS <sir_dest> (STOre String data)
STOSB / STOSW / STOSD
Transferă un element din acumulator (AL, AX, EAX) în şirul destinație, şi actualizează DI, pentru a referi următorul element.
Instrucțiunea poate fi precedată de prefixul REP, şi astfel se poate inițializa un şir cu o constantă.

1) Copierea unui şir de octeți dintr-o zonă de memorie într-alta.

```
date_sir    segment     word  public 'data'
        sir1    db      1000 dup (7,0f0h)    ; şirul sursă
        lung1   equ     $ - sir              ; lungimea şirului sursă
        sir2    db      1000 dup (2 dup (?)) ; rezervare pt. şir dest.
        ptr_sir1        dd      sir1         ; pointer sir1
        ptr_sir2        dd      sir2         ; pointer sir2
date_sir    ends
cod     segment     word  public 'code'
        assume cs:cod, ds:date_sir, es:date_sir
start:  mov     ax, date_sir    ; inițializare registru segment DS
        mov     ds, ax                  ; şi apoi adresele celor două şiruri
        mov     es, ax                  ; sau:  les     di,ptr_sir2
        lea     di, sir2                ;       lds     si,ptr_sir1
        lea     si, sir1
        mov     cx, lung1               ; contorul transferului = lungimea sursei
        cld                             ; direcția de parcurgere a şirului (df=0)
repeta:
        lodsb
```

```
        stosb                   ; sau: movs sir2,sir1 , sau movsb
        loop    repeta                  ; sau:  rep     movsb
        mov     ax, 4c00h               ; revenire DOS
        int     21h
cod     ends
end     start
```

2) Determinarea poziției unui anumit caracter (sau a unui şir de caractere) într-un fişier sursă de pe disc.

```
.model small
.stack  10h
.data
car     equ     'A'     ; caracterul de identificat (sau şirul)
lung equ 2048; dimensiunea maximă a fişierului-4 sectoare
buffer db lung dup (?); spațiu de mem. pentru fişier
nume_fis db 'fisier.asm', 0 ; nume fişi.- ASCIIZ, adică după
                        ; numele poate fi precedat şi de calea de acces
pozitie dw      ?       ; poziția caracterului în fişier
nr_logic        dw      ? ; numărul logic atribuit fişierului
contor dw       ?       ; numărul efectiv de caractere citite
mes_lipsa               db      'nu exista fisier cu acest nume$'
mes_err_cit             db      'eroare de citire de pe disc$'
mes_car_lipsa           db      'caracterul cautat nu este in fisier$'
.code
start:  mov     ax, @data       ; inițializare DS
        mov     ds, ax
        mov     ah, 3dh         ; apel DOS pentru deschidere de fişier
        mov     al, 0           ; în modul "citeşte" (1-scrie,2-citire/scriere)
        lea     dx, nume_fis    ; adresa numelui fişierului
        int     21h             ; apel funcție 'deschide fişier'
        jc      lipsa_fis
        mov     nr_logic, ax    ; se depune numărul sectorului
        mov     bx, ax          ; şi în BX, pentru funcția de citire
        mov     cx, lung        ; contor număr maxim de caractere citite
        lea     dx, buffer      ; adresa unde se vor depune caracterele
        mov     ah, 3fh         ; funcția de citire din fişier
        int     21h
        jc      err_cit         ; dacă a apărut eroare al citire
        mov     contor, ax      ; la 'contor' se depune numărul de car.
        mov     cx, ax          ; contor de căutare 'car'
        push ds                 ; se încarcă în ES adresa de segment
        pop     es              ; a 'buffer'-ului
        lea     di, buffer      ; şi în registrul (DI) offsetul acestuia
        mov     al, car         ; caracterul de căutat
        cld                     ; stabilire direcție de parcurgere, (DF)=0
repne   scasb           ; continuă scanarea până-l găseşte
```

```
        je      gasit           ; dacă nu s-a găsit nu se face saltul
        lea     dx, mes_car_lipsa     ; şi se tipăreşte mesajul
        mov     ah, 9   ; mesajul: 'caracterul cautat nu este in fisier'
        int     21h
        jmp     gata
gasit:  dec     di      ; poziția caracterului căutat, (DI)-1
        mov     pozitie, di     ; deoarece (DI) a fost actualizat după      jmp     gata
        ; scanare
lipsa_fis:
lea     dx, mes_lipsa           ; nu s-a găsit fişierul cu
        mov     ah, 9                   ; numele specificat
        int     21h
        jmp     gata
err_cit:
        lea     dx, mes_err_cit         ; eroare la citirea fişierului
        mov     ah, 9
        int     21h
gata:   mov     ah, 3eh         ; închiderea fişierului deschis
        mov     bx, nr_logic
        int     21h
        mov     ax, 4c00h               ; revenire DOS
        int     21h
end     start
```

Instrucțiuni de transfer al controlului programului

Instrucțiunile de transfer al controlului programului operează asupra lui IP şi CS. La apariția unui astfel de transfer, coada de instrucțiuni nu mai conține instrucțiunea următoare şi EU va transmite noua adresă la BU, care va obține instrucțiunea următoare direct din memorie, utilizând noile valori pentru CS:IP; instrucțiunea va fi transferată către IU şi EU, pentru execuție, şi BU apoi începe reumplerea cozii de la noua locație.

Există patru tipuri de instrucțiuni de transfer control:

- instrucțiuni de transfer necondiționat;
- instrucțiuni de transfer condiționat;
- instrucțiuni de ciclare;
- instrucțiuni de întrerupere;

Dintre acestea numai instrucțiunile de întrerupere afectează indicatorii.

Instrucțiuni de transfer control, necondiționat

Transferul controlului se poate face la o locație (țintă) în segmentul de cod curent (transfer intrasegment/NEAR) sau la un segment de cod diferit (transfer intersegment/FAR).

CALL <nume_procedură>

Această instrucțiune activează o procedură, salvând adresa de revenire, în stivă, pentru a permite unei instrucțiuni RET (return), din procedură, să transfere controlul înapoi la instrucțiunea următoare lui CALL.

Asamblorul generează tipuri diferite de instrucțiuni maşină pentru CALL, după modul cum programatorul a definit numele procedurii, cu atributul NEAR sau FAR.
Pentru returnarea corectă a controlului, tipul instrucțiunii CALL trebuie să fie potrivit cu tipul instrucțiunii RET. Diferitele forme ale instrucțiunii CALL permit ca adresa procedurii țintă să fie obținută direct din instrucțiune (direct), sau de la o locație de memorie sau dintr-un registru referit de instrucțiune (indirect).

a) - CALL direct intrasegment:

(SP) ← (SP) - 2
((SP)+1:(SP)) ← (IP)
(IP) ← (IP) + deplasament

La 386/486 deplasamentul poate avea şi 32 biți, deci adunarea se va face modulo 4G.

```
call    near_etich
call    near_proc
```

b) - CALL direct intersegment:

(SP) ← (SP) - 2
((SP)+1:(SP)) ← (CS)
(CS) ← al 2-lea cuvânt din instr. (adresa de segment)
(SP) ← (SP) - 2
((SP)+1:(SP)) ← (IP)
(IP) ← primul cuvânt din instr. (adresa de offset)

```
call    far_etich;          call    far_proc
```

c) - CALL indirect intrasegment:

(SP) ← (SP) - 2
((SP)+1:(SP)) ← (IP)
(IP) ← registru / memorie;

Exemple:

```
call    cx;                 call    word ptr nume_var
call    word ptr [bx][si];  call    mem_word
```

d) - CALL indirect intersegment, poate fi făcut numai prin memorie:

(SP) ← (SP) - 2
((SP)+1:(SP)) ← (CS)
(CS) ← al 2-lea cuvânt mem. referit de instrucțiune
(SP) ← (SP) - 2
((SP)+1:(SP)) ← (IP)
(IP) ← primul cuvânt mem. referit de instrucțiune

```
call    dword ptr [bx];     call    dword ptr nume_var[bp][si]
```

Când se utilizează apeluri indirecte trebuie asigurat că tipul lui CALL este potrivit cu tipul instrucțiunii de revenire RET->erori.

RET [val_pop] (RETurn from procedure)

Asamblorul generează 2 tipuri de instr. maşină: RET intrasegment dacă procedura a fost definită NEAR, şi RET intersegment, dacă procedura este de tip FAR. Valoarea *val_pop*, care este opțională, este prevăzută pentru a descărca parametrii din stivă.

(IP) ← ((SP)+1:(SP))
(SP) ← (SP) + 2

```
        if      inter_segment then
                (CS) ← ((SP)+1:(SP))
                (SP) ← (SP) + 2
        if      val_pop <> 0 then
                (SP) ← (SP) + val_pop
```

Tipul instrucțiunii RET trebuie să se potrivească cu tipul lui CALL:

call word ptr [bx] ; pentru o procedură de tip FAR.

- se va salva în stivă doar offsetul de revenire (IP), iar la execuția instrucțiunii RET, din procedura de tip FAR, se va reface din stivă, pe lângă IP şi CS, cu următorul cuvânt din stivă (şi se va pierde controlul asupra calculatorului).

call dword ptr [bp][si] ; pentru o procedură de tip NEAR.

- se vor salva în stivă adresele de revenire pentru CS:IP, dar la terminarea procedurii se va reface din stivă numai IP, rămânând în stivă un cuvânt, care în final va duce la pierderea controlului.

JMP <țintă>

Spre deosebire de CALL, JMP nu salvează în stiva nici o informație.

a) *JMP direct intrasegment*, modifică IP prin adunarea deplasamentului relativ al țintei, conținut în instrucțiune la valoarea, actualizată, a lui IP. Auto-relative (relocabile dinamic).

```
        jmp     near_etich      ; salt direct intrasegment
        jmp     short near_etich; salt de tip short
```

b) - *JMP direct intersegment* (instr. are 5 octeți/286, sau 5/7 la 386):

```
        jmp     far_etich       ; salt în alt segment
        jmp     far ptr etich   ; 'etich' poate fi şi în acelaşi segment
```

c) - *JMP indirect intrasegment*:

```
        jmp     ax                      jmp     tabela[bx]
        jmp     si                      jmp     word ptr [bp][di]
```

d) - *JMP indirect intersegment*, poate fi făcut numai prin memorie: jmp etich_dword

```
        jmp     dword ptr       [bx][si]
```

Programul următor execută diferite secvențe de program, în funcție de opțiunea utilizatorului, introdusă de la tastatură.

```
.model small
.data
executie db     0Dh, 0Ah, 'Executa secventa (1,2,3 sau 4=stop):$'
mes_secv1       db      'S-a executat secventa 1',0dh,0ah,'$'
mes_secv2       db      'S-a executat secventa 2',0dh,0ah,'$'
mes_secv3       db      'S-a executat secventa 3',0dh,0ah,'$'
tab_secv        label   word
                dw      secv1
                dw      secv2
                dw      secv3
                dw      gata
.code
start:  mov     ax, @data       ; inițializare adresa de segment
        mov     ds, ax
iar:    lea     dx, executie    ; solicită secvența dorită, utilizatorului
```

```
        mov   ah, 9          ; se tipăreşte mesajul de selecţie secvenţă
        int   21h            ; apel funcţia 9, tipărire mesaj
        mov   ah, 1          ; apel funcţia 1, citire caracter
        int   21h            ; caracter returnat în AL
        sub   al, 31h        ; intervalul '1'÷'4' -> 0÷3 şi verifică dacă
        jc    iar            ; este în intervalul 0÷3: dacă nu, cere
        cmp   al, 4          ; o valoare, în acest interval, fără a executa
        jnc   iar            ; vreuna dintre secvenţe
        cbw                  ; extensie pe 16 biţi
        mov   bx, ax
        shl   bx, 1          ; *2, pentru a obţine adresa relativă
        jmp  word ptr tab_secv[bx] ; în tabela cu adr. secvenţe
secv1:  lea   dx, mes_secv1; se execută prima secvenţă
        mov   ah, 9
        int   21h
        jmp   short  iar
secv2:  lea   dx, mes_secv2; se execută a doua secvenţă
        mov   ah, 9
        int   21h
        jmp   short  iar
secv3:  lea   dx, mes_secv3; se execută a treia secvenţă
        mov   ah, 9
        int   21h
        jmp   short  iar
gata:   mov   ax, 4c00h              ; revenire DOS
        int   21h
.stack  20h
end
```

Instrucţiuni de transfer control, condiţionat

Jcond <ţintă>

Toate salturile condiţionate sunt de tip SHORT, adică ţinta trebuie să fie în segmentul de cod curent, în intervalul de [-128, +127] octeţi faţă de instrucţiunea de transfer (de ex. JMP 00, realizează saltul la primul octet al următoarei instrucţiuni). La 386/486 aceste instrucţiuni permit şi realizarea unui salt de tip NEAR, adică în cadrul aceluiaşi segment (64K sau 4G, depinde de atributul de dimensiune adresă). Deoarece adresa de salt este determinată prin adunarea deplasamentului relativ al ţintei, la IP-ul actualizat, toate salturile condiţionate sunt autorelative.
În cazul când se doreşte un salt condiţionat pe o distanţă mai mare de 128 octeţi, la 286, sau un salt într-un alt segment, atunci trebuie să se utilizeze două instrucţiuni de salt: un salt condiţionat (pentru condiţia negată, adică, de exemplu JE, pentru JNE), în intervalul [-128,+127], iar, imediat după instrucţiunea de salt condiţionat, o instrucţiune de salt, necondiţionat, pe distanţa dorită.

JZ / JE ZF = 1, dacă este egalitate (d) = (s)
JNZ / JNE ZF = 0, dacă nu este egalitate (d) <> (s)

JL / JNGE SF <> OF, dacă (d) < (s), pentru numere cu semn
JLE / JNG SF <> OF sau ZF = 1, (d)<=(s), numere cu semn
JNL / JGE SF = OF, dacă (d) >= (s), pentru numere cu semn
JNLE / JG SF = OF şi ZF = 0, (d) > (s), pentru numere cu semn
JB /JNAE /JC CF = 1, dacă (d) < (s), numere fără semn
JBE / JNA CF = 1 sau ZF = 1, (d)<=(s), numere fără semn
JNB /JAE /JNC CF = 0, dacă (d) >= (s), numere fără semn
JNBE / JA CF = 0 şi ZF = 0, (d) > (s), pentru numere fără semn
JP / JPE PF = 1, dacă numărul are paritate pară
JNP / JPO PF = 0, dacă numărul are paritate impară
JO/JNO OF = 1/0, dacă este/ nu este depăşire de reprezentare
JS /JNS SF = 1/0, dacă numărul este negativ/ pozitiv
J(E)CXZ (E)CX = 0, test asupra registrului ECX sau CX

condiție	ZF	CF
d > s	0	0
d = s	1	0
d < s	0	1

pentru numere fără semn

condiție	OF	SF	ZF
d > s	0/1	0/1	0
d = s	0	0	1
d < s	0/1	1/0	0

pentru numere cu semn

salt pentru	*numere fără semn*	*numere cu semn*
d = s	**JE / JZ**	**JE / JZ**
d <> s	**JNE / JNZ**	**JNE / JNZ**
d < s	**JB / JNAE / JC**	**JL / JNGE**
d > s	**JA / JNBE**	**JG / JNLE**
d <= s	**JBE / JNA**	**JLE / JNG**
d >= s	**JAE / JNB / JNC**	**JGE / JNL**

Determinarea valorii maxime dintr-un şir de valori (cu semn).

```
.model small
.stack 10h
.data
        sir     db      10,20,-30,100,-100,200
        lung    dw      $-sir
        maxim db        ?
        mes_sir_vid     db      'sir vid de valori$'
.code
        mov     ax, @data       ; iniţializare registru segment
        mov     ds, ax
        mov     cx, lung        ; contor număr de valori din şir
        jcxz    sir_vid         ; dacă şirul este vid se tipăreşte mesaj
        lea     si, sir         ; index elemente din şir
        cld                     ; direcţia de parcurgere
        mov     bl, [si]        ; se iniţializează valoarea maximă din şir
        inc     si      ; actualizare index elemente
        dec     cx      ; actualizare contor
```

```
        jcxz    gata    ; dacă a fost un singur element s-a terminat
iar:    lodsb           ; citeşte element curent din şir
        cmp     bl, al          ; se compară cu maximul curent
        jge     urm     ; dacă max > val. curentă trece la elem. următor
                        ; pentru numere fără semn se înlocuieşte jge cu jnc
        mov bl, al              ; altfel se actualizează valoarea maximă
urm:    loop    iar     ; se reia ciclul dacă mai sunt elem. în şir
gata:   mov     maxim, bl       ; se depune valoarea maximă
iesire: mov     ax, 4c00h       ; revenire DOS
        int     21h
sir_vid:
        lea     dx, mes_sir_vid         ; se tipăreşte mesajul
        mov     ah, 9           ; 'sir vid de valori'
        int     21h
        jmp     iesire          ; revenire DOS
end
```

Instrucțiuni de ciclare

Aceste instrucțiuni pot fi folosite pentru a controla repetiția unor cicluri soft. Ele permit o programare ușoară a structurilor de control de tip ciclu cu test la sfârșit. Ele utilizează drept contor, al ciclului, registrul CX. Instrucțiunile de ciclare sunt auto-relative, și pot transfera controlul la o locație (țintă) aflată în intervalul [-128,+127] de octeți față de ele, deci realizează numai transferuri de tip SHORT (atât pentru 286 cât și pentru 386/486). Aceste instrucțiuni nu modifică nici un indicator.

LOOP <short_etich> (LOOP control with CX counter)

(CX) ← (CX) - 1
if (CX) <> 0 then
    (IP) ← (IP) + deplasament (short_etich);

Calculul sumei elementelor unui șir de tip cuvânt.

```
        mov     cx, lung_sir    ; contor număr de elemente
        mov     ax, 0           ; suma va fi pe două cuvinte
        mov     bx, ax          ; şi se inițializează cu 0
        mov     si, ax  ; index pentru adresarea elementelor
aduna:
        add     ax, sir[si]
        adc     bx, 0   ; transporturile se acumulează în BX
        add     si, type sir    ; actualizare index
        loop    aduna           ; dacă nu e gata se reia adunarea
        mov     suma, ax        ; se depune rezultatul, începând cu
        mov     suma[2], bx     ; cu cuvântul mai puțin semnificativ
```

LOOPNE / LOOPNZ **<short_etich>**

(LOOP while Not Equal/Zero, with CX counter)

(CX) ← (CX) - 1
if ((CX) <> 0) and (ZF=0) then
(IP) ← (IP) + deplasament (short_etich);

Determinarea ultimului element dintr-o listă înlănțuită. Ultimul element dintr-o listă va conține valoarea zero în câmpul de adresă.

```
        lea     bx, offset cap_lista    ; adresa de început a listei
        mov     cx, N                   ; dimensiunea maximă a listei
urm: mov bx, [bx + lung_info]           ; adresa elementului următor
        cmp     bx, 0                   ; se compară cu 0
        loopne urm              ; se reia dacă nu s-a găsit
        je gasit            ; s-a găsit ultimul element din listă,  adr. în BX
; dacă nu se execută saltul → nu s-a găsit ultimul element printre cele N elemente
```

Instrucțiuni de întrerupere

Aceste instrucțiuni permit rutinelor de servire a întreruperilor să fie activate prin program la fel ca la apariția unei întreruperi hardware externe. Singura instrucțiune ce alterează indicatorii este IRET.

INT <tip_întrerupere> (call to INTerrupt procedure)

(SP) ← (SP) -2
((SP)+1:(SP)) ← indicatori
(tf), (if) ← 0
(SP) ← (SP) - 2
((SP)+1:(SP)) ← (CS)
(CS) ← (n * 4 + 2)
(SP) ← (SP) - 2
((SP)+1:(SP)) ← (IP)
(IP) ← (n * 4)

INTO (INTerrupt on Overflow)

Această instrucțiune generează întrerupere dacă OF=1.

BOUND <dest_reg>, <sursa_mem>

(check array index against BOUNDs)

Determină dacă valoarea, cu semn, conținută în registrul destinație (16/32 biți), specificat de instrucțiune, se află între limitele specificate de operandul sursă, care trebuie să fie operand din memorie (16/32 biți). Se compară operandul destinație cu două cuvinte (sau cuvinte duble), aflate în memorie de la adresa specificată de cel de-al doilea operand. Dacă valoarea destinației nu este între cele două limite (la locații succesive), specificate de sursă, din memorie, se generează întrerupere pe nivelul 5 (INT 5).

IRET / IRETD (Interrupt RETurn)

Transferă controlul înapoi la punctul de întrerupere, prin refacerea IP, CS şi a indicatorilor, din stivă. Astfel instrucțiunea IRET afectează toți indicatorii, prin restaurarea lor, la valorile anterior salvate.

Împărțirea unui număr de 32 biți (A_1,A_0) la un număr de 16 biți (B). Rezultatul, câtul, poate avea, cel mult, 32 de biți (Q_1,Q_0), iar restul 16 biți (R_0).
Se va încerca mai întâi o împărțire obişnuită, şi dacă câtul este de 16 biți, operația este terminată. Dacă nu, deci dacă câtul este de 32 de biți atunci se va face împărțirea în modul următor:

$$A_1 * 2^{16} + A_0 = (Q_1 * 2^{16} + Q_0) * B + R_0$$

unde $Q_1 = [A_1 / B]$

şi deci $A_1 = Q_1 * B + R_1$

înlocuind A_1 în prima relație rezultă:

$$R_1 * 2^{16} + A_0 = Q_0 * B + R_0$$

Deci ordinea operațiilor va fi:

- se împarte A_1 la B şi rezultă: Q_1 şi R_1;
- se împarte ($R_1 * 2^{16} + A_0$) la B şi rezultă: Q_0 şi R_0.

```
;       procedura de împărțire, primeşte:
;               (DX, AX) - deîmpărțit (32 biți)
;               (BX)    - împărțitor (16 biți)
;       şi va returna:
;               (BX, AX) - câtul
;               (DX)    - restul
pdiv            proc
                cmp bx, 0       ; împărțitor = 0 ?
                jnz cdiv        ; dacă nu se continuă împărțirea, altfel
                int 0 ; se apelează procedura de eroare la împărțire
                jmp eroare
cdiv:   push es                 ; se salvează registrele de lucru
                push di
                push cx
                mov di, 0
                mov es, di      ; ES:DI=0:0, adresa întrerupere nivel 0
                push es:[di]    ; se salvează adresa de tratare întrerupere
                push es:[di+2] ; nivel 0 (eroare la împărțire) în stivă
; se înlocuieşte adresa salvată (proc. de eroare la împărțire) cu
; adresa procedurii prezentate anterior
                lea cx, int_0   ; offset procedura de împărțire
                mov es:[di], cx         ; se depune în locul proc. de eroare
                mov cx, cs              ; adresa de segment a noii proceduri
                mov es:[di+2], cx       ; se pune în locul celei de eroare
;       se încearcă împărțirea normală
                div bx ; dacă câtul are 16 biți s-a terminat şi BX = 0
; dacă nu, deci dacă câtul are 32 biți, se va genera întrerupere
; pe nivelul 0, care a fost înlocuită cu procedura de împărțire
                xor bx, bx              ; BX = 0, câtul este doar în AX
revenire: pop es:[di+2]         ; se reface adresa procedurii
                pop es:[di]     ; de întrerupere, nivel 0, eroare împărțire
                pop cx          ; se refac registrele salvate în stivă
                pop di
                pop es
```

```
                ret
; procedura propriu-zisă de împărțire, care a fost pusă în locul
; procedurii de tratare eroare la împărțire (nivel 0)
int_0:  push bp         ; salvare BP, pentru a accesa stiva
                mov bp, sp      ; se va înlocui adresa salvată, a instr.
                ; următoare împărțirii cu cea a instrucțiunii de 'revenire'
                mov word ptr [bp+2], offset revenire
                        ; sau, întrucât instr. 'xor bx, bx' are doi octeți.
                        ; se poate utiliza şi: add word ptr [bp+2], 2
                push ax                 ; se salvează A0
                mov ax, dx              ; (AX) ← A1
                sub dx, dx              ; pregătire deîmpărțit (DX=0)
;       se realizează împărțirea A1/B -> (R1,Q1) = (DX,AX)
                div bx
                pop cx          ; CX = A0
                push ax                 ; se salvează Q1
                mov ax, cx              ; (DX,AX) = (R1,A0)
;       se realizează împărțirea (R1,A0)/B -> (R0,Q0) = (DX,AX)
                div bx          ; (DX,AX) = (R0,Q0)
                pop bx          ; se reface câtul: (BX) = Q1
                pop bp
                iret
eroare:         . . . . .   ret
pdiv    endp
```

Instrucțiuni de control procesor

Aceste instrucțiuni permit programelor să controleze diferite funcțiuni ale procesorului. Un grup de instrucțiuni actualizează indicatorii, altul este utilizat pentru sincronizarea microprocesorului cu evenimente externe, şi mai există o instrucțiune care nu realizează nimic (NOP).

Instrucțiuni pentru poziționarea indicatorilor

CLC (CLear Carry flag)

CF ← 0

STC (SeT Carry flag)

CF ← 1

CMC (CoMplement Carry flag)

CF ← not (CF)

Aceste instrucțiuni de poziționare a lui CF se utilizează la operațiile de rotație, deplasare sau la adunări/scăderi de numere reprezentate pe mai mulți octeți/cuvinte etc.

CLD (CLear Direction flag)

DF ← 0

STD (SeT Direction flag)

DF ← 1

Acest indicator va determina direcția de parcurgere a şirurilor.

CLI (CLear Interrupt enable flag)

IF ← 0

STI (SeT Interrupt enable flag)

IF ← 1

Indicatorul IF dezactivează sau activează, prin poziționare pe 0, respectiv 1, întreruperile externe, ce apar pe linia INTR. Sunt dezactivate numai întreruperile mascabile (INTR), nu şi întreruperea nemascabilă NMI, care este servită, indiferent de poziția lui IF. La activarea sistemului de întreruperi (STI), întreruperea, pe linia INTR, este recunoscută după execuția instrucțiunii următoare lui STI.

Instrucțiuni de sincronizare cu evenimente externe

HLT (HaLT)

Determină intrarea proc. în starea 'halt', din care iese doar dacă:

- se activează linia RESET;
- se transmite semnal de întrerupere pe linia NMI;
- se transmite semnal de întrerupere pe linia INTR, dacă întreruperile nu au fost dezactivate.

Această instrucțiune se poate folosi la sfârşitul unei bucle soft, în situația în care programul trebuie să aştepte un eveniment extern (întrerupere de la un dispozitiv extern). În acest caz se va salva în stivă adresa instrucțiunii următoare lui HLT.

WAIT (WAIT)

Determină intrarea proc. în starea 'wait', cât timp linia BUSY\ este activă. Procesorul intră într-o stare 'idle', şi repetă testarea liniei BUSY\, ciclic la intervale de şase perioade de ceas. Când linia BUSY\ devine inactivă, se continuă execuția programului cu instrucțiunea următoare lui WAIT.

Această stare poate fi întreruptă, printr-un semnal de întrerupere, dar după tratarea întreruperii se reintră în această stare, întrucât se salvează în stivă adresa acestei instrucțiuni.

ESC <Cod_Op>,<sursa> (ESCape)

Această instrucțiune este specifică coprocesorului matematic, şi ea este identificată prin primii cinci biți ai codului de operare care sunt 11011; ceilalți biți din câmpul codului operației identifică tipul operației coprocesorului. De fapt coprocesorul urmăreşte magistrala sistemului, şi când identifică execuția acestei instrucțiuni, el o capturează; dacă operandul sursă este un registru procesorul nu face nimic cu el, iar dacă este un operand din memorie, îl citeşte dar îl ignoră. În schimb coprocesorul poate captura acest operand când este citit din memorie.

LOCK (assert LOCK# signal prefix)

Acesta este un prefix, de un octet, care determină activarea semnalului de magistrală LOCK, cât timp se execută instrucțiunea precedată de acest prefix. Într-un sistem multiprocesor, acest semnal poate fi utilizat pentru a asigura procesorului acces exclusiv la utilizarea magistralei, şi deci a unei resurse (memorii) partajate.

```
mov    al, 1
astept:
       lock   xchg   al, semafor    ; citire şi setare semafor
       test   al, al          ; testare semafor = 0 ?,
       jnz    astept          ; dacă nu, resursa este ocupată
                              ; şi se aşteaptă eliberarea ei
       . . . .    .           ; utilizare exclusivă a resursei
       mov    semafor, 0      ; eliberare resursă
```

NOP (No OPeration)

După cum spune şi numele acestei instrucțiuni, ea nu realizează nici o prelucrare; nu afectează nici un indicator. Ea durează trei perioade de ceas, şi are codul maşină al instrucțiunii XCHG AX,AX, sau XCHG EAX,EAX.