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

26. Exemple de programe pentru fiecare tip de instrucțiuni.

Instrucțiuni de transfer date.

Exemplu: să inversăm, între ei, octeții dintr-un şir de cuvinte.

```
        .model small
        .data
          mesajdw      'Ex','em','pl','u ','pr','og','ra','m ','2$'
          lung_mesaj   equ     ($-mesaj)/2
          linie_noua   db      0dh,0ah,'$'
.code
        start:  mov     ax,@data        ; inițializare registru segment DS
                mov     ds,ax           ; cu adresa segmentului de date
                lea     dx,mesaj        ; tipărim mesajul sub forma inițială
                mov     ah,9            ; utilizând funcția 9, din DOS :
                int     21h             ; "xEmelp urpgora m"
                lea     dx,linie_noua ; se trece pe o linie noua
                mov     ah,9
                int     21h
                ; pentru a aduce mesajul la forma dorită trebuie inversați
                ; octeții, fiecărui cuvânt între ei
                lea     si,mesaj
                mov     cx,lung_mesaj
iar:    mov     ax,[si]         ; se ia câte un cuvânt
                xchg    al,ah           ; şi se inversează octeții
                mov     [si],ax         ; se depune la aceeaşi adresă
                add     si,type mesaj   ; se actualizează adresa curentă
                loop    iar             ; şi se continuă pentru tot şirul
                                        ; se tipăreşte mesajul astfel obținut
                lea     dx,mesaj
                mov     ah,9            ; se va tipări mesajul:
                int     21h             ; 'Exemplu program 2'
                mov     ax,4c00h        ; revenire DOS
                int     21h
end     start
```

Instrucțiuni de transfer specifice acumulatorului

Programul citeşte de la tastatură codul ASCII al unei taste şi afişează acest cod (sub forma a două cifre hexa).

```
seg_date        segment
                tab_conv        db      '0123456789ABCDEF'
                mesaj   db      '-are codul ASCII-'
                tasta   db      2 dup (?), 0dh,0ah,'$'
        seg_date        ends
        seg_cod         segment
```

```
        assume cs : seg_cod, ds : seg_date
        start:
                mov     ax,seg_date
                mov     ds,ax
                mov     ah,1            ; citire cu ecou a unei taste
                int     21h             ; fără ecou este funcția 8
                mov     ah,al           ; salvare cod tastă
                lea     bx,tab_conv     ; inițializare BX, pentru XLAT
                and     al,0fh          ; se reține doar a doua tetradă
                xlat    tab_conv        ; codul ASCII al acestei tetrade
                mov     tasta+1,al      ; este cea de-a doua cifră a codului
                mov     al,ah           ; codul inițial al tastei
                mov     cl,4    ; contor număr de deplasări la dreapta
                shr     al,cl   ; deplasare logică la dreapta cu 4 biți
                xlat    tab_conv        ; conversia primei tetrade
                mov     tasta,al        ; codul ASCII al primei tetrade
                lea     dx,mesaj
                mov     ah,9            ; se tipăreşte codul tastei
                int     21h
                mov     ax,4c00h        ; revenire DOS
                int     21h
        seg_cod         ends
end start
```

Instrucțiuni Aritmetice

Programul adună două numere, fără semn, reprezentate pe mai multe cuvinte (octeți), dar de lungimi diferite, iar rezultatul se va depune peste numărul mai lung.

```
add_data_2      segment
        primul  dw      0a8adh, 7fe2h, 0a876h,0
        lung_1  dw      ($ - primul)/2
        al_doilea       dw      0fedch, 0abcdh, 0cdefh, 52deh, 5678h, 0
        lung_2  dw      ($ - al_doilea)/2
add_data_2      ends
multi_add_2     segment
        assume cs: multi_add_2, ds: add_data_2
start:
        mov     ax,add_data_2
        mov     ds,ax
; se determină cel mai mare număr, unde se pune rezultatul
; vom considera că      (BX)= adresa numărului mai lung
;                       (DX)= dimensiunea numărului mai lung
;                       (BP)= adresa numărului mai scurt
;                       (CX)= dimensiunea acestuia
; numărul de octeți ai numărului mai mic va controla bucla_1
```

```
; în care ambele numere au cuvinte ce trebuie adunate
; diferența dimensiunilor celor două numere va controla bucla_2
; necesară dacă apare transport, pentru propagarea acestuia
        mov     dx,lung_2       ; presupun, inițial numărul al
        lea     bx,al_doilea    ; doilea mai mare
        mov     cx,lung_1
        lea     bp,primul
        cmp     dx,cx           ; verific presupunerea făcută
        jge     num2_mai_mare       ; dacă e respectată se continuă
        xchg    bx,bp           ; dacă nu se schimbă între ele conținutul
        xchg    cx,dx           ; perechilor de registre, respective
num2_mai_mare:
        sub     dx,cx           ; determin diferența dintre lungimile lor
        clc                     ; (CF)=0
        mov     si,0            ; se inițializează indexul elementelor
bucla_1:
        mov     ax,ds:[bp][si]
        adc     [bx][si],ax
        inc     si              ; actualizare index
        inc     si
        loop    bucla_1         ; (CX)=contor pentru adunare
        mov     cx,dx           ; contor pentru propagarea transportului
bucla_2:
        jnc     gata            ; dacă nu mai este transport de propagat
        adc     word ptr [bx][si],0
        inc     si
        inc     si
        loop    bucla_2
; trebuie testat (CF), şi dacă avem transport, adică se
; depăşeşte dimensiunea inițială a numărului mai lung,
; transportul trebuie memorat la adresa următoare,
; dacă s-a rezervat spațiu, sau semnalată depăşirea: jc eroare . . .
gata:   mov     ax,4c00h
        int     21h
multi_add_2  ends
end     start
; (DI) = va conține extensia de semn a numărului mai scurt
num2_mai_mare:
        push    cx      ; salvare lungime număr mai scurt
        mov     di,0    ; extensie de semn pentru număr pozitiv
        mov     si,bp   ; adresa de început a numărului mai scurt
        shl     cx,1    ; lungimea * 2 = număr octeți ai numărului
        add     si,cx   ; se poziționează pe primul element
        sub     si,2    ; care conține bitul de semn
        mov     ax,[si] ; şi îi testăm bitul de semn
        cmp     ax,0    ; dacă este pozitiv se continuă cu valoarea
```

```
        jp      cont    ; inițială pentru (di)=0
        mov     di,0ffffh ; altfel (di)=ffffh, extensie semn număr negativ
cont:   pop     cx      ; refacerea contorului
        sub     dx,cx   ; determin diferența dintre lungimile lor
        clc             ; (CF)=0
mov     si,0    ; se inițializează indexul elementelor
bucla_1:
        mov     ax,ds:[bp][si]
        adc     [bx][si],ax
        inc     si      ; actualizare index
        inc     si
        loop    bucla_1         ; (CX)=contor pentru adunare
        mov     cx,dx   ; contor pentru propagarea transportului
bucla_2:
        adc     word ptr [bx][si],di    ; şi a semnului numărului mai scurt
        inc     si
        inc     si
        loop    bucla_2
; în acest punct se testează OF pentru a detecta o depăşire
; a dimensiunii inițiale a numărului mai lung
```

Instrucțiunile de înmulțire / împărțire

Afişarea în zecimal a conținutului registrului AX:

```
; (AX) conține o valoare care este în intervalul 0÷65535
; dacă se împarte numărul la 100 se obține ca prim rest
; o valoare ce conține ultimele două cifre (zeci, unități)
; dacă se împarte câtul anterior din nou la 100 se va obține
; o valoare ce conține următoarele două cifre (mii, sute)
; iar acest ultim rest va avea valoarea primei cifre
; a zecilor de mii, care este în intervalul: 0 - 6.
        mov     bx,100          ; împărțitor
        mov     dx,0            ; extensie semn deîmpărțit pozitiv
        div     bx              ; dx = ultimele 2 cifre (zeci, unități)
        mov     cx,dx           ; ax = primele 3 cifre, cx = ultimele 2 cifre
        mov     dx,0            ; extensie semn deîmpărțit pozitiv
div     bx              ; dx = cifrele: mii, sute; ax = zeci de mii
        push dx                 ; se salvează dx deoarece va fi modificat
        add     al,30h          ; se tipăreşte cifra zecilor de mii(0÷6)
        mov     dl,al           ; caracterul de tipărit în DL
        mov     ah,2            ; se apelează funcția 2 din DOS
        int     21h             ; care tipăreşte caracterul din (DL)
        pop     ax              ; se iau din stiva valoarea pt. mii, sute
        aam                     ; conversie la format zecimal neîmpachetat
        add     ax,3030h        ; conversie la cod ASCII
        mov     dx,ax           ; se salvează cele două cifre
        xchg    dh,dl           ; se tipăreşte cifra miilor
```

```
        mov     ah,2
        int     21h
        xchg    dh,dl           ; se tipăreşte cifra sutelor
        mov     ah,2
        int     21h
        mov     ax,cx           ; ultimele două cifre: zeci, unități
        aam                     ; conversie la format zecimal neîmpachetat
        add     ax,3030h        ; conversie la cod ASCII
        mov     dx,ax           ; se salvează cele două cifre
        xchg    dh,dl           ; se tipăreşte cifra zecilor
        mov     ah,2
        int     21h
        xchg    dh,dl           ; se tipăreşte cifra unităților
        mov     ah,2
        int     21h
```

Instrucțiuni de prelucrare la nivel de bit

Determinarea numărului de biți egali cu 1 dintr-o variabilă.

```
.model small
.stack  100h
.data
        lung_var        equ     8
        variabila       dw      lung_var        dup (0acfh)
        nr_unitati      db      ?
.code
assume          cs: @code, ds: @data
start:  mov     ax, @data
        mov     ds, ax
        mov     si, 0           ; indexul curent al cuvintelor din variabila
        mov     dl, lung_var    ; contor număr de cuvinte
        mov     bl, 0           ; contor număr de unități găsite
bucla2: mov     cx, 16  ; contorul de biți pentru un cuvânt
        mov     ax, variabila[si]       ; se citeşte cuvântul curent
bucla1:
        rcl     ax, 1                   ; o rotație pentru a deplasa un bit
        adc     bl, 0                   ; se contorizează numărul de unități
        loop    bucla1                  ; pentru un cuvânt
        add     si, type variabila      ; se actualizează indexul
        dec     dl                      ; se testează dacă mai sunt cuvinte
        jnz     bucla2                  ; dacă da se reia citirea cuvintelor
        mov     nr_unitati, bl          ; dacă nu se depune rezultatul
        mov     ax, 4C00h               ; revenire DOS
        int     21h
end
```

Aceeaşi problemă poate fi rezolvată utilizând instrucțiunile *BSF* şi *BTR*, astfel:

```
        mov    cx, lung_var  ; contor dimensiune variabilă
        mov    bl, 0         ; contor unități
        mov    si, 0         ; index cuvinte
bucla:  mov    ax, variabila[si]
 scanare:
        bsf    dx, ax        ; determină primul bit 1 (indexul în DX)
        jz     gata_cuv      ; dacă ZF=0, toți biții sunt 0
        btr    ax, dx        ; transferă 1 din AX, din poziția DX în CF,
        adc    bl, 0         ; iar bitul =0, sau inc bl, numără unitățile
        jmp    scanare       ; se reia scanarea cuvântului curent
gata_cuv:
        add    si, 2         ; indexul cuvântului următor
        loop bucla     ; decrementare contor număr de cuv. variabilă
                       ; dacă nu s-a terminat variabila ia cuv. următor
```

Tipărirea conținutului registrului DX, în format octal:

```
tip_car         proc   far
; procedura tipăreşte caracterul al cărui cod ASCII
; îl primeşte în registrul AL
                push   dx          ; se salvează registrul de lucru DX
                mov    dl, al      ; se apelează funcția 2 din DOS care
                mov    ah, 2       ; tipăreşte caracterul al cărui cod
                int    21h         ; ASCII se transmite în registrul DL
                pop    dx          ; se reface registrul salvat
                ret
tip_car         endp
tip_octal       proc   far
        ; tipăreşte în octal valoarea, fără semn transmisă în DX
                push   cx      ; se salvează registrele de lucru
                push   ax
; prima cifră de tipărit este doar de 1 bit
        rol    dx, 1           ; care este rotit pe ultimul bit
        mov    al, dl          ; şi dusă în registrul AL
        and    al, 1           ; se şterg ceilalți biți
        add    al, 30h         ; este convertită la codul ASCII
        call   tip_car         ; şi se tipăreşte
                               ; următoarele 5 cifre sunt de câte 3 biți
        mov    cx, 5           ; contor număr de cifre de tipărit
octal1: push   cx              ; se salvează contorul în stivă
        mov    cl, 3           ; contor număr de rotiri la stânga
        rol    dx, cl          ; cifra este rotită pe ultimii 3 biți
        mov    al, dl          ; şi adusă în AL
        and    al, 7           ; sunt şterşi ceilalți biți
        add    al, 30h         ; şi convertită la codul ASCII
        call   tip_car                 ; tipărirea cifrei
```

```
        pop     cx              ; se citeşte valoarea contorului de cifre
        loop    octal1
        pop     ax              ; se refac registrele salvate în stivă
        pop     cx
        ret
tip_octal       endp
```

Instrucțiuni de operare pe şiruri

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    ; iniţializare 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                             ; direcţia 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
```

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 (?); spaţiu 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
```

Determinarea poziției unui şir de caractere într-un fişier:

```
        sir_cardb       'asamblare'
        lung_sir        dw      $-sir_car
        ptr_sir         dd      sir_car
        ptr_buf         dd      buffer
        pozitie         dw      ?       ; poziția şirului de caractere
        . . . . . . . . . . . . . . . . . .
        les     di, ptr_buf
        lds     si, ptr_sir
        mov     cx, contor      ; dimensiune buffer
        sub     cx, lung_sir    ; căutarea se va face începând de la 0
        inc     cx              ; până la (contor) - (lung_sir)
        cld                     ; direcția de parcurgere (DF=0)
reia:   push si; se salvează adresa şirului căutat
        push di         ; se salvează adresa şirului în care se caută
        push cx; se salvează contorul numărului maxim de căutări
        mov     cx, lung_sir    ; contorul de comparații
        repe cmps buffer, sir_car     ; se compară cât timp sunt egale
        pop     cx      ; reface resurse salvate în stivă: contorul
        pop     di      ; şi adrese şiruri: destinație - şirul în care se
        pop     si      ; caută, sursă – (sub)şirul care se caută
        je      gasit   ; salt dacă s-a găsit sir_car în buffer
        inc     di      ; căutarea se va relua de la următorul caracter
        loop    reia    ; din buffer, dacă (contor) <> 0
; -> şirul de caractere nu este în fişierul dat (respectiv în buffer)
        jmp     nu_gasit
gasit:  mov     pozitie, di     ; valoarea salvată pentru (DI)
        . . . . . . . . . . . . . . . . . .
nu_gasit:       ; tipărire mesaj: 'Nu s-a gasit sirul de caractere'
```

Instrucțiuni de transfer al controlului programului

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 selecție secvență
       int  21h            ; apel funcția 9, tipărire mesaj
       mov  ah, 1          ; apel funcția 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 secvențe
       cbw                 ; extensie pe 16 biți
       mov  bx, ax
       shl  bx, 1          ; *2, pentru a obține adresa relativă
       jmp  word ptr tab_secv[bx] ; în tabela cu adr. secvențe
secv1: lea  dx, mes_secv1; se execută prima secvență
       mov  ah, 9
       int  21h
       jmp  short  iar
secv2: lea  dx, mes_secv2; se execută a doua secvență
       mov  ah, 9
       int  21h
       jmp  short  iar
secv3: lea  dx, mes_secv3; se execută a treia secvență
       mov  ah, 9
       int  21h
       jmp  short  iar
gata:  mov  ax, 4c00h             ; revenire DOS
       int  21h
.stack 20h
end
```

Instrucțiuni de transfer control, condiționat

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      $-sir
        maximdb         ?
        mes_sir_vid     db      'sir vid de valori$'
.code
        mov     ax, @data       ; inițializare 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                     ; direcția de parcurgere
        mov     bl, [si]        ; se inițializează 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

Calculul sumei elementelor unui şir 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
```

Determinarea primului element diferit de 0, dintr-un şir.

```
                mov cx, lung_sir        ; contorul şirului
```

```
        mov si, -1              ; inițializare SI
reia:   inc     si
        cmp sir[si], 0  ; se compară cu 0
        loope reia              ; dacă este 0 se reia comparația
        jz sir_zero     ; dacă ZF=1, tot şirul conține numai 0
aici:           ; s-a determinat primul element <> 0, la adresa SI
```

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
```