



UNIUNEA EUROPEANĂ



GUVERNUL ROMÂNIEI



Instrumente Structurale
2007-2013



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

Programare în limbaj de asamblare

27. Definirea și utilizarea procedurilor.

Definirea și utilizarea procedurilor

O procedură este constituită dintr-o secvență de instrucțiuni, care poate fi apelată ori de câte ori este nevoie, și se declară folosind următoarea sintaxă:

```
nume_procedura      proc      [ near / far ]  
                    .  
                    < corpul procedurii >  
                    .  
                    ret  
nume_procedura      endp
```

Parametrii (datele de intrare) pot fi transmiși în diferite moduri:

- **prin registre**; această modalitate poate avea, însă, următoarele dezavantaje:
 - neconvenabilă, dacă se transmit un număr mare de parametri, caz în care numărul registrelor poate fi insuficient, sau
 - neeconomică sub aspectul timpului de execuție, necesitând frecvente salvări și refaceri de registre.
- **prin memorie**, care presupune rezervarea de memorie pentru acești parametri, deci poate fi neeconomică din punct de vedere al spațiului de memorie ocupat (pentru structuri mari de date), mai ales dacă astfel de date sunt necesare doar temporar;
- **prin stivă**, care este metoda cea mai avantajoasă, întrucât permite accesul ușor la informație (cu instrucțiunile PUSH, respectiv POP), iar după utilizare memoria alocată în acest scop este eliberată, deci zona respectivă de memorie devine disponibilă. În cazul transmiterii parametrilor prin stivă, aceasta trebuie manipulată cu atenție de către procedură, astfel încât, în final, în vârful stivei să se găsească adresa de revenire, pentru a se executa corect ultima instrucțiune din procedură: RET.

În ceea ce privește parametrii transmiși prin stivă, aceștia pot fi:

- valoare, se transmite valoarea parametrului;
- adresă, se transmite adresa parametrului, deci operațiile descrise de procedură se vor efectua direct asupra parametrului respectiv; în acest caz, procedura nu mai trebuie să returneze programului apelant parametrul respectiv.

În ceea ce privește returnarea parametrilor, această operație se poate realiza utilizând una din metodele folosite pentru transmiterea parametrilor către procedură, descrise anterior (registre, memorie sau stivă).

În general, pentru definirea unei proceduri se parcurg următoarele etape:

- stabilirea prelucrărilor efectuate de procedură;
- definirea datelor de intrare/ieșire, precum și a modului de transmitere a parametrilor (registre, memorie sau stivă);
- definirea tipului procedurii (NEAR/FAR), mai ales dacă parametrii se transmit prin stivă, situație în care trebuie determinată poziția acestora în stivă;
- pentru directivele simplificate de segmentare nu este necesară precizarea tipului procedurii,

deoarece tipul este determinat de asamblor, din directiva MODEL; dacă în procedură se utilizează nume simbolice, pentru a obține în mod corect accesul la acestea trebuie salvat DS în stivă, după care se încarcă noua valoare și se folosește directiva ASSUME pentru a preciza asocierea segmentului la DS, iar la sfârșit se reface registrul DS cu valoarea din stivă.

- înainte de orice prelucrare trebuie salvate în stivă registrele utilizate de procedură, iar înainte de revenirea în programul apelant, acestea trebuie restaurate.

1) Programul adună câte două numere de la adrese succesive, și depune suma la locațiile imediat următoare.

```

valori segment
    lista1 db 25
           db 37
           db ?
    lista2 db 27
           db 35
           db ?
    .....
valori ends
stiva segment
    dw 4 dup (?)
    varf_stiva label word
stiva ends
adunare segment
    assume cs:adunare, ds:valori, ss:stiva
    mov ax, valori
    mov ds, ax
    mov ax, stiva
    mov ss, ax
    mov sp, offset varf_stiva
    lea si, lista1
    call aduna ; se adună elementele din prima listă
    jc depasire ; dacă apare transport se semnalează
    .....
    lea si, lista2
    call aduna ; se adună elementele din lista2
    jc depasire
    .....
depasire: ..... ; tipărire mesaj de depășire
    mov ax, 4c00h
    int 21h
aduna proc
    push ax
    mov al, [si] ; primul element din listă
    add al, [si+1] ; se adună cu următorul
    mov [si+2], al ; și se depune rezultatul în continuare
    pop ax

```

```

        ret
    aduna endp
adunare ends
end

```

2) Adunarea elementelor unui vector, de orice lungime, cu păstrarea rezultatului în acumulator. Procedura primește în BX - adresa vectorului, iar în CX - lungimea sa, returnează suma în AX.

```

date_vectori segment
    vector1    db    6, 63, 100, 123, -134, 112, 77
    contor1    equ    $ - vector1
    rezultat1  dw    ?
    .....
date_vectori ends
dv    equ    <date_vectori>
pg    equ    <procedura_generala>
procedura_generala segment
    assume cs:pg, ds:dv
    aduna proc
        push    si        ; se salvează registrul de adresare vector
        mov     ax, 0      ; se inițializează suma cu 0
        mov     si, 0      ; inițializare index componente vector
reia:   add     al, [bx][si] ; adunare element curent
        adc     ah, 0      ; se contorizează transporturile
        jc     gata       ; dacă apare depășire se termină procedura
        inc     si        ; actualizare index vector
        loop   reia       ; dacă nu s-a terminat vectorul, se reia
gata:   pop     si        ; se reface registrul salvat
        ret
    aduna endp
start:  mov     ax, dv
        mov     ds, ax
        mov     cx, contor1 ; contorul pentru primul vector
        lea     bx, vector1 ; adresa de start a vectorului
        call    aduna      ; apel procedura de adunare
        mov     rezultat1, ax ; se depune rezultatul
        jc     depasire    ; pt. depășire se afișează mesaj
        .....
    depasire: .....      ; afișare mes. de depășire dim. rez.
        mov     ax, 4c00h
        int     21h
procedura_generala ends
end     start

```

3) Se va rescrie exemplul anterior utilizând, în acest caz, pentru transmiterea parametrilor, stiva. Deci prin stivă se va transmite: adresa rezultatului, adresa vectorului și lungimea acestuia.

```

date segment

```

```

vector1    db    6, 63, 100, 123, -134, 112, 77
contor1    equ   $ - vector1
rezultat1  dw    ?
vector2    db    16, 163, 101, 232, -144, 26, -79
contor2    equ   $ - vector2
rezultat2  dw    ?      . . . . .
mes_depasire db 'eroare-depasire dimensiune rezultat', 0dh,
'$'
date      ends
stiva     segment    stack 'stack'
           dw        20 dup (?)
stiva     ends
utilizare_stiva     segment
assume cs:utilizare_stiva, ds:date, ss:stiva
aduna     proc     far
           push bp           ; se salvează valoarea lui BP, pentru acces
           mov  bp, sp       ; la parametrii din stivă
           push bx           ; se salvează registrele de lucru
           push cx
           mov  cx, [bp+6]    ; CX = lungime vector
           mov  bx, [bp+8]    ; BX = adresa de început a vectorului
           mov  ax, 0         ; inițializare sumă
reia:     add  al, [bx]       ; adună elementul curent
           adc  ah, 0         ; actualizare transport
           jc   iesire       ; dacă apare depășire se termină procedura
           inc  bx           ; actualizare index vector
           loop reia         ; dacă mai sunt elemente se reia adunarea
mov       bx, [bp+10]        ; se citește adresa rezultatului, din stivă
           mov  [bx], ax      ; și se depune rezultatul la această adresă
iesire:
           ;                                     ↑   Adrese mici
           pop  cx           ; CX - salvat
           pop  bx           ; BX - salvat
           pop  bp           ; BP - salvat ← [BP]
           ; IP - revenire
           ; CS - revenire
           ; contor1        ← [BP+6]
           ; offset vector1 ← [BP+8]
           ; offset rezultat1 ← [BP+10]
           ;                                     ↓   Adrese mari
           ret  6           ; descărcarea stivei de parametrii (cei 6 octeți)
aduna     endp
utilizare_stiva     endsaduna_date     segment
assume cs:aduna_date, ds:date, ss:stiva
start:    mov  ax, date
           mov  ds, ax

```

```

lea ax, rezultat1 ; adresa rezultatului se
push ax           ; depune în stivă
lea ax, vector1   ; adresa de început a vectorului
push ax           ; se depune în stivă
mov ax, contor1   ; lungimea vectorului se
push ax           ; se depune în stivă
call aduna        ; se adună elementele primului vector
jc eroare_add ; depășire dim. rezultat (de tip cuvânt)
.....
lea ax, rezultat2 ; adresa rezultatului se
push ax           ; depune în stivă
lea ax, vector2   ; adresa de început a vectorului
push ax           ; se depune în stivă
mov ax, contor2   ; lungimea vectorului se
push ax           ; se depune în stivă
call aduna        ; se adună cel de-al doilea vector
jc eroare_add ; depășire dim. rezultat (de tip cuvânt)
.....
eroare_add:      ; se va tipări un mesaj de eroare
lea dx, mes_depasire
mov ah, 9
int 21h
mov ax, 4c00h
int 21h
aduna_date ends
end start

```

4) Procedură pentru citirea unui număr zecimal întreg, caractere ASCII, și convertirea numărului la o valoare întregă, pe 16 biți.

```

; conversie zecimal (ASCII) → binar (16 biți), rez. în AX.
; numărul este citit cifră cu cifră, termină citirea cu un caracter
; care nu este cifra ASCII, adică nu este în intervalul '0'÷'9',
; sau dacă s-au citit 5 cifre (val. maximă pe 16 biți
; este 65535); se citește un număr de forma:  $c_n c_{n-1} \dots c_1 c_0$ 
; valoarea sa în orice bază este:  $c_n * b^n + c_{n-1} * b^{n-1} + \dots + c_1 * b + c_0$ 
; care se poate scrie:  $(\dots((c_n * b + c_{n-1}) * b + c_{n-2}) * b + \dots + c_1) * b + c_0$ 
; deci relația de calcul este  $val = val * baza + cifra$ 
zeabin proc far
push bx           ; se salvează registrele de lucru
push cx
push dx
mov cx, 5         ; numărul maxim de cifre ale numărului
xor dx, dx       ; inițial val = 0
mov bx, 10       ; baza în care se citește numărul
reia_cit: mov ah, 1 ; citire cu ecou de la tastatură
int 21h

```

```

sub    al, 30h      ; domeniul '0'÷'9' se transformă în 0÷9
jb    gata_cit     ; dacă nu este în intervalul 0÷9
cmp   al, 9        ; s-a terminat citirea cifrelor
ja    gata_cit
mov   ah, 0        ; extensie semn '+' pt. cifra citită
push ax           ; salvare ultima cifră citită
mov   ax, dx       ; (AX) = val
mul   bx           ; (DX,AX) = val * baza
pop   dx           ; ultima cifră (valoare) citită
jc    gata         ; dacă apare depășire, CF=1, și se termină
add   dx, ax       ; val = val * baza + cifra
jc    gata         ; eroare, (CF=1), depășire rez. (65535)
loop  reia_cit     ; nu s-au citit 5 cifre, continuă citirea
gata_cit:
clc    ; ieșire normală, fără eroare, (CF=0)
mov   ax, dx       ; depunerea rezultatului în (AX)
gata:
pop   dx           ; refacere registre salvate
pop   cx
pop   bx
ret
zecbin endp

```

Programul care apelează procedura va testa starea indicatorului CF; dacă (CF) = 1, eroare ; dacă (CF) = 0, rezultatul conversiei în AX.