



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

29. Transferul parametrilor pentru proceduri în limbajele de nivel înalt.

Transferul parametrilor pentru proceduri în limbaje de nivel înalt

În cazul limbajului PASCAL, depunerea parametrilor în stivă se face în ordinea în care aceștia apar în lista de parametri efectivi, la apelul procedurii (adică ordinea în care au fost specificați parametrii formali la definirea procedurii, deci de la stânga la dreapta). Revenirea din procedură se face cu o instrucțiune de forma RET n, pentru a descărca stiva, unde n este numărul de octeți ocupați de argumentele din stivă.

Să considerăm o procedură definită în PASCAL:

```

procedure      modul (a : integer; var b : real);
begin
    .....
end;
```

iar apelul de forma:

```
modul (x, y);
```

Pentru această procedură se va genera secvența:

```

push bp
mov bp,sp
.....
mov ax,[bp+8];(AX)←x
.....
les di,[bp+4]
;(ES):(DI) ← adresa lui y
.....
```

BP	salvare BP
→	
+ 2	adresa de revenire
+ 4	adr_off var. b
+ 6	adr_seg var. b
+ 8	valoarea var. a

În final, se va descărca stiva:

```

pop bp
ret 6
```

Pentru apelul acestei proceduri, se va genera secvența următoare:

```

mov ax,x
push ax          ; se depune in stiva valoarea parametrului x
mov di,offset y
push ds         ; se depune adresa de segment a parametrului y
push di        ; se depune offsetul parametrului y
call modul     ; apelul procedurii
```

Spre deosebire de PASCAL, în limbajul C argumentele sunt puse în ordine inversă față de ordinea în care apar în lista de apel (adică de la dreapta la stânga). Adresele de revenire pot fi

de tip NEAR, ca în exemplul anterior, sau de tip FAR, în funcție de modul de definire al procedurii.

Pointerii salvați în stivă ca referință de parametru sau ca adresă de revenire respectă definiția modelului de memorie stabilit în directiva *model*.

De exemplu, aspectul pentru un singur argument este următorul:

■ modelul medium

push bp	SP, BP→	salvare bp
mov bp,sp	+ 2	adresa revenire - offset
mov di,[bp+6]		adresa revenire - segment
mov ax,[di] ; acces argument	+ 6	adresa argument
; tip cuvânt		

■ modelul large

push bp	SP, BP→	salvare bp
mov bp,sp	+ 2	adresa revenire - offset
les di,[bp+6]		adresa revenire - segment
mov ax,es:[di]; acces argument	+ 6	adresa argument – offset
; tip cuvânt		adresa argument – segment

Pentru a nu modifica prea multe linii, în funcție de modelul utilizat se pot folosi directivele de asamblare condiționată; de exemplu, pentru cazul anterior se poate defini o constantă, astfel:

```

Ptr_FAR          equ    1
.....
.....
                    push  bp
                    mov   bp,sp
.....
ifdef            Ptr_FAR
                    les  di,[bp+6]
                    mov  ax,es:[di]
else
                    mov  di,[bp+6]
                    mov  ax,[di]
endif

```

care, dacă este definită, va considera modelul *large*, iar dacă nu este definită va considera modelul *medium*.

Transferul parametrilor pentru subprograme în PASCAL și C

Să scriem, pentru exemplificare, un program simplu în Pascal, care citește doi vectori, de aceeași lungime îi schimbă între ei și apoi îi afișează. Vom scrie procedura care schimbă între ei cei doi vectori în limbaj de asamblare.

```

program schimba_vectori;
type
    tip_vector = array [1..20] of real;
var
    vector1, vector2 : tipsir;
    n,i : integer;
{$F+}
{$L SCHIMB.OBJ}
procedure InterSchimb (var v1, v2 : tipsir; lung: word); External;
begin
    { se citesc : dimensiunea si cei doi vectori }
    InterSchimb (vector1, vector2, sizeof(vector1));
    { se afiseaza cei doi vectori dupa interschimbare }
end.

```

Directiva de compilare {\$F+} – Force FAR calls – precizează că în cadrul codului generat pentru apeluri de proceduri și funcții se folosește modelul FAR (utilizând acest tip, o procedură poate fi apelată de oriunde din memorie). Dacă directiva este dezactivată {\$F-}, compilatorul determină automat codul care trebuie generat pentru o instrucțiune de apel, în funcție de tipul apelului, în interiorul sau în afara segmentului curent.

Compilatorul, utilizând opțiunea {\$F+}, presupune că rutina se găsește în alt segment, și că orice parametru variabil utilizat de rutină este menționat prin adresele de segment și offset.

Directiva {\$L nume_fișier} – Link object file – impune ca pentru un fișier să fie editate legăturile împreună cu programul compilat. Fișierul pentru care se editează legăturile trebuie să fie în format .OBJ; extensia de nume este implicită. Șirul de caractere care specifică fișierul poate conține și directoarele (calea) de acces către fișier.

Procedura Interschimb, care se află în fișierul SCHIMB.OBJ, este următoarea:

```

.model TPascal ; sau
model large,Pascal
.code
public InterShimb
InterSchimb proc far
stiva struc
    val_BP dw ?
    IP_rev dw ? ; sau adr_rev dd ?
    CS_rev dw ?
    lung dw ?
    ofs_sir1 dw ? ; sau adr_sir1 dd ?
    seg_sir1 dw ?
    ofs_sir2 dw ? ; sau adr_sir2 dd ?
    seg_sir2 dw ?
stiva ends

```

```

push bp
mov bp, sp
push ds ; se salveaza registrele de lucru
push si
push es
push di
push ax
push cx ; urmeaza initializarile de adrese si contor
mov es, [bp].seg_sir1 ; sau les di, [bp].adr_sir1, sau
mov di, [bp].offs_sir1 ; les di, [bp+8]
mov ds, [bp].seg_sir2 ; sau lds si, [bp].adr_sir2, sau
mov si, [bp].offs_sir2 ; lds si, [bp+12]
mov cx, [bp].lung ; sau mov cx, [bp+6]
iar: mov al, ds:[si]
xchg al, es:[di]
mov ds:[si], al
inc si
inc di
loop iar
pop cx ; refacerea registrelor salvate in stiva
pop ax
pop di
pop es
pop si
pop ds
ret 10 ; se descarca stiva de cei 10 octeti
InterSchimb endp
end

```

Linia de declarare a procedurii poate fi și de forma:

```

InterSchimb proc far
Ad_sir1:dword,Ad_sir2:dword,lung:word

```

iar în acest caz, preluarea parametrilor se poate face și mai simplu:

```

les di,Ad_sir2
lds si,Ad_sir1
mov cx,lung

```

pentru care asamblorul va genera o secvență asemănătoare celei prezentate în procedură. Modalitatea de definire și utilizare a unei structuri este definită în paragraful următor.

Vom prezenta în continuare și modul de definire și utilizare a unei funcții în Pascal; funcția prezentată va însuma elementele unui șir de tip word (integer), fără a lua în considerare eventualul transport; dacă dorim, putem să reprezentăm rezultatul pe un format dublu (longint),

contorizând în registrul DX eventualele transporturi, iar rezultatul poate fi returnat în perechea DX,AX, în loc de AX.

Vom utiliza și o variabilă locală, cu toate că în acest caz este inutilă, doar pentru a prezenta modul de definire și utilizare a acesteia în programul principal, la fel ca în exemplul anterior. Funcția va fi definită astfel:

```

function          Suma          (sir: tip_vector; n:word):integer;External;
.model large, Pascal
public Suma
.code
Suma          proc  far
    adr_sir    equ <[bp+8]>
    n          equ <[bp+6]>
    s          equ <[bp-2]> ; variabila locala pentru suma
    push  bp
    mov   bp, sp
    sub  sp, 2          ; rezervare spatiu pentru variabila locala s
    push ds            ; salvarea registrelor de lucru
    push si
    push cx
    lds  si, adr_sir   ; initializare adresa, contor si suma
    mov  cx, n
    mov  ax, 0
    mov  s, ax
    cld                                ; directia de parcurgere a sirului
    jcxz gata          ; in cazul în care contorul a fost 0, s-a terminat
adun: lodsw
    add  s, ax
    loop adun
gata: mov  ax, s
    pop  cx
    pop  si
    pop  ds
    mov  sp, bp        ; eliberarea spatiului alocat variabilei locale s
    pop  bp
    ret  6             ; descarcarea stivei de parametri
Suma          endp
end

```

Să considerăm, acum, și cazul limbajului C. Vom rescrie funcția anterioară, *suma()*, care va fi apelată din funcția *main()*. Întrucât codul generat de compilatorul C transformă denumirile parametrilor funcției *suma()*, adăugând caracterul „_“ la început, codul modulului apelat cu numele *suma()* se va numi de fapt *_suma()*; același lucru se întâmplă și pentru ceilalți parametri utilizați în ambele module, în sensul că parametrii din C utilizați și în modulul scris în limbaj de asamblare vor fi precedați, în limbaj de asamblare, de caracterul „_“. Modulul în limbaj de

asamblare trebuie să salveze, la începutul său, și să refacă, la sfârșit, valorile conținute de registrele BP, SP, CS, DS, SS, DI și SI, dacă programul utilizează aceste registre.

```
.model small, C
public      _suma
.code
    public  _suma
_suma proc
    push   bp
    mov    bp, sp
    push   cx
    push   ds
    push   si
    mov    cx, [bp+8]
    lds   si, [bp+4]
    mov    ax, 0          ; initializarea sumei cu 0
aduna:
    add    ax, [si]
    add    si, 2
    loop  aduna
    pop    si
    pop    ds
    pop    cx
    pop    bp
    ret
_suma endp
end

#include <stdio.h>
int suma (int vect[], int dim);
void main(void)
{
    int i, dim, vect[100];
    printf("Numarul de elemente din vector:");
    scanf("%d", &dim);
    printf("Elementele vectorului:\n");
    for (i=0; i<dim; i++)
    {
        printf("vect[%d]=", i+1);
        scanf("%d", &vect[i]);
    }
    printf("Suma elementelor vectorului este: %d\n", suma(vect, dim));
}
```