Reutilizarea codului

Macro

- Pseudo-operatie care permite includerea repetata de cod in program
- Asamblorul expandeaza numele macro-ului in codul corespunzator de fiecare data cand intalneste numele in cod
- Codul se repeta de cate ori este apelat => program de dimensiuni mari
- Utilizare: bucata de cod de dimensiuni mici (cateva linii) care se repeta in program

Creare macro-uri

- Create in cadrul programului
- Grupate in alt fisier -> biblioteca de macrouri
- Biblioteca de macro-uri contine cod neasamblat (fisier text)
- Trebuie inclus in programul sursa folosind pseudo-instructiunea “include nume_fisier”

```
.code
nume_m MACRO <parametrii>
      LOCAL <etichete locale>
      <corpul macro-ului>
ENDM

start:
      nume_m p1,p2
      nume_m p3,p4
END start
```

Biblioteca de macrouri

```
;bibm.asm
nume_m1 MACRO <parametrii>
      LOCAL <etichete locale>
      <corpul macro-ului>
ENDM
nume_m MACRO <parametrii>
      LOCAL <etichete locale>
      <corpul macro-ului>
ENDM
```

```
;program.asm
include bibm.asm

.data
p1 dd 10
...

.code
start:
      nume_m1 p1,p2
      nume_m2 p3,p4
END start
```

Expandare macro

```
;program.asm
.data
p1 DD 12
p2 DD 15
.code
suma MACRO n1, n2
  e1: mov eax, n1
      add eax, n2
      mov n1, eax
ENDM
start:
      suma p1, p2
      suma p2, p1
END start
```

```
;expandat.asm
.data
p1 DD 12
p2 DD 15
.code
start:
 e1:  mov eax, p1
      add eax, p2
      mov p1, eax
 e1:  mov eax, p2
      add eax, p1
      mov p2, eax
END start
```

Etichete in macro-uri

- La copierea repetata, etichetele se duplica => eroare la asamblare
- Solutie: etichete locale
  - “LOCAL e1,e2,…”
- Asamblorul va genera un **nume unic** pentru fiecare eticheta, de fiecare data cand macro-ul este expandat

```
;program.asm
.data
p1 DD 12
p2 DD 15
.code
suma MACRO p1, p2
LOCAL e1
e1:  mov eax, p1
       add eax, p2
       mov p1, eax
ENDM
start:
       suma p1, p2
       suma p2, p1
END start
```

Avantaje si dezavantaje

AVANTAJE

- pot fi create “instructiuni” noi
- poate duce la o programare mai eficienta
- timp de executie mai mic in comparatie cu apelurile de proceduri

DEZAVANTAJE

- pot ascunde operatii care afecteaza continutul registrilor
- ingreuneaza intelegerea si mentenanta programului

Proceduri

- procedura/functie/rutina
- portiune de cod reutilizata
- in acelasi fisier cu programul principal sau in biblioteci de procedure
- biblioteci: cod asamblat (format obiect) care este inclus in program prin link-editare
- prin modificarea fluxului de executie -> salt la adresa la care se afla codul
- mod de apel:
  - CALL nume_procedura
- revenire in programul apelant:
  - RET

```
.code
nume_p PROC
      <corpul procedurii>
      ret [<dimensiune param>]
nume_p ENDP

start:
      push p1
      push p2
      call nume_p
END start
```

Proceduri vs Macro-uri

MACRO

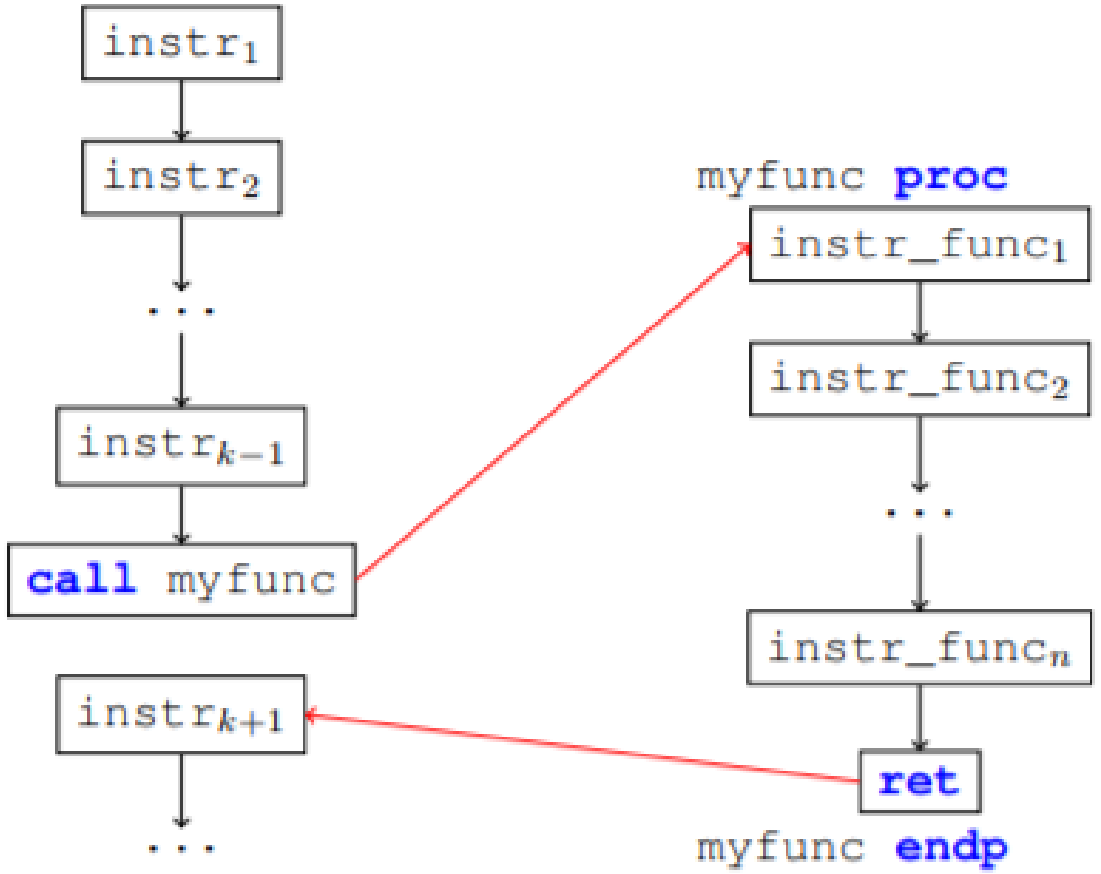
- la fiecare apel se copiaza secventa de instructiuni
- nu sunt necesare instructiuni de apel (CALL) si de revenire din rutina (RET)
- nu se foloseste stiva pentru transferul de parametri, apel si revenire
- transfer de parametri prin copierea numelui/valorii parametrului

PROCEDURA

- o singura copie pt. mai multe apeluri (codul este asamblat, apoi inclus in programla link-editare)
- se folosesc instructiuni de apel (CALL) si de revenire din rutina (RET)
- se utilizeaza stiva la apel si la revenire
- transfer de parametri prin registri sau stiva

Apel proceduri aflate in biblioteci

```
…
includelib bib.lib
extern nume_p:proc
public start
.data
p1 DD 12
p2 DD 15
.code
start:
        push p1
        push p2
        call nume_p
        ;eliberare stiva de params
END start
```



Conventii de apel

- Set de reguli prin care se specifica modul de transmitere al parametrilor si de intoarcere a rezultatului
- “Contract” intre autorul procedurii si utilizatorul acesteia
- Nu tine de sintaxa limbajului
- Nu este nevoie sa se specifice asamblorului conventia folosita

Conventia CDECL

- Parametrii se pun pe stiva in ordine inversa (de la dreapta la stanga)
- Rezultatul este in EAX (intregi), ST(0) (flotant)
- Apelantul curata parametrii de pe stiva

```
;int myfunc(int x, int y, int z, int t)
;res = myfunc(a, b, c, d)


…
push d
push c
push b
push a
call myfunc
add esp, 16
mov res, eax
…
```

Conventia STDCALL

- Parametrii se pun pe stiva in ordine inversa (de la dreapta la stanga)
- Rezultatul este in EAX (intregi), ST(0) (flotant)
- In functie se curata parametrii de pe stiva

```
;int myfunc(int x, int y, int z, int t)
;res = myfunc(a, b, c, d)


…
push d
push c
push b
push a
call myfunc
mov res, eax
…
```

Conventia FASTCALL

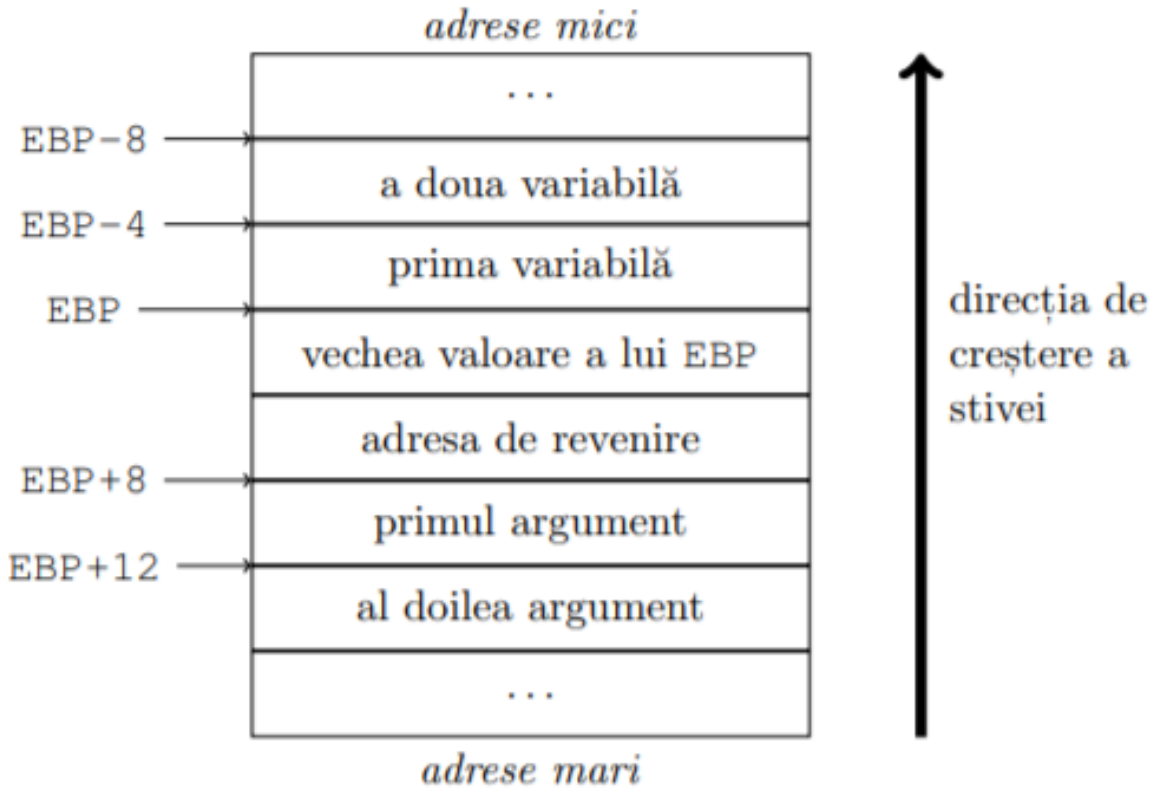
- Primii 2 parametri se pun in ECX si EDX
- Restul parametrilor se pun pe stiva in ordine inversa (de la dreapta la stanga)
- Rezultatul este in EAX (intregi), ST(0) (flotant)
- In functie se curata parametrii de pe stiva

```
;int myfunc(int x, int y, int z, int t)
;res = myfunc(a, b, c, d)


…
mov ecx, a
mov edx, b
push d
push c
call myfunc
mov res, eax
…
```

Scrierea unei proceduri

```
nume_p PROC
      push ebp
      mov ebp, esp
      sub esp, <dim_var_locale>
      <corpul procedurii>
      mov esp, ebp
      pop ebp
      ret [<dimensiune param>]
nume_p ENDP
```



Asamblare conditionata

Directiva IF-ENDIF

- Sintaxa:

```
if <expresie>
    <secventa de instructiuni>
else
    <secventa de instructiuni>
```

- Daca expresia este diferita de 0, se asambleaza prima secventa, altfel se asambleaza a doua

IFDEF, IFNDEF

- Sintaxa:

```
ifdef <nume>
    <secventa de instructiuni>
endif
```

- Daca a fost definita o constanta cu numele respective, atunci se va asambla secventa de instructiuni

Apelarea de functii din bibliotecile SO

Functia *scanf*

*int scanf (const char * format, ...);*

Citeste date cu format de la STDIN

Format:

- %d - intregi pe 32 de biti zecimal
- %x - intregi in hexazecimal
- %ld - intregi pe 64 de biti
- %lf - virgula mobila pe 64 de biti
- %c - character
- %s - sir de caractere

Restul argumentelor depind de format si sunt de tip referinta (se pune pe stiva adresa variabilei)

Conventia de apel CDECL

Functia returneaza numarul de argumente citite cu success, 0 sau un numar negativ daca sunt erori

```
…
includelib msvcrt.lib
extern scanf:proc
public start
.data
f DB "%d",0
n DD 0
.code
start:
        push offset n
        push offset f
        call scanf
        add esp,8
END start
```

Functia *printf*

*int printf (const char * format, ...);*

Scrie un sir de caractere cu format de la STDOUT

Format:

- %d - intregi pe 32 de biti zecimal
- %x - intregi in hexazecimal
- %ld - intregi pe 64 de biti
- %lf - virgula mobila pe 64 de biti
- %c - character
- %s - sir de caractere

Restul argumentelor depind de format si sunt de tip valoare

Conventia de apel CDECL

Functia returneaza numarul de caractere scrise cu success sau un numar negativ daca sunt erori

```
…
includelib msvcrt.lib
extern printf:proc
public start
.data
f DB "n=%d",0
n DD 12
.code
start:
        push n
        push offset f
        call printf
        add esp,8
END start
```

Functiile *fopen, fclose, fscanf, fprintf*

*FILE * fopen (const char * filename, const char * mode);*

Mod: "r", "w", "a", "b", …

Returneaza pointer catre fisier

*int fclose (FILE * stream);*

Returneaza 0 in caz de success sau EOF in caz de eroare

EOF = -1

*int fscanf (FILE * stream, const char * format, ...);*

*int fprintf (FILE * stream, const char * format, ...);*

Similare cu *scanf* si *printf*

Conventia CDECL

Exemplu scriere in fisier

```
…
includelib msvcrt.lib
extern fprintf:proc
extern fopen:proc
extern pclose:proc
public start
.data
f DB "n=%d",0
n DD 12
File DD 0
FileName DB "date.txt",0
md DB "w",0

.code
start:
```

```
        push offset md
        push offset FileName
        call fopen
        add esp,8
        mov File, eax
        push n
        push offset f
        push File
        call fprintf
        add esp,12
        push File
        call fclose
        add esp, 4
…
END start
```

Exercitiu

- Scieti un program care citeste character cu character dintr-un fisier, pana la intalnirea EOF

```
…
includelib msvcrt.lib
extern exit: proc
extern fopen: proc
extern fscanf: proc
extern fclose: proc
public start
.data
f DB "%c",0
cc DB 0
File DD 0
FileName DB "date.txt",0
md DB "r",0

.code
start:
        push offset md
        push offset FileName
        call fopen
        add esp,8
        mov File, eax
        et:     push offset cc
                push offset ff
                push File
                call fscanf
                add esp,12
                cmp eax, -1
                jne et


        push File
        call fclose
…
end start
```

Exercitii

- Scrieti o functie care primeste ca parametri 4 numere intregi pe 32 de biti si returneaza suma lor, folosind conventia de apel:
  - CDECL
  - STDCALL
  - FASTCALL
- Scrieti un MACRO care ordoneaza un sir de numere intregi pe 16 biti, primind ca parametri adesa si lungimea sirului.
- Scrieti o procedura care calculeaza minimul si media unui sir de intregi fara semn pe 32 de biti si primeste ca parametri adresa, lungimea sirului si variabilele in care se vor salva minimul si media