Seminar VI ASC

Programare multi-modul în limbajul de asamblare

- Programare multi-modul = se creează un fișier executabil compus din mai multe module obj
- Se vor scrie mai multe fișiere sursă (modul1.asm, modul2.asm, ...), se vor asambla (compila) separat folosind comanda (în linia de comanda)

  nasm.exe –fobj modul1.asm

  ……..

  nasm.exe –fobj modulN.asm

  și se vor link-edita într-un fișier executabil folosind comanda:

  alink.exe -oPE -subsys console -entry start modul1.obj modul2.obj …modulN.obj

  se obține fișierul executabil (pe platforma windows): modul1.exe
- Un modul va conține un program principal iar celelalte module vor descrie funcții/proceduri care vor fi apelate de modulul principal.
- Problemele de la laborator necesită scrierea doar a două module (un modul pentru programul principal și un modul pentru funcția apelată).
- Folosind cuvântul rezervat **global** se poate exporta un simbol (variabilă sau procedură) definită în modulul curent pentru a putea fi folosit în alte module; celelalte module vor importa simbolul extern folosind cuvântul rezervat **import**.
  Obs. constantele nu pot fi exportate deoarece pentru ele nu se rezervă memorie.

Transmiterea parametrilor unei funcții/proceduri definite în alt modul se poate face în următoarele moduri:

- folosind regiștrii – există un număr limitat de regiștrii care pot fi folosiți pentru a transmite parametrii;
- prin declararea parametrilor ca fiind globali (vizibilitate în toate modulele) – acest mod încalcă un principiu vechi din programare: *modularizare* (un program este mai bine întreținut dacă este compus din mai multe module independente). Dacă totul devine global în același spațiu (namespace) pot să apară inconsistențe între variabile (același simbol definit în mai multe locuri).
- folosind stiva – aceasta este soluția preferată (cea mai flexibilă).

Mai jos se dau exemple pentru cele trei mecanisme. Exemplele rezolvă expresia x=a+b.

Ex.1. Parametrii sunt transmiși modulelor folosind regiștrii.

Module main.asm	**Module function.asm**
bits 32 **global start** **extern exit** **import exit msvcrt.dll** **; we import the 'addition' function from the** **; function.asm module** **extern addition** **segment data use32 class=data public** **a db 2** **b db 3** **x db 0**	**bits 32** **; we export the 'addition' function in order to be** **; used in the main module** **global addition** **segment code use32 class=code public** **; the code segment contains only the addition** **; function** **addition:** **; the parameters are in: BL=a, BH=b** **; we will return the result in AL** **mov al, bl** **add al, bh**

```
segment code use32 class=code public
start:
    ; put the parameters in registers
    mov bl, [a]
    mov bh, [b]
    ; call the function
    call addition

    ; result is in AL
    mov [x], al

    ; call exit(0)
    push dword 0
    call [exit]
```

```
    ; return from function
    ; (it removes the Return Address from the stack
    ; and jumps to the Return Address)
    ret
```

Ex.2. Parametrii sunt transmiși funcției prin variabile globale.

Module main.asm

```
bits 32
global start
extern exit
import exit msvcrt.dll
; we import the 'addition' function from the
; function.asm module
extern addition

; we export variables a, b and x in order to be used
; in the other module
global a
global b
global x

segment data use32 class=data public
    a db 2
    b db 3
    x db 0

segment code use32 class=code public
start:
    ; there is no need to do anything with the
    ; parameters. They are already accessible to the
    ; other module (because they are global).

    ; call the function
    call addition

    ; the result is already placed in x by the addition
    ; function

    ; call exit(0)
    push dword 0
    call [exit]
```

Module function.asm

```
bits 32
; we export the 'addition' function in order to be
; used in the main module
global addition

; import the a, b, x variables from the other module
extern a, b, x

segment code use32 class=code public
; the code segment contains only the addition
; function
addition:
    ; the parameters are directly accessible in global
    ; variables a, b and x (which are global)
    mov al, [a]
    add al, [b]
    mov [x], al

    ; return from function
    ; (it removes the Return Address from the stack
    ; and jumps to the Return Address)
    ret
```

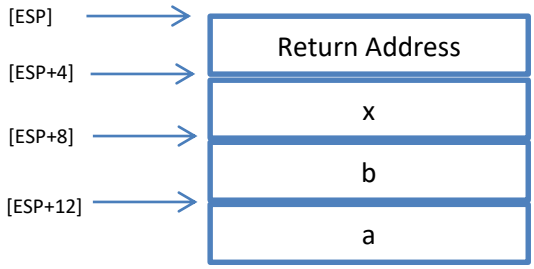
Ex.3. Parametrii sunt transmiși funcției din modulul secundar folosind stiva.

Module main.asm

```
bits 32
global start
extern exit
import exit msvcrt.dll
; we import the 'addition' function from the
; function.asm module
extern addition

segment data use32 class=data public
    a  db  2
    b  db  3
    x  db  0

segment code use32 class=code public
start:
    ; put the parameters a, b and x on the stack
    ; we can not put bytes on the stack, we will put
    ; dwords
    mov eax, 0
    mov al, [a]
    push eax
    mov al, [b]
    push eax
    mov al, [x]
    push eax

    ; call the function
    call addition
    ; the result is in the dword from the top of the stack
    pop eax
    mov [x], al          ; x := a + b
    ; we still have to remove 2 dwords from the stack
    ; (the dwords corresponding to 'a' and 'b')
    add esp, 4*2
    ; instead of the above instruction we could have
    ; used two 'pop eax' instructions

    ; call exit(0)
    push dword 0
    call [exit]
```

Module function.asm

```
bits 32
; we export the 'addition' function in order to be
; used in the main module
global addition

segment code use32 class=code public
; the code segment contains only the addition
; function
addition:
    ; the parameters are on the stack
    ; the stack looks like this:
```




```
    ; remember that a stack element is 4 bytes
    ; (dword) and the stack grows toward smaller
    ; addresses (meaning that the dword from the top
    ; of the stack is placed at the smallest memory
    ; address).
    ; the Return Address was placed on the stack by
    ; the 'call addition' instruction in the main module.

    mov eax, dword [esp+12]
    mov bl, al       ; bl = a
    mov eax, dword [esp+8]
    add bl,  al      ; bl = a + b
    mov al, bl
    mov dword [esp+4], eax     ; place a+b on the stack
                               ; for the main module

    ; return from function
    ; ('ret' removes the Return Address from the
    ; top of the stack and jumps to the Return Address)
    ret
```

Ex. 4. Scrieți un program care concatenează două șiruri prin apelul unei funcții scrise într-un modul secundar și apoi va afișa rezultatul pe ecran.

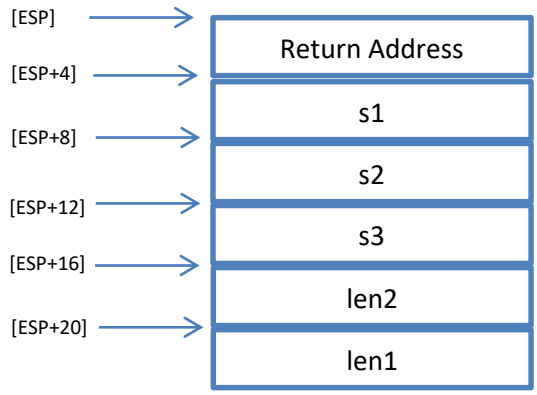
Main.asm module:

```
bits 32
global start
```

```
extern exit, printf
extern concatenare          ; import 'concatenare' from the other module
import printf msvcrt.dll
import exit msvcrt.dll


segment data use32 class=data public
    s1 db 'abcd'
    len1 equ $-s1
    s2 db '1234'
    len2 equ $-s2
    s3 times len1+len2+1 db 0

segment code use32 class=code public
start:
    ; we place all the parameters on the stack
    push dword len1
    push dword len2
    push dword s3
    push dword s2
    push dword s1
    call concatenare
    add esp, 4*5

    push dword s3
    call [printf]

    push dword 0
    call [exit]
```

Function.asm module:

```
bits 32
global concatenare              ; export concatenare

segment code use32 class=code public
   concatenare:
        ; the stack looks like this:
```




```
        ; first copy s1 in s3
        mov esi, [esp+4]        ; ESI = the offset of the source string (s1)
        mov edi, [esp+12]       ; EDI = the offset of the destination string(s3)
        mov ecx, [esp+20]       ; ECX = len1
        cld
        rep movsb               ; rep repeats movsb ECX times

        ; then, copy s2 at the end of s3
        mov esi, [esp+8]        ; ESI = the offset of the source string (s2)
                                ; EDI already contains the offset of the destination string
        mov ecx, [esp+16]       ; ECX = len2

        rep movsb               ; rep repeats movsb ECX times

        ret
```