



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

31. Programe compuse din mai multe module.

Programe compuse din mai multe module

Este posibil însă să dezvoltăm un program care constă dintr-un modul principal, legat (editate legăturile) împreună cu unul sau mai multe programe asamblate separat. Motivele organizării programelor în subprograme sunt următoarele:

- pentru a face legătura între diferite limbaje, pentru a combina ușurința programării în LNI cu eficiența de prelucrare a LA;
- pentru a facilita dezvoltarea de proiecte mari, în care diferite echipe elaborează separat modulele lor;
- pentru a suprapune părți ale unui program pe durata execuției sale, datorită dimensiunii mari a programului.

Programele de dimensiuni mai mari se împart în mai multe module. Vom denumi modul, un program compus din unul sau mai multe segmente, cuprins într-un fișier. Fiecare modul este asamblat separat, generând propriul său modul obiect. Editorul de legături realizează apoi legarea modulelor obiect într-un singur modul executabil. Modulele trebuie să comunice între ele.

În cazul simbolurilor, definite în cadrul unui modul, ce trebuie referite din alte module, acestea trebuie declarate 'PUBLIC', în modulul unde sunt definite, și declarate 'EXTRN' :

BYTE, WORD, DWORD, FWORD, QWORD,

TBYTE - ptr. variabile

ABS - pentru constante,

NEAR, FAR, PROC - pentru etichete sau proceduri; pentru ultimul tip

(PROC) asamblorul determină tipul procedurii.

Deci la definirea simbolurilor se declară 'PUBLIC', iar la referire, din alt modul, se declară 'EXTRN'.

Să considerăm un program principal, scris într-un modul, care citește (*cit_sir*) un număr de la tastatură, în zecimal, și-l depune în memorie (sau stivă) sub forma unui șir de caractere ASCII (cel mult 5 cifre sau se va termina cu un caracter diferit de cifră), eventual și lungimea șirului (sau este depus în mem. sub forma unui șir ASCIIZ). O altă procedură (*conv_binar*) va face conversia acestui șir de cifre la un număr binar de 16 biți, și-l returnează în AX, sau la o locație din segmentul de date (*rezultat*). După conversie (dacă nu apare o eroare), prog. principal, va tipări valoarea în hexa apelând o tot o proc. (*tip_numar*). Procedurile vor fi incluse într-un modul separat (*PROC.ASM*) de modulul programului principal (*MAIN.ASM*).

MAIN.ASM

.model small

.stack 100h

.data

public numar_ASCII, rezultat, lungime, valoare_hexa

public tab_conv

tab_conv db '0123456789abcdef'

lungime equ 5

numar_ASCII db lungime dup (?) ; rez. pt. coduri ASCII

valoare_hexa db 4 dup (?), 0dh, 0ah, '\$' ; valoarea hexa

rezultat dw ?

```

        mes_err      db      0dh, 0ah,
                        'Eroare de conversie: numar > 65535'
CRLF db      0dh, 0ah, '$'
.code
extrn  conv_binar: proc, cit_sir: proc, tip_numar: proc
start:
    mov  ax, @data
    mov  ds, ax          ; inițializare DS
    call cit_sir         ; se citește numărul
    call conv_binar     ; se realizează conversia la binar
    jc  tip_mes_err    ; mesaj de err, pt. eroare la conversie
    lea dx, CRLF ; avans la linie noua
    mov ah, 9
    int 21h
    call tip_numar     ; se tipărește numărul în hexazecimal
    jmp  gata_prg      ; terminare program
tip_mes_err:
    lea dx, mes_err    ; tipărire mesaj de eroare, la conversie
    mov ah, 9
    int 21h
gata_prg:
    mov ax, 4c00h      ; revenire DOS
    int 21h
end start

```

PROC.ASM

```

.model small
.data
    extrn numar_ASCII: byte, rezultat : word, lungime : abs
    extrn valoare_hexa : byte, tab_conv : byte
.code
    public cit_sir, conv_binar, tip_numar
    ori10 macro          ; înmulțește cu 10 registrul AX
        push  bx          ; se salvează reg. de lucru
        shl   ax, 1       ; * 2
        mov   bx, ax
        shl   ax, 1       ; * 4
        shl   ax, 1       ; * 8
        add   ax, bx
        pop   bx          ; reface registru salvat
    endm
cit_sir proc
    push ax              ; salvare registre de lucru
    push bx
    push cx
    lea  bx, numar_ASCII ; adresa unde se depun caracterele

```

```

    mov  cx, lungime  ; dimensiunea maximă a șirului
cit:
    mov  ah, 1        ; citirea unui caracter
    int  21h
    mov  [bx], al     ; se depune caracterul la numar_ASCII
    inc  bx          ; actualizarea adresei șirului
    cmp  al, '0'      ; test dacă caracterele sunt cifre
    jb  gata         ; dacă nu sunt cifre s-a terminat citirea
    cmp  al, '9'
    ja  gata
    loop cit          ; se reia citirea, pt. cel mult 5 cifre
gata:
    pop  cx          ; refacerea registrelor salvate      pop  bx
    pop  ax
    ret
cit_sir endp
conv_binar  proc
    push si          ; salvare registre de lucru
    push bp
    push ax
    push cx
    lea  si, numar_ASCII ; adresa cifrelor ASCII
    mov  cx, lungime  ; număr maxim de cifre: 5
    cld          ; direcția de parcurgere a șirului
    mov  bp, 0      ; valoarea convertită/ rezultatul binar
reia:
    mov  ah, 0      ; se pune 0 pentru a extinde valoarea în AX
    lodsb          ; se citește o cifră
    cmp  al, '0'    ; test de sfârșit număr
    jnb  cont_cmp  ; dacă nu, se continua comp. / conv.
    cld
    jmp  gata_conv  ; dacă da, se termină conversia (CF=0)
cont_cmp:
    cmp  al, '9'    ; test sfârșit număr
    ja  gata_conv
    sub  al, '0'     ; conversie de la cod ASCII la valoare
    xchg bp, ax     ; (AX) = valoarea
    ori10          ; se înmulțește valoarea cu baza
    jc  gata_conv  ; dacă apare depășire se termină
    add  bp, ax     ; se adună ultima cifră citită
    jc  gata_conv  ; testarea condiției de eroare
    loop reia      ; valoarea a rămas în BP (și în AX)
gata_conv:
    ; dacă a apărut err CF = 1, altfel CF = 0.
    mov  rezultat, bp ; se depune rezultatul
    pop  cx          ; refacerea registrelor salvate
    pop  ax

```

```

    pop    bp
    pop    si
    ret
conv_binar    endp
tip_numar    proc
    push si        ; salvare registre de lucru
    push ax
    push cx
    push bx
    push dx
    lea si, valoare_hexa ; adresa unde se depun cifrele hexa
    mov dx, rezultat    ; valoarea de tipărit în hexa
    mov  cx, 4          ; contor număr de cifre de tipărit
    lea bx, tab_conv    ; adresa tabelii de conversie
iar:
    push cx          ; salvare contor
    mov  cl, 4       ; contor număr de rotiri
    rol  dx, cl      ; se rotesc primii 4 biți pe ultimii 4
    mov  al, dl
    and  al, 0fh     ; se rețin doar ultimii patru biți
    xlat tab_conv    ; conversie binar → hexa (ASCII)
    mov  [si], al    ; depunere la valoare_hexa
    inc  si          ; actualizare index pentru depunere
    pop  cx          ; contor număr de cifre hexa de tipărit
    loop iar         ; se reia conversia pentru cele 4 cifre
    lea  dx, valoare_hexa ; adresa cifrelor hexa, pentru
    mov  ah, 9       ; tipărire. se utilizează funcția
    int  21h        ; 9, din DOS
    pop  dx          ; refacerea registrelor salvate
    pop  bx
    pop  cx
    pop  ax
    pop  si
    ret
tip_numar    endp
end          ; sfârșitul modulului 'PROC.ASM'

```