Ministerul Educaţiei, Culturii și Cercetării Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare Informatica și Microelectronică

RAPORT

Laboratorul nr.4

la disciplina Arhitectura Calculatoarelor

Tema: Programe cu ramificări şi bucle. Subprograme și macroinstrucțiuni.

A realizat: st.gr Popescu Victoria TI-173

A verificat: Colesnic V.

Chișinău 2019

Scopul lucrării:

Lucrarea urmărește deprinderea studenților cu proiectarea și implementarea programelor cu subprograme și macroinstrucțiuni, cu ramificații și bucle in limbaj de asamblare. Se prezintă câteva exemple tipice de astfel de programe, incluzând câteva metode elementare de sortare şi căutare. De asemenea se prezintă câteva implementări de operații aritmetice care necesită utilizarea unor structuri de control de tip ramificare şi buclare. Se vor utiliza instrucțiunile de comparare, salt şi buclare. Se prezintă de asemenea şi câteva exemple de utilizare a instrucțiunilor logice.

Indicații teoretice:

Subprograme (proceduri) și macroinstrucțiuni

În general definirea unui subprogram se face cu directiva *PROC* în maniera următoare:

nume PROC {*NEAR* | FAR}

corp

RET {*constanta*}

nume ENDP

Dacă atributul *NEAR* şi *FAR* lipsesc, în cazul utilizării definiţiilor complete se consideră implicit *NEAR*, iar în cazul definiţiilor simplificate se folosesc valorile implicite în funcţie de modelul de memorie utilizat.

Tehnicile de transfer a parametrilor combină diversele modalităţi de alegere a tipurilor de locaţii fizice pentru păstrarea parametrilor transmiși: registre, locații de memorie fixate, codul apelant, stiva procesorului, tehnica blocurilor (tabelelor) de parametri, cu ceea ce se transmite efectiv referitor la un anumit parametru: adresa sau valoarea acestuia.

***Instrucţiunea CALL* (apel de procedură)**

Poate apărea sub una din formele:

CALL nume_proc

CALL NEAR nume_proc

CALL FAR nume_proc

Tipul apelului poate fi dedus din tipul procedurii (primul caz) sau specificat explicit prin NEAR si FAR. Tipul apelului trebuie să coincidă cu tipul procedurii şi cu tipul instrucțiunii RETURN din interiorul procedurii.

***Instrucţiunea RET (RETURN*)**

Forma generală:

RET [n]

unde n este o constantă întreagă opțională.

Dacă instrucţiunea RET este de tip NEAR semnificaţia sa este:

(IP) ← SS: ((SP) + 1:(SP))

(SP) ← (SP) + 2

[(SP) ← (SP) + n]

adică se reface (IP) prin copierea conţinutului vârfului stivei şi incrementarea cu 2 a lui (SP). Dacă în instrucţiunea RET apare şi constanta **n** atunci aceasta constanta se aduna la (SP), adică se descărcă stiva.

Exemplu:

```
.data
a    DWORD  55555h
b    DWORD  77777h
s    DWORD  ?
.code
...
; încarcă primul număr in DX:AX
    mov  ax,WORD PTR a[0]
    mov  dx,WORD PTR a[2]
        ; încarcă al doilea number in DI:SI
    mov  si,WORD PTR b[0]
    mov  di,WORD PTR b[2]
; încarcă adresa rezultatului in EBX
    mov  ebx,OFFSET s
; apelează procedura
    call pro_ad
...
; codul procedurii
pro_ad PROC  NEAR
    add  ax,si
    adc  dx,di
```

```
    mov  [ebx],ax
    mov  [ebx+2],dx
    ret
pro_ad ENDP
...
```

O macroinstrucțiune reprezintă o secvență de cod sursă căreia i se atribuie un nume simbolic, conținutul acestei secvențe putând fi repetat ori de câte ori este apelat în cadrul unui program prin simpla referire la numele simbolic respectiv. Utilizarea unei macroinstrucțiuni necesită parcurgerea a doi pași:

1.Definirea macroinstrucțiunii, care se marchează printr-o macrodefiniție. Aceasta cuprinde o secvență de cod, intre directivele MACRO si ENDM. Sintaxa este:

```
nume  MACRO {parametrii}
cod
ENDM
```

unde:

- **nume** reprezintă numele simbolic dat macroinstrucțiunii ;
- **parametrii** reprezintă parametrii formali opționali ai macroinstrucțiunii, separați prin virgulă, blancuri sau TAB-uri. La apelul macroinstrucțiunii, acești parametri formali sunt înlocuiți textual cu parametrii actuali.

2.Apelul macroinstrucțiunii, care se realizează printr-un macroapel, cu sintaxa:

```
nume {argumente}
```

unde:

- **nume** reprezintă numele simbolic al macroinstrucțiunii apelate;
- **argumente** reprezintă lista parametrilor actuali, separați prin virgula, blancuri sau TAB-uri.

Apelul macroinstrucțiunii are ca efect includerea textuală a codului din definiția macroinstrucțiunii în corpul programului.

Mersul lucrării:

Exemplul 7. {Implementarea unei structuri de selectie} Acest exemplu (pe 16 biti) prezinta o metoda de implementare a unei structuri de selectie utilizind salturi indirecte. Se citeste un caracter de la consola si functie de valoarea acestuia se executa urmatoarele actiuni: daca valoarea caracterului este 'a' se afiseaza sirul 'Alfa', daca valoarea caracterului este 'g' se afiseaza sirul 'Gama', daca valoarea caracterului este 'd' se afiseaza sirul 'Delta' altfel se va afisa sirul '***':

```
        .DATA
cars    DB      'a','g','d'
sir_a   DB      'Alfa',13,10,'$'
sir_g   DB      'Gama',13,10,'$'
sir_d   DB      'Delta',13,10,'$'
sir     DB      '***',13,10,'$'
act_tbl DW      act_a,act_g,act_d
        .CODE

        mov     ah,01h
        int     21h
        xor     bx,bx
urm:    cmp     bx,02h
        ja      act
        cmp     al,cars[bx]
        je      gasit
        inc     bx
        jmp     SHORT urm
gasit:  shl     bx,1
        jmp     act_tbl[bx]
act_a:  mov     dx,OFFSET sir_a
        mov     ah,09h
        int     21h
        jmp     SHORT exit
act_g:  mov     dx,OFFSET sir_g
        mov     ah,09h
        int     21h
        jmp     SHORT exit
act_d:  mov     dx,OFFSET sir_d
        mov     ah,09h
        int     21h
        jmp     SHORT exit
act:    mov     dx,OFFSET sir
        mov     ah,09h
        int     21h
exit:   mov ah,01
                int 21h
                    exit
```

Utilizând macro-uri programul poate fi rescris în următorul mod:

```
        .DATA
cars    byte      'a','g','d'
sir_a   byte      'Alfa',13,10,'$'
sir_g   byte      'Gama',13,10,'$'
sir_d   byte      'Delta',13,10,'$'
sir     byte      '***',13,10,'$'
act_tbl DW        act_a,act_g,act_d


    afis macro x,y
            mov     dx,OFFSET x
      mov     ah,09h
      int     21h
      jmp     SHORT y
    endm
      .CODE

      mov     ah,01h
      int     21h
      xor     bx,bx
urm:  cmp     bx,02h
      ja      act
      cmp     al,cars[bx]
      je      gasit
      inc     bx
      jmp     SHORT urm
gasit: shl    bx,1
      jmp     act_tbl[bx]
act_a: afis sir_a,exit
act_g: afis sir_g,exit
act_d: afis sir_d,exit
act:  mov     dx,OFFSET sir
      mov     ah,09h
      int     21h

    exit:           mov ah,01
                        int 21h
            exit
```

Exemplul 12. {Umplere ecran cu un sir de caractere citit de la tastatura} Urmatorul program (pe 16 biti) citeste un sir de maxim 16 caractere de la tastatura si afiseaza 25 de linii a cite 80 de caractere pe linie, continind sirul citit:

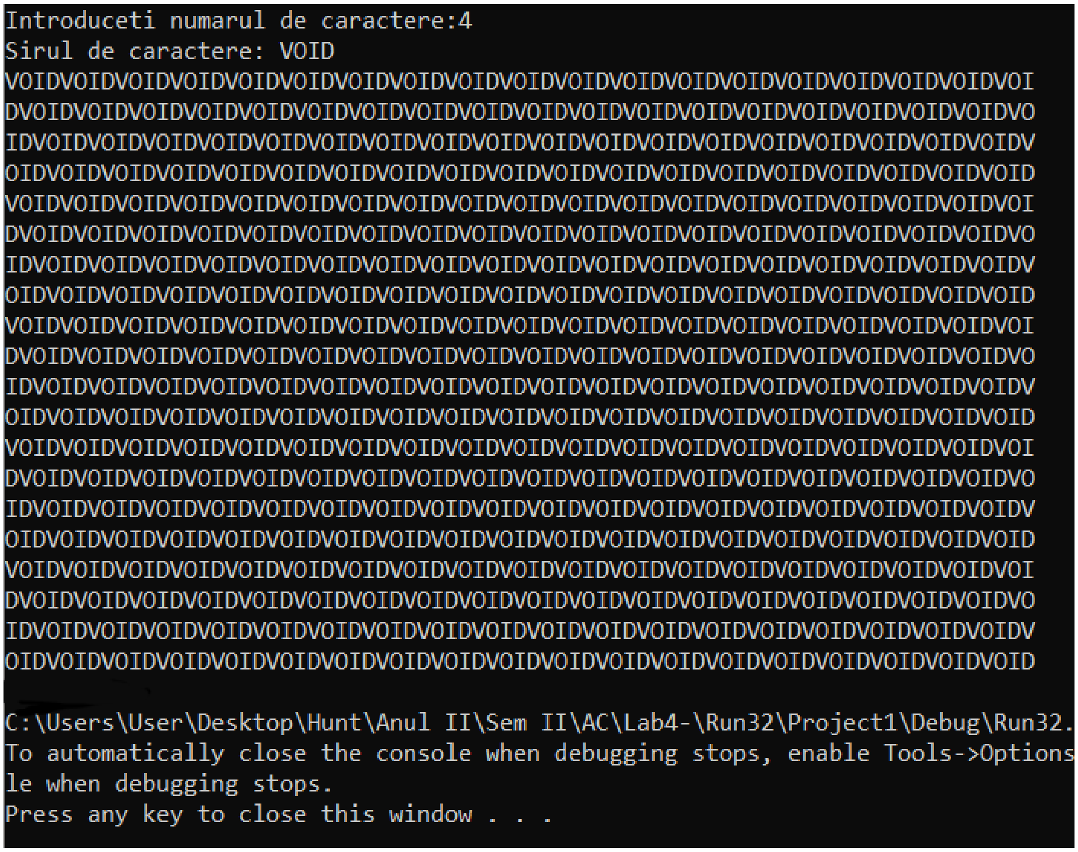
```
            .DATA
buffer        LABEL     BYTE
max_length    DB        ?    ; Lungimea maxima
chars_entered DB        ?    ; Numarul caracterelor citite
string        DB        17 DUP(?) ;  Tampon pentru 17 caractere,
                        ; se considera  si un CR
strings_per_line DW     0    ; Cite siruri incap pe o linie
crlf          DB        0Dh,0Ah,'$'
            .CODE

            mov       dx,OFFSET buffer ; Citire sir
            mov       buffer,17
            mov       ah,0Ah
            int       21h
            xor       bx,bx
            mov       bl,chars_entered ; In bl lungimea
                              ; sirului
            mov       buffer[bx+2],'$'
            mov       al,80
            cbw
            div       chars_entered ; De cite ori incape sirul
                              ; pe linie
            xor       ah,ah
            mov       strings_per_line,ax
            mov       cx,25
display_screen: push      cx
            mov       cx,strings_per_line
display_line:  mov        dx,OFFSET string ; Afisare sir
            mov       ah,09h
            int       21h
            loop      display_line
            mov       dx,OFFSET crlf
            mov       ah,09h
            int       21h
            pop       cx
            loop      display_screen
                      mov ah,01
                      int 21h
            exit
```

Sarcina individuală:

8. Scrieți o secvență de program care citește un șir de maxim 10 caractere de la tastatură și afișează 20 de linii a câte 75 de caractere pe linie, conținând șirul citit.

```
INCLUDE Irvine32.inc
at MACRO nr
mov ax,nr
mov fin, al
mov index,0
lea esi, msg
show:
mov ax,nr
cmp ax,0
je afisrestu;
mov ax,[esi]
call WriteChar
inc esi
dec nr
jmp show;

afisrestu:
mov ax,index
cmp ax,0
je rindnou;
mov al,fin
cmp ax,len
je iesire1;

a:
mov al,[esi]
call WriteChar
inc esi
inc fin
jmp afisrestu;

rindnou:
call Crlf
dec linii
inc index
jmp afisrestu;
 iesire1:
ENDM


.data
mes1 byte "Introduceti numarul de caractere:",0
        mes2 byte "Sirul de caractere: ",0
                max = 10
    sc dw 75
        linii dw 20
    msg     DB    10 DUP(?)
        len Dw ?
        tru dw ?
```

```
        index dw 1
        fin db 0
        nl dw 0
.code
main proc
mov edx,OFFSET mes1
call WriteString                ; afisarea mes1
call ReadDec                    ; introducerea de la tastatura
 mov len,ax                     ;
 mov edx,OFFSET mes2
call WriteString                ; afisarea mes2

mov edx,OFFSET msg
mov ecx,max
call ReadString

afis:
mov ax,sc
cmp len,ax
jg trunchi;
mov edx,OFFSET msg
call WriteString
mov ax,sc
sub ax,len
mov sc,ax
mov ax,linii
cmp ax,0
je iesire;
jmp afis;

trunchi:
mov ax,len
sub ax,sc
mov nl,ax
at sc
mov ax,75d
sub ax,nl
mov sc,ax
jmp afis;

iesire:
exit


main ENDP
END main
```



```
Introduceti numarul de caractere:4
Sirul de caractere: VOID
VOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOI
DVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVO
IDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDV
OIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOID
VOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOI
DVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVO
IDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDV
OIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOID
VOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOI
DVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVO
IDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDV
OIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOID
VOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOI
DVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVO
IDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDV
OIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOID
VOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOI
DVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVO
IDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDV
OIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOIDVOID

C:\Users\User\Desktop\Hunt\Anul II\Sem II\AC\Lab4-\Run32\Project1\Debug\Run32.
To automatically close the console when debugging stops, enable Tools->Options
le when debugging stops.
Press any key to close this window . . .
```

Figura1 – Afișarea cuvântului VOID în 20 rânduri a câte 75 caractere

Concluzie:

Scopul propus de această lucrare de laborator este ințelegerea și implimentarea subprogramelor și macroinstrucțiunilor în limbajul Assembler. Pe lângă altele este necesară și înțelegerea logicii în acest sistem.

In cadrul sarcinei principale a servit rularea programelor scrise pe limbajul Assembler și transformarea numerelor în cod BCD. Pe lângă aceasta am făcut o familiarizare cu șirurile de caractere.

In timpul rulării programelor au aparut unele probleme cu înțelegerea BCD și în genere a limbajului însă spre sfârșit au fost studiate și deja înțelese.

Datorita acestei lucrari de laboratori am înțeles cum se impachetează și despachetează numerele în cod BCD.