Programming in Assembler

Laboratory manual

Exercise 4

Program segments, INVOKE and PROC directives

During the Exercise No.4 students are to debug the sorting program using the CodeView Debugger. Next step is to check the functionality of sorting procedures using different data tables. On the last step other program should be modified to improve its' speed of working.
Program is attached to the documentation in `lab4.asm` file.

During the laboratory students are to:

1. Create the project to the `lab4.asm` file with options for debugging and generating listing file.

2. Assemble the project to the `*.exe` file and run the program in debugger using step-by-step mode.

3. Check correctness of sorting procedures.

4. View the source in mixed mode to notice that some of assembled instructions have the 66h or 67h prefixes.

5. Notice what instructions have been generated in place of the INVOKE and PROC directives.

6. Modify the program above to improve its' performance by eliminating the 66h and 67h prefixes.

7. Debug the modified program with CodeView and run the program.

The report should consist of:

- Title page.
- Explanation of 66h and 67h pefixes function.
- Explanation of .586 directive function.
- Description of code generated by INVOKE and PROC directives.
- Modified program listing file.
- Conclusions.

Source code:

```
;*********************************************************************
;*                                                                   *
;*               LAB4.ASM - Assembler Laboratory ZMiTAC              *
;*                                                                   *
;*                        Sorting program                            *
;*                                                                   *
;*********************************************************************

.model small
.586

.data
table          dword 10 dup (?)
          org offset table
               dword  9, 1, 0, 3, 7, 2, 4, 5, 8, 6
table_length   equ length table
t_table        equ type table

.stack

.code
;-------------------------------------
; simple insert sorting
;-------------------------------------

sort_smp_in proc near uses eax ecx esi edi

          mov  ecx, table_length             ; number of table elements
          dec  ecx
          xor  esi, esi                      ; esi<-0
sort:     inc  esi
          push esi
insert:   mov  eax, table[esi*t_table]
          cmp  eax, table[esi*t_table-t_table]
          jge  go_on
zamiana:  mov  edi, table[esi*t_table-t_table]
          mov  table[esi*t_table-t_table], eax
          mov  table[esi*t_table], edi
          dec  esi
          cmp  esi, 0
          jg   insert
go_on:    pop  esi
          loop sort
          ret
sort_smp_in endp

;-------------------------------------
; simple take sorting
;-------------------------------------
```

```
sort_smp_take proc near uses eax ecx edx esi

         mov  ecx, table_length
         dec  ecx
         xor  esi, esi
sort:    mov  edx, esi
         inc  esi
         push esi
         mov  eax, table[esi*t_table-t_table]
find_min:
         cmp  table[esi*t_table], eax
         jge  go_on
new_min:
         mov  eax, table[esi*t_table]
         mov  edx, esi
go_on:   inc  esi
         cmp  esi, table_length
         jl   find_min
exchange:pop  esi
         push esi
         dec  esi
         mov  edi, table[esi*t_table]
         mov  table[esi*t_table], eax
         mov  esi, edx
         mov  table[esi*t_table], edi
         pop  esi
         loop sort
         ret
sort_smp_take endp

fill_table proc near

  mov table, 3   ; fill the table with some values
  mov table+4, 5
  mov table+8, 0
  mov table+12, 7
  mov table+16, 1
  mov table+20, 8
  mov table+24, 2
  mov table+28, 4
  mov table+32, 9
  mov table+36, 6
        ret
fill_table endp

;***************************************
; Main
;***************************************
.startup
```

```
invoke sort_smp_in
invoke fill_table
invoke sort_smp_take

.exit
end
```