1) Exercise specification

Write an assembly language program, which prepares all the needed processor's structures for switching to a protected mode, than enables the processor's protected mode in a supervisor privilege level. After switching to a protected mode the program should display a simple text message to the console. At the end of operation, the program should switch the processor back to a real mode and exit.

2) The Protected Mode

Protected mode is an operational mode of IA-32 compatible processors. It was first introduced in the Intel's 80286 processor, and later extended with the release of other IA family processors. Protected mode allows a (operating) system software to utilize features such as safe multitasking, virtual memory handling or restricting the application's software to have a direct access to hardware devices.

3) Preparing a program which runs in a Protected Mode

When an IA-32 computer system (PC) starts after switching the power on, the CPU, due to backward compatibility reasons, starts to operate in a *real mode*. In this processor's mode of operation any application has full access to any hardware device installed, all the available memory (up to 1 MB) the ports interrupts, etc. It's of course a very uncomfortable situation for any operating system to let it's applications have an unrestricted access to all computer system's hardware. So when the operating system is being loaded it installs it's low level procedures (such as a kernel and all kinds of drivers), prepares a processor to run in the protected mode and as soon as possible switches the processor to the protected mode. Since that moment the operating system fully controls the PC, and any application (which can be loaded only by an operating system) must obey the rules that the system set up.

In the first exercise, the program will act as a toy operating system, which starts, goes to run in the protected mode with all privileges displays a string to the console, goes back to the real mode and finally returns a control to the host DOS operating system. All program code should be written in the assembly language and prepared using standard tools ie. the *tasm* compiler and *tlink* linker with /3 option for 32-bit code linking. The program should use *large* memory model with 32-bit instructions enabled (.386p assembler directive). It also must be divided into the following segments:

- 16-bit code, data and stack segments (for running in the real mode)
- 32-bit code, data and stack segments (for running in the protected mode)

The exemplary directives which enable creating 16 or 32 bit-segments are shown below:

```
thestack    segment stack use16     ;create 256 byte, 16-bit (for operating in real mode)
            db 100h dup(0)          ;stack segment
thestack    ends

thecode     segment use32           ;create 32-bit (for operating in protected mode)
            . . .                   ;code segment
thecode     ends
```

The real mode data segment should contain one *global descriptor table (GDT)*. GDT should be filled with, subsequently: *a zero descriptor*, the descriptors for all program segments (16 and 32-bit) and a single descriptor which describes 4000 (decimally) bytes long segment for enabling the program to write data (a string) to a video memory, which is, as always in the text mode, located at a real address of 0B800:0000 (in segment:offset notation). Descriptors should be initially filled with proper attributes, excluding the given segments' base addresses and lengths. Base addresses and the lengths of program segments described in descriptors should be filled later, just before entering the protected mode. To obtain a segment length in bytes one can use the following exemplary construction:

```
thesegment    segment                          ;standard 16-bit code and/or data segment
              . . .                            ;instructions and/or data
ending_label:                                  ;label at the end of the defined segment
thesegment    ends

thedata       segment                          ;a word variable containing
              theseglen dw offset ending_label ;length in bytes of the segment
              . . .                            ;named 'thesegment'
thedata       ends
```

It's also important that the real mode data segment should be 64 KB long and it's descriptor should have properly defined attributes (see lecture's section *PM memory management* about attributes of real segments).

4) Entering and leaving Protected Mode

To enter the protected mode one has to accomplish the following steps:

- properly fill the descriptors in the GDT table (see the section above)
- clear the processor's interrupt flag not to allow interrupts to occur when working in protected mode (it has to be done because, firstly: we do not have a properly filled *Interrupt Descriptor Table*, and secondly: we do not have properly defined interrupt procedures – standard DOS real mode procedures cannot be executed in protected mode)
- load processor's *Global Descriptor Table Register* (*GDTR*) with GDT address and limit
- switch on the processor to a protected mode (see the lecture)
- make a *far jump* to a first instruction in the program's 32-bit code segment, which is to be executed in the protected mode (see notes below)

The mentioned *far jump* to the first instruction to be executed in protected mode has to be made to reload the CS register so that it is loaded with a selector not with a segment number as in the real mode. Unfortunately *tasm* compiler (as a 16-bit compiler) does not usually recognize the need for using the far jumps so it often, without the programmer's control, forces jump instructions output during compilation to be so called *near jumps* which do not reload CS register when they are executed. To force *tasm* to compile a jump instruction as a far jump a programmer has to do it manually by coding directly the jump instruction in a pure machine code. This can done as follows:

```
thecode16   segment use16          ;a 16-bit code segment definition
            . . .                  ;some instructions
            db 0eah                ;far jump instruction code for 16-bit code
            dw offset label        ;offset to where to jump
            dw SELECTOR            ;segment selector to where to jump
thecode16   ends

thecode32   segment use32          ;a 32-bit code segment definition
            . . .                  ;some instructions
label:      xor eax, eax           ;instruction to which the far jump is performed
            . . .                  ;some other instructions
thecode32   ends
```

After passing program control to 32-bit code segment's instructions first thing to do is to load segment registers with proper selectors and then display the text message. After displaying the string program must ones again pass control to 16-bit code segment using the same pure machine code far jump technique. It's however important to keep track of offset argument sizes in the far jump instruction code while jumping from 16 or 32-bit segments (the sizes are not the same, see Intel manual for far jump instruction explanation). Being still in protected mode but now executing 16-bit code it's important to properly reload all segment registers (see the note above just before the current paragraph **3))** before switching back to a real mode. Having reloaded segment registers application has to switch back the processor to operate in the real mode. Just after switching ones again the program has to make a far jump (with the same method as explained above) to the next instruction after that jump. It has to be done for the same reason as before – now to reload the CS register so that it is loaded not with with a current segment's selector but with a current segment's segment number (paragraph), because the processor is already running in the real mode. To sum up for the program to leave the protected mode it has to accomplish the following steps:

- make a far jump to the proper instruction in a 16-bit code segment
- reload all segment registers (except CS) with proper selectors for the real mode
- switch the processor to a real mode
- immediately make a far jump to the next instruction (after that jump) to reload CS register
- exit to DOS

4) Summary

Application should successfully enter and leave the IA-32 processor protected mode displaying a simple string on the (console) screen while the processor is operating in protected mode.