Министерство образования Республики Беларусь

Учреждение образования Белорусский государственный университет информатики и радиоэлектроники

Кафедра ЭВМ

Отчёт по лабораторной работе №6

“Защищенный и реальный режим процессора. Переход из одного режима в другой и обработка прерываний”

Проверил: Выполнил:

к.т.н., доцент студент гр.150502

Одинец Дмитрий Николаевич Альхимович Н. Г.

Минск 2023

**Задача**

Написать программу, которая выполняет следующие действия:

1. Переход из реального режима в защищенный;

2. Перехватывание аппаратных прерываний от клавиатуры и заданного аппаратного прерывания (от таймера или часов реального времени).

Для прерывания от клавиатуры необходимо считывать скан-коды клавиш и выводить их на экран.

По нажатию определенной клавиши (любой на выбор студента) осуществляется обратный переход в реальный режим и выход из программы.

Необходимо отслеживать количество вызовов прерывания и отсчитывать секунды, выводя их на экран.

3. По прошествии определенного количества секунд (вводится с клавиатуры) определенного события выполняет обратный переход из защищенного режима в реальный и завершает свою работу.

**Алгоритм**

Программа состоит из нескольких частей, представляющих собой некоторые функции, с помощью которых осуществляется следующее:

1. Ввод времени нахождения в защищенном режиме;
2. Настройка режима работы часов реального времени;
3. Открытие линии А20;
4. Сохранение маски прерываний;
5. Заполнение таблицы дескрипторов базовыми адресами сегментов;
6. Заполнение таблицы дескрипторов прерываний;
7. Переход в защищенный режим;
8. Обработка прерывания от клавиатуры;
9. Обработка прерывания от таймера;
10. Изменение размера сегментов;
11. Возвращение в реальный режим;
12. Перезагрузка сегментов на дескрипторы;
13. Реинициализация контроллера прерываний;
14. Восстановление маски прерываний;
15. Закрытие линии А20.

**Листинг программы**

.386P

.model large

;CODE\_REAL SEGMENT

code\_real segment para use16 ;16-digit segment

code\_real\_start = $ ;start of the code segment in real mode

assume cs:code\_real, ds:data, es:data ;matches segment registers with segments

begin:

mov ax, data

mov ds, ax

mov es, ax

lea dx, ask\_for\_input

mov ah, 9h

int 21h

call readTime

mov ds:[time], al

call readCR0Real

lea dx, buffer\_cr0\_real ;prints the contents of cr0 register

mov ah, 9h

int 21h

lea dx, welcome\_protected

mov ah,9h

int 21h

mov ah, 7h ;read pressed symbol without showing it on the screen

int 21h

RTC:

mov al, 0Bh

out 70h, al ;state register B, 70h - index port (RTC)

in al, 71h ;read data from RTC, 71h - data port (RTC)

or al, 00000100b ;the 2nd bit - data mode - binary

out 71h, al

PREPARE\_FOR\_PROTECTED\_MODE:

A20: ;A20-line is used for getting access to addresses > 1Mb, which is necessary for protected mode

in al, 92h ;92h is used for system settings

or al, 2 ;the first bit in 1 to connect A20 line

out 92h, al

save\_mask:

in al, 21h ;reads the mask of masking interrupts

mov mask\_master, al

in al, 0A1h ;reads the mask of unmasking interrupts

mov mask\_slave, al

disable\_interrupts: ;not necessary

cli ;disable masking interrupts

in al, 70h ;70h is used for accessing CMOS register

or al, 10000000b ;the 7th bit in 1 to disable unmasking interrupts

out 70h, al

nop ;delay

set\_gdt: ;GDT - global descriptor table

mov ax, data

mov dl, ah

xor dh, dh

shl ax, 4 ;ax \* 16

shr dx, 4 ;delets the 4 lowest bits

mov si, ax

mov di, dx

fill\_gdt: ;with values of basic address of the segment, so that sysytem could know, where the segments are

lea bx, gdt\_elem ;will be used for accessing GDT elements

mov ax, si

mov dx, di

add ax, offset GDT ;gets the actual address of a GDT element

adc dx, 0 ;adds values, counting carry in (though here there is no carry)

mov [bx][S\_DESC.BASE\_L], ax

mov [bx][S\_DESC.BASE\_M], dl

mov [bx][S\_DESC.BASE\_H], dh

write\_code\_real:

lea bx, GDT\_code ;the address of the code descriptor

mov ax, cs ;cs points to the current code segment

xor dh, dh

mov dl, ah

shl ax, 4 ;\* 16

shr dx, 4 ;gets the highest half of the basic address of the segment

mov [bx][S\_DESC.BASE\_L], ax

mov [bx][S\_DESC.BASE\_M], dl

mov [bx][S\_DESC.BASE\_H], dh

write\_data:

lea bx, GDT\_data

mov ax, si ;basic address of the data segment

mov dx, di ;the highet half

mov [bx][S\_DESC.BASE\_L], ax

mov [bx][S\_DESC.BASE\_M], dl

mov [bx][S\_DESC.BASE\_H], dh

write\_stack:

lea bx, GDT\_stack

mov ax, ss ;ss - segment selector, which points to the current stack segment

xor dh, dh ;the highest half of the basic address

mov dl, ah

shl ax, 4

shr dx, 4 ;gets the highest half of the basic address of the segment

mov [bx][S\_DESC.BASE\_L], ax

mov [bx][S\_DESC.BASE\_M], dl

mov [bx][S\_DESC.BASE\_H], dh

write\_code\_protected:

lea bx, GDT\_code\_protected

mov ax, code\_protected ;segment selector points to the code segment in the protected mode

xor dh, dh

mov dl, ah

shl ax, 4

shr dx, 4

mov [bx][S\_DESC.BASE\_L], ax

mov [bx][S\_DESC.BASE\_M], dl

mov [bx][S\_DESC.BASE\_H], dh

IDT\_to\_GDT: ;IDT - Interrupt Descriptor Table

lea bx, GDT\_IDT ;loads base address

mov ax, si

mov dx, di

add ax, offset IDT ;gets the actual address

adc dx, 0

mov [bx][S\_DESC.BASE\_L], ax

mov [bx][S\_DESC.BASE\_M], dl

mov [bx][S\_DESC.BASE\_H], dh

mov IDTR.IDT\_L, ax ;IDTR - Interrupt Descriptor Table Register

mov IDTR.IDT\_H, dx

fill\_IDT:

irpc N, 0123456789ABCDEF ;macroinstruction (iterate over characters) provides iteration of values from 0 to F

lea eax, EXC\_0&N ;creates the address of an interrupt handler

mov IDT\_0&N.OFFS\_L, ax ;loads the address in the descriptor of an interrupt handler, IDT0 - from 00 to 0F

shr eax, 16h

mov IDT\_0&N.OFFS\_H, ax

endm

irpc N, 0123456789ABCDEF

lea eax, EXC\_1&N

mov IDT\_1&N.OFFS\_L, ax ;IDT1 - from 10 to 1F

shr eax, 16h

mov IDT\_1&N.OFFS\_H, ax

endm

lea eax, timer\_handler ;creates a dummy handler (no useful actions) for an interrupt handler (0x20)

mov IDT\_timer.OFFS\_L, ax

shr eax, 16h

mov IDT\_timer.OFFS\_H, ax

lea eax, keyboard\_handler ;0x21

mov IDT\_keyboard.OFFS\_L, ax

shr eax, 16h

mov IDT\_keyboard.OFFS\_H, ax

irpc N, 234567

lea eax, IRQ\_master ;0x22-0x27

mov IDT\_2&N.OFFS\_L, AX

shr eax, 16h

mov IDT\_2&N.OFFS\_H, AX

endm

irpc N, 89ABCDEF

lea eax, IRQ\_slave ;0x28-0x2F

mov IDT\_2&N.OFFS\_L, ax

shr eax, 16h

mov IDT\_2&N.OFFS\_H, ax

endm

lgdt fword ptr gdt\_elem ;loads gdt\_elem into register GDT

lidt fword ptr IDTR ;loads register IDTR

mov eax, cr0

or al, 00000001b ;bit Protection Enable in 1

mov cr0, eax

overload\_cs: ;reload code segment to its descriptor

db 0eah ;1-byte custom comand for far shifting

dw offset overload\_ds\_es\_sc ;changes the contents of IP and CS for descriptors selector

dw descriptor\_code

overload\_ds\_es\_sc:

mov ax, descriptor\_data

mov ds, ax

mov es, ax

mov ax, descriptor\_stack

mov ss, ax

xor ax, ax

mov fs, ax ;reset

mov gs, ax

lldt ax ;LDTR is not used

PREPARE\_FOR\_REAL\_MODE:

push cs

push offset return\_to\_real

lea edi, enter\_protected

mov eax, descriptor\_code\_protected

push eax

push edi

reinit\_interrupt\_controller\_for\_protected:

mov al, 00010001b ;ICW1 - config byte for reinit

out 20h, al ;master

out 0A0h, al ;slave

mov al, 20h ;base interrupt vector (ICW2) for master controller

out 21h, al ;set it

mov al, 28h ;-||- for slave

out 0A1h, al

mov al, 04h ;ICW3 - controler connected to the 3rd line

out 21h, al

mov al, 02h

out 0A1h, al

mov al, 11h ;ICW4 - special nesting mode

out 21h, al

mov al, 01h ;ICW4 - general nesting mode

out 0A1h, al

mov al, 0 ;unmask interrupts

out 21h, al

out 0A1h, al

enable\_interrupts\_0:

in al, 70h

and al, 01111111b ;7th bit in 0 for forbidding unmasking interrupts

out 70h, al

nop

sti ;permit masking interrupts

go\_to\_code\_protected:

db 66h ;states, that the next instruction is 32-bit

retf

return\_to\_real:

cli

in al, 70h

or AL, 10000000b

out 70h, AL

nop

reinit\_interrupt\_controller: ;again

mov al, 00010001b ;ICW1 - config byte for reinit

out 20h, al ;master

out 0A0h, al ;slave

mov al, 20h ;base interrupt vector (ICW2) for master controller

out 21h, al ;set it

mov al, 28h ;-||- for slave

out 0A1h, al

mov al, 04h ;ICW3 - controler connected to the 3rd line

out 21h, al

mov al, 02h

out 0A1h, al

mov al, 11h ;ICW4 - special nesting mode

out 21h, al

mov al, 01h ;ICW4 - general nesting mode

out 0A1h, al

mov al, 0 ;unmask interrupts

out 21h, al

out 0A1h, al

prepare\_segments\_for\_real:

mov GDT\_code.LIMIT, 0FFFFh ;64 Kb

mov GDT\_data.LIMIT, 0FFFFh

mov GDT\_stack.LIMIT, 0FFFFh

db 0EAH ;overload cs to code segment in real mode

dw offset continue

dw descriptor\_code

continue:

mov ax, descriptor\_data

mov ds, ax

mov es, ax

mov fs, ax

mov gs, ax

mov ax, descriptor\_stack

mov ss, ax

enter\_real:

mov eax, cr0

and al, 11111110b

mov cr0, eax

db 0EAH

dw offset continue2

dw code

continue2:

mov ax, stack

mov ss, ax

mov ax, data

mov ds, ax

mov es, ax

xor ax, ax

mov fs, ax

mov gs, ax

mov IDTR.LIMIT, 3FFh

mov dword ptr IDTR+2, 0 ;reset to base address

lidt fword ptr IDTR ;load IDT to processor

restore\_mask:

mov al, mask\_master

out 21h, al

mov al, mask\_slave

out 0A1h, al

enable\_interrupts: ;again

in al, 70h

and al, 01111111b ;7th bit in 0 for forbidding unmasking interrupts

out 70h, al

nop

sti ;permit masking interrupts

A20\_close:

in al, 92h

and al, 11111101b ;the 1st bit in 0

out 92h, al

exit:

mov ax, 3h ;reset videomode

int 10h

lea dx, welcome\_back\_real

mov ah, 9h

int 21h

call readCR0Real

lea dx, buffer\_cr0\_real

mov ah, 9h

int 21h

mov ax, 4C00h

int 21h

;PROCEDURES\_CODE\_REAL

readTime proc near ;reads number of seconds

mov ah, 0ah ;reads a string

xor di, di

mov dx, offset ds:[time\_string]

int 21h

mov dl, 0ah

mov ah, 2h ;prints a symbol

int 21h

mov si, offset time\_string+2 ;skip the first two symbols

cmp byte ptr [si], "-" ;DELETE

jnz no\_minus

mov di, 1 ;there is a minus

inc si ;skip minus

no\_minus:

xor ax, ax ;digital value of time will be stored in ax

mov bx, 10

convert:

mov cl, [si] ;loads another symbol of the string into cl

cmp cl, 0dh ;if the end of the string is reached

jz done

cmp cl, '0'

jl not\_a\_digit ;if a symbol is not a digit

cmp cl, '9'

ja not\_a\_digit ;if a symbol is not a digit

sub cl, '0' ;converts char into int

mul bx

add ax, cx

inc si ;next symbol

jmp convert

not\_a\_digit:

mov dx, offset error\_wrong\_type

mov ah, 9h

int 21h

int 20h ;exit

done:

ret

readTime endp

readCR0Real proc near ;gets information from cr0 register - control register, which stores flags, including the 0th bit - enable protected mode

push eax

push esi

push dx

mov eax, cr0

xor dx, dx

mov cx, 32 ;number of bits in cr0

lea esi, buffer\_cr0\_real

read\_cr0:

mov dl, al

shl dl, 7 ;shift dl to the left by 7, leaving the highest bit in the lowest halfbyte

shr dl, 7 ;the lowest bit - 0

shr eax, 1 ;eliminates the highest bit

add dl, 48 ;48 - code of '0'

mov [esi], dl

inc esi ;next buffer element

xor dl, dl

loop read\_cr0

pop dx

pop esi

pop eax

ret

readCR0Real endp

size\_code = ($ - code\_BEGIN) ;Лимит сегмента кода

code\_real ends

;CODE\_PROTECTED SEGMENT

code\_protected segment para use32

code\_protected\_start = $

assume cs:code\_protected, ds:data, es:data

enter\_protected:

call clear

xor edi, edi

lea esi, greetings\_protected

call printBuffer

add edi, 160 ;\n

lea esi, keystroke

call printBuffer

mov edi, 320

lea esi, left\_time

call printBuffer

mov edi, 480

lea esi, counter\_of\_interrupts

call printBuffer

call fillCR0

mov edi, 640

lea esi, buffer\_cr0

call printBuffer

mov DS:[counter\_int], 0

wait\_for\_ESC:

jmp wait\_for\_ESC ;if not ESC

exit\_from\_protected:

db 66H

retf

return\_from\_interrupt:

popad

pop es

pop ds

pop eax ;pop EIP - Instruction Pointer

pop eax ;pop CS

pop eax ;pop EFLAGS

sti

db 66H

retf

;PROCEDURES\_CODE\_PROTECTED

clear proc near

push es

pushad

mov ax, descriptor\_txt

mov es, ax

xor edi, edi ;указатель назначения операций копирования на начало текстового сегмента или окна

mov ecx, 80\*25 ;number of words

mov ax, 700h ;' '

rep stosw ;fills the screen with 700h

popad

pop es

ret

clear endp

fillCR0 proc near

push eax

push esi

push dx

mov eax, cr0

xor dx, dx

mov cx, 32

lea esi, buffer\_cr0

fillCR0\_loop:

mov dl, al

shl dl, 7

shr dl, 7 ;the lowest bit of cr0 in dl

shr eax, 1 ;next bit

add dl, 48 ;'0'

mov [esi], dl ;save bit to buffer

inc esi

xor dl, dl

loop fillCR0\_loop

pop dx

pop esi

pop eax

ret

fillCR0 endp

wordToDec proc near

pushad

movzx eax, ax

xor cx, cx

mov bx, 10

atoi:

xor dx, dx ;for remainders

div bx

add dl, '0'

push dx

inc cx

test ax, ax ;if the end is reached

jnz atoi

store\_sym\_loop:

pop dx

mov [di], dl

inc di

loop store\_sym\_loop

popad

ret

wordToDec endp

digitToHex proc near

add al, '0'

cmp al, '9'

jle done\_hex ;<=

add al, 7 ;>

done\_hex:

ret

digitToHex endp

byteToHex proc near

push ax

mov ah, al

shr al, 4 ;the highest halfbyte is in al

call digitToHex

mov [di], al

inc di

mov al, ah ;restore the lowest byte

and al, 0Fh ;the highest halfbyte is reset

call digitToHex

mov [di], al

inc di

pop ax

ret

byteToHex endp

;EXCEPTIONS

M = 0

IRPC N, 0123456789ABCDEF

EXC\_0&N label word

cli

jmp exceptionReturn

endm

M = 010H

IRPC N, 0123456789ABCDEF

EXC\_1&N label word ;Обработчики исключений 10h..1Fh

cli ;Запрет прерываний

jmp exceptionReturn

endm

exceptionReturn proc near

call clear

lea esi, exception

mov edi, 40\*2

call printBuffer

pop eax

pop eax

pop eax

sti

db 66H

retf ;Переход в 16-битный сегмент кода

exceptionReturn ENDP

;DUMMY INTERRUPTS

IRQ\_master proc near

push eax

mov al, 20h ;reset mask of interrupts

out 20h, al

pop eax

iretd ;restores IF and return address

IRQ\_master endp

IRQ\_slave proc near

push eax

mov al, 20h

out 20h, al

out 0A0h, al

pop eax

iretd

IRQ\_slave endp

timer\_handler proc near

push ds

push es

pushad

mov ax, descriptor\_data

mov ds, ax

inc ds:[counter\_int]

lea edi, ds:[counter\_buffer]

mov ax, ds:[counter\_int]

call wordToDec

mov edi, 538

lea esi, counter\_buffer

call printBuffer

countdown:

mov al, 0h ;choose seconds register

out 70h, al

in al, 71h ;read seconds

cmp al, ds:[current\_sec] ;if it did not change

je skip\_sec

mov ds:[current\_sec], al

mov al, ds:[time] ;what time is left?

cmp ds:[time], 0 ;if it passed

je leave\_protected

xor ah, ah

lea edi, ds:[buffer\_time]

call wordToDec

mov edi, 356

lea esi, buffer\_time

call printBuffer

dec ds:[time]

lea esi, buffer\_time

call clearBuffer

jmp skip\_sec

leave\_protected:

mov al, 20h ;reset timer interrup

out 20h, al

db 0eah ;far jump

dd offset return\_from\_interrupt

dw descriptor\_code\_protected

skip\_sec:

mov al, 20h

out 20h, al

popad

pop es

pop ds

iretd

timer\_handler endp

keyboard\_handler proc near

push ds

push es

pushad

in al, 60h ;read scan-code of the latest pressed keystroke

cmp al, 1 ;if it was ESC

je exit\_keyboard

mov ds:[keystroke\_code], al

lea edi, ds:[buffer\_scan\_code]

mov al, ds:[keystroke\_code]

xor ah, ah

call byteToHex

mov edi, 200

lea esi, buffer\_scan\_code

call printBuffer

jmp return\_from\_keyboard

exit\_keyboard:

mov al, 20h

out 20h, al

db 0eah

dd OFFSET return\_from\_interrupt

dw descriptor\_code\_protected

return\_from\_keyboard:

mov al, 20h

out 20h, al

popad

pop es

pop ds

iretd

keyboard\_handler endp

clearBuffer proc near ;Процедура очистки буфера

mov al, ' '

mov [esi+0], al

mov [esi+1], al

mov [esi+2], al

mov [esi+3], al

mov [esi+4], al

mov [esi+5], al

mov [esi+6], al

mov [esi+7], al

ret

clearBuffer endp

printBuffer proc near ;buffer must end with 0

push es

pushad

mov ax, descriptor\_txt

mov es, ax

printf\_buffer\_loop:

lodsb ;next byte

or al, al

jz done\_print ;if the end of buffer is reached

stosb ;from al to es:edi

inc edi

jmp printf\_buffer\_loop

done\_print:

popad

pop es

ret

printBuffer ENDP

size\_code\_protected = ($ - code\_protected\_start)

code\_protected ENDS

;DATA SEGMENT

data segment para use16

data\_start = $

S\_DESC struc ;structure of segment descriptor

LIMIT dw 0

BASE\_L dw 0 ;base address, lowest part

BASE\_M db 0 ;base address, middle part

ACCESS db 0

ATTRIBS db 0 ;segment limit and attributes

BASE\_H db 0 ;base address, highest part

S\_DESC ends

I\_DESC struc ;structure of ITR descriptor

OFFS\_L dw 0 ;address of a handler (0:15)

SEL dw 0 ;code selector

PARAM\_CNT db 0

ACCESS db 0

OFFS\_H dw 0 ;address of a handler (31:16)

I\_DESC ends

R\_IDTR struc ;IDTR structure:

LIMIT dw 0 ;16-Bit table limit (0..15)

IDT\_L dw 0 ;32-Bit linear base address (16..31)

IDT\_H dw 0 ;32-Bit linear base address (32..47)

R\_IDTR ends

;GDT - глобальная таблица дескрипторов

GDT\_BEGIN = $

GDT label word ;starting point

GDT\_0 S\_DESC <0, 0, 0, 0, 0, 0>

gdt\_elem S\_DESC <GDT\_size-1, , , 10010010b, 0,>

GDT\_code S\_DESC <size\_code-1, , , 10011010b, 0,>

GDT\_data S\_DESC <data\_size-1, , , 11110010b, 0,>

GDT\_stack S\_DESC <1000h-1, , , 10010010b, 0,>

GDT\_text S\_DESC <2000h-1, 8000h, 0Bh, 11110010b, 0, 0>

GDT\_code\_protected S\_DESC <size\_code\_protected-1, , , 10011010b, 01000000b,>

GDT\_IDT S\_DESC <IDT\_size-1, , , 10010010b, 0,>

GDT\_size = ($ - GDT\_BEGIN)

;SELECTORS

descriptor\_code = (GDT\_code - GDT\_0)

descriptor\_data = (GDT\_data - GDT\_0)

descriptor\_stack = (GDT\_stack - GDT\_0)

descriptor\_txt = (GDT\_TEXT - GDT\_0)

descriptor\_code\_protected = (GDT\_code\_protected - GDT\_0)

descriptor\_IDT = (GDT\_IDT - GDT\_0)

;IDT

IDTR R\_IDTR <IDT\_size, 0, 0> ;format of IDTR

IDT label word

IDT\_start = $

irpc N, 0123456789ABCDEF

IDT\_0&N I\_DESC <0, descriptor\_code\_protected, 0, 10001111b, 0> ;00...0F

endm

irpc N, 0123456789ABCDEF

IDT\_1&N I\_DESC <0, descriptor\_code\_protected, 0, 10001111b, 0> ;10...1F

endm

IDT\_timer I\_DESC <0, descriptor\_code\_protected, 0, 10001110b, 0> ;IRQ0

IDT\_keyboard I\_DESC <0, descriptor\_code\_protected, 0, 10001110b, 0> ;IRQ1

irpc N, 23456789ABCDEF

IDT\_2&N I\_DESC <0, descriptor\_code\_protected, 0, 10001110b, 0> ;22...2F

endm

IDT\_size = ($ - IDT\_start)

ask\_for\_input db "Enter time: $"

error\_wrong\_type db "Wrong type of time$"

welcome\_protected db "Press any key to enter the protected mode...", 13, 10, "$"

greetings\_protected db "Protected mode. To exit: press ESC", 0

welcome\_back\_real db "Back to real mode", 13, 10, "$"

keystroke db "Scan-code:", 0

left\_time db "Returning to real mode in XXXXXXX seconds", 0

counter\_of\_interrupts db "Number of interrupt calls:", 0

exception db "Exception: XX", 0

mask\_master db 1 dup(?) ;register of masks of master controller

mask\_slave db 1 dup(?)

keystroke\_code db 1 dup(?)

current\_sec db 1 dup(?)

time\_string db 6, 7 dup(?)

time db 1 dup(10)

counter\_int dw 1 dup(0) ;counter of interrupt calls (from 0 to 65535)

counter\_buffer db 8 dup(' ')

db 1 dup(0)

buffer\_scan\_code db 8 dup(' ')

db 1 dup(0)

buffer\_time db 8 dup(' ')

db 1 dup(0)

buffer\_cr0 db 32 dup('?')

db 1 dup(0)

buffer\_cr0\_real db 32 dup('?'), 13, 10, "$"

HEX\_TAB db "0123456789ABCDEF" ;table consists of exception numbers

ESP32 dd 1 dup(?) ;top of the stack

data\_size = ($ - data\_start)

data ends

;STACK SEGMENT

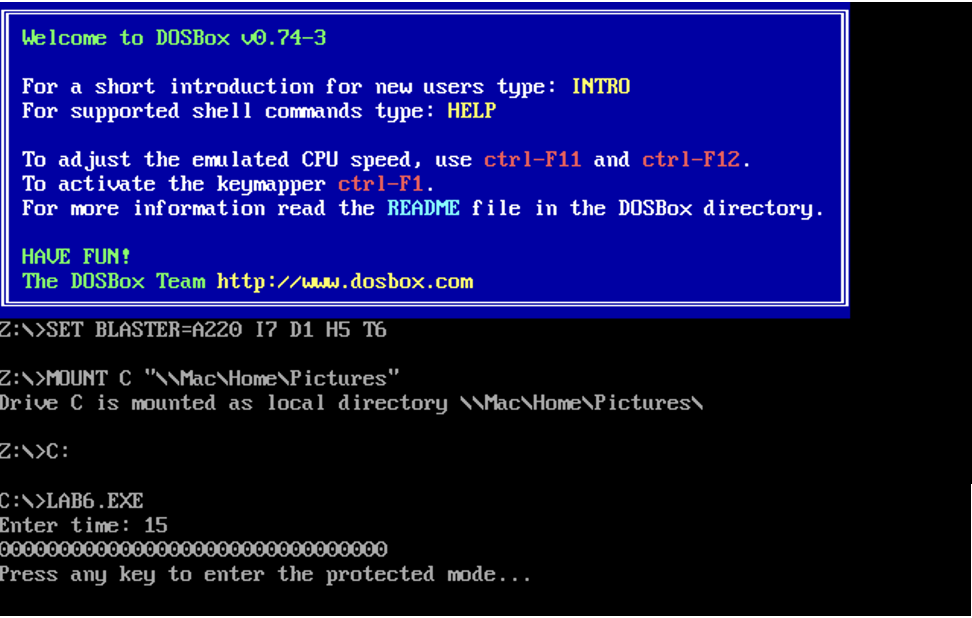
stack segment para stack

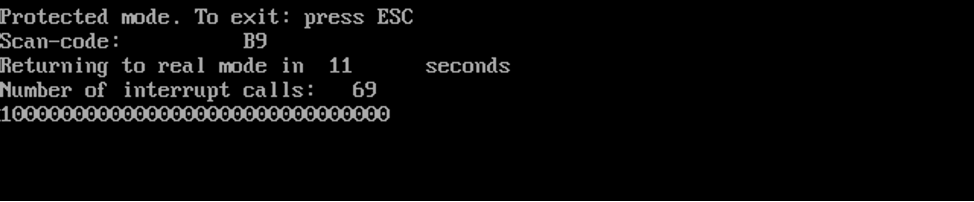
db 1000h dup(?)

stack ends

end begin

**Тест**

****

****

**Заключение**

В данной лабораторной работе разработана программа, которая осуществляет переход из реального режима в защищенный и обратно, а также выводит скан-коды нажимает клавиш, количество прерываний и отсчитывает время до завершения работы посредством системного таймера.

Для запуска исполняемого файла использовался DOSBox 0.74-3.