Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и   
радиоэлектроники»

Кафедра электронных вычислительных машин

Лабораторная работа №6

«Защищенный и реальный режим процессора.

Переход из одного режима в другой и обработка прерываний»

Вариант 11

Проверил: Выполнил:  
к. т. н., доцент студент гр. 150501

Одинец Д. Н. Климович А. Н.

Минск, 2023

**1 ПОСТАНОВКА ЗАДАЧИ**

Написать программу, которая выполняет следующие действия:

* Переход из реального режима в защищенный.
* Перехватывает аппаратное прерывание от клавиатуры и прерывание от таймера, в обработчике которого выполняет определенные действия.

По наступлению определенного события выполняет обратный переход из защищенного режима в реальный и завершает свою работу.

Для прерываний от таймера и часов реального времени обработчик

прерывания должен отслеживать количество вызовов прерывания и отсчитывать секунды, выводя их на экран. Количество секунд после которых выполняется обратный переход в реальный режим и выход из программы (то самое определенное событие) считывается с клавиатуры перед переходом в защищенный режим.

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.

При выполнении данной лабораторной работы должны быть соблюдены следующие условия:

* После завершения работы программы компьютер должен продолжать корректно функционировать. Зависания, перезагрузки и другие аналогичные «события» недопустимы.
* Переход в защищенный режим процессора должен быть выполнен по алгоритму, используемому в процессорах начиная с 386. Переход в защищенный режим с использованием алгоритма для 286 процессора недопустим.

2 АЛГОРИТМ

1. Начало.

2. Вводим время нахождения в защищенном режиме в секундах.

3. Настраиваем таймер на частоту 20 Гц.

4. Загружаем линейные адреса сегментов в дескрипторы.

5. Загружаем таблицу глобальных дескрипторов в регистр gdtr.

6. Запрещаем прерываний.

7. Сохраняем маски прерываний.

8. Инициализируем контроллеры.

9. Загружаем таблицу дескрипторов исключений в регистр idtr.

10. Переходим в защищенный режим.

11. Загружаем селекторы в регистры сегментов.

12. Разрешаем прерывания.

13. Производится обработка прерываний от клавиатуры и таймера.

14. Запрещаем прерываний.

15. Настраиваем теневые регистры сегментов для работы в реальном режиме.

16. Переходим в реальный режим.

17. Восстанавливаем значения сегментов.

18. Восстанавливаем таблицу векторов прерываний.

19. Инициализируем контроллеры.

20. Восстанавливаем маски прерываний.

21. Конец.

3 КОД ПРОГРАММЫ

Далее приведен листинг программы, которая выполняет поставленные задачи.

.386P

.MODEL LARGE

CODE\_RM segment para use16 ;16-разрядный сегмент

CODE\_RM\_BEGIN = $

assume cs:CODE\_RM,DS:DATA,ES:DATA

;===================================================================

START:

mov ax,DATA ;загрузка сегмента данных

mov ds,ax

mov es,ax

lea dx,MSG\_ENTER ;сообщение о времени нахождения

mov ah,9h

int 21h

call INPUT ;ввод данных с клавиатуры

mov ds:[TIME], al

call FILL\_CR\_0\_BUFFER\_RM ;чтение cr0

lea dx, MSG\_BUFFER\_CR\_0\_RM ;сообщение о выводе cr0

mov ah, 9h

int 21h

lea dx, BUFFER\_CR\_0\_RM ;вывод cr0

mov ah, 9h

int 21h

lea dx, MSG\_HELLO

mov ah,9h

int 21h

mov ah,7h ;press any key...

int 21h

PREPARE\_RTC: ;Подготовка часов RTC

mov al,0Bh

out 70h,al ;Выбрать регистр состояния 0Bh

in al,71h

or al,00000100b ;Установить бит DM в 1 - формат представления время в двоичном виде

out 71h,al

ENABLE\_A20: ;Открыть линию A20 - она отвечает за доступ к  
 ;памяти, физические адреса которой превышают 1 Мбайт

in al,92h

or al,2 ;Установить бит 1 в 1

out 92h,al

SAVE\_MASK: ;Сохранить маски прерываний

in al,21h

mov INT\_MASK\_M,al

in al,0A1h

mov INT\_MASK\_S,al

DISABLE\_INTERRUPTS: ;Запрет маскируемых и немаскируемых прерываний

cli ;Запрет маскирумых прерываний

in al,70h

or al,10000000b ;Установить 7 бит в 1 для запрета немаскируемых прерываний

out 70h,al

nop

LOAD\_GDT: ;Загружаем глобальную таблицу дескрипторов

mov ax,DATA

mov dl,ah

xor dh,dh

shl ax,4

shr dx,4

mov si,ax

mov di,dx

WRITE\_GDT: ;вычислим и загрузим в GDT линейный адрес сегмента глобальных таблиц

lea bx,GDT\_GDT ;bx - смещение дескриптора

mov ax,si

mov dx,di

add ax,offset GDT

adc dx,0

mov [bx][S\_DESC.BASE\_L],ax ;загрузка младшей части базы

mov [bx][S\_DESC.BASE\_M],dl ;загрузка средней части базы

mov [bx][S\_DESC.BASE\_H],dh ;загрузка старшей части базы

WRITE\_CODE\_RM: ;Заполнить дескриптор сегмента кода реального режима

lea bx,GDT\_CODE\_RM ;bx - смещение дескриптора

mov ax,cs

xor dh,dh

mov dl,ah

shl ax,4

shr dx,4

mov [bx][S\_DESC.BASE\_L],ax ;загрузка младшей части базы

mov [bx][S\_DESC.BASE\_M],dl ;загрузка средней части базы

mov [bx][S\_DESC.BASE\_H],dh ;загрузка старшей части базы

WRITE\_DATA: ;Записать дескриптор сегмента данных

lea bx,GDT\_DATA ;bx - смещение дескриптора

mov ax,si

mov dx,di

mov [bx][S\_DESC.BASE\_L],ax ;загрузка младшей части базы

mov [bx][S\_DESC.BASE\_M],dl ;загрузка средней части базы

mov [bx][S\_DESC.BASE\_H],dh ;загрузка старшей части базы

WRITE\_STACK: ;Записать дескриптор сегмента стека

lea bx, GDT\_STACK ;bx - смещение дескриптора

mov ax,ss

xor dh,dh

mov dl,ah

shl ax,4

shr dx,4

mov [bx][S\_DESC.BASE\_L],ax ;загрузка младшей части базы

mov [bx][S\_DESC.BASE\_M],dl ;загрузка средней части базы

mov [bx][S\_DESC.BASE\_H],dh ;загрузка старшей части базы

WRITE\_CODE\_PM: ;Записать дескриптор кода защищенного режима

lea bx,GDT\_CODE\_PM ;bx - смещение дескриптора

mov ax,CODE\_PM

xor dh,dh

mov dl,ah

shl ax,4

shr dx,4

mov [bx][S\_DESC.BASE\_L],ax ;загрузка младшей части базы

mov [bx][S\_DESC.BASE\_M],dl ;загрузка средней части базы

mov [bx][S\_DESC.BASE\_H],dh ;загрузка старшей части базы

WRITE\_IDT: ;Записать дескриптор IDT

lea bx,GDT\_IDT ;bx - смещение дескриптора

mov ax,si

mov dx,di

add ax,OFFSET IDT

adc dx,0

mov [bx][S\_DESC.BASE\_L],ax ;загрузка младшей части базы

mov [bx][S\_DESC.BASE\_M],dl ;загрузка средней части базы

mov [bx][S\_DESC.BASE\_H],dh ;загрузка старшей части базы

mov IDTR.IDT\_L,ax ;загрузим вудущий

mov IDTR.IDT\_H,dx ;загрузим ведомый

FILL\_IDT: ;Заполнить таблицу дескрипторов векторов прерываний

irpc N, 0123456789ABCDEF ;создание таблицы исключений 0

lea eax, EXC\_0&N

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

shr eax, 16

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

endm

irpc N, 0123456789ABCDEF ;создание таблицы исключений 1

lea eax, EXC\_1&N

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

shr eax, 16

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

endm

lea eax, TIMER\_HANDLER ;Поместить обработчик прерывания таймера на 20 шлюз

mov IDT\_TIMER.OFFS\_L,ax ;заносим прерывание в таблицу

shr eax, 16

mov IDT\_TIMER.OFFS\_H,ax

lea eax, KEYBOARD\_HANDLER ;Поместить обработчик прерывания клавиатуры на 21 шлюз

mov IDT\_KEYBOARD.OFFS\_L,ax ;заносим прерывание в таблицу

shr eax, 16

mov IDT\_KEYBOARD.OFFS\_H,ax

irpc N, 234567 ;Заполнить вектора IRQ2 - IRQ7 заглушками

lea eax,IDLE\_IRQ\_MASTER

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

shr eax,16

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

endm

irpc N, 89ABCDEF ;Заполнить вектора IRQ8-IRQF заглушками

lea eax,IDLE\_IRQ\_SLAVE

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

shr eax,16

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

endm

;Подготовка системных регистров и переход в РМ:

lgdt fword ptr GDT\_GDT ;Загрузить регистр GDTR

lidt fword ptr IDTR ;Загрузить регистр IDTR, fword == far word

mov eax,cr0 ;Получить управляющий регистр cr0

or al,00000001b ;Установить бит PE в 1

mov cr0,eax ;Записать измененный cr0 и тем самым включить защищенный режим

OVERLOAD\_CS: ;Перезагрузить сегмент кода на его дескриптор

db 0eah ;Искуственно сконструированная команда дальнего перехода

dw offset OVERLOAD\_SEGMENT\_REGISTERS ;Приведет к смене содержимого IP и CS на   
 ;селектор дескриптора

dw CODE\_RM\_DESC

OVERLOAD\_SEGMENT\_REGISTERS: ;Переинициализировать остальные сегментные регистры на дескрипторы

mov ax,DATA\_DESC

mov ds,ax ;В DS находится селектор дескриптора сегмента данных

mov es,ax ;Так же как в ES

mov ax,STACK\_DESC

mov ss,ax ;В SS селектор дескриптора сегмента стека

xor ax,ax

mov fs,ax ;Обнулить регистр fs

mov gs,ax ;Обнулить регистр gs

lldt ax ;Обнулить регистр LDTR - не использовать таблицы локальных дескрипторов

PREPARE\_TO\_RETURN: ;Подготовится к возврату в реальный режим

push cs ;Сохраним сегмент кода в стек

push offset BACK\_TO\_RM ;Сохраним смещение точки возврата

lea edi,ENTER\_PM ;Получить точку входа в защищенный режим

mov eax,CODE\_PM\_DESC ;Получить дескриптор кода защищенного режима

push eax ;Занести их в стек

push edi

REINITIALIAZE\_CONTROLLER\_FOR\_PM: ;Переинициализировать контроллер прерываний   
 ;на вектора 20h, 28h

mov al,00010001b ;ICW1 - переинициализация контроллера прерываний

out 20h,al ;Переинициализируем ведущий контроллер

out 0A0h,al ;Переинициализируем ведомый контроллер

mov al,20h ;ICW2 - номер базового вектора прерываний

out 21h,al ;ведущего контроллера

mov al,28h ;ICW2 - номер базового вектора прерываний

out 0A1h,al ;ведомого контроллера

mov al,04h ;ICW3 - ведущий контроллер подключен к 3 линии

out 21h,al

mov al,02h ;ICW3 - ведомый контроллер подключен к 3 линии

out 0A1h,al

mov al,11h ;ICW4 - режим специальной полной вложенности для ведущего контроллера

out 21h,al

mov al,01h ;ICW4 - режим обычной полной вложенности для ведомого контроллера

out 0A1h,al

mov al, 0 ;Размаскировать прерывания

out 21h,al ;Ведущего контроллера

out 0A1h,al ;Ведомого контроллера

ENABLE\_INTERRUPTS\_0: ;Разрешить маскируемые и немаскируемые прерывания

in al,70h

and al,01111111b ;Установить 7 бит в 0 для запрета немаскируемых прерываний

out 70h,al

nop

sti ;Разрешить маскируемые прерывания

GO\_TO\_CODE\_PM: ;Переход к сегменту кода защищенного режима

db 66h ;переход в 32 бит

retf

;===================================================================

BACK\_TO\_RM: ;Точка возврата в реальный режим

cli ;Запрет маскируемых прерываний

in al,70h ;И не маскируемых прерываний

or AL,10000000b ;Установить 7 бит в 1 для запрета немаскируемых прерываний

out 70h,AL

nop ;Ничего не делать(для задержки)

REINITIALISE\_CONTROLLER: ;Переиницализация контроллера прерываний

mov al,00010001b ;ICW1 - переинициализация контроллера прерываний

out 20h,al ;Переинициализируем ведущий контроллер

out 0A0h,al ;Переинициализируем ведомый контроллер

mov al,8h ;ICW2 - номер базового вектора прерываний

out 21h,al ;ведущего контроллера

mov al,70h ;ICW2 - номер базового вектора прерываний

out 0A1h,al ;ведомого контроллера

mov al,04h ;ICW3 - ведущий контроллер подключен к 3 линии

out 21h,al

mov al,02h ;ICW3 - ведомый контроллер подключен к 3 линии

out 0A1h,al

mov al,11h ;ICW4 - режим специальной полной вложенности для ведущего контроллера

out 21h,al

mov al,01h ;ICW4 - режим обычной полной вложенности для ведомого контроллера

out 0A1h,al

PREPARE\_SEGMENTS: ;Подготовка сегментных регистров для возврата в реальный режим

mov GDT\_CODE\_RM.LIMIT,0FFFFh ;Установка лимита сегмента кода в 64KB

mov GDT\_DATA.LIMIT,0FFFFh ;Установка лимита сегмента данных в 64KB

mov GDT\_STACK.LIMIT,0FFFFh ;Установка лимита сегмента стека в 64KB

db 0EAH ;Перезагрузить регистр cs

dw offset CONTINUE

dw CODE\_RM\_DESC ;На сегмент кода реального режима

CONTINUE:

mov ax,DATA\_DESC ;Загрузим сегментные регистры дескриптором сегмента данных

mov ds,ax

mov es,ax

mov fs,ax

mov gs,ax

mov ax,STACK\_DESC

mov ss,ax ;Загрузим регистр стека дескриптором стека

ENABLE\_REAL\_MODE: ;Включим реальный режим

mov eax,cr0

and al,11111110b ;Обнулим 0 бит регистра cr0

mov cr0,eax

db 0EAH

dw offset CONTINUE2

dw CODE\_RM

CONTINUE2: ;Перезагрузим регистр кода

mov ax,STACK\_A

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

lidt fword ptr IDTR

REPEAIR\_MASK: ;Восстановить маски прерываний

mov al,INT\_MASK\_M

out 21h,al ;Ведущего контроллера

mov al,INT\_MASK\_S

out 0A1h,al ;Ведомого контроллера

ENABLE\_INTERRUPTS: ;Разрешить маскируемые и немаскируемые прерывания

in al,70h

and al,01111111b ;Установить 7 бит в 0 для разрешения немаскируемых прерываний

out 70h,al

nop

sti ;Разрешить маскируемые прерывания

DISABLE\_A20: ;Закрыть линию A20

in al,92h

and al,11111101b ;Обнулить 1 бит - запретить линию A20

out 92h, al

;-------------------------------------------------------------------

EXIT: ;Выход из программы

mov ax,3h

int 10H ;Очистить видео-режим

lea dx,MSG\_EXIT

mov ah,9h

int 21h ;Вывести сообщение

call FILL\_CR\_0\_BUFFER\_RM ;Вывести содержимое рагистра CR0

lea dx, MSG\_BUFFER\_CR\_0\_RM ;сообщение о выводе cr0

mov ah, 9h

int 21h

lea dx, BUFFER\_CR\_0\_RM

mov ah, 9h

int 21h

mov ax,4C00h

int 21H ;Выход в dos

;-------------------------------------------------------------------

INPUT proc near ;Процедура ввода время-нахождения в защищенном режиме

mov ah,0ah

xor di,di

mov dx,offset ds:[INPUT\_TIME]

int 21h

mov dl,0ah

mov ah,02

int 21h

mov si,offset INPUT\_TIME+2

cmp byte ptr [si],"-"

jnz ii1

mov di,1

inc si

II1:

xor ax,ax

mov bx,10

II2:

mov cl,[si]

cmp cl,0dh

jz ii3

cmp cl,'0'

jl er

cmp cl,'9'

ja er

sub cl,'0'

mul bx

add ax,cx

inc si

jmp ii2

ER:

mov dx, offset MSG\_ERROR

mov ah,09

int 21h

int 20h

II3:

ret

INPUT endp

;-------------------------------------------------------------------

FILL\_CR\_0\_BUFFER\_RM proc near ;заполнение cr0

push eax

push esi

push dx

mov eax, cr0

xor dx, dx

mov cx, 32

lea esi, BUFFER\_CR\_0\_RM

fill\_cr\_0\_loop\_rm:

mov dl, al

shl dl, 7

shr dl, 7

shr eax, 1

add dl, 48

mov [esi], dl

inc esi

xor dl, dl

loop fill\_cr\_0\_loop\_rm

pop dx

pop esi

pop eax

ret

FILL\_CR\_0\_BUFFER\_RM endp

;-------------------------------------------------------------------

SIZE\_CODE\_RM = ($ - CODE\_RM\_BEGIN) ;Лимит сегмента кода

CODE\_RM ends

;===================================================================

;Сегмент кода защищенного режима

;===================================================================

CODE\_PM segment para use32

CODE\_PM\_BEGIN = $

assume cs:CODE\_PM,ds:DATA,es:DATA ;Указание сегментов для компиляции

;-------------------------------------------------------------------

ENTER\_PM: ;Точка входа в защищенный режим

call CLRSCR ;Процедура очистки экрана

xor edi,edi ;В edi смещение на экране

lea esi,MSG\_HELLO\_PM ;В esi адрес буфера

call BUFFER\_OUTPUT ;Вывести строку-приветствие в защищенном режиме

add edi,160 ;Перевести курсор на следующую строку

lea esi,MSG\_KEYBOARD

call BUFFER\_OUTPUT ;Вывести поле для вывода скан-кода клавиатуры

mov edi,320

lea esi,MSG\_TIME

call BUFFER\_OUTPUT ;Вывести поле для вывода время

mov edi,480

lea esi,MSG\_COUNT

call BUFFER\_OUTPUT

call FILL\_CR\_0\_BUFFER ;обновить cr0

mov edi, 640 ;курсор на новую строку

lea esi, MSG\_BUFFER\_CR\_0\_PM

call BUFFER\_OUTPUT

mov edi, 672

lea esi, BUFFER\_CR\_0

call BUFFER\_OUTPUT

mov DS:[COUNT],0

WAITING\_ESC: ;Ожидание нажатия кнопки выхода из защищенного режима

jmp WAITING\_ESC ;Если был нажат не ESC

EXIT\_PM: ;Точка выхода из 32-битного сегмента кода

db 66H

retf ;Переход в 16-битный сегмент кода

EXIT\_FROM\_INTERRUPT: ;Точка выхода для выхода напрямую из обработчика прерываний

popad

pop es

pop ds

pop eax ;Снять со стека старый EIP

pop eax ;CS

pop eax ;И EFLAGS

sti ;Обязательно, без этого обработка аппаратных прерываний отключена

db 66H

retf ;Переход в 16-битный сегмент кода

;-------------------------------------------------------------------

WORD\_TO\_DEC proc near

pushad

movzx eax,ax

xor cx,cx

mov bx,10

LOOP1:

xor dx,dx

div bx

add dl,'0'

push dx

inc cx

test ax,ax

jnz LOOP1

LOOP2:

pop dx

mov [di],dl

inc di

loop LOOP2

popad

ret

WORD\_TO\_DEC endp

;-------------------------------------------------------------------

FILL\_CR\_0\_BUFFER proc near

push eax

push esi

push dx

mov eax, cr0

xor dx, dx

mov cx, 32

lea esi, BUFFER\_CR\_0

fill\_cr\_0\_loop:

mov dl, al

shl dl, 7

shr dl, 7

shr eax, 1

add dl, 48

mov [esi], dl

inc esi

xor dl, dl

loop fill\_cr\_0\_loop

pop dx

pop esi

pop eax

ret

FILL\_CR\_0\_BUFFER endp

;-------------------------------------------------------------------

DIGIT\_TO\_HEX proc near

add al,'0'

cmp al,'9'

jle DTH\_END

add al,7

DTH\_END:

ret

DIGIT\_TO\_HEX endp

;-------------------------------------------------------------------

BYTE\_TO\_HEX proc near

push ax

mov ah,al

shr al,4

call DIGIT\_TO\_HEX

mov [di],al

inc di

mov al,ah

and al,0Fh

call DIGIT\_TO\_HEX

mov [di],al

inc di

pop ax

ret

BYTE\_TO\_HEX endp

;-------------------------------------------------------------------

M = 0

IRPC N, 0123456789ABCDEF

EXC\_0&N label word ;Обработчики исключений 00h..0Fh

cli ;Запрет прерываний

jmp EXC\_HANDLER

endm

M = 010H

IRPC N, 0123456789ABCDEF

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

cli ;Запрет прерываний

jmp EXC\_HANDLER

endm

;-------------------------------------------------------------------

EXC\_HANDLER proc near ;Процедура вывода обработки исключений

call CLRSCR ;Очистка экрана

lea esi, MSG\_EXC

mov edi, 40\*2

call BUFFER\_OUTPUT ;Вывод предупреждения

pop eax ;Снять со стека старый EIP

pop eax ;CS

pop eax ;И EFLAGS

sti ;Обязательно, без этого обработка аппаратных прерываний отключена

db 66H

retf ;Переход в 16-битный сегмент кода

EXC\_HANDLER ENDP

;-------------------------------------------------------------------

IDLE\_IRQ\_MASTER proc near ;Заглушка для аппаратных прерываний ведущего контроллера

push eax

mov al,20h

out 20h,al

pop eax

iretd

IDLE\_IRQ\_MASTER endp

;-------------------------------------------------------------------

IDLE\_IRQ\_SLAVE proc near ;Заглушка для аппаратных прерываний ведомого контроллера

push eax

mov al,20h

out 20h,al

out 0A0h,al

pop eax

iretd

IDLE\_IRQ\_SLAVE endp

;-------------------------------------------------------------------

TIMER\_HANDLER proc near ;Обработчик прерываний системного таймера

push ds

push es

pushad ;Занести в стек расширенные регистры общего назначения

mov ax,DATA\_DESC ;Переинициализировать сегментные регистры

mov ds,ax

inc ds:[COUNT] ;Увеличить значение счетчика

lea edi,ds:[BUFFER\_COUNT]

mov ax,ds:[COUNT]

call WORD\_TO\_DEC ;Преобразовать счётчик в строку

mov edi,538

lea esi,BUFFER\_COUNT

call BUFFER\_OUTPUT ;Вывести значение счетчика

SHOW\_TIMER:

mov al,0h ;Выбрать регистр секунд cmos

out 70h,al

in al,71h ;Прочитать значение секунд

cmp al,ds:[SECOND] ;Если секунда та же самая

je SKIP\_SECOND ;То пропустить вывод

mov ds:[SECOND],al ;Иначе записать значение новой секунды

mov al,ds:[TIME] ;Получить значение оставшегося время

cmp ds:[TIME],0 ;Если время подошло к концу

je DISABLE\_PM ;То на выход из защищенного режима

xor ah,ah

lea edi,ds:[BUFFER\_TIME]

call WORD\_TO\_DEC ;Преобразовать его в строку

mov edi,356

lea esi,BUFFER\_TIME

call BUFFER\_OUTPUT ;Вывести значение оставшегося время

dec ds:[TIME] ;Уменьшить значение оставшегося времени

lea esi,BUFFER\_TIME

call BUFFER\_CLEAR ;Очистка буфера

jmp SKIP\_SECOND ;На выход из обработки время

DISABLE\_PM: ;Выход из защищенного режима

mov al,20h

out 20h,al

db 0eah ;Дальний jmp

dd OFFSET EXIT\_FROM\_INTERRUPT ;На метку

dw CODE\_PM\_DESC ;В сегменте

SKIP\_SECOND: ;Секунда та же, не надо производить никаких действий

mov al,20h

out 20h,al ;Отпарвка сигнала контроллеру прерываний

popad

pop es

pop ds

iretd

TIMER\_HANDLER endp

;-------------------------------------------------------------------

KEYBOARD\_HANDLER proc near ;Обработчик прерывания клавиатуры

push ds

push es

pushad ;Сохранить расширенные регистры общего назначения

in al,60h ;Считать скан код последней нажатой клавиши

cmp al,1 ;Если был нажат 'ESC'

je KEYBOARD\_EXIT ;Тогда на выход из защищенного режима

mov ds:[KEY\_SCAN\_CODE],al ;Записать его в память

lea edi,ds:[BUFFER\_SCAN\_CODE]

mov al,ds:[KEY\_SCAN\_CODE]

xor ah,ah

call BYTE\_TO\_HEX ;Преобразовать скан-код в строку

mov edi,200

lea esi,BUFFER\_SCAN\_CODE

call BUFFER\_OUTPUT ;Вывести строку со скан-кодом

jmp KEYBOARD\_RETURN

KEYBOARD\_EXIT:

mov al,20h

out 20h,al

db 0eah

dd OFFSET EXIT\_FROM\_INTERRUPT

dw CODE\_PM\_DESC

KEYBOARD\_RETURN:

mov al,20h

out 20h,al ;Отпарвка сигнала контроллеру прерываний

popad ;Восстановить значения регистров

pop es

pop ds

iretd ;Выход из прерывания

KEYBOARD\_HANDLER endp

;-------------------------------------------------------------------

CLRSCR proc near ;Процедура очистки консоли

push es

pushad

mov ax,TEXT\_DESC ;Поместить в ax дескриптор текста

mov es,ax

xor edi,edi

mov ecx,80\*25 ;Количество символов в окне

mov ax,700h

rep stosw

popad

pop es

ret

CLRSCR endp

;-------------------------------------------------------------------

BUFFER\_CLEAR 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

BUFFER\_CLEAR endp

;-------------------------------------------------------------------

BUFFER\_OUTPUT proc near ;Процедура вывода текстового буфера, оканчивающегося 0

push es

PUSHAD

mov ax,TEXT\_DESC ;Поместить в es селектор текста

mov es,ax

OUTPUT\_LOOP: ;Цикл по выводу буфера

lodsb

or al,al

jz OUTPUT\_EXIT ;Если дошло до 0, то конец выхода

stosb

inc edi

jmp OUTPUT\_LOOP

OUTPUT\_EXIT: ;Выход из процедуры вывода

popad

pop es

ret

BUFFER\_OUTPUT ENDP

;===================================================================

SIZE\_CODE\_PM = ($ - CODE\_PM\_BEGIN)

CODE\_PM ENDS

;Сегмент данных реального/защищенного режима

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DATA SEGMENT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*8

DATA segment para use16 ;Сегмент данных реального/защищенного режима

DATA\_BEGIN = $

;Структуры данных

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

S\_DESC struc ;Структура сегментного дескриптора

LIMIT dw 0 ;Лимит сегмента

BASE\_L dw 0 ;Адрес базы, младшая часть

BASE\_M db 0 ;Адрес базы, средняя часть

ACCESS db 0 ;Байт доступа

ATTRIBS db 0 ;Лимит сегмента и атрибуты

BASE\_H db 0 ;Адрес базы, старшая часть

S\_DESC ends

I\_DESC struc ;Структура дескриптора таблицы прерываний

OFFS\_L dw 0 ;Адрес обработчика (0:15)

SEL dw 0 ;Селектор кода, содержащего код обработчика

PARAM\_CNT db 0 ;Параметры

ACCESS db 0 ;Уровень доступа

OFFS\_H dw 0 ;Адрес обработчика (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 (low) 16..31

IDT\_H dw 0 ; 32-Bit Linear Base Address (high) 32..47

R\_IDTR ends

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

;GDT - глобальная таблица дескрипторов

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

GDT\_BEGIN = $

GDT label word ;Метка начала GDT

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

GDT\_GDT S\_DESC <GDT\_SIZE-1,,,10010010b,0,>

GDT\_CODE\_RM S\_DESC <SIZE\_CODE\_RM-1,,,10011010b,0,>

GDT\_DATA S\_DESC <SIZE\_DATA-1,,,11110010b,0,>

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

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

GDT\_CODE\_PM S\_DESC <SIZE\_CODE\_PM-1,,,10011010b,01000000b,>

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

GDT\_SIZE = ($ - GDT\_BEGIN) ;Размер GDT

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

;Селлекторы сегментов

CODE\_RM\_DESC = (GDT\_CODE\_RM - GDT\_0)

DATA\_DESC = (GDT\_DATA - GDT\_0)

STACK\_DESC = (GDT\_STACK - GDT\_0)

TEXT\_DESC = (GDT\_TEXT - GDT\_0)

CODE\_PM\_DESC = (GDT\_CODE\_PM - GDT\_0)

IDT\_DESC = (GDT\_IDT - GDT\_0)

;IDT - таблица дескрипторов прерываний

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

IDTR R\_IDTR <SIZE\_IDT,0,0> ;Формат регистра IDTR

IDT label word ;Метка начала IDT

IDT\_BEGIN = $

IRPC N, 0123456789ABCDEF

IDT\_0&N I\_DESC <0, CODE\_PM\_DESC,0,10001111b,0> ; 00...0F

ENDM

IRPC N, 0123456789ABCDEF

IDT\_1&N I\_DESC <0, CODE\_PM\_DESC, 0, 10001111b, 0> ; 10...1F

ENDM

IDT\_TIMER I\_DESC <0,CODE\_PM\_DESC,0,10001110b,0> ;IRQ 0 - прерывание системного таймера

IDT\_KEYBOARD I\_DESC <0,CODE\_PM\_DESC,0,10001110b,0> ;IRQ 1 - прерывание клавиатуры

IRPC N, 23456789ABCDEF

IDT\_2&N I\_DESC <0, CODE\_PM\_DESC, 0, 10001110b, 0> ; 22...2F

ENDM

SIZE\_IDT = ($ - IDT\_BEGIN) ;Размер IDT

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

MSG\_HELLO db "Press key to change mode to PM",13,10,"$"

MSG\_HELLO\_PM db "We are in PM. Press ESC or wait till timer ends to exit PM",0

MSG\_EXIT db "We are in RM",13,10,"$"

MSG\_BUFFER\_CR\_0\_RM db "CR0 [b0...b31]: $"

MSG\_BUFFER\_CR\_0\_PM db "CR0 [b0...b31]: ",0

MSG\_KEYBOARD db "Scan code:",0

MSG\_TIME db "Go back to RM in XXXXXXX seconds",0

MSG\_COUNT db "Amount of interrupt calls:",0

MSG\_EXC db "Exception: XX",0

MSG\_ENTER db "Enter time in protected mode: $"

MSG\_ERROR db "incorrect error$"

HEX\_TAB db "0123456789ABCDEF" ;Таблица номеров исключений

ESP32 dd 1 dup(?) ;Указатель на вершину стека

INT\_MASK\_M db 1 dup(?) ;Значение регистра масок ведущего контроллера

INT\_MASK\_S db 1 dup(?) ;Значение регистра масок ведомого контроллера

KEY\_SCAN\_CODE db 1 dup(?) ;Ска-код последней нажатой клавиши

SECOND db 1 dup(?) ;Текущее значение секунд

TIME db 1 dup(10) ;Время нахождения в зазищенном режиме

COUNT dw 1 dup(0) ;Количество вызовов прерывания (диапазон от 0 до 65535)

BUFFER\_COUNT db 8 dup(' ') ;Буфер для вывода количества вызовов прерываинй

db 1 dup(0)

BUFFER\_SCAN\_CODE db 8 dup(' ') ;Буфер для вывода скан-кода клавиатуры

db 1 dup(0)

BUFFER\_TIME db 8 dup(' ') ;Буфер для вывода оставшегося время в защищенном режиме

db 1 dup(0)

INPUT\_TIME db 6,7 dup(?) ;Буфер для ввода время

BUFFER\_CR\_0 db 32 dup('?')

db 1 dup(0)

BUFFER\_CR\_0\_RM db 32 dup('?'), 13, 10, "$"

SIZE\_DATA = ($ - DATA\_BEGIN) ;size of data segment

DATA ends

;Сегмент стека реального/защищенного режима

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* STACK SEGMENT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

STACK\_A segment para stack

db 1000h dup(?)

STACK\_A ends

end START

4 ТЕСТИРОВАНИЕ

На рисунке 4.1 приведен скриншот ввода времени нахождения в защищенном режиме после запуска программы.

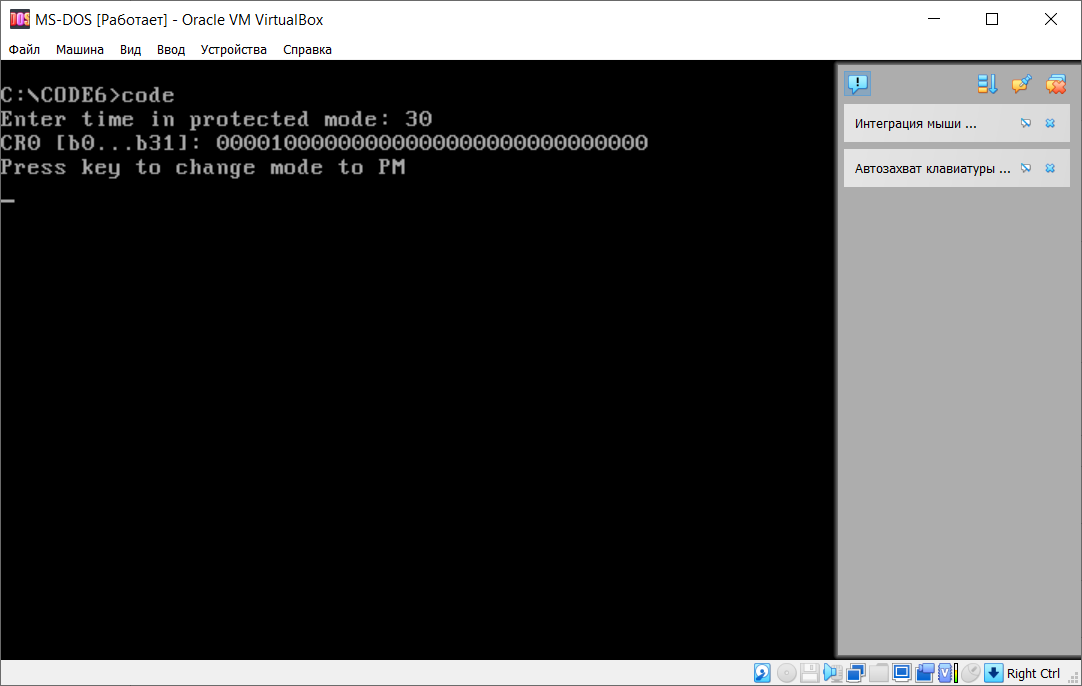


Рисунок 4.1 – Ввод времени нахождения в защищенном режиме после   
запуска программы в реальном режиме

Далее на рисунке 4.2 приведен скриншот пребывания программы   
в защищенном режиме.

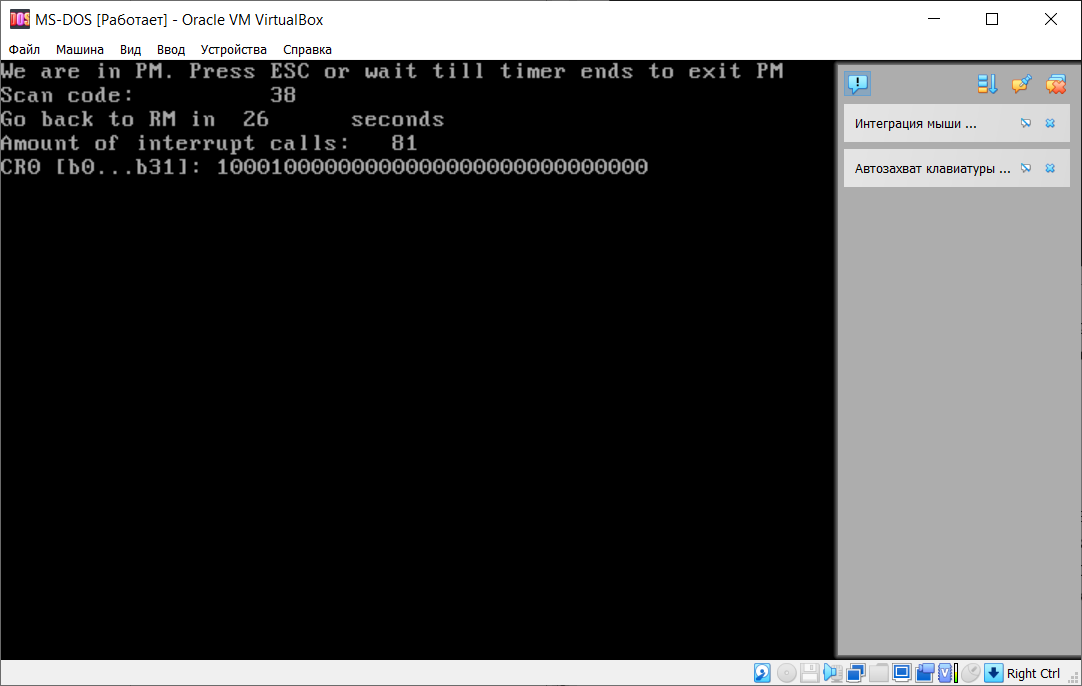


Рисунок 4.1 – Работа программы в защищенном режиме   
запуска программы в реальном режиме

На рисунке 4.3 изображен выход из защищенного режима в реальный.

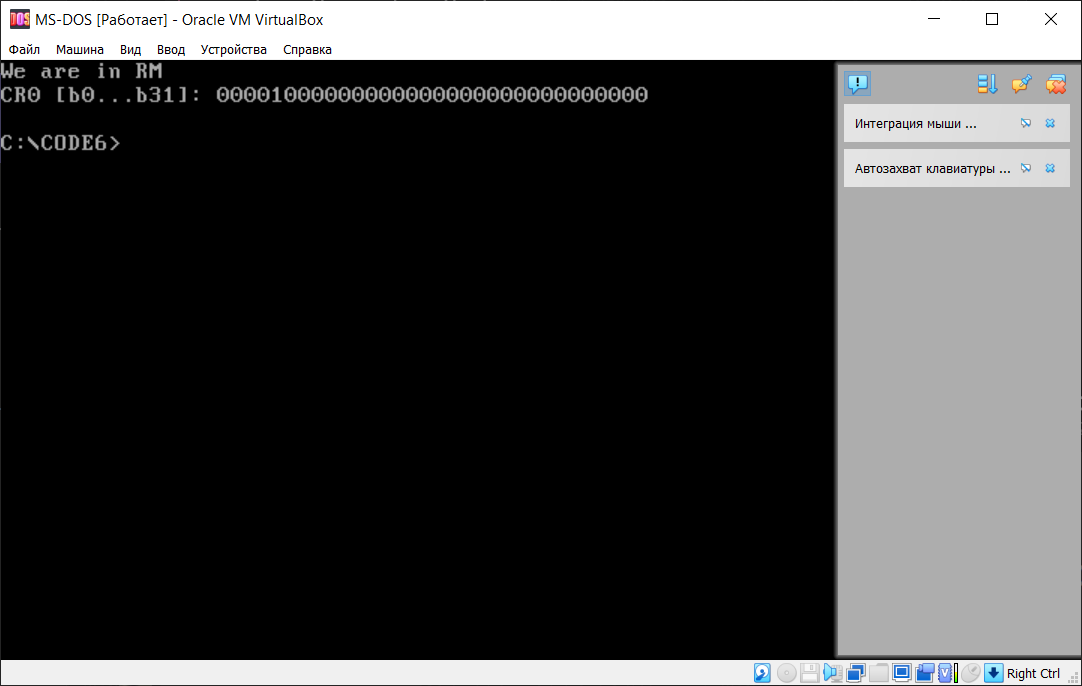


Рисунок 4.3 – Реальный режим процессора после работы в защищенном   
режиме

5 ЗАКЛЮЧЕНИЕ

В данной лабораторной работе были выполнены все поставленные задачи: был выполнен успешный переход в защищенный режим и возврат из него. Были написаны обработчики прерываний клавиатуры и таймера, выполняющие свою работу в защищенном режиме.

Программа запускалась в DOS, который эмулировался с помощью   
Oracle Virtual VM VirtualBox на хосте Windows 10.