Белорусский Государственный Университет Информатики и Радиоэлектроники

Кафедра ЭВМ

Отчет по лабораторной работе № 6

Тема: «Защищенный режим процессора. Мультизадачность»

Выполнил студент гр.05050X: Проверил:

MRY Одинец Д.Н.

Минск 2012

1. **Задание**

Написать программу, которая выполняет следующие действия:

1. Выполняет переход из реального режима в защищенный.
2. Организует минимум три задачи, переключающиеся между собой.
3. Выполняет обратный переход из защищенного режима в реальный.

Каждая задача должна выполнять действие, по которому можно будет сделать вывод о том, что она работает (например, одна задача выводит символы справа налево и сверху вниз, другая организует счетчик, третья выводит символы слева на право и снизу вверх).

Возврат в реальный режим также осуществляется по нажатию на “Esc”.

Переключение задач должно выполняться с использованием аппаратных прерываний по вариантам:

1. Прерывание от таймера.
2. Прерывание от часов реального времени.

Для прерываний от таймера и часов реального времени переключение задач осуществляется через определенные промежутки времени, вводимые с клавиатуры до перехода в защищенный режим.

При выполнении данной лабораторной работы должны быть выполнены следующие условия:

1. После завершения работы программы компьютер должен продолжать корректно функционировать. Зависания, перезагрузки и другие аналогичные «события» недопустимы.
2. 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.
3. **Разработка алгоритма**
   1. Структура программы

Программа состоит из нескольких процедур:

BUFFER\_CLEAR – процедура очистки буфера

BUFFER\_OUTPUT – процедура вывода буфера через видеопамять

CLR\_SCR – процедура очистки экрана

KEYBOARD\_HANDLER – обработчик прерывания от клавиатуры

TIMER\_HANDLER – обработчик прерывания системного таймера, осуществляет переключение задач

DUMMY\_IRQ\_MASTER/SLAVE – заглушка для обработчиков прерываний ведущего/ведомого контроллера

EXC\_HANDLER – обработчик исключений

BYTE\_TO\_HEX – перевод числа в шестнадцатеричный вид

DIGIT\_TO\_DEC – перевод цифры в десятичный вид

WORD\_TO\_DEC – перевод числа в десятичный вид

INPUT – ввод количества секунд

TASK\_2 – вторая дополнительная задача, осуществляет вывод красно/белой полосы слева-направо

TASK\_3 – третья дополнительная задача, осуществляет вывод зелено-желтой полосы справа-налево

WAITING\_ESC – главная задача, осуществляет вывод символов при нажатии клавиш 1-9

* 1. Описание переменных

GDT – глобальная таблица дескрипторов;

CODE\_RM\_DESC – селектор сегмента кода реального режима;

DATA\_DESC – селектор сегмента данных реального/защищенного режима;

STACK\_DESC – сегмент стека реального/защищенного режима;

TEXT\_DESC – селектор видеопамяти компьютера;

CODE\_PM\_DESC – селектор кода защищенного режима;

CODE\_IDT – селектор таблицы IDT;

TIME – время, которое компьютер будет находится в защищенном режиме

COUNT – количество вызовов прерывания системного таймера

KEY\_SCAN\_CODE – последняя введенная клавиша

SECOND – текущая секунда времени

CS\_2\_DESC – селектор кода второй задачи

CS\_3\_DESC – селектор кода третей задачи

SS\_2\_DESC – селектор стека второй задачи

SS\_3\_DESC – селектор стека третей задачи

TSS\_MAIN\_DESC – селектор TSS главной задачи

TSS\_2\_DESC – селектор TSS второй задачи

TSS\_3\_DESC – селектор TSS третей задачи

* 1. Описание алгоритма программы

При запуске пользователь вводит число секунд, которые программа будет находиться в защищенном режиме. Затем после нажатия любой клавиши программа начинает переходить в защищенный режим.

Для этого:

1. Подготавливаются часы RTC – формат представления данных в двоичном виде
2. Разрешается линия A20
3. Сохраняются маски прерываний
4. Запрещаются маскируемые и немаскируемые прерывания
5. Таблица GDT заполняется дескрипторами + **заполняются дескрипторы сегментов кода и стека дополнительный задач**
6. **Заполняются дескрипторы TSS дополнительных задач**
7. Заполняется таблица IDT
8. 0 бит регистра cr0 ставится в 1
9. Перезагружается регистр cs.
10. Перезагружаются другие сегментные регистры
11. В стек заносятся регистр cs и точка возврата в реальный режим
12. В стек заносятся точка входа в защищенный режим и дескриктор сегмента кода защищенного режима
13. **Заполняется TSS второй и третьей задачи**
14. Переинициализируется контроллеры прерываний на шлюзы 20 – 2F
15. Разрешаются маскируемые и немаскируемые прерывания
16. Программа переходит в сегмент 32-битного кода
17. Вызываются процедуры очистки экрана и вывода сообщения
18. **Загружается регистр TR, тем самым включается режим мультизадачности и заполняется TSS главной задачи**
19. Запускается бесконечный цикл, в котором **организуется выполнение главной задачи (вывод на экран символов 1-9), а также переключение по обработчику прерываний на выполнение второй задачи(вывод красно-белой строки) и третьей задачи(вывод зелено-желтой строки).** По команде выхода программа возвращается в сегмент кода реального режима
20. По возвращению вновь запрещаются все прерывания
21. Контроллеры прерываний переинициализируются на старые вектора
22. Устанавливаются новые лимиты сегментов данных, кода и стека
23. Перезагружается регистр cs
24. Включается реальный режим
25. Восстанавливаются сегментные регистры
26. Восстанавливаются маски контроллера прерываний
27. Запрещается линия A20
28. Программа выводит сообщение о успешном возврате
29. **Текст программы**

.386P

.MODEL LARGE

INCLUDE STRUCT.ASM

CODE\_RM segment para use16 ;Сегмент кода реального режима

CODE\_RM\_BEGIN = $

assume cs:CODE\_RM,DS:DATA,ES:DATA ;Инициализация регистров для ассемблирования

START:

mov ax,DATA ;Инициализиция сегментных регистров

mov ds,ax

mov es,ax

lea dx,MSG\_ENTER

mov ah,9h

int 21h

call INPUT ;Ввод времени

mov ds:[TIME], al

lea dx,MSG\_HELLO

mov ah,9h

int 21h

mov ah,7h

int 21h ;Ожидание подтверждения

PREPARE\_RTC: ;Подготовка часов RTC

mov al,0Bh

out 70h,al ;Выбрать регистр состояния 0Bh

in al,71h ;Получить значение регистра 0Bh

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

out 71h,al ;Записать измененное значение

ENABLE\_A20: ;Открыть линию A20

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: ;Заполнить 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: ;Заполнить дескриптор GDT

lea bx,GDT\_GDT

mov ax,si

mov dx,di

add ax,offset GDT

adc dx,0

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

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

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

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

lea bx,GDT\_CODE\_RM

mov ax,cs

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

WRITE\_DATA: ;Записать дескриптор сегмента данных

lea bx,GDT\_DATA

mov ax,si

mov dx,di

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

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

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

WRITE\_STACK: ;Записать дескриптор сегмента стека

lea bx, GDT\_STACK

mov ax,ss

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

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

lea bx,GDT\_CODE\_PM

mov ax,CODE\_PM

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

or [bx][S\_DESC.ATTRIBS],40h

WRITE\_IDT: ;Записать дескриптор IDT

lea bx,GDT\_IDT

mov ax,si

mov dx,di

add ax,offset IDT

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

mov IDTR.IDT\_H,dx

WRITE\_TASK\_CODE: ;Записать дескриптор кода второй и третей задачи

lea bx,GDT\_CS\_2

mov ax,CODE\_2

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

or [bx][S\_DESC.ATTRIBS],40H ;32-битный сегмент

lea bx, GDT\_CS\_3

mov ax, CODE\_3

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

or [bx][S\_DESC.ATTRIBS],40H

WRITE\_TASK\_STACK: ;Записать дескрипторы сегментов стека дополнительных задач

lea bx,GDT\_SS\_2

mov ax,STCK\_2

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

lea bx,GDT\_SS\_3

mov ax,STCK\_3

xor dh,dh

mov dl,ah

shl ax,4

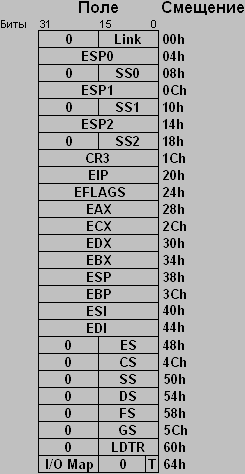
shr dx,4

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

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

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

WRITE\_TSS: ;Записать TSS задач ;

 LEA bx,GDT\_TSS\_MAIN

MOV ax,si

MOV dx,di

ADD ax,offset TSS\_MAIN

ADC dx,0

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

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

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

lea BX, GDT\_TSS\_2

mov AX, SI

mov DX, DI

add AX, offset TSS\_2

adc DX, 0

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

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

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

lea bx,GDT\_TSS\_3

mov ax,si

mov dx,di

add ax,offset TSS\_3

adc dx,0

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

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

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

FILL\_IDT: ;Заполнить таблицу дескрипторов шлюзов прерываний

irpc N, 0123456789ABCDEF ;Заполнить шлюзы 00-0F исключениями

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 ;Заполнить шлюзы 10-1F исключениями

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

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

shr eax, 16

mov IDT\_KEYBOARD.OFFS\_H,ax

irpc N, 234567 ;Заполнить вектора 22-27 заглушками

lea eax,DUMMY\_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 28-2F заглушками

lea eax,DUMMY\_IRQ\_SLAVE

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

shr eax,16

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

endm

lgdt fword ptr GDT\_GDT ;Загрузить регистр GDTR

lidt fword ptr IDTR ;Загрузить регистр IDTR

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

dw $+4

dw CODE\_RM\_DESC

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

mov ax,DATA\_DESC

mov ds,ax

mov es,ax

mov ax,STACK\_DESC

mov ss,ax

xor ax,ax

mov fs,ax ;Обнулить регистр fs

mov gs,ax ;Обнулить регистр gs

lldt ax ;Обнулить регистр LDTR - не использовать таблицы LDT

PREPARE\_TO\_RETURN:

push cs ;Сегмент кода

push offset BACK\_TO\_RM ;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

PREPARE\_TSS: ;Подготавливаю TSS для дополнительных задач

or TSS\_2.R\_EFLAGS, 3200H ;Установить 3 уровень привелегий и разрешить аппаратные прерывания

mov TSS\_2.R\_CS,CS\_2\_DESC ;Установить регистр сегмента кода

mov TSS\_2.R\_EIP,offset TASK\_2 ;Установить регистр команд на точку входа в задачу

mov TSS\_2.R\_SS, SS\_2\_DESC ;Установить сегмент стека задачи

mov TSS\_2.R\_ESP,0 ;Установить указатель стека на начало стека

mov ax, 0 ;Обнулить сегментные регистры

mov TSS\_2.R\_DS, AX

mov TSS\_2.R\_ES, AX

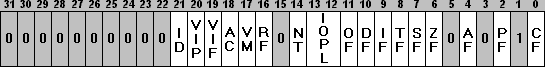
mov TSS\_2.R\_FS, AX

mov TSS\_2.R\_GS, AX

mov TSS\_2.SS0, STACK\_DESC ;Установить сегмент стека 0 уровня доступа

mov TSS\_2.ESP0, 500h ;И указатель в стеке

or TSS\_3.R\_EFLAGS,3200H ;Установить 3 уровень привелегий и разрешить аппаратные прерывания

 mov TSS\_3.R\_CS,CS\_3\_DESC

mov TSS\_3.R\_EIP,offset TASK\_3

mov TSS\_3.R\_SS, SS\_3\_DESC

mov TSS\_3.R\_ESP,0

mov ax, 0

mov TSS\_3.R\_DS, AX

mov TSS\_3.R\_ES, AX

mov TSS\_3.R\_FS, AX

mov TSS\_3.R\_GS, AX

mov TSS\_3.SS0, STACK\_DESC

mov TSS\_3.ESP0, 600h

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

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

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 $+4

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

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 $+4

dw CODE\_RM ;Перезагрузим регистр кода

mov al,20h

out 20h,al

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,20h

out 20h,al

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 вентиль 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

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

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

CODE\_RM ends

;Сегмент кода реального 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

MOV ax,TSS\_MAIN\_DESC

ltr ax ;Загрузить регистр tr и заполнить TSS для главной задачи

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

mov DS:[COUNT],0

mov edi,480

lea esi,MSG\_TASK\_1 ;Подготовить место для задач

call BUFFER\_OUTPUT

mov edi,640

lea esi,MSG\_TASK\_2

call Buffer\_OUTPUT

mov edi,800

lea esi,MSG\_TASK\_3

call BUFFER\_OUTPUT

mov edi,480

mov ax,TEXT\_DESC

mov es,ax

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

mov al,10

cmp ds:[KEY\_PRESSED],0

je NO\_KEY

mov al,ds:[KEY\_PRESSED]

dec al

add al,'0'

mov es:[di],al

mov ds:[KEY\_PRESSED],0

add di,2

cmp di,554

jne NO\_KEY

mov cx,37

mov di,480

FFLUSH:

mov al,' '

mov es:[di],al

add di,2

loop FFLUSH

mov di,480

mov al,ds:[KEY\_PRESSED]

dec al

add al,'0'

mov es:[di],al

mov ds:[KEY\_PRESSED],0

add di,2

NO\_KEY:

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

INCLUDE IRQ.ASM ;Файл с обработчиками аппаратных прерываний

INCLUDE PROC.ASM ;Файл с процедурами главной задачи защищенного режима

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

CODE\_PM ENDS

;Сегмент данных реального/защищенного режима

DATA segment para use16 ;Сегмент данных реального/защищенного режима

DATA\_BEGIN = $

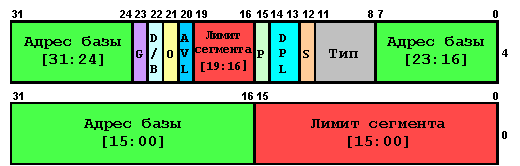
;GDT - глобальная таблица дескрипторов

GDT\_BEGIN = $

GDT label word ;Метка начала GDT (GDT: не работает)

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

GDT\_GDT S\_DESC <GDT\_SIZE-1,,,ACS\_DATA,0,>

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

GDT\_DATA S\_DESC <SIZE\_DATA-1,,,ACS\_DATA+ACS\_DPL\_3,0,>

GDT\_STACK S\_DESC <1000h-1,,,ACS\_DATA,0,>

GDT\_TEXT S\_DESC <2000h-1,8000h,0Bh,ACS\_DATA+ACS\_DPL\_3,0,0>

GDT\_CODE\_PM S\_DESC <SIZE\_CODE\_PM-1,,,ACS\_CODE+ACS\_READ,0,>

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

GDT\_CS\_2 S\_DESC <SIZE\_CS\_2-1,,,ACS\_CODE+ACS\_DPL\_3,0,>

GDT\_CS\_3 S\_DESC <SIZE\_CS\_3-1,,,ACS\_CODE+ACS\_DPL\_3,0,>

GDT\_SS\_2 S\_DESC <1000h-1,,,ACS\_DATA+ACS\_DPL\_3,0,>

GDT\_SS\_3 S\_DESC <1000h-1,,,ACS\_DATA+ACS\_DPL\_3,0,>

GDT\_TSS\_MAIN S\_DESC <SIZE\_TSS-1,,,ACS\_TSS,0,>

GDT\_TSS\_2 S\_DESC <SIZE\_TSS-1,,,ACS\_TSS,0,>

GDT\_TSS\_3 S\_DESC <SIZE\_TSS-1,,,ACS\_TSS,0,>

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

;Селлекторы сегментов

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

DATA\_DESC = (GDT\_DATA - GDT\_0)+ 3

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

TEXT\_DESC = (GDT\_TEXT - GDT\_0)+ 3

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

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

CS\_2\_DESC = (GDT\_CS\_2 - GDT\_0)+ 3

CS\_3\_DESC = (GDT\_CS\_3 - GDT\_0)+ 3

SS\_2\_DESC = (GDT\_SS\_2 - GDT\_0)+ 3

SS\_3\_DESC = (GDT\_SS\_3 - GDT\_0) + 3

TSS\_MAIN\_DESC = (GDT\_TSS\_MAIN - GDT\_0)

TSS\_2\_DESC = (GDT\_TSS\_2 - GDT\_0)

TSS\_3\_DESC = (GDT\_TSS\_3 - GDT\_0)

;Дескрипторы задач

TSS\_MAIN S\_TSS <>

TSS\_2 S\_TSS <>

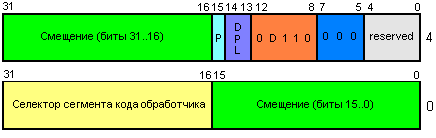
TSS\_3 S\_TSS <>

;Список задач

TASK\_LIST dw TSS\_MAIN\_DESC, TSS\_2\_DESC, TSS\_3\_DESC, 0

TASK\_INDEX dw 2 ;Номер включаемой задачи

TASK\_ADDR df 1 dup(0) ;6-байтный адрес для jmp far на задачу

 ;IDT - таблица дескрипторов прерываний

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

IDT label word ;Метка начала IDT

IDT\_BEGIN = $

IRPC N, 0123456789ABCDEF

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

ENDM

IRPC N, 0123456789ABCDEF

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

ENDM

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

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

IRPC N, 23456789ABCDEF

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

ENDM

SIZE\_IDT = ($ - IDT\_BEGIN)

MSG\_HELLO db "press any key to go to the protected mode",13,10,"$"

MSG\_HELLO\_PM db "wellcome in protected mode |",0

MSG\_EXIT db "wellcome back to real mode",13,10,"$"

MSG\_KEYBOARD db "keyboard scan code: | press 'ESC' to come back to the real mode",0

MSG\_TIME db "quantity of interrupt calls: | go back to RM in XXXXXXX seconds",0

MSG\_TASK\_1 db " | here is line for task 1",0

MSG\_TASK\_2 db " | here is line for task 2",0

MSG\_TASK\_3 db " | here is line for task 3",0

MSG\_OUTPUT\_TASK\_1 db 0DBh,0

MSG\_OUTPUT\_TASK\_2 db " ",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

KEY\_PRESSED db 1 dup(0) ;Последняя нажатая клавиша

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)

INTERVAL db 1 dup(0) ;Интервал между переключениями

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

SIZE\_DATA = ($ - DATA\_BEGIN) ;Размер сегмента данных

DATA ends

;Сегмент стека реального/защищенного режима

STACK\_A segment pra stack

db 1000h dup(?)

STACK\_A ends

INCLUDE TASKS.ASM ;Файл с двумя дополнительными задачами

end START

1. **Структура файла TASKS.ASM**

CODE\_2 segment para use32 ;Сегмент кода второй задачи - выводит красно-белую полосу слева-направо

assume cs:CODE\_2

CS\_2\_BEGIN = $

TASK\_2 proc

mov ax,DATA\_DESC

mov ds,ax

mov ax,TEXT\_DESC

mov es,ax

TASK\_2\_START:

mov edi,640

mov ecx,37

TASK\_2\_LOOP\_FILL: ;Цикл по выводу красной полосы

mov al,0DBh

mov es:[di],al

inc di

mov al,74h

mov es:[di],al

inc di

mov bx,cx

mov cx,60000

TASK\_2\_SLEEP\_0:

rept 90

nop

endm

loop TASK\_2\_SLEEP\_0

mov cx,bx

loop TASK\_2\_LOOP\_FILL

mov edi,640

mov ecx,37

TASK\_2\_LOOP\_CLEAN: ;Цикл по выводу белой полосы

mov al,' '

mov es:[di],al

inc di

mov al,74h

mov es:[di],al

inc di

mov bx,cx

mov cx,60000

TASK\_2\_SLEEP\_1:

rept 90

nop

endm

loop TASK\_2\_SLEEP\_1

mov cx,bx

loop TASK\_2\_LOOP\_CLEAN

TASK\_2\_END:

jmp TASK\_2\_START

TASK\_2 endp

SIZE\_CS\_2 = ($ - CS\_2\_BEGIN)

CODE\_2 ends

CODE\_3 segment para use32 ;Сегмент кода 3 задачи - выводит зелено-желтую полосу справа-налево

assume cs:CODE\_3

CS\_3\_BEGIN = $

TASK\_3 proc near

mov ax,DATA\_DESC

mov ds,ax

mov ax,TEXT\_DESC

mov es,ax

TASK\_3\_START:

mov edi,872

mov ecx,37

TASK\_3\_LOOP\_FILL: ;Цикл по выводу зеленой полосы

mov al,0DBh

mov es:[di],al

inc di

mov al,46

mov es:[di],al

sub di,3

mov bx,cx

mov cx,60000

TASK\_3\_SLEEP\_0:

rept 90

nop

endm

loop TASK\_3\_SLEEP\_0

mov cx,bx

loop TASK\_3\_LOOP\_FILL

mov edi,872

mov ecx,37

TASK\_3\_LOOP\_CLEAN: ;Цикл по выводу желтой полосы

mov al,' '

mov es:[di],al

inc di

mov al,46

mov es:[di],al

sub di,3

mov bx,cx

mov cx,60000

TASK\_3\_SLEEP\_1:

rept 90

nop

endm

loop TASK\_3\_SLEEP\_1

mov cx,bx

loop TASK\_3\_LOOP\_CLEAN

TASK\_3\_END:

jmp TASK\_3\_START

TASK\_3 endp

SIZE\_CS\_3 = ($ - CS\_3\_BEGIN)

CODE\_3 ends

STCK\_2 segment para

db 1000h DUP (?)

STCK\_2 ends

STCK\_3 segment para

db 1000h DUP (?)

STCK\_3 ends

1. **Структура файла IRQ.ASM**

M = 0

IRPC N, 0123456789ABCDEF

EXC\_0&N label word ;Обработчики исключений

cli

jmp EXC\_HANDLER

endm

M = 010H

IRPC N, 0123456789ABCDEF ;Обработчики исключений

EXC\_1&N label word

cli

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

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

push eax

mov al,20h

out 20h,al

pop eax

iretd

DUMMY\_IRQ\_MASTER endp

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

push eax

mov al,20h

out 20h,al

out 0A0h,al

pop eax

iretd

DUMMY\_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,378

lea esi,BUFFER\_COUNT

call BUFFER\_OUTPUT ;Вывести значение счетчика

SHOW\_TIMER:

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

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

jbe DISABLE\_PM ;То на выход из защищенного режима

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

xor ah,ah

lea edi,ds:[BUFFER\_TIME]

call WORD\_TO\_DEC ;Преобразовать его в строку

mov edi,432

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 bx,offset ds:[TSS\_MAIN\_DESC]

mov al,byte ptr [ bx + 5 ]

test al,00000010b

jnz GO

db 0eah

dd 0

dw TSS\_MAIN\_DESC

GO:

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

mov bx,offset TSS\_MAIN\_DESC

and byte ptr [bx+5],11111101b ;Обнулить флаг занятости главной задачи

mov bx,offset TSS\_2\_DESC

and byte ptr [bx+5],11111101b ;Обнулить флаг занятости 0 задачи

mov bx,offset TSS\_3\_DESC

and byte ptr [bx+5],11111101b ;Обнулить флаг занятости 1 задачи

movzx eax,ds:[TASK\_INDEX] ;Получить номер текущей задачи

add ds:[TASK\_INDEX],2

movzx eax,ds:[TASK\_LIST+eax] ;Получить дескриптор следующей задачи

or eax,eax

jnz TIMER\_NEXT\_TASK ;Перейти к следующей задаче

movzx eax,ds:[TASK\_LIST]

mov ds:[TASK\_INDEX],0 ;Перейти к первой задаче

TIMER\_NEXT\_TASK:

mov word ptr ds:[TASK\_ADDR+4],ax ;Записать в поле дескриптор задачи

jmp fword ptr ds:[TASK\_ADDR] ;Переключиться на новую задачу

mov al,20h

out 20h,al

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

mov ax,DATA\_DESC

mov ds,ax

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

cmp al,81

je KEYBOARD\_EXIT

cmp al,11 ;Если клавишу отпустили

jae KEY\_UP

mov ds:[KEY\_PRESSED],al

KEY\_UP:

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 ds:[TIME],0 ;Установить оставшееся время в 0 - выход будет по прерыванию таймера

KEYBOARD\_RETURN:

mov al,20h

out 20h,al ;Отпарвка сигнала контроллеру прерываний

popad ;Восстановить значения регистров

pop es

pop ds

iretd ;Выход из прерывания

1. **Структура файла STRUCT.ASM**

KEYBOARD\_HANDLER endp

SIZE\_TSS EQU 7Eh

ACS\_PRESENT EQU 10000000B ;PXXXXXXX - бит присутствия, сегмент присутствует в оперативной памяти

ACS\_CSEG EQU 00011000B ;XXXXIXXX - тип сегмента, для данных = 0, для кода 1

ACS\_DSEG EQU 00010000B ;XXXSXXXX - бит сегмента, данный объект сегмент(системные объекты могут быть не сегменты)

ACS\_READ EQU 00000010B ;XXXXXXRX - бит чтения, возможность чтения из другого сегмента

ACS\_WRITE EQU 00000010B ;XXXXXXWX - бит записи, для сегмента данных разершает запись

ACS\_CODE = ACS\_PRESENT or ACS\_CSEG ;AR сегмента кода

ACS\_DATA = ACS\_PRESENT or ACS\_DSEG or ACS\_WRITE;AR сегмента данных

ACS\_STACK= ACS\_PRESENT or ACS\_DSEG or ACS\_WRITE;AR сегмента стека

ACS\_INT\_GATE EQU 00001110B

ACS\_TRAP\_GATE EQU 00001111B ;XXXXSICR - сегмент, подчиненный сегмент кода, доступен для чтения

ACS\_IDT EQU ACS\_DATA ;AR таблицы IDT

ACS\_INT EQU ACS\_PRESENT or ACS\_INT\_GATE

ACS\_TRAP EQU ACS\_PRESENT or ACS\_TRAP\_GATE

ACS\_TSS EQU ACS\_PRESENT OR 00001001B

ACS\_DPL\_3 EQU 01100000B

;Структуры данных

S\_DESC struc ;Структура сегментного дескриптора

LIMIT dw 0 ;Лимит сегмента (15:00)

BASE\_L dw 0 ;Адрес базы, младшая часть (15:0)

BASE\_M db 0 ;Адрес базы, средняя часть (23:16)

ACCESS db 0 ;Байт доступа

ATTRIBS db 0 ;Лимит сегмента (19:16) 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 ;Структура IDTR

LIMIT dw 0

IDT\_L dw 0 ;Смещение биты (0-15)

IDT\_H dw 0 ;Смещение биты (31-16)

R\_IDTR ends

S\_TSS STRUC

LINK dw 0, 0 ;Дескриптор предыдущей задачи

ESP0 dd 0 ;Указатели на стек различных уровней превилегий

SS0 dw 0, 0 ;Необходимы для разделения стека на разные уровни доступа

ESP1 dd 0 ;НЕЛЬЗЯ ИЗМЕНИТЬ(!)

SS1 dw 0, 0

ESP2 dd 0

SS2 dw 0, 0

R\_CR3 dd 0 ;Регистр CR3 - параметры страницы задачи

R\_EIP dd 0

R\_EFLAGS dd 0

R\_EAX dd 0 ;Регистры общего назначения

R\_ECX dd 0 ;

R\_EDX dd 0 ;

R\_EBX dd 0 ;

R\_ESP dd 0 ;

R\_EBP dd 0 ;

R\_ESI dd 0 ;

R\_EDI dd 0 ;

R\_ES dw 0, 0 ;Сегментные регистры

R\_CS dw 0, 0 ;

R\_SS dw 0, 0 ;

R\_DS dw 0, 0 ;

R\_FS dw 0, 0 ;

R\_GS dw 0, 0 ;

R\_LDTR dw 0, 0 ;Дескриптор LDT

TRACE dw 0 ;Флаг трассировки сброшен - не надо генерировать прерывание 1

IO\_MAP\_ADDR dw 68h ;Адрес карты ввода/вывода - доступ к портам

IO\_MAP db 14h dup (0) Все биты сброшены - доступны все порты

S\_TSS ENDS