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»

Выполнил: Проверил:

Студент группы 250502 Преподаватель

Бекетова М. А. Одинец Д.Н.

Минск, 2024

1. **Постановка задачи**

Написать программу, которая выполняет следующие действия:

Переход из реального режима в защищенный.

Перехватывает аппаратное прерывание от клавиатуры и заданное

аппаратное прерывание, в обработчике которого выполняет определенные действия. Прерывание от клавиатуры реализуют все, номер аппаратного прерывания задается по вариантам (два варианта).

Прерывание от таймера.

Прерывание от часов реального времени.

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.

1. **Листинг программы**

Листинг программы, реализующей все поставленные задачи:

.386P

.MODEL LARGE

CODE\_RM segment para use16

CODE\_RM\_BEGIN = $

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

START:

mov ax,DATA *; Загрузка сегмента данных в регистр DS*

mov ds,ax *; Установка сегмента данных*

mov es,ax *; Установка сегмента данных*

lea dx,MSG\_ENTER *; Загрузка адреса сообщения в DX*

mov ah,9h *; Функция DOS для вывода строки*

int 21h *; Вызов прерывания DOS*

call GET\_USER\_INPUT *; Вызов процедуры для получения ввода пользователя*

mov ds:[TIME], al *; Сохранение введенного времени в переменную TIME*

call FILL\_CR0\_BUFFER\_REAL\_MODE *; Вызов процедуры для заполнения буфера текущим значением CR0 в реальном режиме*

lea dx, BUFFER\_CR\_0\_RM *; Загрузка адреса буфера в DX*

mov ah, 9h *; Функция DOS для вывода строки*

int 21h *; Вызов прерывания DOS*

lea dx,MSG\_HELLO *; Загрузка адреса сообщения в DX*

mov ah,9h *; Функция DOS для вывода строки*

int 21h *; Вызов прерывания DOS*

mov ah,7h *; Функция DOS для ожидания клавиши*

int 21h *; Вызов прерывания DOS*

*; Функция UPDATE\_RTC\_DM\_BIT включает бит DM (Daylight Saving Time) в регистре RTC (Real Time Clock),*

*; что позволяет активировать режим перехода на летнее время.*

UPDATE\_RTC\_DM\_BIT:

mov al,0Bh *; Загрузка значения 0Bh в AL для доступа к регистрам RTC*

out 70h,al *; Отправка значения AL в порт 70h для выбора регистра RTC*

in al,71h *; Чтение значения из порта 71h в AL*

or al,00000100b *; Установка бита 2 в AL (бит DM)*

out 71h,al *; Отправка обновленного значения AL обратно в порт 71h*

*; Включает бит A20, что позволяет адресной шине доступ к 1 МБ памяти.*

SET\_A20\_ENABLE:

in al,92h *; Чтение значения из порта 92h в AL*

or al,2 *; Установка бита 1 в AL (бит A20)*

out 92h,al *; Отправка обновленного значения AL обратно в порт 92h*

*; Сохраняет текущие маски прерываний для восстановления после перехода в защищенный режим.*

STORE\_INTERRUPT\_MASK:

in al,21h *; Чтение значения из порта 21h в AL (маска прерываний)*

mov INT\_MASK\_M,al *; Сохранение маски прерываний в переменную INT\_MASK\_M*

in al,0A1h *; Чтение значения из порта 0A1h в AL (маска прерываний)*

mov INT\_MASK\_S,al *; Сохранение маски прерываний в переменную INT\_MASK\_S*

*; Отключает прерывания и устанавливает бит NMI, что предотвращает прерывания от аппаратных устройств.*

DISABLE\_NMI\_AND\_INTERRUPTS:

cli *; Отключение прерываний*

in al,70h *; Чтение значения из порта 70h в AL (маска прерываний)*

or al,10000000b *; Установка бита 7 в AL (бит NMI)*

out 70h,al *; Отправка обновленного значения AL обратно в порт 70h*

nop *; Нулевая инструкция, не выполняющая никаких действий*

*; Устанавливает указатель на таблицу глобального дескриптора (GDT) и обновляет записи в GDT.*

SET\_GDT\_POINTER:

mov ax,DATA *; Загрузка сегмента данных в AX*

mov dl,ah *; Копирование старшего байта AX в DL*

xor dh,dh *; Очистка DH*

shl ax,4 *; Сдвиг AX влево на 4 бита*

shr dx,4 *; Сдвиг DX вправо на 4 бита*

mov si,ax *; Копирование AX в SI*

mov di,dx *; Копирование DX в DI*

*; Обновляет запись GDT для кода в реальном режиме, данных, стека и кода в защищенном режиме.*

UPDATE\_GDT\_ENTRY:

lea bx,GDT\_GDT *; Загрузка адреса GDT в BX*

mov ax,si *; Копирование SI в AX*

mov dx,di *; Копирование DI в DX*

add ax,offset GDT *; Добавление смещения GDT к AX*

adc dx,0 *; Добавление переноса к DX*

mov [bx][S\_DESC.BASE\_L],ax *; Сохранение AX в базовом адресе GDT*

mov [bx][S\_DESC.BASE\_M],dl *; Сохранение DL в базовом адресе GDT*

mov [bx][S\_DESC.BASE\_H],dh *; Сохранение DH в базовом адресе GDT*

*; Обновляет запись GDT для кода в реальном режиме.*

UPDATE\_CODE\_RM\_SEGMENT:

lea bx,GDT\_CODE\_RM *; Загрузка адреса записи GDT для кода в реальном режиме в BX*

mov ax,cs *; Загрузка сегмента кода в AX*

xor dh,dh *; Очистка DH*

mov dl,ah *; Копирование старшего байта AX в DL*

shl ax,4 *; Сдвиг AX влево на 4 бита*

shr dx,4 *; Сдвиг DX вправо на 4 бита*

mov [bx][S\_DESC.BASE\_L],ax *; Сохранение AX в базовом адресе записи GDT*

mov [bx][S\_DESC.BASE\_M],dl *; Сохранение DL в базовом адресе записи GDT*

mov [bx][S\_DESC.BASE\_H],dh *; Сохранение DH в базовом адресе записи GDT*

*; Обновляет запись GDT для данных.*

UPDATE\_DATA\_SEGMENT:

lea bx,GDT\_DATA *; Загрузка адреса записи GDT для данных в BX*

mov ax,si *; Копирование SI в AX*

mov dx,di *; Копирование DI в DX*

mov [bx][S\_DESC.BASE\_L],ax *; Сохранение AX в базовом адресе записи GDT*

mov [bx][S\_DESC.BASE\_M],dl *; Сохранение DL в базовом адресе записи GDT*

mov [bx][S\_DESC.BASE\_H],dh *; Сохранение DH в базовом адресе записи GDT*

*; Обновляет запись GDT для стека.*

UPDATE\_STACK\_SEGMENT:

lea bx, GDT\_STACK *; Загрузка адреса записи GDT для стека в BX*

mov ax,ss *; Загрузка сегмента стека в AX*

xor dh,dh *; Очистка DH*

mov dl,ah *; Копирование старшего байта AX в DL*

shl ax,4 *; Сдвиг AX влево на 4 бита*

shr dx,4 *; Сдвиг DX вправо на 4 бита*

mov [bx][S\_DESC.BASE\_L],ax *; Сохранение AX в базовом адресе записи GDT*

mov [bx][S\_DESC.BASE\_M],dl *; Сохранение DL в базовом адресе записи GDT*

mov [bx][S\_DESC.BASE\_H],dh *; Сохранение DH в базовом адресе записи GDT*

*; Обновляет запись GDT для кода в защищенном режиме.*

UPDATE\_CODE\_PM\_SEGMENT:

lea bx,GDT\_CODE\_PM *; Загрузка адреса записи GDT для кода в защищенном режиме в BX*

mov ax,CODE\_PM *; Загрузка сегмента кода в защищенном режиме в AX*

xor dh,dh *; Очистка DH*

mov dl,ah *; Копирование старшего байта AX в DL*

shl ax,4 *; Сдвиг AX влево на 4 бита*

shr dx,4 *; Сдвиг DX вправо на 4 бита*

mov [bx][S\_DESC.BASE\_L],ax *; Сохранение AX в базовом адресе записи GDT*

mov [bx][S\_DESC.BASE\_M],dl *; Сохранение DL в базовом адресе записи GDT*

mov [bx][S\_DESC.BASE\_H],dh *; Сохранение DH в базовом адресе записи GDT*

*; Устанавливает указатель на таблицу дескрипторов прерываний (IDT) и обновляет записи в IDT.*

UPDATE\_IDT\_POINTER:

lea bx,GDT\_IDT *; Загрузка адреса записи GDT для IDT в BX*

mov ax,si *; Копирование SI в AX*

mov dx,di *; Копирование DI в DX*

add ax,OFFSET IDT *; Добавление смещения IDT к AX*

adc dx,0 *; Добавление переноса к DX*

mov [bx][S\_DESC.BASE\_L],ax *; Сохранение AX в базовом адресе записи GDT*

mov [bx][S\_DESC.BASE\_M],dl *; Сохранение DL в базовом адресе записи GDT*

mov [bx][S\_DESC.BASE\_H],dh *; Сохранение DH в базовом адресе записи GDT*

mov IDTR.IDT\_L,ax *; Сохранение AX в базовом адресе IDTR*

mov IDTR.IDT\_H,dx *; Сохранение DX в верхнем адресе IDTR*

*; Заполняет таблицу прерываний (IDT) для обработчиков исключений и прерываний.*

FILL\_IDT\_TABLE:

irpc N, 0123456789ABCDEF *; Цикл для заполнения IDT*

lea eax, EXC\_0&N *; Загрузка адреса обработчика исключений в EAX*

mov IDT\_0&N.OFFS\_L,ax *; Сохранение младшего слово OFFS\_L*

shr eax, 16 *; Сдвиг EAX вправо на 16 бит*

mov IDT\_0&N.OFFS\_H,ax *; Сохранение старшего слово OFFS\_H*

endm

irpc N, 0123456789ABCDEF

lea eax, EXC\_1&N

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

shr eax, 16

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

endm

*; Заполнение IDT для обработчиков прерываний таймера и клавиатуры*

lea eax, TIMER\_HANDLER

mov IDT\_TIMER.OFFS\_L,ax

shr eax, 16

mov IDT\_TIMER.OFFS\_H,ax

lea eax, KEYBOARD\_HANDLER

mov IDT\_KEYBOARD.OFFS\_L,ax

shr eax, 16

mov IDT\_KEYBOARD.OFFS\_H,ax

*; Заполнение IDT для обработчиков прерываний от мастер-контроллера и слейв-контроллера*

irpc N, 234567

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

lea eax,IDLE\_IRQ\_SLAVE

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

shr eax,16

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

endm

*; Загрузка GDT и IDT, а также включение защищенного режима*

lgdt fword ptr GDT\_GDT

lidt fword ptr IDTR

mov eax,cr0

or al,00000001b

mov cr0,eax

*; Переключение на защищенный режим*

SET\_CS\_OVERLOAD:

db 0eah *; Используем прерывание для переключения на защищенный режим*

dw offset SET\_SEGMENT\_REGISTERS *; Указываем адрес следующей инструкции*

dw CODE\_RM\_DESC *; Описатель сегмента кода в реальном режиме*

*; Установка сегментных регистров в защищенном режиме*

SET\_SEGMENT\_REGISTERS:

mov ax,DATA\_DESC *; Загружаем описатель сегмента данных*

mov ds,ax *; Устанавливаем DS*

mov es,ax *; Устанавливаем ES*

mov ax,STACK\_DESC *; Загружаем описатель стека*

mov ss,ax *; Устанавливаем SS*

xor ax,ax

mov fs,ax *; Очищаем FS*

mov gs,ax *; Очищаем GS*

lldt ax *; Загружаем описатель локального дескриптора таблицы (LDT)*

*; Возврат в реальный режим*

SETUP\_RETURN\_TO\_RM:

push cs *; Сохраняем CS*

push offset SWITCH\_TO\_REAL\_MODE *; Сохраняем адрес возврата*

lea edi,ENTER\_PM *; Загружаем адрес метки ENTER\_PM*

mov eax,CODE\_PM\_DESC *; Загружаем описатель сегмента кода в защищенном режиме*

push eax *; Сохраняем описатель сегмента кода*

push edi *; Сохраняем адрес метки ENTER\_PM*

*; Перенастройка контроллера прерываний для работы в защищенном режиме*

REINITIALIZE\_CONTROLLER\_FOR\_PM:

mov al,00010001b *; Настраиваем начальные значения для контроллера прерываний*

out 20h,al *; Записываем в порт 20h*

out 0A0h,al *; Записываем в порт 0A0h*

mov al,20h *; Задаем значение для порта 21h*

out 21h,al *; Записываем в порт 21h*

mov al,28h *; Задаем значение для порта 0A1h*

out 0A1h,al *; Записываем в порт 0A1h*

mov al,04h *; Задаем значение для порта 21h*

out 21h,al *; Записываем в порт 21h*

mov al,02h *; Задаем значение для порта 0A1h*

out 0A1h,al *; Записываем в порт 0A1h*

mov al,11h *; Задаем значение для порта 21h*

out 21h,al *; Записываем в порт 21h*

mov al,01h *; Задаем значение для порта 0A1h*

out 0A1h,al *; Записываем в порт 0A1h*

mov al, 0 *; Задаем значение для порта 21h*

out 21h,al *; Записываем в порт 21h*

out 0A1h,al *; Записываем в порт 0A1h*

*; Включение прерываний*

ENABLE\_INTERRUPTS\_ZERO:

in al,70h *; Читаем из порта 70h*

and al,01111111b *; Сбрасываем бит прерывания*

out 70h,al *; Записываем обратно в порт 70h*

nop *; Нейтральная инструкция*

sti *; Включаем прерывания*

*; Переключение на защищенный режим*

SWITCH\_TO\_CODE\_PM:

db 66h *; Префикс для выполнения следующей инструкции в 32-битном режиме*

retf *; Возврат из прерывания*

*; Возврат в реальный режим*

SWITCH\_TO\_REAL\_MODE:

cli *; Отключаем прерывания*

in al,70h *; Читаем из порта 70h*

or AL,10000000b *; Устанавливаем бит прерывания*

out 70h,AL *; Записываем обратно в порт 70h*

nop *; Нейтральная инструкция*

*; Перенастройка контроллера прерываний для работы в реальном режиме*

REINITIALIZE\_CONTROLLER:

mov al,00010001b *; Настраиваем начальные значения для контроллера прерываний*

out 20h,al *; Записываем в порт 20h*

out 0A0h,al *; Записываем в порт 0A0h*

mov al,8h *; Задаем значение для порта 21h*

out 21h,al *; Записываем в порт 21h*

mov al,70h *; Задаем значение для порта 0A1h*

out 0A1h,al *; Записываем в порт 0A1h*

mov al,04h *; Задаем значение для порта 21h*

out 21h,al *; Записываем в порт 21h*

mov al,02h *; Задаем значение для порта 0A1h*

out 0A1h,al *; Записываем в порт 0A1h*

mov al,11h *; Задаем значение для порта 21h*

out 21h,al *; Записываем в порт 21h*

mov al,01h *; Задаем значение для порта 0A1h*

out 0A1h,al *; Записываем в порт 0A1h*

*; Инициализация сегментов в реальном режиме*

INITIALIZE\_SEGMENTS:

mov GDT\_CODE\_RM.LIMIT,0FFFFh *; Устанавливаем предел сегмента кода в реальном режиме*

mov GDT\_DATA.LIMIT,0FFFFh *; Устанавливаем предел сегмента данных*

mov GDT\_STACK.LIMIT,0FFFFh *; Устанавливаем предел стека*

db 0EAH *; Используем прерывание для переключения на реальный режим*

dw offset CONTINUE

dw CODE\_RM\_DESC *; Описатель сегмента кода в реальном режиме*

CONTINUE:

mov ax,DATA\_DESC *; Загружаем описатель сегмента данных*

mov ds,ax *; Устанавливаем DS*

mov es,ax *; Устанавливаем ES*

mov fs,ax *; Устанавливаем FS*

mov gs,ax *; Устанавливаем GS*

mov ax,STACK\_DESC

mov ss,ax *; Устанавливаем SS*

*; Возврат в реальный режим*

ENTER\_REAL\_MODE:

mov eax,cr0 *; Читаем CR0*

and al,11111110b *; Сбрасываем бит защищенного режима*

mov cr0,eax *; Записываем обратно в CR0*

db 0EAH *; Используем прерывание для переключения на реальный режим*

dw offset CONTINUE2

dw CODE\_RM *; Описатель сегмента кода в реальном режиме*

CONTINUE2:

mov ax,STACK\_A *; Загружаем описатель стека*

mov ss,ax *; Устанавливаем SS*

mov ax,DATA *; Загружаем описатель сегмента данных*

mov ds,ax *; Устанавливаем DS*

mov es,ax *; Устанавливаем ES*

xor ax,ax *; Очищаем AX*

mov fs,ax *; Очищаем FS*

mov gs,ax *; Очищаем GS*

mov IDTR.LIMIT, 3FFH *; Устанавливаем предел таблицы дескрипторов прерываний*

mov dword ptr IDTR+2, 0

lidt fword ptr IDTR *; Загружаем таблицу дескрипторов прерываний*

*; Установка маски прерываний*

SET\_INTERRUPT\_MASK:

mov al,INT\_MASK\_M *; Загружаем маску прерываний для мастер-пин*

out 21h,al *; Записываем в порт 21h*

mov al,INT\_MASK\_S *; Загружаем маску прерываний для слейв-пин*

out 0A1h,al *; Записываем в порт 0A1h*

*; Включение прерываний*

ENABLE\_INTERRUPTS:

in al,70h *; Читаем из порта 70h*

and al,01111111b *; Сбрасываем бит прерывания*

out 70h,al *; Записываем обратно в порт 70h*

nop *; Нейтральная инструкция*

sti *; Включаем прерывания*

*; Отключение линии A20*

DISABLE\_A20\_LINE:

in al,92h *; Читаем из порта 92h*

and al,11111101b *; Сбрасываем бит A20*

out 92h, al *; Записываем обратно в порт 92h*

*; Выход из программы*

EXIT\_PROGRAM:

mov ax,3h *; Загружаем код функции для выхода из программы*

int 10H *; Выполняем прерывание 10h для восстановления видеорежима*

lea dx,MSG\_EXIT *; Загружаем адрес сообщения об выходе*

mov ah,9h *; Загружаем код функции для вывода строки*

int 21h *; Выполняем прерывание 21h для вывода сообщения*

call FILL\_CR0\_BUFFER\_REAL\_MODE *; Вызываем процедуру для заполнения буфера значением регистра CR0 в реальном режиме*

lea dx, BUFFER\_CR\_0\_RM *; Загружаем адрес буфера*

mov ah, 9h *; Загружаем код функции для вывода строки*

int 21h *; Выполняем прерывание 21h для вывода значения CR0*

mov ax,4C00h *; Загружаем код функции для завершения программы*

int 21H *; Выполняем прерывание 21h для завершения программы*

*; Получение ввода пользователя*

GET\_USER\_INPUT proc near

mov ah,0ah *; Загружаем код функции для чтения строки*

xor di,di *; Очищаем DI*

mov dx,offset ds:[INPUT\_TIME] *; Загружаем адрес буфера для ввода*

int 21h *; Выполняем прерывание 21h для чтения строки*

mov dl,0ah *; Загружаем символ новой строки*

mov ah,02 *; Загружаем код функции для вывода символа*

int 21h *; Выполняем прерывание 21h для вывода символа*

mov si,offset INPUT\_TIME+2 *; Загружаем адрес начала строки*

cmp byte ptr [si],"-" *; Сравниваем первый символ со знаком минуса*

jnz ii1 *; Если не равен, переходим к следующей инструкции*

mov di,1 *; Устанавливаем флаг отрицательного числа*

inc si *; Сдвигаем указатель на следующий символ*

II1:

xor ax,ax *; Очищаем AX*

mov bx,10 *; Загружаем основание системы счисления*

II2:

mov cl,[si] *; Загружаем текущий символ*

cmp cl,0dh *; Сравниваем с символом новой строки*

jz ii3 *; Если равен, переходим к следующей инструкции*

cmp cl,'0' *; Сравниваем с символом* '0'

jl er *; Если меньше, переходим к ошибке*

cmp cl,'9' *; Сравниваем с символом* '9'

ja er *; Если больше, переходим к ошибке*

sub cl,'0' *; Преобразуем символ в число*

mul bx *; Умножаем текущее значение на основание системы счисления*

add ax,cx *; Добавляем текущее значение к результату*

inc si *; Сдвигаем указатель на следующий символ*

jmp ii2 *; Повторяем цикл*

ER:

mov dx, offset MSG\_ERROR *; Загружаем адрес сообщения об ошибке*

mov ah,09 *; Загружаем код функции для вывода строки*

int 21h *; Выполняем прерывание 21h для вывода сообщения*

int 20h *; Выполняем прерывание 20h для завершения программы*

II3:

ret *; Возвращаемся из процедуры*

GET\_USER\_INPUT endp

*; Процедура для заполнения буфера значением регистра CR0 в реальном режиме*

FILL\_CR0\_BUFFER\_REAL\_MODE proc near

push eax *; Сохраняем регистр EAX*

push esi *; Сохраняем регистр ESI*

push dx *; Сохраняем регистр DX*

mov eax, cr0 *; Читаем значение регистра CR0*

xor dx, dx *; Очищаем DX*

mov cx, 32 *; Загружаем счетчик для цикла*

lea esi, BUFFER\_CR\_0\_RM *; Загружаем адрес буфера*

fill\_cr\_0\_loop\_rm:

mov dl, al *; Копируем младший байт AL в DL*

shl dl, 7 *; Сдвигаем биты влево*

shr dl, 7 *; Сдвигаем биты вправо*

shr eax, 1 *; Сдвигаем биты вправо*

add dl, 48 *; Преобразуем число в символ ASCII*

mov [esi], dl *; Записываем символ в буфер*

inc esi *; Сдвигаем указатель на следующий символ*

xor dl, dl *; Очищаем DX*

loop fill\_cr\_0\_loop\_rm *; Повторяем цикл*

pop dx *; Восстанавливаем регистр DX*

pop esi *; Восстанавливаем регистр ESI*

pop eax *; Восстанавливаем регистр EAX*

ret *; Возвращаемся из процедуры*

FILL\_CR0\_BUFFER\_REAL\_MODE endp

*; Определение размеров сегментов кода и данных в реальном режиме*

CODE\_PM segment para use32

CODE\_PM\_BEGIN = $

assume cs:CODE\_PM,ds:DATA,es:DATA

ENTER\_PM:

call CLRSCR *; Очистка экрана*

xor edi,edi *; Инициализация регистра EDI нулями*

lea esi,MSG\_HELLO\_PM *; Загрузка адреса сообщения о переходе в защищенный режим*

call BUFFER\_OUTPUT *; Вывод сообщения на экран*

add edi,160 *; Перемещение указателя на следующую строку*

lea esi,MSG\_KEYBOARD

call BUFFER\_OUTPUT *; Вывод сообщения о сканировании клавиатуры*

mov edi,320

lea esi,MSG\_TIME

call BUFFER\_OUTPUT *; Вывод сообщения о времени*

mov edi,480

lea esi,MSG\_COUNT

call BUFFER\_OUTPUT *; Вывод сообщения о количестве прерываний*

call FILL\_CR\_0\_BUFFER *; Заполнение буфера значением регистра CR0*

mov edi, 640

lea esi, BUFFER\_CR\_0

call BUFFER\_OUTPUT *; Вывод значения регистра CR0 на экран*

mov DS:[COUNT],0 *; Инициализация счетчика прерываний нулем*

WAITING\_ESC:

jmp WAITING\_ESC *; Бесконечный цикл ожидания нажатия клавиши ESC*

EXIT\_PM:

db 66H *; Команда для перехода в реальный режим*

retf *; Возврат из прерывания*

EXIT\_FROM\_INTERRUPT:

popad *; Восстановление регистров*

pop es *; Восстановление сегмента ES*

pop ds *; Восстановление сегмента DS*

pop eax *; Восстановление регистра EAX*

pop eax

pop eax

sti *; Включение прерываний*

db 66H *; Команда для перехода в реальный режим*

retf *; Возврат из прерывания*

*; Преобразование 16-битного числа в десятичную строку*

WORD\_TO\_DEC proc near

pushad *; Сохраняем все регистры*

movzx eax,ax *; Расширяем 16-битное число до 32-битного*

xor cx,cx *; Обнуляем CX для подсчета цифр*

mov bx,10 *; Устанавливаем делитель для преобразования в десятичную систему*

LOOP1:

xor dx,dx *; Обнуляем DX для деления*

div bx *; Делим AX на 10, результат в AL, остаток в AH*

add dl,'0' *; Преобразуем остаток в символ ASCII*

push dx *; Сохраняем символ в стеке*

inc cx *; Увеличиваем счетчик цифр*

test ax,ax *; Проверяем, не равно ли число нулю*

jnz LOOP1 *; Если нет, продолжаем цикл*

LOOP2:

pop dx *; Извлекаем символ из стека*

mov [di],dl *; Записываем символ в буфер*

inc di *; Увеличиваем указатель на буфер*

loop LOOP2 *; Повторяем, пока не обработаем все цифры*

popad *; Восстанавливаем все регистры*

ret *; Возвращаемся из процедуры*

WORD\_TO\_DEC endp

*; Заполнение буфера значением регистра CR0*

FILL\_CR\_0\_BUFFER proc near

push eax *; Сохраняем регистр EAX*

push esi *; Сохраняем регистр ESI*

push dx *; Сохраняем регистр DX*

mov eax, cr0 *; Загружаем значение регистра CR0 в EAX*

xor dx, dx *; Обнуляем DX*

mov cx, 32 *; Устанавливаем счетчик для цикла*

lea esi, BUFFER\_CR\_0 *; Загружаем адрес буфера*

fill\_cr\_0\_loop:

mov dl, al *; Перемещаем младший байт EAX в DL*

shl dl, 7 *; Сдвигаем биты влево на 7 позиций*

shr dl, 7 *; Сдвигаем биты вправо на 7 позиций (возвращаем исходное значение)*

shr eax, 1 *; Сдвигаем биты вправо на 1 позицию*

add dl, 48 *; Преобразуем числовое значение в символ ASCII*

mov [esi], dl *; Записываем символ в буфер*

inc esi *; Увеличиваем указатель на буфер*

xor dl, dl *; Обнуляем DL для следующей итерации*

loop fill\_cr\_0\_loop *; Повторяем цикл*

pop dx *; Восстанавливаем регистр DX*

pop esi *; Восстанавливаем регистр ESI*

pop eax *; Восстанавливаем регистр EAX*

ret *; Возвращаемся из процедуры*

FILL\_CR\_0\_BUFFER endp

*; Процедура DIGIT\_TO\_HEX преобразует числовой символ в его шестнадцатеричное представление.*

DIGIT\_TO\_HEX proc near

add al,'0' *; Добавляем к содержимому регистра AL значение символа* '0'*. Это преобразует числовое значение в ASCII-код символа* '0'*.*

cmp al,'9' *; Сравниваем значение в AL с символом* '9'*. Если AL меньше или равно* '9'*, это означает, что мы работаем с числом от 0 до 9.*

jle DTH\_END *; Если AL меньше или равно* '9'*, переходим к метке DTH\_END. Это означает, что мы не нуждаемся в дополнительной обработке, так как символ уже представляет собой число от 0 до 9.*

add al,7 *; Если AL больше* '9'*, это означает, что мы работаем с символом, который нужно преобразовать в шестнадцатеричный символ. Добавляем к AL значение 7, чтобы преобразовать числовой символ в соответствующий шестнадцатеричный символ.*

DTH\_END:

ret *; Возвращаемся из процедуры. В этом моменте AL содержит либо исходный числовой символ (если он был от 0 до 9), либо его шестнадцатеричное представление.*

DIGIT\_TO\_HEX endp

*; Преобразование байта в шестнадцатеричную строку*

BYTE\_TO\_HEX proc near

push ax *; Сохраняем регистр AX*

mov ah,al *; Копируем младший байт в старший байт*

shr al,4 *; Сдвигаем младший байт вправо на 4 бита*

call DIGIT\_TO\_HEX *; Преобразуем старший байт в символ ASCII*

mov [di],al *; Записываем символ в буфер*

inc di *; Увеличиваем указатель на буфер*

mov al,ah *; Копируем старший байт обратно в младший*

and al,0Fh *; Оставляем только младшие 4 бита*

call DIGIT\_TO\_HEX *; Преобразуем младший байт в символ ASCII*

mov [di],al *; Записываем символ в буфер*

inc di *; Увеличиваем указатель на буфер*

pop ax *; Восстанавливаем регистр AX*

ret *; Возвращаемся из процедуры*

BYTE\_TO\_HEX endp

*; Обработчики прерываний для исключений*

M = 0

IRPC N, 0123456789ABCDEF

EXC\_0&N label word

cli *; Отключаем прерывания*

jmp EXC\_HANDLER *; Переходим к обработчику исключений*

endm

M = 010H

IRPC N, 0123456789ABCDEF

EXC\_1&N label word

cli *; Отключаем прерывания*

jmp EXC\_HANDLER *; Переходим к обработчику исключений*

endm

*; Обработчик исключений*

EXC\_HANDLER proc near

call CLRSCR *; Очищаем экран*

lea esi, MSG\_EXC *; Загружаем адрес сообщения об исключении*

mov edi, 40\*2 *; Устанавливаем адрес в буфере*

call BUFFER\_OUTPUT *; Выводим сообщение на экран*

pop eax *; Восстанавливаем регистр EAX*

pop eax

pop eax

sti *; Включаем прерывания*

db 66H *; Команда для перехода в реальный режим*

retf *; Возвращаемся из прерывания*

EXC\_HANDLER ENDP

*; Обработчик прерывания от мастер-PIC*

IDLE\_IRQ\_MASTER proc near

push eax *; Сохраняем регистр EAX*

mov al,20h *; Команда EOI для мастер-PIC*

out 20h,al *; Отправляем команду*

pop eax *; Восстанавливаем регистр EAX*

iretd *; Возвращаемся из прерывания*

IDLE\_IRQ\_MASTER endp

*; Обработчик прерывания от слейв-PIC*

IDLE\_IRQ\_SLAVE proc near

push eax *; Сохраняем регистр EAX*

mov al,20h *; Команда EOI для мастер-PIC*

out 20h,al *; Отправляем команду*

out 0A0h,al *; Команда EOI для слейв-PIC*

pop eax *; Восстанавливаем регистр EAX*

iretd *; Возвращаемся из прерывания*

IDLE\_IRQ\_SLAVE endp

*; Обработчик прерывания от таймера*

TIMER\_HANDLER proc near

push ds *; Сохраняем сегментные регистры*

push es *; Сохраняем сегментные регистры*

pushad *; Сохраняем все регистры*

mov ax,DATA\_DESC *; Загружаем селектор сегмента данных*

mov ds,ax *; Устанавливаем сегмент данных*

inc ds:[COUNT] *; Увеличиваем счетчик прерываний*

lea edi,ds:[BUFFER\_COUNT] *; Загружаем адрес буфера для счетчика*

mov ax,ds:[COUNT] *; Загружаем значение счетчика*

call WORD\_TO\_DEC *; Преобразуем счетчик в десятичную строку*

mov edi,538 *; Устанавливаем адрес в буфере*

lea esi,BUFFER\_COUNT *; Загружаем адрес буфера счетчика*

call BUFFER\_OUTPUT *; Выводим счетчик на экран*

TIMER\_HANDLER endp

*; Процедура SHOW\_TIMER отвечает за чтение и отображение времени с таймера, а также за обработку истечения времени.*

SHOW\_TIMER:

mov al,0h *; Сбрасываем AL до 0, чтобы начать чтение с начала регистров таймера.*

out 70h,al *; Отправляем команду на чтение регистра таймера через порт 70h.*

in al,71h *; Считываем значение из регистра таймера через порт 71h.*

cmp al,ds:[SECOND] *; Сравниваем считанное значение с сохраненным значением секунды.*

je SKIP\_SECOND *; Если значения совпадают, переходим к SKIP\_SECOND, чтобы пропустить обновление времени.*

mov ds:[SECOND],al *; Сохраняем считанное значение как текущую секунду.*

mov al,ds:[TIME] *; Загружаем значение оставшегося времени.*

cmp ds:[TIME],0 *; Сравниваем оставшееся время с 0.*

je DISABLE\_PM *; Если время истекло, переходим к DISABLE\_PM, чтобы отключить защищенный режим.*

xor ah,ah *; Очищаем AH, чтобы использовать только AL для вызова процедуры преобразования.*

lea edi,ds:[BUFFER\_TIME] *; Загружаем адрес буфера для временного хранения преобразованного времени.*

call WORD\_TO\_DEC *; Вызываем процедуру преобразования слов в десятичные числа.*

mov edi,356 *; Задаем адрес для вывода на экран.*

lea esi,BUFFER\_TIME *; Загружаем адрес буфера с преобразованным временем.*

call BUFFER\_OUTPUT *; Вызываем процедуру вывода содержимого буфера на экран.*

dec ds:[TIME] *; Уменьшаем оставшееся время на 1.*

lea esi,BUFFER\_TIME *; Загружаем адрес буфера с преобразованным временем.*

call BUFFER\_CLEAR *; Вызываем процедуру очистки буфера.*

jmp SKIP\_SECOND *; Переходим к SKIP\_SECOND, чтобы пропустить обновление времени.*

*; Процедура DISABLE\_PM отключает прерывания таймера и переключает систему на реальный режим.*

DISABLE\_PM:

mov al,20h *; Загружаем значение 20h в AL, чтобы отключить прерывания таймера.*

out 20h,al *; Отправляем команду на отключение прерываний таймера через порт 20h.*

db 0eah *; Используем прерывание для переключения на реальный режим.*

dd OFFSET EXIT\_FROM\_INTERRUPT *; Указываем адрес возврата после прерывания.*

dw CODE\_PM\_DESC *; Указываем описатель сегмента кода в защищенном режиме.*

*; Процедура SKIP\_SECOND отключает прерывания таймера и восстанавливает регистры, используемые в прерывании.*

SKIP\_SECOND:

mov al,20h *; Загружаем значение 20h в AL, чтобы отключить прерывания таймера.*

out 20h,al *; Отправляем команду на отключение прерываний таймера через порт 20h.*

popad *; Восстанавливаем регистры общего назначения.*

pop es *; Восстанавливаем регистр сегмента ES.*

pop ds *; Восстанавливаем регистр сегмента DS.*

iretd *; Возвращаемся из прерывания.*

TIMER\_HANDLER endp

*; Обработчик прерывания от клавиатуры*

KEYBOARD\_HANDLER proc near

push ds *; Сохраняем сегментные регистры*

push es *; Сохраняем сегментные регистры*

pushad *; Сохраняем все регистры*

in al,60h *; Читаем скан-код клавиши*

cmp al,1 *; Проверяем, не нажата ли клавиша ESC*

je KEYBOARD\_EXIT *; Если нажата, переходим к выходу*

mov ds:[KEY\_SCAN\_CODE],al *; Сохраняем скан-код*

lea edi,ds:[BUFFER\_SCAN\_CODE] *; Загружаем адрес буфера для скан-кода*

mov al,ds:[KEY\_SCAN\_CODE] *; Загружаем скан-код*

xor ah,ah

call BYTE\_TO\_HEX *; Преобразуем скан-код в шестнадцатеричную строку*

mov edi,200 *; Устанавливаем адрес в буфере для скан-кода*

lea esi,BUFFER\_SCAN\_CODE *; Загружаем адрес буфера скан-кода*

call BUFFER\_OUTPUT *; Выводим скан-код на экран*

jmp KEYBOARD\_RETURN *; Переходим к возврату из обработчика*

*; Процедура KEYBOARD\_EXIT отвечает за отправку команды EOI (End Of Interrupt) для мастер-PIC,*

*; что сигнализирует о завершении обработки прерывания клавиатуры. Затем происходит переход в защищенный режим.*

KEYBOARD\_EXIT:

mov al,20h *; Команда EOI для мастер-PIC*

out 20h,al *; Отправляем команду*

db 0eah *; Команда для перехода в защищенный режим*

dd OFFSET EXIT\_FROM\_INTERRUPT *; Адрес возврата после перехода*

dw CODE\_PM\_DESC *; Селектор сегмента кода в защищенном режиме*

*; Процедура KEYBOARD\_RETURN восстанавливает состояние регистров после обработки прерывания клавиатуры*

*; и возвращает управление из прерывания.*

KEYBOARD\_RETURN:

mov al,20h *; Команда EOI для мастер-PIC*

out 20h,al *; Отправляем команду*

popad *; Восстанавливаем все регистры*

pop es *; Восстанавливаем сегментные регистры*

pop ds *; Восстанавливаем сегментные регистры*

iretd *; Возвращаемся из прерывания*

KEYBOARD\_HANDLER endp

*; Очистка экрана*

CLRSCR proc near

push es *; Сохраняем сегментные регистры*

pushad *; Сохраняем все регистры*

mov ax,TEXT\_DESC *; Загружаем селектор сегмента текста*

mov es,ax *; Устанавливаем сегмент текста*

xor edi,edi *; Обнуляем указатель на буфер*

mov ecx,80\*25 *; Устанавливаем счетчик для очистки экрана*

mov ax,700h *; Загружаем символ пробела в AX*

rep stosw *; Заполняем экран символами пробела*

popad *; Восстанавливаем все регистры*

pop es *; Восстанавливаем сегментные регистры*

ret *; Возвращаемся из процедуры*

CLRSCR endp

*; Очистка буфера*

BUFFER\_CLEAR proc near

mov al,' ' *; Загружаем символ пробела в AL*

mov [esi+0],al *; Записываем символ в буфер*

mov [esi+1],al *; Записываем символ в буфер*

mov [esi+2],al *; Записываем символ в буфер*

mov [esi+3],al *; Записываем символ в буфер*

mov [esi+4],al *; Записываем символ в буфер*

mov [esi+5],al *; Записываем символ в буфер*

mov [esi+6],al *; Записываем символ в буфер*

mov [esi+7],al *; Записываем символ в буфер*

ret *; Возвращаемся из процедуры*

BUFFER\_CLEAR endp

*; Вывод буфера на экран*

BUFFER\_OUTPUT proc near

push es *; Сохраняем сегментные регистры*

PUSHAD *; Сохраняем все регистры*

mov ax,TEXT\_DESC *; Загружаем селектор сегмента текста*

mov es,ax *; Устанавливаем сегмент текста*

OUTPUT\_LOOP:

lodsb *; Загружаем байт из буфера в AL*

or al,al *; Проверяем, не равен ли байт нулю*

jz OUTPUT\_EXIT *; Если равен, выходим из цикла*

stosb *; Записываем байт в экран*

inc edi *; Увеличиваем указатель на экран*

jmp OUTPUT\_LOOP *; Повторяем цикл*

OUTPUT\_EXIT:

popad *; Восстанавливаем все регистры*

pop es *; Восстанавливаем сегментные регистры*

ret *; Возвращаемся из процедуры*

BUFFER\_OUTPUT ENDP

*; Определение размеров и адресов сегментов*

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

CODE\_PM ENDS

DATA segment para use16

DATA\_BEGIN = $

*; Определение структур и макросов для описания сегментов*

S\_DESC struc

LIMIT dw 0

BASE\_L dw 0

BASE\_M db 0

ACCESS db 0

ATTRIBS db 0

BASE\_H db 0

S\_DESC ends

I\_DESC struc

OFFS\_L dw 0

SEL dw 0

PARAM\_CNT db 0

ACCESS db 0

OFFS\_H dw 0

I\_DESC ends

R\_IDTR struc

LIMIT dw 0

IDT\_L dw 0

IDT\_H dw 0

R\_IDTR ends

*; Определение начала и размера глобальной дескрипторной таблицы (GDT)*

GDT\_BEGIN = $

GDT label word

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)

*; Определение селекторов для доступа к сегментам*

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)

*; Определение и инициализация дескриптора прерываний (IDTR)*

IDTR R\_IDTR <SIZE\_IDT,0,0>

IDT label word

IDT\_BEGIN = $

IRPC N, 0123456789ABCDEF

IDT\_0&N I\_DESC <0, CODE\_PM\_DESC,0,10001111b,0>

ENDM

IRPC N, 0123456789ABCDEF

IDT\_1&N I\_DESC <0, CODE\_PM\_DESC, 0, 10001111b, 0>

ENDM

IDT\_TIMER I\_DESC <0,CODE\_PM\_DESC,0,10001110b,0>

IDT\_KEYBOARD I\_DESC <0,CODE\_PM\_DESC,0,10001110b,0>

IRPC N, 23456789ABCDEF

IDT\_2&N I\_DESC <0, CODE\_PM\_DESC, 0, 10001110b, 0>

ENDM

SIZE\_IDT = ($ - IDT\_BEGIN)

*; Определение сообщений и переменных для вывода на экран*

MSG\_HELLO db "Press any key to switch to Protected Mode",13,10,"$"

MSG\_HELLO\_PM db "Now in Protected Mode. Press ESC or wait for the timer to end to return to Real Mode",0

MSG\_EXIT db "Returning to Real Mode",13,10,"$"

MSG\_KEYBOARD db "Keyboard scan code:",0

MSG\_TIME db "Time remaining in Protected Mode: XXXXXXX seconds",0

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

MSG\_EXC db "Exception occurred: XX",0

MSG\_ENTER db "Enter the time in seconds to stay in Protected Mode: $"

MSG\_ERROR db "Invalid input. Please enter a number.$"

HEX\_TAB db "0123456789ABCDEF"

ESP32 dd 1 dup(?)

INT\_MASK\_M db 1 dup(?)

INT\_MASK\_S db 1 dup(?)

KEY\_SCAN\_CODE db 1 dup(?)

SECOND db 1 dup(?)

TIME db 1 dup(10)

COUNT dw 1 dup(0)

BUFFER\_COUNT 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)

INPUT\_TIME db 6,7 dup(?)

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

db 1 dup(0)

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

SIZE\_DATA = ($ - DATA\_BEGIN)

DATA ends

STACK\_A segment para stack

db 1000h dup(?)

STACK\_A ends

end START

1. **Тестирование программы**