БГУИР

Кафедра ЭВМ

Отчет по лабораторной работе

Тема: "Защищённый режим процессора"

Выполнил:

Студент группы 750501 Брюшков М.И.

Проверил:

к.т.н., доцент кафедры ЭВМ Одинец Д.Н.

Минск 2019

***Постановка задачи:***

Создать консольное приложение, в котором пользователь с клавиатуры вводит некоторое количество секунду. После этого осуществить переход в защищённый режим. В защищённом режиме выводить на экран оставшееся время. Вернуться в реальный режим по истечении указанного времени.

***Алгоритм решение задачи и краткие теоретические сведения, необходимые для решения:***

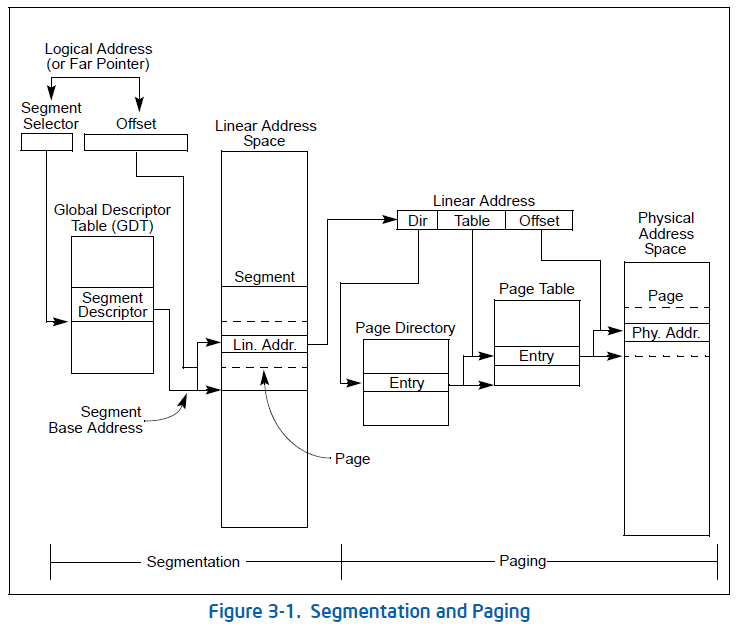
В защищённом режиме программа может записывать данные только в те области памяти, которые выделены ей операционной системой. Это сильно повышает надёжность работы мультизадачных и, в частности, мультипользовательских операционных систем. Кроме того, реально только механизм страничной виртуальной памяти может обеспечить прикладные программы относительно быстрой оперативной памятью, размер которой больше размера физической памяти, установленной в компьютере.

Перечислим несколько преимуществ, которые получает программа при работе в защищенном режиме процессора:

* возможность непосредственной адресации памяти за пределами первого мегабайт;
* для процессоров i80386 и i80486 реализован механизм страничной виртуальной памяти, позволяющий программам работать с памятью, размер которой может быть много больше физической оперативной памяти, установленной в компьютере;
* аппаратная поддержка мультизадачности позволяет создавать на основе процессоров, работающих в защищённом режиме высокопроизводительные мультизадачные и мультипользовательские системы;
* эффективная работа нескольких программ, составленных для MS-DOS, основанная на использовании виртуального режима работы процессора.

**Адресация памяти в защищённом режиме:**

Процессор использует два этапа преобразования адресов для получения физического адреса: трансляция логического адреса и «пэйджинг» линейного адресного пространства.



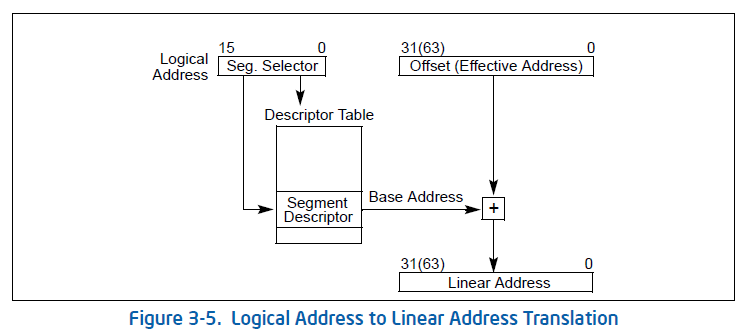
Логический адрес состоит из 16-битного селектора сегмента и 32-битного смещения. Segment selector определяет сегмент, в котором находится байт, а смещение указывает местоположение байта в сегменте относительно базового адреса сегмента. Процессор переводит каждый логический адрес в линейный адрес. Линейный адрес – это 32-битный адрес в линейном адресном пространстве процессора. Линейное адресное пространство является несегментированным.

Чтобы преобразовать логический адрес в линейный адрес, процессор делает следующее:

1. Используя значение селектора, находит нужный дескриптор в G/LDT

2. Исследует дескриптор сегмента, чтобы проверить права доступа и диапазон сегмента, чтобы убедиться, что сегмент доступен и что смещение находится в пределах сегмента.

3. Добавляет базовый адрес сегмента из дескриптора сегмента к смещению, чтобы сформировать линейный адрес.



Если «пейджинг» не используется, процессор отображает линейный адрес непосредственно на физический адрес (то есть линейный адрес выходит на адресную шину процессора). Если линейное адресное пространство разбито на страницы, второй уровень преобразования адресов используется для преобразования линейного адреса в физический адрес.

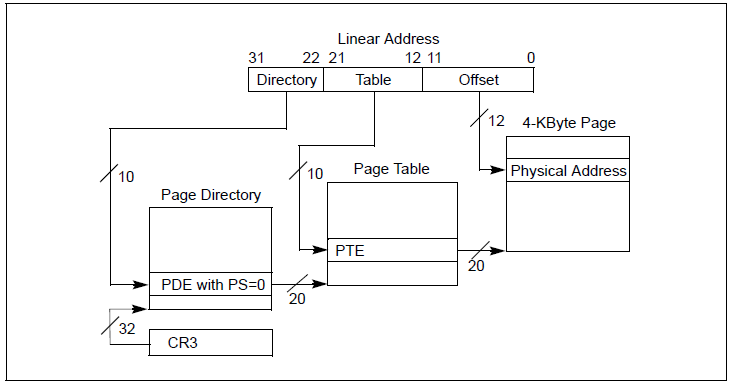
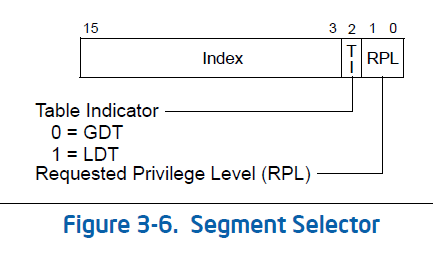


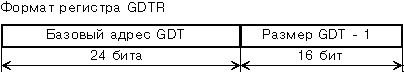
Рисунок 1.1 - Преобразование линейного адреса в физический



Поле RPL является запрошенном программой уровнем привилегий. Поле TI (Table Indicator) состоит из одного бита. Если бит равен нулю, используется глобальная таблица дескрипторов (GTD), в противном случае – локальная таблица дескрипторов (LDT).

Таблица дескрипторов - это просто таблица преобразования адресов, содержащая базовые 24-разрядные физические адреса сегментов и некоторую другую информацию. То есть каждый элемент таблицы дескрипторов (дескриптор) содержит 24-разрядный базовый адрес сегмента и другую информацию, описывающую сегмент.

Регистр GDTR, указывающий расположение в физической памяти и размер глобальной таблицы дескрипторов GDT является ключевым в схеме адресации защищённого режима. Регистр GDTR имеет длину 5 байт. Старшие 3 байта содержат 24-разрядный физический адрес таблицы GDT, младшие два байта - длину таблицы GDT, уменьшенную на 1. Длина GDT, уменьшенная на единицу, называется пределом таблицы GDT (GDT limit). Она используется для проверки правильности задаваемых программой селекторов.



**Перед переходом в защищённый режим программа должна создать в оперативной памяти таблицу GDT и загрузить регистр GDTR и регистр IDTR при помощи специальной команды LGDT.**

lgdt fword ptr GDT\_GDT

Для представления старших 20 битов физического адреса таблицы страниц в дескрипторе используются биты 12-31. Младшие 12 битов адреса таблицы всегда равны нулю, таким образом, таблица страниц должна быть выровнена в памяти на границу 4096 байт (на границу страницы)



Формат дескриптора каталога таблиц страниц



Формат дескриптора страницы

**Бит A** называется битом обращения к сегменту и для нашей программы должен быть установлен в 0.

**Бит P** называется битом присутствия сегмента в памяти. Для тех сегментов, которые находятся в физической памяти этот бит должен быть установлен в 1.

Любая попытка программы обратиться к сегменту памяти, в дескрипторе которого бит P установлен в 0, приведёт к прерыванию.

**Бит R**, называемое битом разрешения чтения сегмента. Если этот бит установлен в 1, программа может считывать содержимое сегмента кода. В противном случае процессор может только выполнять этот код.

**Поле W** называется битом разрешения записи в сегмент. Если этот бит установлен в 1, наряду с чтением возможна и запись в данный сегмент. В противном случае при попытке чтения выполнение программы будет прервано.

**Поле D** задаёт направление расширения сегмента. Обычный сегмент данных расширяется в область старших адресов (расширение вверх). Если же в сегменте расположен стек, расширение происходит в обратном направлении - в область младших адресов (расширение вниз). Для сегментов, в которых организуются стеки, необходимо устанавливать поле D равным 1.

**Поле TYPE** дескриптора определяет способ, которым можно использовать тот или иной сегмент. Разделение сегментов на типы позволяет защититься от случайного или преднамеренного использования сегментов не по назначению.

На выполнение прямого вызова целевого сегмента или прямого перехода к целевому сегменту влияет **бит подчинения C**, который располагается в бите 2 байта доступа дескриптора целевого сегмента.

**Если бит C** установлен в 0, целевой сегмент называется несогласованным. Несогласованный сегмент может быть вызван только такой программой, которая имеет большие или такие же привилегии, что и целевой сегмент. Т.е. должно выполняться условие CPL <= DPL.

В согласованном сегменте кода **бит подчинения C** байта доступа установлен в 1. Согласованный сегмент можно вызывать из программ, находящихся в любом кольце. Но в любом случае при вызове этот сегмент будет выполняться с привилегиями вызывающей программы. Если согласованный сегмент вызывается из кольца 0, он будет выполняться с уровнем привилегий 0, если из кольца 3 - с уровнем привилегий 3.

CR1 — зарезервирован.

CR2 — используется при страничной организации памяти в ситуации, когда происходит обращение к виртуальной странице, отсутствующей в памяти. Возникает исключительная ситуация, и линейный 32-битный адрес команды, вызвавшей это исключение, записывается в регистр CR2. Обработчик исключения использует адрес для подкачки заданной страницы в память.

CR3 — используется при страничной организации памяти. Регистр каталога страниц первого уровня. Содержит 32-битный физический базовый адрес каталога страниц текущей задачи. Каталог содержит 1024 32-битных дескриптора, каждый из которых содержит адрес таблицы страниц второго уровня. В свою очередь каждая из таблиц страниц второго уровня содержит 1024 32-битных дескриптора, адресующих страничные кадры в памяти.

Самый первый дескриптор в таблицах GDT и LDT никогда не используется. Программа может загрузить в сегментный регистр селектор, соответствующий первому дескриптору (поле индекса в таком селекторе равно нулю), однако при попытке использовать такой селектор произойдёт прерывание работы программы. Селектор с нулевым полем индекса (пустой селектор) загружается операционной системой в неинициализированные сегментные регистры перед передачей управления запущенной программе.

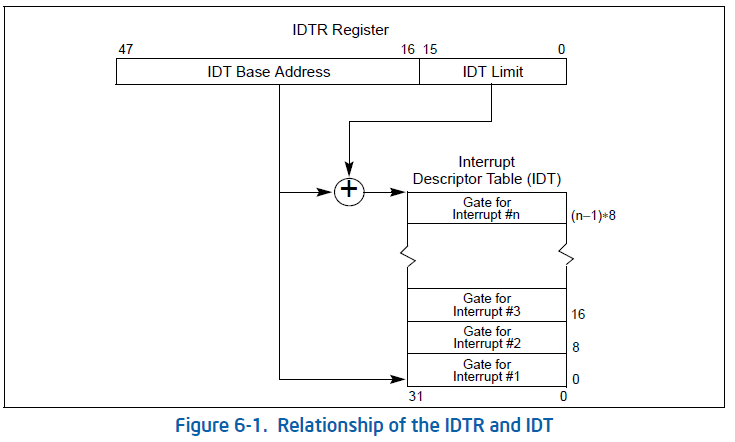
**Таблица прерываний защищённого режима.**

Обработка прерываний и исключений в защищённом режиме по аналогии с реальным режимом базируется на таблице прерываний. Но таблица прерываний защищённого режима является таблицей дескрипторов, которая содержит так называемые вентили прерываний, вентили исключений и вентили задач.

Таблица прерываний защищённого режима называется дескрипторной таблицей прерываний IDT (Interrupt Descriptor Table). Также как и таблицы GDT и LDT, таблица IDT содержит 8-байтовые дескрипторы. Причём это системные дескрипторы - вентили прерываний, исключений и задач. Поле TYPE вентиля прерывания содержит значение 6, а вентиля исключения - значение 7.



Рисунок 1.4 – Формат элементов дескрипторной таблицы прерываний IDT



Расположение таблицы дескрипторов определяется содержимым 5-байтового внутреннего регистра процессора IDTR. Формат регистра IDTR полностью аналогичен формату регистра GDTR, для его загрузки используется команда LIDT. Так же, как регистр GDTR содержит 24-битовый физический адрес таблицы GDT и её предел, так и регистр IDTR содержит 24-битовый физический адрес дескрипторной таблицы прерываний IDT и её предел.

Для переключения процессора из реального режима в защищённый можно использовать, например, такую последовательность команд:

mov ax, cr0

or ax, 1

mov cr0, ax

Для совместимости с процессором i80286 оставлена возможность переключения в защищённый режим с помощью команды LMSW.

Для возврата в реальный режим необходимо сбросить бит PE:

mov ax, cr0

and ax, 0fffe

mov cr0, ax

***Алгоритм решения задачи:***

1. Подготовка часов реального, а именно установка формата времени в двоичный.
2. Проинициализировать глобальную таблицу селекторов, а именно определить селектор на код, стек, данные.
3. Проинициализировать таблицу прерываний на все аппаратные прерывания.
4. Загрузить таблицы прерываний и глобальную селекторов в соответствующие регистры.
5. Перевести процессор в защищенный режим, и jump far на следующую команду за командой перехода.
6. Пока не отошло отведенное время ожидаем.
7. Проинициализировать контроллер прерываний.
8. Приводим сегменты к совместимости с реальным режимом.
9. Включаем реальный режим и делаем jump far на команду за последней командой в защищенном режиме.

***Листинг программы:***

.386p

.MODEL LARGE

SegDescr struc ; структура дескриптора

limit dw 0 ; предел сегмента

offsetLow dw 0 ; мл. слово физического адреса

offsetMid db 0 ; 3-ий байт физического адреса

access db 0 ; права доступа

attributes db 0 ; старшая часть предела и флаги GDXU

offsetHigh db 0 ; 4-ый байт физического адреса

SegDescr ends

IntDescr struc ; вентиль прерывания

ioffsetLow dw 0 ; младшее слово смещения обработчика

iselector dw 0 ; селектор обработчика

alwaysZero db 0 ; зарезервировано, всегда 0

access db 0 ; байт доступа

ioffsetHigh dw 0 ; старшее слово смещения обработчика

IntDescr ends

Special\_IDT\_ptr struc

plimit dw 0 ; предел IDT

idtLowPointer dw 0 ; младшее слово физического адреса

idtHighPointer dw 0 ; старшее слово физического адреса

Special\_IDT\_ptr ends

; Права доступа

ACCESS\_READ EQU 00000010B

ACCESS\_CODE EQU 10011000B ; сегмент кода

ACCESS\_DATA EQU 10010010B ; сегмент данных

ACCESS\_STACK EQU 10010010B ; сегмент стека

ACCESS\_IDT EQU 10010010B ; IDT в GDT

INTERRUPT\_GATE EQU 10001110B ; вентиль прерывания

EXCEPTION\_GATE EQU 10001111B ; вентиль исключения

ACCESS\_ALL EQU 01100000B ; общий доступ

DATA segment para use16

DATA\_SEGMENT\_BEGIN = $

exit\_message db 10, 13, "Welcom back to RM!","$"

protected\_greet\_message db "you have left:",0

enter\_seconds\_message db "Input seconds to be in PM: $"

interrupt\_masks\_master db ?

interrupt\_masks\_slave db ?

SECOND db ?

TIME db 0

BUFFER\_TIME db 8 dup(' ')

db 0

input\_time\_buffer db 6, 7 dup(?)

; GDT

GDT\_BEGIN = $ ;отмечаем начало GDT

GDT label word

GDT\_0 SegDescr <0, 0, 0, 0, 0, 0> ;первый элемент не используется

;;;;;;ОПИСАНИЕ СЕГМЕНТОВ

GDT\_GDT SegDescr <GDT\_SIZE - 1, 0, 0, ACCESS\_DATA, 0, 0>

GDT\_CODE\_RM SegDescr <SIZE\_CODE\_RM - 1, 0, 0, ACCESS\_CODE, 0, 0>

GDT\_DATA SegDescr <DATA\_SEGMENT\_SIZE - 1, 0, 0, ACCESS\_DATA + ACCESS\_ALL, 0, 0>

GDT\_STACK SegDescr <1000h - 1, 0, 0, ACCESS\_DATA, 0, 0>

GDT\_TEXT SegDescr <2000h - 1,8000h,0Bh, ACCESS\_DATA + ACCESS\_ALL, 0, 0>

GDT\_CODE\_PM SegDescr <SIZE\_CODE\_PM - 1, 0, 0, ACCESS\_CODE + ACCESS\_READ, 40h, 0>

GDT\_IDT SegDescr <SIZE\_IDT - 1, 0, 0, ACCESS\_IDT, 0, 0>

GDT\_SIZE = ($ - GDT\_BEGIN) ;вычисляем размер GDT

; Селекторы, определённые в таблице GDT

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)

IDTptr Special\_IDT\_ptr <SIZE\_IDT, 0, 0>

; IDT

IDT label word

IDT\_BEGIN = $

; Вентили исключений 00...0F

irpc N, 0123456789ABCDEF

IDT\_0&N IntDescr <0, CODE\_PM\_DESC, 0, EXCEPTION\_GATE, 0>

endm

; Вентили исключений 10...1F

irpc N, 0123456789ABCDEF

IDT\_1&N IntDescr <0, CODE\_PM\_DESC, 0, EXCEPTION\_GATE, 0>

endm

; Вентили прерываний для таймера и клавиатуры

IDT\_TIMER IntDescr <0, CODE\_PM\_DESC, 0, INTERRUPT\_GATE, 0>

IDT\_KEYBOARD IntDescr <0, CODE\_PM\_DESC, 0, INTERRUPT\_GATE, 0>

;Вентили прерываний 22...27

irpc N, 234567

IDT\_2&N IntDescr <0, CODE\_PM\_DESC, 0, INTERRUPT\_GATE, 0>

endm

; Вентиль прерывания для часов реального времени

IDT\_RTC IntDescr <0, CODE\_PM\_DESC, 0, INTERRUPT\_GATE, 0>

; Вентили прерываний 28...2F

irpc N, 89ABCDEF

IDT\_2&N IntDescr <0, CODE\_PM\_DESC, 0, INTERRUPT\_GATE, 0>

endm

SIZE\_IDT = ($ - IDT\_BEGIN)

DATA\_SEGMENT\_SIZE = ($ - DATA\_SEGMENT\_BEGIN)

DATA ends

CODE\_RM segment para use16 ;использование 16-битного регистра

CODE\_RM\_BEGIN = $

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

@START: ;;;;;;; START

mov ax, DATA ;;Инициализиция сегментных регистров

mov ds, ax

mov es, ax

lea dx, enter\_seconds\_message

mov ah, 9h

int 21h

call input\_time ;Ввод времени

mov ds:[TIME], al

@PREPARE\_RTC: ;Подготовка часов RTC

mov al, 0Ah

out 70h, al

in al, 71h

or al, 00101111b

out 71h, al

mov al, 0Bh

out 70h, al ;Выбрать регистр состояния 0Bh

in al, 71h ;Получить значение регистра 0Bh

or al, 01010110b ;Установить бит DM в 1 - формат представления время в двоичном виде

out 71h, al ;Записать измененное значение

@SAVE\_MASK: ;Сохранить маски прерываний

in al, 21h

mov interrupt\_masks\_master, al

in al, 0A1h

mov interrupt\_masks\_slave, al

cli

@LOAD\_GDT: ;Заполнить глобальную таблицу дескрипторов

mov ax, DATA

;загружаем в ax адрес сегмента данных

;;формируем в dl:ax физический адрес, соответствующий сегментному адресу

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

add ax, offset GDT ;складываем со смещением

adc dx, 0

; Записываем физический адрес GDT в элемент GDT,

; описывающий саму GDT

mov [bx][SegDescr.offsetLow], ax

mov [bx][SegDescr.offsetMid], dl

mov [bx][SegDescr.offsetHigh], dh

@WRITE\_CODE\_RM: ;Заполнить дескриптор сегмента кода реального режима. Заполнение дескриптора аналогично заполнению глобальной таблицы дескрипторов.

lea bx, GDT\_CODE\_RM

mov ax,cs

xor dh,dh

mov dl,ah

shl ax,4

shr dx,4

mov [bx][SegDescr.offsetLow], ax

mov [bx][SegDescr.offsetMid], dl

mov [bx][SegDescr.offsetHigh],dh

@WRITE\_DATA: ;Записать дескриптор сегмента данных. Заполнение дескриптора аналогично заполнению глобальной таблицы дескрипторов.

lea bx, GDT\_DATA

mov ax, si

mov dx, di

mov [bx][SegDescr.offsetLow], ax

mov [bx][SegDescr.offsetMid], dl

mov [bx][SegDescr.offsetHigh], 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][SegDescr.offsetLow], ax

mov [bx][SegDescr.offsetMid], dl

mov [bx][SegDescr.offsetHigh], 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][SegDescr.offsetLow],ax

mov [bx][SegDescr.offsetMid],dl

mov [bx][SegDescr.offsetHigh],dh

@WRITE\_IDT: ;Записать дескриптор IDT

lea bx, GDT\_IDT

mov ax, si

mov dx, di

add ax, offset IDT

adc dx, 0

mov [bx][SegDescr.offsetLow], ax

mov [bx][SegDescr.offsetMid], dl

mov [bx][SegDescr.offsetHigh], dh

mov IDTptr.idtLowPointer, ax

mov IDTptr.idtHighPointer, dx

@FILL\_IDT: ;Заполнить таблицу дескрипторов шлюзов прерываний

lea eax, TIMER\_HANDLER ;Заполнить шлюзы 00-0F исключениями

mov IDT\_TIMER.ioffsetLow,ax

shr eax, 16

mov IDT\_TIMER.ioffsetHigh,ax

lea eax, KEYBOARD\_HANDLER

mov IDT\_KEYBOARD.ioffsetLow,ax

shr eax, 16

mov IDT\_KEYBOARD.ioffsetHigh,ax

irpc N, 234567 ;Заполнить шлюзы 10-1F исключениями

lea eax,DUMMY\_IRQ\_MASTER

mov IDT\_2&N.ioffsetLow, ax

shr eax,16

mov IDT\_2&N.ioffsetHigh, ax

endm

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

mov IDT\_RTC.ioffsetLow,ax

shr eax, 16

mov IDT\_RTC.ioffsetHigh,ax

irpc N, 9ABCDEF

lea eax,DUMMY\_IRQ\_SLAVE

mov IDT\_2&N.ioffsetLow,ax

shr eax,16

mov IDT\_2&N.ioffsetHigh,ax

endm

lgdt fword ptr GDT\_GDT ;Загрузить регистр GDTR

lidt fword ptr IDTptr ;Загрузить регистр IDTR

mov eax, cr0 ;Получить управляющий регистр cr0

or al, 1 ;Установить бит PE в 1

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

; Выполняем команду межсегментного перехода,

; в качестве селектора указываем селектор текущего

; сегмента кода

db 0EAH

dw $ + 4

dw CODE\_RM\_DESC

@LOAD\_SELECTORS:

mov ax, DATA\_DESC

mov ds, ax

mov es, ax

mov ax, STACK\_DESC

mov ss, ax

xor ax, ax

mov fs, ax

mov gs, ax

lldt ax

@PREPARE\_TO\_RETURN:

push cs

push offset @BACK\_TO\_RM

lea edi, @ENTER\_PM

mov eax, CODE\_PM\_DESC

push eax

push edi

@REINITIALIAZE\_CONTROLLER\_FOR\_PM:

mov al, 00010001b

out 20h, al

out 0A0h, al

mov al, 20h

out 21h, al

mov al, 28h

out 0A1h, al

mov al, 04h

out 21h, al

mov al, 02h

out 0A1h, al

mov al, 11h

out 21h, al

mov al, 01h

out 0A1h, al

mov al, 0

out 21h, al

out 0A1h, al

nop

sti

@GO\_TO\_CODE\_PM:

db 66h

retf

@BACK\_TO\_RM:

cli

in al, 70h

or al, 0FEh ;команда отключения

out 70h, al

nop

@REINITIALISE\_CONTROLLER:

mov al, 00010001b

out 20h, al

out 0A0h, al

mov al, 8h

out 21h, al

mov al, 70h

out 0A1h, al

mov al, 04h

out 21h, al

mov al, 02h

out 0A1h, al

mov al, 11h

out 21h, al

mov al, 01h

out 0A1h, al

@PREPARE\_SEGMENTS:

mov GDT\_CODE\_RM.limit, 0FFFFh

mov GDT\_DATA.limit, 0FFFFh

mov GDT\_STACK.limit, 0FFFFh

db 0EAH

dw $ + 4

dw CODE\_RM\_DESC

mov ax, DATA\_DESC

mov ds, ax

mov es, ax

mov fs, ax

mov gs, ax

mov ax, STACK\_DESC

mov ss, ax

@ENABLE\_REAL\_MODE:

mov eax, cr0

and al, 11111110b

mov cr0, eax

db 0EAH

dw $ + 4

dw CODE\_RM

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 IDTptr.plimit, 3FFH

mov dword ptr IDTptr + 2, 0

lidt fword ptr IDTptr

@REPEAIR\_MASK:

mov al, interrupt\_masks\_master

out 21h, al

mov al, interrupt\_masks\_slave

out 0A1h, al

@ENABLE\_INTERRUPTS:

nop

sti

@EXIT:

lea dx, exit\_message

mov ah,9h

int 21h

mov ax,4C00h

int 21H

input\_time proc near

mov ah, 0ah

xor di, di

mov dx, offset ds:[input\_time\_buffer]

int 21h

mov dl, 0ah

mov ah, 02

int 21h

mov si, offset input\_time\_buffer + 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 @input\_time\_return

sub cl, '0'

mul bx

add ax, cx

inc si

jmp @II2

@input\_time\_return:

ret

input\_time endp

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

CODE\_RM ends

CODE\_PM segment para use32

CODE\_PM\_BEGIN = $

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

@ENTER\_PM:

mov edi, 3840

lea esi, protected\_greet\_message

call BUFFER\_OUTPUT

@INFINITE\_LOOP:

jmp @INFINITE\_LOOP

@EXIT\_PM:

db 66H

retf

@EXIT\_FROM\_INTERRUPT:

popad

pop es

pop ds

pop eax

pop eax

pop eax

sti

db 66H

retf

WORD\_TO\_DEC proc near

pushad ;str in di

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

DUMMY\_IRQ\_MASTER proc near

push eax

mov al, 20h

out 20h, al

pop eax

iretd

DUMMY\_IRQ\_MASTER endp

DUMMY\_IRQ\_SLAVE proc near

push eax

mov al, 20h

out 20h, al

out 0A0h, al

pop eax

iretd

DUMMY\_IRQ\_SLAVE endp

TIMER\_HANDLER proc near

; jmp RTC\_HANDLER

TIMER\_HANDLER endp

RTC\_HANDLER proc near

push ds

push es

pushad

mov ax, DATA\_DESC

@SHOW\_TIMER: ;///////////////

mov al, 0h

out 70h, al

in al, 71h

cmp al, ds:[SECOND]

je @SKIP\_SECOND

mov ds:[SECOND], al

mov al, ds:[TIME]

cmp ds:[TIME], 0

je @DISABLE\_PM

xor ah, ah

lea edi, ds:[BUFFER\_TIME]

call WORD\_TO\_DEC

mov edi, 3980

lea esi, BUFFER\_TIME

call BUFFER\_OUTPUT

dec ds:[TIME]

lea esi, BUFFER\_TIME

call BUFFER\_CLEAR

jmp @SKIP\_SECOND

@DISABLE\_PM:

mov al, 20h

out 20h, al

jmp @EXIT\_FROM\_INTERRUPT

@SKIP\_SECOND:

mov al, 20h

out 20h, al

popad

pop es

pop ds

iretd

RTC\_HANDLER endp

KEYBOARD\_HANDLER proc near

in al, 60h

push eax

mov al, 20h

out 20h, al

pop eax

iretd

KEYBOARD\_HANDLER endp

BUFFER\_CLEAR proc near

mov al, ' '

push ebx

xor ebx, ebx

@BUFFER\_CLEAR\_CYCLE:

mov byte ptr [esi + ebx], al

inc ebx

cmp ebx, 8

jne @BUFFER\_CLEAR\_CYCLE

pop ebx

ret

BUFFER\_CLEAR endp

BUFFER\_OUTPUT proc near

push es

pushad

mov ax, TEXT\_DESC

mov es, ax

@OUTPUT\_LOOP:

lodsb

or al, al

jz @OUTPUT\_EXIT

stosb

inc edi

jmp @OUTPUT\_LOOP

@OUTPUT\_EXIT:

popad

pop es

ret

BUFFER\_OUTPUT ENDP

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

CODE\_PM ENDS

STACK\_A segment para stack

db 1000h dup(?)

STACK\_A ends

***Результат работы программы:***

******

***Заключение:***