Московский государственный университет леса

Кафедра Вычислительной Техники

Лабораторная работа №1

Отчет

Выполнил: студент гр. ВТ-32
Чернышев И.Е.
Проверил: Ефремов Н.В.

Москва 2005

Цель лабораторной работы

Целью лабораторной работы является изучение механизма сегментной адресации памяти в реальном и защищенном режимах работы процессора i486, структуры дескрипторов сегментов памяти и дескрипторных таблиц, особенностей работы видеобуфера графического адаптера, способов переключения из реального режима в защищенный и обратно.

Общие сведения

а) Механизм сегментной адресации памяти в реальном и защищенном режимах работы процессора.

В реальном режиме:

Базовый адрес сегмента находится в сегментном регистре. Сегменты в памяти выровнены по параграфам, поэтому сегментный регистр содержит 16 старших разрядов базового адреса.

Команды выбираются из CS:IP, данные из SS:SP,DS:SI/DI..

В защищенном режиме:

Каждый сегмент характеризуется специальной восьмибайтовой структурой данный, которая называется дескриптор сегмента. С помощью этой структуры можно задать 32х разрядный базовый адрес сегмента с любого места в линейном адресном пространстве. А размер сегмента можем указать до 1мб с точностью 1 байт и до 4Гб с точностью 1 страница (4 кб).

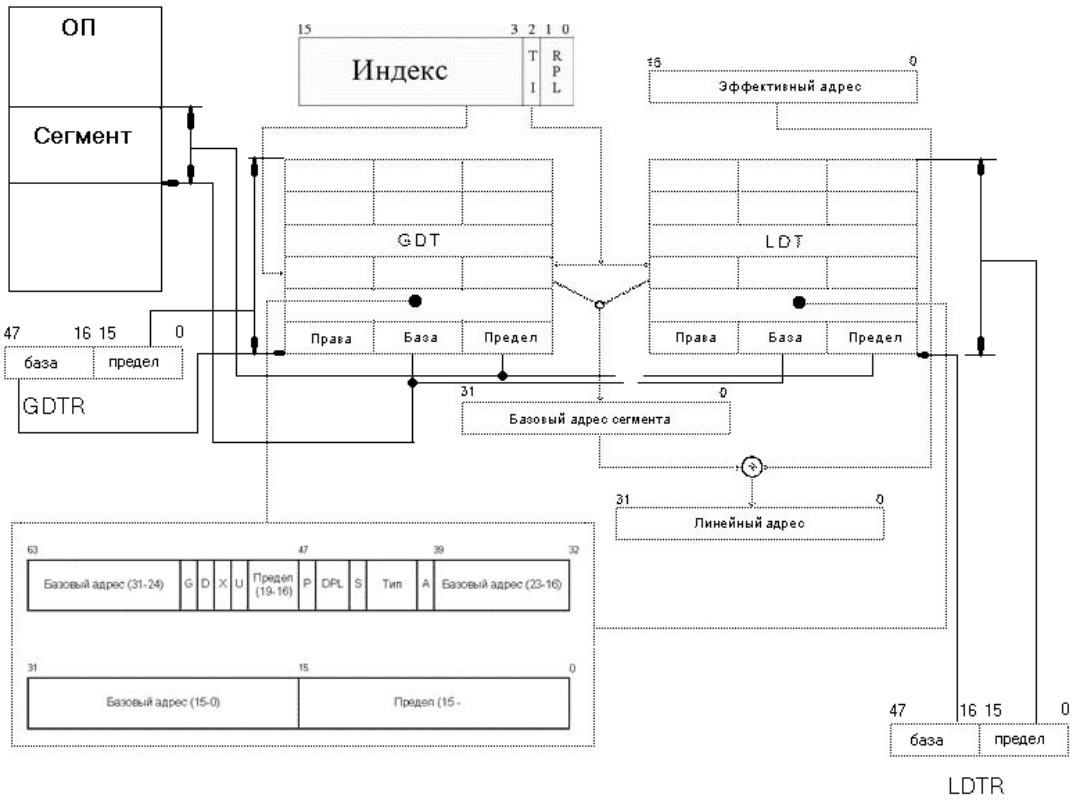
Дескрипторы объединяются в таблицы, есть 3 типа дескрипторных таблиц:

-прерываний IDT (одна) – аналог таблицы векторов прерываний

-глобальная (GDT)(одна)

-локальная (LDT) –содержит дескрипторы сегментов, которые принадлежат одной задаче.

б) Структура и назначение GDT, LDT,GDTR,LDTR, типы дескрипторов. Назначение отдельных разрядов и полей дескрипторов памяти.



G-бит гранулярности.

0 –предел определен в байтах

1 – предел определен в страницах

P- бит присутствия

1 – сегмент присутствует в ОП

0 – сегмент отсутствует в ОП

DPL-уровень привилегий дескриптора

0-максимальный уровень привилегий (ядро ОС)

1-ОС

2-системы программирования

3-пользовательские программы

S-что определяет дескриптор

0 – системный объект

1 – сегмент памяти

Тип:

	сегмент	выполнение	запись	чтение
000	Data			*
001	Data		*	*
010	---			
011	Stack		*	*
100	Code	*		
101	Code	*		*
110	Slave Code	*		
111	Slave Code	*		*

A-бит обращения.

0 – не было обращения

1 – было обращение к сегменту

D-разрядность операндов сегмента

0 – 16x

1 - 32x

U-предназначен для использования системными программистами по их усмотрению

X – зарезервирован.

Ti-указатель таблицы

0 – обращение к GDT

1 – обращене к LDT

RPL – запрашиваемый уровень привилегий.

в) Способы адресации операндов.

-неявная

-регистровая

-непосредственная

-прямая (абсалютная)

-базовая

-базовая со смещением

-индексная со смещением

-базовая индексная со смещением

-стековая

г) Особенности работы видеобуфера графического адаптера.

Рассмотрим цветной текстовый видеорежим разрешением 80x25 . Как правило, этот режим устанавливается по умолчанию при загрузке компьютера.

В этом режиме информация, отображаемая на экране, размещается в видеопамяти с адреса 8000H. Каждый отображаемый на экране символ занимает два последовательных байта памяти: младший – код символа, старший – цвет символа. При этом младшая тетрада задает цвет символа, старшая – цвет фона. Цвет кодируется по системе RGB следующим образом: самый младший бит - синий; следующий – зеленый; далее – красный. Самый старший бит в тетраде задает интенсивность цвета (обычный или яркий). При этом в тетраде, отвечающей за цвет фона, этот бит в зависимости от режима работы адаптера может управлять не интенсивностью, а миганием символа.

д) Формат и назначение разрядов служебного регистра CR0.

Регистр CR0 содержит системные флаги, управляющие режимами работы микропроцессора и отражающие его состояние глобально, независимо от конкретных выполняющихся задач. Назначение системных флагов:

Pe (Protect Enable), бит 0 – разрешение защищенного режима работы. Состояние этого флага показывет, в каком из двух режимов – реальном (Pe=0) или защищенном (Pe=1) – работает микропроцессор в данный момент времени.

Mp (Math Present), бит 1 – наличие сопроцессора. Всегда 1;

Em (Emulation), бит 2 – бит эмуляции

Ts (Task Switched), бит 3 – переключение задач. Процессор автоматически устанавливает этот бит при переключении на выполнение другой задачи;

Et (Extension Type), бит 4 – тип расширения – совместно с другими показывает поддержку команд FPU;

Ne (Numeric Error), бит 5 – численная ошибка FPU;

Wp (Write Protect), бит 16 – защита от записи; при установленном бите осуществляется защита от записи страниц уровня пользователя;

Am (Alignment Mask), бит 18 – маска выравнивания. Этот бит разрешает (am =1) или запрещает (am=0) контроль выравнивания;

Nw (Not Write Through), бит 29 – не сквозная запись.

Cd (Cache Disable), бит 30 – запрещение кэш-памяти. С помощью этого бита можно запретить (cd=1) или разрешить (cd=0) использование внутренней кэш-памяти (кэш-памяти первого уровня);

Pg (PaGing), бит 31 – разрешение (pg=1) или запрещение (pg=0) страничного преобразования. Регистр CR0 используется при страничной модели организации памяти.

е) Способы переключения процессора из реального режима в защищенный и обратно.

1) Для перехода в защищённый режим и возврата из него в реальный будем использовать только один бит из регистра CR0 - это нулевой бит PE . Если установить этот бит в 1, процессор перейдёт в защищённый режим, если сбросить - то в реальный.

1. Переводим процессор в защищённый режим.
```
Mov eax,cr0
Or al,1
Mov cr0,eax
```
1-я команда программы, которая выполнится в защищённом режиме.
```
...
```
2. Переводим процессор в реальный режим.
```
Mov eax,cr0
And al,0feh
Mov cr0,eax
```

2) Для перехода в защищённый режим можно также использовать специальную команду LMSW, загружающую регистр состояния процессора (Mashine Status Word). Младший бит этого регистра указывает режим работы процессора. Значение, равное 0, соответствует реальному режиму работы, а значение 1 - защищённому.
1. Переходим в защищенный режим:
```
mov     ax, 1
lmsw    ax
```

К сожалению, с помощью команды LMSW невозможно переключить процессор обратно в реальный режим. Для этого необходимо использовать другой способ.

3)
1.Переходим в защищенный режим (i286):
Записываем адрес возврата в реальный режим в область данных BIOS по адресу 0040h:0067h.Записываем в CMOS-память в ячейку 0Fh код 5, это обеспечит после выполнения сброса процессора передачу управления по адресу, подготовленному нами в области данных BIOS по адресу 0040h:0067h. Для того, чтобы немаскируемые прерывания были запрещены, устанавливаем в 1 старший бит при определении ячейки CMOS.

```
cli
mov al,8f
out 70h,al
jmp next1
next1:
mov al, 5
out 70h+1,al ; код возврата
        Открываем адресную линию A20
mov al,0d1h ; команда управления линией A20
out 64h,al
mov al,0dfh ; открыть A20
out 60h,al
```

Устанавливаем защищённый режим работы процессора
mov ax,0001h ; бит перехода в защищённый режим
lmsw ax

2.Возвращаемся в реальный режим
mov al,0feh команда сброса процессора
out 64h,al
wait_reset:
hlt
 закрываем адресную линию A20
mov al,0d1h
out 64h,al
mov al,0ddh
out 60h,al
out 70h,al ; порт для доступа к CMOS-памяти
in al,21h ; порт для маскирования прерываний
and al,0 ; сбрасываются все биты
out 21h,al ; порт для маскирования прерываний
sti

4) Функция 89h прерывания 15h выполняет переход в защищенный режим и некоторые другие действия.

Ah=89h
Bl=новый номер для аппаратного прерывания уровня irq0. Уровни irq1..7 будут иметь следующие по порядку номера;
Bh= новый номер для аппаратного прерывания уровня irq8. Уровни irq9..a будут иметь следующие по порядку номера;
Ds:si =адрес GDT для защищенного режима;
Cx=адрес первой выполняемой команды в защищенном режиме.

Эта функция предполагает, что дескрипторы в таблице GDT расположены в определенной последовательности:
0h – пустой дескриптор;
8h – дескриптор таблицы GDT;
10h – дескриптор таблицы LDT;
18h – дескриптор сегмента данных, на него указывает селектор в регистре ds;
20h – дескриптор дополнительного сегмента данных, на него указывает селектор в регистре es:
28h – дескриптор сегмента стека, на него указывает селектор в регистре ss;
30h – дескриптор сегмента кода, на него указывает селектор в регистре cs;
- остальные дескрипторы.

Листинг программы

```
.386p                    ; разрешение команд 386 и 486
descr struc
lim        dw        0      ;предел (биты 15-0)
base_1     dw        0      ;база (биты 15-0)
base_m     db        0      ;база (биты 23-16)
attr_1     db        0      ;байт атрибутов 1
attr_2     db        0      ;предел (биты 19-16)   и атрибуты 2
base_h     db        0      ;база (биты 31-24)
descr      ends
data       segment  use16     ; начало сегмента данных
gdt_null   descr      <0,0,0,0,0,0>                ;нулевой дескриптор
gdt_data   descr      <data_size-1,0,0,92h,0,0>      ;дескр сег данн
gdt_code   descr      <code_size-1,0,0,98h,0,0>      ;дескр сег кода
gdt_stack  descr      <255,0,0,92h,0,0>              ;дескр сег стека
gdt_new            descr          <768,0,0,92h,80h,80h>
gdt_screen  descr     <4095,8000h,0bh,92h,0,0>       ;дескр видеобуфера
gdt_size=$-gdt_null
pdescr     dq        0      ; псевдодескриптор
sym        db        40h      ; символ
attr       db        0dh    ; его атрибут
mes        db ' Вернулись в реальный режим  $';
mes_pr     db ' Находимся в защищенном режиме $';
data_size=$-gdt_null         ; размер сегмента данных
data       ends            ; конец сегмента данных
text       segment  'code' use16  ;16-разрядный код
           assume   cs:text,ds:data
main       proc
           xor     eax,eax
           mov     ax,data
           mov     ds,ax   ; адрес сегмента данных
; вычислим 32-битовый линейный адрес сегмента данных и загрузим его
; в дескриптор сегмента данных в GDT
           shl     eax,4
           mov     ebp,eax
           mov     bx,offset gdt_data
           mov     [bx].base_1,ax
           rol     eax,16
           mov     [bx].base_m,al
; аналогично для линейного адреса сегмента команд
           xor     eax,eax
           mov     ax,cs
           shl     eax,4
           mov     bx,offset gdt_code
           mov     [bx].base_1,ax
           rol     eax,16
           mov     [bx].base_m,al
           xor     eax,eax
           mov     ax,stk
           shl     eax,4
           mov     bx,offset gdt_stack
```

```
        mov     [bx].base_1,ax
        rol     eax,16
        mov     [bx].base_m,al
; подготовим псевдодескриптор и загрузим GDTR
        mov     dword ptr pdescr+2,ebp      ; база GDT,биты 31-0
        mov     word ptr pdescr,gdt_size-1  ; предел GDT
        lgdt    pdescr
; подготовимся к переходу в защищенный режим
        cli             ; запрет аппаратных прерываний
        mov     al,80h    ; запрет NMI
        out     70h,al    ; порт КМОП микросхемы
; Переходим в защищенный режим
        mov     eax,cr0
        or      eax,1       ; установим бит PE
        mov     cr0,eax
; Процессор работает в защищенном режиме
; загружаем в CS:IP селектор: смещение точки continue
; и заодно очищаем очередь команд
        db      0eah            ;код команды jmp far
        dw      offset continue ; смещение
        dw      16              ; селектор
continue:
; адресуем данные
        mov     ax,8            ;селектор сегмента данных
        mov     ds,ax           ;
; адресуем стек
        mov     ax,24           ;селектор сегмента стека
        mov     ss,ax           ;
; инициализируем ES и выводим символы
        mov     ax,40           ;селектор сегмента видеобуфера
        mov     es,ax           ;
        mov     bx,160*3+80     ;начальное смещение на экране
        mov     cx,0fh          ;число выводимых символов
        mov     ax,word ptr sym  ;начальный символ с атрибутом
screen:    mov     es:[bx],ax       ;вывод в видеобуфер
        add     bx,162          ;смещение в видеобуфере
        inc     ax              ;следующий символ
        push    cx              ;пауза
        mov     ecx,0fffffh     ;для
payza:     db 67h               ;наглядности
        loop    payza
        pop     cx
        loop    screen          ;цикл вывода на экран
; Подготовим переход в реальный режим
; Сформируем и загрузим дескрипторы для реального режима
        mov     gdt_data.lim,0ffffh  ; предел сегмента данных
        mov     gdt_code.lim,0ffffh  ;               кода
        mov     gdt_stack.lim,0ffffh ;               стека
        mov     gdt_screen.lim,0ffffh;       допол сегм данных
 ; загрузим теневые регистры сегментов
        mov     ax,8
        mov     ds,ax
```

```
        mov     ax,24
        mov     ss,ax
        mov     ax,40               ;
        mov     es,ax
; Выполним дальний переход чтобы снова загрузить селектор
;в CS и модифицировать его теневой регистр
        db      0eah                ; код JMP far
        dw      offset go           ; смещение
        dw      16                  ; селектор
; Переключим режим процессора
go:       mov     eax,cr0
        and     eax,0fffffffeh      ;сбросим бит PE
        mov     cr0,eax
        db      0eah                ;код jmp far
        dw      offset return       ;смещение
        dw      text                ;сегмент
;-----------------------------------------------------------------
; Теперь процессор снова работает в реальном режиме
;Восстановим операционную среду реального режима
return:   mov  ax,data
        mov  ds,ax
        mov  ax,stk
        mov  ss,ax
; Разрешим аппаратные и немаскируемые прерывания
        sti         ; разрешение прерываний
        mov  al,0    ; сброс бита 7 в порте КМОП-
        out  70h,al  ; разрешение NMI
        mov  ah,09h
        mov  dx,offset mes
        int  21h
;-----------------------------------------------------------------
                    mov bx,offset gdt_null
                    mov cx,6
mega:               mov dl,[bx].attr_1
                    mov ah,02h
                    and dl,1
                    add dl,30h
                    int 21h
                    add bx,8
loop mega
;-----------------------------------------------------------------
        mov  ax,4c00h  ; завершим программу обычным образом
        int  21h
main      endp
code_size=$-main        ;размер сегмента кода
text      ends
stk       segment   stack 'stack' use16
        db        256 dup ('^')
stk       ends
        end main; переход в защищенный режим из реального и наоборот
```

Объектный код

```
Turbo Assembler        Version 2.0        01/21/05 16:09:49      Page 1
ch1.asm

    1                                  .386p                          ;     разрешение
команд 386 и 486
    2                                  ;  Структура для описания дескрипторов
        сегментов
    3   00000000                       descr struc
    4   00000000  01*(0000)            lim         dw          0       ;предел (биты
15-0)
    5   00000002  01*(0000)            base_1              dw          0       ;база
(биты 15-0)
    6   00000004  01*(00)              base_m              db          0       ;база
(биты 23-16)
    7   00000005  01*(00)              attr_1      db          0       ;байт атрибутов
1
    8   00000006  01*(00)              attr_2      db          0       ;предел (биты
19-16)
    9                                  ;                                   и атрибуты 2
    10  00000007  01*(00)              base_h              db          0       ;база
(биты 31-24)
    11  00000008                       descr       ends
    12
    13      0000                       data        segment  use16      ; начало
сегмента данных
    14                                 ; Таблица глобальных дескрипторов GDT
    15      0000  0000 0000 00 00 00 +         gdt_null    descr       <0,0,0,0,0,0>
                  ;нулевой дескриптор
    16            00
    17      0008  0077 0000 00 92 00 +         gdt_data    descr       <data_size-
1,0,0,92h,0,0>  ;дескр сег данн
    18            00
    19      0010  0106 0000 00 98 00 +         gdt_code    descr       <code_size-
1,0,0,98h,0,0>  ;дескр сег кода
    20            00
    21      0018  00FF 0000 00 92 00 +         gdt_stack   descr       <255,0,0,92h,0,0>
                  ;дескр сег стека
    22            00
    23      0020  0300 0000 00 92 80 +         gdt_new     descr
<768,0,0,92h,80h,80h>
    24            08
    25      0028  0FFF 8000 0B 92 00 +         gdt_screen  descr
<4095,8000h,0bh,92h,0,0>     ;дескр видеобуфера
    26            00
    27            = 0030               gdt_size=$-gdt_null
    28                                 ; Поля данных программы
    29      0030  0000000000000000     pdescr      dq          0       ;
псевдодескриптор
    30      0038  40                   sym         db          40h     ; символ
    31      0039  0D                   attr        db          0dh     ; его атрибут
```

```
   32    003A  20 82A5 E0 AD E3 AB+     mes        db   ' Вернулись в реальный
режим  $';
   33              A8 E1          EC 20 A2 20 E0+
   34              A5 A0          AB EC AD EB A9+
   35              20 E0 A5 A6 A8 AC 20+
   36              20 24
   37    0058  20 8DA0 E5 AE A4 A8+     mes_pr              db   ' Находимся в
защищенном режиме          $';
   38              AC E1          EF 20 A2 20 A7+
   39              A0 E9          A8 E9 A5 AD AD+
   40              AE AC          20 E0 A5 A6 A8+
   41              AC A5          20 24
   42              = 0078                 data_size=$-gdt_null       ; размер   сегмента
данных
   43    0078                     data       ends                ; конец сегмента
      данных
   44    0000                     text       segment  'code' use16  ;16-разрядный
код
   45                             ;
   46                                        assume   cs:text,ds:data
   47    0000                     main       proc
   48    0000  66| 33 C0                     xor     eax,eax
   49    0003  B8 0000s                      mov     ax,data
   50    0006  8E D8                                 mov     ds,ax   ;     адрес
сегмента данных
   51                             ; вычислим 32-битовый линейный адрес
сегмента данных и    загрузим его
   52                             ; в дескриптор          сегмента данных       в
GDT
   53    0008  66| C1 E0 04                  shl     eax,4
   54    000C  66| 8B E8                     mov     ebp,eax
   55    000F  BB 0008r                      mov     bx,offset   gdt_data
   56    0012  89 47 02                      mov     [bx].base_1,ax
   57    0015  66| C1 C0 10                  rol     eax,16

Turbo Assembler        Version 2.0       01/21/05 16:09:49      Page 2
ch1.asm


   58    0019  88 47 04                      mov     [bx].base_m,al
   59                             ; аналогично для линейного адреса сегмента
команд
   60    001C  66| 33 C0                     xor     eax,eax
   61    001F  8C C8                                 mov     ax,cs
   62    0021  66| C1 E0 04                  shl     eax,4
   63    0025  BB 0010r                      mov     bx,offset   gdt_code
   64    0028  89 47 02                      mov     [bx].base_1,ax
   65    002B  66| C1 C0 10                  rol     eax,16
   66    002F  88 47 04                      mov     [bx].base_m,al
   67                             ; аналогично для линейного адреса сегмента
стека
   68    0032  66| 33 C0                     xor     eax,eax
   69    0035  8C D0                                 mov     ax,ss
```

```
    70      0037  66| C1 E0 04                   shl     eax,4
    71      003B  BB 0018r                      mov     bx,offset   gdt_stack
    72      003E  89 47 02                      mov     [bx].base_1,ax
    73      0041  66| C1 C0 10                  rol     eax,16
    74      0045  88 47 04                      mov     [bx].base_m,al
    75                                  ; подготовим псевдодескриптор и загрузим
GDTR
    76      0048  66| 89 2E 0032r               mov     dword ptr  pdescr+2,ebp    ;
база GDT,биты 31-0
    77      004D  C7 06         0030r 002F      mov     word ptr
pdescr,gdt_size-1  ; предел GDT
    78      0053  0F 01 16 0030r                lgdt    pdescr
    79                                  ; подготовимся      к переходу в
защищенный режим
    80
    81      0058  FA                            cli             ; запрет аппаратных
прерываний
    82      0059  B0 80                         mov     al,80h      ; запрет NMI
    83      005B  E6 70                         out     70h,al          ; порт
        КМОП микросхемы
    84                                  ; Переходим в защищенный режим
    85      005D  0F 20         C0              mov     eax,cr0
    86      0060  66| 83 C8 01                  or      eax,1           ; установим бит
PE
    87      0064  0F 22 C0                      mov     cr0,eax
    88                                  ; Процессор работает в      защищенном
режиме
    89                                  ; загружаем в CS:IP селектор: смещение
        точки continue
    90                                  ; и заодно очищаем очередь команд
    91      0067  EA                            db      0eah            ;код команды
jmp far
    92      0068  006Cr                         dw      offset continue ;
смещение
    93      006A  0010                          dw      16              ; селектор
    94      006C                        continue:
    95                                  ; адресуем данные
    96      006C  B8 0008                       mov     ax,8    ;селектор  сегмента
данных
    97      006F  8E D8                         mov     ds,ax           ;
    98                                  ; адресуем стек
    99      0071  B8 0018                       mov     ax,24           ;селектор
        сегмента стека
   100      0074  8E D0                         mov     ss,ax           ;
   101                                  ;       mov     ax,48
   102                                  ;       mov     gs,ax
   103                                  ; инициализируем ES и выводим символы
   104      0076  B8 0028                       mov     ax,40           ;селектор
        сегмента видеобуфера
   105      0079  8E C0                         mov     es,ax           ;
   106
```

```
   107     007B  BB 0230                   mov    bx,160*3+80     ;начальное
смещение на экране
   108     007E  B9 000F                   mov    cx,0fh          ;число
выводимых символов
   109     0081  A1 0038r                  mov    ax,word ptr sym  ;начальный
символ          с атрибутом
   110     0084  26: 89 07          screen:  mov    es:[bx],ax      ;вывод в
видеобуфер
   111     0087  81 C300A2                 add    bx,162          ;смещение
        в видеобуфере
   112     008B  40                        inc    ax              ;следующий
символ
   113     008C  51                        push   cx              ;пауза
   114     008D  66| B9 000FFFFF           mov    ecx,0fffffh     ;для

Turbo Assembler        Version 2.0       01/21/05 16:09:49      Page 3
ch1.asm

   115     0093  67                 payza:  db   67h               ;наглядности
   116     0094  E2 FD                      loop   payza
   117     0096  59                        pop    cx
   118     0097  E2 EB                      loop   screen          ;цикл
вывода на экран
   119
   120                              ; Подготовим переход в      реальный режим
   121                              ; Сформируем и       загрузим дескрипторы
для реального режима
   122     0099  C7 060008r FFFF           mov    gdt_data.lim,0ffffh  ; предел
сегмента данных
   123     009F  C7 060010r FFFF           mov    gdt_code.lim,0ffffh  ;
           кода
   124     00A5  C7 06        0018r FFFF           mov    gdt_stack.lim,0ffffh ;
                  стека
   125     00AB  C7 06        0028r FFFF           mov    gdt_screen.lim,0ffffh;
         допол          сегм данных
   126                               ; загрузим теневые регистры сегментов
   127     00B1  B8 0008                   mov    ax,8
   128     00B4  8E D8                     mov    ds,ax           ;
   129     00B6  B8 0018                   mov    ax,24
   130     00B9  8E D0                     mov    ss,ax
   131     00BB  B8 0028                   mov    ax,40           ;
   132     00BE  8E C0                     mov    es,ax
   133                              ; Выполним дальний переход чтобы снова
        загрузить селектор
   134                              ; в CS и модифицировать его теневой
регистр
   135     00C0  EA                        db     0eah            ; код JMP
far
   136     00C1  00C5r                     dw     offset go       ;
смещение
   137     00C3  0010                      dw     16              ; селектор
```

```
   138                                  ; Переключим режим процессора
   139     00C5  0F 20C0                go:         mov     eax,cr0
   140     00C8  66| 83 E0 FE                       and     eax,0fffffffeh          ;сбросим
бит PE
   141     00CC  0F 22          C0                  mov     cr0,eax
   142     00CF  EA                                 db      0eah                    ;код jmp
far
   143     00D0  00D4r                              dw      offset return
;смещение
   144     00D2  0000s                              dw      text            ;сегмент
   145
   146                                  ;-------------------------------------------------------
---------
   147                                  ; Теперь процессор снова работает в
реальном режиме
   148                                  ;-------------------------------------------------------
----------
   149                                  ;Восстановим операционную среду
реального режима
   150     00D4  B8 0000s               return:    mov  ax,data
   151     00D7  8E D8                              mov  ds,ax
   152     00D9  B8 0000s                           mov  ax,stk
   153     00DC  8E D0                              mov  ss,ax
   154                                  ; Разрешим аппаратные и немаскируемые
прерывания
   155     00DE  FB                                 sti             ; разрешение
прерываний
   156     00DF  B0 00                              mov  al,0       ; сброс бита 7
         в порте          КМОП-
   157     00E1  E6 70                              out    70h,al   ; разрешение NMI
   158                                  ; Проверяем выполнение     функций
         ДОС после возврата в реальный режим
   159     00E3  B4 09                              mov  ah,09h
   160     00E5  BA 003Ar                           mov  dx,offset mes
   161     00E8  CD 21                              int     21h
   162                                  ;-------------------------------------------------------
-----------
   163     00EA  BB 0000r                           mov bx,offset gdt_null
   164     00ED  B9 0006                            mov cx,6
   165     00F0                         mega:
   166     00F0  8A 57          05                  mov dl,[bx].attr_1
   167     00F3  B4 02                              mov ah,02h
   168     00F5  80 E201                            and dl,1
   169     00F8  80 C230                            add dl,30h
   170     00FB  CD 21                              int 21h
   171     00FD  83 C3          08                  add bx,8

Turbo Assembler         Version 2.0         01/21/05 16:09:49       Page 4
ch1.asm

   172     0100  E2 EE                              loop mega
```

```
   173                                  ;-------------------------------------------------------
-----------
   174     0102  B8 4C00                           mov  ax,4c00h  ; завершим
программу     обычным       образом
   175     0105  CD 21                                    int    21h
   176     0107                         main       endp
   177           = 0107                 code_size=$-main      ;размер сегмента кода
   178     0107                         text       ends
   179     0000                         stk        segment   stack 'stack' use16
   180     0000  0100*(5E)                         db         256 dup ('^')
   181     0100                         stk        ends
   182                                             end  main; переход в
         защищенный режим из реального и          наоборот

Turbo Assembler        Version 2.0       01/21/05 16:09:49      Page 5
Symbol Table

Symbol Name                   Type   Value

??DATE                                Text   "01/21/05"
??FILENAME                    Text   "ch1     "
??TIME                                Text   "16:09:47"
??VERSION                     Number 0200
@CPU                          Text   0D8FH
@CURSEG                               Text   STK
@FILENAME                     Text   CH1
@WORDSIZE                             Text   2
ATTR                          Byte   DATA:0039
CODE_SIZE                     Number 0107
CONTINUE                      Near   TEXT:006C
DATA_SIZE                     Number 0078
GDT_CODE                      Struct DATA:0010 DESCR
GDT_DATA                      Struct DATA:0008 DESCR
GDT_NEW                               Struct DATA:0020 DESCR
GDT_NULL                      Struct DATA:0000 DESCR
GDT_SCREEN                            Struct DATA:0028 DESCR
GDT_SIZE                      Number 0030
GDT_STACK                     Struct DATA:0018 DESCR
GO                            Near   TEXT:00C5
MAIN                          Near   TEXT:0000
MEGA                          Near   TEXT:00F0
MES                           Byte   DATA:003A
MES_PR                                Byte   DATA:0058
PAYZA                                 Near   TEXT:0093
PDESCR                                Qword          DATA:0030
RETURN                                Near   TEXT:00D4
SCREEN                                Near   TEXT:0084
SYM                           Byte   DATA:0038
```

Structure Name	Type	Offset		
DESCR				
LIM	Word	0000		
BASE_1	Word	0002		
BASE_M	Byte	0004		
ATTR_1	Byte	0005		
ATTR_2	Byte	0006		
BASE_H	Byte	0007		

Groups & Segments	Bit	Size	Align	Combine	Class
DATA	16	0078	Para	none	
STK	16	0100	Para	Stack	STACK
TEXT	16	0107	Para	none	CODE

Выполнение задания

1) Вставьте в текст программы строки, которые будут пытаться в защищенном режиме с помощью функции 9 прерывания DOS 21h вывести на экран сообщение « Находимся в защищенном режиме».

Выполнение такой программы вызывает перезагрузку компьютера. Так как в защищенном режиме прерывания обрабатываются несколько другим образом чем в реальном режиме. Самая очевидная ошибка в данном случае – это то, что мы не сформировали IDT. По этой причине мы запрещаем на время выполнения программы все аппаратные прерывания, так как первое же прерывание от таймера может завесить компьютер.

2) Вставьте в программу строки загрузки в сегментный регистр, например GS, селектора 48. Запишите в отчет что получится.

В этом случае компьютер также перезагружается. В программе определены 6 дескрипторов (размер GDT определяет поле <предел> в GDTR), а селектор 48 заставит обращаться к седьмому. Происходит исключение, а дескрипторная таблица прерываний не сформирована.

3) Внесите в программу изменения в соответствии с индивидуальным вариантом задания. Отладьте программу. Занесите в отчет выводимую программой на экран информацию. Объясните содержимое выводимой битовой строки
   a) Включить в **заданную** строку таблицы GDT дескриптор, описывающий сегмент данных **заданного** размера, начиная с **заданного** адреса линейного адресного пространства процессора.

   gdt_new descr <768,0ffffh,0ffh,92h,80h,7fh>

   Адрес второй половины адресного пространства начинается с адреса 7FFFFFFFH (2Гб-1байт т.к. адресация с 0)
   768 это 3*1Мб/4Кб делим на 4кб, т.к. размер сегмента превышает 1 мб,

соответственно бит гранулярности надо установить в 1 (80h) и задавать размер в страницах. А размер страницы равен 4кб.

b) После перехода в защищенный режим программа должна вывести на экран монитора символы **заданного** столбца таблицы ASCII **заданным** цветом, на **заданном** фоне, начиная с **заданной** позиции на экране, **заданным** способом.

```
...
sym       db        40h      ; символ
attr      db        0dh    ; его атрибут
...
; инициализируем ES и выводим символы
        mov    ax,40          ;селектор сегмента видеобуфера
        mov    es,ax          ;
        mov    bx,160*3+80    ;начальное смещение на экране: 40 столбец, 3
строка
        mov    cx,0fh         ;число выводимых символов
        mov    ax,word ptr sym  ;начальный символ с атрибутом
screen:   mov    es:[bx],ax     ;вывод в видеобуфер
        add    bx,162         ;переходим на строку ниже и на один столбец правее
        inc    ax             ;следующий символ
        push   cx             ;пауза
        mov    ecx,0fffffh    ;для
payza:    db 67h              ;наглядности
        loop   payza
        pop    cx
        loop   screen         ;цикл вывода на экран
```

c) После возврата в реальный режим программа должна выполнить анализ разряда A дескрипторов сегментов памяти из таблицы GDT, и вывести на экран монитора строку 0 и 1, отражающую содержимое бита A дескрипторов GDT.

```
mov bx,offset gdt_null  ; начнем анализ с бита A нулевого дескриптора
                mov cx,6; у нас анализируется 6 дескрипторов
mega:
                mov dl,[bx].attr_1;читаем  атрибуты, содержащие бит A
                mov ah,02h;ф-я вывода на экран символа
                and dl,1;выделяем из байта атрибутов бит A
                add dl,30h;
                int 21h;выводим символ
                add bx,8;переходим к следующему дескриптору
loop mega
```

В итоге получаем следующую строку:011101

- это значит(слева направо):

*к сегменту, описываемому нулевым дескриптором обращений не было
* к сегменту данных были обращения (например, mov ax,word ptr sym)
*к сегменту кода были обращения
*к сегменту стека были обращения (например, push cx)
*к сегменту, описанному мной по заданию, обращений не было
*к сегменту, определяющему видеопамять было обращение (screen: mov es:[bx],ax)

4)Модифицируйте программу таким образом, чтобы возврат в реальный режим осуществлялся через сброс процессора. Для этого используйте файл lab1_1.asm, содержащий необходимый фрагмент программы. Добейтесь правильной работы программы.

```
.386p                    ; разрешение команд 386 и 486
;  Структура для описания дескрипторов сегментов
descr struc
lim        dw        0      ;предел (биты 15-0)
base_1     dw        0      ;база (биты 15-0)
base_m     db        0      ;база (биты 23-16)
attr_1     db        0      ;байт атрибутов 1
attr_2     db        0      ;предел (биты 19-16)
;                           и атрибуты 2
base_h     db        0      ;база (биты 31-24)
descr      ends

data       segment  use16     ; начало сегмента данных
; Таблица глобальных дескрипторов GDT
gdt_null   descr      <0,0,0,0,0,0>                ;нулевой дескриптор
gdt_data   descr      <data_size-1,0,0,92h,0,0>    ;дескр сег данн
gdt_code   descr      <code_size-1,0,0,98h,0,0>    ;дескр сег кода
gdt_stack  descr      <255,0,0,92h,0,0>            ;дескр сег стека
gdt_new            descr          <768,0,0,92h,80h,80h>
gdt_screen  descr     <4095,8000h,0bh,92h,0,0>     ;дескр видеобуфера
gdt_size=$-gdt_null
; Поля данных программы
pdescr     dq        0      ; псевдодескриптор
sym        db        40h    ; символ
attr       db        0dh    ; его атрибут
mes        db ' Вернулись в реальный режим  $';
mes_pr     db ' Находимся в защищенном режиме $';
data_size=$-gdt_null        ; размер сегмента данных
data       ends             ; конец сегмента данных
text       segment  'code' use16  ;16-разрядный код
;
           assume   cs:text,ds:data
main       proc
           xor      eax,eax
           mov      ax,data
           mov      ds,ax   ; адрес сегмента данных
; вычислим 32-битовый линейный адрес сегмента данных и загрузим его
; в дескриптор сегмента данных в GDT
           shl      eax,4
           mov      ebp,eax
           mov      bx,offset gdt_data
           mov      [bx].base_1,ax
           rol      eax,16
           mov      [bx].base_m,al
; аналогично для линейного адреса сегмента команд
           xor      eax,eax
           mov      ax,cs
           shl      eax,4
```

```
        mov     bx,offset gdt_code
        mov     [bx].base_1,ax
        rol     eax,16
        mov     [bx].base_m,al
; аналогично для линейного адреса сегмента стека
        xor     eax,eax
        mov     ax,ss
        shl     eax,4
        mov     bx,offset gdt_stack
        mov     [bx].base_1,ax
        rol     eax,16
        mov     [bx].base_m,al
; подготовим псевдодескриптор и загрузим GDTR
        mov     dword ptr pdescr+2,ebp      ; база GDT,биты 31-0
        mov     word ptr pdescr,gdt_size-1  ; предел GDT
; Начало текста программы совпадает с программой lab1.asm

; Загрузим регистр GDTR
        lgdt    pdescr
; Подготовимся к возврату из защищенного режима в реальный
        mov     ax,40h    ; настроим ES на область
        mov     es,ax     ; данных BIOS
        mov     word ptr es:[67h],offset return ; смещение возврата
        mov     word ptr es:[69h],cs            ; сегмент возврата
; Подготовимся к переходу в защищенный режим
        cli               ; запрет аппаратных прерываний
        mov     al,8fh    ; запрет NMI (80h) и выборка
                          ; байта состояния отключения 0fh
        out     70h,al    ; порт КМОП микросхемы
        jmp     $+2       ; задержка
        mov     al,0ah    ; установим режим восстановления
        out     71h,al    ; после сброса процессора
; Переходим в защищенный режим
        smsw    ax        ; запишем слово состояния машины в ax
        or      ax,1      ; слово сост машины есть младшая половина CR0
        lmsw    ax
; Теперь процессор работает в защищенном режиме
;
; далее текст программы совпадает с текстом lab1.asm

; загружаем в CS:IP селектор: смещение точки continue
; и заодно очищаем очередь команд
        db      0eah            ;код команды jmp far
        dw      offset continue ; смещение
        dw      16              ; селектор
continue:
; адресуем данные
        mov     ax,8            ;селектор сегмента данных
        mov     ds,ax           ;
; адресуем стек
        mov     ax,24           ;селектор сегмента стека
        mov     ss,ax           ;
```

```
; инициализируем ES и выводим символы
        mov     ax,40           ;селектор сегмента видеобуфера
        mov     es,ax           ;
        mov     bx,160*3+80     ;начальное смещение на экране
        mov     cx,0fh          ;число выводимых символов
        mov     ax,word ptr sym  ;начальный символ с атрибутом
screen:     mov     es:[bx],ax      ;вывод в видеобуфер
        add     bx,162          ;смещение в видеобуфере
        inc     ax              ;следующий символ
        push    cx              ;пауза
        mov     ecx,0fffffh     ;для
payza:      db 67h              ;наглядности
        loop    payza
        pop     cx
loop    screen
        mov  ah,09h
        mov  dx,offset mes_pr
        int  21h

; Вернемся в реальный режим
        mov     al,0feh     ; команда сброса процессора
        out     64h,al      ; в порт 64h
        hlt                 ; останов процессора
;---------------------------------------------------------------
; Теперь процессор снова работает в реальном режиме
;---------------------------------------------------------------
; Теперь процессор снова работает в реальном режиме
;---------------------------------------------------------------
;Восстановим операционную среду реального режима
return:    mov  ax,data
        mov  ds,ax
        mov  ax,stk
        mov  ss,ax
; Разрешим аппаратные и немаскируемые прерывания
        sti         ; разрешение прерываний
        mov  al,0    ; сброс бита 7 в порте КМОП-
        out  70h,al  ; разрешение NMI
; Проверяем выполнение функций ДОС после возврата в реальный режим
        mov  ah,09h
        mov  dx,offset mes
        int  21h
;---------------------------------------------------------------
                mov bx,offset gdt_null
                mov cx,6
mega:
                mov dl,[bx].attr_1
                mov ah,02h
                and dl,1
                add dl,30h
                int 21h
                add bx,8
loop mega
```

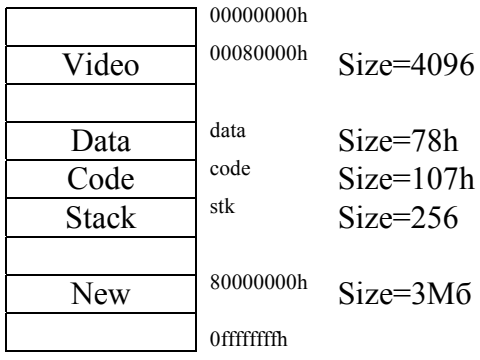
```
;-------------------------------------------------------------------
        mov  ax,4c00h  ; завершим программу обычным образом
        int  21h
main      endp
code_size=$-main        ;размер сегмента кода
text      ends
stk       segment   stack 'stack' use16
        db       256 dup ('^')
stk       ends
        end main; переход в защищенный режим из реального и наоборот
```

5) Вставьте в текст модифицированной программы строки, которые будут пытаться в защищенном режиме с помощью функции 9 прерывания DOS 21h вывести на экран сообщение « Находимся в защищенном режиме». Запишите в отчет что получится. Попробуйте дать объяснение.

В данном случае компьютер не выводит строку и не перезагружается как было в первом случае, хотя IDT все равно не сформирована. Анализ показывает, что исключение происходит, и процессор все равно перезагружается, но так мы записали адрес возврата в реальный режим в область данных BIOS по адресу 0040h:0067h. И обеспечили после выполнения сброса процессора передачу управления по адресу, подготовленному нами, то получается просто «досрочный» выход в реальный режим. А выполнение программы продолжается с места:

```
...
--> return:    mov  ax,data
          mov  ds,ax
          mov  ax,stk
....
```

Примерное расположение сегментов, описанных в GDT в линейном адресном пространстве



Заключение

Работа в реальном режиме достаточно проста, но имеет множество недостатков. Основной – это то, что нельзя адресовать больше одного мегабайта ОП. К тому же нет контроля прав доступа к различным областям памяти. Такой подход возможен при создании однопользовательских систем, но кгда речь идет о мультизадачных системах повышенной надежности, работать в таком режиме становиться не просто невозможно, а опасно.

Защищенный режим работы дает программисту больше возможностей (возможность использования до 4Гб адресного пространства, сегмент может начинаться с “произвольного адреса”, контроль прав доступа, максимальный размер сегмента 4Гб, минимаьный – 1 байт и т.д.) – в современных условиях все это делает необходимым освоение данного раздела .