ФГБОУ ВО “Чувашский государственный университет им. И. Н. Ульянова” Факультет: ИВТ
Кафедра: Вычислительной техники
Предмет: Электронно-вычислительная машина и периферийные устройства

Лабораторная работа №6

Защищённый режим микропроцессоров intel X86

Вариант №6

Выполнил: студент группы ИВТ-41-20
Галкин Дмитрий
Проверил: доцент Андреева А.А.

Чебоксары 2022 г.

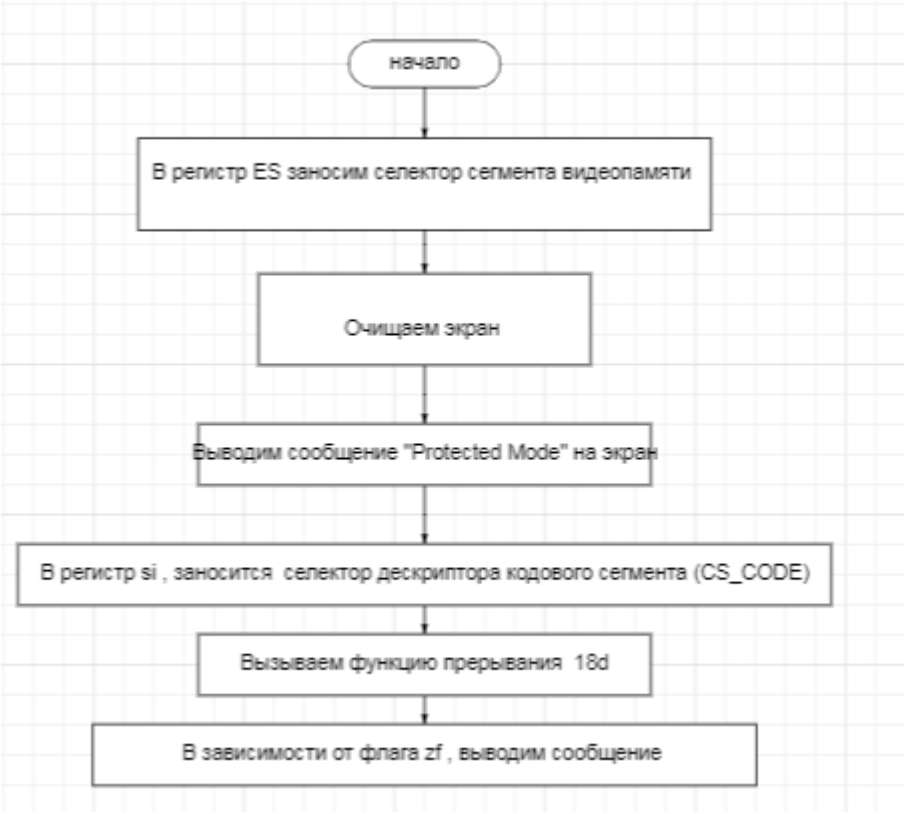
Цель работы

Изучить и закрепить на практике теоретические сведения по защищенному режиму микропроцессоров intel x86, написать обработчик ошибки переполнения.

Текст задания

Написать обработчик прерывания INT 18d, выводящий на экран байт прав доступа заданного дескриптора в двоичной форме. Селектор дескриптора задается в качестве параметра при вызове этой функции. Для получения байта прав доступа можно воспользоваться командой LAR

Общая схема задания



Текст программы

```
;       Написать обработчик прерывания INT 18d,
; выводящий на экран байт прав доступа заданного дескриптора в двоичной форме.
; Селектор дескриптора задается в качестве параметра при вызове этой функции.
; Для получения байта прав доступа можно воспользоваться командой LAR.
```

; Рассмотрим работу программы:
; - Формируем в памяти таблицы GDT и IDT. GDT содержит деск-
;риптор, описывающий саму таблицу GDT; дескриптор, описывающий
;сегмент программы как сегмент кода; дескриптор, описывающий сег-
;мент программы как сегмент данных; дескрипторы, описывающие таб-
;лицу IDT в реальном и в защищенном режимах; дескриптор, описыва-
;ющий сегмент текстового видеобуфера. Таблица IDT содержит 16оаф
;шлюзов ловушек, ссылающихся на обработчики исключительных ситу-
;аций, 22 пустых дескриптора и 1 шлюз прерывания, который ссыла-
;ется на процедуру вывода строки на экран. Такая структура таб-
;лицы IDT приведена в качестве примера использования прерываний.
; - Запрещаем прерывания.
; - Открываем адресную линию A20.
; - Формируем в памяти адреса для косвенных межсегментных
;переходов, которые нужно будет выполнить, сразу после переклю-
;чения режимов работы процессора, для загрузки CS новым содержи-
;мым и очистки очереди команд.
; - Загружаем регистры GDTR и IDTR.
; - Переключаемся в защищенный режим, устанавливая бит 0 ре-
;гистра CR0.
; - Выполняем косвенный межсегментный переход по адресу,
;сформированному в памяти (Protect_Jump). В результате в регистр
;CS заносится селектор дескриптора кодового сегмента (CS_CODE),
;и управление передается на метку Protect.
;

```
;    - Заносим в сегментные  регистры DS, ES и SS селектор деск-
;риптора сегмента данных.
;    - Вызываем рабочую процедуру,  выполняющую очистку экрана и
;вывод сообщения, причем сообщение  выводится при помощи  вызова
;через шлюз прерывания функции  вывода строки  (присутствующей в
;том же сегменте).
;    - Запрещаем прерывания.
;    - Переключаемся в реальный режим, очищая бит 0 регистра CR0.
;    - Выполняем  косвенный  межсегментный  переход  по  адресу,
;сформированному в памяти (Real_Jump). В результате в регистр CS
;заносится сегментный адрес программы и управление передается на
;метку Real.
;    - Загружаем регистр IDTR значением, необходимым для адреса-
;ции таблицы прерываний реального режима (База = 0000:0000, гра-
;ница = 3FFh).
;    - Восстанавливаем сегментные регистры.
;    - Закрываем адресную линию A20.
;    - Разрешаем прерывания.
;    - Передаем управление в DOS.
;
;Программа транслируется в COM - файл:
; TASM Code.asm
; Tlink Code.obj /t
; Code.asm
.386p                           ; Разрешение трансляции
                                ; всех инструкций 80386
Gdt_Descriptor  STRUC           ; Шаблон дескриптора GDT
 Seg_Limit      dw   0          ; Длина сегмента
 Base_Lo_Word   dw   0          ; Младшие 16 бит базового
                                ; адреса
 Base_Hi_Byte   db   0          ; Биты 16..23 базового адр.
 Acces_Rights   db   0          ; Байт прав доступа
 Base_Top_Byte  dw   0          ; Биты 24..31 базового адр.
Gdt_Descriptor  ENDS
Idt_Descriptor  STRUC           ; Шаблон дескриптора IDT
 Int_Offset     dw   0          ; Точка входа в процедуру
                                ;  обработки прерывания
```

```
 Int_Selector    dw   0            ; Селектор сегмента в GDT
                 db   0            ;
 Access          db   0            ; Права доступа
                 dw   0            ;
Idt_Descriptor  ENDS

Code_Seg_Access  Equ  10011011b    ; Байт прав  доступа деск-
                                   ; риптора сегмента кода
Data_Seg_Access  Equ  10010011b    ; Байт прав  доступа деск-
                                   ; риптора сегмента данных
Disable_Bit20    Equ  11011101b    ; Код команды 8042 для за-
                                   ; крывания линии A20
Enable_Bit20     Equ  11011111b    ; Код команды 8042 для от-
                                   ; крывания линии A20
Port_A           Equ  060h         ; Порт A 8042
Status_port      Equ  064h         ; Порт состояния 8042
Cmos_Port        Equ  070h         ; Адрес порта CMOS-памяти
; Макро для записи базового адреса сегмента в дескриптор
FILLDESCR MACRO   Seg_Addr,Offset_Addr,Descr
        xor     edx,edx             ; EDX := 0
        xor     ecx,ecx             ; ECX := 0
        mov     dx,Seg_Addr         ; Сегментная часть
        mov     cx,offset Offset_Addr  ; Смещение
        call    Form_32Bit_Address  ; CX:DX := линейный
                                    ; адрес
        mov     &Descr.Base_Lo_Word,dx  ; Занесение базового
        mov     &Descr.Base_Hi_Byte,cl  ; адреса в дескрип-
        mov     &Descr.Base_Top_Byte,cx ; тор
       ENDM

CSEG      SEGMENT  Para USE16 public 'code'
        ASSUME  cs:Cseg,ds:Cseg
        ORG     100h
Start:    jmp     Main

; Глобальная дескрипторная таблица GDT
 EVEN
```

```
Gdt    label   word
Gdt_nil       EQU $-gdt
Gdt0          Gdt_Descriptor < >

;******Дескриптор, описывающий таблицу Gdt как сегмент данных *
Gdt_Desc      EQU $-gdt          ; Селектор дескриптора
Gdt1          Gdt_Descriptor <gdt_leng,,,data_seg_access,>

;****** Дескриптор, описывающий сегмент Cseg как кодовый ******
Cs_Code       EQU $-gdt          ; Селектор дескриптора
Gdt2          Gdt_Descriptor<cseg_leng,,,code_seg_access,>

;** Дескриптор, описывающий Cseg как сегмент данных с пределом*
;** 0FFFEh. Он будет использоваться также в роли стекового. ***
Cs_Data       EQU $-gdt          ; Селектор дескриптора
Gdt3          Gdt_Descriptor<cseg_leng,,,data_seg_access,>

;************** Дескриптор, описывающий таблицу IDT **************
Idt_Pointer   Gdt_Descriptor<idt_leng-1,,,data_seg_access>

;** Дескриптор, описывающий таблицу IDT реального режима *******
Idt_Real      Gdt_Descriptor<3FFh,,,data_seg_access>

;********* Дескриптор, описывающий сегмент видеопамяти *********
Video_Desc    EQU $-gdt          ; Селектор дескриптора
GdtB800       Gdt_Descriptor<1000h,8000h,0bh,data_seg_access>

Gdt_Leng      EQU $-gdt          ; Длина таблицы GDT

;Таблица дескрипторов прерываний IDT.
EVEN
Idt   label   word
ex0   Idt_Descriptor<offset ex0_proc,cs_code,0,10000111b,0>
ex1   Idt_Descriptor<offset ex1_proc,cs_code,0,10000111b,0>
ex2   Idt_Descriptor<offset ex2_proc,cs_code,0,10000110b,0>
ex3   Idt_Descriptor<offset ex3_proc,cs_code,0,10000111b,0>
ex4   Idt_Descriptor<offset ex4_proc,cs_code,0,10000111b,0>
ex5   Idt_Descriptor<offset ex5_proc,cs_code,0,10000111b,0>
ex6   Idt_Descriptor<offset ex6_proc,cs_code,0,10000111b,0>
ex7   Idt_Descriptor<offset ex7_proc,cs_code,0,10000111b,0>
```

```
ex8     Idt_Descriptor<offset ex8_proc,cs_code,0,10000111b,0>
ex9     Idt_Descriptor<offset ex9_proc,cs_code,0,10000111b,0>
ex10    Idt_Descriptor<offset ex10_proc,cs_code,0,10000111b,0>
ex11    Idt_Descriptor<offset ex11_proc,cs_code,0,10000111b,0>
ex12    Idt_Descriptor<offset ex12_proc,cs_code,0,10000111b,0>
ex13    Idt_Descriptor<offset ex13_proc,cs_code,0,10000111b,0>
ex14    Idt_Descriptor<offset ex14_proc,cs_code,0,10000111b,0>
ex15    Idt_Descriptor<offset ex15_proc,cs_code,0,10000111b,0>
ex16    Idt_Descriptor<offset ex16_proc,cs_code,0,10000111b,0>
        Idt_Descriptor <>
Int18   Idt_Descriptor<offset int18_proc,cs_code,0,10000110b,0>                 ;;;;;

                Idt_Descriptor 20 dup(<>)
               ;;;;;

Int39   Idt_Descriptor<offset int10_proc,cs_code,0,10000110b,0>
 Idt_Leng       EQU $-Idt           ; Длина таблицы IDT

Real_Jump       dd  ?           ; Адрес межсегментного
                                ; перехода в реальном режиме
Protect_Jump    dd  ?           ; Адрес межсегментного пере-
                                ; хода в защищенном режиме
Mess            db  'Protected Mode$'

MessError               db  'Error ZF = 0$'
        ;;;;;
MessAr                          db  'AR = 00000000b$'
               ;;;;;

Len             dw  14d
Gate_Failure    db "Error open A20$"

Main:   FillDescr  cs,Gdt,Gdt1    ; Формирование 32-разряд-
                                ; ного адреса из CS:GDT и
                                ; запись его в дескриптор
                                ; с номером Gdt_Desc.
        FillDescr  cs,0,gdt2    ; Дескриптор Cs_Code ука-
                                ; зывает на CSEG как на
                                ; кодовый сегмент.
        FillDescr  cs,0,gdt3    ; Дескриптор Cs_Data ука-
                                ; зывает на CSEG как на
                                ; сегмент данных.
    FillDescr  cs,Idt,Idt_Pointer ; Дескриптор Idt_Pointer
                                ; указывает на IDT.
        cli                     ; Запрет прерываний
        mov     al,8fh          ; Запрет немаскируемых
        out     cmos_port,al    ; прерываний
        jmp     short $+2
```

```
        mov     al,5
        out cmos_port+1,al

        mov     ah,Enable_Bit20       ; Открываем адрес-
        call    Gate_A20              ; ную линию A20
        or      al,al                 ; Если произошла
        jz      A20_Opened            ; ошибка, то
        mov     dx,offset Gate_Failure ; выдать сообщение
        mov     ah,9                  ; на экран, разре-
        int     21h                   ; шить прерывания и
        sti                           ; вернуться в DOS
        int     20h
A20_Opened:
      ;  lea     di,Real_Jump         ; Формирование адреса
      ;  mov     word ptr [di],offset Real ; для перехода
      ;  mov     word ptr [di+2],cs       ; в реальный режим
      ;  lea     di,Protect_Jump      ; Формирование адреса
      ;  mov     word ptr [di], offset Protect ; для перехода
      ;  mov     word ptr [di+2],Cs_Code      ; в защищенный
                                       ; режим
        lea     di,Real_CS
        mov     word ptr cs:[di],cs

        lgdt    Gdt1            ; Загрузка GDTRF
        lidt    Idt_Pointer     ; Загрузка IDTR
        mov     eax,cr0         ; Переходим в защищенный
        or      eax,1           ; режим, устанавливая
        mov     cr0,eax         ; бит 0 в регистре CR0
    ;   jmp     dword ptr Protect_Jump ; Переход на метку
                                ; Protect
        db 0EAh
        dw offset Protect
        dw CS_Code
; Работа в защищенном режиме.
Protect:  mov     ax,Cs_Data
        mov     ss,ax           ; Регистры DS, ES и SS
        mov     ds,ax           ; содержат  селектор
        mov     es,ax           ; сегмента Cs_Data
        call    My_Proc         ; Вызов рабочей процедуры
        cli
        mov     eax,cr0         ; Переходим в реальный
        and     eax,0FFFFFFFEh  ; режим, сбрасывая бит 0
        mov     cr0,eax         ; регистра CR0
```

```
;       jmp     dword ptr Real_Jump ; Косвенный межсегмент-
                        ; ный переход на метку
                        ; Real
        db 0EAh
        dw offset Real
Real_CS    dw 0
; Работа в реальном режиме.
Real:     lidt    Idt_Real        ; Загружаем регистр IDTR
                        ; для работы в реальном
                        ; режиме
        mov     dx,cs           ; Восстанавливаем
        mov     ds,dx           ; сегментные
        mov     ss,dx           ; регистры
        mov     ah,Disable_Bit20  ; Закрытие адресной
        call    Gate_A20        ; линии A20
        sti                     ; Разрешение прерываний
        int     20h             ; Выход в DOS

ex0_proc:  iret                 ; Обработчики особых
ex1_proc:  iret                 ; ситуаций
ex2_proc:  iret                 ; Здесь установлены
ex3_proc:  iret                 ; заглушки вместо
ex4_proc:  iret                 ; обработчиков
ex5_proc:  iret
ex6_proc:  iret
ex7_proc:  iret
ex8_proc:  iret
ex9_proc:  iret
ex10_proc: iret
ex11_proc: iret
ex12_proc: iret
ex13_proc: iret
ex14_proc: iret
ex15_proc: iret
ex16_proc: iret

;************************************************************
;Управление прохождением сигнала A20
;ВХОД: (AH)=0DDH   установить A20 всегда равным нулю
;      (AH)=0DFh   открыть адресный разряд A20
```

```
;ВЫХОД: (AL)=0     8042 принял команду
;       (AH)=2     сбой
;*****************************************************************
Gate_A20 PROC
        cli                     ; Запрет прерываний
        call    Empty_8042
        jnz     Gate_1
        mov     al,0d1h         ; Выдаем команду  8042 для
        out     Status_Port,al    ; записи в выходной порт
        call    Empty_8042
        jnz     Gate_1
        mov     al,ah           ; Записываем в порт A 8042
        out     Port_A,al         ; код команды
        call    Empty_8042
Gate_1:    ret
Gate_A20 ENDP
;*****************************************************************
;Ждать пока буфер 8042 не опустеет
;Вход: нет
;Выход:(AL)=0   буфер пуст
;       (AL)=2   не пуст
;*****************************************************************
Empty_8042  PROC
        push    cx
        xor     cx,cx           ; CX = 0 (256 повторений)
Empty_1:   in      al,Status_Port    ; Порт 8042
        and     al,00000010b      ; Бит 2 очищен ?
        loopnz  Empty_1
        pop     cx
        ret
Empty_8042  ENDP
;*****************************************************************
; Формирование 32-разрядного адреса
; Вход : CX:DX - адрес в формате <сегмент:смещение>
; Выход: CX:DX - 32-разрядный линейный адрес
Form_32Bit_Address   PROC
        shl     edx,4
```

```
        add     edx,ecx
        mov     ecx,edx
        shr     ecx,16
        ret
Form_32Bit_Address   ENDP

;*************************************************************
;   Процедура вывода строки на экран, работает в качестве
;   обработчика прерывания.
; Вход : DS:BX - адрес сообщения
;        DL - строка экрана
;        DH - колонка экрана
;*************************************************************
Int10_Proc   Proc  Near             ; Обработчик прерывания
        pusha                   ; INT 39d
        xor     cx,cx           ; Очистка CX
        mov     cl,dh           ; CL = колонка
        sal     cl,1            ; CL = CL*2
        xor     dh,dh           ; DX = строка
        imul    dx,160d    ; Умножаем на число байт в строке
        add     dx,cx      ; Прибавляем смещение в строке
                        ; Результат: DX = смещение в
                        ; видеопамяти
        push    Video_Desc
        pop     es         ; ES = сегмент видеопамяти
        mov     di,dx      ; DI = смещение в этом сегменте
m:        mov     ax,[bx]    ; AL = очередной символ строки
        cmp     al,'$'     ; Конец строки ?
        jz      Ex         ; Да - выход
        mov     cx,es:[di] ; Получить атрибут в CH
        mov     ah,ch      ; AX = символ с атрибутом
        stosw              ; Записать символ в видеопамять
        inc     bx         ; Перейти к следующему символу
        jmp     short   m
ex:       popa
        iret               ; Возврат из прерывания
Int10_Proc Endp
```

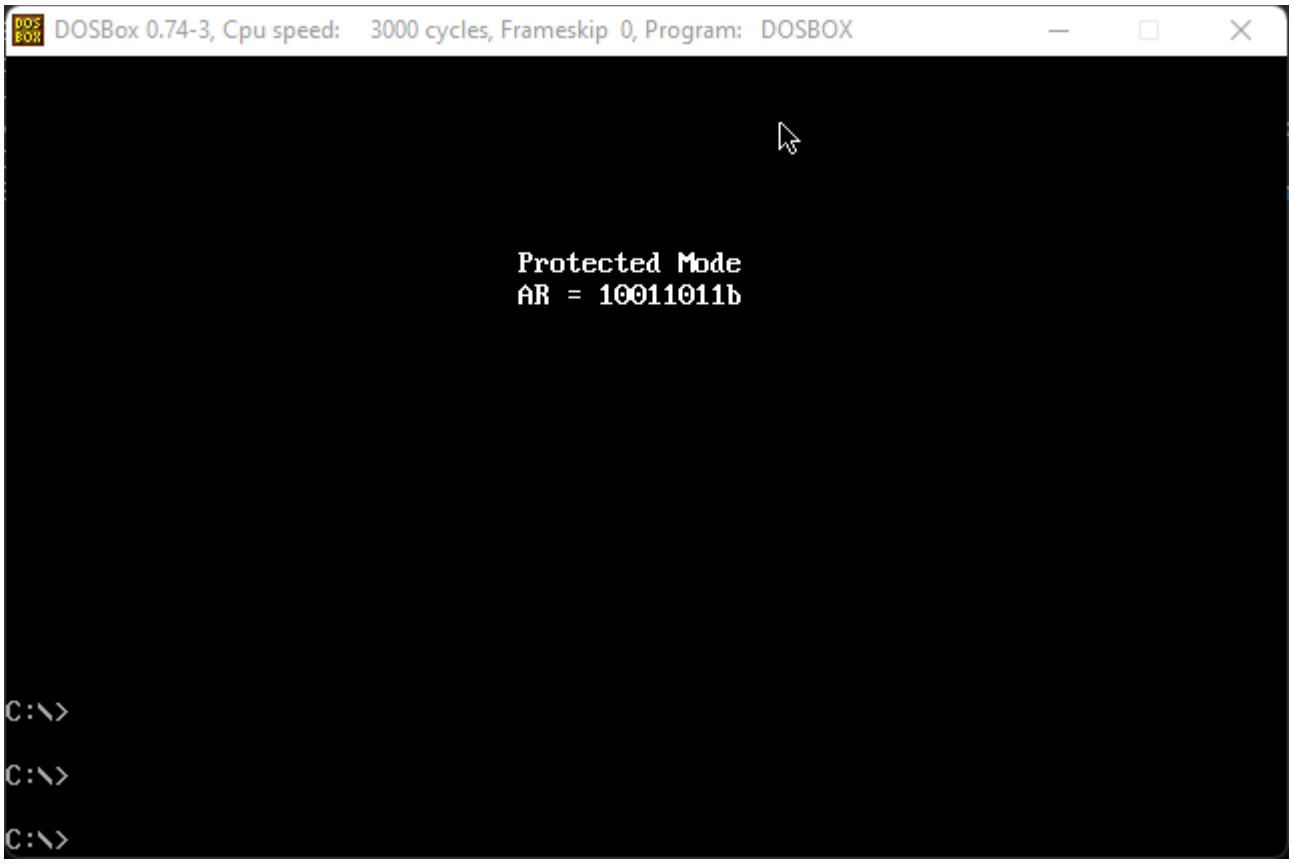
```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Int18_proc      Proc    Near                    ; обработчик прерывания
                lar     di,si                   ; загрузить байт прав доступа
                        ; дескриптора(устанавливает zf)
                jnz     outError        ; проверить zf
                mov cx, 8                               ; счетчик байт = 8 бит
                lea bx, MessAr          ; исходное сообщение
                add bx, 5               ; смещение для изменения
                mov ax, 15              ; байт прав доступа с 15 по 7 бит
start1:
                bt  di, ax              ; проверка бита(установка cf)
                mov dl,'0'              ; заносим символ 0 чтобы не было рандомного значения
                adc dl, 0               ; прибавляем флаг cf
                mov [bx],dl             ; заменяем символ на символ dl
next1:    ; выполенние команды int18 в защищеном режиме
                dec ax                  ;
                inc bx                  ;
                loop start1             ; повторять с 15 по 7 бит дескриптора
        lea bx, MessAr                  ; загрузить измененную строку
                jmp Print
outError:
        lea bx,MessError        ; загрузить сообщение об ошибке
Print:
                mov  dx,200Ch           ; Координаты вывода
                int  39d                                ; вывод
                iret                    ; возврат
                Int18_proc endp

;*************************************************************
;Процедура выполняющая какие-либо действия в защищенном режиме
;*************************************************************
MY_PROC    PROC
        pusha
        push    es
        push    Video_Desc      ; В регистр ES заносим се-
        pop     es              ; лектор сегмента видеопа-
                        ; мяти
        mov     dh,0fh          ; Очищаем экран
        call    Paint_Screen
        mov     ax,Cs_Data
        mov     ds,ax           ; DS - сегмент данных
        lea     bx,Mess         ; Адрес сообщения
        mov     dx,200Bh        ; Координаты вывода
        int     39d             ; Вывод строки на экран
                mov     si, Cs_Code     ; ;;;;;
                mov si, 50*8                    ;


                int     18d                             ; ;;;;;
```

```
        pop     es
        popa
        ret
MY_PROC ENDP
;*************************************************************
; Процедура очищает экран и устанавливает цвета в соответствии
; с заданным атрибутом.
; Вход : ES - селектор дескриптора текстового видеобуфера
;        DH - атрибут.
;*************************************************************
PAINT_SCREEN  PROC
        push    cx si di es
        mov     cx,80*25        ; Размер видеопамяти (слов)
        xor     si,si           ; SI и DI установим на
        xor     di,di           ; начало видеопамяти
Paint1:    lodsw                ; Увеличиваем смещение в
                                ; видеопамяти
        mov     ah,dh           ; Байт атрибута символа
        mov     al,20h          ; Код символа "ПРОБЕЛ"
        stosw                   ; Записываем символ с ат-
                                ; рибутом в видеопамять
```

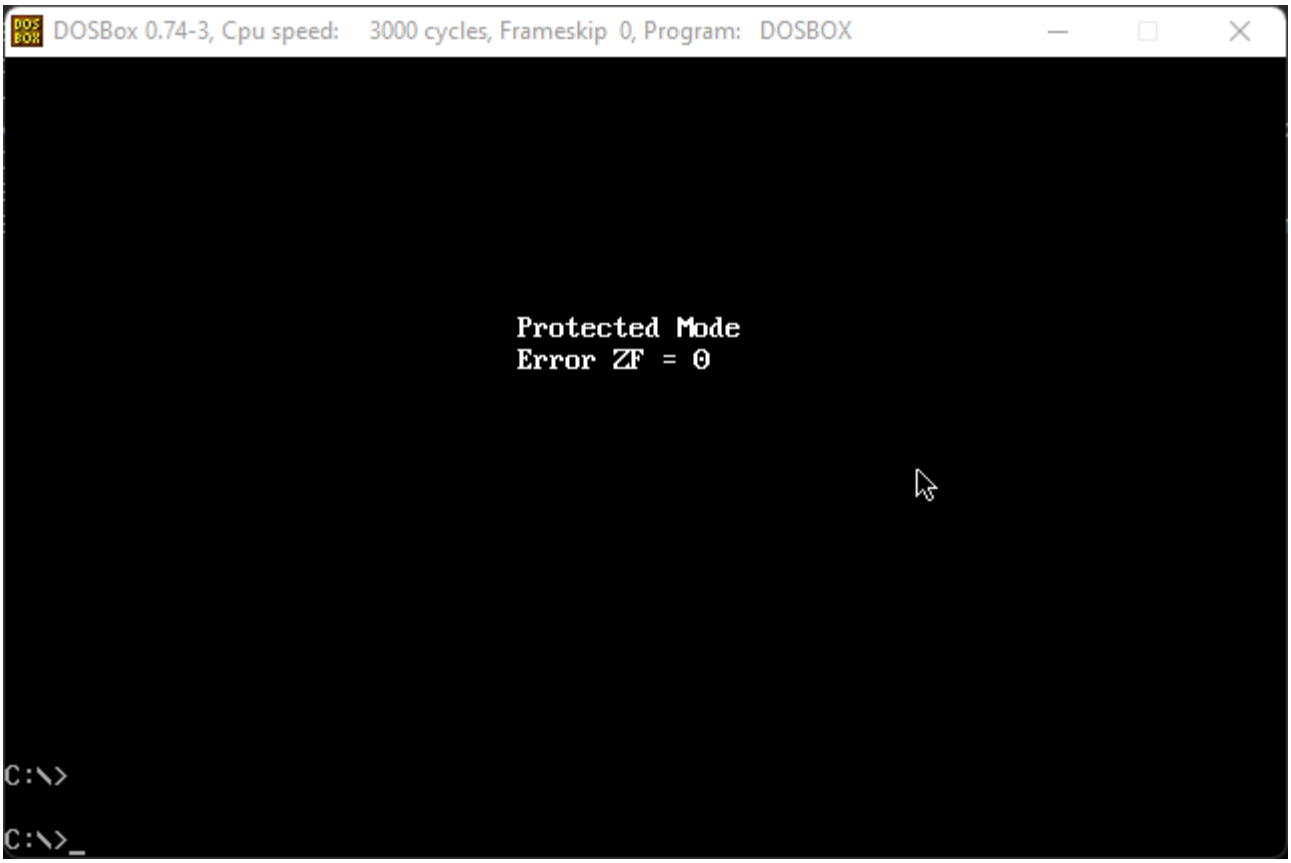
```
        loop    Paint1          ; Повторить для каждого
                                ; символа на экране
        pop     es di si cx
        RET
PAINT_SCREEN ENDP
Cseg_Leng  Equ     $            ; Длина сегмента Cseg
Cseg       Ends
End        Start
```

Пример работы программы

Результат выполнения программы



Результат выполнения программы с ошибкой



Вывод

В процессе выполнения лабораторной работы были изучены и закреплены на практике теоретические сведения по защищенному режиму микропроцессоров intel x86, написан обработчик ошибки переполнения.