Федеральное государственное бюджетное образовательное учреждение высшего

образования

«Чувашский государственный университет им. И.Н. Ульянова»

Кафедра вычислительной техники

# **Лабораторная работа 6.**

**ЗАЩИЩЕННЫЙ РЕЖИМ МИКРОПРОЦЕССОРОВ INTEL X86**

Вариант 9

Выполнил: Иванов В.С.

студент группы ИВТ-41-22

Проверила:

Доцент Андреева А.А.

Чебоксары, 2024

Цель работы: изучение работы в защищенном режиме и использование полученных знаний на практике.

Задание: написать обработчик ошибки неверного кода операции, выводящий информацию на экран. Для проверки вызвать генерацию данной особой ситуации.  
**Общая схема решения задачи**

1. **Определение структуры дескрипторов**:

Создание структур для глобальной дескрипторной таблицы (GDT) и таблицы прерываний (IDT). Эти структуры содержат информацию о сегментах и обработчиках прерываний.

1. **Инициализация дескрипторов**:

Инициализация дескрипторов GDT и IDT, включая установку селекторов и прав доступа. Это необходимо для правильной работы сегментов и обработки прерываний.

1. **Настройка обработчиков прерываний**:

Определение обработчиков для различных исключений, включая обработчик для ошибки неверного кода операции. Программа создает соответствующий дескриптор в IDT, указывая адрес обработчика.

1. **Генерация ошибки**:

Вызов недопустимой инструкции (ud2), чтобы сгенерировать ошибку неверного кода операции. Это позволяет протестировать работу обработчика.

1. **Обработка исключения**:

Когда возникает ошибка, управление передается в обработчик, который выводит сообщение об ошибке на экран.

1. **Возврат из обработчика**:

После вывода сообщения обработчик завершает свою работу и возвращает управление обратно в программу с помощью инструкции iret.

Текст программы:  
;Программа транслируется в COM - файл:

; TASM demo.asm

; Tlink demo.obj /t

; Demo.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

Limit&Acces db 0 ; +

Base\_Top\_Byte db 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,ch ; 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

;\*\*\*\*\*\*\*\*\*\* Дескриптор, описывающий саму таблицу GDT \*\*\*\*\*\*\*\*\*\*

Gdt\_Desc EQU $-gdt ; Селектор дескриптора

Gdt1 Gdt\_Descriptor <gdt\_leng-1,,,data\_seg\_access>; !

;\*\*\*\*\*\* Дескриптор, описывающий сегмент Cseg как кодовый \*\*\*\*\*\*

Cs\_Code EQU $-gdt ; Селектор дескриптора

Gdt2 Gdt\_Descriptor<0FFFFh,,,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<0FFFFh,,,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<0FFFh,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 22 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$'

;Len dw 14d -

Gate\_Failure db "Error open A20$"

Mess\_Er db "Fatal Error$"

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

lgdt Gdt1 ; Загрузка GDTR

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

; 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

osh: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

; 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:

;Вывести сообщение об ошибке

lea bx,Mess\_Er ; Адрес сообщения

mov dx,200Ch ; Координаты вывода

int 39d ; Вывод строки на экран

;перейти к возврату в реальный режим

jmp osh

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

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;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

;ud2

dw 0B0FH

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

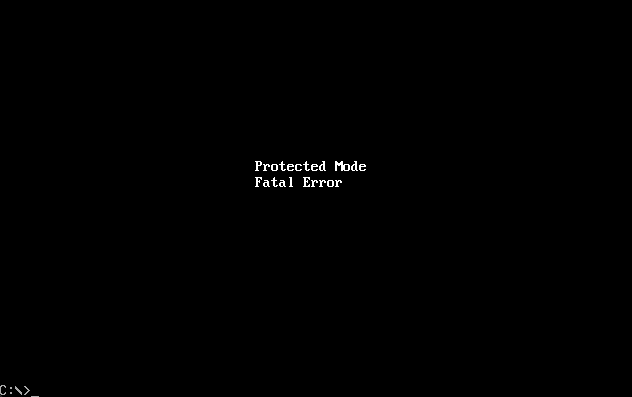
RET

PAINT\_SCREEN ENDP

;Cseg\_Leng Equ $ ; Длина сегмента Cseg -

Cseg Ends

End Start

Результат выполнения программы:  


Вывод: изучил работу в защищенном режиме и использовал полученные знания на практике.

Ответы на вопросы озвученные на паре:  
**1. Что происходит при переходе к обработчику?**

Когда происходит ошибка неверного кода операции (например, вызов ud2), процессор генерирует прерывание, которое приводит к переходу к соответствующему обработчику прерывания. В вашем коде это обработчик ex6\_proc.

**2. Какие обращения к памяти происходят?**

* **Запись в IDT**: при инициализации IDT происходит запись адреса обработчика ex6\_proc в таблицу прерываний. Это делается путем установки соответствующего дескриптора в IDT, который включает селектор сегмента и адрес обработчика.
* **Чтение сообщения об ошибке**: при выполнении обработчика ex6\_proc происходит обращение к памяти для считывания адреса строки сообщения об ошибке (Mess\_Er), которое затем передается в прерывание DOS для вывода на экран.

**3. Какие проверки?**

* **Проверка на наличие ошибки**: обработчик вызывается автоматически при возникновении исключения.

**4. Какие ошибки могут возникнуть?**

* **Ошибки доступа к памяти**: если обработчик пытается обратиться к несуществующему адресу или если IDT неправильно настроен, это может привести к ошибкам доступа.
* **Ошибки при выводе**: если прерывание для вывода сообщения (int 21h) по какой-то причине не выполнится (например, если не будет установлен правильный режим работы), это также может вызвать проблемы.

**5. Как вычисляется адрес?**

Адрес обработчика прерывания (ex6\_proc) указывается в IDT. При возникновении прерывания процессор использует этот адрес для перехода к обработчику. Адрес вычисляется на основе смещения, указанного в дескрипторе IDT.

ex6 Idt\_Descriptor<offset ex6\_proc,cs\_code,0,10000111b,0>

**6. Что считывается и куда записывается?**

**Считывание:**

* **При вызове обработчика прерывания ex6\_proc считывается адрес сообщения об ошибке, который хранится в переменной (Mess\_Er). Этот адрес загружается в регистр bx с помощью инструкции lea bx, Mess\_Er.**
* **Также считываются координаты для вывода сообщения, которые задаются в регистре dx (например, mov dx, 200Ch).**

**Запись:**

* **Адрес обработчика прерывания (функция ex6\_proc) записывается в таблицу прерываний (IDT) при инициализации системы, что позволяет процессору знать, куда переходить при возникновении ошибки.**
* **В процессе работы, когда происходит вызов прерывания для вывода сообщения (int 39d), сообщение об ошибке, адрес которого хранится в bx, выводится на экран в координатах, указанных в dx.**

**7. Обработчик прерывания написан каким шлюзом (IntN)?**

В коде обработчик прерывания для ошибки неверного кода операции написан как "шлюз" прерывания (interrupt gate). В архитектуре x86 это обозначается как IntN, где N - номер прерывания (в данном случае это 0, если считать от 0 до 255). В коде это реализовано через структуру Idt\_Descriptor, которая включает:

* Смещение обработчика (Int\_Offset)
* Селектор сегмента (Int\_Selector)
* Права доступа (Access)

Пример записи в IDT для обработчика:

ex6 Idt\_Descriptor <offset ex6\_proc, cs\_code, 0, 10000111b, 0>

Это определяет шлюз прерывания, который будет использоваться для обработки данного исключения.

Три формулы Какие обращения к памяти происходят к обработчику памяти в защищенном режиме

1 обращение к памяти: чтение шлюза прерывания

2 обращение – чтение дескриптора сегмента кода

3 обращение – сохранение в стеке регистра флагов и адреса возврата

1)Address*IDT*​=Базовый адрес таблицы прерываний+(номер прерывания\*8)

2)Address*Descriptor*​=базовый адрес таблицы дескрипторов​+(индекс дескриптора×8)

3) **Сохранение адреса возврата**:

* + При вызове функции: ESP=ESP−4(или−2 для 16-битной архитектуры)ESP=ESP−4(или−2 для 16-битной архитектуры) Stack[ESP]=Адрес возврата
  + Stack[ESP]=Адрес возврата

**Сохранение регистра флагов**:

* + Сохранение регистра флагов: ESP=ESP−2(или−4 для 32-битной архитектуры)ESP=ESP−2(или−4 для 32-битной архитектуры)
  + Stack[ESP]=FLAGSStack[ESP]=FLAGS

|  |  |  |
| --- | --- | --- |
| Биты | Номер двойного слова и биты в нём | Назначение |
| 0-15 | 1:0-15 | Зависит от типа |
| 16-31 | 1:16-31 | Селектор (зависит от типа) |
| 32-36 | 2:0-4 | Зарезервировано |
| 37-39 | 2:5-7 | Зависит от типа |
| 40-42 | 2:8-10 | Тип шлюза |
| 43 | 2:11 | Зависит от типа |
| 44 | 2:12 | Всегда 0 |
| 45,46 | 2:13,14 | Уровень привилегий дескриптора |
| 47 | 2:15 | Бит присутствия |
| 48-63 | 2:16-31 | Зависит от типа |



Сегментный регистр, такой как CS, на самом деле хранит не прямой индекс дескриптора, а значение, которое нужно интерпретировать. В 16-битной архитектуре x86, например, сегментный регистр может быть представлен следующим образом:

* **Старшие 16 бит**: это индекс дескриптора.
* **Младшие 16 бит**: это смещение (offset) внутри сегмента.

**Получение индекса дескриптора**

Чтобы получить индекс дескриптора из сегментного регистра, выполните следующие шаги:

1. **Посмотрите значение сегментного регистра**: Например, если CS = 0x10, это значение нужно будет использовать для вычисления индекса.
2. **Разделите значение на 8**: Каждый дескриптор занимает 8 байт в таблице дескрипторов. Поэтому, чтобы получить индекс дескриптора, вы должны разделить значение сегментного регистра на 8.

Индекс дескриптора=Значение сегментного регистра / 8.

**Чтение дескриптора сегмента кода**

Адрес дескриптора сегмента, содержащий обработчик прерывания, определяется с помощью селектора сегмента кода, который хранится в IDT дескрипторе.

Процессор извлекает селектор сегмента кода из дескриптора IDT и обращается к GDT (Global Descriptor Table), чтобы получить сам дескриптор сегмента кода. Адрес дескриптора вычисляется следующим образом:

*Адрес дескриптора сегмента кода=база GDT из GDTR+(селектор сегмента кода>>3)∗8*

Здесь:

* База GDT — это адрес, хранящийся в регистре GDTR.
* селектор сегмента кода >> 3 — индекс дескриптора в GDT (сдвиг на 3 бита вправо позволяет получить номер сегмента, так как младшие 3 бита селектора — это привилегии).

Умножение на 8 — поскольку размер одного дескриптора в GDT составляет 8 байт.

Процессор затем загружает 8 байт из GDT, которые содержат информацию о сегменте кода, включая базовый адрес и лимит сегмента.

