

**МИНОБРНАУКИ РОССИИ  
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ  
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)  
Кафедра МОЭВМ**

**ОТЧЕТ  
по лабораторной работе №5  
по дисциплине «Операционные системы»  
Тема: Сопряжение стандартного и пользовательского обработчиков  
прерываний**

Студент гр. 0381

Павлов Е. А.

Преподаватель

Ефремов М. А.

Санкт-Петербург  
2022

## **Цель работы.**

Исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Пользовательский обработчик прерывания получает управление по прерыванию (int 09h) при нажатии клавиши на клавиатуре. Он обрабатывает скан-код и осуществляет определенные действия, если скан-код совпадает с определенными кодами, которые он должен обрабатывать. Если скан-код не совпадает с этими кодами, то управление передаётся стандартному прерыванию.

## **Задание.**

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет такие же функции, как в программе ЛР 4, а именно:

1) Проверяет, установлено ли пользовательское прерывание с вектором 09h.

2) Если прерывание не установлено то, устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний. Адрес точки входа в стандартный обработчик прерывания находится в теле пользовательского обработчика. Осуществляется выход по функции 4Ch прерывания int 21h.

3) Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h. Выгрузка прерывания по соответствующему значению параметра в командной строке /un. Выгрузка прерывания состоит в восстановлении стандартного вектора прерываний и освобождении памяти, занимаемой резидентом. Затем осуществляется выход по функции 4Ch прерывания int 21h.

Для того чтобы проверить установку прерывания, можно поступить следующим образом. Прочитать адрес, записанный в векторе прерывания. Предположим, что этот адрес указывает на точку входа в установленный резидент. На определенном, известном смещении в теле резидента располагается

сигнатура, некоторый код, который идентифицирует резидент. Сравнив известное значение сигнатуры с реальным кодом, находящимся в резиденте, можно определить, установлен ли резидент. Если значения совпадают, то резидент установлен. Длину кода сигнатуры должна быть достаточной, чтобы сделать случайное совпадение маловероятным. Программа должна содержать код устанавливаемого прерывания в виде удаленной процедуры. Этот код будет работать после установки при возникновении прерывания. Он должен выполнять следующие функции:

- 1) Сохранить значения регистров в стеке при входе и восстановить их при выходе.
- 2) При выполнении тела процедуры анализируется скан-код.
- 3) Если этот код совпадает с одним из заданных, то требуемый код записывается в буфер клавиатуры.
- 4) Если этот код не совпадает ни с одним из заданных, то осуществляется передача управления стандартному обработчику прерывания.

Шаг 2. Запустите отлаженную программу и убедитесь, что резидентный обработчик прерывания 09h установлен. Работа прерывания проверяется введением различных символов, обрабатываемых установленным обработчиком и стандартным обработчиком.

Шаг 3. Также необходимо проверить размещение прерывания в памяти. Для этого запустите программу ЛР 3, которая отображает карту памяти в виде списка блоков МСВ. Полученные результаты поместите в отчет.

Шаг 4. Запустите отлаженную программу еще раз и убедитесь, что программа определяет установленный обработчик прерываний. Полученные результаты поместите в отчет.

Шаг 5. Запустите отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также следует запустить программу ЛР 3. Полученные результаты поместите в отчет.

Шаг 6. Ответьте на контрольные вопросы.

### Основные теоретические положения.

Клавиатура содержит микропроцессор, который воспринимает каждое нажатие на клавишу и посылает скан-код в порт микросхемы интерфейса с периферией. Когда скан-код поступает в порт, то вызывается аппаратное прерывание клавиатуры (int 09h).

Процедура обработки этого прерывания считывает номер клавиши из порта 60h, преобразует номер клавиши в соответствующий код, выполняет установку флагов в байтах состояния, загружает номер клавиши и полученный код в буфер клавиатуры.

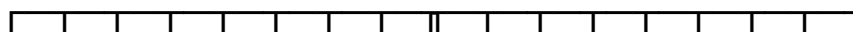
В прерывании клавиатуры можно выделить три основных шага:

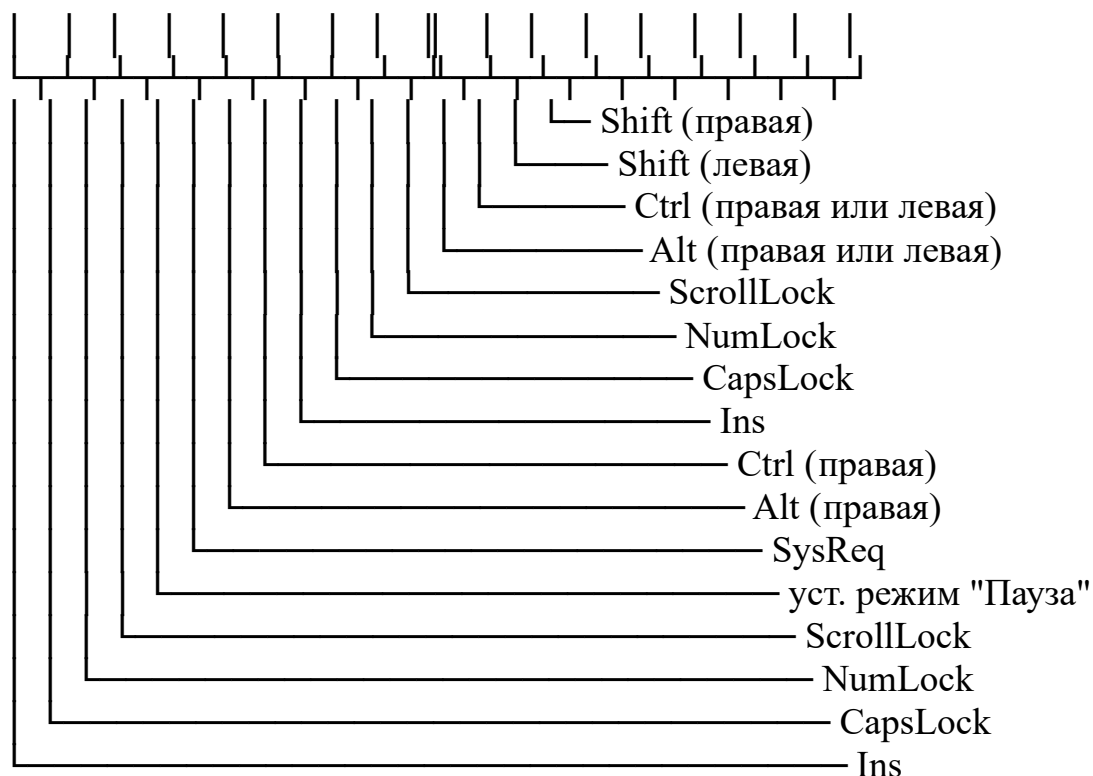
1. Прочитать скан-код и послать клавиатуре подтверждающий сигнал.
2. Преобразовать скан-код в номер кода или в установку регистра статуса клавиш-переключателей.
3. Поместить код клавиши в буфер клавиатуры.

Текущее содержимое буфера клавиатуры определяется указателями на начало и конец записи. Расположение в памяти необходимых данных представлено в таблице.

Адрес в памяти	Размер в байтах	Содержимое
0040:001A	2	Адрес начала буфера клавиатуры
0040:001C	2	Адрес конца буфера клавиатуры
0040:001E	32	Буфер клавиатуры
0040:0017	2	Байты состояния

Флаги в байтах состояния устанавливаются в 1, если нажата соответствующая клавиша или установлен режим. Соответствие флагов и клавиш показано ниже.





В момент вызова прерывания скан-код будет находиться в порте 60h. Поэтому сначала надо этот код прочитать командой IN и сохранить на стеке. Затем используется порт 61H, чтобы быстро послать сигнал подтверждения микропроцессору клавиатуры. Надо просто установить бит 7 в 1, а затем сразу изменить его назад в 0. Заметим, что бит 6 порта 61H управляет сигналом часов клавиатуры. Он всегда должен быть установлен в 1, иначе клавиатура будет выключена. Эти адреса портов применимы и к АТ, хотя он и не имеет микросхемы интерфейса с периферией 8255.

Сначала скан-код анализируется на предмет того, была ли клавиша нажата (код нажатия) или отпущена (код освобождения). Код освобождения состоит из двух байтов: сначала 0F0H, а затем скан-код. Все коды освобождения отбрасываются, кроме случая клавиш-переключателей, для которых делаются соответствующие изменения в байтах их статуса. С другой стороны, все коды нажатия обрабатываются. При этом опять могут изменяться байты статуса клавиш-переключателей. В случае же символьных кодов, надо проверять байты статуса, чтобы определить, например, что скан-код 30 соответствует нижнему или

верхнему регистру буквы А. После того как введенный символ идентифицирован, процедура ввода с клавиатуры должна найти соответствующий ему код ASCII или расширенный код. Приведенный пример слишком короток, чтобы рассмотреть все случаи. В общем случае скан-коды сопоставляются элементам таблицы данных, которая анализируется инструкцией XLAT. XLAT принимает в AL число от 0 до 255, а

возвращает в AL 1-байтное значение из 256-байтной таблицы, на которую указывает DS:BX. Таблица может находиться в сегменте данных. Если в AL находился скан-код 30, то туда будет помещен из таблицы байт номер 30 (31-й байт, так как отсчет начинается с нуля). Этот байт в таблице должен быть установлен равным 97, давая код ASCII для "a". Конечно для получения заглавной А нужна другая таблица, к которой обращение будет происходить, если статус сдвига установлен. Или заглавные буквы могут храниться в другой части той же таблицы, но в этом случае к скан-коду надо будет добавлять смещение, определяемое статусом клавиш-переключателей.

Номера кодов должны быть помещены в буфер клавиатуры. Процедура должна сначала проверить, имеется ли в буфере место для следующего символа. Буфер устроен как циклическая очередь. Ячейка памяти 0040:001A содержит указатель на голову буфера, а 0040:001C - указатель на хвост. Эти словные указатели дают смещение в области данных BIOS (которая начинается в сегменте 40H) и находятся в диапазоне от 30 до 60. Новые символы вставляются в ячейки буфера с более старшими адресами, а когда достигнута верхняя граница, то следующий символ переносится в нижний конец буфера. Когда буфер полон, то указатель хвоста на 2 меньше указателя на голову - кроме случая, когда указатель на голову равен 30 (начало области буфера), а в этом случае буфер полон, когда указатель хвоста равен 60. Для вставки символа в буфер, надо поместить

его в позицию, на которую указывает хвост буфера и затем увеличить указатель хвоста на 2; если указатель хвоста был равен 60, то надо изменить его значение на 30.

### Код для обработки прерывания 09H

```
push ax
in al,60H ;читать ключ
cmp al,REQ_KEY ;это требуемый код?
je do_req ; да, активизировать обработку REQ_KEY
; нет, уйти на исходный обработчик
pop ax
jmp cs:[int9_vect] ;переход на первоначальный обработчик
do_req:
;следующий код необходим для обработки аппаратного прерывания
in al,61H ;взять значение порта управления клавиатурой
mov ah,al ; сохранить его
or al,80h ;установить бит разрешения для клавиатуры
out 61H,al ; и вывести его в управляющий порт
xchg ah,al ;извлечь исходное значение порта
out 61H,al ;и записать его обратно
mov al,20H ;послать сигнал
"конец прерывания"
out 20H,al ; контроллеру прерываний 8259
;----- дальше - прочие проверки
```

Записать символ в буфер клавиатуры можно с помощью функции 05h

прерывания 16h:

```
mov ah,05h ; Код функции
mov cl,'D' ; Пишем символ в буфер клавиатуры
mov ch,00h ;
int 16h ;
or al,al ; проверка переполнения буфера
jnz skip ; если переполнен идем skip
; работать дальше
skip: ; очистить буфер и повторить
```

### Контрольные вопросы по лабораторной работе.

- 1) Какого типа прерывания использовались в работе?
- 2) Чем отличается скан код от кода ASCII?

### Выполнение работы.

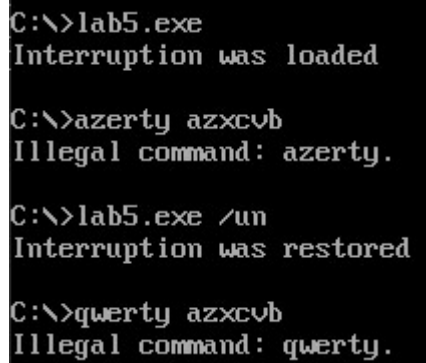
Шаг 1.

Был написан и отлажен программный модуль типа .EXE, который выполняет функции, поставленные в задании. Прерывание заменяет символы

“q”, “w” на символы “a”, “z” или на “A”, “Z”, если также была нажата клавиша “Shift”.

Шаг 2.

Программа была отлажена и запущена. Резидентный обработчик прерывания 09h установлен и размещен в памяти.



```
C:\>lab5.exe
Interruption was loaded

C:\>azerty azxcvb
Illegal command: azerty.

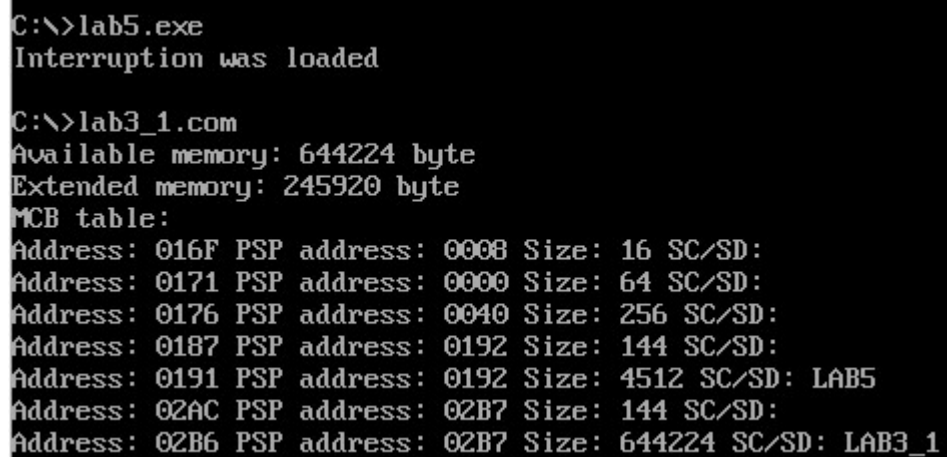
C:\>lab5.exe /un
Interruption was restored

C:\>qwerty azxcvb
Illegal command: qwerty.
```

Рисунок 1 - Работа .EXE-модуля.

Шаг 3.

Была запущена программа ЛР 3, которая отображает карту памяти в виде списка блоков МСВ. Прерывание размещено в памяти.



```
C:\>lab5.exe
Interruption was loaded

C:\>lab3_1.com
Available memory: 644224 byte
Extended memory: 245920 byte
MSB table:
Address: 016F PSP address: 0008 Size: 16 SC/SD:
Address: 0171 PSP address: 0000 Size: 64 SC/SD:
Address: 0176 PSP address: 0040 Size: 256 SC/SD:
Address: 0187 PSP address: 0192 Size: 144 SC/SD:
Address: 0191 PSP address: 0192 Size: 4512 SC/SD: LAB5
Address: 02AC PSP address: 02B7 Size: 144 SC/SD:
Address: 02B6 PSP address: 02B7 Size: 644224 SC/SD: LAB3_1
```

Рисунок 2 - Корректная установка и размещение дфй5 прерывания.

Шаг 4.

Отлаженная программа была запущена еще раз. Программа определяет установленный обработчик прерываний.



```

C:\>lab5.exe
Interruption was loaded

C:\>lab3_1.com
Available memory: 644224 byte
Extended memory: 245920 byte
MCB table:
Address: 016F PSP address: 0008 Size: 16 SC/SD:
Address: 0171 PSP address: 0000 Size: 64 SC/SD:
Address: 0176 PSP address: 0040 Size: 256 SC/SD:
Address: 0187 PSP address: 0192 Size: 144 SC/SD:
Address: 0191 PSP address: 0192 Size: 4512 SC/SD: LAB5
Address: 02AC PSP address: 02B7 Size: 144 SC/SD:
Address: 02B6 PSP address: 02B7 Size: 644224 SC/SD: LAB3_1

C:\>lab5.exe
Interruption is already loaded

```

Рисунок 3 - Корректное определение установленного обработчика прерываний.

Шаг 5.

Отлаженная программа была запущена с ключом выгрузки. Резидентный обработчик прерывания выгружен, сообщения на экран не выводятся, а память, занятая резидентом освобождена.

```

C:\>lab5.exe /un
Interruption was restored

C:\>lab3_1.com
Available memory: 648912 byte
Extended memory: 245920 byte
MCB table:
Address: 016F PSP address: 0008 Size: 16 SC/SD:
Address: 0171 PSP address: 0000 Size: 64 SC/SD:
Address: 0176 PSP address: 0040 Size: 256 SC/SD:
Address: 0187 PSP address: 0192 Size: 144 SC/SD:
Address: 0191 PSP address: 0192 Size: 648912 SC/SD: LAB3_1

```

Рисунок 4 - Корректная выгрузка обработчика прерывания.

### Ответы на контрольные вопросы.

1. Какого типа прерывания использовались в работе?

Были использованы: 09h, 16h - аппаратные прерывания, 10h, 21h - программные.

2. Чем отличается скан-код от кода ASCII?

Скан-код - код клавиши клавиатуры, который обработчик прерываний от клавиатуры преобразует в код символа, например, код символа из таблицы ASCII.

### **Выводы.**

Были исследованы возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Был написан пользовательский обработчик прерывания, который получает управление по прерыванию (int 09h) при нажатии клавиши на клавиатуре и обрабатывает скан-код, осуществляя определенные действия, если скан-код совпадает с определенными кодами, которые он должен обрабатывать. В случае, если скан-код не совпадает с этими кодами, управление передаётся стандартному прерыванию.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab5.asm

```

STACK SEGMENT STACK
    DW 200 DUP(?)
STACK ENDS

DATA SEGMENT
    interruption_already_loaded_string db 'Interruption is already
loaded', 0DH, 0AH, '$'
    interruption_loaded_string db 'Interruption was loaded', 0DH,
0AH, '$'
    interruption_not_loaded_string db 'Interruption is not loaded',
0DH, 0AH, '$'
    interruption_restored_string db 'Interruption was restored',
0DH, 0AH, '$'
    test_string db 'test', 0DH, 0AH, '$'
DATA ENDS

CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, SS:STACK

    WRITEWRD PROC NEAR
        push ax
        mov ah, 9
        int 21h
        pop ax
        ret
    WRITEWRD ENDP

    WRITEBYTE PROC NEAR
        push ax
        mov ah, 02h
        int 21h
        pop ax
        ret
    WRITEBYTE ENDP

    ENDLINE PROC NEAR
        push ax
        push dx

        mov dl, 0dh
        call WRITEBYTE

        mov dl, 0ah
        call WRITEBYTE

        pop dx
        pop ax
        ret
    ENDLINE ENDP
```

OUTPUTAL PROC NEAR

```
    push ax
    push bx
    push cx
    mov ah, 09h
    mov bh, 0
    mov cx, 1
    int 10h
    pop cx
    pop bx
    pop ax
    ret
```

OUTPUTAL ENDP

OUTPUTBP PROC NEAR

```
    push ax
    push bx
    push dx
    push CX
    mov ah,13h ; функция
    mov al, 0 ; sub function code
    ; 1 = use attribute in BL; leave cursor at end of string
    mov bh,0 ; видео страница
    mov dh,22 ; DH,DL = строка, колонка (считая от 0)
    mov dl,0
    int 10h
    pop CX
    pop dx
    pop bx
    pop ax
    ret
```

OUTPUTBP ENDP

MY\_INTERRUPT PROC FAR

```
    jmp start
```

```
    STD_KEY db 0h
    SHIFT_PRESSED db 0
    interruption_signature dw 7777h
```

```
    int_keep_ip dw 0
    int_keep_cs dw 0
    psp_address dw ?
    int_keep_ss dw 0
    int_keep_sp dw 0
    int_keep_ax dw 0
    IntStack dw 64 dup(?)
```

start:

```
    mov int_keep_sp, sp
    mov int_keep_ax, ax
    mov ax, ss
    mov int_keep_ss, ax

    mov sp, OFFSET start
```

```

        mov ax, seg IntStack
        mov ss, ax

        mov ax, int_keep_ax

        push ax
        push cx
        push dx
push es

        mov STD_KEY, 0h
        mov SHIFT_PRESSED, 0h

        mov ax, 40h
        mov es, ax
        mov ax, es:[17h]
        and ax, 11b
        cmp ax, 0h
        je read_symbol
        mov SHIFT_PRESSED, 1h

read_symbol:
        in al, 60h
        cmp al, 10h
        je key_q
        cmp al, 11h
        je key_w
        mov STD_KEY, 1h
        jmp interruption_end

key_q:
        mov al, 'a'
        jmp do_req

key_w:
        mov al, 'z'
        jmp do_req

do_req:
        push ax
        ;отработка аппаратного прерывания
        in al, 61H
        mov ah, al
        or al, 80h
        out 61H, al
        xchg ah, al
        out 61H, al
        mov al, 20H
        out 20H, al
        pop ax

        cmp SHIFT_PRESSED, 0h
        je print_key
        sub al, 20h

```

```

print_key:
    mov ah, 05h
    mov cl, al
    mov ch, 00h
    int 16h
    or al, al
    jz interruption_end
    mov ax, 0040h
    mov es, ax
    mov ax, es:[1ah]
    mov es:[1ch], ax
    jmp print_key

```

```

interruption_end:

```

```

pop es

```

```

    pop dx
    pop cx
    pop ax

```

```

    mov sp, int_keep_sp
    mov ax, int_keep_ss
    mov ss, ax
    mov ax, int_keep_ax

```

```

    mov al, 20h
    out 20h, al

```

```

    cmp STD_KEY, 1h
    jne interruption_iret

```

```

    jmp dword ptr cs:[int_keep_ip]

```

```

interruption_iret:
    iret

```

```

MY_INTERRUPTION ENDP
interruption_last_byte:

```

```

CHECK_CLI_OPT PROC near
    push ax
    push bp

```

```

    mov cl, 0h

```

```

    mov bp, 81h

```

```

    mov al, es:[bp + 1]
    cmp al, '/'
    jne lafin

```

```

    mov al, es:[bp + 2]
    cmp al, 'u'
    jne lafin

```

```

    mov al, es:[bp + 3]

```

```

        cmp al, 'n'
        jne lafin

        mov cl, 1h

lafin:
        pop bp
        pop ax
        ret
CHECK_CLI_OPT ENDP

CHECK_LOADED PROC NEAR
        push ax
        push dx
        push es
        push si

        mov cl, 0h

        mov ah, 35h
        mov al, 09h
        int 21h

        mov si, offset interruption_signature
        sub si, offset MY_INTERRUPTION
        mov dx, es:[bx + si]
        cmp dx, interruption_signature
        jne checked

        mov cl, 1h ;already loaded

checked:
        pop si
        pop es
        pop dx
        pop ax
        ret
CHECK_LOADED ENDP

LOAD_INTERRUPTION PROC near
        push ax
        push cx
        push dx

        call CHECK_LOADED
        cmp cl, 1h
        je int_already_loaded

        mov psp_address, es

        mov ah, 35h
        mov al, 09h
        int 21h

```

```

        mov int_keep_cs, es
mov int_keep_ip, bx

push es
push bx
push ds

        lea dx, MY_INTERRUPTION
        mov ax, SEG MY_INTERRUPTION
        mov ds, ax

        mov ah, 25h
        mov al, 09h
        int 21h

        pop ds
        pop bx
        pop es

        mov dx, offset interruption_loaded_string
        call WRITEWRD

        lea dx, interruption_last_byte
        mov cl, 4h
        shr dx, cl
        inc dx ;dx - size in paragraphs

        add dx, 100h

        xor ax,ax

        mov ah, 31h
        int 21h

        jmp fin_load_interruption

int_already_loaded:
        mov dx, offset interruption_already_loaded_string
        call WRITEWRD

fin_load_interruption:
        pop dx
        pop cx
        pop ax
        ret
LOAD_INTERRUPTION ENDP

UNLOAD_INTERRUPTION PROC near
        push ax
        push si

        call CHECK_LOADED
        cmp cl, 1h
        jne interruption_is_not_loaded

```



```

cli

push ds
push es

mov ah, 35h
mov al, 09h
int 21h

mov si, offset int_keep_ip
sub si, offset MY_INTERRUPTION
mov dx, es:[bx + si]
mov ax, es:[bx + si + 2]
    mov ds, ax

mov ah, 25h
mov al, 09h
int 21h

mov ax, es:[bx + si + 4]
    mov es, ax
    push es

        mov ax, es:[2ch]
        mov es, ax
        mov ah, 49h
        int 21h

    pop es
    mov ah, 49h
    int 21h

pop es
pop ds

sti

mov dx, offset interruption_restored_string
call WRITEWRD

jmp int_unloaded

interruption_is_not_loaded:
    mov dx, offset interruption_not_loaded_string
    call WRITEWRD

int_unloaded:
    pop si
    pop ax
    ret
UNLOAD_INTERRUPTION ENDP

MAIN PROC FAR

```

```
        mov     ax, DATA
        mov     ds, ax

        call    CHECK_CLI_OPT
        cmp     cl, 0h
        jne     opt_unload

        call    LOAD_INTERRUPTION
        jmp     main_end

opt_unload:

        call    UNLOAD_INTERRUPTION

main_end:
        xor     al, al
        mov     ah, 4ch
        int     21h

MAIN     ENDP
```

CODE ENDS

END MAIN