

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

**ОТЧЕТ**  
**по практической работе № 4**  
**по дисциплине «Операционные системы»**  
**Тема: Обработка стандартных прерываний**

Студент гр. 8383

Степанов В.Д.

Преподаватель

Губкин А.Ф.

Санкт-Петербург

2020

### **Цель работы.**

Построить обработчик прерываний сигналов таймера на языке Ассемблера.

### **Выполнение работы.**

1. Был написан программный модуль типа .EXE, который выполняет следующий функции:

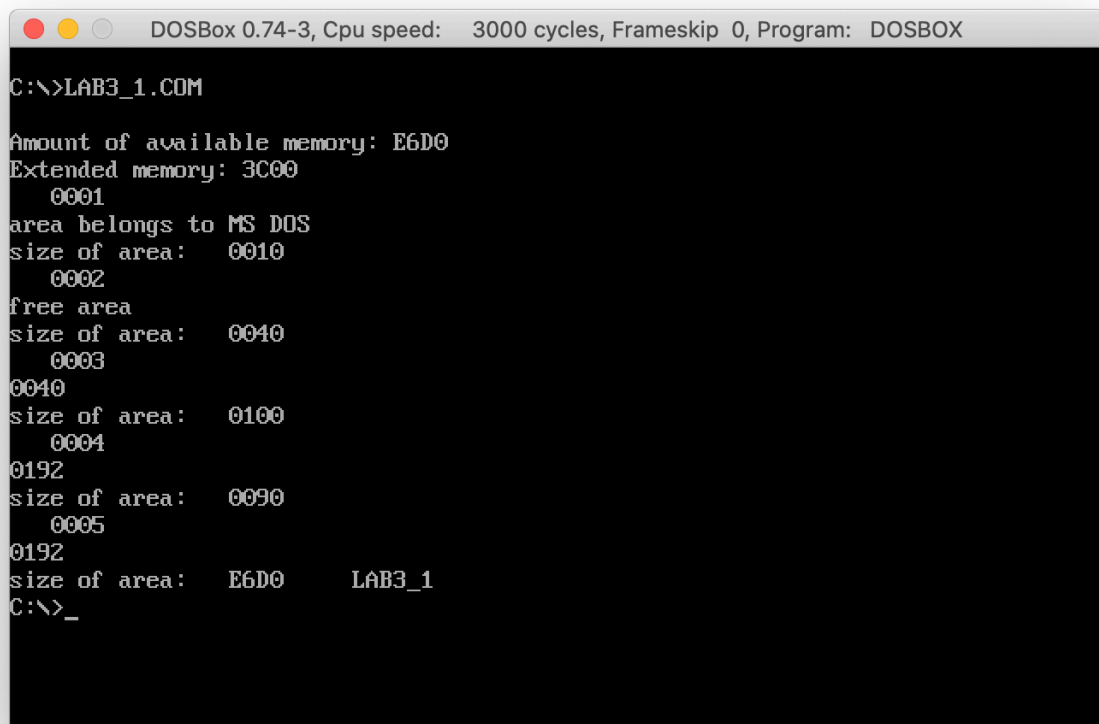
- 1) Проверяет, установлено ли пользовательское прерывание с вектором 1Ch.
- 2) Устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний, если прерывание не установлено, и осуществляет выход по функции 4Ch прерывания int 21h.
- 3) Если прерывание установлено, то выводится соответствующие сообщение и осуществляется выход по функции 4Ch прерывания int 21h.
- 4) Выгрузка прерывания по соответствующему значению параметра командной строке /un.

Программа содержит код устанавливаемого прерывания в виде удаленной процедуры INTER. Он выполняет функции:

- 1) Сохраняет значения регистров в стек при входе и восстанавливает при выходе.
  - 2) При выполнении тела процедуры накапливает общее суммарное число пребывания и выводит на экран.
2. С помощью лабораторной работы №3 посмотрим карту памяти до загрузки прерывания. Результат представлен на рисунке 1. Теперь загрузим пребывания и опять выведем карту памяти. Результат представлен на рисунке 2. В конце выгрузим прерывание и выведем карту памяти. Результат представлен на рисунке 3.

### **Контрольные вопросы.**

- 1) Как реализован механизм прерывания от часов?
- Системный таймер вырабатывает прерывание INT 8h приблизительно 18,2

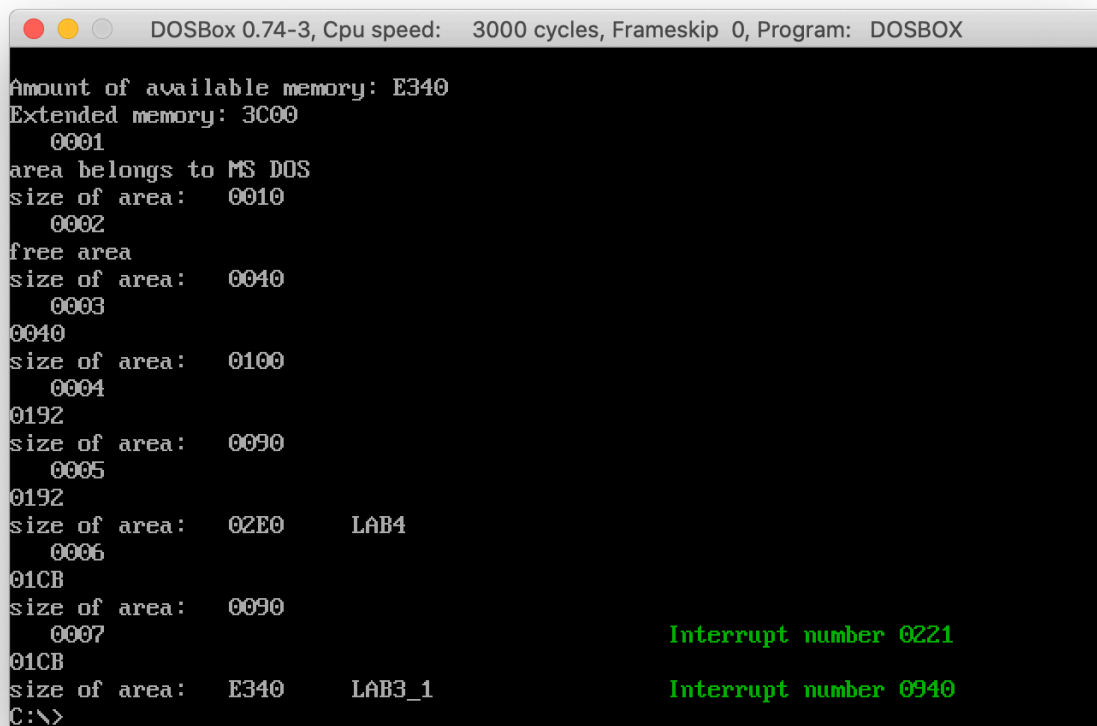
A screenshot of a DOSBox window. The title bar reads "DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX". The command prompt shows "C:\>LAB3\_1.COM". The output displays memory information: "Amount of available memory: E6D0", "Extended memory: 3C00", and a series of memory areas with their sizes. The last line of output is "C:\>\_".

```
C:\>LAB3_1.COM

Amount of available memory: E6D0
Extended memory: 3C00
0001
area belongs to MS DOS
size of area: 0010
0002
free area
size of area: 0040
0003
0040
size of area: 0100
0004
0192
size of area: 0090
0005
0192
size of area: E6D0    LAB3_1
C:\>_
```

Рисунок 1 – Вывод карты памяти до загрузки прерывания

раза в секунду. При инициализации BIOS устанавливает свой обработчик для прерывания таймера. Этот обработчик каждый раз увеличивает на 1 текущее значение четырехбайтовой переменной, располагающейся в области данных BIOS по адресу 0000:046Ch - счетчик тиков таймера. Если этот счетчик переполняется (прошло более 24 часов с момента запуска таймера), в ячейку 0000:0470h заносится 1. Следующие действие, которое выполняет обработчик прерывания таймера - вызов прерывания INT 1Ch. После инициализации системы вектор INT 1Ch указывает на команду IRET, т.е. ничего не выполняется. Программа может установить собственный обработчик этого прерывания для того чтобы выполнять какие-либо периодические действия. Необходимо отметить, что прерывание INT 1Ch вызывается обработчиком прерывания INT 8h до сброса контроллера прерывания, поэтому во время выполнения прерывания INT 1Ch все аппаратные прерывания запрещены.

The image shows a DOSBox window with a black background and white text. The title bar reads "DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX". The text inside the window is a memory dump. It starts with "Amount of available memory: E340" and "Extended memory: 3C00". Then it lists several memory areas with their addresses and sizes. For example, "0001 area belongs to MS DOS size of area: 0010". There are also labels like "LAB4" and "LAB3\_1" next to some entries. At the bottom right, there are two green lines of text: "Interrupt number 0221" and "Interrupt number 0940". The prompt "C:\>\_" is visible at the bottom left.

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
Amount of available memory: E340
Extended memory: 3C00
0001
area belongs to MS DOS
size of area: 0010
0002
free area
size of area: 0040
0003
0040
size of area: 0100
0004
0192
size of area: 0090
0005
0192
size of area: 02E0 LAB4
0006
01CB
size of area: 0090
0007
01CB
size of area: E340 LAB3_1
C:\>_
Interrupt number 0221
Interrupt number 0940
```

Рисунок 2 – Вывод карты памяти после загрузки прерывания

2) Какого типа прерывания использовались в работе?

int 1Ch – пользовательское прерывание по таймеру (аппаратное)

int 10h – видео сервис (программное)

int 21h – сервис DOS (программное)

### Выводы.

В ходе лабораторной работы был построен обработчик прерываний сигналов таймера на языке Ассемблер

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

C:\>lab4
Interrupt does not installed
C:\>lab4/un
Interrupt number 0091

C:\>LAB3_1.COM

Amount of available memory: E6D0
Extended memory: 3C00
0001
area belongs to MS DOS
size of area: 0010
0002
free area
size of area: 0040
0003
0040
size of area: 0100
0004
0192
size of area: 0090
0005
0192
size of area: E6D0 LAB3_1
C:\>_
```

Рисунок 3 – Вывод карты памяти после выгрузки прерывания

## ПРИЛОЖЕНИЕ А

### КОД ПРОГРАММЫ

```
LAB3CODE SEGMENT
    ASSUME CS: LAB3CODE, ds:DATA, es:NOTHING, SS:LAB3STACK

INTER PROC FAR

    jmp START
    STR_OUT db 'Interrupt number 0000$'
    INTER_ID dw 0804h
    KEEP_AX dw 0
    KEEP_SS dw 0
    KEEP_SP dw 0
    KEEP_IP dw 0
    KEEP_CS dw 0
    PSP_SEG dw 0
    INTER_STACK dw 128 dup(0)

START:
    mov KEEP_SS, ss
    mov KEEP_SP, sp
    mov KEEP_AX, ax
    mov ax, seg INTER_STACK
    mov ss, ax
    mov ax, offset INTER_STACK
    add ax, 256
    mov sp, ax
    push bx
    push cx
    push dx
    push si
    push ds
    push bp
    push es

    mov ah, 03h
    mov bh, 00h
    int 10h

    push dx
```

```

mov ah,09h
mov bh,0
mov cx,0
int 10h

mov ah,02h
mov bh,0
mov dh,17h
mov dl,30h
int 10h

mov ax, seg STR_OUT
push ds
mov ds, ax
mov si, offset STR_OUT
add si, 20
mov cx, 4
CYCLE:
mov ah, [si]
inc ah
mov [si], ah
cmp ah, ':'
jne END_CYCLE
mov ah, '0'
mov [si], ah
dec si
loop CYCLE
END_CYCLE:
pop ds

push es
push bp
mov ax, SEG STR_OUT
mov es, ax
mov bp, offset STR_OUT

mov ah,13h
mov al,1
mov bl, 2h
mov cx, 21
mov bh, 0
int 10h

```

```

    pop bp
    pop es

    pop dx
    mov ah, 02h
    mov bh, 0
    int 10h

    pop es
    pop bp
    pop ds
    pop si
    pop DX
    pop cx
    pop bx
    mov sp, KEEP_SP
    mov ax, KEEP_SS
    mov ss, ax
    mov ax, KEEP_AX
    mov al, 20h
    OUT 20h, al
    IRET
    ret
INTER ENDP
INTER_END:

CHECK_INTER PROC NEAR

    push ax
    push bx
    push si

    MOV ah, 35H
    MOV al, 1CH
    int 21h
    mov si, offset INTER_ID
    sub si, offset INTER
    mov ax, es:[bx + si]
    cmp ax, 0804h
    jne CHECK_END
    mov DOES_INTER_INSTAL, 1

```



```

CHECK_END:

    pop si
    pop bx
    pop ax

    ret
CHECK_INTER ENDP
;-----
PRINT PROC near

    push ax
    sub ax, ax
    mov ah, 9h
    int 21h
    pop ax

    ret
PRINT ENDP
;-----
CHECK_UN PROC NEAR

    push ax
    push es

    mov ax, PSP_SEG
    mov es, ax
    cmp byte ptr es:[82h], '/'
    jne CHECK_UN_END
    cmp byte ptr es:[83h], 'u'
    jne CHECK_UN_END
    cmp byte ptr es:[84h], 'n'
    jne CHECK_UN_END
    mov IS_UN, 1

CHECK_UN_END:
    pop es
    pop ax
    ret
CHECK_UN ENDP
;-----
UNSTE_INTER PROC NEAR

```

```

        CLI
        push ax
        push bx
        push dx
        push ds
        push es
        push si

        mov ah, 35h
        mov al, 1Ch
        int 21h

        mov si, offset KEEP_IP
        sub si, offset INTER

        mov DX, es:[bx + si]
        mov ax, es:[bx + si + 2]
        push ds
        mov ds, ax
        mov ah, 25h
        mov al, 1Ch
        int 21h
        pop ds
        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 si
        pop es
        pop ds
        pop dx
        pop bx
        pop ax
        STI
        ret
UNSTE_INTER ENDP

```

```

;-----
SET_INTER PROC NEAR

    push ax
    push bx
    push cx
    push dx
    push ds
    push es

    mov ah, 35h
    mov al, 1Ch
    int 21H
    mov KEEP_IP, bx
    mov KEEP_CS, es

    push ds
    mov DX, offset INTER
    mov ax, seg INTER
    mov ds, ax
    mov ah, 25H
    mov al, 1CH
    int 21H
    pop ds

    mov dx, offset INTER_END
    add dx, 10Fh
    mov cl, 4h
    shr dx, cl
    inc dx
    xor ax, ax
    mov ah, 31h
    int 21h

    pop es
    pop ds
    pop dx
    pop cx
    pop bx
    pop ax
    ret

```

```

SET_INTER ENDP
;-----
MAIN PROC

    push ds
    xor ax, ax
    push ax
    mov ax, DATA
    mov ds, ax
    mov PSP_SEG, es

    call CHECK_INTER
    call CHECK_UN

    mov al, IS_UN
    cmp al, 1
    je UNLOAD_INTER

    mov al, DOES_INTER_INSTAL
    cmp al, 1
    jne CHECK_NOT_OK
    mov dx, offset INTER_INSTAL
    call PRINT
    jmp MAIN_END

CHECK_NOT_OK:
    mov dx, offset INTER_DOES_NOT_INSTAL
    call PRINT
    call SET_INTER
    jmp MAIN_END

UNLOAD_INTER:
    mov al, DOES_INTER_INSTAL
    cmp al, 1
    jne FAIL_UNLOAD

    call UNSTE_INTER

    jmp MAIN_END

FAIL_UNLOAD:
    mov dx, offset INTER_DOES_NOT_INSTAL
    call PRINT

```

```

MAIN_END:
    mov ah, 4Ch
    int 21h

MAIN ENDP
LAB3CODE ENds
;-----
LAB3STACK SEGMENT STACK
    dw 128 dup(0)
LAB3STACK ENds
;-----
DATA SEGMENT
    IS_UN db 0
    DOES_INTER_INSTAL db 0
    INTER_INSTAL db 'Interrupt instaled$'
    INTER_DOES_NOT_INSTAL db 'Interrupt does not instaled$'
DATA ENds
;-----

END MAIN

```