

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

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

Студент гр.8382

\_\_\_\_\_

Синельников М.Р

Преподаватель

\_\_\_\_\_

Ефремов М.А

Санкт-Петербург

2020

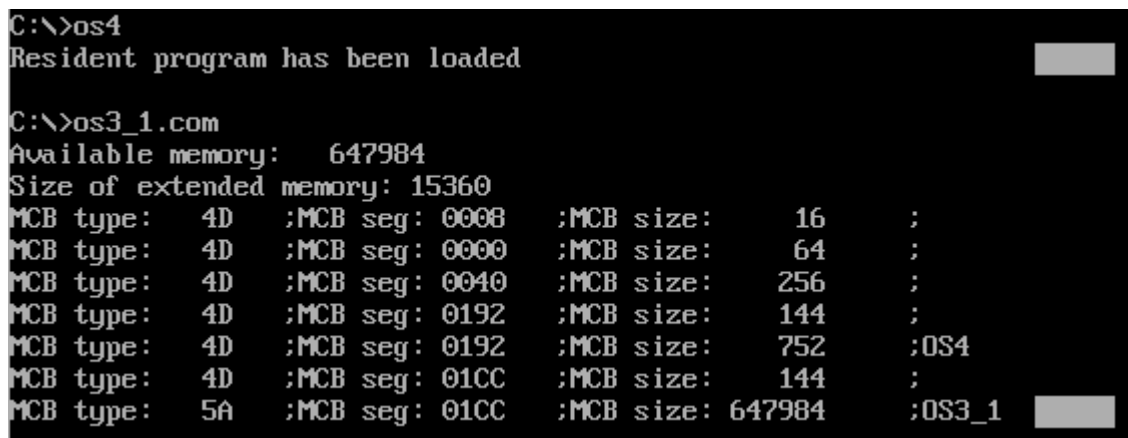
### Цель работы.

В архитектуре компьютера существуют стандартные прерывания, за которыми закреплены определённые вектора прерываний. Вектор прерываний хранит адрес подпрограммы обработчика прерываний. При возникновении прерывания, аппаратура компьютера передаёт управление соответствующему адресу вектора прерывания. Обработчик прерываний получает управление и выполняет соответствующие действия.

В лабораторной работе № 4 предлагается построить обработчик прерываний сигналов таймера. Эти сигналы генерируются аппаратурой через определённые интервалы времени и, при возникновении такого сигнала, возникает прерывание с определённым значением вектора. Таким образом, управление будет передано функции, чья точка входа записана в соответствующий вектор прерывания.

### Ход работы.

#### 1) Шаг 1



```
C:\>os4
Resident program has been loaded

C:\>os3_1.com
Available memory: 647984
Size of extended memory: 15360
MCB type: 4D ;MCB seg: 0008 ;MCB size: 16 ;
MCB type: 4D ;MCB seg: 0000 ;MCB size: 64 ;
MCB type: 4D ;MCB seg: 0040 ;MCB size: 256 ;
MCB type: 4D ;MCB seg: 0192 ;MCB size: 144 ;
MCB type: 4D ;MCB seg: 0192 ;MCB size: 752 ;OS4
MCB type: 4D ;MCB seg: 01CC ;MCB size: 144 ;
MCB type: 5A ;MCB seg: 01CC ;MCB size: 647984 ;OS3_1
```

рисунок 1 — счётчик и состояние памяти после загрузки обработчика прерывания

## 2) Шаг 2

```
MCB type: 4D ;MCB seg: 0000 ;MCB size: 64 ;
MCB type: 4D ;MCB seg: 0040 ;MCB size: 256 ;
MCB type: 4D ;MCB seg: 0192 ;MCB size: 144 ;
MCB type: 4D ;MCB seg: 0192 ;MCB size: 752 ;OS4
MCB type: 4D ;MCB seg: 01CC ;MCB size: 144 ;
MCB type: 5A ;MCB seg: 01CC ;MCB size: 647984 ;OS3_1
C:\>os4
Resident program is already loaded
```

рисунк 2 - повторная загрузка обработчика  
прерывания

## 3) Шаг 3

```
C:\>os4/un
Resident program unloaded

C:\>os3_2.com
Available memory: 648912
Size of extended memory: 15360
MCB type: 4D ;MCB seg: 0008 ;MCB size: 16 ;
MCB type: 4D ;MCB seg: 0000 ;MCB size: 64 ;
MCB type: 4D ;MCB seg: 0040 ;MCB size: 256 ;
MCB type: 4D ;MCB seg: 0192 ;MCB size: 144 ;
MCB type: 4D ;MCB seg: 0192 ;MCB size: 12576 ;OS3_2
MCB type: 5A ;MCB seg: 0000 ;MCB size: 636320 ;7h@P7.4P
```

рисунк 3 — выгрузка обработчика и состояние памяти

### Контрольные вопросы.

#### 1) Как реализован механизм прерывания от часов?

Каждые 55 мс сначала сохраняется состояние регистров, затем определяется источник прерывания, который определяет в свою очередь адрес вектора прерывания в таблице векторов прерываний. Первые два байта помещаются в регистр IP, а вторые два байта – в CS. Затем управление передаётся по адресу CS:IP и происходит обработка соответствующего

прерывания. После завершения обработки управление возвращается прерванной программе.

2) Какого типа прерывания использовались в программе?

При выполнении работы использовались программные прерывания `int 21h`, `int 10h`, а также аппаратное прерывание `int 1Ch`, возникающее каждые 55 мс по системному таймеру

### **Вывод.**

В ходе выполнения работы был реализован обработчик прерываний сигналов системного таймера.

## Приложение А

### Исходный код файла OS\_4.asm

```
CODE SEGMENT

ASSUME CS:CODE, DS:DATA, SS:AStack

getCurs PROC
    push ax
    push bx
    mov ah, 03h
    mov bh, 0
    int 10h
    pop bx
    pop ax
    ret
getCurs ENDP

SetCurs PROC
    push ax
    push bx
    mov ah, 02h
    mov bh, 0
    int 10h
    pop bx
    pop ax
    ret
SetCurs ENDP

HANDLER PROC FAR
    jmp HANDLER_CODE

HANDLER_DATA:
    HANDLER_SIGNATURE    DW    6000h
    KEEP_CS              DW    0
    keep_ip dw 0
    keep_psp dw 0
    KEEP_SS              DW    0
    KEEP_SP              DW    0
```

KEEP_AX	DW	0	
COUNTER	DW	0	
COUNTER_STR	DB	'00000\$'	
HANDLER_STACK	DW	100	DUP(0)

```

HANDLER_CODE:
mov KEEP_SS, ss
mov KEEP_SP, sp
mov KEEP_AX, ax
mov ax, seg HANDLER_STACK
mov ss, ax
mov sp, offset HANDLER_CODE
push bx
push cx
push dx
push si
push ds

mov ax, seg HANDLER_DATA
mov ds, ax

inc COUNTER
mov ax, COUNTER
mov dx, 0
mov si, offset COUNTER_STR
add si, 4
call WRD_TO_DEC
call getCurs
push dx
mov bh, 0
mov dx, 1640h
mov ah, 02h
int 10h
push es
push bp
mov ax, seg COUNTER_STR
mov es, ax
mov bp, offset COUNTER_STR
mov al, 1
mov bh, 0
mov cx, 5

```

```

mov ah, 13h
int 10h
pop bp
pop es
pop dx
call setCurs
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

```

HANDLER ENDP

WRD\_TO\_DEC PROC near

```

push ax
push bx
mov bx, 10
div_loop:
div bx
add dl, 30h
mov [si], dl
dec si
mov dx, 0
cmp ax, 0
jne div_loop

pop bx
pop ax
ret

```

WRD\_TO\_DEC ENDP

HANDLER\_END:

```

Un_check PROC      FAR
    cmp byte ptr es:[82h], '/'
    jne FALSE
    cmp byte ptr es:[83h], 'u'
    jne FALSE
    cmp byte ptr es:[84h], 'n'
    jne FALSE

    jmp TRUE

FALSE:
    mov ax, 0
    ret
TRUE:
    mov ax, 1
    ret
Un_check ENDP

check_on_1ch PROC FAR
    push bx
    push si
    push es
    mov si, offset HANDLER_SIGNATURE
    sub si, offset HANDLER
    mov ah, 35h
    mov al, 1ch
    int 21h
    mov ax, es:[bx+si]
    mov bx, HANDLER_SIGNATURE
    cmp ax, bx
    je CHECK_TRUE
    mov ax, 0
    jmp finish_1ch

CHECK_TRUE:
    mov ax, 1
finish_1ch:
    pop es
    pop si
    pop bx
    ret

```



check\_on\_1ch endp

```
Keep_interr    PROC
                push ax
                push bx
                push es
                mov ah, 35h
                mov al, 1Ch
                int 21h
                mov keep_ip, bx
                mov keep_cs, es
                pop es
                pop bx
                pop ax
                ret
```

Keep\_interr ENDP

```
Load_handler    PROC

                push ax
                push bx
                push dx
                push es
                call keep_interr
                push ds
                mov dx, offset Handler
                mov ax, seg Handler
                mov ds, ax
                mov ah, 25h
                mov al, 1ch
                int 21h
                pop ds
                pop es
                pop dx
                pop bx
                pop ax
                ret
```

Load\_handler ENDP

Unload\_handler PROC

```

push ax
push bx
push dx
push es
push si
mov si,offset keep_cs
sub si,offset Handler
mov ah, 35h
mov al,1ch
int 21h
cli
push ds
mov dx,es:[bx + si + 2]
mov ax,es:[bx + si]
mov ds,ax
mov ah,25h
mov al,1ch
int 21h
pop ds
sti

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 dx
pop bx
pop ax
ret

```

Unload\_handler ENDP

```

Make_resident PROC
    mov dx, offset HANDLER_END
    mov cl, 4
    shr dx, cl

    add dx, 16h
    inc dx

    mov ax, 3100h
    int 21h
Make_resident ENDP

```

```

print_message PROC
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
print_message ENDP

```

```

Main PROC
    push ds
    xor ax, ax
    push ax
    mov ax, data
    mov ds, ax
    mov Keep_PSP, es
    call check_on_1ch
    cmp ax, 1
    jne loading
    call Un_check
    cmp ax, 1
    jne alr_loaded
    call Unload_handler
    lea dx, Message2
    call print_message
    mov ax, 4c00h
    int 21h
    jmp finish
loading:
    call Load_Handler
    lea DX, Message1

```

```

        call print_message
        call Make_resident
alr_loaded:
        lea dx, Message3
        call print_message
        mov ax, 4C00h
        int 21h

        finish:

Main    ENDP
CODE                      ENDS

AStack      SEGMENT STACK
            DW 64 DUP(0)
AStack      ENDS

DATA        SEGMENT
    Message1 db 'Resident program has been loaded', 0dh, 0ah, '$'
    Message2 db 'Resident program unloaded', 0dh, 0ah, '$'
    Message3 db 'Resident program is already loaded', 0dh, 0ah, '$'
DATA        ENDS
            END Main

```