**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. 7381 _____ Адамов Я.В.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2019

## 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 по 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 определенные интервалы времени и, при возникновении такого сигнала, возникает прерывание с определенным значением вектора. Таким образом, управление будет передано функции, чья точка входа записана в соответствующий вектор прерывания.

**Описание функций.**

| Название функции | Описание |
|---|---|
| PrintMsg | Печать строки, адрес которой помещен в DX. |
| setCurs | Установка курсора в позицию, указанную в DX (строка в DH, столбец в DL). |
| getCurs | Возврат положения курсора в DX. |
| ROUT | Обработчик прерываний, считающий и выводящий на экран количество его вызовов. |
| CHECKING | Проверка, загружен ли обработчик прерываний. |
| SET_INTERRUPT | Установка нового обработчик прерывания с запоминанием данных для восстановления предыдущего обработчика прерываний. |

**Описание структур данных.**

| Название | Тип | Описание |
|---|---|---|
| wasloaded | db | Строка: «Interruption had already been loaded.». |
| unloaded | db | Строка: «Interruption is restored.». |
| loading | db | Строка: «Interruption is loaded.». |
| COUNTER | db | Строка для вывода количества вызовов обработчика прерываний. |

**Описание работы утилиты.**

Программа проверяет, установлено ли пользовательское прерывание с вектором 1Ch. Устанавливает обработчик прерываний, если он не установлен, в ином случае выводится соответствующее сообщение. Программа выгружает прерывания по соответствующему значению параметра в командной строке /un, восстановления стандартного вектора прерывания.

Состояние памяти до запуска программы получено с помощью утилиты, написанной в третье лабораторной работе, и представлено на рис. 1. Демонстрация работы программы представлена на рис. 2. Состояние памяти после запуска программы представлено на рис. 3. Результат повторного запуска программы представлен на рис. 4. Запуск программы с ключом выгрузки /un представлен на рис. 5. Состояние памяти после выгрузки представлено на рис. 6.

```
Amount of available memory: 648912 B
Extended memory size: 15360 KB

 MCB Adress   MCB Type   Owner          Size        Name
 016F         4D         0008             16
 0171         4D         0000             64
 0176         4D         0040            256
 0187         4D         0192            144
 0191         5A         0192         648912        LAB3
```

Рисунок 1 – состояние памяти до запуска программы.


Рисунок 2 – демонстрация работы программы.

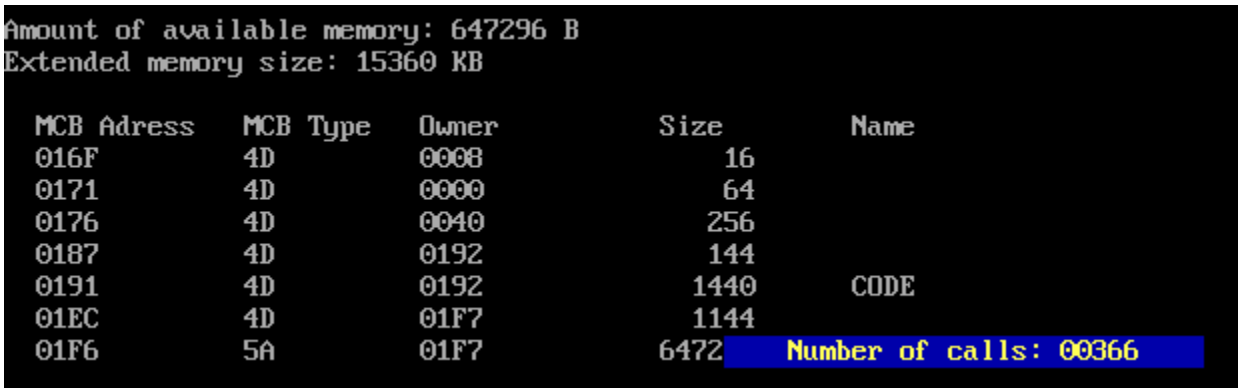


Рисунок 3 – состояние памяти после запуска программы.


Рисунок 4 – повторный запуск программы.

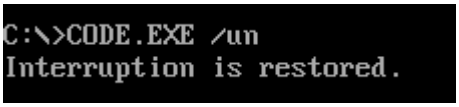


Рисунок 5 – запуск программы с ключом выгрузки /un.

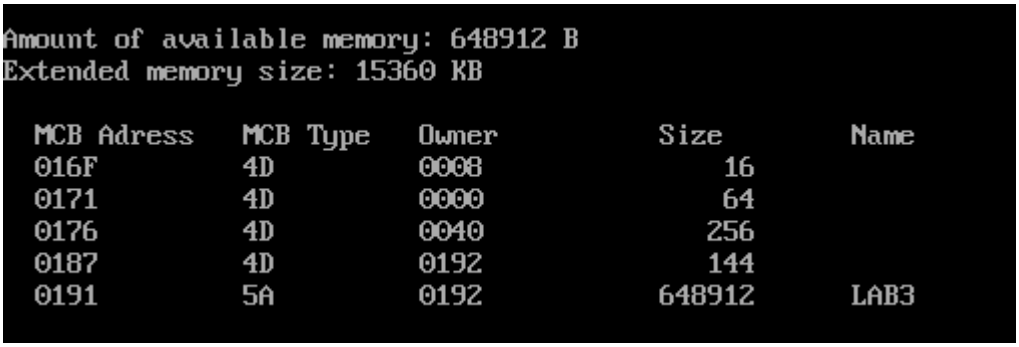


Рисунок 6 – состояние памяти после выгрузки.

**Вывод.**

В ходе выполнения лабораторной работы был построен обработчик прерываний сигналов таймера.

**Ответы на контрольные вопросы.**

*Как реализован механизм прерывания от часов?*

Прерывание по таймеру вызывается каждые 55 мс – 18 раз в секунду (по каждому тику аппаратных часов). При каждом вызове содержимое регистров сохраняется, затем определяется источник прерывания, по его номеру определяется смещение в таблице векторов прерывания. Значения из таблицы копируются в CS и IP (управление переходит обработчику прерывания). После обработки прерывания происходит возврат значений CS и IP (управление возвращается программе).

*Какого типа прерывания использовались в работе?*

В работе использовались программные (int 10h, int 21h) и аппаратные (int 1Ch) прерывания.

## Приложение A. lab3.asm.

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H

START: JMP BEGIN

; ___________________________________________
; Данные


AvailableMemory   db   0dh,0ah,'Amount   of   available   memory:
B',0dh,0ah,'$'
ExtendedMemorySize db 'Extended memory size:       KB',0dh,0ah,'$'
TableHead db 0dh,0ah,'  MCB Adress   MCB Type   Owner        Size
Name    ',0dh,0ah,'$'
MCB                                db                                '
',0dh,0ah,'$'


; ___________________________________________
; Процедуры

TETR_TO_HEX PROC near
    and al,0fh
    cmp al,09
    jbe NEXT
```

```
    add al,07
NEXT: add al,30h
    ret
TETR_TO_HEX ENDP

BYTE_TO_HEX PROC near
    push cx
    mov ah,al
    call TETR_TO_HEX
    xchg al,ah
    mov cl,4
    shr al,cl
    call TETR_TO_HEX
    pop cx
    ret
BYTE_TO_HEX       ENDP

WRD_TO_HEX PROC near
    push bx
    mov bh,ah
    call BYTE_TO_HEX
    mov [di],ah
    dec di
    mov [di],al
    dec di
    mov al,bh
    call BYTE_TO_HEX
    mov [di],ah
```

```
    dec di
    mov [di],al
    pop bx
    ret
WRD_TO_HEX ENDP

BYTE_TO_DEC PROC near
    push cx
    push dx
    xor ah,ah
    xor dx,dx
    mov cx,10
loop_bd: div cx
    or dl,30h
    mov [si],dl
    dec si
    xor dx,dx
    cmp ax,10
    jae loop_bd
    cmp al,00h
    je end_l
    or al,30h
    mov [si],al
end_l: pop dx
    pop cx
    ret
BYTE_TO_DEC ENDP
```

```
WRD_TO_DEC PROC near
    push cx
    push dx
    push ax
    mov cx,10
loop_wd:
    div cx
    or dl,30h
    mov [si],dl
    dec si
    xor dx,dx
    cmp ax,10
    jae loop_wd
    cmp ax,00h
    jbe end_l_2
    or al,30h
    mov [si],al
end_l_2:
    pop ax
    pop dx
    pop cx
    ret
WRD_TO_DEC ENDP

PrintMsg PROC near
    push ax
    mov ah,09h
    int 21h
```

```
    pop ax
    ret
PrintMsg ENDP


PrintAvailableMemory PROC near
    push ax
    push bx
    push dx
    push si

    mov ah,04Ah
    mov bx,0FFFFh
    int 21h
    mov ax,10h
    mul bx
    lea si,AvailableMemory
    add si,35
    call WRD_TO_DEC
     lea dx,AvailableMemory
     call PrintMsg

    pop si
    pop dx
    pop bx
    pop ax
    ret
PrintAvailableMemory ENDP
```

```
PrintExtendedMemorySize PROC near
    push ax
    push bx
     push dx
    push si

    mov al,30h
    out 70h,al
    in al,71h
    mov bl,al
    mov al,31h
    out 70h,al
    in al,71h
    mov ah,al
    mov al,bl
    sub dx,dx
    lea si,ExtendedMemorySize
    add si,26
    call WRD_TO_DEC
     lea dx,ExtendedMemorySize
    call PrintMsg

    pop si
    pop dx
    pop bx
    pop ax
```

```
    ret
PrintExtendedMemorySize ENDP


PrintMCB PROC near
    ; Address
    lea di,MCB
    mov ax,es
    add di,5
    call WRD_TO_HEX

    ; Type
    lea di,MCB
    add di,15
    xor ah,ah
    mov al,es:[0]
    call BYTE_TO_HEX
    mov [di],al
    inc di
    mov [di],ah

    ; Owner
    lea di,MCB
    mov ax,es:[1]
    add di,29
    call WRD_TO_HEX

    ; Size
```

```
    lea di,MCB
    mov ax,es:[3]
    mov bx,10h
    mul bx
    add di,46
    push si
    mov si,di
    call WRD_TO_DEC
    pop si

    ; Name
    lea di,MCB
    add di,53
    mov bx,0
print_:
    mov dl,es:[bx+8]
    mov [di],dl
    inc di
    inc bx
    cmp bx,8
    jne print_
    mov ax,es:[3]
    mov bl,es:[0]
    ret
PrintMCB ENDP


PrintMemoryManagementUnits PROC near
```

```
    lea dx,TableHead
    call PrintMsg
    mov ah,52h
    int 21h
    sub bx,2h
    mov es,es:[bx]
metka_1:
    call PrintMCB
    lea dx,MCB
    call PrintMsg
    mov cx,es
    add ax,cx
    inc ax
    mov es,ax
    cmp bl,4Dh
    je metka_1
    ret
PrintMemoryManagementUnits ENDP


; _____________________________________________
; Код

BEGIN:
    call PrintAvailableMemory
    call PrintExtendedMemorySize
    call PrintMemoryManagementUnits
```

```
    xor al,al
    mov ah,4ch
    int 21h

TESTPC ENDS
    END START
```

**Приложение Б. code.asm.**

```
AStack SEGMENT STACK
     DW 100h DUP(?)
AStack ENDS

;_____________________________________________
DATA SEGMENT

wasloaded db 'Interruption had already been loaded.',0DH,0AH,'$'
unloaded db 'Interruption is restored.',0DH,0AH,'$'
loading db 'Interruption is loaded.',0DH,0AH,'$'

DATA ENDS


;_____________________________________________
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, ES:DATA, SS:AStack


PrintMsg PROC NEAR
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
PrintMsg ENDP
```

```
setCurs PROC
    push ax
    push bx
    push cx

    mov ah,02h
    mov bh,00h
    int 10h

    pop cx
    pop bx
    pop ax
    ret
setCurs ENDP


getCurs PROC
    push ax
    push bx
    push cx

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

    pop cx
```

```
    pop bx
    pop ax
    ret
getCurs ENDP


ROUT PROC FAR
    jmp ROUT_
_DATA:
    SIGN db '0000'
    KEEP_CS dw 0
    KEEP_IP dw 0
    KEEP_PSP dw 0
    VALUE db 0
    COUNTER db '    Number of calls: 00000    $'
    STACK_    dw   64 dup (?)
    KEEP_SS dw 0
    KEEP_AX   dw ?
    KEEP_SP dw 0

ROUT_:
    mov KEEP_SS,ss
    mov KEEP_AX,ax
    mov KEEP_SP,sp
    mov ax,seg STACK_
    mov ss,ax
    mov sp,0
    mov ax,KEEP_AX
```

```
    push ax
    push dx
    push ds
    push es
    cmp VALUE,1
    je ROUT_RES
    call getCurs
    push dx
    mov dh,22
    mov dl,45
    call setCurs

ROUT_SUM:
    push si
    push cx
    push ds
    push ax
    mov ax,SEG COUNTER
    mov ds,ax
    mov bx,offset COUNTER
    add bx,22
    mov si,3
next_:
    mov ah,[bx+si]
    inc ah
    cmp ah,58
    jne ROUT_NEXT
```

```
    mov ah,48
    mov [bx+si],ah
    dec si
    cmp si,0
    jne next_
ROUT_NEXT:
    mov [bx+si],ah
    pop ds
    pop si
    pop bx
    pop ax
    push es
    push bp
    mov ax,SEG COUNTER
    mov es,ax
    mov ax,offset COUNTER
    mov bp,ax
    mov ah,13h
    mov al,0
    mov cx,30
    mov bh,0
    int 10h
    pop bp
    pop es
    pop dx
    call setCurs
    jmp ROUT_END
```

```
ROUT_RES:
    cli
    mov dx,KEEP_IP
    mov ax,KEEP_CS
    mov ds,ax
    mov ah,25h
    mov al,1Ch
    int 21h
    mov es,KEEP_PSP
    mov es,es:[2Ch]
    mov ah,49h
    int 21h
    mov es,KEEP_PSP
    mov ah,49h
    int 21h
    sti

ROUT_END:
    pop es
    pop ds
    pop dx
    pop ax

    mov ax,KEEP_SS
    mov ss,ax
    mov sp,KEEP_SP
    mov ax,KEEP_AX
```

```
    iret
ROUT ENDP


CHECKING PROC
    mov ah,35h
    mov al,1Ch
    int 21h
    mov si,offset SIGN
    sub si,offset ROUT
    mov ax,'00'
    cmp ax,es:[bx+si]
    jne UNLOAD
    cmp ax,es:[bx+si+2]
    je LOAD

UNLOAD:
    call SET_INTERRUPT
    mov dx,offset LAST_BYTE
    mov cl,4
    shr dx,cl
    inc dx
    add dx,CODE
    sub dx,KEEP_PSP
    xor al,al
    mov ah,31h
    int 21h
LOAD:
```

```
    push es
    push ax
    mov ax,KEEP_PSP
    mov es,ax
    cmp byte ptr ES:[82h],'/'
    jne BACK
    cmp byte ptr ES:[83h],'u'
    jne BACK
    cmp byte ptr ES:[84h],'n'
    je UNLOAD_
BACK:
    pop ax
    pop es
    mov dx,offset wasloaded
    call PrintMsg
    ret
UNLOAD_:
    pop ax
    pop es
    mov byte ptr ES:[BX+SI+10],1
    mov dx,offset unloaded
    call PrintMsg
    ret
CHECKING ENDP


SET_INTERRUPT PROC
    push dx
```

```
    push ds
    mov ah,35h
    mov al,1Ch
    int 21h
    mov KEEP_IP,bx
    mov KEEP_CS,es
    mov dx,offset ROUT
    mov ax,seg ROUT
    mov ds,AX
    mov ah,25h
    mov al,1Ch
    int 21h
    pop ds
    mov dx,offset loading
    call PrintMsg
    pop dx
    ret
SET_INTERRUPT ENDP


BEGIN:
    mov ax,DATA
    mov ds,ax
    mov KEEP_PSP,es
    call CHECKING
    xor al,al
    mov ah,4Ch
    int 21H
```

```
LAST_BYTE:

CODE ENDS
    END BEGIN
```