

**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. 9383

\_\_\_\_\_

Орлов Д.С.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2021

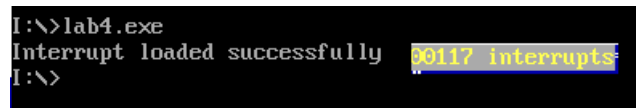
## Цель работы

Построить обработчик прерываний сигналов таймера. Эти сигналы генерируются аппаратурой через определенные интервалы времени и, при возникновении такого сигнала, возникает прерывание с определенным значением вектора.

## Выполнение работы

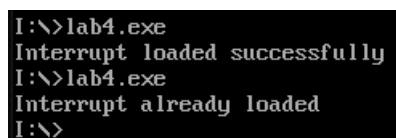
**Шаг 1.** Был написан и отлажен программный модуль типа .EXE, который выполняет следующие функции:

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 соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h.
4. Выгрузка прерывания по соответствующему значению параметра в командной строке \un. Выгрузка прерывания состоит в восстановлении стандартного вектора прерываний и освобождения памяти, занимаемой резидентом. Затем осуществляется выход по функции 4ch прерывания int 21h.



```
I:\>lab4.exe
Interrupt loaded successfully 0017 interrupts
I:\>
```

Рис. 1. Работа установленного прерывания



```
I:\>lab4.exe
Interrupt loaded successfully
I:\>lab4.exe
Interrupt already loaded
I:\>
```

Рис. 2. Пример обработки попытки повторной установки прерывания

```
I:\>lab4.exe /un
Interrupt unloaded successfully
I:\>
```

Рис. 3. Пример выгрузки прерывания  
(после запуска программы счетчик остановился).

**Шаг 2.** Убеждаемся, что резидентный обработчик прерывания 1Ch установлен и программа работает исправно. Для этого была запущена программа из лабораторной работы №3:

```
I:\>lab4.exe
Interrupt loaded successfully
I:\>lab3_1.com
Available memory (bytes): 647792
Extended memory (bytes): 245920
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: 944 SC/SD: LAB4
Address: 01CD PSP address: 01D8 Size: 144 SC/SD:
Address: 01D7 PSP address: 01D8 Size: 647792 SC/SD: LAB3_1
```

Рис. 4. Пример работы программы из лабораторной работы №3

## Контрольные вопросы

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

Сигнал от часов генерируется аппаратурой через определенные интервалы времени (около 18 раз в секунду). За каждым таким сигналом следует возникновение прерывания с вектором 1Ch. Соответственно, при генерации сигнала управление передается функции, определенной в таблице прерываний с номером 1Ch.

### 2. Какого типа прерывания использовались в работе?

В работе использовались программные прерывания 21h и 10h, и аппаратное прерывание 21h с вектором 1Ch.

## Заключение

В процессе выполнения лабораторной работы был изучен механизм работы аппаратного таймера, а также были получены навыки реализации собственных резидентных прерываний.

## Приложение А

```
stack segment stack
```

```
    db 256 dup (?)
```

```
stack ends
```

```
data segment
```

```
flag db 0
```

```
ls db 'Interrupt loaded successfully$' ;успешно загружено
```

```
us db 'Interrupt unloaded successfully$' ;успешно выгружено
```

```
ial db 'Interrupt already loaded$'
```

```
iau db 'Interrupt already unloaded$'
```

```
data ends
```

```
code segment
```

```
assume CS:code, DS:data
```

```
inter proc far
```

```
    jmp begint
```

```
ID dw 0FFFFh
```

```
PSP dw ?
```

```
keepCS dw 0
```

```
keepIP dw 0
```

```
keepSS dw 0
```

```
keepSP dw 0
```

```
keepAX dw 0
```

```
intstr db '00000 interrupts'
```

```
lenstr = $ - intstr
```

```
intstk db 128 dup (?)
```

```
endstk:
```

```
begint: mov keepSS, SS
```

```

mov    keepSP, SP
mov    keepAX, AX
mov    AX, CS
mov    SS, AX
mov    SP, offset endstk
push   BX
push   CX
push   DX
push   DS
push   ES
push   SI
push   DI
push   BP
; -----
    mov    AH, 03h    ;получение курсора
    mov    BH, 0
    int    10h
    push   DX
; -----
    mov    AH, 02h    ;установка курсора
    mov    BH, 0
    mov    DX, 0
    int    10h
; -----
    push   BP
    push   DS
    push   SI
    mov    DX, seg intstr
    mov    DS, DX
    mov    SI, offset intstr
    mov    CX, 5
incr:  mov    BP, CX

```

```

    dec    BP
    mov    AL, byte ptr [SI+BP]
    inc    AL
    mov    [SI+BP], AL
    cmp    AL, 3Ah
    jne    good
    mov    AL, 30h
    mov    byte ptr [SI+BP], AL
    loop   incr
good: pop    SI
    pop    DS

    push   ES
    mov    DX, seg intstr
    mov    ES, DX
    mov    BP, offset intstr
    mov    AH, 13h
    mov    AL, 1
    mov    BH, 0
    mov    CX, lenstr
    mov    DX, 0
    int    10h
    pop    ES
    pop    BP
; -----
    mov    AH, 02h    ;возвращаем курсор
    mov    BH, 0
    pop    DX
    int    10h

    pop    BP
    pop    DI

```

```

pop    SI
pop    ES
pop    DS
pop    DX
pop    CX
pop    BX
mov    AX, keepSS
mov    SS, AX
mov    SP, keepSP
mov    AX, keepAX
mov    AL, 20h
out    20h, AL
iret

```

endint:

inter endp

load proc

```

push   AX
push   CX
push   DX

```

; -----

```

mov    AH, 35h ;в программе при загрузке обработчика прерывания
mov    AL, 1Ch
int     21h
mov    keepIP, BX
mov    keepCS, ES

```

; -----

```

push   DS ;Настройка прерывания
mov    DX, offset inter
mov    AX, seg inter
mov    DS, AX
mov    AH, 25h

```

```
mov    AL, 1Ch
int     21h
pop     DS
```

```
mov     DX, offset endint
mov     CL, 4
shr     DX, CL
inc     DX
mov     AX, CS
sub     AX, PSP
add     DX, AX
xor     AX, AX
mov     AH, 31h
int     21h
pop     DX
pop     CX
pop     AX
ret
```

```
load    endp
```

```
unload proc
```

```
push    AX
push    DX
push    SI
push    ES
```

```
cli          ;в программе при выгрузке обработчика прерывания
push    DS
mov     AH, 35h
mov     AL, 1Ch
int     21h
mov     SI, offset keepCS
```



```

sub    SI, offset inter
mov    DX, ES:[BX+SI+2]
mov    AX, ES:[BX+SI]
mov    DS, AX
mov    AH, 25h
mov    AL, 1Ch
int    21h
pop    DS
mov    AX, ES:[BX+SI-2]
mov    ES, AX
push   ES
mov    AX, ES:[2Ch]
mov    ES, AX
mov    AH, 49h
int    21h
pop    ES
mov    AH, 49h
int    21h
sti
pop    ES
pop    SI
pop    DX
pop    AX
ret

```

unload endp

isParam proc

```

push   AX
mov    AL, ES:[82h]
cmp    AL, '/'
jne    nparam
mov    AL, ES:[83h]

```

```

    cmp    AL, 'u'
    jne    nparam
    mov    AL, ES:[84h]
    cmp    AL, 'n'
    jne    nparam
    mov    flag, 1
nparam: pop    AX
    ret
isParam endp

```

```

isLoad proc
    push    AX
    push    DX
    push    SI
    mov     flag, 1
    mov     AH, 35h
    mov     AL, 1Ch
    int     21h
    mov     SI, offset ID
    sub     SI, offset inter
    mov     DX, ES:[BX+SI]
    cmp     DX, 0FFFFh
    je      ld
    mov     flag, 0
ld:  pop     SI
    pop     DX
    pop     AX
    ret
isLoad endp

```

```

PRINT_STR proc
    push    AX

```

```

    mov    AH, 09h
    int    21h
    pop    AX
    ret
PRINT_STR endp

```

```

main  proc far
    mov    AX, data
    mov    DS, AX
    mov    PSP, ES
    mov    flag, 0
    call   isParam
    cmp    flag, 1
    je     un

    call   isLoad    ;Loading
    cmp    flag, 0
    je     notld
    mov    DX, offset ial
    call   PRINT_STR
    jmp    fin

notld: mov    DX, offset ls
    call   PRINT_STR
    call   load
    jmp    fin

un:     call   isLoad    ;Unloading
    cmp    flag, 0
    jne    alrld
    mov    DX, offset iau
    call   PRINT_STR
    jmp    fin

```

```
alrld: call  unload
        mov   DX, offset us
        call  PRINT_STR

fin:  mov   AX, 4C00h   ;завершение
      int   21h
main  endp
code  ends
      end   main
```