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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Операционные системы»

Тема: Обработка стандартных прерываний

Студент гр. 8383 _____ Ларин А.

Преподаватель _____ Ефремов М.А,

Санкт-Петербург

2020

Цель работы.

Научится обрабатывать стандартные прерывания и загружать в память резидентные программы. Написать резидентный обработчик сигнала таймера.

Выполнение

Написан код .EXE модуля, который проверяет наличие переопределенного обработчика прерывания, парсит агрументы коммандной строки в поиске «/un» и в зависимости от его наличия загружает либо выгружает реализованный обработчик прерываний таймера

При помощи программы из лабораторной работы 3 проверяется состояние памяти на каждом этапе.

Состояние памяти до загрузки обработчика в память представлено в приложении А, Mem_Init.

Загрузка в память и работа прерывания продемонстрирована на рис. 1 в приложении А.

Состояние памяти после загрузки обработчика в память представлено на в приложении А, Mem_Loaded.

Результат попытки повторной загрузки обработчика представлен на рис. 2 в приложении А.

Результат выгрузки продемонстрирован на рис. 3 в приложении А.

Состояние памяти после выгрузки представлено в приложении А, Mem_Unoaded.

Контрольные вопросы

1. Как реализован механизм прерывания от часов?

   Прерывание вызывается аппаратно. Оно отночится к маскируемым, потому проверяется флаг I. Если прерывания разрешены происходит вызов обработчика. Для этого в стак помещается регистр флагов и

управление передается функции по адресу из вектора прерываний(в нашем случае нашему кастомному обработчику). При этом контроллер прерываний запрещает вызов прерываний с более низким приоритетом. Обработчик исполняется.

В конце работы обработчик восстанавливает регистры, разрешает исполнение прерываний с более низким приоритетом и командой iret возвращает управление, предварительно восстановив значения флагов

2. Какого типа прерывания использовались в программе?

   Прерывание таймера. Маскируемое, аппаратное.

Выводы.

В результате работы были разобраны некоторые концепции языка ассемблера и работы операционной системы DOS. Были исследованы способы загрузки резидентной программы и установки своего обработчика прерываний. Написан модуль EXE с требуемым функционалом.

ПРИЛОЖЕНИЕ А

Mem_Init – Состояние памяти до загрузки обработчика в память

```
Avaible memory: E6D0
 Extended memory: 3C00

 MCB 0001
 Owner  MS DOS

 Size 0010



 MCB 0002
 Owner  free

 Size 0040



 MCB 0003
 Owner 0040
 Size 0100



 MCB 0004
 Owner 0192
 Size 0090



 MCB 0005
 Owner 0192
 Size E6D0

L3_1
```

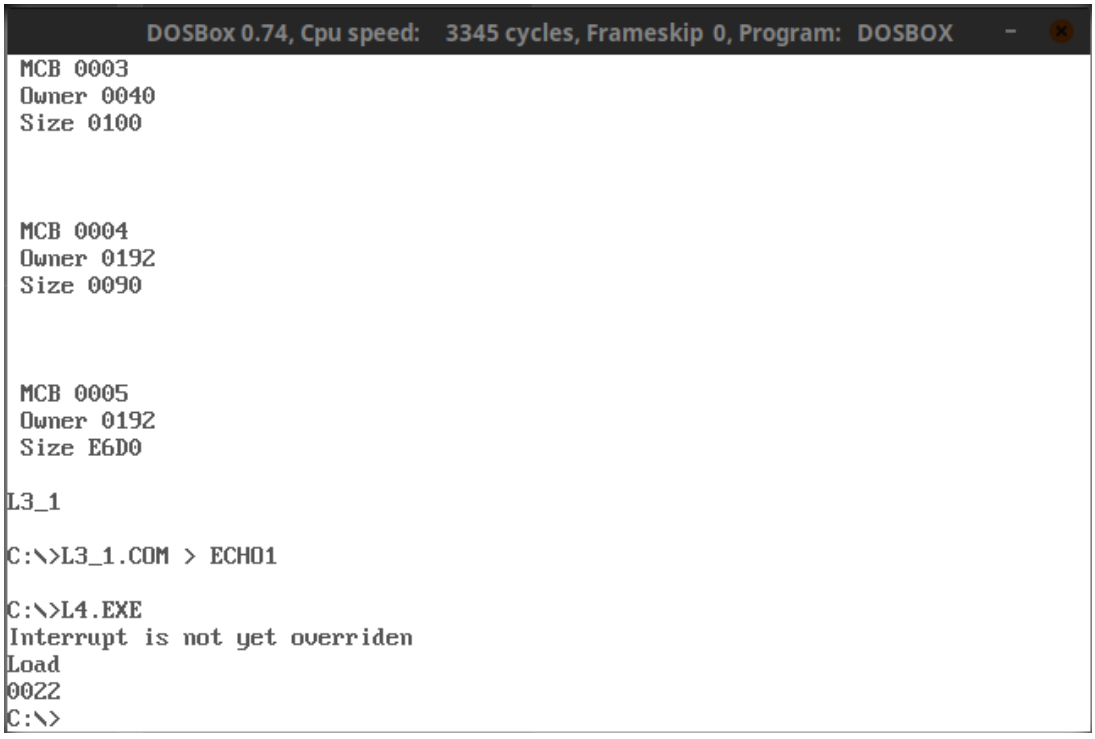

Рисунок 1 – Загрузка в память и работа прерывания

Mem_Loaded – Состояние памяти после загрузки обработчика в память

```
Avaible memory: E120
Extended memory: 3C00

MCB 0001
Owner  MS DOS

Size 0010



MCB 0002
Owner  free

Size 0040



MCB 0003
Owner 0040
Size 0100



MCB 0004
Owner 0192
Size 0090



MCB 0005
Owner 0192
Size 0500

L4

MCB 0006
Owner 01ED
Size 0090



MCB 0007
Owner 01ED
Size E120

L3_1
```

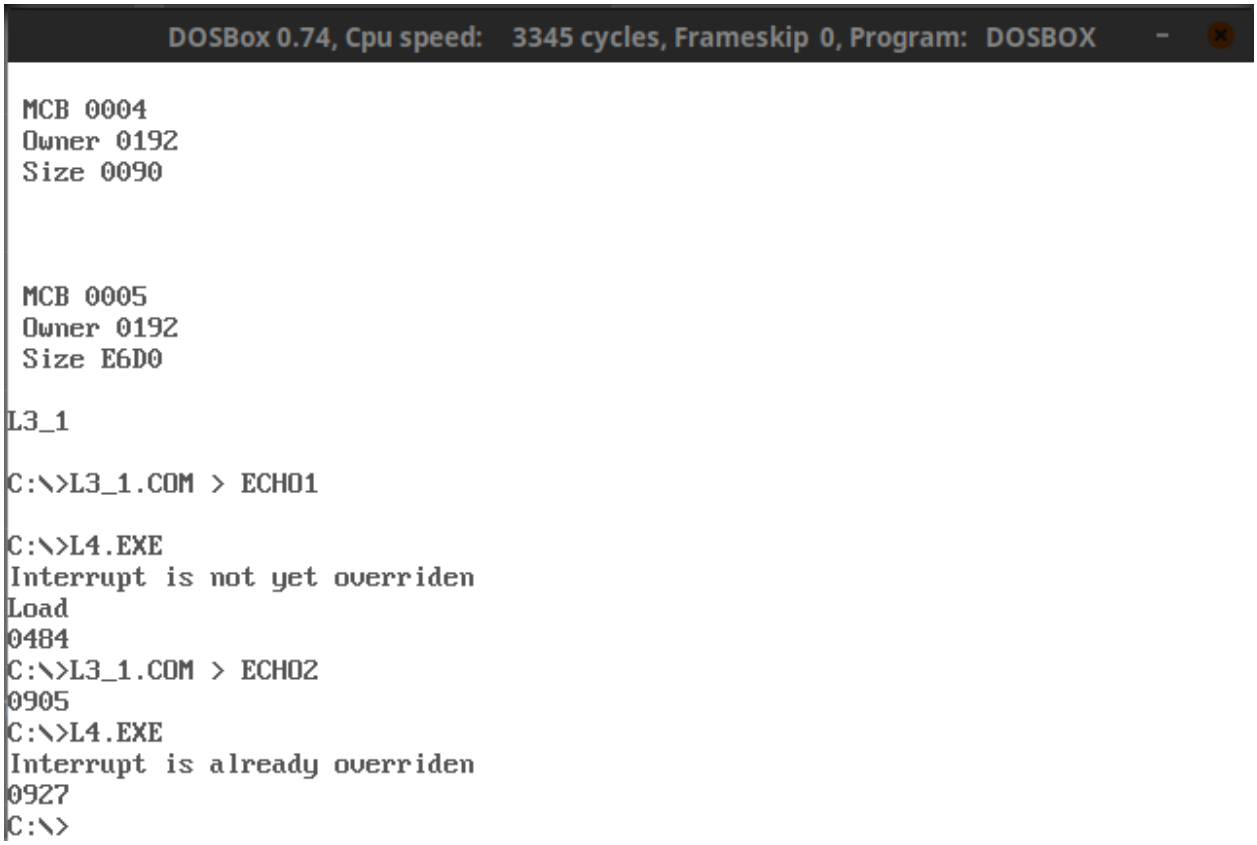

Рисунок 2 – Повторная загрузка обработчика

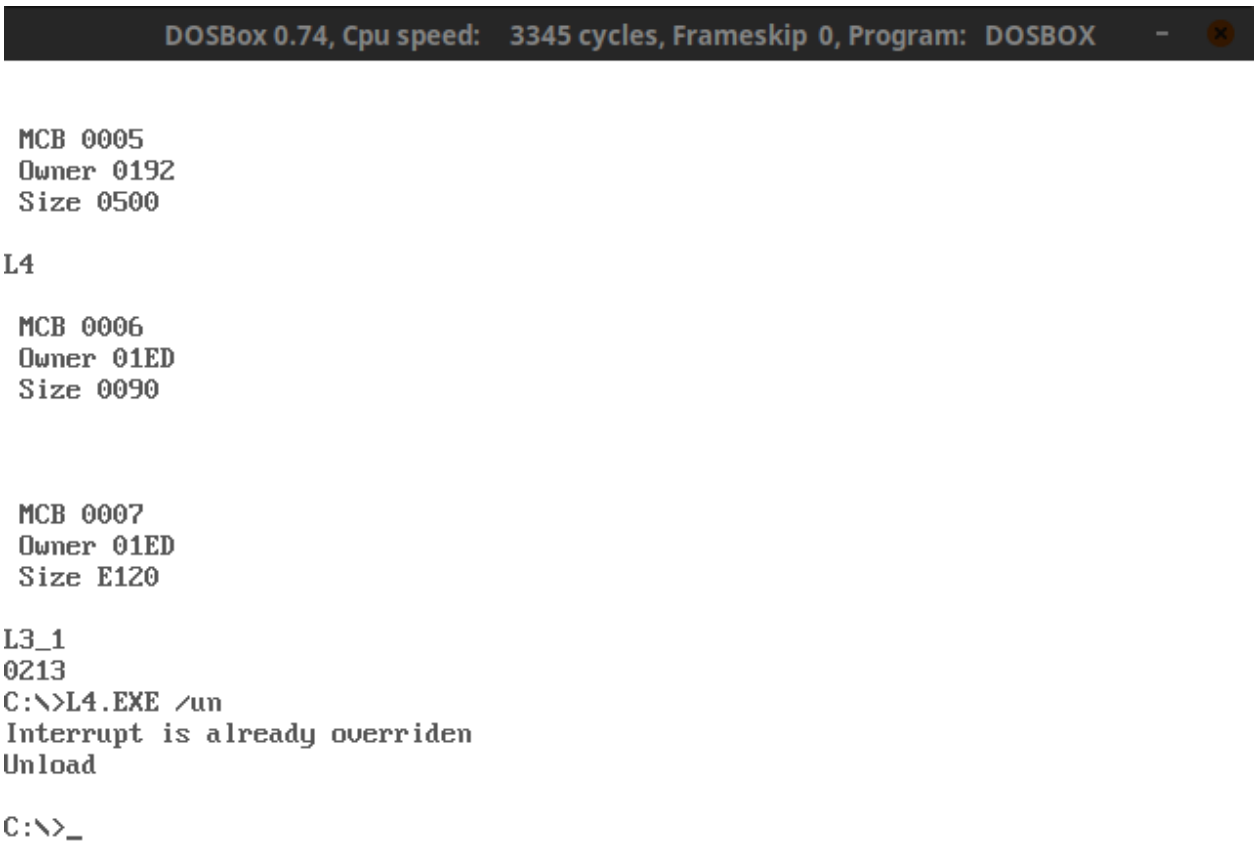

Рисунок 3 – Выгрузка обработчика

Mem_Unloaded – Состояние памяти после выгрузки обработчика

```
Avaible memory: E6D0
Extended memory: 3C00

MCB 0001
Owner  MS DOS

Size 0010



MCB 0002
Owner  free

Size 0040



MCB 0003
Owner 0040
Size 0100



MCB 0004
Owner 0192
Size 0090



MCB 0005
Owner 0192
Size E6D0

L3_1
```

ПРИЛОЖЕНИЕ Б

L4.ASM

```
DATA SEGMENT
  IS_OVERRIDEN db 0
  UN_ARG db 0
  STR_NEW_LINE db 0DH,0AH,'$'
  STR_OVERRIDEN db 'Interrupt is already overriden$'
  STR_NOT_OVERRIDEN db 'Interrupt is not yet overriden$'
  STR_UN_ARG db 'Unload $'
  STR_NO_UN_ARG db 'Load $'

DATA ENDS

PSTACK SEGMENT STACK
  dw 128 dup(0)
PSTACK ENDS


CODE SEGMENT
  assume CS:CODE, DS:DATA, SS:PSTACK, ES:NOTHING
;===============================================
  INTERRUPT_HANDLER PROC FAR
    jmp BEGIN
    STR_CNT db '0000$'

    SAVE_AX dw 0
    SAVE_SS dw 0
    SAVE_SP dw 0
    KEEP_IP dw 0
    KEEP_CS dw 0
    PSP_SEGMENT DW 0
    ATR db 0
    HANDLER_ID dw 1234h
    HANDLER_STACK dw 128 dup(0)

  BEGIN:
    mov SAVE_AX, AX
    mov SAVE_SP, SP
    mov SAVE_SS, SS
    mov AX, SEG HANDLER_STACK
    mov SS, AX
    mov AX, offset HANDLER_STACK
    add AX, 256
    mov SP, AX
    push BX
```

```
  push CX
  push DX
  push SI
  push DS
  push BP
  push ES



  mov AX, SEG STR_CNT
  mov DS, AX

  mov AH, 03h
  mov BH, 00h
  int 10h
  push DX;Keep cursor pos

  call PRINT_AL


  mov AH, 02h
  mov BH, 00h
  mov DX, 1650h;Upon last line
  int 10h
  mov AX, SEG STR_CNT
  push DS
  mov DS, AX
  mov SI, offset STR_CNT
  add SI, 4
  mov CX, 4
CNT_INC:
  mov AH, [SI]
  inc AH
  mov [SI], AH
  cmp AH, '9'
  jbe END_INC
  mov AH, '0'
  mov [SI], AH
  dec SI
  loop CNT_INC
END_INC:
  pop DS

  push ES
  push BP
  mov AX, SEG STR_CNT
  mov ES, AX
  mov BP, offset STR_CNT
```

```
    mov AH, 13h;Print line
    mov AL, 1h;Use BL atr
    mov CX, 4;Len
    mov BH, 0;Pg
    mov bl,ATR;Atr. Set on handler set
    int 10h
    pop BP
    pop ES

    pop DX;Reset cursor
    mov AH, 02h
    mov BH, 0h
    int 10h

    pop ES
    pop BP
    pop DS
    pop SI
    pop DX
    pop CX
    pop BX
    mov SP, SAVE_SP
    mov AX, SAVE_SS
    mov SS, AX
    mov AX, SAVE_AX
    mov AL, 20h
    out 20h, AL
    iret
  INTERRUPT_HANDLER ENDP
;--------------------------------------------------------

  PRINT_WORD proc near
  ;AX - word
    xchg AL,AH
    call PRINT_AS_HEX
    xchg AL,AH
    call PRINT_AS_HEX
    ret
  PRINT_WORD ENDP
;--------------------------------------------------------
  PRINT_AS_HEX proc near
  ;AL - number
  ;breaks AX,CX,BX
    push ax
    ;push bx
    push cx
    ;push dx
    ;mov bx,dx
```

```
    mov ch,al
    mov cl,4
    shr al,cl
;   call DIGIT_TO_CHAR
    mov al,ch
;   call DIGIT_TO_CHAR
    ;pop dx
    pop cx
    ;pop bx
    pop ax
    ret
 PRINT_AS_HEX ENDP
;--------------------------------------------------------
;DIGIT_TO_CHAR PROC near
; ;AL
;   and al,0Fh
;   cmp al,09h
;   jle BLW
;   add al,'A'
;   sub al, 0Ah
;   jmp DTC_CONT
;BLW:
;   add al,'0'
;DTC_CONT:
;   call PRINT_AL
;   ret
;DIGIT_TO_CHAR ENDP
;--------------------------------------------------------
 PRINT_AL PROC
    push ax
    push bx
    push cx


    xor ax,ax
    xor bx,bx
    mov bl,ATR
    mov ah,09h
    mov bh,0
    mov cx,1
    int 10h

    pop cx
    pop bx
    pop ax
    ret
 PRINT_AL ENDP
```

```
 HANDLER_MEM_EDGE:
;=================================================
 CHECK_INTERRUPT_OVERRIDE PROC
 ;result: IS_OVERRIDEN
   push AX
   push BX
   push SI
   mov IS_OVERRIDEN, 0
   mov AH, 35h
   mov AL, 1Ch
   int 21h
   mov  SI, offset HANDLER_ID
   sub SI, offset INTERRUPT_HANDLER
   mov AX, ES:[BX + SI]
   cmp AX, 1234h
   jne NOT_OVERRIDEN
   mov IS_OVERRIDEN, 1
 NOT_OVERRIDEN:

   pop SI
   pop BX
   pop AX
   ret
 CHECK_INTERRUPT_OVERRIDE ENDP
;=================================================
 CHECK_UN_ARG PROC
   push AX
   push ES

   mov AX, PSP_SEGMENT
   mov ES, AX
   cmp byte ptr ES:[82h], '/'
   jne CHECK_UN_ARG_END
   cmp byte ptr ES:[83h], 'u'
   jne CHECK_UN_ARG_END
   cmp byte ptr ES:[84h], 'n'
   jne CHECK_UN_ARG_END
   mov UN_ARG, 1

 CHECK_UN_ARG_END:
   pop ES
   pop AX
   ret
 CHECK_UN_ARG ENDP
;=================================================
 LOAD_HANDLER PROC
```

```
    push AX
    push BX
    push CX
    push DX
    push DS
    push ES

    mov AH, 35h;Keep prev handler
    mov AL, 1Ch
    int 21h
    mov KEEP_CS, ES
    mov KEEP_IP, BX
    push DS;Set new handler
    mov DX, offset INTERRUPT_HANDLER
    mov AX, SEG INTERRUPT_HANDLER
    mov DS, AX
    mov AH, 25h
    mov AL, 1Ch
    int 21h
    pop DS
    mov DX, offset HANDLER_MEM_EDGE;Make resident
    add DX, 300h ;Wtf?
    mov CL, 4h; Leave DX*16 bytes(in pr)
    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
  LOAD_HANDLER ENDP
;=========================================================
  UNLOAD_HANDLER PROC

    push AX
    push BX
    push DX
    push DS
    push ES
    push SI

    CLI
```

```
    mov AH, 35h
    mov AL, 1Ch
    int 21h; get prev set vector
    mov SI, offset KEEP_IP;get it's fields
    sub SI, offset INTERRUPT_HANDLER
    mov DX, ES:[BX + SI];IP
    mov AX, ES:[BX + SI + 2];CS
    push DS
    mov DS, AX
    mov AH, 25h
    mov AL, 1Ch
    int 21h; set g'old vector
    pop DS
    mov AX, ES:[BX + SI + 4];psp
    mov ES, AX
    push ES
    mov AX, ES:[2Ch];Env
    mov ES, AX
    mov AH, 49h
    int 21h
    pop ES;PSP
    mov AH, 49h
    int 21h

    pop SI
    pop ES
    pop DS
    pop DX
    pop BX
    pop AX

    STI

    ret
  UNLOAD_HANDLER ENDP

;UTILS
;=================================================
  PRINT PROC NEAR
        PUSH AX
        MOV AH, 09H
        INT 21H
    POP AX
    RET
  PRINT   ENDP
;=================================================
  LN PROC
```

```
    push AX
    push DX
    mov DX, offset STR_NEW_LINE
    mov AH, 9h
    int 21h
    pop DX
    pop AX
    ret
  LN ENDP
;=====================================================



;==========MAIN=====MAIN====MAIN====MAIN===============
  MAIN PROC
  ;MAIN init
    mov  ax, DATA                          ;ds setup
      mov  ds, ax
    mov PSP_SEGMENT, ES

    mov ah,08h
    mov bh,0
    int 10h
    mov ATR,ah
  ;CHECK OVERRIDEN
    call CHECK_INTERRUPT_OVERRIDE
    cmp IS_OVERRIDEN, 1
    je PRINT_OVERRIDEN
    mov DX,offset STR_NOT_OVERRIDEN
    call PRINT
    call LN
    jmp END_OVERRIDEN_PRINT
  PRINT_OVERRIDEN:
    mov DX,offset STR_OVERRIDEN
    call PRINT
    call LN
  END_OVERRIDEN_PRINT:
  ;CHECK ARGUMENT /un
    call CHECK_UN_ARG
    cmp UN_ARG, 1
    je PRINT_UN_ARG
    cmp IS_OVERRIDEN,1
    je END_MAIN
    mov DX,offset STR_NO_UN_ARG
    call PRINT
    call LN
    jmp LOAD
    jmp END_UN_ARG_PRINT
```

```
  PRINT_UN_ARG:
    mov DX,offset STR_UN_ARG
    call PRINT
    call LN
    jmp UNLOAD
  END_UN_ARG_PRINT:

  ;LOAD
  LOAD:
    call LOAD_HANDLER
    jmp END_LOAD
  END_LOAD:
    jmp END_MAIN

  ;UNLOAD
  UNLOAD:
    cmp IS_OVERRIDEN,1
    jne END_MAIN
    call UNLOAD_HANDLER
    jmp END_UNLOAD
  END_UNLOAD:
    jmp END_MAIN


  END_MAIN:
    xor AL, AL
    mov AH, 4Ch
    int 21h


  MAIN ENDP


CODE ENDS




END MAIN
```