**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра MOЭBM**

отчет

**по лабораторной работе №4**

**по дисциплине «Операционные системы»**

**Тема: Обработка стандартных прерываний**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6381 |  | Вергейчик Г. Л. |
| Преподаватель |  | Губкин А. Ф. |

Санкт-Петербург

2018

**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 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.

**Описание функций**

* ROUT - Функция обработчика прерывания.
* IS\_LOADED - Функция проверки закружен ли резидент в память.
* IS\_UNLOADED - Функция проверки ключа выгрузки резидента из памяти.
* RES\_LOAD - Функция загрузки резидента в память.
* RES\_UNLOAD - Функция выгрузки резидента из памяти.
* PRINT - Функция печати на экран.

**Описание данных**

* resNotSet - Применяется для вывода информации о том, что резидент не загружен в память.
* resUnload - Применяется для вывода информации о том, что резидент выгружен из памяти.
* resAlrSet - Применяется для вывода информации о том, что резидент уже загружен в память.
* resLoad - Применяется для вывода информации о том, что резидент загружен в память.
* COUNTER - Счетчик обработчика прерывания
* INTER\_ADR - Применяется для проверки того, загружен ли уже резидент в память или нет.
* KEEP\_IP - Применяется для запоминания смещения вектора прерывания.
* KEEP\_CS - Применяется для запоминания сегмента вектора прерывания.

**Ход работы:**

**Шаг 1:**

Загрузка резидента в память(см. Рис1).

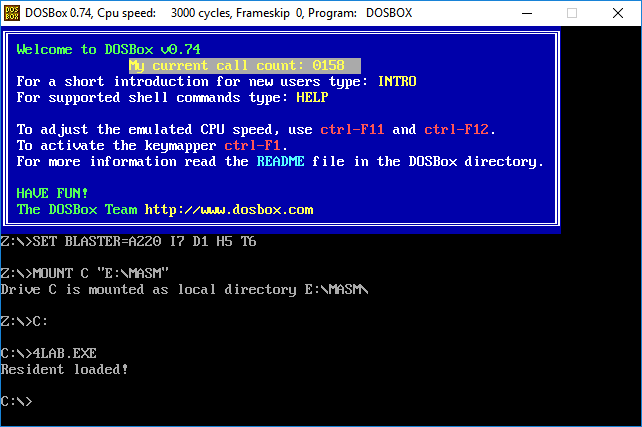


Рис. 1 Загрузка резидента в память

**Шаг 2:**

Размещение прерывания в памяти(см. Рис2).

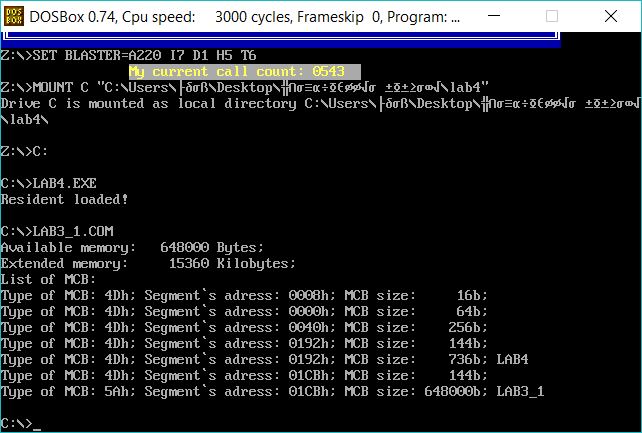


Рис. 2 Размещение прерывания в памяти

**Шаг 3:**

Повторная загрузка резидента в память(см. Рис3).

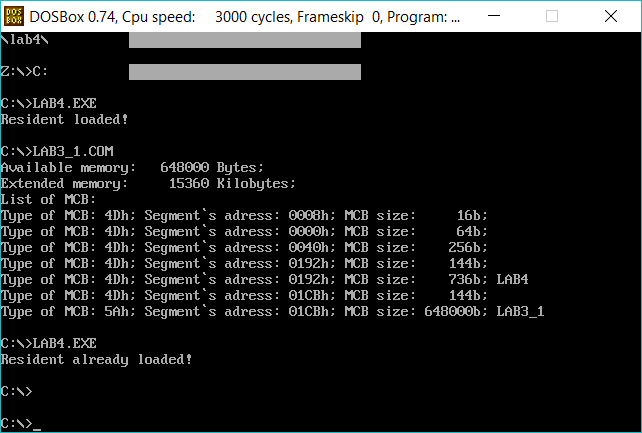


Рис. 3 Повторная загрузка резидента в память

**Шаг 4:**

Состояние памяти после выгрузки резидента(см. Рис4).

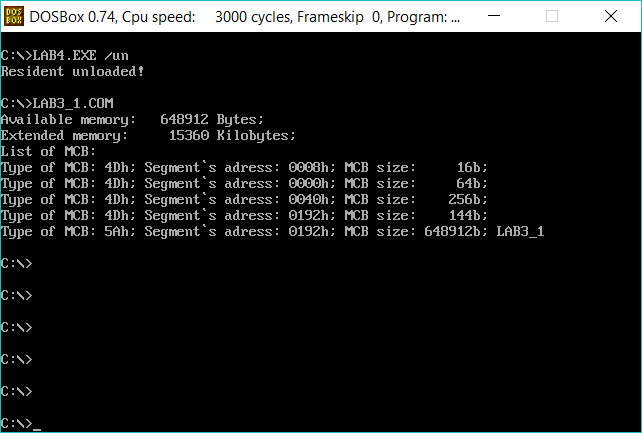


Рис. 4 Состояние памяти после выгрузки резидента

**Ответы на контрольные вопросы:**

1. Как реализован механизм прерывания от часов?

При каждом такте таймера (часов) сначала сохраняется состояние регистров, затем определяется источник прерывания (по номеру источника прерывания определяется смещение в таблице векторов прерываний), далее эти данные сохраняются, т. е. - первые два байта помещаются в IP, второе два байта в CS, далее запускается обработчик прерывания по адресу CS:IP и обрабатывается прерывание, в конце происходит возврат управления прерванной программе

1. Какого типа прерывания использовались в работе?

Были использованы пользовательские прерывания, такие как int 10h и int 21h и аппаратные прерывания (1Ch).

**Вывод**

В результате выполнения данной лабораторной работы были исследованы организация и управление прерываниями. Была разработана программа, в которой построен обработчик прерываний сигналов таймера.

**Приложение**

1. Файл lab4.asm

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

CODE SEGMENT

SSTACK SEGMENT STACK

DW 256 DUP(?)

SSTACK ENDS

;----------------------------------------------------------------------

;-------------------------

localStack db 256 dup(?) ;локальный стек

;-------------------------

ROUT PROC FAR

jmp start

;-------------------------

KEEP\_SS dw 0 ;для сохранения внешнего стека

KEEP\_SP dw 0

;-------------------------

PSP\_AD1 dw 0

PSP\_AD2 dw 0

KEEP\_CS dw 0

KEEP\_IP dw 0

INTER\_ADR dw 1234h

COUNTER db 'My current call count: 0000 $'

start:

;-------------------------

MOV KEEP\_SS,SS ;сохранение и настройка на локальный стек

MOV KEEP\_SP,SP

MOV AX, SEG CODE

MOV SS, AX

MOV SP, 100H

;-------------------------

push ax

push bx

push cx

push dx

mov ah, 03h

mov bh, 00h

int 10h

push dx

;установка курсора

mov ah, 02h

mov bh, 00h

mov dx, 0210h

int 10h

push si

push cx

push ds

mov ax, SEG COUNTER

mov ds, ax

mov si, offset COUNTER

add si, 1Ah

mov ah,[si]

inc ah

mov [si], ah

cmp ah, 3Ah

jne end\_count

mov ah, 30h

mov [si], ah

mov bh, [si - 1]

inc bh

mov [si - 1], bh

cmp bh, 3Ah

jne end\_count

mov bh, 30h

mov [si - 1], bh

mov ch, [si - 2]

inc ch

mov [si - 2], ch

cmp ch, 3Ah

jne end\_count

mov ch, 30h

mov [si - 2], ch

mov dh, [si - 3]

inc dh

mov [si - 3], dh

cmp dh, 3Ah

jne end\_count

mov dh, 30h

mov [si - 3],dh

end\_count:

;выводим строку на экран

pop ds

pop cx

pop si

push es

push bp

mov ax, SEG COUNTER

mov es, ax

mov ax, offset COUNTER

mov bp, ax

mov ah, 13h

mov al, 00h

mov cx, 1Dh

mov bh, 0

int 10h

pop bp

pop es

pop dx

mov ah, 02h

mov bh, 0h

int 10h

pop dx

pop cx

pop bx

pop ax

;----------------------------

MOV SS,KEEP\_SS ;восстановление внешнего стека

MOV SP,KEEP\_SP

;----------------------------

iret

ROUT ENDP

;---------------------------------------------------------------------------

unfree\_mem:

IS\_LOADED PROC NEAR

push bx

push dx

push es

mov ah, 35h ;получение вектора

mov al, 1Ch ; прерываний

int 21h

mov dx, es:[bx + 15]

cmp dx, 1234h ; проверка на совпадение кода прерывания

je to\_set

mov al, 00h

jmp end\_set

to\_set:

mov al, 01h

jmp end\_set

end\_set:

pop es

pop dx

pop bx

ret

IS\_LOADED ENDP

;--------------------------------------------------------------------------

IS\_UNLOADED PROC NEAR

push es

mov ax, ourPSP

mov es, ax

mov bx, 0082h

mov al, es:[bx]

cmp al, '/'

jne end\_metka

mov al, es:[bx+1]

cmp al, 'u'

jne end\_metka

mov al, es:[bx+2]

cmp al, 'n'

jne end\_metka

mov al, 0001h

end\_metka:

pop es

ret

IS\_UNLOADED ENDP

;------------------------------------------------------------------------

RES\_LOAD PROC NEAR

push ax

push bx

push dx

push es

mov ah, 35h

mov al, 1Ch

int 21h

mov KEEP\_IP, bx

mov KEEP\_CS, es

push ds

mov dx, offset ROUT

mov ax, seg ROUT

mov ds, ax

mov ah, 25h

mov al, 1Ch

int 21h

pop ds

mov dx, offset resLoad

call PRINT

pop es

pop dx

pop bx

pop ax

ret

RES\_LOAD ENDP

;-------------------------------------------------------------------

RES\_UNLOAD PROC NEAR

push ax

push bx

push dx

push es

mov ah, 35h

mov al, 1Ch

int 21h

cli

push ds

mov dx, es:[bx + 9+4]

mov ax, es:[bx + 7+4]

mov ds, ax

mov ah, 25h

mov al, 1Ch

int 21h

pop ds

sti

mov dx, offset resUnload

call PRINT

push es

mov cx,es:[bx+3+4]

mov es,cx

mov ah,49h

int 21h

pop es

mov cx,es:[bx+5+4]

mov es,cx

int 21h

pop es

pop dx

pop bx

pop ax

mov ah, 4Ch

int 21h

ret

RES\_UNLOAD ENDP

;----------------------------------------------------------------

PRINT PROC NEAR

push ax

mov ah, 09h

int 21h

pop ax

ret

PRINT ENDP

;---------------------------------------------------------------

DATA SEGMENT

resNotSet db "Resident didnt loaded now!", 0dh, 0ah, '$'

resUnload db "Resident unloaded!", 0dh, 0ah, '$'

resAlrSet db "Resident already loaded!", 0dh, 0ah, '$'

resLoad db "Resident loaded!", 0dh, 0ah, '$'

ourPSP dw 0

DATA ENDS

BEGIN PROC FAR

push ds

call IS\_LOADED

cmp al, 01h

je start\_prog

mov bx, 02Ch

mov ax, [bx]

mov PSP\_AD2, ax

mov PSP\_AD1, ds

start\_prog:

mov dx, ds

sub ax, ax

xor bx, bx

mov ax, DATA

mov ds, ax

mov ourPSP, dx

xor dx, dx

call IS\_UNLOADED

cmp al, 01h

je unload\_block

call IS\_LOADED

cmp al, 01h

jne not\_load\_block

mov dx, offset resAlrSet

call PRINT

jmp exit\_block

not\_load\_block:

call RES\_LOAD

lea DX, unfree\_mem

mov CL,4

shr DX,CL

inc DX

add DX,CODE

sub dx, psp\_ad1

xor AL,AL

mov AH,31h

int 21h

mov ax, 3100h

int 21h

unload\_block:

call IS\_LOADED

cmp al, 00h

je not\_set\_block

call RES\_UNLOAD

jmp exit\_block

not\_set\_block:

mov dx, offset resNotSet

call PRINT

jmp exit\_block

exit\_block:

mov ah, 4Ch

int 21h

BEGIN ENDP

CODE ENDS

END BEGIN