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 _____ Трушников А.П.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2019

Цель работы.

Построение обработчика прерываний сигналов таймера.

Описание функций.

Название	Описание
Write_message	Вывод сообщения на экран
outputAL	Вывод в текущее положение курсора символа из AL
SaveCurs	Запоминает в DX текущую позицию курсора
SetCurs	Устанавливает курсор в указанную в DX позицию
My_2F	Собственный обработчик прерывания для 2F. Проверяет, была ли программа установлена резидентной в памяти
My_1C	Собственный обработчик прерывания для 1C. Выводит в 33-ую позицию курсора количество прерываний, которые были вызваны (счётчик обновляется каждые 10 раз). Прерывания генерируются системным таймером с частотой примерно 18.2 раза в секунду.
Un_check	Проверяет, нет указал ли пользователь флаг «/un» при вызове программы
Keep_interr	Запоминает старые обработчики прерывания, используя функцию 35h прерывания int 21h
Load_interr	Устанавливает новые обработчики прерывания, используя функцию 25h прерывания int 21h
Unload_interr	Восстанавливает сохранённые заранее обработчики прерываний и выгружает резидентную программу
Make_resident	Оставляет программу резидентной в памяти
Main	Основная функция

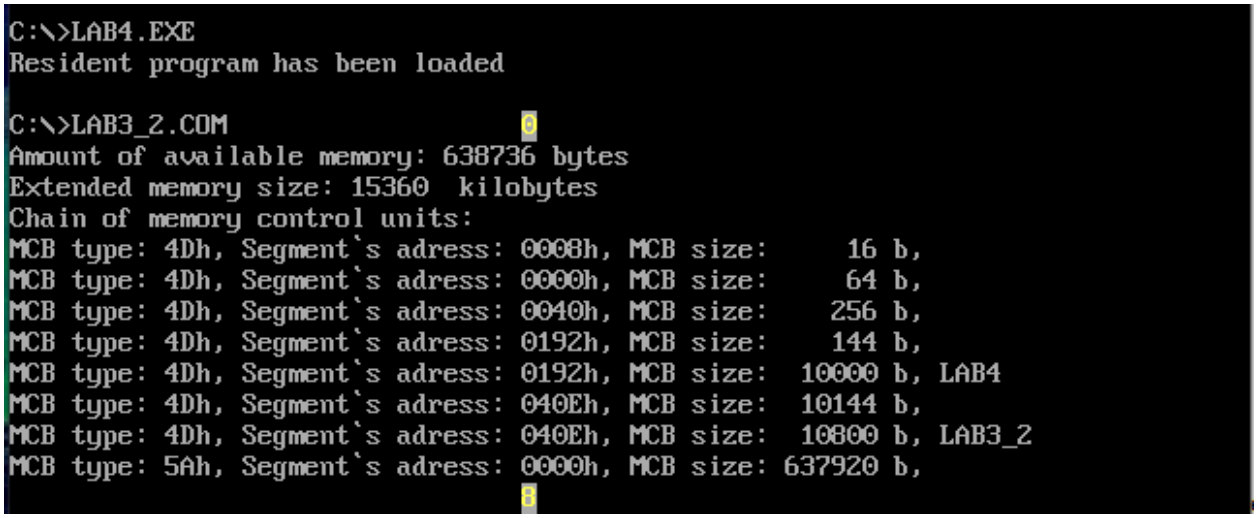
Описание структур данных.

Название	Описание
Count	запоминает количество вызванных прерываний
flag	флаг, равный 1, если программа не является резидентной, и 0, если наоборот
Message1	Сообщение о том, что программа только что была загружена в память резидентной
Message2	Хранит тип текущего блока памяти
Message3	Хранит сегментный адрес текущего блока памяти
keep_1c	Переменная для хранения сегмента и смещения старого прерывания 1c
keep_2f	Переменная для хранения сегмента и смещения старого прерывания 2f

keep_PSP	Переменная для хранения старого значения ES до того, как программа была оставлена резидентной в памяти

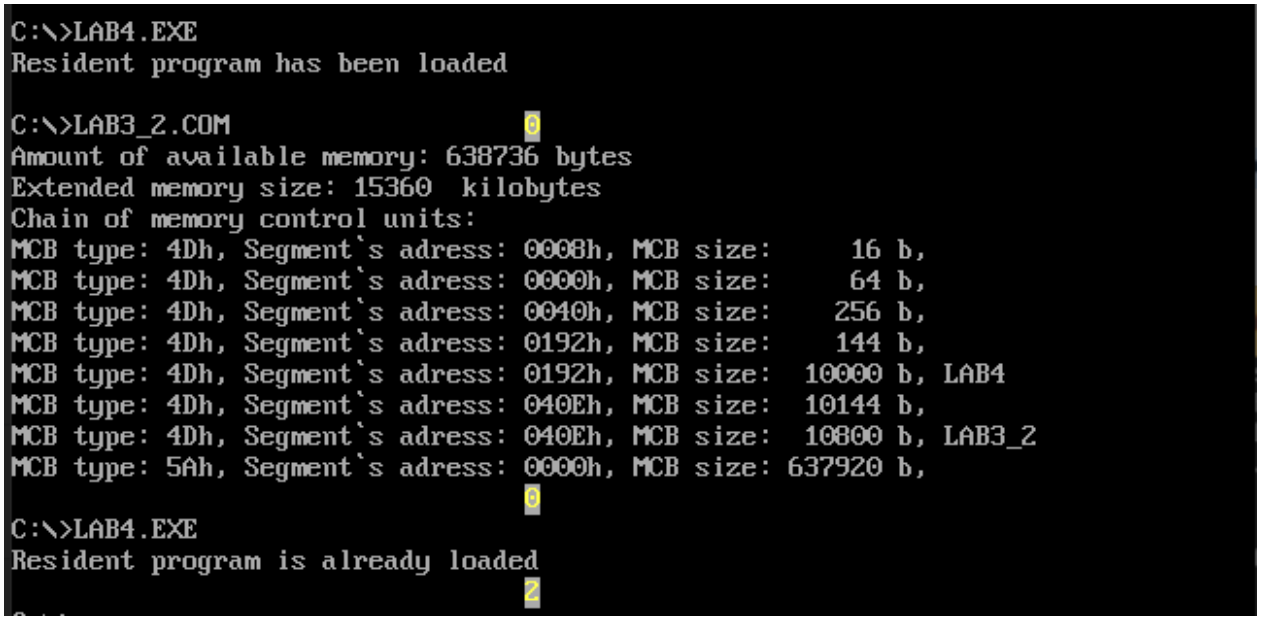
Выполнение работы.

Шаг 1.



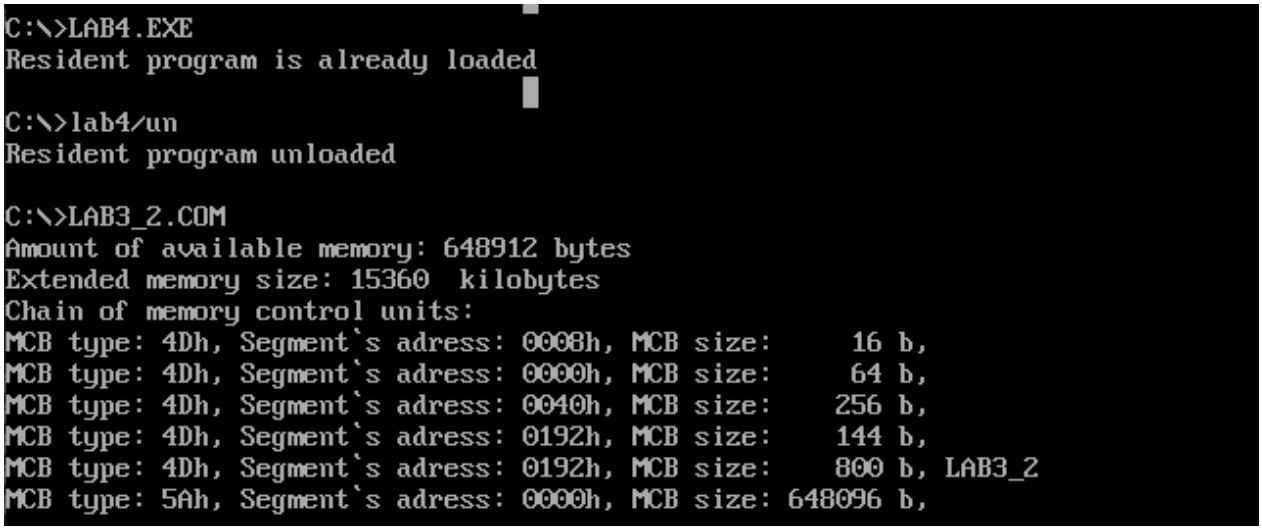
Убедились, что программа отображает результат работы прерывания и остаётся резидентной.

Шаг 2.



Убедились, что программа распознаёт то, что она уже была загружена резидентной в память.

Шаг 3.



Программа успешно выгружается по команде «/un».

Ответы на контрольные вопросы.

1. Как реализован механизм прерывания часов ?

Каждые 55 мс сначала сохраняется состояние регистров, затем определяется источник прерывания, определяющий в свою очередь адрес (смещение) вектора прерывания в таблице векторов прерываний (значения могут быть от 0000:0000 до 0000:03FF. 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 работе ?

В работе использовались пользовательские прерывания int 21h, int 10h, int 2Fh, а также аппаратное прерывание int 1Ch, возникающее каждые 55 мс по системному таймеру.

Выводы.

В ходе данной лабораторной работы было изучено создание резидентных программ, а также построен обработчик прерывания от часов.

ПРИЛОЖЕНИЕ А

LAB4.ASM

```
CODE   SEGMENT
                KEEP_1C             DD 0 ;ПЕРЕМЕННАЯ ДЛЯ ХРАНЕНИЯ СЕГМЕНТА И СМЕЩЕНИЯ СТАРОГО ПРЕРЫВАНИЯ 1C
                KEEP_2F             DD 0 ;ПЕРЕМЕННАЯ ДЛЯ ХРАНЕНИЯ СЕГМЕНТА И СМЕЩЕНИЯ СТАРОГО ПРЕРЫВАНИЯ 2F
                KEEP_PSP    DW ?
        ASSUME CS:CODE, DS:DATA, SS:ASTACK

; ФУНКЦИЯ ВЫВОДА В ТЕКУЩЕЕ ПОЛОЖЕНИЕ КУРСОРА СИМВОЛА ИЗ AL
OUTPUTAL            PROC
                PUSH AX
                PUSH BX
                PUSH CX

                MOV AH, 09H
                MOV BH, 0
                MOV CX, 1
                INT 10H

                POP CX
                POP BX
                POP AX
                RET
OUTPUTAL            ENDP

;ФУНКЦИЯ СЧИТЫВАНИЯ ПОЗИЦИИ КУРСОРА: DH - ТЕКУЩАЯ СТРОКА, DL - КОЛОНКА КУРСОРА
SAVECURS  PROC
                PUSH AX
                PUSH BX

                MOV AH, 03H
                MOV BH, 0
                INT 10H

                POP BX
                POP AX
                RET
SAVECURS  ENDP

;ФУНКЦИЯ, УСТАНАВЛИВАЮЩАЯ КУРСОР В ЗАРАНЕЕ ОПРЕДЕЛЁННУЮ В DX ПОЗИЦИЮ
SETCURS  PROC
                PUSH AX
                PUSH BX
```

```
                MOV AH, 02H
                MOV BH, 0
                INT 10H

                POP BX
                POP AX
                RET
        SETCURS  ENDP

        ;СОБСТВЕННЫЙ ОБРАБОТЧИК ПРЕРЫВАНИЯ ДЛЯ 2F
        MY_2F PROC
                CMP   AH, 080H ;СРАВНИВАЕМ ЗНАЧЕНИЕ В AH С УСТАНОВЛЕННЫМ
РАНЕЕ ПЕРЕД ПРЕРЫВАНИЕМ
                JNE   NOT_LOADED ;ЕСЛИ ЗНАЧЕНИЯ НЕ РАВНЫ - ПРОГРАММА НЕ
УСТАНОВЛЕНА РЕЗИДЕНТНОЙ В ПАМЯТИ
                MOV   AL, 0FFH ;УСТАНАВЛИВАЕМ В AL ЗНАЧЕНИЕ FF, ЧТО НА ВЫХОДЕ
ИЗ ПРЕРЫВАНИЯ ОЗНАЧАЕТ, ЧТО ПРОГРАММА УСТАНОВЛЕНА
            NOT_LOADED:
                IRET ;ИНАЧЕ ПРОСТО ВОЗВРАЩАЕМСЯ В ПРОГРАММУ
        MY_2F ENDP

        ;СОБСТВЕННЫЙ ОБРАБОТЧИК ПРЕРЫВАНИЯ ДЛЯ 1C
        MY_1C PROC
                PUSH AX
                PUSH BX
                PUSH CX
                PUSH DX
                PUSH ES

                INC COUNT
                CMP COUNT, 57 ;ДИАПАЗОН СИМВОЛОВ [0..9] = [48..57]
                JNE SHOW
                MOV COUNT, 48
            SHOW:

                ; СОХРАНЯЕМ ТЕКУЩЕЕ ПОЛОЖЕНИЕ КУРСОРА
                CALL SAVECURS
                MOV CX, DX

                ; ПЕРЕНОСИМ КУРСОР В УКАЗАНУЮ ПОЗИЦИЮ И ВЫВОДИМ ТУДА
СИМВОЛ
                MOV DH, 23
                MOV DL, 33
                CALL SETCURS
                PUSH AX
                MOV AL, COUNT
                CALL OUTPUTAL
```

```
                POP AX

                ; ВОЗВРАЩАЕМ КУРСОР
                MOV DX, CX
                CALL SETCURS

                MOV AL, 20H
                OUT 20H, AL

                POP   ES
                POP DX
                POP CX
                POP BX
                POP AX
                IRET
LAST_BYTE:
MY_1C           ENDP

; ФУНКЦИЯ ПРОВЕРКИ НЕ ВВЁЛ ЛИ ПОЛЬЗОВАТЕЛЬ КОМАНДУ /UN
UN_CHECK  PROC    FAR
                PUSH AX

                MOV   AX, KEEP_PSP
                MOV   ES, AX
                SUB   AX, AX
                CMP   BYTE PTR ES:[82H],'/'
                JNE   NOT_UN
                CMP   BYTE PTR ES:[83H],'U'
                JNE   NOT_UN
                CMP   BYTE PTR ES:[84H],'N'
                JNE   NOT_UN
                MOV   FLAG,0

        NOT_UN:
                POP   AX
                RET
UN_CHECK  ENDP

;ФУНКЦИЯ, СОХРАНЯЮЩАЯ СТАНДАРТНЫЕ ОБРАБОТЧИКИ ПРЕРЫВАНИЙ
KEEP_INTERR  PROC
                PUSH AX
                PUSH BX
                PUSH ES

                MOV AH, 35H ;ФУНКЦИЯ, ВЫДАЮЩАЯ ЗНАЧЕНИЕ СЕГМЕНТА В ES,
СМЕЩЕНИЕ В BX
                MOV AL, 1CH ;ДЛЯ ПРЕРЫВАНИЯ 1C
                INT 21H
```

```
                MOV WORD PTR KEEP_1C, BX
                MOV WORD PTR KEEP_1C+2, ES

                MOV AH, 35H ;ФУНКЦИЯ, ВЫДАЮЩАЯ ЗНАЧЕНИЕ СЕГМЕНТА В ES, СМЕЩЕНИЕ В BX
                MOV AL, 2FH ;ДЛЯ ПРЕРЫВАНИЯ 2F
                INT 21H
                MOV WORD PTR KEEP_2F, BX
                MOV WORD PTR KEEP_2F+2, ES

                POP ES
                POP BX
                POP AX
                RET
        KEEP_INTERR  ENDP

        ; ФУНКЦИЯ, ЗАГРУЖАЮЩАЯ СОБСТВЕННЫЕ ОБРАБОТЧИКИ ПРЕРЫВАНИЯ
        LOAD_INTERR          PROC
                PUSH DS
                PUSH DX
                PUSH AX

                CALL KEEP_INTERR ;СОХРАНЯЕМ СТАРЫЕ ОБРАБОТЧИКИ ПРЕРЫВАНИЙ

                PUSH DS
                MOV DX, OFFSET MY_1C
                MOV AX, SEG MY_1C
                MOV DS, AX
                MOV AH, 25H          ;ФУНКЦИЯ, МЕНЯЮЩАЯ ОБРАБОТЧИК ПРЕРЫВАНИЙ НА УКАЗАННЫЙ В DX И AX
                MOV AL, 1CH     ;ДЛЯ ПРЕРЫВАНИЯ 1C
            INT 21H

            MOV DX, OFFSET MY_2F
                MOV AX, SEG MY_2F
                MOV DS, AX
                MOV AH, 25H          ;ФУНКЦИЯ, МЕНЯЮЩАЯ ОБРАБОТЧИК ПРЕРЫВАНИЙ НА УКАЗАННЫЙ В DX И AX
                MOV AL, 2FH     ;ДЛЯ ПРЕРЫВАНИЯ 2F
            INT 21H
                POP DS

                POP AX
                POP DX
                POP DS
                RET
        LOAD_INTERR  ENDP
```

```
; ВЫГРУЖАЕМ ОБРАБОТЧИКИ ПРЕРЫВАНИЙ
UNLOAD_INTERR  PROC
                PUSH DS

                MOV AH, 35H
                MOV AL, 1CH
                INT 21H
                MOV DX, WORD PTR ES:KEEP_1C
                MOV AX, WORD PTR ES:KEEP_1C+2
                MOV WORD PTR KEEP_1C, DX
                MOV WORD PTR KEEP_1C+2, AX

                MOV AH, 35H
                MOV AL, 2FH
                INT 21H
                MOV DX, WORD PTR ES:KEEP_2F
                MOV AX, WORD PTR ES:KEEP_2F+2
                MOV WORD PTR KEEP_2F, DX
                MOV WORD PTR KEEP_2F+2, AX

                CLI
                MOV DX, WORD PTR KEEP_1C
                MOV AX,         WORD PTR KEEP_1C+2
                MOV DS, AX
                MOV AH, 25H  ;ВЫГРУЖАЕМ ОБРАБОТЧИК ДЛЯ 1C
                MOV AL, 1CH
                INT 21H

                MOV DX, WORD PTR KEEP_2F
                MOV AX,         WORD PTR KEEP_2F+2
                MOV DS, AX
                MOV AH, 25H  ;ВЫГРУЖАЕМ ОБРАБОТЧИК ДЛЯ 2F
                MOV AL, 2FH
                INT 21H
                STI

                POP DS

                MOV ES, ES:KEEP_PSP
                MOV AX, 4900H                ;ОСВОБОЖДАЕМ ПАМЯТЬ ПО АДРЕСУ
ES:KEEP_PSP
                INT 21H

                MOV FLAG, 1                  ;ЗАПОМИНАЕМ, ЧТО ПАМЯТЬ БЫЛА
ОСВОБОЖДЕНА
                MOV DX, OFFSET MESSAGE2
                CALL WRITE_MESSAGE  ;ВЫВОДИМ СООТВЕТСТВУЮЩЕЕ СООБЩЕНИЕ
```

```
                MOV ES, ES:[2CH]
                MOV AX, 4900H                ;ОСВОБОЖДАЕМ ПАМЯТЬ ПО АДРЕСУ
ES:[2CH]
                INT 21H

                MOV AX, 4C00H                ;ВЫХОД ИЗ ПРОГРАММЫ ЧЕРЕЗ ФУНКЦИЮ 4C
                INT 21H
        UNLOAD_INTERR  ENDP

        MAKE_RESIDENT  PROC
                MOV AX, ES
                MOV KEEP_PSP, AX
                MOV DX, OFFSET LAST_BYTE
                ADD DX, 200H

                MOV AH, 31H ;31H ЗАВЕРШАЕТ ПРОГРАММУ, ОСТАВЛЯЯ ЕЁ
РЕЗИДЕНТНОЙ В ПАМЯТИ
                MOV AL, 0
                INT 21H
        MAKE_RESIDENT  ENDP

        ; ФУНКЦИЯ ВЫВОДА СООБЩЕНИЯ НА ЭКРАН
        WRITE_MESSAGE       PROC
                PUSH AX
                MOV AH, 09H
                INT 21H
                POP AX
                RET
        WRITE_MESSAGE               ENDP

        ; ГЛАВНАЯ ФУНКЦИЯ
        MAIN   PROC
                PUSH DS
                XOR AX, AX
                PUSH AX
                MOV AX, DATA
                MOV DS, AX
                MOV KEEP_PSP, ES
                MOV COUNT, 48

                MOV    AX, 8000H ;НАМ НУЖНЫ НОМЕРА В АН ОТ 80H ДО 0FFH
                INT 2FH
                CMP    AL,0FFH        ; 2FH ВОЗВРАЩАЕТ 0FFH, ЕСЛИ ПРОГРАММА
УСТАНОВЛЕНА РЕЗИДЕНТНОЙ В ПАМЯТИ
                JNE LOADING

                CALL UN_CHECK
                CMP FLAG, 0
```

```
                JNE ALR_LOADED

                CALL UNLOAD_INTERR      ;ПОЛЬЗОВАТЕЛЬ ВВЁЛ /UN И ПРОГРАММА ЕЩЁ НЕ БЫЛА ВЫГРУЖЕНА
        LOADING:                        ;ПРОГРАММА НЕ ЯВЛЯЕТСЯ РЕЗИДЕНТНОЙ В ПАМЯТИ
                CALL LOAD_INTERR

                LEA DX, MESSAGE1
                CALL WRITE_MESSAGE

                CALL MAKE_RESIDENT
        ALR_LOADED:                     ;ПРОГРАММА УЖЕ БЫЛА РЕЗИДЕНТНОЙ
                LEA DX, MESSAGE3
                CALL WRITE_MESSAGE
                MOV AX, 4C00H
                INT 21H
MAIN    ENDP
CODE            ENDS

ASTACK          SEGMENT  STACK
    DW 256 DUP(?)
ASTACK  ENDS

DATA    SEGMENT
        COUNT                   DB ?
  FLAG                  DW 1
  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
```