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

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Организация ЭВМ и систем»
Тема: Написание собственного прерывания.

Студент гр. 0382 _____ Кривенцова Л.С.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2021

Цель работы.

Изучить понятие прерывания и написать своё собственное; реализовать программу, решающую с его помощью поставленную задачу.

Задание.

Вариант 9.

2b.

2 - 60h - прерывание пользователя - должно генерироваться в программе;

B - Выдача звукового сигнала с заданной высотой звука.

Основные теоретические положения.

Прерывание - это процесс вызова процедур для выполнения некоторой задачи, обычно связанной с обслуживанием некоторых устройств (обработка сигнала таймера, нажатия клавиши и т.д.).
Когда возникает прерывание, процессор прекращает выполнение текущей программы (если ее приоритет ниже) и запоминает в стеке вместе с регистром флагов адрес возврата(CS:IP) - места, с которого будет продолжена прерванная программа. Затем в CS:IP загружается адрес программы обработки прерывания и ей передается управление.
Адреса 256 программ обработки прерываний, так называемые векторы прерывания, имеют длину по 4 байта (в первых двух хранится значение IP , во вторых - CS) и хранятся в младших 1024 байтах памяти.
Программа обработки прерывания должна заканчиваться инструкцией IRET (возврат из прерывания), по которой из стека восстанавливается адрес возврата и регистр флагов.

Программа обработки прерывания - это отдельная процедура, имеющая структуру:

```
SUBR_INT  PROC FAR
      PUSH AX   ; сохранение изменяемых регистров

        <действия по обработке прерывания>

      POP  AX   ; восстановление регистров
      MOV  AL, 20H
       OUT  20H,AL
       IRET
```

```
SUBR_INT  ENDP
```

Две последние строки перед IRET необходимы для разрешения обработки прерываний с более низкими уровнями, чем только что обработанное.

Замечание: в лабораторной работе действиями по обработке прерывания может быть вывод на экран некоторого текста, вставка задержки в вывод сообщений, включение звукового сигнала и т.п.

Программа, использующая новые программы обработки прерываний, при своем завершении должна восстанавливать оригинальные векторы прерываний. Функция 35 прерывания 21H возвращает текущее значение вектора прерывания, помещая значение сегмента в ES, а смещение в BX. В этом случае программа должна содержать следующие инструкции:

```
; -- в сегменте данных
    KEEP_CS DW 0 ; для хранения сегмента
    KEEP_IP DW 0 ; и смещения прерывания

; -- в начале программы
    MOV  AH, 35H   ; функция получения вектора
    MOV  AL, 1CH   ; номер вектора
    INT  21H
    MOV  KEEP_IP, BX  ; запоминание смещения
    MOV  KEEP_CS, ES  ; и сегмента
```

Для задания адреса собственного прерывания с заданным номером в таблицу векторов прерываний используется функция 25H прерывания 21H, которая устанавливает вектор прерывания на указанный адрес нового обработчика.

```
    PUSH DS
    MOV  DX, OFFSET ROUT ; смещение для процедуры в DX
    MOV  AX, SEG ROUT    ; сегмент процедуры
    MOV  DS, AX          ; помещаем в DS
    MOV  AH, 25H         ; функция установки вектора
    MOV  AL, 60H         ; номер вектора
    INT  21H             ; меняем прерывание
```

```
POP  DS
```

В конце программы восстанавливается старый вектор прерывания

```
CLI
PUSH DS
MOV  DX, KEEP_IP
MOV  AX, KEEP_CS
MOV  DS, AX
MOV  AH, 25H
MOV  AL, 1CH
INT  21H        ; восстанавливаем вектор
POP  DS
STI
```

Логические операции с битами - OR, XOR, AND, NOT.Эти команды работают с приемником и источником, исключение команда 'NOT'. Каждый бит в приемнике сравнивается с тем же самым битом в источнике, и в зависимости от команды, 0 или 1 помещается в бит приемника:

команда	**AND**	**OR**	**XOR**	**NOT**										
Бит источника	0	0	1	1	0	0	1	1	0	0	1	1	0	1
Бит приемника	0	1	0	1	0	1	0	1	0	1	0	1	X	X
Бит результата	0	1	0	1	0	1	1	1	0	1	1	0	1	0

AND (логическое И)устанавливает бит результата в 1, если оба бита, бит источника и бит приемника установлены в 1.

OR (логическое ИЛИ)устанавливает бит результата в 1, если один из битов, бит источника или бит приемника установлен в 1.

XOR (НЕ ИЛИ)устанавливает бит результата в 1, если бит источника отличается от бита приемника.

NOT инвертирует бит источника.

Команда OUT в Ассемблере выполняет вывод данных в порт. Синтаксис: OUT ПРИЁМНИК, ИСТОЧНИК

Инструкция OUT выводит данные из регистра AL или AX (ИСТОЧНИК) в порт ввода-вывода. Номер порта должен быть указан в ПРИЁМНИКЕ.

ПРИЁМНИК может быть указан как непосредственное значение (константа), если номер порта укладывается в байт (не более 255). Если номер порта более 255, то он должен быть предварительно помещён в регистр DX, и тогда этот регистр должен быть указан в качестве ПРИЁМНИКА.

С помощью команды OUT выполняется вывод на все стандартные устройства ввода-вывода, такие как клавиатура, последовательные и параллельные порты, жёсткий диск и другие накопители.

Также можно использовать эту команду для прямой записи в видеопамять. Кроме того, через порты выполняется управление почти всем железом: таймером, динамиком и т.п.

Как вывести данные в порт

Вывод данных в порт в Ассемблере - это довольно распространённое действие. Через порты выполняется обмен данными с внешними и внутренними устройствами компьютера. В качестве примера рассмотрим программу, которая включает динамик компьютера (спикер) на некоторое время.

Пример программы:

```
;Эта программа воспроизводит звук
;через динамик компьютера
;61h - адрес порта динамика
;42h - адрес порта таймера

 .modeltiny
 .code
 ORG  100h

start:

 IN   AL,  61h      ;Получить состояние динамика
 PUSH AX            ;и сохранить его
 OR   AL,  00000011b ;Установить два младших бита
 OUT  61h, AL       ;Включить динамик
 MOV  AL,  10       ;Высота звука (частота)
 OUT  42h, AL       ;Включить таймер, который
                    ;будет выдавать импульсы на
                    ;динамик с заданной частотой
 MOV  CX,  Delay    ;Установить длительность звука

;Цикл, который определяет продолжительность звучания
;Поскольку одного цикла для задержки будет недостаточно
```

```
;(пробежит очень быстро и динамик просто не успеет включиться)
;используется вложенный цикл
Zvuk:
  PUSH CX
  MOV  CX,  Delay
  Cicle:
    LOOP Cicle
  POP  CX
  LOOP Zvuk

  POP  AX            ;Получить исходное состояние
  AND  AL,  11111100b ;Сбросить два младших бита
  OUT  61h, AL       ;Выключить динамик

RET

Delay  DW 3000       ;Длительность звука

END     start
```

Выполнение работы.

Написана процедура *MY_SUBR_INT* - программа обработки прерывания. Нынешнее состояние регистров сохраняется в стек, и восстанавливаются в конце процедуры.

В метке считывания (*input*) сначала реализуем вывод звука: для управления таймером подключенного к динамику подаём значение на порт *43h* (в двух командах, т.к. канал таймера двухбитовый). Значение высоты звука хранится в регистре *BX*. Устанавливается значение порта вывода (*61h*) для передачи сигнала динамику. После цикла, отвечающего за продолжительность звучания сигнала (*forloop*), в *61h* записывается первоначальное значение и динамик отключается.

Затем с клавиатуры считывается символ. С помощью *CMP* он сравнивается с ключами команд («*H*» и «*L*»), и в зависимости от результата совершает прыжок к соответствующей метке *(«higher_sound»* или «*low_sound*»). Если был передан иной ключ, совершается прыжок к метке («*exit*») и прерывание окончено. В метках значение *BX* проверяется на крайние значения, и если лимит не нарушен, изменяет значение регистра на 1000.

Главная процедура обращается к функции *21h - 35h*, таким образом мы получаем текущий вектор прерывания *60h*. Потом значения регистров сохраняются в переменные (*BX* - в *KEEP_IP, ES* - в *KEEP_CS*). В *BX* устанавливается первоначальное значение высоты звучания сигнала. С помощью *21h* по смещению *60h* записывается пользовательское прерывание - *MY_SUBR_INT.*

Вызывается прерывание, а после выхода из него восстанавливается прежнее прерывание и происходит выход из программы.

Исходный код см. в приложении А.

Файл листинга см. в приложении Б.

Вывод.

Изучены основы прерываний в Ассемблере, написано собственное, которое реализуется в программе и помогает выполнить поставленное задание.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл lb5.asm

```
ASSUME CS:CODE, DS:DATA, SS:STACK

STACK SEGMENT STACK
      DW 512 DUP(?)
STACK ENDS

DATA SEGMENT
      KEEP_CS DW 0
     KEEP_IP DW 0

DATA ENDS

CODE SEGMENT
      MY_SUBR_INT PROC FAR
         push AX
            push CX
      input:
            mov AL, 10110110b
            out 43h, AL
            mov AX, BX
            out 42h, AL
            mov AL, AH
            out 42h, AL
            in AL, 61h
            mov AH, AL
            or AL, 3
            out 61h, AL
            sub CX, CX
            forloop: loop forloop
            mov AL, AH
            out 61h, AL
         mov AL, 20h
         out 20h, AL
            mov AH, 0h
            int 16h
            cmp AL, 'H'
```

```
      je higher_sound
      cmp AL, 'L'
      je low_sound
      jmp exit
      low_sound:
            cmp BX, 10000
            jge input
            add BX, 1000
            jmp input
      higher_sound:
            cmp BX, 1000
            jle input
            sub BX, 1000
            jmp input
      exit:
            pop AX
            pop CX
            iret
MY_SUBR_INT  ENDP

MAIN PROC FAR
      mov AX, 3560h
      int 21h
      mov KEEP_IP, BX
      mov KEEP_CS, ES
      mov BX, 5000

      push DS
      mov dx, offset MY_SUBR_INT
      mov AX, seg MY_SUBR_INT
      mov DS, AX
      mov AX, 2560h
      int 21h
      pop DS
      int 60h
            cli
            push DS
            mov DX, KEEP_IP
            mov AX, KEEP_CS
```

```
                mov DS, AX
                mov AH, 25h
                mov AL, 60h
                int 21h
                pop  DS
                sti
                mov AH, 4ch
                int 21h
     MAIN ENDP
CODE ENDS
END MAIN
```

ПРИЛОЖЕНИЕ Б
ЛИСТИНГОВЫЙ ФАЙЛ ПРОГРАММ

Файл lb5.lst

```
Microsoft (R) Macro Assembler Version 5.10                  12/13/21
02:18:2
                                                             Page
1-1


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

0000                    STACK SEGMENT STACK
0000  0200[                      DW 512 DUP(?)
       ????
                ]

0400                    STACK ENDS

0000                    DATA SEGMENT
0000  0000                       KEEP_CS DW 0
0002  0000                     KEEP_IP DW 0

0004                    DATA ENDS

0000                    CODE SEGMENT
0000                         MY_SUBR_INT PROC FAR
0000  50                       push AX
0001  51                          push CX
0002                         input:
0002  B0 B6                              mov AL, 10110110b
0004  E6 43                              out 43h, AL
0006  8B C3                              mov AX, BX
0008  E6 42                              out 42h, AL
000A  8A C4                              mov AL, AH
000C  E6 42                              out 42h, AL
000E  E4 61                              in AL, 61h
0010  8A E0                              mov AH, AL
0012  0C 03                              or AL, 3
0014  E6 61                              out 61h, AL
```

```
0016  2B C9                         sub CX, CX
0018  E2 FE                         forloop: loop forloop
001A  8A C4                         mov AL, AH
001C  E6 61                         out 61h, AL
001E  B0 20                       mov AL, 20h
0020  E6 20                       out 20h, AL
0022  B4 00                         mov AH, 0h
0024  CD 16                         int 16h
0026  3C 48                         cmp AL, 'H'
0028  74 13                         je higher_sound
002A  3C 4C                         cmp AL, 'L'
002C  74 03                         je low_sound
002E  EB 19 90                      jmp exit
0031                              low_sound:
0031  81 FB 2710                            cmp BX, 10000
0035  7D CB                                 jge input
0037  81 C3 03E8                           add BX, 1000
003B  EB C5                                 jmp input
003D                              higher_sound:
003D  81 FB 03E8                            cmp BX, 1000
0041  7E BF                                 jle input
0043  81 EB 03E8                           sub BX, 1000
0047  EB B9                                 jmp input
0049                              exit:
```



```
0049  58                         pop AX
004A  59                         pop CX
004B  CF                         iret
004C                      MY_SUBR_INT  ENDP

004C                      MAIN PROC FAR
004C  B8 3560                    mov AX, 3560h
004F  CD 21                      int 21h
0051  89 1E 0002 R               mov KEEP_IP, BX
```

```
0055  8C 06 0000 R                    mov KEEP_CS, ES
0059  BB 1388                         mov BX, 5000

005C  1E                          push DS
005D  BA 0000 R                   mov dx, offset MY_SUBR_INT
0060  B8 ---- R                   mov AX, seg MY_SUBR_INT
0063  8E D8                            mov DS, AX
0065  B8 2560                          mov AX, 2560h
0068  CD 21                            int 21h
006A  1F                          pop DS
006B  CD 60                            int 60h
006D  FA                               cli
006E  1E                               push DS
006F  8B 16 0002 R                          mov DX, KEEP_IP
0073  A1 0000 R                        mov AX, KEEP_CS
0076  8E D8                                 mov DS, AX
0078  B4 25                                 mov AH, 25h
007A  B0 60                                 mov AL, 60h
007C  CD 21                                 int 21h
007E  1F                               pop  DS
007F  FB                               sti
0080  B4 4C                                 mov AH, 4ch
0082  CD 21                                 int 21h
0084                         MAIN ENDP
0084                    CODE ENDS
                        END MAIN
```

Microsoft (R) Macro Assembler Version 5.10 12/13/21 02:18:2

Symbols-1

Segments and Groups:

N a m e	Length	Align	Combine	Class
CODE	0084	PARA	NONE	
DATA	0004	PARA	NONE	
STACK	0400	PARA	STACK	

```
Symbols:

                N a m e         Type   Value      Attr

EXIT . . . . . .                L NEAR     0049  CODE

FORLOOP  . . . . . .            L NEAR     0018  CODE

HIGHER_SOUND . . . . . .        L NEAR     003D  CODE

INPUT  . . . . . .              L NEAR     0002  CODE

KEEP_CS  . . . . . .            L WORD     0000  DATA
KEEP_IP  . . . . . .            L WORD     0002  DATA

LOW_SOUND  . . . . . .          L NEAR     0031  CODE

MAIN . . . . . .                F PROC     004C  CODE  Length  =
0038
MY_SUBR_INT  . . . . . .        F PROC     0000  CODE  Length  =
004C

@CPU . . . . . .                TEXT  0101h
@FILENAME  . . . . . .          TEXT  lb5
@VERSION . . . . . .            TEXT  510


     86 Source  Lines
     86 Total   Lines
     17 Symbols

  48034 + 461273 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors
```