

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ЭВМ

Отчет по лабораторной работе №2  
«Программирование контроллера прерываний»

Проверил:  
к.т.н., доцент  
Одинец Д.Н.

Выполнил:  
студент гр. 150501  
Почебут А.С.

Минск 2023

## 1. Постановка задачи

Написать резидентную программу, выполняющую перенос всех векторов аппаратных прерываний ведущего и ведомого контроллера на пользовательские прерывания. При этом необходимо написать обработчики аппаратных прерываний, которые будут установлены на используемые пользовательские прерывания и будут выполнять следующие функции:

1. Выводить на экран в двоичной форме следующие регистры контроллеров прерывания (как ведущего, так и ведомого):

- регистр запросов на прерывания;
- регистр обслуживаемых прерываний;
- регистр масок.

При этом значения регистров должны выводиться всегда в одно и то же место экрана.

2. Осуществлять переход на стандартные обработчики аппаратных прерываний, для обеспечения нормальной работы компьютера.

## 2. Алгоритм

- Все векторы аппаратных прерываний ведущего и ведомого контроллера переносятся на пользовательские прерывания с помощью функций `getvect` и `setvect`.
- Производится инициализация контроллеров, заключающаяся в последовательности команд: `ICW1`, `ICW2`, `ICW3` и `ICW4`.
- С помощью функции `_dos_keep` осуществляется выход в DOS, при этом программа остаётся резидентной.
- В каждом обработчике выводятся в видеопамять в двоичной форме значения регистров запросов на прерывания, обслуживаемых прерываний, масок. Затем вызываются стандартные обработчики прерываний.

## 3. Листинг программы

Далее приведен листинг резидентной программы, выполняющей перенос всех векторов аппаратных прерываний ведущего и ведомого контроллера на пользовательские прерывания.

```
#include <dos.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

struct VIDEO
{
    unsigned char symbol;
    unsigned char attribute;
```

```

};

char color = 0x5f;

void print()
{
    char temp;
    int i, val;
    VIDEO far* screen = (VIDEO far*)MK_FP(0xB800, 0);

    val = inp(0x21); // get mask Master register
    for (i = 0; i < 8; i++)
    {
        temp = val % 2;
        val = val >> 1;
        screen->symbol = temp + '0';
        screen->attribute = color;
        screen++;
    }
    screen++;

    val = inp(0xA1); // get mask Slave register
    for (i = 0; i < 8; i++)
    {
        temp = val % 2;
        val = val >> 1;
        screen->symbol = temp + '0';
        screen->attribute = color;
        screen++;
    }
    screen += 63;
    outp(0x20, 0x0A);

    val = inp(0x20); // get Master's request register
    for (i = 0; i < 8; i++)
    {
        temp = val % 2;
        val = val >> 1;
        screen->symbol = temp + '0';
        screen->attribute = color;
        screen++;
    }
    screen++;

    outp(0xA0, 0x0A);
    val = inp(0xA0); // get Slave's request register
    for (i = 0; i < 8; i++)
    {
        temp = val % 2;
        val = val >> 1;
        screen->symbol = temp + '0';
        screen->attribute = color;
        screen++;
    }
    screen += 63;

    outp(0x20, 0x0B);
    val = inp(0x20); // get Master's service register

```

```

    for (i = 0; i < 8; i++)
    {
        temp = val % 2;
        val = val >> 1;
        screen->symbol = temp + '0';
        screen->attribute = color;
        screen++;
    }
    screen++;

    outp(0xA0, 0x0B);
    val = inp(0xA0);
    for (i = 0; i < 8; i++)
    {
        temp = val % 2;
        val = val >> 1;
        screen->symbol = temp + '0';
        screen->attribute = color;
        screen++;
    }
}

// IRQ 0-7

void interrupt(*oldint8) (...); // IRQ 0 - interrupt of timer (18,2 times per second)
void interrupt(*oldint9) (...); // IRQ 1 - interrupt of keypad (press and release key)
void interrupt(*oldint10) (...); // IRQ 2 - interrupt for cascade interruptions in AT machines
void interrupt(*oldint11) (...); // IRQ 3 - interrupt of async port COM 2
void interrupt(*oldint12) (...); // IRQ 4 - interrupt of async port COM 1
void interrupt(*oldint13) (...); // IRQ 5 - interrupt of hard disk controller (for XT)
void interrupt(*oldint14) (...); // IRQ 6 - interrupt of floppy disk controller (when finish operation with floppy disk)
void interrupt(*oldint15) (...); // IRQ 7 - interrupt of printer (when printer is ready to work)

// IRQ 8-15

void interrupt(*oldint70) (...); // IRQ 8 - interrupt of real time clock
void interrupt(*oldint71) (...); // IRQ 9 - interrupt of EGA controller
void interrupt(*oldint72) (...); // IRQ 10 - reserved interrupt
void interrupt(*oldint73) (...); // IRQ 11 - reserved interrupt
void interrupt(*oldint74) (...); // IRQ 12 - reserved interrupt
void interrupt(*oldint75) (...); // IRQ 13 - interrupt of mathematic soprocessor
void interrupt(*oldint76) (...); // IRQ 14 - interrupt of hard disk
void interrupt(*oldint77) (...); // IRQ 15 - reserved interrupt

void interrupt newint08(...) { print(); oldint8(); }
void interrupt newint09(...) { print(); oldint9(); }
void interrupt newint0A(...) { print(); oldint10(); }
void interrupt newint0B(...) { print(); oldint11(); }
void interrupt newint0C(...) { print(); oldint12(); }
void interrupt newint0D(...) { print(); oldint13(); }
void interrupt newint0E(...) { print(); oldint14(); }
void interrupt newint0F(...) { print(); oldint15(); }

void interrupt newintC0(...) { print(); oldint70(); }
void interrupt newintC1(...) { print(); oldint71(); }

```

```

void interrupt newintC2(...) { print(); oldint72(); }
void interrupt newintC3(...) { print(); oldint73(); }
void interrupt newintC4(...) { print(); oldint74(); }
void interrupt newintC5(...) { print(); oldint75(); }
void interrupt newintC6(...) { print(); oldint76(); }
void interrupt newintC7(...) { print(); oldint77(); }

void initialize()
{
    oldint8 = getvect(0x08);
    oldint9 = getvect(0x09);
    oldint10 = getvect(0x0A);
    oldint11 = getvect(0x0B);
    oldint12 = getvect(0x0C);
    oldint13 = getvect(0x0D);
    oldint14 = getvect(0x0E);
    oldint15 = getvect(0x0F);

    oldint70 = getvect(0x70);
    oldint71 = getvect(0x71);
    oldint72 = getvect(0x72);
    oldint73 = getvect(0x73);
    oldint74 = getvect(0x74);
    oldint75 = getvect(0x75);
    oldint76 = getvect(0x76);
    oldint77 = getvect(0x77);

    //set new handlers for IRQ 0-7
    setvect(0x80, newint08);
    setvect(0x81, newint09);
    setvect(0x82, newint0A);
    setvect(0x83, newint0B);
    setvect(0x84, newint0C);
    setvect(0x85, newint0D);
    setvect(0x86, newint0E);
    setvect(0x87, newint0F);

    //set new handlers for IRQ8-15
    setvect(0x08, newintC0);
    setvect(0x09, newintC1);
    setvect(0x0A, newintC2);
    setvect(0x0B, newintC3);
    setvect(0x0C, newintC4);
    setvect(0x0D, newintC5);
    setvect(0x0E, newintC6);
    setvect(0x0F, newintC7);

    _disable(); // CLI

    // interrupt initializtion for Master
    outp(0x20, 0x11); //ICW1 - initialize master
    outp(0x21, 0x80); //ICW2 - base vector for master
    outp(0x21, 0x04); //ICW3 - the port bit of Slave (in binary format)
    outp(0x21, 0x01); //ICW4 - default

    // interrupt initialization for Slave
    outp(0xA0, 0x11); //ICW1 - initialize Slave

```

```

    outp(0xA1, 0x08); //ICW2 - base vector for slave
    outp(0xA1, 0x02); //ICW3 - the port number of connected port on Master
    outp(0xA1, 0x01); //ICW4 - default

    _enable(); // STI
}

int main()
{
    unsigned far* fp;
    initialize();
    system("cls");

    printf("                - mask\n");
    printf("                - prepare\n");
    printf("                - service\n");
    printf("Master  Slave\n");

    FP_SEG(fp) = _psp;
    FP_OFF(fp) = 0x2c;
    _dos_freemem(*fp);

    _dos_keep(0, (_DS - _CS) + (_SP / 16) + 1);

    return 0;
}

```

#### 4. Тестирование программ

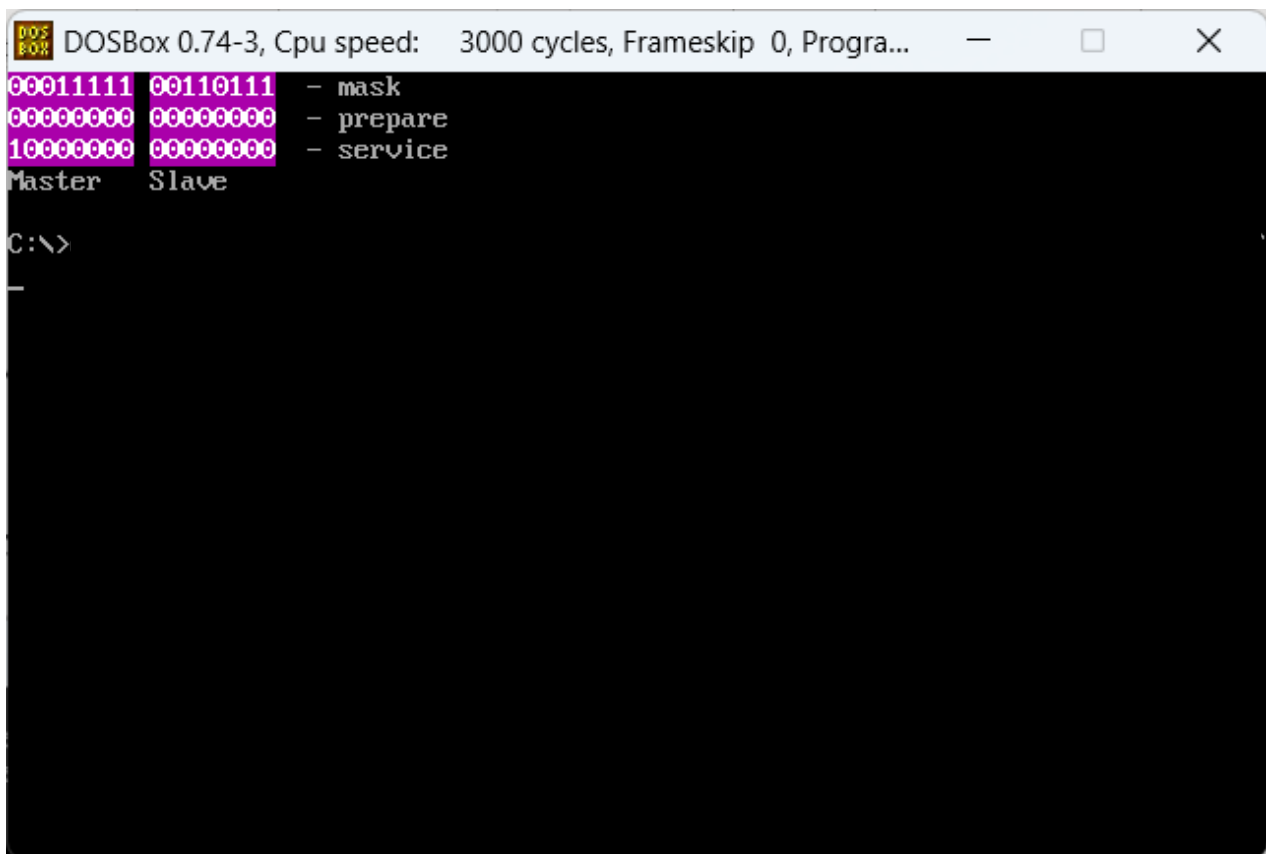


Рисунок 4.1 – Результат работы программы при запуске.



Рисунок 4.2 – Результат работы программы при нажатии клавиш.

## 5. Заключение

В ходе лабораторной удалось выполнить перенос всех векторов аппаратных прерываний ведущего и ведомого контроллера на пользовательские прерывания. Использование контроллера прерываний позволяет ускорить взаимодействие процессора с внешними устройствами. Недостатком программы является клонирование программы в памяти при повторном запуске.

Программа компилировалась в Borland C и запускалась в DOS, который эмулировался с помощью DosBox.