

Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра электронных вычислительных машин

Лабораторная работа №3

«Программирование контроллера прерываний»

Вариант 13

Выполнил:

Студент группы 050504

Матусевич С.К.

Проверил:

Преподаватель

Одинец Д.Н.

Минск, 2022

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>

#define COLOR_COUNT 7

struct VIDEO
```

```

{
    unsigned char symbol;
    unsigned char attribute;
};

unsigned char colors[COLOR_COUNT] =
{0x71,0x62,0x43,0x54,0x35,0x26,0x17};
char color = 0x89;

void changeColor()
{
    color = colors[rand() % COLOR_COUNT];
    return;
}

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

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

    val = inp(0xA1);
    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);

```

```

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);
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);
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++;
    }
}

void interrupt(*oldint8) (...);
void interrupt(*oldint9) (...);
void interrupt(*oldint10) (...);
void interrupt(*oldint11) (...);
void interrupt(*oldint12) (...);
void interrupt(*oldint13) (...);
void interrupt(*oldint14) (...);
void interrupt(*oldint15) (...);

void interrupt(*oldint70) (...);
void interrupt(*oldint71) (...);
void interrupt(*oldint72) (...);
void interrupt(*oldint73) (...);
void interrupt(*oldint74) (...);
void interrupt(*oldint75) (...);
void interrupt(*oldint76) (...);
void interrupt(*oldint77) (...);

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

void interrupt newintC0(...) { changeColor(); print();
oldint70(); }
void interrupt newintC1(...) { changeColor(); print();
oldint71(); }
void interrupt newintC2(...) { changeColor(); print();
oldint72(); }

```

```

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

    setvect(0x80, newint08);
    setvect(0x81, newint09);
    setvect(0x82, newint0A);
    setvect(0x83, newint0B);
    setvect(0x84, newint0C);
    setvect(0x85, newint0D);
    setvect(0x86, newint0E);
    setvect(0x87, newint0F);

    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();

    outp(0x20, 0x11);
    outp(0x21, 0x80);
    outp(0x21, 0x04);
    outp(0x21, 0x01);

    outp(0xA0, 0x11);
    outp(0xA1, 0x08);
    outp(0xA1, 0x02);
    outp(0xA1, 0x01);

    _enable();
}

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

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

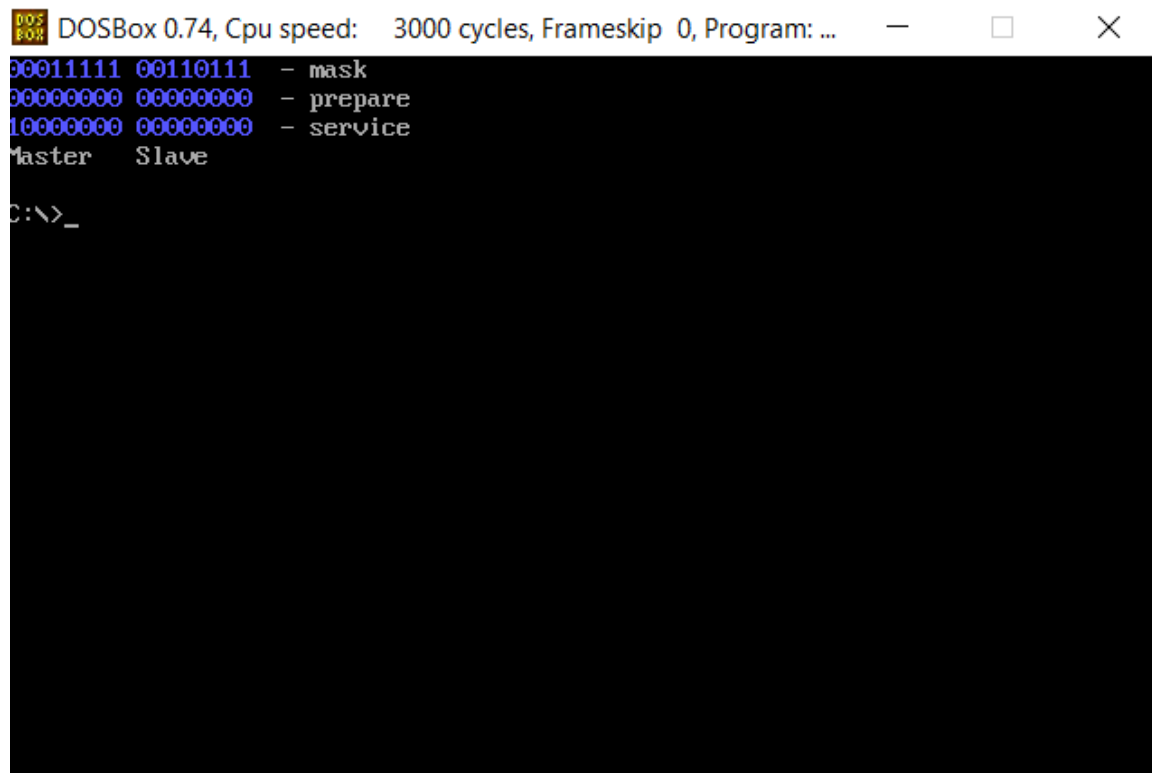
    FP_SEG(fp) = _psp;
    FP_OFF(fp) = 0x2c;
    _dos_freemem(*fp);

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

    return 0;
}

```

4. Тестирование программы

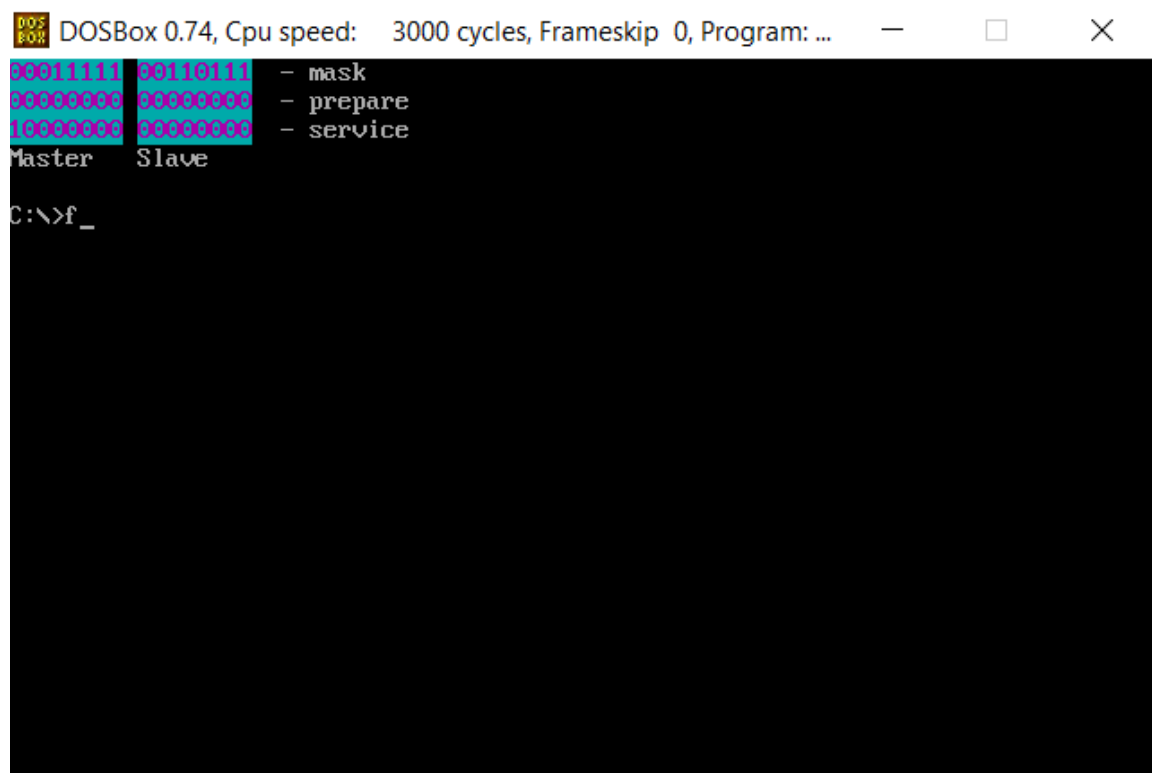


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: ...

```
00011111 00110111 - mask
00000000 00000000 - prepare
10000000 00000000 - service
Master   Slave

C:\>_
```

Рисунок 4.1 – Результат работы программы при запуске.



DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: ...

```
00011111 00110111 - mask
00000000 00000000 - prepare
10000000 00000000 - service
Master   Slave

C:\>f_
```

Рисунок 4.2 – Результат работы программы при нажатии клавиши.

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 Turbo C++ и запускалась в DOS, который эмулировался с помощью DosBox.