

Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра электронных вычислительных машин

Лабораторная работа №2

«Программирование контроллера прерываний»

Вариант 14

Выполнил:

Студент группы 150504
Желубовский С.В.

Проверил:

Преподаватель
Одинец Д.Н.

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>

#define COLORS 7 // amount of counts for colors

struct VIDEO
{
```

```

        unsigned char symbol;
        unsigned char attribute;
    };

    // array of counts for video buffer:
    unsigned char colors[COLORS] = { 0x71, 0x62, 0x43, 0x54, 0x35,
0x26, 0x17 };
    char color = 0x89; // default color (blue);

    void changeColor() // change color randomly
    {
        color = colors[rand() % COLORS];
        return;
    }

    void print()
    {
        char temp; // current bit
        int i, val;
        VIDEO far* screen = (VIDEO far*)MK_FP(0xB800, 0); // get
video card buffer

        //-----MASK-----
        val = inp(0x21); // get
mask MASTER register
        for (i = 0; i < 8; i++) // find
bits
        {
            temp = val % 2; // get
last bit
            val = val >> 1; // move
bits on right
            screen->symbol = temp + '0'; // save
bit as a char ('0' or '1')
            screen->attribute = color; // set
color of print symbol by video card
            screen++; // get
to next symbol
        }
        screen++; // space
between master and slave

        val = inp(0xA1); // get
mask SLAVE register
        for (i = 0; i < 8; i++)
// find bits
        {
            temp = val % 2;
            // get last bit
            val = val >> 1;
            // move bits on right
            screen->symbol = temp + '0'; //
save bit as a char ('0' or '1')

```

```

        screen->attribute = color;
// set color of print symbol by video card
        screen++;
// get to next symbol
    }
    screen += 63;
// next line
    //-----MASK-----
-----

    //-----REQUEST-----
-----

        outp(0x20, 0x0A);                                //
switch to MASTERS's request register
        val = inp(0x20);                                // get
MASTERS's request register
        for (i = 0; i < 8; i++)
// find bits
    {
        temp = val % 2;
        // get last bit
        val = val >> 1;
        // move bits on right
        screen->symbol = temp + '0';                        //
save bit as a char ('0' or '1')
        screen->attribute = color;
// set color of print symbol by video card
        screen++;
// get to next symbol
    }
    screen++;                                            // space

        outp(0xA0, 0x0A);                                //
switch to SLAVE's request register
        val = inp(0xA0);                                // get
SLAVE's request register
        for (i = 0; i < 8; i++)
// find bits
    {
        temp = val % 2;
        // get last bit
        val = val >> 1;
        // move bits on right
        screen->symbol = temp + '0';                        //
save bit as a char ('0' or '1')
        screen->attribute = color;
// set color of print symbol by video card
        screen++;
// get to next symbol
    }
    screen += 63;                                        // go to
next line
    //-----REQUEST-----
-----

```

```

//-----SERVICE-----
-----
    outp(0x20, 0x0B); //
switch to MASTER's service register
    val = inp(0x20); // get
MASTER's service register
    for (i = 0; i < 8; i++) // find
bits
    {
        temp = val % 2;
        // get last bit
        val = val >> 1;
        // move bits on right
        screen->symbol = temp + '0'; //
save bit as a char
        screen->attribute = color;
// set color of print
        screen++;
// get to next symbol
    }
    screen++; // space

    outp(0xA0, 0x0B); //
switch to SLAVE's service register
    val = inp(0xA0); // get-
ting SLAVES's service register
    for (i = 0; i < 8; i++) // find
bits
    {
        temp = val % 2;
// get last bit
        val = val >> 1;
        // move bits on right
        screen->symbol = temp + '0'; //
save bit as a char
        screen->attribute = color;
// set color of print
        screen++;
// get to next symbol
    }
//-----SERVICE-----
-----
}

//-----IRQ 0-7-----
-----
    void interrupt(*oldint8) (...);
// IRQ 0 - interrupt of timer (18,2 times per second) oldint8 -
переменная, являющаяся дальним указателем на функцию таймера
    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 0-7-----
-----

        //-----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
        //-----IRQ 8-15-----
-----

        //-----NEW INTERRUPTIONS-----
-----
        void interrupt newint08(...) { print(); oldint8(); }
// set function for service interrupt int8
        void interrupt newint09(...) { changeColor(); print();
oldint9(); } // set function for service interrupt int9
        void interrupt newint0A(...) { changeColor(); print();
oldint10(); } // set function for service interrupt int10
        void interrupt newint0B(...) { changeColor(); print();
oldint11(); } // set function for service interrupt int11
        void interrupt newint0C(...) { changeColor(); print();
oldint12(); } // set function for service interrupt int12
        void interrupt newint0D(...) { changeColor(); print();
oldint13(); } // set function for service interrupt int13
        void interrupt newint0E(...) { changeColor(); print();
oldint14(); } // set function for service interrupt int14
        void interrupt newint0F(...) { changeColor(); print();
oldint15(); } // set function for service interrupt int15

        void interrupt newintC8(...) { changeColor(); print();
oldint70(); } // set function for service interrupt int70

```

```

    void interrupt newintC9(...) { changeColor(); print();
oldint71(); } // set function for service interrupt int71
    void interrupt newintCA(...) { changeColor(); print();
oldint72(); } // set function for service interrupt int72
    void interrupt newintCB(...) { changeColor(); print();
oldint73(); } // set function for service interrupt int73
    void interrupt newintCC(...) { changeColor(); print();
oldint74(); } // set function for service interrupt int74
    void interrupt newintCD(...) { changeColor(); print();
oldint75(); } // set function for service interrupt int75
    void interrupt newintCE(...) { changeColor(); print();
oldint76(); } // set function for service interrupt int76
    void interrupt newintCF(...) { changeColor(); print();
oldint77(); } // set function for service interrupt int77
    //-----NEW INTERRUPTIONS-----
    -----

void initialize()
{
    //IRQ 0-7
    oldint8 = getvect(0x08); // IRQ 0 - system timer
    oldint9 = getvect(0x09); // IRQ 1 - keyboard controller
    oldint10 = getvect(0x0A); // IRQ 2 - cascaded sig-
nals from IRQs 8-15 (any devices configured to use IRQ2 will actually
be using IRQ9)
    oldint11 = getvect(0x0B); // IRQ 3 - serial port
controller for serial port 2 (shared with serial port 4, if present)
    oldint12 = getvect(0x0C); // IRQ 4 - serial port
controller for serial port 1 (shared with serial port 3, if present)
    oldint13 = getvect(0x0D); // IRQ 5 - parallel port
2 and 3 or sound card
    oldint14 = getvect(0x0E); // IRQ 6 - floppy disk
controller
    oldint15 = getvect(0x0F); // IRQ 7 - parallel port
1. It is used for printers or for any parallel port if a printer is
not present

    //IRQ 8-15
    oldint70 = getvect(0x70); // IRQ 8 - real-time
clock (RTC)
    oldint71 = getvect(0x71); // IRQ 9 - Advanced Con-
figuration and Power Interface (ACPI) system control interrupt
    oldint72 = getvect(0x72); // IRQ 10 - the interrupt
is left open for the use of peripherals (open interrupt/available,
SCSI or NIC)
    oldint73 = getvect(0x73); // IRQ 11 - the interrupt
is left open for the use of peripherals (open interrupt/available,
SCSI or NIC)
    oldint74 = getvect(0x74); // IRQ 12 - mouse on PS/2
connector
    oldint75 = getvect(0x75); // IRQ 13 - CPU co-proces-
sor or integrated floating point unit or inter-processor interrupt
(use depends on OS)
    oldint76 = getvect(0x76); // IRQ 14 - primary ATA
channel (ATA interface usually serves hard disk drives and CD drives)

```

```

        oldint77 = getvect(0x77);          // IRQ 15 - secondary ATA
channel

        //-----NEW HANDLERS-----
-----
        //set new handlers for IRQ 0-7
        setvect(0xC8, newint08);          // set new handler for IRQ 0
        setvect(0xC9, newint09);          // set new handler for IRQ 1
        setvect(0xCA, newint0A);          // set new handler for IRQ 2
        setvect(0xCB, newint0B);          // set new handler for IRQ 3
        setvect(0xCC, newint0C);          // set new handler for IRQ 4
        setvect(0xCD, newint0D);          // set new handler for IRQ 5
        setvect(0xCE, newint0E);          // set new handler for IRQ 6
        setvect(0xCF, newint0F);          // set new handler for IRQ 7

        //set new handlers for IRQ 8-15
        setvect(0x08, newintC8);          // set new handlers for IRQ 8
        setvect(0x09, newintC9);          // set new handlers for IRQ 9
        setvect(0x0A, newintCA);          // set new handlers for IRQ 10
        setvect(0x0B, newintCB);          // set new handlers for IRQ 11
        setvect(0x0C, newintCC);          // set new handlers for IRQ 12
        setvect(0x0D, newintCD);          // set new handlers for IRQ 13
        setvect(0x0E, newintCE);          // set new handlers for IRQ 14
        setvect(0x0F, newintCF);          // set new handlers for IRQ 15
        //-----NEW HANDLERS-----
-----

        _disable(); // CLI

        // interrupt initializtion for Master
        outp(0x20, 0x11);                  // ICW1 - initialize MAS-
TER
        outp(0x21, 0xC8);                  // ICW2 - base vector for
MASTER (C0H)
        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 (08H)
        outp(0xA1, 0x02);                  // ICW3 - the port number
of connected port on MASTER
        outp(0xA1, 0x01);                  // ICW4 - default

        _enable(); // STI
    }

    int main()
    {
        unsigned far* p;                  // declare
pointer that can link to other segment
        initialize();                      // execute in-
itIALIZATION

```



```

        system("cls");                // clear con-
sole

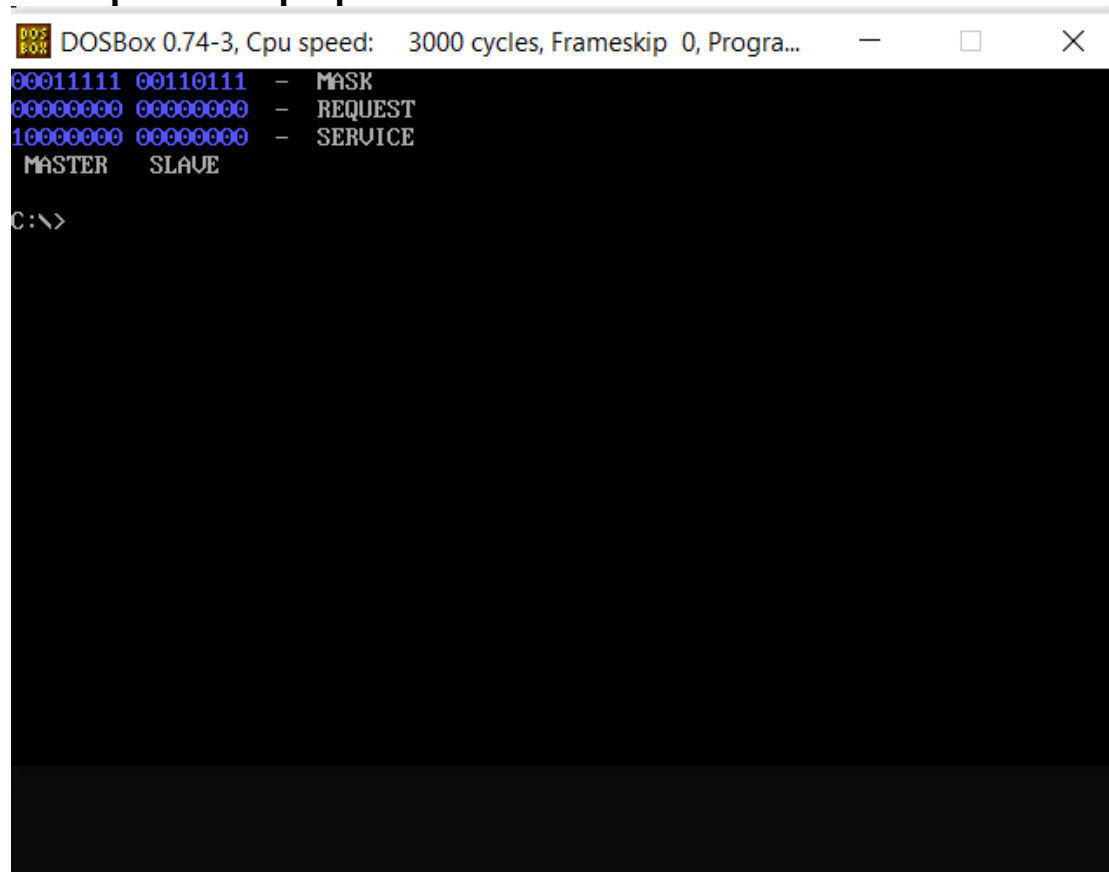
        printf("                - MASK\n");    // the mask of
Master and Slave
        printf("                - REQUEST\n");  // the request
of Master and Slave
        printf("                - SERVICE\n");  // the service
of Master and Slave
        printf(" MASTER    SLAVE\n");

        // resident
        FP_SEG(p) = _psp;                // get segment
        FP_OFF(p) = 0x2c;                // get data
segment offset with environment variables
        _dos_freemem(*p);                // free for
DOS

        _dos_keep(0, (_DS - _CS) + (_SP / 16) + 1);    // left pro-
gram resident, 1st param - ending code,
        // 2nd param - volume of memory, that must be reserved for
program after ending
        return 0;
}

```

4. Тестирование программы

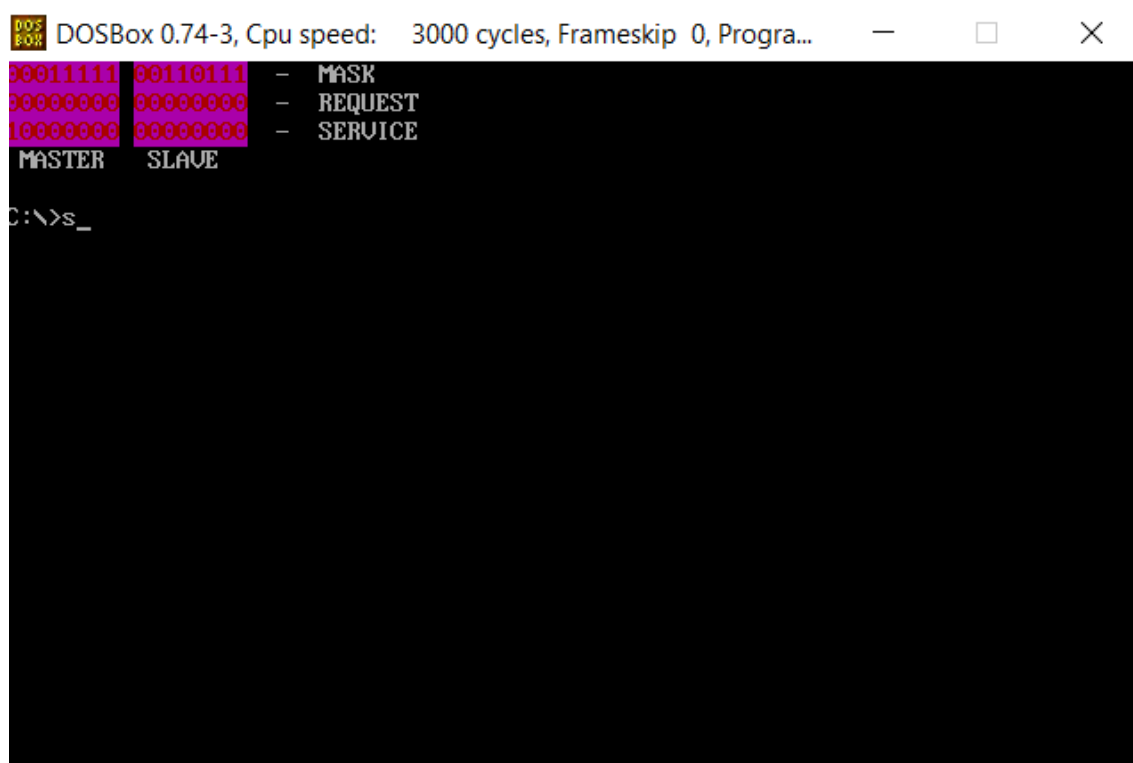


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...

```
00011111 00110111 - MASK
00000000 00000000 - REQUEST
10000000 00000000 - SERVICE
MASTER  SLAVE

C:\>
```

Рисунок 4.1 – Результат работы программы при запуске.



DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...

```
00011111 00110111 - MASK
00000000 00000000 - REQUEST
10000000 00000000 - SERVICE
MASTER  SLAVE

C:\>s_
```

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.