Белорусский Государственный Университет Информатики и Радиоэлектроники

Кафедра ЭВМ

Отчет по лабораторной работе № 4

Тема: «Программирование контроллера прерываний»

Выполнил:

студент гр.XXXXXX

XXXXX X.X.

Проверил:

Третьяков А.Г.

Минск 2016

#### 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.

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

#### Алгоритм решения задачи.

#### 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. В обработчике прерывания от клавиатуры также проверяется нажатие клавиши Esc – в случае нажатия восстанавливаются сохранённые обработчики событий.

#### Листинг программы.

#### #include <dos.h>

#### #include <stdio.h>

#### #include <stdlib.h>

#### void WriteStringToScreen(char\* str, int x, int y, unsigned char attribute);

#### void ByteToBinaryString(unsigned char temp, char \*str);

#### void ShowICsStatuses();

#### void interrupt new\_8(void);

#### void interrupt new\_9(void);

#### void interrupt new\_10(void);

#### void interrupt new\_11(void);

#### void interrupt new\_12(void);

#### void interrupt new\_13(void);

#### void interrupt new\_14(void);

#### void interrupt new\_15(void);

#### void interrupt new\_70(void);

#### void interrupt new\_71(void);

#### void interrupt new\_72(void);

#### void interrupt new\_73(void);

#### void interrupt new\_74(void);

#### void interrupt new\_75(void);

#### void interrupt new\_76(void);

#### void interrupt new\_77(void);

#### void interrupt (\*old\_8)(void);

#### void interrupt (\*old\_9)(void);

#### void interrupt (\*old\_10)(void);

#### void interrupt (\*old\_11)(void);

#### void interrupt (\*old\_12)(void);

#### void interrupt (\*old\_13)(void);

#### void interrupt (\*old\_14)(void);

#### void interrupt (\*old\_15)(void);

#### void interrupt (\*old\_70)(void);

#### void interrupt (\*old\_71)(void);

#### void interrupt (\*old\_72)(void);

#### void interrupt (\*old\_73)(void);

#### void interrupt (\*old\_74)(void);

#### void interrupt (\*old\_75)(void);

#### void interrupt (\*old\_76)(void);

#### void interrupt (\*old\_77)(void);

#### void RestoreInterrupts();

#### 

#### void main()

#### {

#### unsigned char value;

#### // изменяем таблицу векторов прерывания

#### old\_8 = getvect(0x8); // IRQ0 прерывание таймера, возникает 18,2 раза в секунду.

#### old\_9 = getvect(0x9); // IRQ1 прерывание от клавиатуры.

#### old\_10 = getvect(0xA); // IRQ2 используется для каскадирования аппаратных прерываний

#### old\_11 = getvect(0xB); // IRQ3 прерывание асинхронного порта COM2.

#### old\_12 = getvect(0xC); // IRQ4 прерывание асинхронного порта COM1.

#### old\_13 = getvect(0xD); // IRQ5 прерывание от контроллера жесткого диска для XT.

#### old\_14 = getvect(0xE); // IRQ6 прерывание генерируется контроллером флоппи диска после завершения операции

#### old\_15 = getvect(0xF); //IRQ7 прерывание принтера.

#### old\_70 = getvect(0x70); //IRQ8 прерывание от часов реального времени.

#### old\_71 = getvect(0x71); //IRQ9 прерывание от контроллера EGA.

#### old\_72 = getvect(0x72); //IRQ10 зарезервировано.

#### old\_73 = getvect(0x73); //IRQ11 зарезервировано.

#### old\_74 = getvect(0x74); //IRQ12 зарезервировано.

#### old\_75 = getvect(0x75); //IRQ13 прерывание от математического сопроцессора.

#### old\_76 = getvect(0x76); //IRQ14 прерывание от контроллера жесткого диска.

#### old\_77 = getvect(0x77); //IRQ15 зарезервировано.

#### 

#### setvect(0x78, new\_8);

#### setvect(0x79, new\_9);

#### setvect(0x7A, new\_10);

#### setvect(0x7B, new\_11);

#### setvect(0x7C, new\_12);

#### setvect(0x7D, new\_13);

#### setvect(0x7E, new\_14);

#### setvect(0x7F, new\_15);

#### setvect(0x80, new\_70);

#### setvect(0x81, new\_71);

#### setvect(0x82, new\_72);

#### setvect(0x83, new\_73);

#### setvect(0x84, new\_74);

#### setvect(0x85, new\_75);

#### setvect(0x86, new\_76);

#### setvect(0x87, new\_77);

#### 

#### // ИНИЦИАЛИЗАЦИЯ

#### // Ведущий контроллер

#### value = inp(0x21); // Запоминаем регистр масок

#### outp(0x20, 0x11); // ICW1 (инициализация ведущего контроллера) - ICW4 будет вызвана,

#### // размер вектора прерывания - 8 байт

#### outp(0x21, 0x78); // ICW2

#### outp(0x21, 0x04); // ICW3

#### outp(0x21, 0x01); // ICW4

#### outp(0x21, value); // Восстанавливаем регистр масок

#### 

#### // Ведомый контроллер

#### value = inp(0xA1); // Запоминаем регистр масок

#### outp(0xA0, 0x11); // ICW1

#### outp(0xA1, 0x80); // ICW2

#### outp(0xA1, 0x02); // ICW3

#### outp(0xA1, 0x01); // ICW4

#### outp(0xa1, value); // Восстанавливаем регистр масок

#### clrscr();

#### \_dos\_keep(0,(\_DS-\_CS)+(\_SP/16)+1); // Оставить программу резидентной

#### }

#### 

#### void ByteToBinaryString(unsigned char temp, char \*str)

#### {

#### int i;

#### str[8] = 0;

#### i=7;

#### 

#### while(temp)

#### {

#### str[i]='0'+temp%2;

#### temp=temp/2;

#### i--;

#### }

#### 

#### for(;i>-1;i--)

#### str[i]='0';

#### }

#### 

#### void WriteStringToScreen(char\* str, int x, int y, unsigned char attribute)

#### {

#### int i = 0;

#### char far\* start = (char far\*)0xb8000000;

#### start += x+160\*y;

#### 

#### while(str[i] != 0)

#### {

#### \*start = str[i];

#### start++;

#### \*start = attribute;

#### start++;

#### i++;

#### }

#### }

#### 

#### void ShowICsStatuses()

#### {

#### char str[9];

#### 

#### unsigned char isr\_master, isr\_slave; // Interrupt Service Register - Регистр обслуживаемых прерываний

#### unsigned char irr\_master, irr\_slave; // Interrupt Request Register - Регистр запросов на прерывания

#### unsigned char imr\_master, imr\_slave; // Interrupt Mask Register - Регистр масок

#### 

#### imr\_master = inp(0x21);

#### imr\_slave = inp(0xA1);

#### outp(0x20, 0x0A);

#### irr\_master = inp(0x20);

#### outp(0x20, 0x0B);

#### isr\_master = inp(0x20);

#### outp(0xA0,0x0A);

#### irr\_slave = inp(0xA0);

#### outp(0xA0,0x0B);

#### isr\_slave = inp(0xA0);

#### 

#### WriteStringToScreen("Master IC ISR: ",0, 0, 0x4E);

#### ByteToBinaryString(isr\_master, str);

#### WriteStringToScreen(str, 30, 0, 0x4E);

#### 

#### WriteStringToScreen("|IRR: ",46, 0, 0x4E);

#### ByteToBinaryString(irr\_master, str);

#### WriteStringToScreen(str, 58, 0, 0x4E);

#### WriteStringToScreen("|MASK: ", 74, 0, 0x4E);

#### ByteToBinaryString(imr\_master, str);

#### WriteStringToScreen(str, 88, 0, 0x4E);

#### 

#### WriteStringToScreen("Slave IC ISR: ", 0, 1, 0x4E);

#### ByteToBinaryString(isr\_slave, str);

#### WriteStringToScreen(str, 30, 1, 0x4E);

#### 

#### WriteStringToScreen("|IRR: ", 46, 1, 0x4E);

#### ByteToBinaryString(irr\_slave, str);

#### WriteStringToScreen(str, 58, 1, 0x4E);

#### WriteStringToScreen("|MASK: ",74, 1, 0x4E);

#### ByteToBinaryString(imr\_slave, str);

#### WriteStringToScreen(str, 88, 1, 0x4E);

#### }

#### 

#### void interrupt new\_8(void)

#### {

#### ShowICsStatuses();

#### (\*old\_8)();

#### }

#### void interrupt new\_9(void)

#### {

#### unsigned char value;

#### 

#### ShowICsStatuses();

#### (\*old\_9)();

#### 

#### value = inp(0x60);

#### 

#### if (value == 0x01)

#### {

#### RestoreInterrupts();

#### clrscr();

#### }

#### }

#### void interrupt new\_10(void)

#### {

#### ShowICsStatuses();

#### (\*old\_10)();

#### }

#### void interrupt new\_11(void)

#### {

#### ShowICsStatuses();

#### (\*old\_11)();

#### }

#### void interrupt new\_12(void)

#### {

#### ShowICsStatuses();

#### (\*old\_12)();

#### }

#### void interrupt new\_13(void)

#### {

#### ShowICsStatuses();

#### (\*old\_13)();

#### }

#### void interrupt new\_14(void)

#### {

#### ShowICsStatuses();

#### (\*old\_14)();

#### }

#### void interrupt new\_15(void)

#### {

#### ShowICsStatuses();

#### (\*old\_15)();

#### }

#### void interrupt new\_70(void)

#### {

#### ShowICsStatuses();

#### (\*old\_70)();

#### }

#### void interrupt new\_71(void)

#### {

#### ShowICsStatuses();

#### (\*old\_71)();

#### }

#### void interrupt new\_72(void)

#### {

#### ShowICsStatuses();

#### (\*old\_72)();

#### }

#### void interrupt new\_73(void)

#### {

#### ShowICsStatuses();

#### (\*old\_73)();

#### }

#### void interrupt new\_74(void)

#### {

#### ShowICsStatuses();

#### (\*old\_74)();

#### }

#### void interrupt new\_75(void)

#### {

#### ShowICsStatuses();

#### (\*old\_75)();

#### }

#### void interrupt new\_76(void)

#### {

#### ShowICsStatuses();

#### (\*old\_76)();

#### }

#### void interrupt new\_77(void)

#### {

#### ShowICsStatuses();

#### (\*old\_77)();

#### }

#### void RestoreInterrupts()

#### {

#### disable();

#### 

#### setvect(0x78, old\_8);

#### setvect(0x79, old\_9);

#### setvect(0x7A, old\_10);

#### setvect(0x7B, old\_11);

#### setvect(0x7C, old\_12);

#### setvect(0x7D, old\_13);

#### setvect(0x7E, old\_14);

#### setvect(0x7F, old\_15);

#### setvect(0x80, old\_70);

#### setvect(0x81, old\_71);

#### setvect(0x82, old\_72);

#### setvect(0x83, old\_73);

#### setvect(0x84, old\_74);

#### setvect(0x85, old\_75);

#### setvect(0x86, old\_76);

#### setvect(0x87, old\_77);

#### enable();

#### }

#### Результаты работы программы.

#### Программа является резидентной. После запуска в верхней части экрана выводятся в двоичной форме регистры запросов на прерывания, обслуживаемых прерываний, масок для ведущего и ведомого контроллеров. По нажатию клавиши ESC восстанавливаются стандартные обработчики прерываний.