Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра электронных вычислительных машин

Отчёт

по лабораторной работе № 2

на тему

«Программирование контроллера прерываний»

|  |  |
| --- | --- |
| Студент группы № 230501 | Кочеров Р. С. |
| Преподаватель | Марзалюк А. В. |

Минск 2024

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 решения задачи.**

1) Все векторы аппаратных прерываний ведущего и ведомого контроллера переносятся на пользовательские прерывания с помощью функций getvect и setvect.

2) Производится инициализация контроллеров, заключающаяся в последовательности команд: ICW1, ICW2, ICW3 и ICW4.

3) С помощью функции \_dos\_keep осуществляется выход в DOS, при этом программа остаётся резидентной.

4) В каждом обработчике выводятся в видеопамять в двоичной форме значения регистров запросов на прерывания, обслуживаемых прерываний, масок. Затем вызываются стандартные обработчики прерываний.

1. **Листинг программы.**

#include <dos.h>

#include <time.h>

#include <stdio.h>

#include <stdlib.h>

void printRegister(int reg, char far\* pos) {

for (int i = 0; i < 8; i++) {

\*pos = reg % 2 + '0';

pos += 2;

reg = reg >> 1;

}

}

void print()

{

char temp;

int i, val;

char far\* screen = (char far\*)MK\_FP(0xB800, 0);

screen += 20;

val = inp(0x21); // get mask Master registor

printRegister(val, screen);

screen += 18;

val = inp(0xA1); // регистр масок ведомого контроллера

printRegister(val, screen);

screen += 142;

outp(0x20, 0x0A);

val = inp(0x20);

printRegister(val, screen); // get Masters's request register

screen += 18;

outp(0xA0, 0x0A);

val = inp(0xA0); // get Slave's request register

printRegister(val, screen);

screen += 142;

outp(0x20, 0x0B);

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

printRegister(val, screen);

screen += 18;

outp(0xA0, 0x0B);

val = inp(0xA0); // get Slave's service register

printRegister(val, screen);

}

void prepare() {

\_disable(); //выключает прерыванрие

print();

}

void interrupt(\*oldint1[8])(...); // IRQ 0-7

void interrupt(\*oldint2[8])(...); // IRQ 8-15

void interrupt newint08(...) { \_disable(); print(); oldint1[0](); \_enable(); }

void interrupt newint09(...) { prepare(); oldint1[1](); \_enable(); }

void interrupt newint0A(...) { prepare(); oldint1[2](); \_enable(); }

void interrupt newint0B(...) { prepare(); oldint1[3](); \_enable(); }

void interrupt newint0C(...) { prepare(); oldint1[4](); \_enable(); }

void interrupt newint0D(...) { prepare(); oldint1[5](); \_enable(); }

void interrupt newint0E(...) { prepare(); oldint1[6](); \_enable(); }

void interrupt newint0F(...) { prepare(); oldint1[7](); \_enable(); }

void interrupt newint78(...) { prepare(); oldint2[0](); \_enable(); }

void interrupt newint79(...) { prepare(); oldint2[1](); \_enable(); }

void interrupt newint7A(...) { prepare(); oldint2[2](); \_enable(); }

void interrupt newint7B(...) { prepare(); oldint2[3](); \_enable(); }

void interrupt newint7C(...) { prepare(); oldint2[4](); \_enable(); }

void interrupt newint7D(...) { prepare(); oldint2[5](); \_enable(); }

void interrupt newint7E(...) { prepare(); oldint2[6](); \_enable(); }

void interrupt newint7F(...) { prepare(); oldint2[7](); \_enable(); }

void initialize()

{

int i;

for (i = 0x70; i <= 0x77; i++) {

oldint2[i - 0x70] = getvect(i);

}

for (i = 0x08; i <= 0x0F; i++) {

oldint1[i - 0x08] = getvect(i);

}

//set new handlers for IRQ 0-7

setvect(0x70, newint08);

setvect(0x71, newint09);

setvect(0x72, newint0A);

setvect(0x73, newint0B);

setvect(0x74, newint0C);

setvect(0x75 newint0D);

setvect(0x76, newint0E);

setvect(0x77, newint0F);

//set new handlers for IRQ8-15

setvect(0x08, newint78);

setvect(0x09, newint79);

setvect(0x0A, newint7A);

setvect(0x0B, newint7B);

setvect(0x0C, newint7C);

setvect(0x0D, newint7D);

setvect(0x0E, newint7E);

setvect(0x0F, newint7F);

\_disable(); // CLI

//interrupt initializtion for Master

outp(0x20, 0x11); //ICW1 - initialize master (00010001, 0001 - base, last 1 - next will be)

outp(0x21, 0x70); //ICW2 - base vector for master (10101000, )

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("obsl:\n");

printf("requ:\n");

FP\_SEG(fp) = \_psp;

FP\_OFF(fp) = 0x2c;

\_dos\_freemem(\*fp);

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

return 0;

}

1. **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.

