Министерство образования Республики Беларусь

Учреждение образования Белорусский государственный университет информатики и радиоэлектроники

Кафедра ЭВМ

Отчёт по лабораторной работе №2

“ Программирование контроллера прерываний ”

Проверил: Выполнил:

Минск 2023

#### **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.**

#### Для того чтобы резидентную программу выполняющую перенос всех векторов аппаратных прерываний ведущего и ведомого контроллера на пользовательские прерывания, необходимо выполнить следующие действия:

1. Изучить документацию на ведущий и ведомый контроллеры прерываний, чтобы понимать, как они работают и какие векторы прерываний используются.

2. Написать программу, которая будет перенаправлять все векторы аппаратных прерываний на соответствующие пользовательские прерывания.

3. Протестировать программу на ведущем и ведомом контроллерах, чтобы убедиться, что она работает корректно и не вызывает ошибок или сбоев в системе.

4. Внедрить программу в систему и настроить ее для автоматического запуска при старте компьютера или другого устройства.

5. Провести дополнительные тесты и проверки, чтобы убедиться, что перенос векторов прерываний происходит без ошибок и не влияет на работу других компонентов системы.

1. **Листинг программы**

#include <dos.h>

int attribute = 0x00; //начальный цвет

struct VIDEO

{

unsigned char symb;

unsigned char attr;

};

void get\_reg(); //получить данные из регистров

void print(int offset, int); //заполнить указатель видеоизображения

//IRQ0-7

void interrupt(\*int8) (...);// IRQ 0 - таймер

void interrupt(\*int9) (...);// IRQ 1 - клавиатура

void interrupt(\*intA) (...);// IRQ 2 - каскадное переключение второго контроллера

void interrupt(\*intB) (...);// IRQ 3 - последовательный порт COM 2

void interrupt(\*intC) (...);// IRQ 4 - последовательный порт COM 1

void interrupt(\*intD) (...);// IRQ 5 - параллельный порт LPT 2

void interrupt(\*intE) (...);// IRQ 6 - контроллер флоппи-дисковод

void interrupt(\*intF) (...);// IRQ 7 - параллельный порт LPT 1

//IRQ8-15

void interrupt(\*int0) (...);// IRQ 8 - часы реального времени

void interrupt(\*int1) (...);// IRQ 9 - свободный

void interrupt(\*int2) (...);// IRQ 10 - контроллер видеоадаптера

void interrupt(\*int3) (...);// IRQ 11 - свободный

void interrupt(\*int4) (...);// IRQ 12 - мышь PS/2

void interrupt(\*int5) (...);// IRQ 13 - математический сопроцессор

void interrupt(\*int6) (...);// IRQ 14 - первый контроллер жёсткого диска

void interrupt(\*int7) (...);// IRQ 15 - второй контроллер жёсткого диска

// new interrupt handlers

void interrupt new8(...) { get\_reg(); int8(); } //int8

void interrupt new9(...) { attribute++; get\_reg(); int9(); } //keyboard int9

void interrupt newA(...) { get\_reg(); intA(); } //intA

void interrupt newB(...) { get\_reg(); intB(); } //intB

void interrupt newC(...) { get\_reg(); intC(); } //intC

void interrupt newD(...) { get\_reg(); intD(); } //intD

void interrupt newE(...) { get\_reg(); intE(); } //intE

void interrupt newF(...) { get\_reg(); intF(); } //intF

void interrupt new0(...) { get\_reg(); int0(); } //int0

void interrupt new1(...) { get\_reg(); int1(); } //int1

void interrupt new2(...) { get\_reg(); int2(); } //int2

void interrupt new3(...) { get\_reg(); int3(); } //int3

void interrupt new4(...) { get\_reg(); int4(); } //int4

void interrupt new5(...) { get\_reg(); int5(); } //int5

void interrupt new6(...) { get\_reg(); int6(); } //int6

void interrupt new7(...) { get\_reg(); int7(); } //int7

//заполнить указатель видеоизображения

void print(int offset, int val)

{

char temp; //текущий бит

int i;

VIDEO far\* screen = (VIDEO far\*)MK\_FP(0xB800, 0); //получить видеобуфер

screen += offset;

for (i = 7; i >= 0; i--) //проходим по регистру и ищем бит

{

temp = val % 2; //получить последний бит

val /= 2; //fill the screen

screen->symb = temp + '0'; //сщхранить бит как символ('0' или '1')

screen->attr = attribute; //устанавливаем цвет

screen++; //к следующему символу

}

}

//получить данные из регистров

void get\_reg()

{

int reg;

//ведущий контроллер

reg = inp(0x21); //получить регистр масок ведущего контроллера

print(0, reg);

outp(0x20, 0x0B); //переключение на регистр обслуживания ведущего контроллера

reg = inp(0x20); //получить регистр обслуживания ведущего контроллера

print(9, reg);

outp(0x20, 0x0A); //переключение на регистр запросов ведущего контроллера

reg = inp(0x20); //получить регистр запросов ведущего контроллера

print(18, reg);

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

print(80, reg);

outp(0xA0, 0x0B); //переключение на регистр обслуживания ведомого контроллера

reg = inp(0xA0); //получить регистр обслуживания ведомого контроллера

print(80 + 9, reg);

outp(0xA0, 0x0A); //переключение на регистр запросов ведомого контроллера

reg = inp(0xA0); //получить регистр запросов ведомого контроллера

print(80 + 18, reg);

}

void init() {

//IRQ0-7

int8 = getvect(0x60); // IRQ 0

int9 = getvect(0x61); // IRQ 1

intA = getvect(0x62); // IRQ 2

intB = getvect(0x63); // IRQ 3

intC = getvect(0x64); // IRQ 4

intD = getvect(0x65); // IRQ 5

intE = getvect(0x66); // IRQ 6

intF = getvect(0x67); // IRQ 7

//IRQ8-15

int0 = getvect(0x08); // IRQ 8

int1 = getvect(0x09); // IRQ 9

int2 = getvect(0x0A); // IRQ 10

int3 = getvect(0x0B); // IRQ 11

int4 = getvect(0x0C); // IRQ 12

int5 = getvect(0x0D); // IRQ 13

int6 = getvect(0x0E); // IRQ 14

int7 = getvect(0x0F); // IRQ 15

//переопределение IRQ 0-7 (08-0F)

setvect(0x60, new8);

setvect(0x61, new9);

setvect(0x62, newA);

setvect(0x63, newB);

setvect(0x64, newC);

setvect(0x65, newD);

setvect(0x66, newE);

setvect(0x67, newF);

//переопределение IRQ 8-15 (B0-B7)

setvect(0x08, new0);

setvect(0x09, new1);

setvect(0x0A, new2);

setvect(0x0B, new3);

setvect(0x0C, new4);

setvect(0x0D, new5);

setvect(0x0E, new6);

setvect(0x0F, new7);

\_disable(); // запрет прерывания (Clear Interrupt Flag)

/\*Макрос outp выводит байт, определённый (0x11) в порт, который определяется параметром (0x20)\*/

//Инициализация прерываний для ведущего контроллера

outp(0x20, 0x11); //ICW1 – инициализация ведущего контроллера

outp(0x21, 0x08); //ICW2 – базовый вектор для ведущего

outp(0x21, 0x04); //ICW3 – порт бит ведомого контроллера (в двоичном формате)

outp(0x21, 0x01); //ICW4 – по умолчанию

//Инициализация прерываний для ведомого контроллера

outp(0xA0, 0x11); //ICW1 – инициализация ведомого контроллера

outp(0xA1, 0xB0); //ICW2 – базовый вектор для ведомого

outp(0xA1, 0x02); //ICW3 – количество подключенных портов к ведущему

outp(0xA1, 0x01); //ICW4 – по умолчанию

\_enable(); // возобновление возможности прерываний

}

int main()

{

unsigned far\* fp; //объявляем указатель

init();

FP\_SEG(fp) = \_psp; //получаем сегмент

FP\_OFF(fp) = 0x2c; //получаем смещение сегмента данных с переменными среды

\_dos\_freemem(\*fp); //освобождение памяти под них

\_dos\_keep(0, (\_DS - \_CS) + (\_SP / 16) + 1);//оставляем резидентной, первый параметр – код

//завершения, второй – объём памяти, который должен быть зарезервирован

//для программы после её завершения

return 0;

}

**Тест**

****

#### Результаты работы программы.

Разработал  резидентную программу выполняющую перенос всех векторов аппаратных прерываний ведущего и ведомого контроллера на пользовательские прерывания. Вывел на экран в двоичной форме следующие регистры контроллеров прерывания (как ведущего, так и ведомого):

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