Министерство образования Республики Беларусь

Учреждение образования Белорусский государственный университет информатики и радиоэлектроники

Кафедра ЭВМ

Отчёт по лабораторной работе №2

«Программирование контроллера прерываний»

Проверил: Выполнил:

к.т.н., доцент студент гр., 250502

Одинец Дмитрий Николаевич Грибовская А. А.

Минск 2024

**Задача**

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 регистр запросов прерывания, регистр масок, регистр обслуживаемых прерываний (как ведущего, так и ведомого).
2. Осуществлять переход на стандартные обработчики аппаратных прерываний, для обеспечения нормальной работы компьютера.

**Листинг программы**

#include <dos.h>

#include <time.h>

#include <stdio.h>

#include <stdlib.h>

void printRegister(int value, char far\* position) { //far - дальний

for (int i = 0; i < 8; i++) { //8 битов в байте

\*position = (value & 1) + '0'; // Преобразуем младший бит значения в символ '0' или '1' и записываем его в позицию position

position += 2; // Переходим к следующей позиции на экране

value >>= 1; // Сдвигаем значение на один бит вправо

}

}

void display() {

char temporary;

int index, value;

char far\* display\_memory = (char far\*)MK\_FP(0xB800, 0); // Указатель на экранную память, начиная с адреса 0xB800:0

//0xB800 - это сегментный адрес видеобуфера текстового режима, который //используется для вывода текста на экран в реальном режиме DOS и некоторых //версиях Windows.

/0 - это смещение от начала сегмента видеобуфера.

//Функция MK\_FP(segment, offset) объединяет сегмент и смещение в указатель far.

display\_memory += 20; // Сдвигаем указатель на 20 позиций вправо для вывода информации о масках

value = inp(0x21); // Читаем значение маски мастер-контроллера прерываний

printRegister(value, display\_memory); // Выводим битовое представление маски

display\_memory += 18; // Переходим к следующей строке на экране

value = inp(0xA1); // Читаем значение маски ведомого контроллера прерываний

printRegister(value, display\_memory); // Выводим битовое представление маски

display\_memory += 142; // Переходим к следующему блоку информации на экране

outp(0x20, 0x0A); // Устанавливаем запрос мастер-контроллера для чтения его регистра запросов outp(port, value)

value = inp(0x20); // Читаем регистр запросов мастер-контроллера

printRegister(value, display\_memory); // Выводим битовое представление регистра запросов мастер-контроллера

display\_memory += 18; // Переходим к следующей строке на экране

outp(0xA0, 0x0A); // Устанавливаем запрос ведомого контроллера для чтения его регистра запросов

value = inp(0xA0); // Читаем регистр запросов ведомого контроллера

printRegister(value, display\_memory); // Выводим битовое представление регистра запросов ведомого контроллера

display\_memory += 142; // Переходим к следующему блоку информации на экране

outp(0x20, 0x0B); // Устанавливаем запрос мастер-контроллера для чтения его регистра обслуживания

value = inp(0x20); // Читаем регистр обслуживания мастер-контроллера

printRegister(value, display\_memory); // Выводим битовое представление регистра обслуживания мастер-контроллера

display\_memory += 18; // Переходим к следующей строке на экране

outp(0xA0, 0x0B); // Устанавливаем запрос ведомого контроллера для чтения его регистра обслуживания

value = inp(0xA0); // Читаем регистр обслуживания ведомого контроллера

printRegister(value, display\_memory); // Выводим битовое представление регистра обслуживания ведомого контроллера

}

void prepare() {

\_disable(); // Отключаем прерывания

display(); // Выводим информацию о регистрах

}

// Объявляем массивы указателей на старые обработчики прерываний IRQ 0-7 и 8-15

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

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

// Описываем новые обработчики прерываний для IRQ 0-7

void interrupt newint08(...) {\_prepare(); oldint1[0](); \_enable(); } //enable - eof

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(); }

// Описываем новые обработчики прерываний для IRQ 8-15

void interrupt newint98(...) { prepare(); oldint2[0](); \_enable(); }

void interrupt newint99(...) { prepare(); oldint2[1](); \_enable(); }

void interrupt newintA0(...) { prepare(); oldint2[2](); \_enable(); }

void interrupt newintA1(...) { prepare(); oldint2[3](); \_enable(); }

void interrupt newintA2(...) { prepare(); oldint2[4](); \_enable(); }

void interrupt newintA3(...) { prepare(); oldint2[5](); \_enable(); }

void interrupt newintA4(...) { prepare(); oldint2[6](); \_enable(); }

void interrupt newintA5(...) { prepare(); oldint2[7](); \_enable(); }

// Инициализация контроллеров прерываний

void initialize()

{

int i;

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

oldint2[i - 0x70] = getvect(i); // Сохраняем старые обработчики IRQ 8-15

}

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

oldint1[i - 0x08] = getvect(i); // Сохраняем старые обработчики IRQ 0-7

}

// Устанавливаем новые обработчики IRQ 0-7

setvect(0x08, newint08);

setvect(0x09, newint09);

setvect(0x0A, newint0A);

setvect(0x0B, newint0B);

setvect(0x0C, newint0C);

setvect(0x0D, newint0D);

setvect(0x0E, newint0E);

setvect(0x0F, newint0F);

// Устанавливаем новые обработчики IRQ 8-15

setvect(0x70, newint98); // когда происходит прерывание 08h (например, тик системного таймера), управление передается функции newint98(), где выполняется соответствующая обработка прерывания.

setvect(0x71, newint99);

setvect(0x72, newintA0);

setvect(0x73, newintA1);

setvect(0x74, newintA2);

setvect(0x75, newintA3);

setvect(0x76, newintA4);

setvect(0x77, newintA5);

\_disable(); // Отключаем прерывания

// Инициализируем мастер-контроллер прерываний

outp(0x20, 0x11); // ICW1 - инициализация мастера

outp(0x21, 0x98); // ICW2 - базовый вектор для мастера

outp(0x21, 0x04); // ICW3 - номер порта ведомого контроллера на мастере

outp(0x21, 0x01); // ICW4 - дефолтное значение

// Инициализируем ведомый контроллер прерываний

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

outp(0xA1, 0x08); // ICW2 - базовый вектор для ведомого

outp(0xA1, 0x02); // ICW3 - номер порта подключения на мастере

outp(0xA1, 0x01); // ICW4 - дефолтное значение

\_enable(); // Включаем прерывания

}

int main()

{

unsigned far\* fp;

initialize(); // Инициализируем контроллеры прерываний

system("cls"); // Очищаем экран

printf("mask: \n");

printf("obsl:\n");

printf("requ:\n");

FP\_SEG(fp) = \_psp; // Получаем сегмент данных программы .data

FP\_OFF(fp) = 0x2c; // Получаем смещение сегмента данных программы

\_dos\_freemem(\*fp); // Освобождаем сегмент данных программы

\_dos\_keep(0, (\_DS - \_CS) + (\_SP / 16) + 1); // Оставляем программу резидентной. Этот код вызывает функцию \_dos\_keep(), которая сохраняет программу в резидентной памяти. Это позволяет программе остаться загруженной в памяти после ее завершения. Первый аргумент (0) указывает, что программа остается в памяти, независимо от того, успешно ли она завершилась или была прервана. Это значение представляет собой размер программы в параграфах (параграф - это сегмент памяти размером в 16 байт).

return 0;

}

**Тест**



