УО БГУИР

Кафедра ЭВМ

Отчет по лабораторной работе

Тема: «Подсистема прерываний»

Выполнил:

студент группы 650503 Юревич А.С.

Проверил:

к.т.н., доцент Одинец Д.Н.

Минск 2018

1. Постановка задачи

Вариант 16: D8H/08H

Под MS-DOS написать программу, которая:

1) выполняет инициализацию контроллера прерываний;

2) выводит на экран содержимое регистров запросов, обслуживаний и масок для ведущего и ведомого контроллеров (через видеобуфер).

\*При нажатии на клавиши меняется цвет или фон выводимой информации.

Программа должна быть резидентной. Все вектора прерываний переопределяются, новый базовый адрес выбирается в соответствии с вариантом.

1. Листинг программы

#include <dos.h>

#include <time.h>

#include <stdio.h>

#include <stdlib.h>

#define COLOR\_COUNT 7 //количество цветов

struct VIDEO

{

unsigned char symbol;

unsigned char attribute;

};

unsigned char colors[COLOR\_COUNT] = {0x71,0x62,0x43,0x54,0x35,0x26,0x17}; //массив цветов для видеобуфера

char color = 0x89; //изначальный цвет;

void changeColor() //изменение цвета на случайный

{

color = colors[rand() % COLOR\_COUNT];

return;

}

void print()

{

char temp; //текущий бит

int i, val;

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

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

for (i = 0; i < 8; i++) //проход по регистру

{

temp = val % 2; //получить последний бит

val = val >> 1; //сдвинуть на один бит вправо

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

screen->attribute = color; //установить цвет

screen++; //к следующему символу

}

screen++; //Пробел

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

for (i = 0; i < 8; i++) //проход по регистру

{

temp = val % 2; //получить последний бит

val = val >> 1; //сдвинуть на один бит вправо

screen->symbol = temp + '0'; //сохранить бит как символ ('0' или '1') screen->attribute = color; //установить цвет

screen++; //к следующему символу

}

screen += 63; //переход на следующую строку

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

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

for (i = 0; i < 8; i++) //проход по регистру

{

temp = val % 2; //получить последний бит

val = val >> 1; //сдвинуть на один бит вправо

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

screen->attribute = color; //установить цвет

screen++; //к следующему символу

}

screen++; //Пробел

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

val = inp(0xA0); //получить регистр запросов ведомого контроллера for (i = 0; i < 8; i++) //проход по регистру

{

temp = val % 2; //получить последний бит

val = val >> 1; //сдвинуть на один бит вправо

screen->symbol = temp + '0'; //сохранить бит как символ ('0' или '1') screen->attribute = color; //установить цвет

screen++; //к следующему символу

}

screen += 63; //переход на следующую строку

outp(0x20,0x0B); //переключение на регистр обслуживания ведущего контроллера val = inp(0x20); //получить регистр обслуживания ведущего контроллера

for (i = 0; i < 8; i++) //проход по регистру

{

temp = val % 2; //получить последний бит

val = val >> 1; //сдвинуть на один бит вправо

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

screen->attribute = color; //установить цвет

screen++; //к следующему символу

}

screen++; //Пробел

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

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

for (i = 0; i < 8; i++) //проход по регистру

{

temp = val % 2; //получить последний бит

val = val >> 1; //сдвинуть на один бит вправо

screen->symbol = temp + '0'; //сохранить бит как символ ('0' или '1') screen->attribute = color; //установить цвет

screen++; //к следующему символу

}

}

//IRQ 0-7

void interrupt(\*oldint8) (...); // IRQ 0 - таймер

void interrupt(\*oldint9) (...); // IRQ 1 - клавиатура

void interrupt(\*oldint10) (...); // IRQ 2 – каскадное подключение второго контроллера

void interrupt(\*oldint11) (...); // IRQ 3 – последовательный порт COM 2

void interrupt(\*oldint12) (...); // IRQ 4 – последовательный порт COM 1

void interrupt(\*oldint13) (...); // IRQ 5 – параллельный порт LPT2

void interrupt(\*oldint14) (...); // IRQ 6 – контроллер флоппи-дисководов

void interrupt(\*oldint15) (...); // IRQ 7 – параллельный порт LPT1

//IRQ 8-15

void interrupt(\*oldint70) (...); // IRQ 8 – часы реального времени

void interrupt(\*oldint71) (...); // IRQ 9 – свободный

void interrupt(\*oldint72) (...); // IRQ 10 – контроллер видеоадаптера

void interrupt(\*oldint73) (...); // IRQ 11 - свободный

void interrupt(\*oldint74) (...); // IRQ 12 – мышь PS/2

void interrupt(\*oldint75) (...); // IRQ 13 – математический сопроцессор

void interrupt(\*oldint76) (...); // IRQ 14 – первый контроллер жёсткого диска

void interrupt(\*oldint77) (...); // IRQ 15 – второй контроллер жёсткого диска

/\*Новые прерывания\*/

void interrupt newint08(...) { changeColor(); print(); oldint8(); } // int8

void interrupt newint09(...) { changeColor(); print(); oldint9(); } //int9

void interrupt newint0A(...) { changeColor(); print(); oldint10(); } //int10

void interrupt newint0B(...) { changeColor(); print(); oldint11(); } //int11

void interrupt newint0C(...) { changeColor(); print(); oldint12(); } //int12

void interrupt newint0D(...) { changeColor(); print(); oldint13(); } //int 13

void interrupt newint0E(...) { changeColor(); print(); oldint14(); } //int 14

void interrupt newint0F(...) { changeColor(); print(); oldint15(); } //int 15

void interrupt newintC0(...) { changeColor(); print(); oldint70(); } //int70

void interrupt newintC1(...) { changeColor(); print(); oldint71(); } //int71

void interrupt newintC2(...) { changeColor(); print(); oldint72(); } //int72

void interrupt newintC3(...) { changeColor(); print(); oldint73(); } //int73

void interrupt newintC4(...) { changeColor(); print(); oldint74(); } //int74

void interrupt newintC5(...) { changeColor(); print(); oldint75(); } //int75

void interrupt newintC6(...) { changeColor(); print(); oldint76(); } //int76

void interrupt newintC7(...) { changeColor(); print(); oldint77(); } //int77

void initialize()

{

//IRQ 0-7

oldint8 = getvect(0x08); // IRQ 0

oldint9 = getvect(0x09); // IRQ 1

oldint10 = getvect(0x0A); // IRQ 2

oldint11 = getvect(0x0B); // IRQ 3

oldint12 = getvect(0x0C); // IRQ 4

oldint13 = getvect(0x0D); // IRQ 5

oldint14 = getvect(0x0E); // IRQ 6

oldint15 = getvect(0x0F); // IRQ 7

//IRQ 8-15

oldint70 = getvect(0x70); // IRQ 8

oldint71 = getvect(0x71); // IRQ 9

oldint72 = getvect(0x72); // IRQ 10

oldint73 = getvect(0x73); // IRQ 11

oldint74 = getvect(0x74); // IRQ 12

oldint75 = getvect(0x75); // IRQ 13

oldint76 = getvect(0x76); // IRQ 14

oldint77 = getvect(0x77); // IRQ 15

/\*setvect помещает адрес процедуры обработки прерывания, на которую указывает (newint08), в таблицу векторов прерываний, положение задаётся (0xD8)\*/

//переопределение IRQ 0-7

setvect(0xD8, newint08);

setvect(0xD9, newint09);

setvect(0xDA, newint0A);

setvect(0xDB, newint0B);

setvect(0xDC, newint0C);

setvect(0xDE, newint0D);

setvect(0xDF, newint0E);

setvect(0xE1, newint0F);

//переопределение IRQ8-15

setvect(0x08, newintC0);

setvect(0x09, newintC1);

setvect(0x0A, newintC2);

setvect(0x0B, newintC3);

setvect(0x0C, newintC4);

setvect(0x0D, newintC5);

setvect(0x0E, newintC6);

setvect(0x0F, newintC7);

\_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, 0xD8); //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, 0x08); //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

initialize();

system("cls");

printf(" - mask\n"); //Регистр масок ведущего и ведомого контроллеров

printf(" - prepare\n"); //Регистр запросов ведущего и ведомого

printf(" - service\n"); //Регистр обслуживания ведущего и ведомого

printf("Master Slave\n");

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

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

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

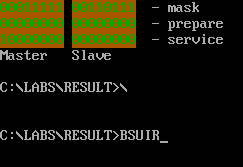
\_dos\_keep(0, (\_DS - \_CS) + (\_SP / 16) + 1); //оставляем резидентной, первый параметр – код завершения, второй – объём памяти, который должен быть зарезервирован для программы после её завершения

return 0;

}

1. Тестовые пример





1. Заключение

В ходе выполнения работы были получены практические знания об обработчике прерываний и изучена организация прерываний в IBM PC с использованием контроллера прерываний.