Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра электронных вычислительных машин

Лабораторная работа №2

«Программирование контроллера прерываний»

Вариант 16

Выполнил: Проверил:

Студент группы 250501 Преподаватель

Лукьянов Е.О. Одинец Д.Н.

Минск, 2024

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

* 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.

Листинг программы

Далее приведен листинг резидентной программы, выполняющей перенос всех векторов аппаратных прерываний ведущего и ведомого контроллера на пользовательские прерывания.

#include <dos.h>

#include <time.h>

#include <stdio.h>

#include <stdlib.h>

#define COLOR\_COUNT 7 //amount of colors

struct VIDEO

{

unsigned char symbol;

unsigned char attribute;

};

unsigned char colors[COLOR\_COUNT] =

{0x71,0x62,0x43,0x54,0x35,0x26,0x17}; //array of colours

char color = 0x89; //default color;

void changeColor() // izmenit' cvet na sluchaynyy {

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

return;

}

void print()

{

char temp; // bit that used now

int i, val;

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

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

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

{

temp = val % 2;

val = val >> 1;

screen->symbol = temp + '0';

screen->attribute = color;

screen++;

}

screen++; // space between master and slave

val = inp(0xA1); // get mask Slave registor

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

{

temp = val % 2;

val = val >> 1;

screen->symbol = temp + '0';

screen->attribute = color;

screen++;

}

screen += 63; //go to next line

outp(0x20,0x0A); //Master's request register

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

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

{

temp = val % 2;

val = val >> 1;

screen->symbol = temp + '0';

screen->attribute = color;

screen++;

}

screen++; // space

outp(0xA0,0x0A); //Slave's request register

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

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

{

temp = val % 2;

val = val >> 1;

screen->symbol = temp + '0';

screen->attribute = color;

screen++;

}

screen+=63; // go to next line

outp(0x20,0x0B); // Master's service register

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

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

{

temp = val % 2;

val = val >> 1;

screen->symbol = temp + '0';

screen->attribute = color;

screen++;

}

screen++; // space

outp(0xA0,0x0B); // Slave's service register

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

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

{

temp = val % 2;

val = val >> 1;

screen->symbol = temp + '0';

screen->attribute = color;

screen++;

}

}

// IRQ 0-7

void interrupt(\*oldint8) (...); // IRQ 0 - interrupt of timer (18,2 times per second)

void interrupt(\*oldint9) (...); // IRQ 1 - interrupt of keypad (press and release key)

void interrupt(\*oldint10) (...);// IRQ 2 - interrupt for cascade interruptions in AT machines

void interrupt(\*oldint11) (...);// IRQ 3 - interrupt of async port COM 2

void interrupt(\*oldint12) (...);// IRQ 4 - interrupt of async port COM 1

void interrupt(\*oldint13) (...);// IRQ 5 - interrupt of hard disk controller (for XT)

void interrupt(\*oldint14) (...);// IRQ 6 - interrupt of floppy disk controller (when finish operation with floppy disk)

void interrupt(\*oldint15) (...);// IRQ 7 - interrupt of printer (when printer is ready to work)

// IRQ 8-15

void interrupt(\*oldint70) (...);// IRQ 8 - interrupt of real time clock

void interrupt(\*oldint71) (...);// IRQ 9 - interrupt of EGA controller

void interrupt(\*oldint72) (...);// IRQ 10 - reserved interrupt

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

void interrupt(\*oldint74) (...);// IRQ 12 - reserved interrupt

void interrupt(\*oldint75) (...);// IRQ 13 - interrupt of mathematic soprocessor

void interrupt(\*oldint76) (...);// IRQ 14 - interrupt of hard disk

void interrupt(\*oldint77) (...);// IRQ 15 - reserved interrupt

void interrupt newint08(...) { print(); oldint8(); } // set function for service interrupt int8

void interrupt newint09(...) { changeColor(); print(); oldint9(); } // set function for service interrupt int9

void interrupt newint0A(...) { changeColor(); print(); oldint10(); } // set function for service interrupt int10

void interrupt newint0B(...) { changeColor(); print(); oldint11(); } // set function for service interrupt int11

void interrupt newint0C(...) { changeColor(); print(); oldint12(); } // set function for service interrupt int12

void interrupt newint0D(...) { changeColor(); print(); oldint13(); } // set function for service interrupt int13

void interrupt newint0E(...) { changeColor(); print(); oldint14(); } // set function for service interrupt int14

void interrupt newint0F(...) { changeColor(); print(); oldint15(); } // set function for service interrupt int15

void interrupt newint78(...) { changeColor(); print(); oldint70(); } // set function for service interrupt int70

void interrupt newint79(...) { changeColor(); print(); oldint71(); } // set function for service interrupt int71

void interrupt newint80(...) { changeColor(); print(); oldint72(); } // set function for service interrupt int72

void interrupt newint81(...) { changeColor(); print(); oldint73(); } // set function for service interrupt int73

void interrupt newint82(...) { changeColor(); print(); oldint74(); } // set function for service interrupt int74

void interrupt newint83(...) { changeColor(); print(); oldint75(); } // set function for service interrupt int75

void interrupt newint84(...) { changeColor(); print(); oldint76(); } // set function for service interrupt int76

void interrupt newint85(...) { changeColor(); print(); oldint77(); } // set function for service interrupt int77

void initialize()

{

//IRQ 0-7

oldint8 = getvect(0x08); // IRQ 0 - system timer

oldint9 = getvect(0x09); // IRQ 1 - keyboard controller

oldint10 = getvect(0x0A); // IRQ 2 - cascaded signals from IRQs 8-15 (any devices configured to use IRQ2 will actually be using IRQ9)

oldint11 = getvect(0x0B); // IRQ 3 - serial port controller for serial port 2 (shared with serial port 4, if present)

oldint12 = getvect(0x0C); // IRQ 4 - serial port controller for serial port 1 (shared with serial port 3, if present)

oldint13 = getvect(0x0D); // IRQ 5 - parallel port 2 and 3 or sound card

oldint14 = getvect(0x0E); // IRQ 6 - floppy disk controller

oldint15 = getvect(0x0F); // IRQ 7 - parallel port 1. It is used for printers or for any parallel port if a printer is not present

//IRQ 8-15

oldint70 = getvect(0x70); // IRQ 8 - real-time clock (RTC)

oldint71 = getvect(0x71); // IRQ 9 - Advanced Configuration and Power Interface (ACPI) system control interrupt

oldint72 = getvect(0x72); // IRQ 10 - the interrupt is left open for the use of peripherals (open interrupt/available, SCSI or NIC)

oldint73 = getvect(0x73); // IRQ 11 - the interrupt is left open for the use of peripherals (open interrupt/available, SCSI or NIC)

oldint74 = getvect(0x74); // IRQ 12 - mouse on PS/2 connector

oldint75 = getvect(0x75); // IRQ 13 - CPU co-processor or integrated floating point unit or inter-processor interrupt (use depends on OS)

oldint76 = getvect(0x76); // IRQ 14 - primary ATA channel (ATA interface usually serves hard disk drives and CD drives)

oldint77 = getvect(0x77); // IRQ 15 - secondary ATA channel

//set new handlers for IRQ 0-7

setvect(0xD8, newint08); //set new handler for IRQ 0

setvect(0xD9, newint09); //set new handler for IRQ 1

setvect(0xDA, newint0A); //set new handler for IRQ 2

setvect(0xDB, newint0B); //set new handler for IRQ 3

setvect(0xDC, newint0C); //set new handler for IRQ 4

setvect(0xDD, newint0D); //set new handler for IRQ 5

setvect(0xDE, newint0E); //set new handler for IRQ 6

setvect(0xDF, newint0F); //set new handler for IRQ 7

//set new handlers for IRQ8-15

setvect(0x08, newint78); //set new handlers for IRQ 8

setvect(0x09, newint79); //set new handlers for IRQ 9

setvect(0x0A, newint80); //set new handlers for IRQ 10

setvect(0x0B, newint81); //set new handlers for IRQ 11

setvect(0x0C, newint82); //set new handlers for IRQ 12

setvect(0x0D, newint83); //set new handlers for IRQ 13

setvect(0x0E, newint84); //set new handlers for IRQ 14

setvect(0x0F, newint85); //set new handlers for IRQ 15

\_disable(); // CLI

// interrupt initializtion for Master

outp(0x20, 0x11); //ICW1 - initialize master

outp(0x21, 0xD8); //ICW2 - base vector for master

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; // declare pointer

initialize(); // execute initialization

system("cls"); // clear console

printf(" - mask\n");

printf(" - prepare\n");

printf(" - service\n");

printf("Master Slave\n");

FP\_SEG(fp) = \_psp; // get segment

FP\_OFF(fp) = 0x2c; // get data segment offset with environment variables

\_dos\_freemem(\*fp); // free for DOS

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

Тестирование программы

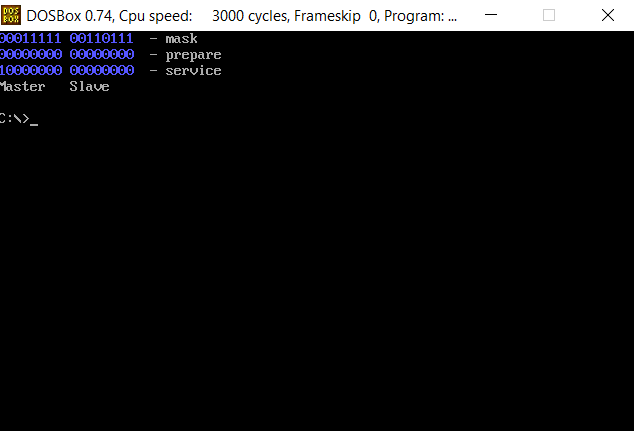


Рисунок 4.1 – Результат работы программы при запуске

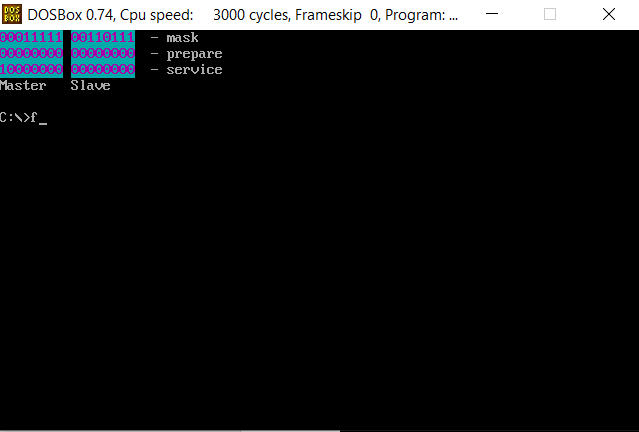


Рисунок 4.2 – Результат работы программы при нажатии клавиши

Заключение

В ходе лабораторной удалось выполнить перенос всех векторов аппаратных прерываний ведущего и ведомого контроллера на пользовательские прерывания. Использование контроллера прерываний позволяет ускорить взаимодействие процессора с внешними устройствами. Недостатком программы является клонирование программы в памяти при повторном запуске.

Программа компилировалась в BorlandC и запускалась в DOS, который эмулировался с помощью DosBox.