Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра электронных вычислительных машин

Лабораторная работа №3

«Программирование контроллера прерываний»

Вариант 21

Выполнил: Проверил:

Студент группы 050504 Преподаватель

Сидорович Я.А Одинец Д.Н.

Минск, 2022

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.

1. Осуществлять переход на стандартные обработчики аппаратных прерываний, для обеспечения нормальной работы компьютера.
2. 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.

1. 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 BUFF\_WIDTH 80

#define CENTER\_OFFSET 3

#define LEFT\_OFFSET 6

#define REG\_SCREEN\_SIZE 9

struct VIDEO

{

    unsigned char symb;

    unsigned char attr;

};

int attribute = 0x5e;           //color

void print(int offset, int value)                                       //transfer 2b

{

    char temp;

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

    screen += CENTER\_OFFSET \* BUFF\_WIDTH + offset;

    for (int i = 7; i >= 0; i--)

    {

        temp = value % 2;

        value /= 2;

        screen->symb = temp + '0';

        screen->attr = attribute;

        screen++;

    }

}

void getRegisterValue()

{                                                                               //Для ведущего

    print(0 + LEFT\_OFFSET, inp(0x21));                                          //Регистр масок

    outp(0x20, 0x0B);

    print(REG\_SCREEN\_SIZE + LEFT\_OFFSET, inp(0x20));                            //Регистр обслуж. прерываний

    outp(0x20, 0x0A);

    print(REG\_SCREEN\_SIZE \* 2 + LEFT\_OFFSET, inp(0x20));                        //Регистр запросов на прерывание

    print(BUFF\_WIDTH + LEFT\_OFFSET, inp(0xA1));                                 //Для ведомого

    outp(0xA0, 0x0B);

    print(BUFF\_WIDTH + REG\_SCREEN\_SIZE + LEFT\_OFFSET, inp(0xA0));

    outp(0xA0, 0x0A);

    print(BUFF\_WIDTH + REG\_SCREEN\_SIZE \* 2 + LEFT\_OFFSET, inp(0xA0));

}

// IRQ 0-7

void interrupt(\*oldintnum8) (...);      // IRQ 0

void interrupt(\*oldintnum9) (...);      // IRQ 1

void interrupt(\*oldintnum10) (...);     // IRQ 2

void interrupt(\*oldintnum11) (...);     // IRQ 3

void interrupt(\*oldintnum12) (...);     // IRQ 4

void interrupt(\*oldintnum13) (...);     // IRQ 5

void interrupt(\*oldintnum14) (...);     // IRQ 6

void interrupt(\*oldintnum15) (...);     // IRQ 7

// IRQ 8-15

void interrupt(\*oldintnum70) (...);     // IRQ 8

void interrupt(\*oldintnum71) (...);     // IRQ 9

void interrupt(\*oldintnum72) (...);     // IRQ 10

void interrupt(\*oldintnum73) (...);     // IRQ 11

void interrupt(\*oldintnum74) (...);     // IRQ 12

void interrupt(\*oldintnum75) (...);     // IRQ 13

void interrupt(\*oldintnum76) (...);     // IRQ 14

void interrupt(\*oldintnum77) (...);     // IRQ 15

void interrupt  newintnum08(...) { getRegisterValue(); oldintnum8(); }

void interrupt  newintnum09(...) { attribute++; getRegisterValue(); oldintnum9(); }

void interrupt  newintnum0A(...) { getRegisterValue(); oldintnum10(); }

void interrupt  newintnum0B(...) { getRegisterValue(); oldintnum11(); }

void interrupt  newintnum0C(...) { getRegisterValue(); oldintnum12(); }

void interrupt  newintnum0D(...) { getRegisterValue(); oldintnum13(); }

void interrupt  newintnum0E(...) { getRegisterValue(); oldintnum14(); }

void interrupt  newintnum0F(...) { getRegisterValue(); oldintnum15(); }

void interrupt  newintnumC0(...) { getRegisterValue(); oldintnum70(); }

void interrupt  newintnumC1(...) { getRegisterValue(); oldintnum71(); }

void interrupt  newintnumC2(...) { getRegisterValue(); oldintnum72(); }

void interrupt  newintnumC3(...) { getRegisterValue(); oldintnum73(); }

void interrupt  newintnumC4(...) { getRegisterValue(); oldintnum74(); }

void interrupt  newintnumC5(...) { getRegisterValue(); oldintnum75(); }

void interrupt  newintnumC6(...) { getRegisterValue(); oldintnum76(); }

void interrupt  newintnumC7(...) { getRegisterValue(); oldintnum77(); }

void initialize()

{

    //Получаем изначальные вектора

    oldintnum8 = getvect(0x08);

    oldintnum9 = getvect(0x09);

    oldintnum10 = getvect(0x0A);

    oldintnum11 = getvect(0x0B);

    oldintnum12 = getvect(0x0C);

    oldintnum13 = getvect(0x0D);

    oldintnum14 = getvect(0x0E);

    oldintnum15 = getvect(0x0F);

    oldintnum70 = getvect(0x70);

    oldintnum71 = getvect(0x71);

    oldintnum72 = getvect(0x72);

    oldintnum73 = getvect(0x73);

    oldintnum74 = getvect(0x74);

    oldintnum75 = getvect(0x75);

    oldintnum76 = getvect(0x76);

    oldintnum77 = getvect(0x77);

    //set new handlers for IRQ 0-7

    setvect(0x08, newintnum08);

    setvect(0x09, newintnum09);

    setvect(0x0A, newintnum0A);

    setvect(0x0B, newintnum0B);

    setvect(0x0C, newintnum0C);

    setvect(0x0D, newintnum0D);

    setvect(0x0E, newintnum0E);

    setvect(0x0F, newintnum0F);

    //set new handlers for IRQ8-15

    setvect(0x80, newintnumC0);

    setvect(0x81, newintnumC1);

    setvect(0x82, newintnumC2);

    setvect(0x83, newintnumC3);

    setvect(0x84, newintnumC4);

    setvect(0x85, newintnumC5);

    setvect(0x86, newintnumC6);

    setvect(0x87, newintnumC7);

    \_disable();                                 //Отключить обработку прерываний CLI

    // interrupt initializtion for Master

    outp(0x20, 0x11);   //ICW1 - Инициализация прерывания контроллера

    outp(0x21, 0x08);   //ICW2 - устанавливает адрес вектора прерывания

    outp(0x21, 0x04);   //ICW3 - the port bit of Slave (in binary format)

    outp(0x21, 0x01);   //ICW4 - настройка доп режимов работы (default)

    // interrupt initialization for Slave

    outp(0xA0, 0x11);  //ICW1 - initialize Slave

    outp(0xA1, 0x80);  //ICW2 - base vector for slave

    outp(0xA1, 0x02);  //ICW3 - the port number of connected port on Master

    outp(0xA1, 0x01);  //ICW4 - default

    \_enable();                                  //Возобнавляем работу прерываний STI

}

int main(){

    unsigned far \*fp;   //модификатор far оказывает воздействие на работу с адресами объекта

    initialize();

    system("cls");

    printf("\n\n        mask    service  prepare\n");

    printf("                                  Master\n");

    printf("                                  Slave\n\n\n");

    //Блок освобождения памяти

    FP\_SEG(fp) = \_psp;

    FP\_OFF(fp) = 0x2c;

    \_dos\_freemem(\*fp);

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

    return 0;

}

}

1. Тестирование программы

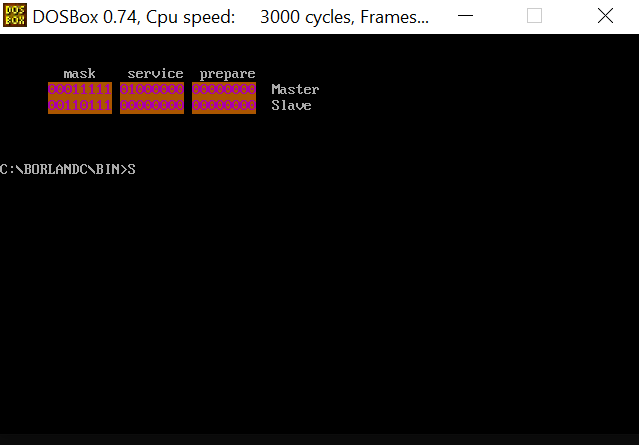


Рисунок 4.2 – Результат работы программы при нажатии клавиши.

1. 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 Turbo C++ и запускалась в DOS, который эмулировался с помощью DosBox.