Министерство образования Республики Беларусь

Учреждение образования

“Белорусский государственный университет информатики и радиоэлектроники”

Кафедра электронных вычислительных машин

Отчёт по лабораторной работе №2

“ Программирование контроллера прерываний.”

Вариант 3

Выполнил: Проверил:

Студент группы 130501 Преподаватель

Блинников В.А. Марзалюк А.В.

Минск 2023

1. **Постановка задачи**

Целью данной работы является разработка программного модуля переноса всех векторов аппаратных прерываний ведущего и ведомого контроллера на пользовательские прерывания.

1. **Алгоритм**

Каждый момент времени центральный процессор может работать только с одним устройством. Циклический опрос каждого устройства с последующей обработкой запроса оказался неэффективным. Решение задачи оказался контроллер прерываний, который принимает запросы от устройств и в соответствии с приоритетом направляет их процессору, если прерывание от данного устройства не замаскировано (разрешено) в регистре масок. Если прерывание разрешено и устройство его запросило, то устанавливается соответствующий устройству бит в регистре запросов.

Контроллер прерываний состоит из двух микросхем, подключенных каскадно (ведущий и ведомый контроллеры), каждая из которых имеет по 8 линий прерываний (IRQ0-IRQ7, IRQ8-IRQ15). За каждой линией закреплено определенное устройство.

Когда процессор получает запрос на прерывание, он сохраняет свое текущее состояние и переключается на выполнение запрошенной операции. При этом устанавливается бит в регистре обслуживания (бит запроса сбрасывается). После обслуживания прерываний сбрасывается бит обслуживания, посылается сигнал EOI (endofinterrupt), процессор переключается на выполняемую ранее задачу.

Для доступа к контроллеру прерываний используются порты 20h и 21h (для ведущего), A0 иA1h (для ведомого).

Регистр масок доступен через порт 21h /A1h. Чтобы изменить определенный бит, нужно считать значение из этого регистра, изменить нужный бит, записать значение обратно.

Чтобы считать регистр запросов, его нужно выбрать (записать в 20h/A0h значение 0Ah), а затем считать содержимое из порта 20h/A0h.

Чтобы считать регистр обслуживания, его нужно выбрать (записать в 20h/A0h значение 0Bh), а затем считать содержимое из порта 20h/A0h.

Для резидентной программы понадобится следующий фрагмент кода:

unsigned far \*fp; //объявляем указатель

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

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

\_dos\_freemem(\*fp); //чтобы его освободить

\_dos\_keep(0,(\_DS -\_CS)+(\_SP/16)+1);//оставляем резидентной, указывая //первым параметром код завершения, а //вторым - объем памяти, который должен //быть зарезервирован для программы //после ее завершения

**Листинг программы**

#include <dos.h>

#include "iostream.h"

struct VIDEO {

unsigned char symb;

unsigned char attr;

};

void get\_reg(); //get data from registres

//IRQ0-7

void interrupt(\*int8) (...);

void interrupt(\*int9) (...);

void interrupt(\*intA) (...);

void interrupt(\*intB) (...);

void interrupt(\*intC) (...);

void interrupt(\*intD) (...);

void interrupt(\*intE) (...);

void interrupt(\*intF) (...);

//IRQ8-15

void interrupt(\*int0) (...);

void interrupt(\*int1) (...);

void interrupt(\*int2) (...);

void interrupt(\*int3) (...);

void interrupt(\*int4) (...);

void interrupt(\*int5) (...);

void interrupt(\*int6) (...);

void interrupt(\*int7) (...);

// new interrupt handlers

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

void interrupt new9(...) { get\_reg(); int9(); }

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

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

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

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

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

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

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

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

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

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

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

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

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

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

//return 0, if no keyboard interruption (second bit check)

int checkKeyBoardInt(int value) {

value /= 2;

return value % 2;

}

void print(int mquery, int mservice, int mmasque, int squery, int sservice, int smasque) {

static int toSwitch = 0; //if we put ke it will be 1

char temp;

int i;

int deltaAddress = 55;

char\* slave = "slave ";

char\* master = "master";

char\* query = " query: ";

char\* service = " service: ";

char\* masque = " masque: ";

toSwitch = (toSwitch + checkKeyBoardInt(mservice)) % 4;

unsigned char defaultAttr = 0x7C;

unsigned char keyboardAttr = 0x1D;

unsigned char currAttr = toSwitch > 1 ? defaultAttr : keyboardAttr;

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

for (i = 0; i < 6; i++) { // print master

screen->symb = master[i];

screen->attr = currAttr;

screen++;

}

for (i = 0; i < 11; i++) { //print query

screen->symb = query[i];

screen->attr = currAttr;

screen++;

}

for (i = 8; i > 0; i--) { //BITS TO CHAR

temp = mquery % 2;

mquery /= 2; //fill the screen

screen->symb = temp + '0';

screen->attr = currAttr;

screen++;

}

screen += deltaAddress;

for (i = 0; i < 6; i++) { //print master

screen->symb = master[i];

screen->attr = currAttr;

screen++;

}

for (i = 0; i < 11; i++) { //print service

screen->symb = service[i];

screen->attr = currAttr;

screen++;

}

for (i = 8; i > 0; i--) { //BITS TO CHAR

temp = mservice % 2;

mservice /= 2; //fill the screen

screen->symb = temp + '0';

screen->attr = currAttr;

screen++;

}

screen += deltaAddress;

for (i = 0; i < 6; i++) { // print master

screen->symb = master[i];

screen->attr = currAttr;

screen++;

}

for (i = 0; i < 11; i++) { //print masque

screen->symb = masque[i];

screen->attr = currAttr;

screen++;

}

for (i = 8; i > 0; i--) { //BITS TO CHAR

temp = mmasque % 2;

mmasque /= 2; //fill the screen

screen->symb = temp + '0';

screen->attr = currAttr;

screen++;

}

screen += deltaAddress;

for (i = 0; i < 6; i++) { // print slave

screen->symb = slave[i];

screen->attr = currAttr;

screen++;

}

for (i = 0; i < 11; i++) { //print query

screen->symb = query[i];

screen->attr = currAttr;

screen++;

}

for (i = 8; i > 0; i--) { //BITS TO CHAR

temp = squery % 2;

squery /= 2; //fill the screen

screen->symb = temp + '0';

screen->attr = currAttr;

screen++;

}

screen += deltaAddress;

for (i = 0; i < 6; i++) { // print slave

screen->symb = slave[i];

screen->attr = currAttr;

screen++;

}

for (i = 0; i < 11; i++) //print service

{

screen->symb = service[i];

screen->attr = currAttr;

screen++;

}

for (i = 8; i > 0; i--) //BITS TO CHAR

{

temp = sservice % 2;

sservice /= 2; //fill the screen

screen->symb = temp + '0';

screen->attr = currAttr;

screen++;

}

screen += deltaAddress;

for (i = 0; i < 6; i++) // print slave

{

screen->symb = slave[i];

screen->attr = currAttr;

screen++;

}

for (i = 0; i < 11; i++) //print masque

{

screen->symb = masque[i];

screen->attr = currAttr;

screen++;

}

for (i = 8; i > 0; i--) //BITS TO CHAR

{

temp = smasque % 2;

smasque /= 2; //fill the screen

screen->symb = temp + '0';

screen->attr = currAttr;

screen++;

}

}

// get data from registers

void get\_reg() {

int mquery, mservice, mmasque, squery, sservice, smasque;

outp(0x20, 0x0A);//master query register

mquery = inp(0x20);

outp(0x20, 0x0B);//master service register

mservice = inp(0x20);

//outp(0x21, 0x0B);//master mask register (check)

mmasque = inp(0x21);

outp(0xA0, 0x0A);//slave query register

squery = inp(0xA0);

outp(0xA0, 0x0B);//slave service register

sservice = inp(0xA0);

//outp(0xA1, 0x0B);//slave mask register

smasque = inp(0xA1);

print(mquery, mservice, mmasque, squery, sservice, smasque);

}

void init() {

//IRQ0-7

int8 = getvect(0x68); // прерывание таймера возникает 18,2 раза в секунду

int9 = getvect(0x69); //IRQ1 прерывание от клавиатуры

intA = getvect(0x70); //IRQ2 используется для каскадирования аппаратных прерываний

intB = getvect(0x71); //IRQ3 прерывание асинхронного порта COM2

intC = getvect(0x72); //IRQ4 прерывание асинхронного порта COM1

intD = getvect(0x73); //IRQ5 прерывание конроллера жесткого диска для XT

intE = getvect(0x74); //IRQ6 прерывание генерируется контроллером флоппи диска

intF = getvect(0x75); //IRQ7 прерывание принтера

int0 = getvect(0x08); //IRQ8 прерывание от часов реального времени

int1 = getvect(0x09); //IRQ9 прерывание контроллера EGA

int2 = getvect(0x0A); //IRQ10 зарезервировано

int3 = getvect(0x0B); //IRQ11 зарезервировано

int4 = getvect(0x0C); //IRQ12 зарезервировано

int5 = getvect(0x0D); //IRQ13 прерывание от математического сопроцессора

int6 = getvect(0x0E); //IRQ14 прерывание от контроллера жесткого диска

int7 = getvect(0x0F); //IRQ15 зарезервировано

//move address of procedure interuption handler

setvect(0x68, new8);

setvect(0x69, new9);

setvect(0x70, newA);

setvect(0x71, newB);

setvect(0x72, newC);

setvect(0x73, newD);

setvect(0x74, newE);

setvect(0x75, newF);

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(); //disable interuption

outp(0xA0, 0x11);

outp(0x20, 0x11); //ICW1 (initialization of master controller)

outp(0x21, 0x08); //ICW2 устанавливает адрес вектора прерывания для IRQ0

outp(0xA1, 0x80);

outp(0xA1, 0x02);

outp(0x21, 0x04); //ICW3

outp(0x21, 0x11); //ICW4 additional work mode

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

\_enable(); //enable interuption

}

int main() {

unsigned far\* fp;

init();

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

FP\_OFF(fp) = 0x2c; //get far type offset by segment

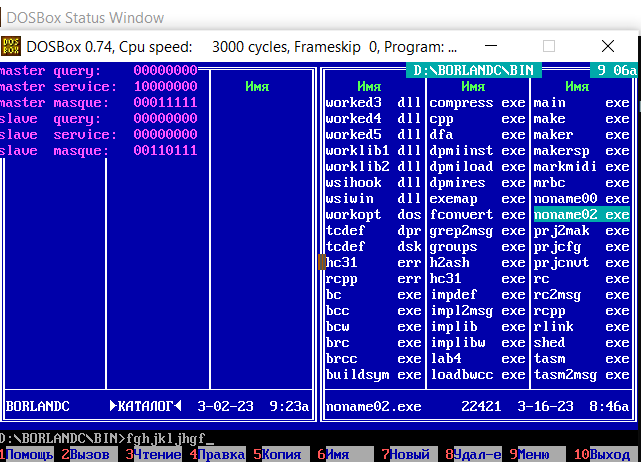
\_dos\_freemem(\*fp); //free memory block

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

return 0;

}

**Тест**

****

**Заключение**

В данной лабораторной работе разработан программный модуль реализации переноса всех векторов аппаратных прерываний ведущего и ведомого контроллера на пользовательские прерывания.