Министерство образования Республики Беларусь

Учреждение образования Белорусский государственный университет информатики и радиоэлектроники

Кафедра ЭВМ

Отчёт по лабораторной работе №3

“Программирование контроллера прерываний”

Проверил: Выполнил:

к.т.н., доцент студент гр.850505

Одинец Дмитрий Николаевич Бурень С. В.

Минск 2020

**Задача**

Целью данной работы является написание резидентной программы, выполняющей перенос всех векторов аппаратных прерываний ведущего и ведомого контроллеров на пользовательский прерывания.

**Алгоритм**

Программа состоит из нескольких подпрограмм (частей программы), представляющих собой некоторые функции. К ним относятся функции:

* вывод на экран значения одного регистра
* вывод на экран значений регистров двух контроллеров
* объявление указателей на обработчики функций (16 функций)
* обработчики прерываний (16 функций)
* функция инициализации

Функция инициализации считывает в указатели на обработчики функций адреса стандартных обработчиков, после чего устанавливаются новые обработчики для ведущего и ведомого контроллера, начиная с векторов 0х80 и 0х08 соответственно. После этого происходит инициализация контроллеров, связывающая полученные векторы с ними.

**Листинг программы**

#include <dos.h>

#define BUFF\_WIDTH 80

#define CENTER\_OFFSET 12

#define LEFT\_OFFSET 20

#define REG\_SCREEN\_SIZE 9

struct VIDEO

{

unsigned char symb;

unsigned char attr;

};

int attribute = 0x5e; //color

void print(int offset, int value);

void getRegisterValue();

void interrupt(\*oldHandle08) (...);

void interrupt(\*oldHandle09) (...);

void interrupt(\*oldHandle0A) (...);

void interrupt(\*oldHandle0B) (...);

void interrupt(\*oldHandle0C) (...);

void interrupt(\*oldHandle0D) (...);

void interrupt(\*oldHandle0E) (...);

void interrupt(\*oldHandle0F) (...);

void interrupt(\*oldHandle70) (...);

void interrupt(\*oldHandle71) (...);

void interrupt(\*oldHandle72) (...);

void interrupt(\*oldHandle73) (...);

void interrupt(\*oldHandle74) (...);

void interrupt(\*oldHandle75) (...);

void interrupt(\*oldHandle76) (...);

void interrupt(\*oldHandle77) (...);

void interrupt newHandleB0(...) { getRegisterValue(); oldHandle08(); }

void interrupt newHandleB1(...) { attribute++; getRegisterValue(); oldHandle09(); }

void interrupt newHandleB2(...) { getRegisterValue(); oldHandle0A(); }

void interrupt newHandleB3(...) { getRegisterValue(); oldHandle0B(); }

void interrupt newHandleB4(...) { getRegisterValue(); oldHandle0C(); }

void interrupt newHandleB5(...) { getRegisterValue(); oldHandle0D(); }

void interrupt newHandleB6(...) { getRegisterValue(); oldHandle0E(); }

void interrupt newHandleB7(...) { getRegisterValue(); oldHandle0F(); }

void interrupt newHandle08(...) { getRegisterValue(); oldHandle70(); }

void interrupt newHandle09(...) { getRegisterValue(); oldHandle71(); }

void interrupt newHandle0A(...) { getRegisterValue(); oldHandle72(); }

void interrupt newHandle0B(...) { getRegisterValue(); oldHandle73(); }

void interrupt newHandle0C(...) { getRegisterValue(); oldHandle74(); }

void interrupt newHandle0D(...) { getRegisterValue(); oldHandle75(); }

void interrupt newHandle0E(...) { getRegisterValue(); oldHandle76(); }

void interrupt newHandle0F(...) { getRegisterValue(); oldHandle77(); }

void print(int offset, int value)

{

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()

{

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

outp(0x20, 0x0B);

print(REG\_SCREEN\_SIZE + LEFT\_OFFSET, inp(0x20));

outp(0x20, 0x0A);

print(REG\_SCREEN\_SIZE \* 2 + LEFT\_OFFSET, inp(0x20));

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

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));

}

void init()

{

// IRQ0-7

oldHandle08 = getvect(0x08); // Timer`

oldHandle09 = getvect(0x09); // Keyboard

oldHandle0A = getvect(0x0A); // Slave IRQ

oldHandle0B = getvect(0x0B); // Random deviece

oldHandle0C = getvect(0x0C); // Random deviece

oldHandle0D = getvect(0x0D); // Random deviece

oldHandle0E = getvect(0x0E); // Random deviece

oldHandle0F = getvect(0x0F); // Random deviece

// IRQ8-15

oldHandle70 = getvect(0x70); // Real time clock

oldHandle71 = getvect(0x71); // Random deviece

oldHandle72 = getvect(0x72); // Random deviece

oldHandle73 = getvect(0x73); // Random deviece or timer

oldHandle74 = getvect(0x74); // PS/2 mouse

oldHandle75 = getvect(0x75); // FPU error

oldHandle76 = getvect(0x76); // Random deviece or first ATA controller

oldHandle77 = getvect(0x77); // Random deviece or second ATA controller ч

setvect(0x08, newHandleB0);

setvect(0x09, newHandleB1);

setvect(0x0A, newHandleB2);

setvect(0x0B, newHandleB3);

setvect(0x0C, newHandleB4);

setvect(0x0D, newHandleB5);

setvect(0x0E, newHandleB6);

setvect(0x0F, newHandleB7);

setvect(0x70, newHandle08);

setvect(0x71, newHandle09);

setvect(0x72, newHandle0A);

setvect(0x73, newHandle0B);

setvect(0x74, newHandle0C);

setvect(0x75, newHandle0D);

setvect(0x76, newHandle0E);

setvect(0x77, newHandle0F);

// Disable interrupts handling (cli)

\_disable();

// Master

outp(0x20, 0x11); // ICW1

outp(0x21, 0xB0); // ICW2

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

outp(0x21, 0x01); // ICW4

// Slave

outp(0xA0, 0x11); // ICW1

outp(0xA1, 0x08); // ICW2

outp(0xA1, 0x02); // ICW3

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

// Enable interrupt handling (sti)

\_enable();

}

int main()

{

unsigned far \*fp; // pointer declaration

init(); // execute initialization

FP\_SEG(fp) = \_psp; // returns the segment of the far pointer

FP\_OFF(fp) = 0x2c; // returns the offset of the far pointer

\_dos\_freemem(\*fp); // frees a block of memory whose first byte is located in a fp

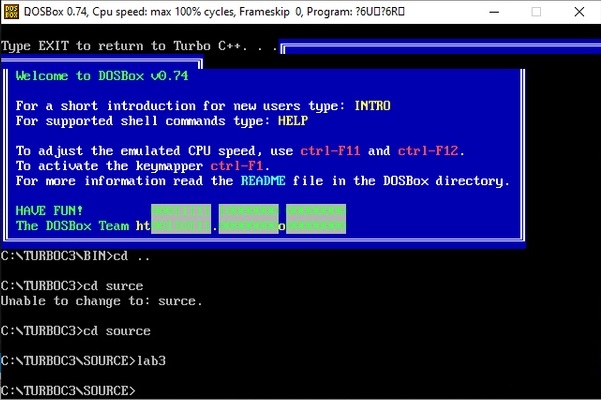
\_dos\_keep(0, (\_DS - \_CS) + (\_SP / 16) + 1); // terminates program execution, although it remains in memory resident

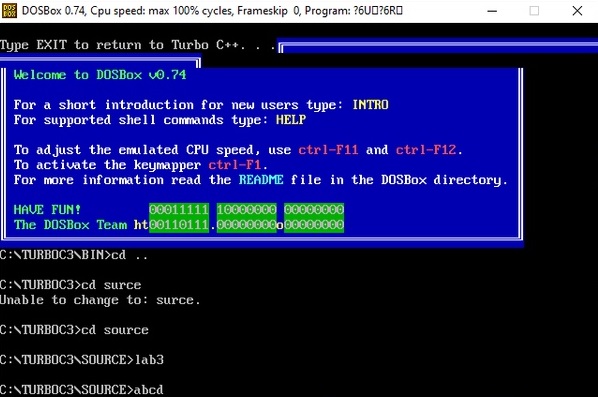
// 1st param - ending code, 2nd param - resident program size

return 0;

}

**Тест**



****

**Заключение**

В данной лабораторной работе разработана программа, которая переносит вектора аппаратных прерываний ведущего и ведомого контроллера на вектора пользовательских прерываний.