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.150501

Одинец Дмитрий Николаевич Черноок А.Ю.

Минск 2023

**Задача**

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

**Алгоритм**

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

* обработчики прерываний (16 функций)
* объявление указателей на обработчики функций (16 функций)
* функция инициализации
* вывод на экран содержимого одного регистра
* вывод на экран значений регистров ведомого и ведущего контроллеров

Функция инициализации считывает в указатели на обработчики функций адреса стандартных обработчиков, после чего устанавливаются новые обработчики для ведущего и ведомого контроллера, начиная с векторов 0х08 и 0хA0 соответственно. После этого происходит инициализация контроллеров, связывающая полученные векторы с ними.

**Листинг программы**

#include <dos.h>

#include <stdio.h>

struct VideoMemSymbol

{

char ascii;

char attribute;

};

void headerPrint();

void registerValuePrint(int value, int offset);

void registerPrint();

// Previous IRQ 0-7:

void interrupt(\*prevHandle08)(...); // timer

void interrupt(\*prevHandle09)(...); // keyboard

void interrupt(\*prevHandle0A)(...); // cascade iconnection of the second controller

void interrupt(\*prevHandle0B)(...); // COM 2

void interrupt(\*prevHandle0C)(...); // COM 1

void interrupt(\*prevHandle0D)(...); // lpt2 (parallel port)

void interrupt(\*prevHandle0E)(...); // floppy disk controller

void interrupt(\*prevHandle0F)(...); // lpt2 (parallel port)

// IRQ 8-15:

void interrupt(\*prevHandle70)(...); // real time clock

void interrupt(\*prevHandle71)(...); // free

void interrupt(\*prevHandle72)(...); // video adapter

void interrupt(\*prevHandle73)(...); // free

void interrupt(\*prevHandle74)(...); // mouse ps/2

void interrupt(\*prevHandle75)(...); // mathematic soprocessor

void interrupt(\*prevHandle76)(...); // hard disk first controller

void interrupt(\*prevHandle77)(...); // hard disk second controller

// New IRQ 0-7:

void interrupt newHandle08(...) { registerPrint(); prevHandle08(); }

void interrupt newHandle09(...) { registerPrint(); prevHandle09(); }

void interrupt newHandle0A(...) { registerPrint(); prevHandle0A(); }

void interrupt newHandle0B(...) { registerPrint(); prevHandle0B(); }

void interrupt newHandle0C(...) { registerPrint(); prevHandle0C(); }

void interrupt newHandle0D(...) { registerPrint(); prevHandle0D(); }

void interrupt newHandle0E(...) { registerPrint(); prevHandle0E(); }

void interrupt newHandle0F(...) { registerPrint(); prevHandle0F(); }

// IRQ 8-15:

void interrupt newHandle70(...) { registerPrint(); prevHandle70(); }

void interrupt newHandle71(...) { registerPrint(); prevHandle71(); }

void interrupt newHandle72(...) { registerPrint(); prevHandle72(); }

void interrupt newHandle73(...) { registerPrint(); prevHandle73(); }

void interrupt newHandle74(...) { registerPrint(); prevHandle74(); }

void interrupt newHandle75(...) { registerPrint(); prevHandle75(); }

void interrupt newHandle76(...) { registerPrint(); prevHandle76(); }

void interrupt newHandle77(...) { registerPrint(); prevHandle77(); }

void VectorsInitialization()

{

// get default vectors:

prevHandle08 = getvect(0x08);

prevHandle09 = getvect(0x09);

prevHandle0A = getvect(0x0A);

prevHandle0B = getvect(0x0B);

prevHandle0C = getvect(0x0C);

prevHandle0D = getvect(0x0D);

prevHandle0E = getvect(0x0E);

prevHandle0F = getvect(0x0F);

prevHandle70 = getvect(0x70);

prevHandle71 = getvect(0x71);

prevHandle72 = getvect(0x72);

prevHandle73 = getvect(0x73);

prevHandle74 = getvect(0x74);

prevHandle75 = getvect(0x75);

prevHandle76 = getvect(0x76);

prevHandle77 = getvect(0x77);

//set new vectors:

setvect(0x08, newHandle08);

setvect(0x09, newHandle09);

setvect(0x0A, newHandle0A);

setvect(0x0B, newHandle0B);

setvect(0x0C, newHandle0C);

setvect(0x0D, newHandle0D);

setvect(0x0E, newHandle0E);

setvect(0x0F, newHandle0F);

setvect(0xA0, newHandle70);

setvect(0xA1, newHandle71);

setvect(0xA2, newHandle72);

setvect(0xA3, newHandle73);

setvect(0xA4, newHandle74);

setvect(0xA5, newHandle75);

setvect(0xA6, newHandle76);

setvect(0xA7, newHandle77);

\_disable(); // Disable interrupts handling (cli)

// Master interrupt initializtion:

outp(0x20, 0x11); //ICW1: start initialization comand // 00010001b

outp(0x21, 0x08); //ICW2: new Master vector

outp(0x21, 0x04); //ICW3: Slave controller bit // 00000100b

outp(0x21, 0x01); //ICW4: operation mode // 00000001b

// Slave interrupt initialization:

outp(0xA0, 0x11); //ICW1: start initialization comand // 00010001b

outp(0xA1, 0xA0); //ICW2: new Slave vector

outp(0xA1, 0x02); //ICW3: number of port connected to Master // 00000010b

outp(0xA1, 0x01); //ICW4: operation mode // 00000001b

\_enable(); // Enable interrupt handling (sti)

}

void registerPrint()

{

//Master registors:

registerValuePrint(inp(0x21), 141); // Mask reg

outp(0x20, 0x0A);

registerValuePrint(inp(0x20), 221); // Request reg

outp(0x20, 0x0B);

registerValuePrint(inp(0x20), 301); // Service reg

//Slave registors:

registerValuePrint(inp(0xA1), 150); // Mask reg

outp(0xA0, 0x0A);

registerValuePrint(inp(0xA0), 230); // Request reg

outp(0xA0, 0x0B);

registerValuePrint(inp(0xA0), 310); // Service reg

}

void registerValuePrint(int value, int offset)

{

headerPrint();

char sAttribute = 0x3A;

VideoMemSymbol far\* vidMemory = (VideoMemSymbol far\*)MK\_FP(0xB800, 0);

vidMemory += 7 + offset;

char temp;

for (int i = 7; i >= 0; i--)

{

temp = value % 2;

value /= 2;

vidMemory->attribute = sAttribute;

vidMemory->ascii = temp + '0';

vidMemory--;

}

}

void headerPrint()

{

char header0[] = " Controller observe: ";

char symbolAttribute = 0x1D; // 0b 0 001 1 101;

VideoMemSymbol far\* vidMemor = (VideoMemSymbol far\*)MK\_FP(0xB800, 0);

vidMemor += 59;

for (int i = 0; header0[i]; i++) {

vidMemor->ascii = header0[i];

vidMemor->attribute = symbolAttribute;

vidMemor++;

}

}

int main()

{

unsigned far\* fp; // pointer

VectorsInitialization();

headerPrint();

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

FP\_OFF(fp) = 0x2c; // shift segment

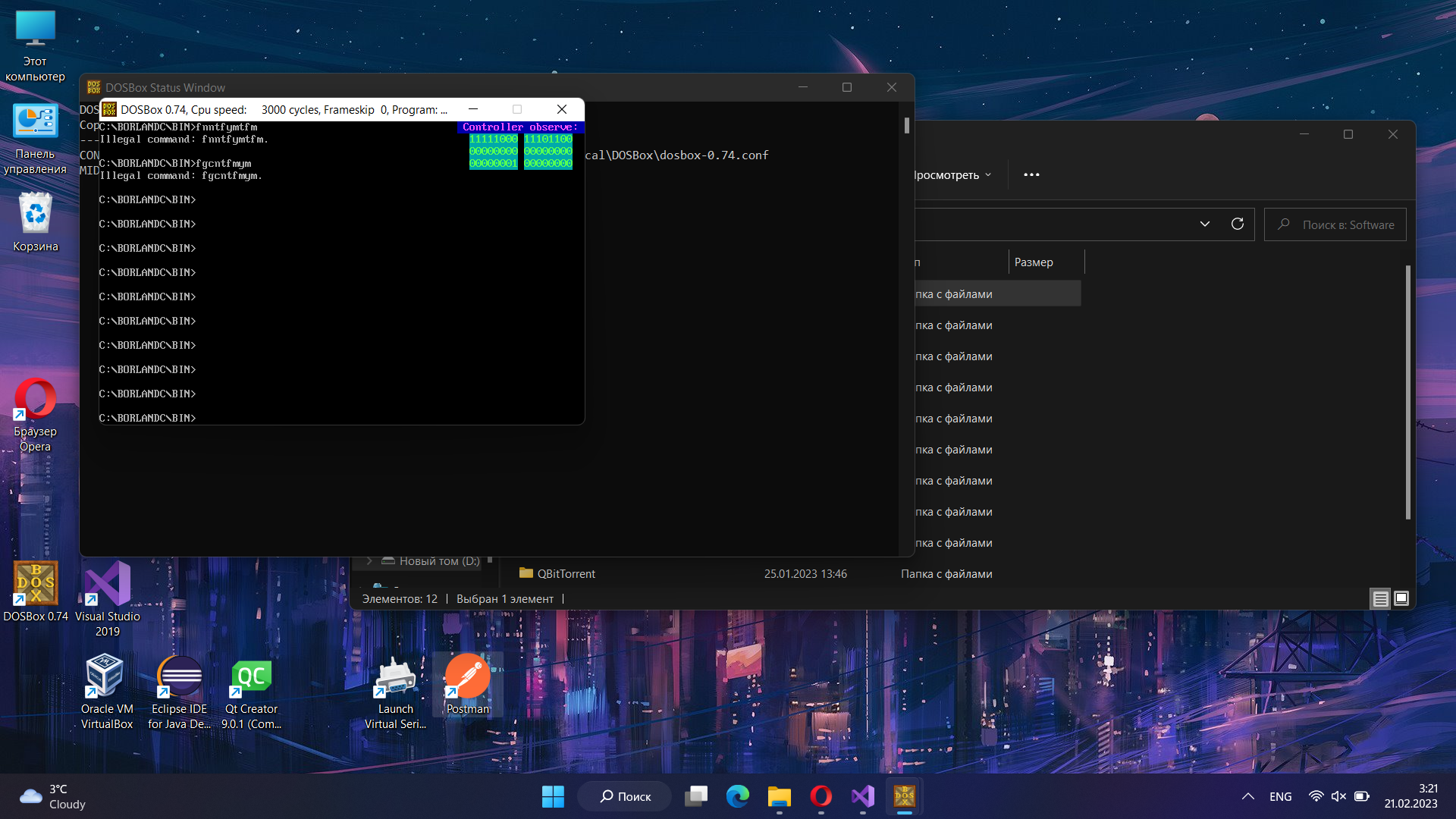
\_dos\_freemem(\*fp);

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

return 0;

}

**Тест**



**Заключение**

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