Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра электронных вычислительных машин

Лабораторная работа №2

«Программирование контроллера прерываний»

Вариант 11

Проверил: Выполнил:  
к. т. н., доцент студент гр. 150501

Одинец Д. Н. Климович А. Н.

Минск, 2023

**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.

2. 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.

3 КОД ПРОГРАММЫ

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

#include<dos.h>

#define ATTRIBUTE\_1 0x6E //yellow color

#define ATTRIBUTE\_2 0x1E //blur color

#define HEADER\_X 130 //position of any header on x

#define DIGITS\_X 142 //position of bits on x

void changeIDT(); //changing of interruption vector table

void initOfControllers(); //initialization

void printToScreen(char\*, int, int, unsigned char); //output on display

void byteToString(unsigned char temp, char\* str); //translator byte to string

void update(); //infinite update

//IRQ0-7

void interrupt(\*old\_int8)(...); // IRQ 0 - таймер

void interrupt(\*old\_int9)(...); // IRQ 1 - клавиатура

void interrupt(\*old\_int10)(...); // IRQ 2 - каскадное переключение второго контроллера

void interrupt(\*old\_int11)(...); // IRQ 3 - последовательный порт COM 2

void interrupt(\*old\_int12)(...); // IRQ 4 - последовательный порт COM 1

void interrupt(\*old\_int13)(...); // IRQ 5 - параллельный порт LPT 2

void interrupt(\*old\_int14)(...); // IRQ 6 - контроллер флоппи-дисковод

void interrupt(\*old\_int15)(...); // IRQ 7 - параллельный порт LPT 1

//IRQ8-15

void interrupt(\*old\_int70)(...); // IRQ 8 - часы реального времени

void interrupt(\*old\_int71)(...); // IRQ 9 - свободный

void interrupt(\*old\_int72)(...); // IRQ 10 - контроллер видеоадаптера

void interrupt(\*old\_int73)(...); // IRQ 11 - свободный

void interrupt(\*old\_int74)(...); // IRQ 12 - мышь PS/2

void interrupt(\*old\_int75)(...); // IRQ 13 - математический сопроцессор

void interrupt(\*old\_int76)(...); // IRQ 14 - первый контроллер жёсткого диска

void interrupt(\*old\_int77)(...); // IRQ 15 - второй контроллер жёсткого диска

// new interrupt handlers

void interrupt new\_int8(...) { update(); (\*old\_int8)(); }

void interrupt new\_int9(...) { update(); (\*old\_int9)(); }

void interrupt new\_int10(...) { update(); (\*old\_int10)(); }

void interrupt new\_int11(...) { update(); (\*old\_int11)(); }

void interrupt new\_int12(...) { update(); (\*old\_int12)(); }

void interrupt new\_int13(...) { update(); (\*old\_int13)(); }

void interrupt new\_int14(...) { update(); (\*old\_int14)(); }

void interrupt new\_int15(...) { update(); (\*old\_int15)(); }

void interrupt new\_int70(...) { update(); (\*old\_int70)(); }

void interrupt new\_int71(...) { update(); (\*old\_int71)(); }

void interrupt new\_int72(...) { update(); (\*old\_int72)(); }

void interrupt new\_int73(...) { update(); (\*old\_int73)(); }

void interrupt new\_int74(...) { update(); (\*old\_int74)(); }

void interrupt new\_int75(...) { update(); (\*old\_int75)(); }

void interrupt new\_int76(...) { update(); (\*old\_int76)(); }

void interrupt new\_int77(...) { update(); (\*old\_int77)(); }

//===========================================================================

//========================BEGINING OF PROGRAM================================

//===========================================================================

void main()

{

changeIDT();

initOfControllers();

unsigned far\* fp; //объявляем указатель

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

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

\_dos\_freemem(\*fp); //освобождение памяти под них

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

//первый параметр – код завершения,

//второй – объём памяти, который должен быть зарезервирован

//для программы после её завершения

}

void changeIDT()

{

old\_int8 = getvect(0x8); // IRQ0 прерывание таймера, возникает 18,2 раза/с.

old\_int9 = getvect(0x9); // IRQ1 прерывание от клавиатуры.

old\_int10 = getvect(0xA); // IRQ2 используется для каскадирования   
 // аппаратных прерываний

old\_int11 = getvect(0xB); // IRQ3 прерывание асинхронного порта COM2.

old\_int12 = getvect(0xC); // IRQ4 прерывание асинхронного порта COM1.

old\_int13 = getvect(0xD); // IRQ5 прерывание от контроллера жесткого диска   
 // для XT.

old\_int14 = getvect(0xE); // IRQ6 прерывание генерируется контроллером флоппи   
 // диска после завершения операции

old\_int15 = getvect(0xF); // IRQ7 прерывание принтера.

old\_int70 = getvect(0x70); // IRQ8 прерывание от часов реального времени.

old\_int71 = getvect(0x71); // IRQ9 прерывание от контроллера EGA.

old\_int72 = getvect(0x72); // IRQ10 зарезервировано.

old\_int73 = getvect(0x73); // IRQ11 зарезервировано.

old\_int74 = getvect(0x74); // IRQ12 зарезервировано.

old\_int75 = getvect(0x75); // IRQ13 прерывание от математического сопроцессора.

old\_int76 = getvect(0x76); // IRQ14 прерывание от контроллера жесткого диска.

old\_int77 = getvect(0x77); // IRQ15 зарезервировано.

//переопределение IRQ 0-7 (вариант 11)

setvect(0xB0, new\_int8);

setvect(0xB1, new\_int9);

setvect(0xB2, new\_int10);

setvect(0xB3, new\_int11);

setvect(0xB4, new\_int12);

setvect(0xB5, new\_int13);

setvect(0xB6, new\_int14);

setvect(0xB7, new\_int15);

//переопределение IRQ 8-15 (вариант 11)

setvect(0x08, new\_int70);

setvect(0x09, new\_int71);

setvect(0x0A, new\_int72);

setvect(0x0B, new\_int73);

setvect(0x0C, new\_int74);

setvect(0x0D, new\_int75);

setvect(0x0E, new\_int76);

setvect(0x0F, new\_int77);

}

void initOfControllers() // инициализация

{

unsigned char register\_of\_masks;

\_disable(); // Clear Interrupt Flag

// Ведущий контроллер

register\_of\_masks = inp(0x21); // запоминаем регистр масок

outp(0x20, 0x11); // ICW1 - инициализация ведущего контроллера

outp(0x21, 0xB0); // ICW2 - базовый вектор для ведущего (вариант 11)

outp(0x21, 0x04); // ICW3 - порт бит ведущего контроллера

outp(0x21, 0x01); // ICW4 - по умолчанию

outp(0x21, register\_of\_masks); // восстанавливаем регистр масок

// Ведомый контроллер

register\_of\_masks = inp(0xA1); // запоминаем регистр масок

outp(0xA0, 0x11); // ICW1 - инициализация ведомого контроллера

outp(0xA1, 0x08); // ICW2 - базовый вектор для ведомого (вариант 11)

outp(0xA1, 0x02); // ICW3 - кол-во подключенных портов к ведущему

outp(0xA1, 0x01); // ICW4 - по умолчанию

outp(0xa1, register\_of\_masks); // восстанавливаем регистр масок

\_enable(); // STI

}

// Байт в строку в двоичном виде

void byteToString(unsigned char register\_int, char\* register\_str)

{

register\_str[8] = 0;

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

{

register\_str[i] = '0' + register\_int % 2;

register\_int /= 2;

}

}

void printToScreen(char\* str, int x, int y, unsigned char attribute)

{

char far\* screen = (char far\*)0xb8000000; //получить видеобуфер

screen += x + 160 \* y; //смещение

for (int i = 0; str[i] != 0; ++i, ++screen)

{

\*screen = str[i];

screen++;

\*screen = attribute;

}

}

void update()

{

unsigned char ISR\_master, ISR\_slave; // Interrupt Service Register -   
 // Регистр обслуживаемых прерываний

unsigned char IRR\_master, IRR\_slave; // Interrupt Request Register -   
 // Регистр запросов на прерывания

unsigned char IMR\_master, IMR\_slave; // Interrupt Mask Register - Регистр масок

IMR\_master = inp(0x21); //получить регистр масок ведущего контроллера

IMR\_slave = inp(0xA1); //получить регистр масок ведомого контроллера

outp(0x20, 0x0A); //переключение на регистр запросов ведущего контроллера

IRR\_master = inp(0x20); //получить регистр запросов ведущего контроллера

outp(0xA0, 0x0A); //переключение на регистр запросов ведомого контроллера

IRR\_slave = inp(0xA0); //получить регистр запросов ведомого контроллера

outp(0x20, 0x0B); //переключение на регистр обслуживания ведущего контроллера

ISR\_master = inp(0x20);//получить регистр обслуживания ведущего контроллера

outp(0xA0, 0x0B); //переключение на регистр обслуживания ведомого контроллера

ISR\_slave = inp(0xA0); //получить регистр обслуживания ведомого контроллера

char str[9];

int string\_Y = 0;

printToScreen("====MASTER====", HEADER\_X, string\_Y++, ATTRIBUTE\_1);

printToScreen("ISR: ", HEADER\_X, string\_Y, ATTRIBUTE\_1);

byteToString(ISR\_master, str);

printToScreen(str, DIGITS\_X, string\_Y++, ATTRIBUTE\_1);

printToScreen("IRR: ", HEADER\_X, string\_Y, ATTRIBUTE\_1);

byteToString(IRR\_master, str);

printToScreen(str, DIGITS\_X, string\_Y++, ATTRIBUTE\_1);

printToScreen("IMR: ", HEADER\_X, string\_Y, ATTRIBUTE\_1);

byteToString(IMR\_master, str);

printToScreen(str, DIGITS\_X, string\_Y++, ATTRIBUTE\_1);

printToScreen("====SLAVE=====", HEADER\_X, string\_Y++, ATTRIBUTE\_2);

printToScreen("ISR: ", HEADER\_X, string\_Y, ATTRIBUTE\_2);

byteToString(ISR\_slave, str);

printToScreen(str, DIGITS\_X, string\_Y++, ATTRIBUTE\_2);

printToScreen("IRR: ", HEADER\_X, string\_Y, ATTRIBUTE\_2);

byteToString(IRR\_slave, str);

printToScreen(str, DIGITS\_X, string\_Y++, ATTRIBUTE\_2);

printToScreen("IMR: ", HEADER\_X, string\_Y, ATTRIBUTE\_2);

byteToString(IMR\_slave, str);

printToScreen(str, DIGITS\_X, string\_Y, ATTRIBUTE\_2);

}

4 ТЕСТИРОВАНИЕ

На рисунках 4.1 и 4.2 приведены скриншоты результата работы программы.

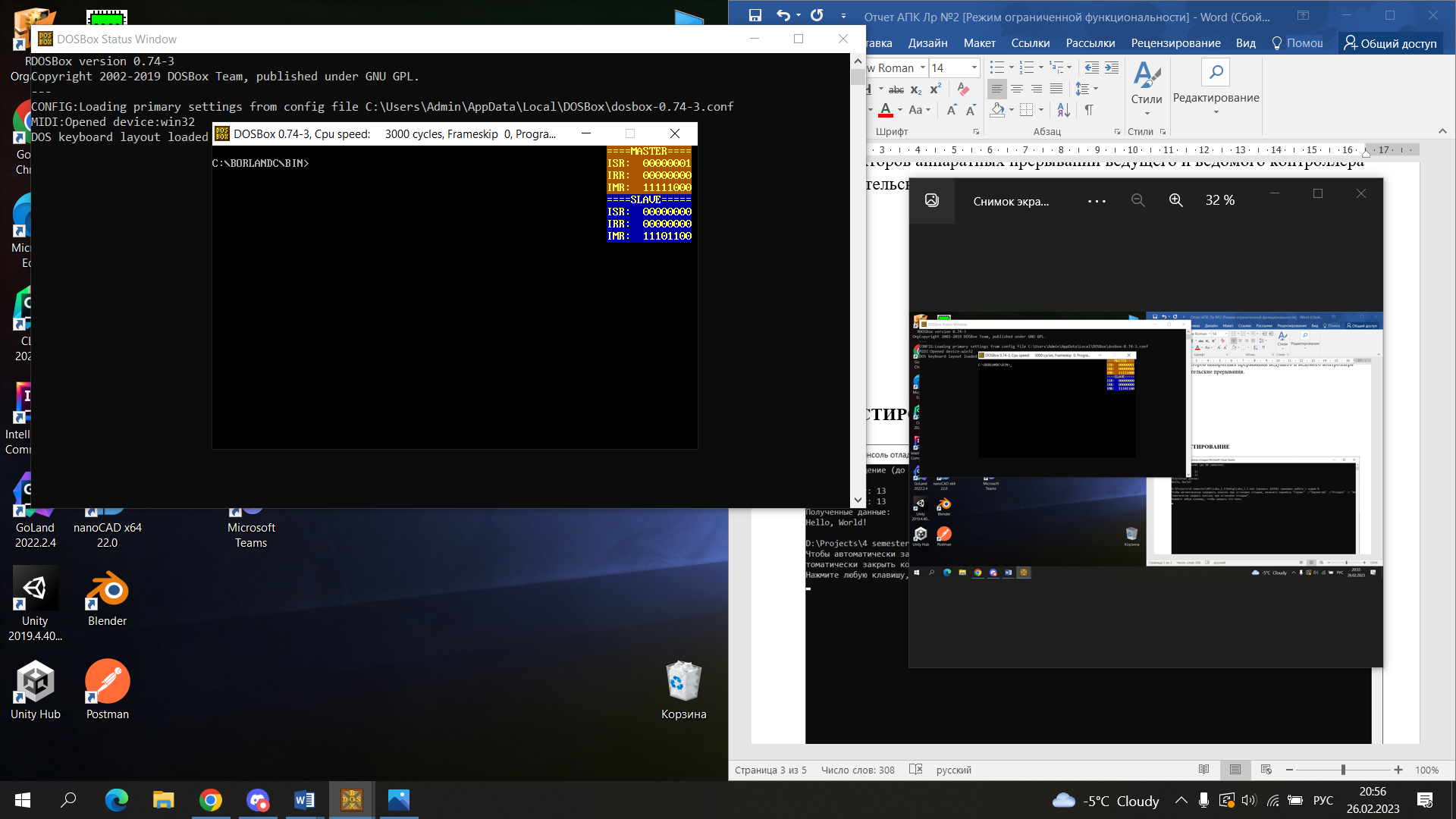


Рисунок 4.1 – Результат работы программы при запуске

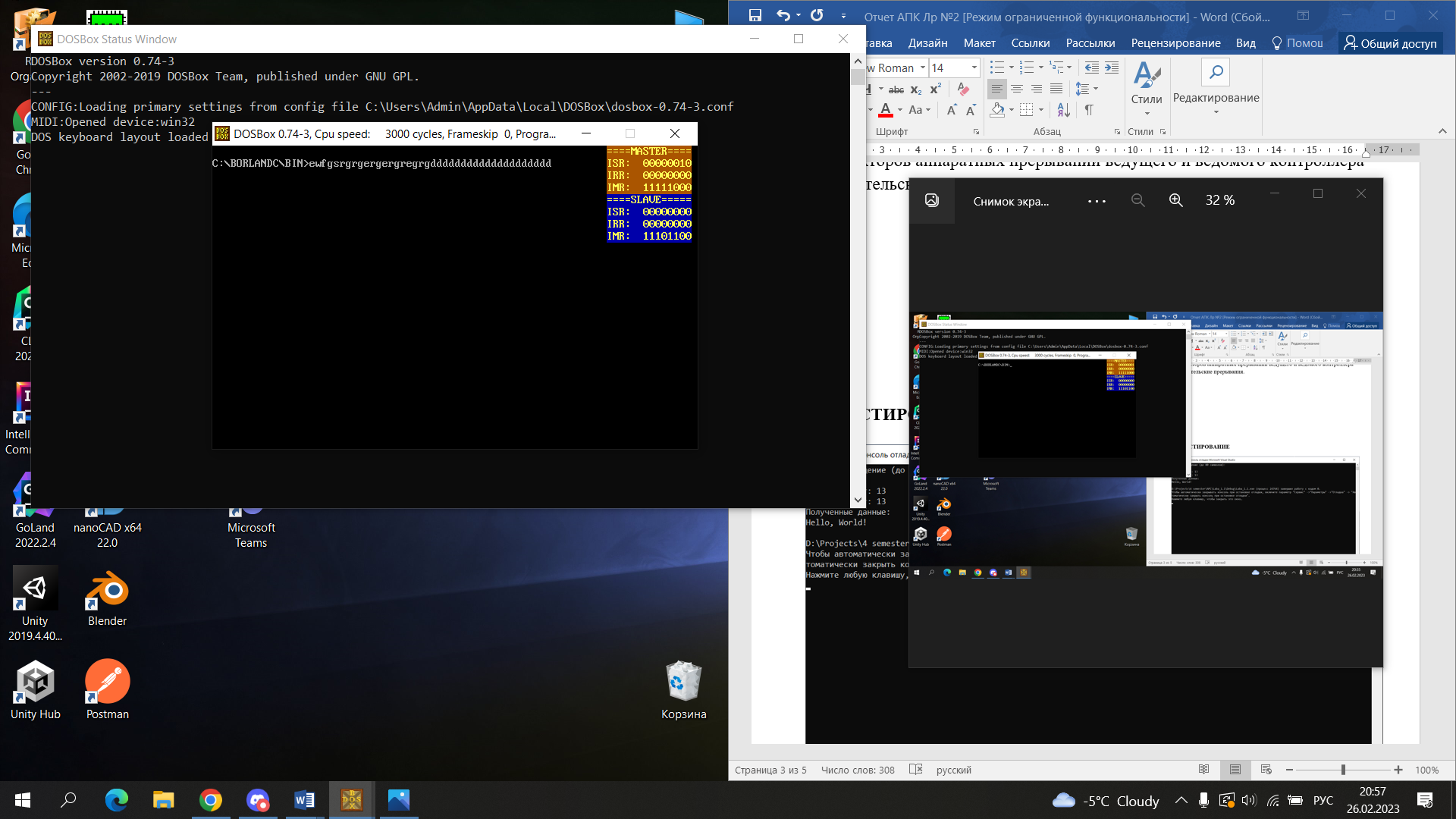


Рисунок 4.2 – Результат работы программы при использовании клавиатуры

5 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. Для эмуляции DOS использовался DOSBox 0.74-3 на хосте 64-ех разрядной Windows 10.