Министерство образования Республики Беларусь

Учреждение образования Белорусский государственный университет информатики и радиоэлектроники

Кафедра ЭВМ

Отчёт по лабораторной работе №2

“Программирования контроллера прерываний”

Вариант 6

Проверил: Выполнил:

к.т.н., доцент студент гр.150504

Одинец Дмитрий Николаевич Горбачевский К.В.

Минск 2020

**Задача**

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.

**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.

**Листинг программы**

#include <dos.h>

#include <time.h>

#include <stdio.h>

#include <stdlib.h>

#define MASTER\_BASE\_VECTOR 0x88

#define SLAVE\_BASE\_VECTOR 0x08

void print\_byte(char far\* screen, unsigned char byte) {

int bit;

int i;

for (i = 0; i < 8; ++i) {

bit = byte % 2;

byte = byte >> 1;

\*screen = '0' + bit; // выводим на текущую позицию видеоадаптера один бит маски

screen += 2;

}

}

void print(void) {

char far\* screen = (char far\*)MK\_FP(0xB800, 0); // возвращает дальний указатель по заданному сегменту (сегмент видеадаптера, смещение - 0)

// Master mask

print\_byte(screen, inp(0x21)); // inp возвращает значение по заданному адресу, регистр масок ведущего контроллера

screen += 18; // сдвиг для вывода в консоли

// Slave mask

print\_byte(screen, inp(0xA1)); // регистр масок для ведомого контроллера

screen += 142; // переход на следующую строку в консоли

// Master request

outp(0x20, 0x0A); // функции записывают данные непосредственно в порт ввода-вывода

print\_byte(screen, inp(0x20)); // регистр запроса для ведущего контроллера

screen += 18;

// Slave request

outp(0xA0, 0x0A); // регистр запроса для ведомого контроллера

print\_byte(screen, inp(0xA0));

screen += 142; // переход на следующую строку в консоли

// Master service

outp(0x20, 0x0B);

print\_byte(screen, inp(0x20));

screen += 18;

// Master service

outp(0xA0, 0x0B);

print\_byte(screen, inp(0xA0));

}

// Master обработчики прерываний

void interrupt (\*old\_irq0\_handler)(void); // Системные часы, прерывание интервального таймера, возникает 18,2 раза в секунду.

void interrupt (\*old\_irq1\_handler)(void); // прерывание от клавиатуры. Генерируется при нажатии и при отжатии клавиши. Используется для чтения данных с клавиатуры.

void interrupt (\*old\_irq2\_handler)(void); // используется для каскадирования аппаратных прерываний в машинах класса AT.

void interrupt (\*old\_irq3\_handler)(void); // прерывание асинхронного порта COM2

void interrupt (\*old\_irq4\_handler)(void); // прерывание асинхронного порта COM1

void interrupt (\*old\_irq5\_handler)(void); // прерывание от контроллера жесткого диска (только для компьютеров IBM PC/XT)

void interrupt (\*old\_irq6\_handler)(void); // прерывание генерируется контроллером НГМД после завершения операции ввода/вывода

void interrupt (\*old\_irq7\_handler)(void); // Генерируется, когда подключенный к адаптеру принтер готов к выполнению очередной операции. Обычно не используется

void interrupt new\_irq0\_handler(void) { print(); old\_irq0\_handler(); } // переопределение прерываний

void interrupt new\_irq1\_handler(void) { print(); old\_irq1\_handler(); }

void interrupt new\_irq2\_handler(void) { print(); old\_irq2\_handler(); }

void interrupt new\_irq3\_handler(void) { print(); old\_irq3\_handler(); }

void interrupt new\_irq4\_handler(void) { print(); old\_irq4\_handler(); }

void interrupt new\_irq5\_handler(void) { print(); old\_irq5\_handler(); }

void interrupt new\_irq6\_handler(void) { print(); old\_irq6\_handler(); }

void interrupt new\_irq7\_handler(void) { print(); old\_irq7\_handler(); }

// Slave обработчики прерываний

void interrupt (\*old\_irq8\_handler)(void); // прерывание от часов реального времени

void interrupt (\*old\_irq9\_handler)(void); // прерывание от контроллера EGA

void interrupt (\*old\_irq10\_handler)(void); // зарезервировано

void interrupt (\*old\_irq11\_handler)(void); // зарезервировано

void interrupt (\*old\_irq12\_handler)(void); // зарезервировано

void interrupt (\*old\_irq13\_handler)(void); // прерывание от арифметического сопроцессора

void interrupt (\*old\_irq14\_handler)(void); // прерывание от контроллера жесткого диска

void interrupt (\*old\_irq15\_handler)(void); // зарезервировано

void interrupt new\_irq8\_handler(void) { print(); old\_irq8\_handler(); }

void interrupt new\_irq9\_handler(void) { print(); old\_irq9\_handler(); }

void interrupt new\_irq10\_handler(void) { print(); old\_irq10\_handler(); }

void interrupt new\_irq11\_handler(void) { print(); old\_irq11\_handler(); }

void interrupt new\_irq12\_handler(void) { print(); old\_irq12\_handler(); }

void interrupt new\_irq13\_handler(void) { print(); old\_irq13\_handler(); }

void interrupt new\_irq14\_handler(void) { print(); old\_irq14\_handler(); }

void interrupt new\_irq15\_handler(void) { print(); old\_irq15\_handler(); }

//-----------------------------------------------------------------------------

void init\_new\_handlers(void) {

old\_irq0\_handler = getvect(0x08); // принимаем старый(дефолтный) обработчик прерывания

setvect(MASTER\_BASE\_VECTOR, new\_irq0\_handler); // устанавливаем по этому вектору переопределенный обработчик

old\_irq1\_handler = getvect(0x09);

setvect(MASTER\_BASE\_VECTOR + 1, new\_irq1\_handler);

old\_irq2\_handler = getvect(0x0A);

setvect(MASTER\_BASE\_VECTOR + 2, new\_irq2\_handler);

old\_irq3\_handler = getvect(0x0B);

setvect(MASTER\_BASE\_VECTOR + 3, new\_irq3\_handler);

old\_irq4\_handler = getvect(0x0C);

setvect(MASTER\_BASE\_VECTOR + 4, new\_irq4\_handler);

old\_irq5\_handler = getvect(0x0D);

setvect(MASTER\_BASE\_VECTOR + 5, new\_irq5\_handler);

old\_irq6\_handler = getvect(0x0E);

setvect(MASTER\_BASE\_VECTOR + 6, new\_irq6\_handler);

old\_irq7\_handler = getvect(0x0F);

setvect(MASTER\_BASE\_VECTOR + 7, new\_irq7\_handler);

old\_irq8\_handler = getvect(0x70);

setvect(SLAVE\_BASE\_VECTOR, new\_irq8\_handler);

old\_irq9\_handler = getvect(0x71);

setvect(SLAVE\_BASE\_VECTOR + 1, new\_irq9\_handler);

old\_irq10\_handler = getvect(0x72);

setvect(SLAVE\_BASE\_VECTOR + 2, new\_irq10\_handler);

old\_irq11\_handler = getvect(0x73);

setvect(SLAVE\_BASE\_VECTOR + 3, new\_irq11\_handler);

old\_irq12\_handler = getvect(0x74);

setvect(SLAVE\_BASE\_VECTOR + 4, new\_irq12\_handler);

old\_irq13\_handler = getvect(0x75);

setvect(SLAVE\_BASE\_VECTOR + 5, new\_irq13\_handler);

old\_irq14\_handler = getvect(0x76);

setvect(SLAVE\_BASE\_VECTOR + 6, new\_irq14\_handler);

old\_irq15\_handler = getvect(0x77);

setvect(SLAVE\_BASE\_VECTOR + 7, new\_irq15\_handler);

// Выключаем контроллер

\_disable();

// Инициализируем новые обработчики. После завершения инициализации контроллеры начинают обрабатывать поступающие запросы прерываний.

// outp функция записывают данные непосредственно в порт ввода-вывода

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

outp(0x21, 0x88); // ICW2

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

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

// Initialize slave

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

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

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

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

// Стартуем котроллер

\_enable();

}

int main(void) {

unsigned far\* fp;

init\_new\_handlers();

system("cls");

puts(" - MASK");

puts(" - REQUEST");

puts(" - SERVICE");

puts("MASTER SLAVE");

// Делаем программу резидентной

// Резидентная программа активизируется каждый раз при возникновении прерывания, вектор которого эта программа изменила на адрес одной из своих процедур.

FP\_SEG(fp) = \_psp; // возвращает сегмент дальнего указателя fp.

FP\_OFF(fp) = 0x2c; // возвращает смещение указателя

\_dos\_freemem(\*fp);

\_dos\_keep(0, (\_DS - \_CS) + (\_SP / 16) + 1); // первый параметр - условие выхода из резидентной программы, второй парметр - кол-во выделенной памяти

// размер памяти в параграфах, сегмент стека в байтах, поэтому его мы делим на 16 и + 1

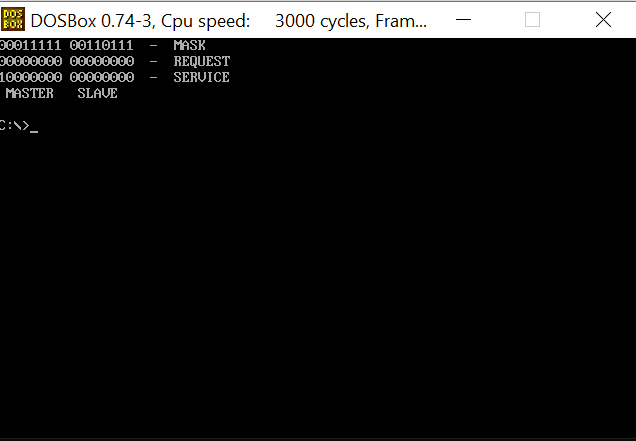
return EXIT\_SUCCESS;

}

|  |  |
| --- | --- |
| **Chip - Purpose** | [**I/Oport**](https://wiki.osdev.org/I/O_ports) |
| Master PIC - Command | 0x0020 |
| Master PIC - Data | 0x0021 |
| Slave PIC - Command | 0x00A0 |
| Slave PIC - Data | 0x00A1 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Chip** | **Interrupt numbers (IRQ)** | **Vector offset** | **Interrupt Numbers** |
| Master PIC | 0 to 7 | 0x08 | 0x08 to 0x0F |
| Slave PIC | 8 to 15 | 0x70 | 0x70 to 0x77 |

**Скриншот консоли**



**Заключение**

В данной лабораторной работе разработана программа для работы с последовательным COM портом. Данные заносятся в COM1 и читаются из COM2.