Белорусский Государственный Университет Информатики и Радиоэлектроники

Кафедра ЭВМ

Контрольная работа № 1

Тема: «Подсистема прерываний»

Выполнил:

студент гр.250541

Власов Р.Е.

Проверил:

Одинец Д.Н.

Минск 2024

#### 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.

1. Осуществлять переход на стандартные обработчики аппаратных прерываний, для обеспечения нормальной работы компьютера.

#### Алгоритм решения задачи.

#### 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. В обработчике прерывания от клавиатуры также проверяется нажатие клавиши Esc – в случае нажатия восстанавливаются сохранённые обработчики событий.

#### Листинг программы.

int attribute = 0x00; // Initial color

typedef struct

{

unsigned char symb;

unsigned char attr;

} VIDEO;

void get\_reg(); // Get data from registers

void print(int offset, int val); // Fill the video image pointer

// Declarations of pointers to old interrupt handlers

// IRQ0-7

void interrupt (\*oldInt8)(); // IRQ 0 - timer

void interrupt (\*oldInt9)(); // IRQ 1 - keyboard

void interrupt (\*oldIntA)(); // IRQ 2 - cascade switching of the second controller

void interrupt (\*oldIntB)(); // IRQ 3 - serial port COM 2

void interrupt (\*oldIntC)(); // IRQ 4 - serial port COM 1

void interrupt (\*oldIntD)(); // IRQ 5 - parallel port LPT 2

void interrupt (\*oldIntE)(); // IRQ 6 - floppy disk controller

void interrupt (\*oldIntF)(); // IRQ 7 - parallel port LPT 1

// IRQ8-15

void interrupt (\*oldInt70)(); // IRQ 8 - real-time clock

void interrupt (\*oldInt71)(); // IRQ 9 - free

void interrupt (\*oldInt72)(); // IRQ 10 - video adapter controller

void interrupt (\*oldInt73)(); // IRQ 11 - free

void interrupt (\*oldInt74)(); // IRQ 12 - PS/2 mouse

void interrupt (\*oldInt75)(); // IRQ 13 - math coprocessor

void interrupt (\*oldInt76)(); // IRQ 14 - first hard disk controller

void interrupt (\*oldInt77)(); // IRQ 15 - second hard disk controller

// New interrupt handlers for IRQ0-7 (master controller)

void interrupt newInt8() { get\_reg(); if (oldInt8) (\*oldInt8)(); outp(0x20, 0x20); } // IRQ0

void interrupt newInt9() { attribute++; get\_reg(); if (oldInt9) (\*oldInt9)(); outp(0x20, 0x20); } // IRQ1 (keyboard)

void interrupt newIntA() { get\_reg(); if (oldIntA) (\*oldIntA)(); outp(0x20, 0x20); } // IRQ2

void interrupt newIntB() { get\_reg(); if (oldIntB) (\*oldIntB)(); outp(0x20, 0x20); } // IRQ3

void interrupt newIntC() { get\_reg(); if (oldIntC) (\*oldIntC)(); outp(0x20, 0x20); } // IRQ4

void interrupt newIntD() { get\_reg(); if (oldIntD) (\*oldIntD)(); outp(0x20, 0x20); } // IRQ5

void interrupt newIntE() { get\_reg(); if (oldIntE) (\*oldIntE)(); outp(0x20, 0x20); } // IRQ6

void interrupt newIntF() { get\_reg(); if (oldIntF) (\*oldIntF)(); outp(0x20, 0x20); } // IRQ7

// New interrupt handlers for IRQ8-15 (slave controller)

void interrupt newInt70() { get\_reg(); if (oldInt70) (\*oldInt70)(); outp(0xA0, 0x20); outp(0x20, 0x20); } // IRQ8

void interrupt newInt71() { get\_reg(); if (oldInt71) (\*oldInt71)(); outp(0xA0, 0x20); outp(0x20, 0x20); } // IRQ9

void interrupt newInt72() { get\_reg(); if (oldInt72) (\*oldInt72)(); outp(0xA0, 0x20); outp(0x20, 0x20); } // IRQ10

void interrupt newInt73() { get\_reg(); if (oldInt73) (\*oldInt73)(); outp(0xA0, 0x20); outp(0x20, 0x20); } // IRQ11

void interrupt newInt74() { get\_reg(); if (oldInt74) (\*oldInt74)(); outp(0xA0, 0x20); outp(0x20, 0x20); } // IRQ12

void interrupt newInt75() { get\_reg(); if (oldInt75) (\*oldInt75)(); outp(0xA0, 0x20); outp(0x20, 0x20); } // IRQ13

void interrupt newInt76() { get\_reg(); if (oldInt76) (\*oldInt76)(); outp(0xA0, 0x20); outp(0x20, 0x20); } // IRQ14

void interrupt newInt77() { get\_reg(); if (oldInt77) (\*oldInt77)(); outp(0xA0, 0x20); outp(0x20, 0x20); } // IRQ15

// Function to display the binary representation of a value on the screen

void print(int offset, int val)

{

char temp; // Current bit

int i;

VIDEO far\* screen = (VIDEO far\*)MK\_FP(0xB800, 0); // Get the address of the video buffer

screen += offset;

for (i = 7; i >= 0; i--) // Iterate through the bits

{

temp = val % 2; // Get the last bit

val /= 2;

screen->symb = temp + '0'; // Convert the bit to character '0' or '1'

screen->attr = attribute; // Set the current color

screen++; // Move to the next character

}

}

// Function to get data from the interrupt controller registers

void get\_reg()

{

int reg;

// Master controller

reg = inp(0x21); // Get the mask register of the master controller

print(0, reg);

outp(0x20, 0x0B); // Switch to the service register of the master controller

reg = inp(0x20); // Get the service register of the master controller

print(9, reg);

outp(0x20, 0x0A); // Switch to the request register of the master controller

reg = inp(0x20); // Get the request register of the master controller

print(18, reg);

// Slave controller

reg = inp(0xA1); // Get the mask register of the slave controller

print(80, reg);

outp(0xA0, 0x0B); // Switch to the service register of the slave controller

reg = inp(0xA0); // Get the service register of the slave controller

print(80 + 9, reg);

outp(0xA0, 0x0A); // Switch to the request register of the slave controller

reg = inp(0xA0); // Get the request register of the slave controller

print(80 + 18, reg);

}

// Function to initialize interrupt controllers and remap vectors

void init()

{

// Save old interrupt vectors for IRQ0-7 (master controller)

oldInt8 = getvect(0x08); // IRQ 0

oldInt9 = getvect(0x09); // IRQ 1

oldIntA = getvect(0x0A); // IRQ 2

oldIntB = getvect(0x0B); // IRQ 3

oldIntC = getvect(0x0C); // IRQ 4

oldIntD = getvect(0x0D); // IRQ 5

oldIntE = getvect(0x0E); // IRQ 6

oldIntF = getvect(0x0F); // IRQ 7

// Save old interrupt vectors for IRQ8-15 (slave controller)

oldInt70 = getvect(0x70); // IRQ 8

oldInt71 = getvect(0x71); // IRQ 9

oldInt72 = getvect(0x72); // IRQ 10

oldInt73 = getvect(0x73); // IRQ 11

oldInt74 = getvect(0x74); // IRQ 12

oldInt75 = getvect(0x75); // IRQ 13

oldInt76 = getvect(0x76); // IRQ 14

oldInt77 = getvect(0x77); // IRQ 15

// Set new interrupt vectors for IRQ0-7 (master controller)

setvect(0x78, newInt8);

setvect(0x79, newInt9);

setvect(0x7A, newIntA);

setvect(0x7B, newIntB);

setvect(0x7C, newIntC);

setvect(0x7D, newIntD);

setvect(0x7E, newIntE);

setvect(0x7F, newIntF);

// Set new interrupt vectors for IRQ8-15 (slave controller)

setvect(0x08, newInt70);

setvect(0x09, newInt71);

setvect(0x0A, newInt72);

setvect(0x0B, newInt73);

setvect(0x0C, newInt74);

setvect(0x0D, newInt75);

setvect(0x0E, newInt76);

setvect(0x0F, newInt77);

\_disable(); // Disable interrupts

// Initialize master controller

outp(0x20, 0x11); // ICW1: Begin initialization

outp(0x21, 0x78); // ICW2: New base vector for master controller (0x78)

outp(0x21, 0x04); // ICW3: Connection line for slave controller (IRQ2)

outp(0x21, 0x01); // ICW4: 8086/88 mode

// Initialize slave controller

outp(0xA0, 0x11); // ICW1: Begin initialization

outp(0xA1, 0x08); // ICW2: New base vector for slave controller (0x08)

outp(0xA1, 0x02); // ICW3: Connection to IRQ2 of master controller

outp(0xA1, 0x01); // ICW4: 8086/88 mode

\_enable(); // Enable interrupts

}

int main()

{

unsigned far \*fp; // Declare a pointer

init(); // Initialize controllers and remap vectors

// Free the memory occupied by the program

FP\_SEG(fp) = \_psp; // Get the PSP segment

FP\_OFF(fp) = 0x2C; // Offset to the environment block

\_dos\_freemem(\*fp); // Free memory occupied by the environment block

// Make the program resident

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

return 0;

}

#### Результаты работы программы.

#### Программа является резидентной. После запуска в верхней части экрана выводятся в двоичной форме регистры запросов на прерывания, обслуживаемых прерываний, масок для ведущего и ведомого контроллеров. По нажатию клавиши ESC восстанавливаются стандартные обработчики прерываний.