Computer Architecture

Lecture IV

Selected external x86 microprocessor elements

Iterrupts

- Iterrupts are used to communicate a computer system with external devices such as a keybard, a printer, system timers, hard and floppy disk controllers, sound cards or graphics cards that need immediate handling.

- Iterrupts were introduced in processor's architecture to avoid wasting processor time for so called polling loops i.e repeatedly sampling the status of all external devices connected to a computer system in order to check weather they need handling.

Keyboard press causing interrupt signal.

Main program

```
mov dx,
sub cx, cx
mov cl, StrLen
mov si, OFFSET Buffer
cmp Buffer, 10
jne StrLoop
inc si
dec cx
jb  cx
cmp BYTE PTR [si], 10
ja
inc si
loop StrLoop
stc
ret
clc
ret
```

Iterrupt

```
pusha
cmp ax, 0
jne NumbOverflow
mov si, OFFSET MinStr
mov di, OFFSET DestStr
 •
 •
 •
mov cx, 7
cld
rep movsb
popa
iret
```

Iterrupt handler

End of Iterrupt handler:
restore processor state
and return to the next instruction
of the interrupted program.

Intel 8259A Programmable Interrupt Controller

Intel 8259A ProgrammableInterrupt Controller is a circuit which controls iterrupt handling.

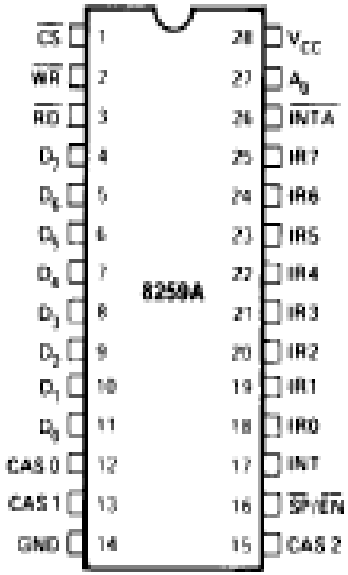


Fig. 1. 8259A Pinout

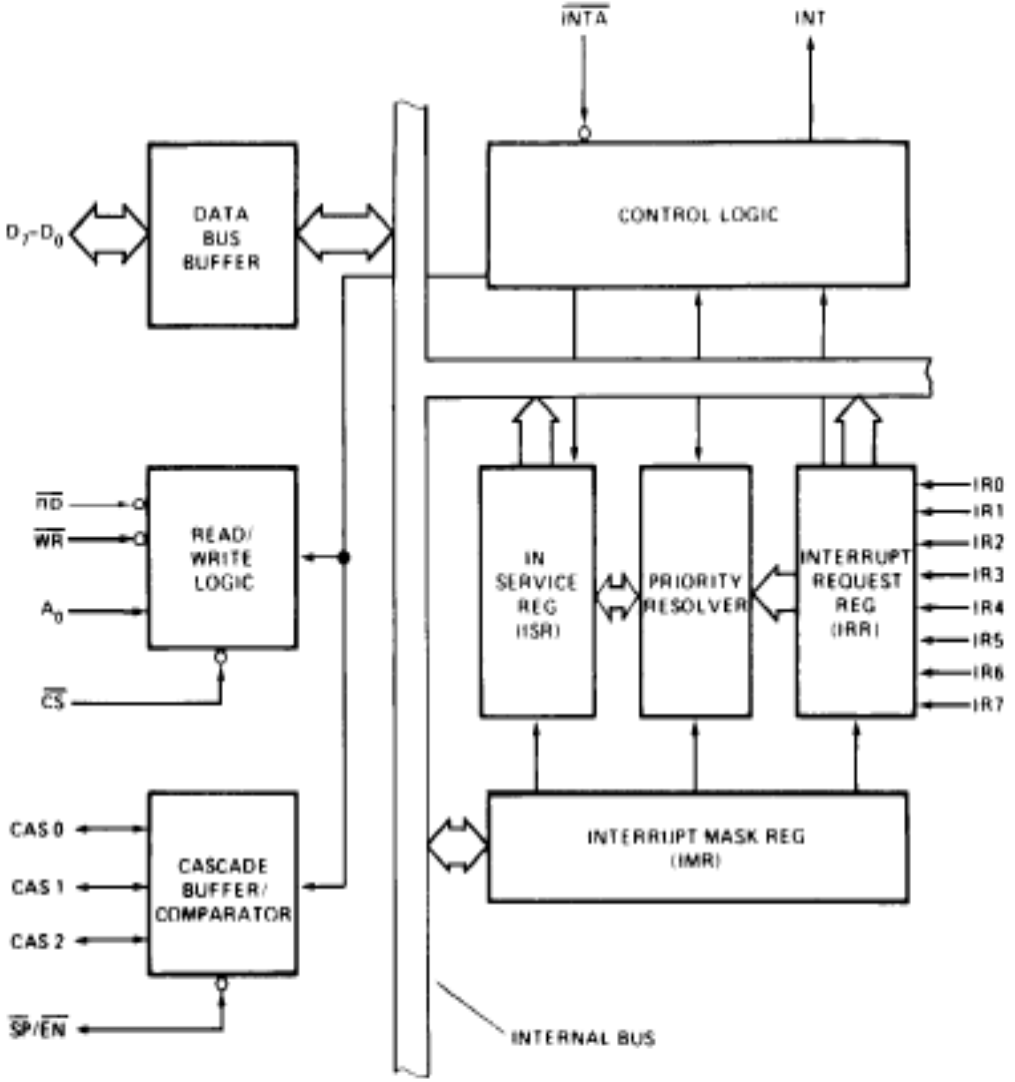


Fig. 2. 8259A Block Diagram

Intel 8259A Programmable Interrupt Controller functional blocks

INTERRUPT REQUEST REGISTER (IRR)
- is used to store all the interrupt levels which are requesting service

IN-SERVICE REGISTER (ISR)
- it is used to store all the interrupt levels which are being serviced.

INTERRUPT MASK REGISTER (IMR)
it is used to store the bits which mask the interrupt lines to be masked.

PRIORITY RESOLVER
This logic block determines the priorites of the interrupts.

READ/WRITE CONTROL LOGIC
The function of this block is to accept commands from the CPU,
it also allows the status of the 8259A to be transferred onto the Data Bus.

DATA BUS BUFFER
Bidirectional 8-bit buffer used to interface the 8259A to the system Data Bus.
Control words and status information are transferred through this buffer.

Intel 8259A Programmable Interrupt Controller functional blocks

THE CASCADE BUFFER/COMPARATOR
This function block stores and compares the IDs of all 8259A's used in the system. The associated three I/O pins (CAS0-2) are outputs when the 8259A is used as a master and are inputs when the 8259A is used as a slave.
As a master, the 8259A sends the ID of the interrupting slave device onto the CAS0±2 lines. The slave thus selected will send its preprogrammed subroutine address onto the Data Bus during the next one or two consecutive INTA pulses.

INT (INTERRUPT)
This output goes directly to the CPU interrupt input.

~INTA (INTERRUPT ACKNOWLEDGE)
INTA pulses will cause the 8259A to release interrupt soubroutine address onto the data bus.

IR0 – IR7
Interrupt request lines.

Intel 8259A Programmable Interrupt Controller functional blocks

~CS (CHIP SELECT)
A LOW on this input enables the 8259A. No reading or writing of the chip will occur unless the device is selected.

~WR (WRITE)
A LOW on this input enables the CPU to write control words to the 8259A.

~RD (READ)
A LOW on this input enables the 8259A to send the status of the Interrupt Request Register (IRR), In Service Register (ISR), the Interrupt Mask Register (IMR) or the Interrupt level onto the Data Bus.

~A0
This input signal is used in conjunction with WR and RD signals to write commands into the various command registers, as well as reading the various status registers of the chip. This line can be tied directly to one of the address lines.

~SP – if LOW a chip works in a slave mode when cascade of 8259A are used.

Intel 8259A Programmable Interrupt Controller operation

1. One or more of the INTERRUPT REQUEST lines (IR7±0) are raised high, setting the corresponding IRR bit(s).
2. The 8259A evaluates these requests, and sends an INT to the CPU, if appropriate.
3. The CPU acknowledges the INT and responds with an INTA pulse.
4. Upon receiving an INTA from the CPU the highest priority ISR bit is set and the corresponding IRR bit is reset. The 8259A will also release a CALL instruction code onto the 8-bit Data Bus through its D7±0 pins.
5. This CALL instruction will initiate two more INTA pulses to be sent to the 8259A from the CPU group.
6. These two INTA pulses allow the 8259A to release its preprogrammed subroutine address onto the Data Bus. The lower 8-bit address is released at the first INTA pulse and the higher 8-bit address is released at the second INTA pulse.
7. This completes the 3-byte CALL instruction released by the 8259A. In the AEOI mode the ISR bit is reset at the end of the third INTA pulse. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt sequence.

Intel 8259A Programmable Interrupt Controller typical system setup

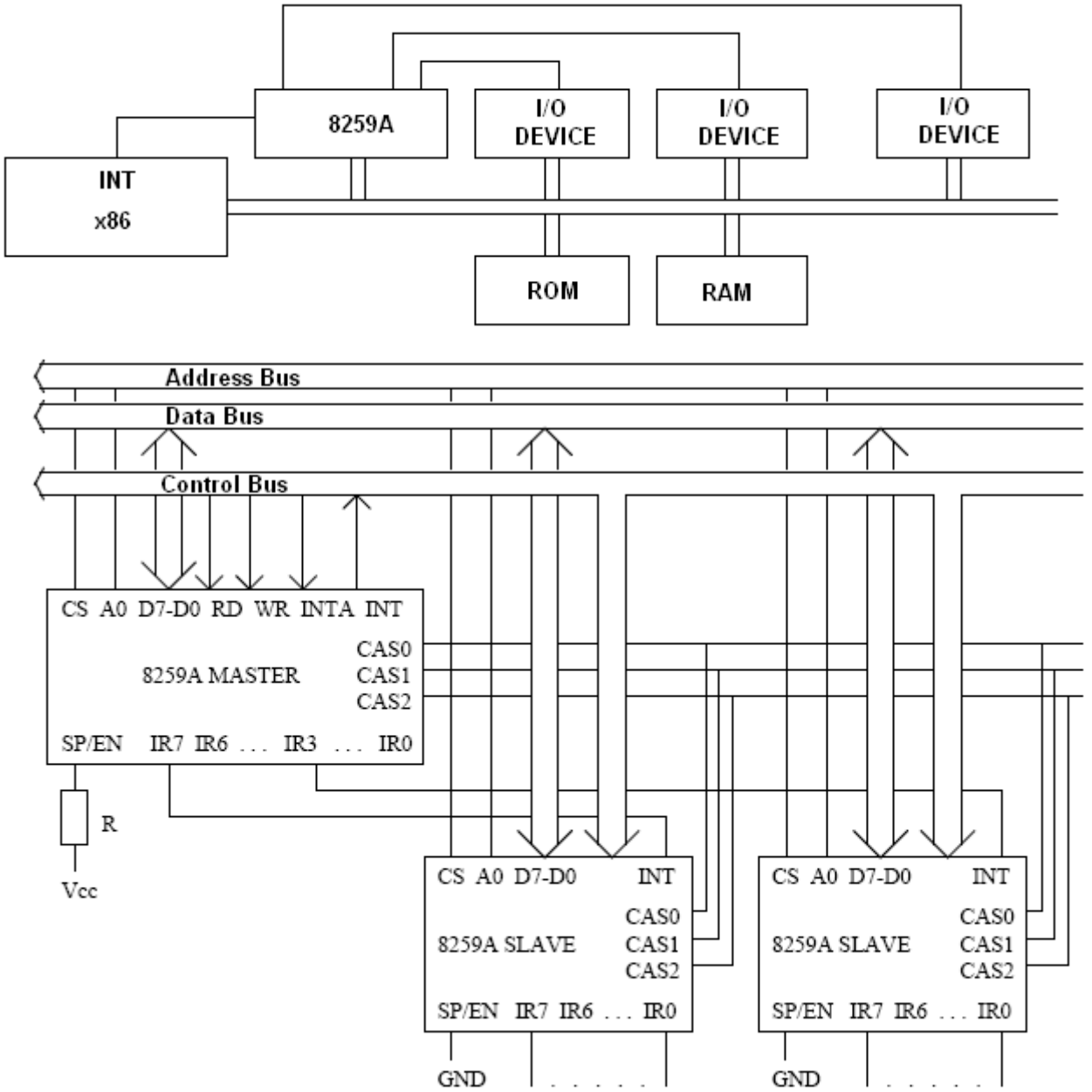


Fig. 3. 8259A system setup in a a cascade mode

Intel 8259A Programmable Interrupt Controller devices

PC/XT Architecture

IRQ0 – Intel 8253 or Intel 8254 Programmable Interval Timer
IRQ1 – Intel 8042 keyboard controller
IRQ2 – not assigned in PC/XT; cascaded to slave 8259 INT line in PC/AT
IRQ3 – 8250 UART serial port COM2 and COM4
IRQ4 – 8250 UART serial port COM1 and COM3
IRQ5 – hard disk controller in PC/XT; Intel 8255 parallel port LPT2 in PC/AT
IRQ6 – Intel 82072A floppy disk controller
IRQ7 – Intel 8255 parallel port LPT1 / spurious interrupt

PC/AT Architecture

IRQ8 – real-time clock (RTC)
IRQ9 – no common assignment
IRQ10 – no common assignment
IRQ11 – no common assignment
IRQ12 – Intel 8042 PS/2 mouse controller
IRQ13 – math coprocessor
IRQ14 – hard disk controller 1
IRQ15 – hard disk controller 2

Intel 8259A Programmable Interrupt Controller priorities

Keyboard press causing interrupt signal.

Main program

```
mov dx,
sub cx, cx
mov cl, StrLen
mov si, OFFSET Buffer
cmp Buffer, 10
jne StrLoop
inc si
dec cx
jb  cx
cmp BYTE PTR [si], 10
ja
inc si
loop StrLoop
stc
ret
clc
ret
```

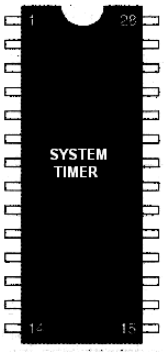
Keyboard Iterrupt

```
pusha
cmp ax, 0
jne NumbOverflow
mov si, OFFSET MinStr
mov di, OFFSET DestStr
 •
 •
 •
mov cx, 7
cld
rep movsb
popa
iret
```

Keyboard Iterrupt handler

End of Keyboard Iterrupt handler: restore processor state and return to the next instruction of the interrupted program.

System timer generates interrupt 18 times per second



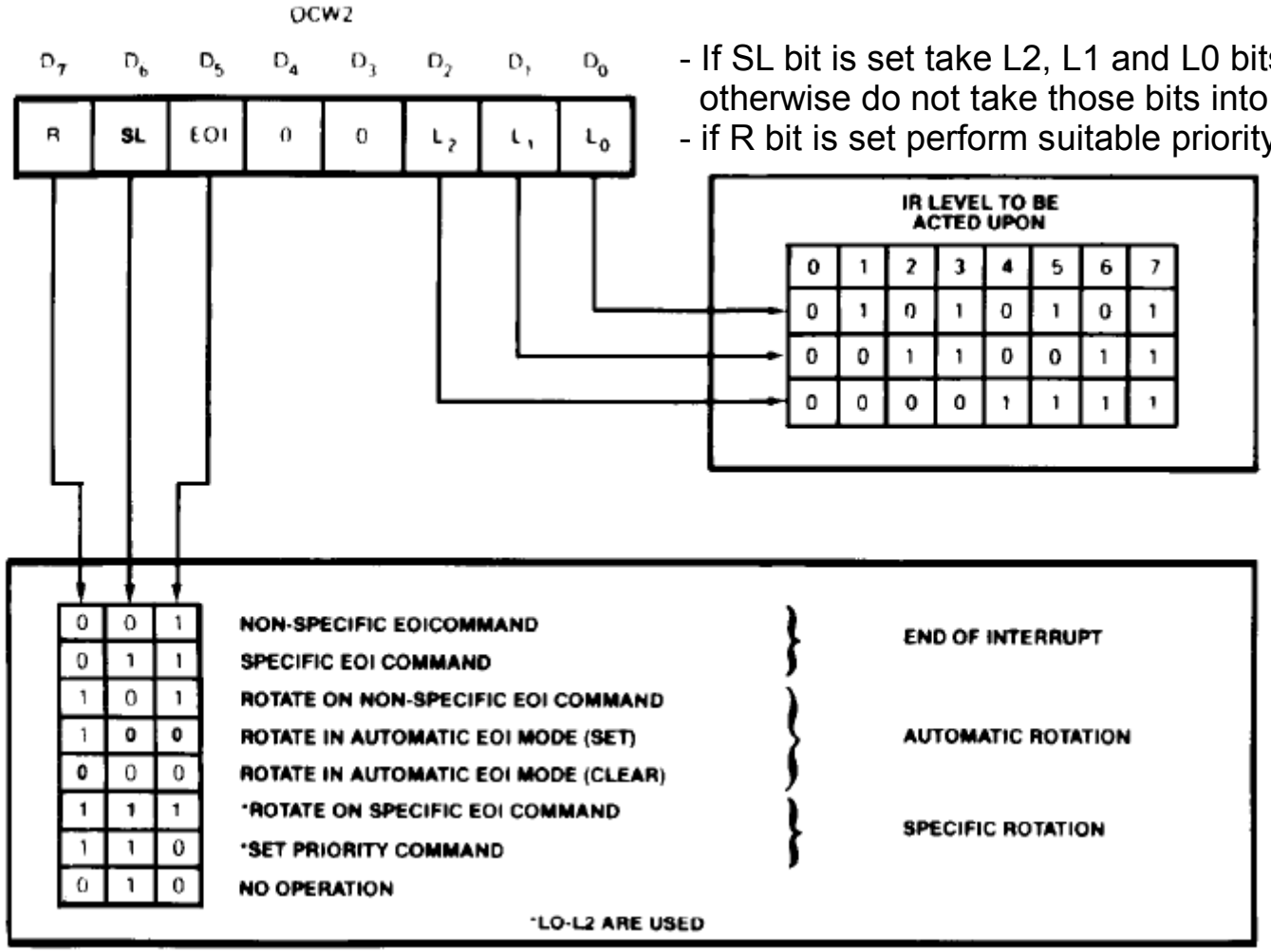
System timer Iterrupt handler has a higher priority so it interrupts keyboard interrupt handler

```
pusha
cmp ax, 0
jne NumbOverflow
mov si, OFFSET MinStr
mov di, OFFSET DestStr
 •
 •
 •
mov cx, 7
cld
rep movsb
popa
iret
```

End of System Timer Iterrupt handler: restore processor state and return to the next instruction of the interrupted program.

Intel 8259A Programmable Interrupt Controller programming

Interrupt Controller is programmed by sending to its Data Buffer special control words **ICW*n*** (Initialization Control Word no. *n*) and **OCW*n*** (Operational control word no. *n*). The most commonly used OCW is OCW2 which enables interrupt priority setup and general interrupt handling.

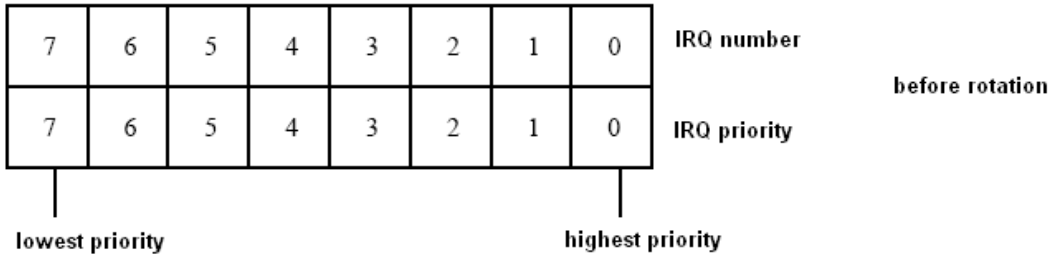


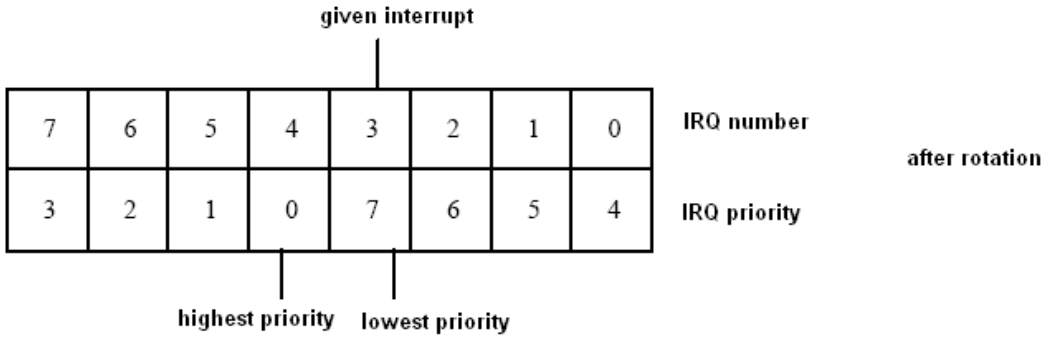
- If SL bit is set take L2, L1 and L0 bits into consideration otherwise do not take those bits into consideration
- if R bit is set perform suitable priority rotation

Fig. 4. 8259A OCW2 format

Intel 8259A Programmable Interrupt Controller programming

Interrupt Controller Data Buffer is located at system port 20h.
For example to send OCW2 to request an IRQ5 to have a lowest priority one can use C++ command: **outp(0x20, 0xC5);**
where **20h** is the port number in which the Data Buffer is located at and **C5h** is the OCW2 command saying „set the lowest priority for IRQ5 and rotate all other priorities" i.e IRQ6 wil have a highest priority, the next highest priorities will be for IRQ7, IRQ0, IRQ1, IRQ2, IRQ3 and IRQ4.
Next example sends OCW2 to inform Interrupt Controller that the interrupt procedure is about to be finished by sending non-speciofic **EOI** (End of Interrupt) command to a controller:**outp(0x20, 0x20);**





Interrupt handlers

Interrupt handler is a procedure which is called by the processor each time an interrupt occures. Every interrupt has it's own number (do not confuse it with a hardware IRQ number) and an interrupt vector associated with it. Interrupt vectors are situated at addres 0000:0000 each has 4 bytes in length and stores the address of a suitable interrupt handler. Interrupt numbers along with their usage is shown below:

00h - 01h	Exception Handlers	-
02h	Non-Maskable IRQ	Non-Maskable IRQ (Parity Errors)
03h - 07h	Exception Handlers	-
08h	Hardware IRQ0	System Timer
09h	Hardware IRQ1	Keyboard
0Ah	Hardware IRQ2	-
0Bh	Hardware IRQ3	Serial Comms. COM2/COM4
0Ch	Hardware IRQ4	Serial Comms. COM1/COM3
0Dh	Hardware IRQ5	Reserved/Sound Card
0Eh	Hardware IRQ6	Floppy Disk Controller
0Fh	Hardware IRQ7	Parallel Comms.
10h - 6Fh	Software Interrupts	-
70h	Hardware IRQ8	Real Time Clock
71h	Hardware IRQ9	Redirected IRQ2
72h	Hardware IRQ10	Reserved
73h	Hardware IRQ11	Reserved
74h	Hardware IRQ12	PS/2 Mouse
75h	Hardware IRQ13	Math's Co-Processor
76h	Hardware IRQ14	Hard Disk Drive
77h	Hardware IRQ15	Reserved
78h - FFh	Software Interrupts	-

Interrupt handlers

One can override original interrupt handler by his/her own procedure by changing the address of an suitable interrupt handler in a interrupt vector table.
For example to change a keyboard interrupt handler to our own procedure we have to write the address of our handler to a 09h-th index of the interrupt vector table. Typical C++ code for doing so is hsown below:

```
void interrupt (* OldKeyboardHandler)(...);     //a variable for preserving
                                                //original keyboard handlrer
                                                //adderss

void interrupt OurKeyboardHandler(...)          //our new keyboard handler
{
 asm sti                                        //enable other interrupts
 OldKeyboardHandler();                          //call original keyborad handler
                                                //to handle key press
 //make some operations                         //make some operations specific
                                                //to our new handler
 outp(0x20,0x20);                               //send EOI to the interrupt controller
}

void main()                                     //main program fucntion
{
 OldKeyboardHandler = getvect(0x09);            //save original keyboard handler address
 setvect(0x09, OurKeyboardHandler);             //set ne keyboard handler address to our new routine
 //make some operations                         //make some program operations
 setvect(0x09, OldKeyboardHandler);             //restore original handler prior to ending the program
}
```

Thank you for today's lecture.