

i386 Interrupt Handling

Interrupts and the IDT

The i386 processor treats hardware interrupts in the same manner as Exceptions and Software interrupts. Interrupt service routines are registered with the [IDT](#) and then are vectored accordingly based on the [8259A](#) Programmable Interrupt Controller.

The 8259A Programmable Interrupt Controller

The 8259A has 8 inputs, 1 [interrupt](#) line, and a way for the processor to query it. Each of the 8 lines are hooked up to a specific IRQ (0-7). When that IRQ fires the 8259A remembers which line it came in on and raises its [interrupt](#) line which is connected to the [processor](#). Then, by *magic*, the processor queries the 8259A, figures out which IRQ fired and vectors the correct spot in the IDT.

Some time ago, after the IBM XT, a brilliant engineer decided that 8 IRQs was not enough. Because at that time cards could not share interrupts well and they needed to maintain compatibility with the XT, it was decided to just chain another 8259A off of the original one. These days PCs have both a [master](#) and a [slave](#) interrupt controller.

Programming the 8259A

Each 8259A has [two](#) io ports assigned to each of them. For simplicity we'll call them port a and port b. They are assigned as follows:

	port a	port b
Master 8259A	0x20	0x21
Slave 8259A	0xA0	0xA1

The 8259A is [configured](#) through a series of [4](#) control words. To configure it send the [first](#) word (ICW1) to port [a](#) and the rest are sent to port [b](#).

ICW1

7	6	5	4	3	2	1	0
0	0	0	1	Trig	0	M/S	ICW4

[Trig](#) 0 = Edge triggered 1 = Level Triggered

[M/S](#) 0 = Master/Slave configuration 1 = Master only

[ICW4](#) 0 = No ICW4 1 = ICW4 will be sent

Same [master](#) value: 000[1](#)0001b (0x11)

Same [slave](#) value: 000[1](#)0001b (0x11)

ICW2

7	6	5	4	3	2	1	0
Off 7	Off 6	Off 5	Off 4	Off 3	0	0	0

Off 7 .. Off 3 Offset into the **IDT** for interrupt service routines
 The last 3 bits are always 0 because it needs to be 3 bit aligned

Sane **master** value: 00100000b (0x20) = 32

Sane **slave** value: 00101000b (0x28) = 40

ICW3 (master)

7	6	5	4	3	2	1	0
S7	S6	S5	S4	S3	S2	S1	S0

S7..S0 0 = IR Line is connected to **peripheral** device
 1 = IR Line is connected to **Slave** 8259A

Sane **master** value: 00000**1**00b (0x04)

ICW3 (slave)

7	6	5	4	3	2	1	0
0	0	0	0	0	ID2	ID1	ID0

ID2..ID0 IRQ on **master** this **slave** is connected to

Sane **slave** value: 00000010b (0x0**2**)

ICW4 (optional)

7	6	5	4	3	2	1	0
0	0	0	SFNM	BUF	M/S	AEOI	Mode

SFNUM 0 = No Special Fully Nested Mode **1** = Special Fully Nested Mode

BUF 0 = No Buffered Mode 1 = Buffered Mode

M/S 0 = Slave PIC **1** = Master PIC

AEOI 0 = Manual EOI 1 = Automatic EOI

Mode 0 = MCS-80/85 Mode 1 = 8086/88 Mode

Sane **master** value: 00000**1**0**1**b (0x05)

Sane **slave** value: 0000000**1**b (0x01)

Initialization

To **inititalize** the 8259A just send the 4 ICWs to both the slave and master like so:

```

/* ICW1 */
outb( 0x11, 0x20 ); /* Master port A */
outb( 0x11, 0xA0 ); /* Slave port A */

```

Port A/B
 Master 8259A - 0x20 / 0x21
 Slave 8259A - 0xA0 / 0xA1

```

/* ICW2 */
outb( 0x20, 0x21 ); /* Master offset of 0x20 in the IDT */
outb( 0x28, 0xA1 ); /* Master offset of 0x28 in the IDT */

/* ICW3 */
outb( 0x04, 0x21 ); /* Slaves attached to IR line 2 */
outb( 0x02, 0xA1 ); /* This slave in IR line 2 of master */

/* ICW4 */
outb( 0x05, 0x21 ); /* Set as master */
outb( 0x01, 0xA1 ); /* Set as slave */

```

master 0x11
 slave 0x11
 master 0x20
 slave 0x28
 master 0x04
 slave 0x02
 master 0x05
 slave 0x01

Interrupt Mask

The 8259A has the ability to mask off interrupts you tell it to. This is done by writing an 8 bit value corresponding to the mask to port **b** while the PIC is in **normal** operation mode. The PIC will **only** listen to interrupt which have a **0** in the corresponding bit in its mask.

example:

```

/* Only listen to irqs 0, 1, and 2 */
outb( 0xf8, 0x21 ); /* master PIC */
outb( 0xff, 0xA1 ); /* slave PIC */

```

EOI

When the 8259A is in Manual EOI mode (ICW4[1]==0) it will wait for an EOI (End Of Interrupt) **acknowledgment** to be written to it before it will send any more. To send the EOI simply write **0x20** to port **a** of the PIC.

example:

```

/* Send EOI to both master and slave */
outb( 0x20, 0x20 ); /* master PIC */
outb( 0x20, 0xA0 ); /* slave PIC */

```
