Interrupts, Exceptions, and IDTs

Part 3 - IDTs

What is an IDT?

An IDT(**I**nterrupt **D**escriptor **T**able) is an array of descriptors(each 8 bytes long) that is used to associate interrupts and exceptions with ISRs. The IDT can hold a maximum of 256 descriptors(as there are 256 interrupts total on the PC). You are not required to have an IDT with a full 256 descriptors in it, just enough descriptors for the interrupts you are going to use. We tell the CPU where the IDT is via the LIDT assembly instruction.

IDT Descriptor Format

The 8 byte long descriptors that are in the IDT follow this format(numbers in dark grey boxes are bits, numbers in light grey boxes are bytes):

31	15	14	12	7	4	0
Offset 31-16(word)	Present(1 bit)	DPL(2 bits)	01110(5 bits)	000(3 bits)	Not Used	4
Selector31-16(word)	Offset 15-0(word)					0

Present: 0 = not present, 1 = present. If Present is not set to 1, an exception will occur
DPL: Descriptor Privilege Level, 00 = ring0, 01 = ring1, 10 = ring2, 11 = ring3
Selector: Code selector that the ISR will use
Offset: The address of the ISR, this is split between two different fields to keep the 386 and higher compatible with the 286

Now let's make a descriptor so we can understand how they work. For this descriptor, let's set the DPL to ring0, present to 1, selector to 0x10(this will differ depending on the layout of the GDT), and the offset(ISR address) to 0x200000. We end up with this:

31	15	14	12	7	4	0
0x20	1	00	01110	000	00000	4
0x10	0x0000					0

Now that we know what the descriptor looks like, let's code it in NASM:

```
dw 0x0000
dw 0x10
dw 0x8E00
dw 0x20
```

So now we know how to make descriptors, and therefore an IDT. But that's not very useful unless we also know how to tell the CPU where to find it. That brings us to the next section...

Loading the IDT via LIDT

We tell the CPU where the IDT is via the LIDT assembly instruction. LIDT takes a paramter, a pointer. This pointer points to a little structure that describes the IDT allowing the CPU to know where it starts, and how many descriptors are in it. This structure(we will call it the IDT pointer) is laid out like this(numbers in dark grey boxes are bits, numbers in light grey boxes are bytes):

31	15	0
	Limit 15-0(word)	4
Base 31-0(double word)		0

Base: address for the start of the IDT
Limit: length of the IDT in bytes

So let's put together a little IDT of just three entries so we can see how to use the LIDT instruction. Below is the NASM code for the IDT:

```
start_of_idt:
;first entry
dw 0x0000
dw 0x10
dw 0x8E00
dw 0x20
;second entry
dw 0x0000
dw 0x10
dw 0x8E00
dw 0x30
;third entry
dw 0x0000
dw 0x10
dw 0x8E00
dw 0x10
end_of_idt:
```

Now the *start_of_idt* and *end_of_idt* gives us addresses to feed into the IDT pointer. So, we put these into our IDT pointer and end up with this:

31	15	0
	end_of_idt - start_of_idt - 1	4
start_of_idt		0

Translated to NASM code, the IDT pointer looks like this:

```
idt_pointer:
dw end_of_idt - start_of_idt - 1
dd start_of_idt
```

Okay, now we have the IDT pointer finished and we can use the LIDT instruction now! All we have to do is give the address of *idt_pointer* like this:

```
lidt [idt_pointer]
```

See? Pretty easy actually.

Tying it all Together

Now let's put all the stuff we've learned in this tutorial together and see a fully working IDT. This IDT will map interrupts 0-16 to an ISR that is located at 0x200000(except for interrupts 2 and 15, the interrupts Intel has reserved).

The code for this in NASM looks like this:

```
[section .text]
;load the IDT, this requires that the data section with
;the IDT exists
lidt [idt_pointer]

[section .data]
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; the IDT with it's descriptors
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
start_of_idt:
;interrupt 0
dw 0x0000
dw 0x10
dw 0x8E00
dw 0x20

;interrupt 1
dw 0x0000
dw 0x10
dw 0x8E00
dw 0x20

;interrupt 2, intel reserved, we set the 'present' bit to 0 on this one
dw 0x0000
dw 0x10
dw 0xE00
dw 0x20

;interrupts 3-14 now, since we are making the descriptors
;identical, we are going to loop to get them all(12 total)
%rep 0xC
  dw 0x0000
  dw 0x10
  dw 0x8E00
  dw 0x20
%endrep

;interrupt 15, intel reserved, we set the 'present' bit to 0 on this one
dw 0x0000
dw 0x10
dw 0xE00
dw 0x20

;interrupt 16
dw 0x0000
dw 0x10
dw 0x8E00
dw 0x20
end_of_idt:

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; now for the IDT pointer
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
idt_pointer:
  dw end_of_idt - start_of_idt - 1
  dd start_of_idt
```

Thanks go to xsism for proofreading/suggestions