Interrupts, Exceptions, and IDTs

Part 2 - Exceptions

What is an Exception and What Triggers Them?

An exception is an interrupt that occurs when something goes wrong with the currently executing code. This might be dividing by zero, trying to access a segment that doesn't exist, or some similar error.

There are 15 exceptions total on the x86 CPU that are spread from interrupt 0 to interrupt 16. Yes, that means that are some 'gaps' in there were there are no exception interrupts. Those gaps have been reserved by Intel. Most likely, Intel was going to or will use them for more exception interrupts later on. The layout of the exception interrupts looks like this:

0 Divide Error
1 Debug Exceptions
2 Intel reserved
3 Breakpoint
4 Overflow
5 Bounds Check
6 Invalid Opcode
7 Coprocessor Not Available
8 Double Fault
9 Coprocessor Segment Overrun
10 Invalid TSS
11 Segment Not Present
12 Stack Exception
13 General Protection Exception(Triple Fault)
14 Page Fault
15 Intel reserved
16 Coprocessor Error

What Does a Simple Exception ISR Look Like?

The template that I gave in the first part of this tutorial for a plain ISR will work for an exception ISR as well. Having said that though, I'm not gonna just give you the template again. Instead I'm gonna show you a simple, complete exception ISR for exception 0. This exception ISR mixes C and assembly code, so depending on the output format that you use, you may have to change a few things(such as the underscores with the C function).

```
isr0:
 pusha
 push gs
 push fs
 push ds
 push es
```

```
 extern _interrupt_0
 call _interrupt_0

 pop es
 pop ds
 pop fs
 pop gs
 popa
 iret
```

Now for the C function, *interrupt_0*:

```
void interrupt_0()
{
   printf("Exception: Divide Error");
   // if we just have ring 0 tasks, we just need to halt the computer
   // if we also had ring 3 tasks, and a ring 3 task was the cause of the
   // exception, then we should delete the ring 3 task and continue on
   asm("cli");
   asm("hlt");
};
```

See? Pretty easy actually, you just need to have a function for each exception. If we wanted, we could pass a variable to the C function telling which exception happened and then we would only need one C function for all the exception interrupts(read the Getting Started tutorial for info on passing info to a C function from assembly to figure out how to do this).

Exception Interrupt 14 - Page Fault

This exception interrupt gets it's own little section because if you are using paging, you will be using this exception interrupt a lot. Basically, if you have paging enabled and a program tries to access a page that is marked as not-present, the CPU calls this exception interrupt. It is your job to either fix the problem or nuke the program that caused it. Here is an example ISR for this exception:

```
isr14:
 pusha
 push gs
 push fs
 push ds
 push es

 mov eax, cr2 ; CR2 contians the address that the program tried to access
 push eax ; now our C function can get it

 extern _interrupt_14
 call _interrupt_14

 pop eax ; have to pop what we pushed or the stack gets messed up

 pop es
 pop ds
 pop fs
 pop gs
 popa
 iret


void interrupt_14(u_long cr2)
{
  // do whatever, the cr2 variable contains the address that the program
  // tried to access that generated the page fault
};
```

That's just a shell for the page fault exception. I'm not going to go deeper into it as this is an interrupts tutorial, not a memory management tutorial.

Thanks go to Jimferd/Akira for proofreading