Interrupts

An **interrupt** causes the computer to stop doing what it is doing, save the current state of the interrupted program and transfers control to an **interrupt handler**, which is in another program or the operating system. When it saves the state information, it must save all the of necessary information so that the interrupted program can resume without any indication of being interrupted. An interrupt is said to be **transparent** to the interrupted program. There are hardware and software interrupts. In general, interrupts can be viewed as a good thing, when the interrupts handlers are properly written and interrupts don't happen too often. Examples of when interrupts are used are: 216 280

- Completion of I/O, such as key presses and releases. This allows for computations in what would be the wasted time between keystrokes and greatly increasing the computers productivity (*throughput*).
- Computers have a **real-time clock** that will cause a set number of interrupts per second. This keeps the time-of-day clock up to date and to give each user a **time-slice** so that the computer can be a **time-shared system or multiprocessing system.** There are also **interval timers** which allow the system to be interrupted at the request of the applications programmer.
- Error interrupts such as divide by zero, illegal opcode, illegal memory address (does not exist or is allocated to another program) and hardware error conditions (out of paper on the printer, floppy disk door open, etc).
- Execution of a group of instructions such as INT 21 as we saw earlier.

Each interrupt has an **interrupt number** which is communicated to the CPU in some way by the causer of the interrupt. This number is used to index a table of handler addresses (**vectored interrupts**).

Interrupts can be **enabled** or **disabled.** There are some operations that can only take place reliably when interrupts are disabled; however disabling interrupts is not something that we want to happen indiscriminately! Additionally, there are some interrupts that can not be disabled, such as caused by power failure or machine errors. We can disable some by changing a bit in a mask, and they are called **maskable interrupts** and the others are called **non-maskable interrupts.**

More than one interrupt can occur during the execution of an instruction. When this happens, at the end of the instruction, the CPU picks one interrupt to be serviced. The others are held until the end of another instruction, when another one will be serviced.

There are many ways to classify interrupts:

- Hardware vs. Software generated interrupts.
- Internal (within the CPU) vs. External interrupts.
- Asynchronous vs. Synchronous interrupts.
- Maskable vs. Non-Maskable interrupts.

Interrupt Processing on the 80X86

There are 256 interrupts possible on the 80x86 computers. Any one of which can be caused to executing the instruction:

```
INT   n      ;cause interrupt n
```

Interrupts can be enables or disable with the instructions:

```
sti     ;set I flag to 1, enable
cli     ;set I flag to 0, disable
```

Memory locations 0 - 1023 hold a four-byte address (segment:offset) in a **interrupt vector table** for each interrupt handler.

The interruption mechanism is:

1. When a device wishes to cause an interrupt, it makes an **interrupt request** that includes the number of the interrupt. When arriving in the CPU it is held until the completion of the current instruction.
2. At the end of each instruction, before the next instruction is fetched, the CPU checks to see if there are any interrupts waiting. If there is, one is selected for **interrupt service** and the others are held.
3. The CPU does the following items:
    1. Push the flag register onto the stack
    2. Disable interrupts.
    3. Push the cs and IP registers.
    4. Load the address of the handler.

The rest is up to the specific interrupt handler, however, it should quickly re-enable interrupts (which implies that the handler itself can be interrupted). Any register that will be used must first be pushed onto the stack. The handler then does what it is suppose to do. Finally, the handler must pop the saved registers, and issue a special return:

```
iret    ;return from interrupt.
```

This will pop the IP, cs and flag registers, which will also re-enable the interrupts.

The interrupt vector table is setup by the boot-up procedure in the BIOS when the computer starts up, setting addresses of interrupt handlers for interrupts that the BIOS handles, and initializing the rest to a handler that consists of nothing but the *iret* instruction. This makes it easy to upgrade the handlers in future releases of the operating system or even to have a different operating system altogether. When the operating system loads, it puts in the address of its handlers where necessary.

WARNING: Since interrupts occur at any time, they wipe out anything that had been put onto the stack previously. Never assume that something you popped is still on the stack waiting for you!

Interrupt Handler

If we want to write an interrupt, we would have to change the address of the handler in the Interrupt Handler Vector Table. There are two macros, _SetInvVec and _SaveInvVec, that will take care of that item. Then we would have to write a handler in the following form:

```
Handler     PROC
                    ; push all registers used
                ...     ; process interrupt
                iret    ; pop all pushed registers
Handler     ENDP
```

Types of Interrupts

- **Hardware vs. Software generated interrupts.**

A hardware interrupt is a signal created and sent to the CPU that is caused by some action taken by a hardware device. E.g. keystroke depressions and mouse movements cause hardware interrupts.

A software interrupt is an interrupt caused by an instruction in the program. E.g. int 21h

- **Internal (within the CPU) vs. External interrupts.**

An internal interrupt is a signal for attention sent to a computer's central processing unit by another component of the computer.

An external interrupt is caused by an external source such as the computer operator, external sensor or monitoring device, or another computer.

- **Asynchronous vs. Synchronous interrupts.**

An asynchronous interrupt can occur randomly at any time. E.g. mouse movements.

A synchronous interrupt is one that always occur in the same stage of execution. E.g. Divide by Zero error or page fault.

- **Maskable vs. Non-maskable interrupts.**

A maskable interrupt are hardware interrupts that can be allowed to occur or prevented from occurring by software.

A non-maskable interrupt can not be ignored and is typically used to signal attention for non-recoverable hardware errors. E.g. Power Failure