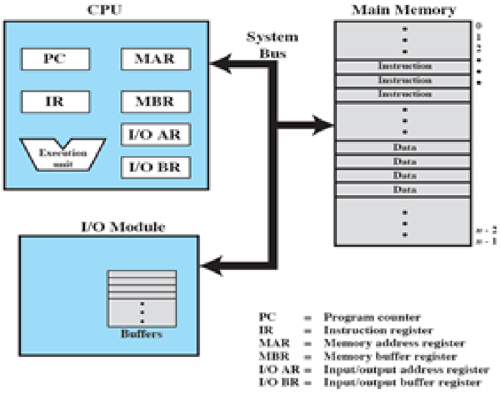
Week 1

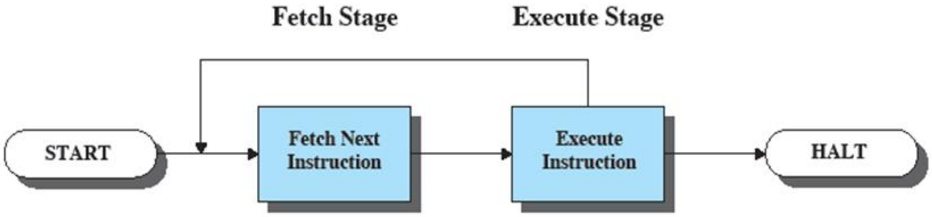
Basic single threaded operation

Each instruction in main memory is 1 **assembly language** instruction



And here is the basic Instruction cycle that processor follows

- processor reads (fetches) instructions from memory
- PC is incremented after every instruction
- processor executes each instruction



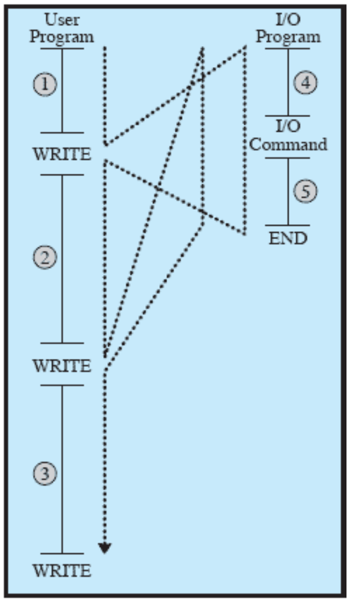
So you could do this forever but it’s not efficient

I/o devices are much slower than the processor (pretty much anything is) so if you try to read or write you are going to spend a lot of time in a busy wait loop,

To give a specific example, consider a PC that operates at 1 GHz, which would allow roughly 10 9 instructions per second. 2 A typical hard disk has a rotational speed of 7200 revolutions per minute for a half-track rotation time of 4 ms, which is 4 million times slower than the processor.

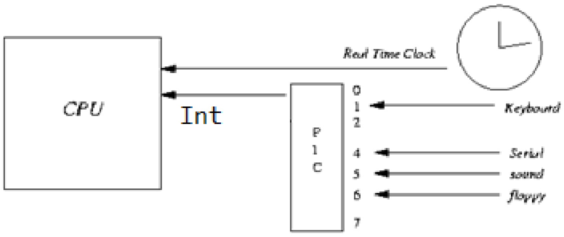
So for the following program, assume that the average number of assembly instructions executed for 1-5 is 100

So the I/O command would take 4,000,000

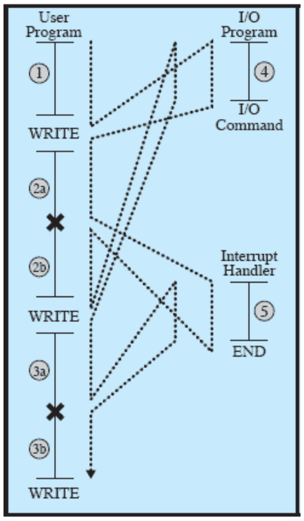


(a) No interrupts

Solution is interrupts, offload the work to an io device and let it interrupt the processor when its finished

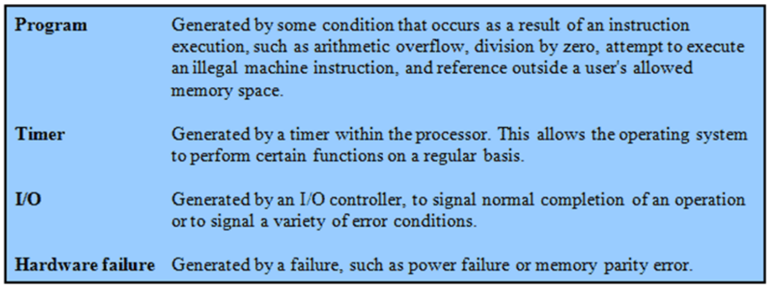


Hardware interrupt example



(b) Interrupts; short I/O wait

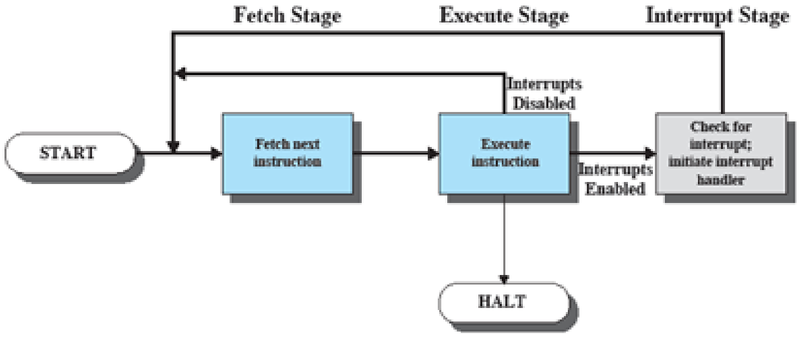
Classes of Interrupts



Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, and reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.
Hardware failure	Generated by a failure, such as power failure or memory parity error.

Now the processor can do other stuff while the I/O device handles the heavy lifting
Still single threaded.

So how does this affect the **processor** instruction cycle?



Think what this means, if we can interrupt for I/O we can interrupt and then load another process. And this is exactly how you do preemptive multiprocessing, interrupts that allow the OS scheduler to slot in another process