Notes Chapter 1

In simple terms: Preemptive multitasking involves the use of an interrupt mechanism which suspends the currently executing process and invokes a scheduler to determine which process should execute next. Therefore, all processes will get some amount of CPU time at any given time.

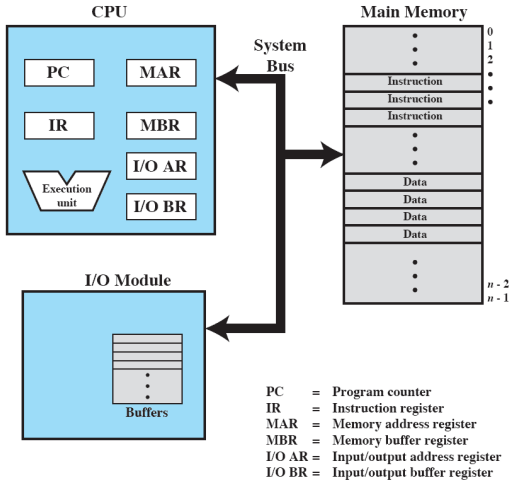
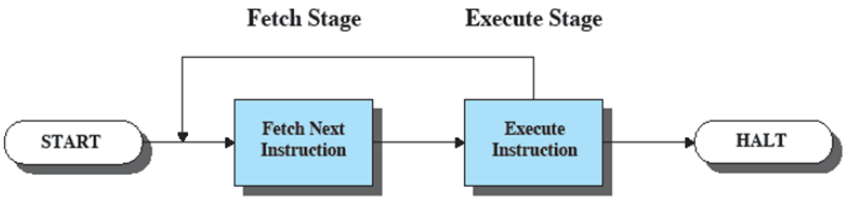


Figure 1.1 Computer Components: Top-Level View

1 assemby = 1 machine

Even using interrupts, current machine instruction will finish

Interrupts happen between machine instructions



PC At the beginning of each instruction cycle, the processor fetches an instruction from memory. Typically, the program counter (PC) holds the address of the next instruction to be fetched. Unless instructed otherwise, the processor always increments the PC after each instruction fetch so that it will fetch the next instruction in sequence

IR Fetched instruction is loaded into Instruction Register (IR). May specify mem access, processor I/o, arithmetic, control flow (branch if)

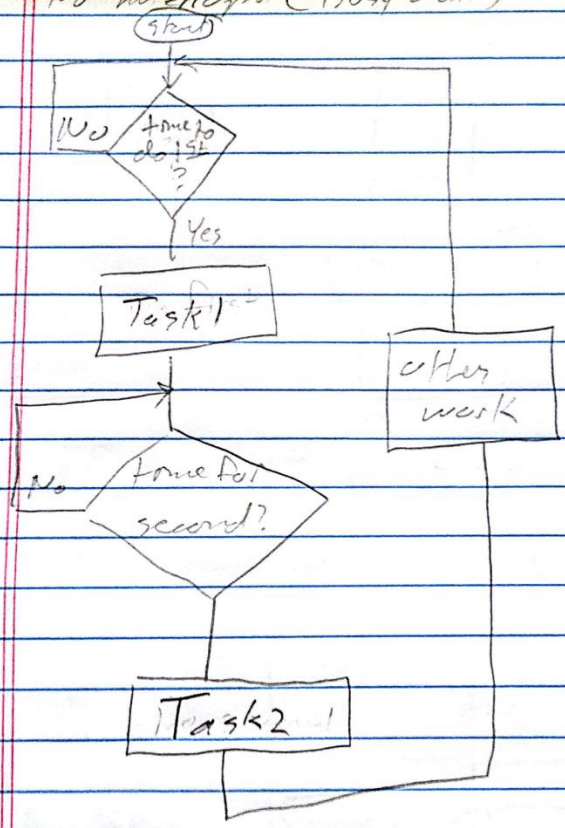
Interrupts improve processor utilization (delegates to slow peripherals disk I/O, keyboard)
Processor waits around or asks periph to notify when done

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.

Timer used for multitasking
I/O used by IO devices to alert processor

With and without interrupts

No interrupts (Busy wait)



T1 write 0

T2 write 1

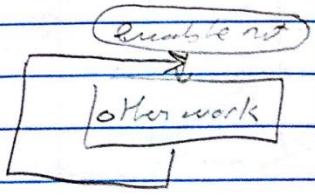
1) can't do 2nd until 1st done
2) stuck in wait loop

if Task1 is 10 likely 4,000,000 times slower than CPU.
2.5 Ghz ⇒ 4×10^{-10}
HDrive ⇒ 4×10^{-3}
10 million times slower
Execute 1 processor ms sec.
1 ms → 1 clo
1 s → 16.5 weeks

use interrupts

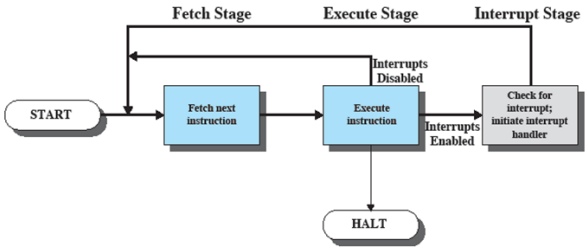
Int1 → task1 → return from Int

Int2 → Task2 → return from int

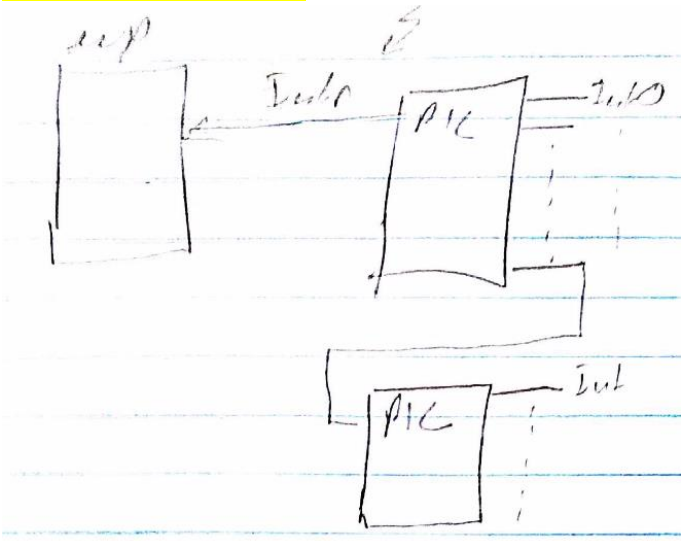


1) push PSW
2) do Int routine
3) return

Instruction cycle with interrupts



How to implement in HW



Must acknowledege int or else interrupted again
PIC programmable interrupt controller

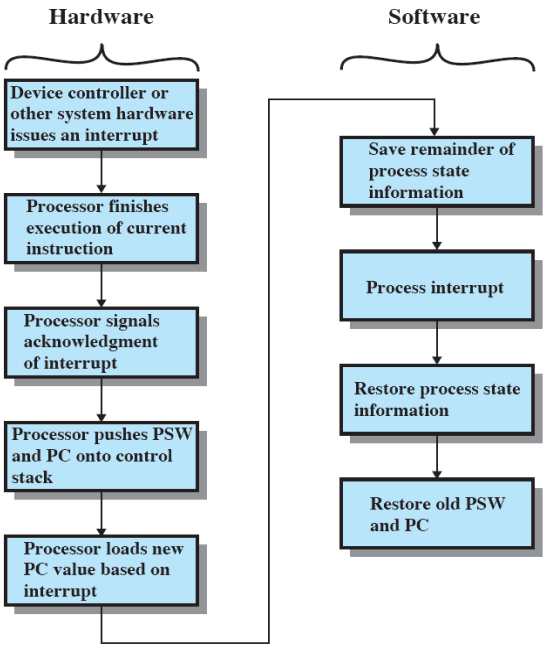


Figure 1.10 Simple Interrupt Processing

1. The device issues an interrupt signal to the processor.

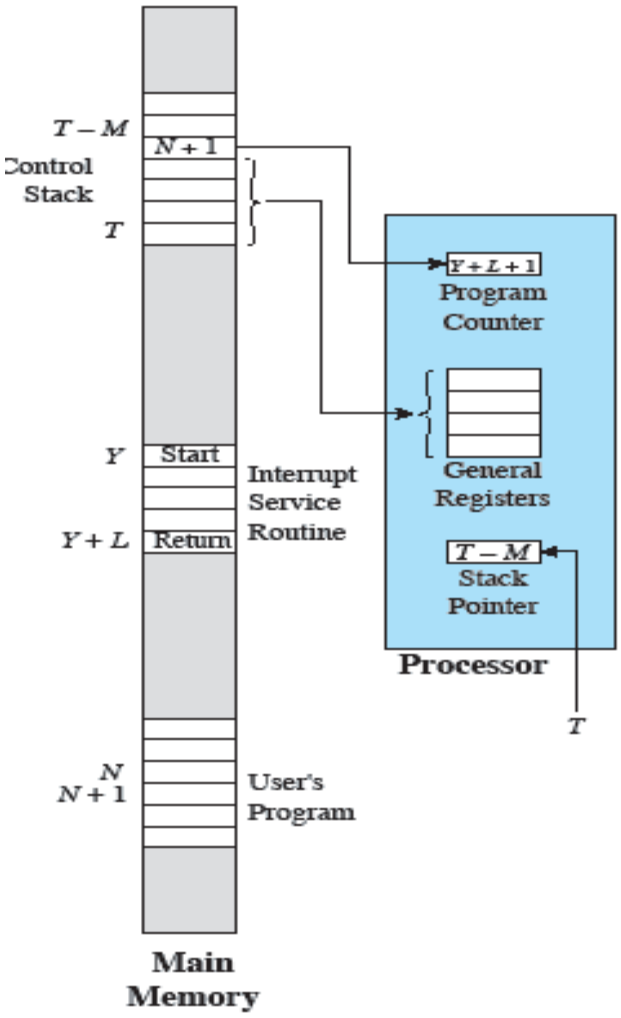
2. The processor finishes execution of the current instruction before responding to the interrupt,

3. The processor tests for a pending interrupt request, determines that there is one, and sends an acknowledgment signal to the device that issued the interrupt. The acknowledgment allows the device to remove its interrupt signal.

4. The processor next needs to prepare to transfer control to the interrupt routine. To begin, it saves information needed to resume the current program at the point of interrupt. The minimum information required is the program status word 3 (PSW) and the location of the next instruction to be executed, which is contained in the program counter (PC). These can be pushed onto a control stack

5. The processor then loads the program counter with the entry location of the interrupt-handling routine that will respond to this interrupt. Depending on the computer architecture and OS design, there may be a single program, one for each type of interrupt, or one for each device and each type of interrupt.

Software implementation



(b) Return from interrupt

Allows efficient utilization

Programmed I/o
timeslicing