Chapter 1
Computer System Overview

(based on original slides by Pearson)

WHAT IS AN OPERATING SYSTEM?

What is an OS

- Informally: Something that helps you use the available hardware
  - **Expensive Hardware, Cheap People**
    Goal: maximize hardware utilization
  - Now: **Cheap Hardware, Expensive People**
    Goal: make it easy for people to use.

- How to make it easier to use? Abstractions

Operating System

- An OS is a standardized abstraction, a VM, that is implemented on the underlying machine.
  - Manages resources: processor, memory, I/O
  - Provides a set of services to system users
  - Consumes resources (complexity vs cost)

WHAT DO WE NEED TO RUN AN OPERATING SYSTEM?

Basic Elements

- What do you need in a computer?
  - A processing unit (CPU)
  - Memory
  - I/O modules
  - System bus connecting them
- Question is: What do we need from these modules?

Note, these are already abstractions of more complicated hardware systems.

Basic Elements

- Processing unit
  - Some registers/buffers
    - Memory address resister (MAR)
      - Specifies the address for the next read or write
    - Memory buffer register (MBR)
      - Contains data written into memory or receives data read from memory
    - I/O address register
    - I/O buffer register

Basic Elements

- Main Memory
  - Volatile, referred to as real memory or primary memory
- I/O Modules
  - Secondary Memory Devices, communications equipment, terminals
- System bus
  - Communication among processors, main memory, and I/O modules

Computer Components

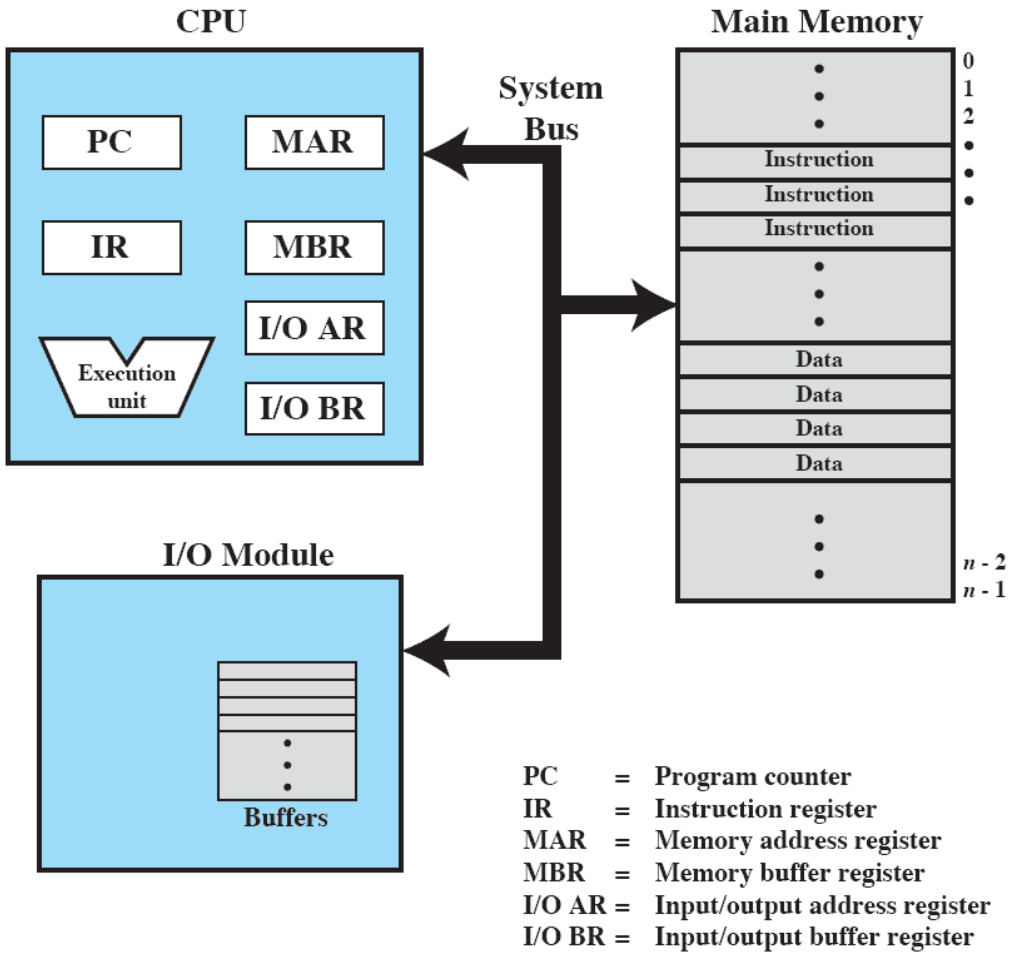


Figure 1.1 Computer Components: Top-Level View

User-Visible Registers

- Enable programmer to minimize main memory references by optimizing register use
- Available to all programs – application programs and system programs
- Depend on computer architecture (E.g. r1-r15; r0 might be hard coded to 0)
- May be referenced by machine language
- Usually the compiler controls them
- Can be user controlled
  - **Register** keyword in C
  - **Volatile** keyword in C

User-Visible Registers

- Two categories of visible registers:

1. **Data registers:** store data (e.g., results from calculations; math data section in the PIC18)
2. **Address register:** point to memory
   - Indexed addressing (Adding an index to a base value to get the effective address)
   - Segment pointer (When memory is divided into segments, memory is referenced by a segment and an offset)
   - Stack pointer (Points to top of stack for pop&push)

Control and Status Registers

- Invisible to the user (on most architectures)
- Used by processor to control operating of the processor
- Used by privileged OS routines to control the execution of programs

Examples of Invis. Registers

- Program counter (PC)
  - Contains the address of an instruction to be fetched
- Instruction register (IR)
  - Contains the instruction most recently fetched
- Program status word (PSW)
  - Contains status information and condition codes (e.g, Interrupts enabled, errors, supervisor mode bit)

Condition Codes or Flags

- Part of the PSW
- Bits set by processor hardware as a result of operations
  - Example: Positive, negative, zero, or overflow result of arithmetic operation
- Used for conditional branching
- Only implicitly accessible on most architectures
- Architecture specific ones include reset status (brownout, watchdog, power button).

SO HOW DO WE EXECUTE PROGRAMS ON THIS?

Instruction Execution

- Simple version:
  1. Processor fetches instruction from memory
  2. Processor executes the instruction
  3. Goto 1
- Reality:
  - Pipeline
  - VLIW Very Long Instruction Word
  - Superscalar architectures
  - … (see your computer architecture class)

Basic Instruction Cycle

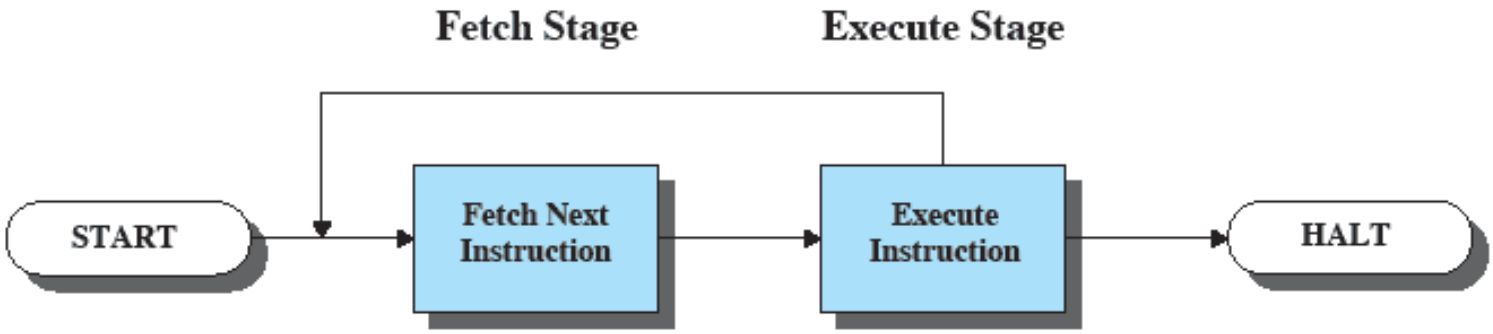


Figure 1.2 Basic Instruction Cycle

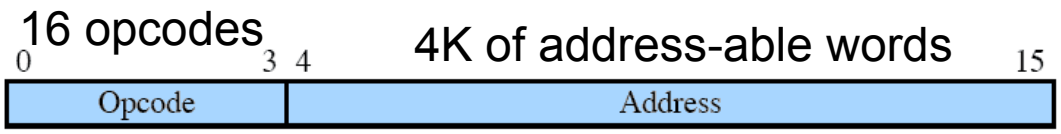
Instruction Fetch and Execute

- The processor fetches the instruction from memory
- Program counter (PC) holds address of the instruction to be fetched next
- PC is incremented after each fetch

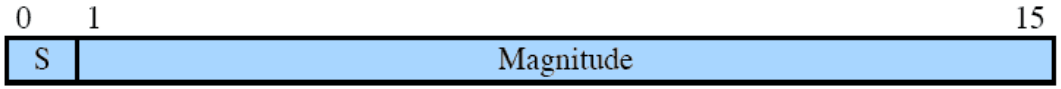
Instruction Register

- Fetched instruction loaded into instruction register (IR)
- Categories of actions dictated by the instruction
  - Processor-memory, processor-I/O, data processing, control (e.g., branching)
  - No clear boundaries; an instruction may include several of these actions

A Hypothetical Machine



(a) Instruction format



(b) Integer format

Program counter (PC) = Address of instruction
Instruction register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

(c) Internal CPU registers

0001 = Load AC from memory
0010 = Store AC to memory
0101 = Add to AC from memory

(d) Partial list of opcodes

Figure 1.3 Characteristics of a Hypothetical Machine

Example of Program Execution

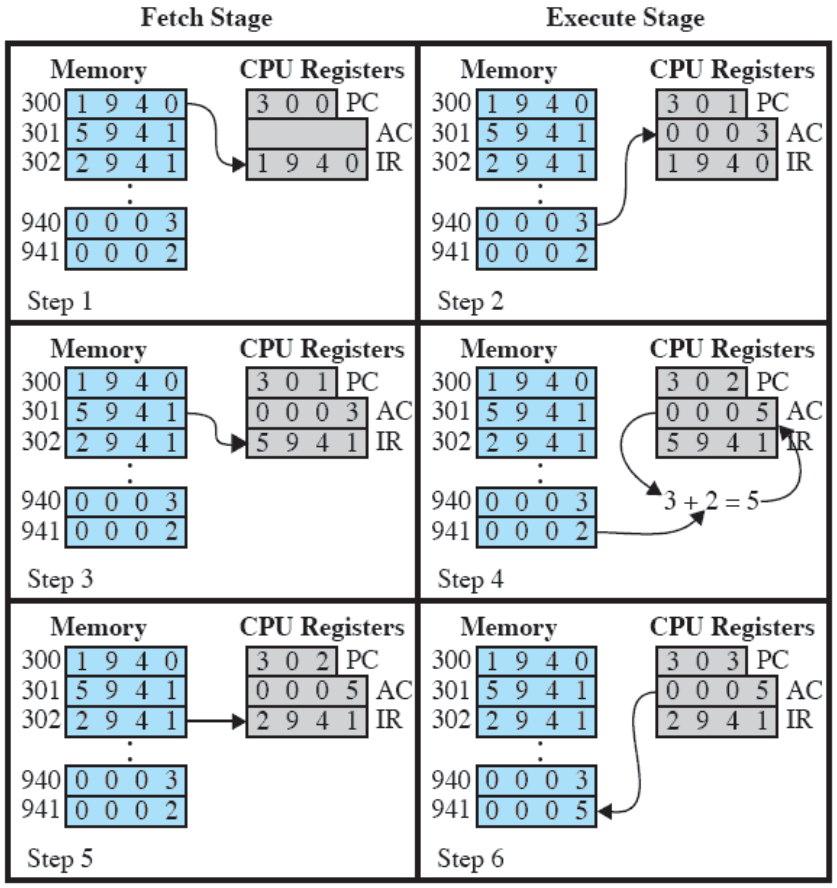


Figure 1.4 Example of Program Execution
(contents of memory and registers in hexadecimal)

HOW TO DEAL WITH PERIPHERALS?

Interrupts

- Most I/O devices are slower than the processor
  - Processor must pause to wait for device
- Interrupts help to improve the processor utilization.
- The change the normal sequencing of the processor

Classes of Interrupts

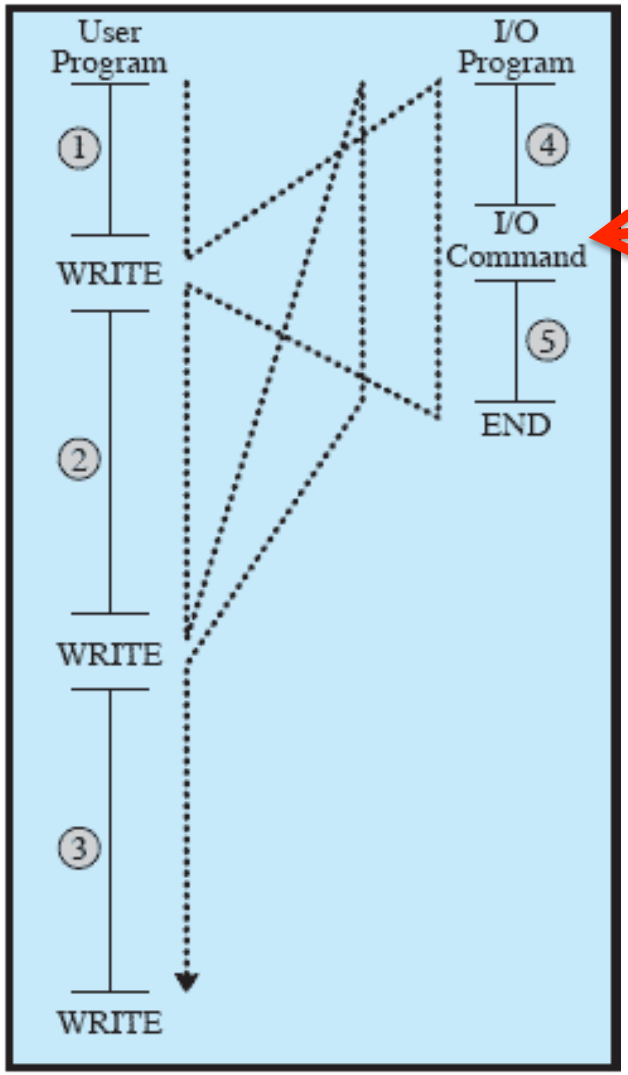
Table 1.1 Classes of Interrupts

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.

Example: Watchdog

Program Flow of Control

No interrupts.

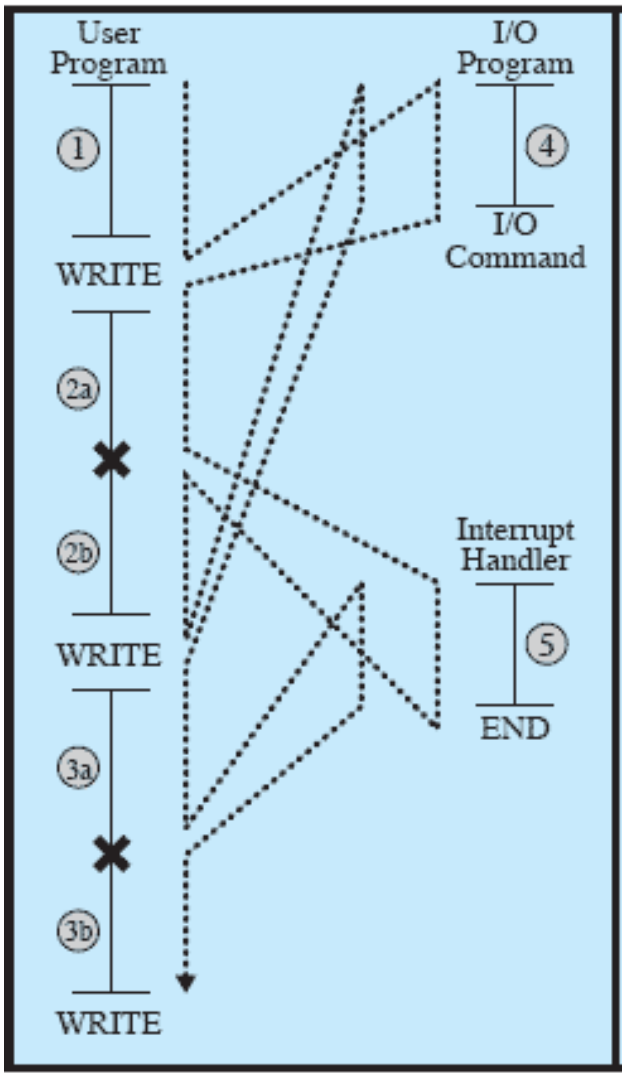


(a) No interrupts

Wait until completion Here.

Program Flow of Control

Interrupt with short I/O wait.

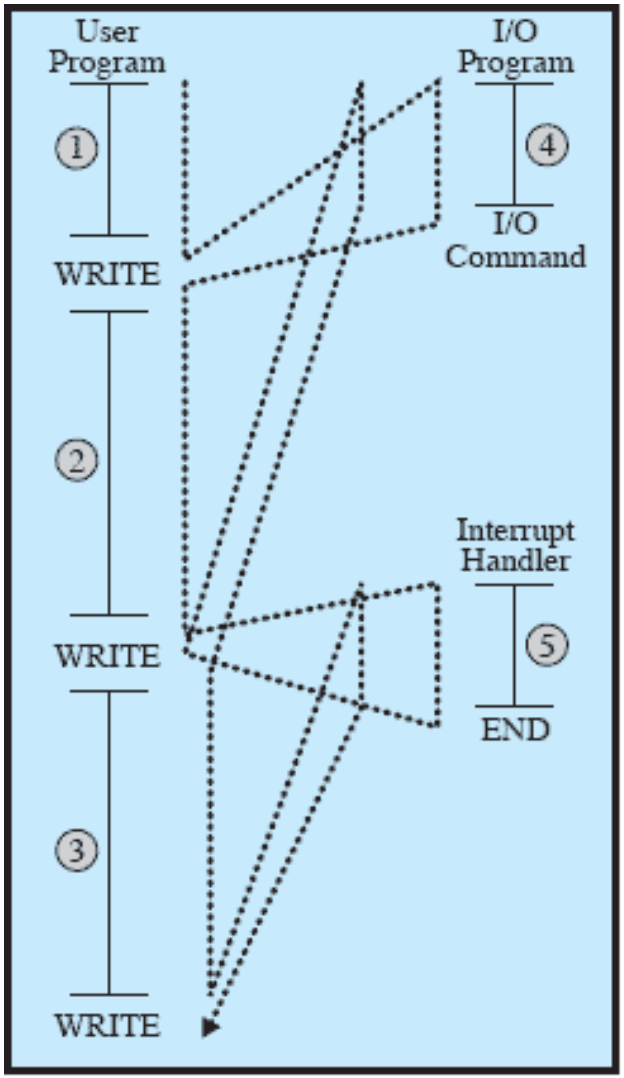


(b) Interrupts; short I/O wait

Program Flow of Control

Interupt with long I/O wait.

Need to wait for first I/O here.



(c) Interrupts; long I/O wait

Interrupt Stage

- Processor checks for interrupts
- If interrupt
  - Suspend execution of program (store snapshot!!)
  - Execute interrupt-handler routine
  - Resume execution of program
- This is a simplified version:
  - Nested interrupts?
  - Interrupt priorities?
  - Counting interrupts?

Transfer of Control via Interrupts

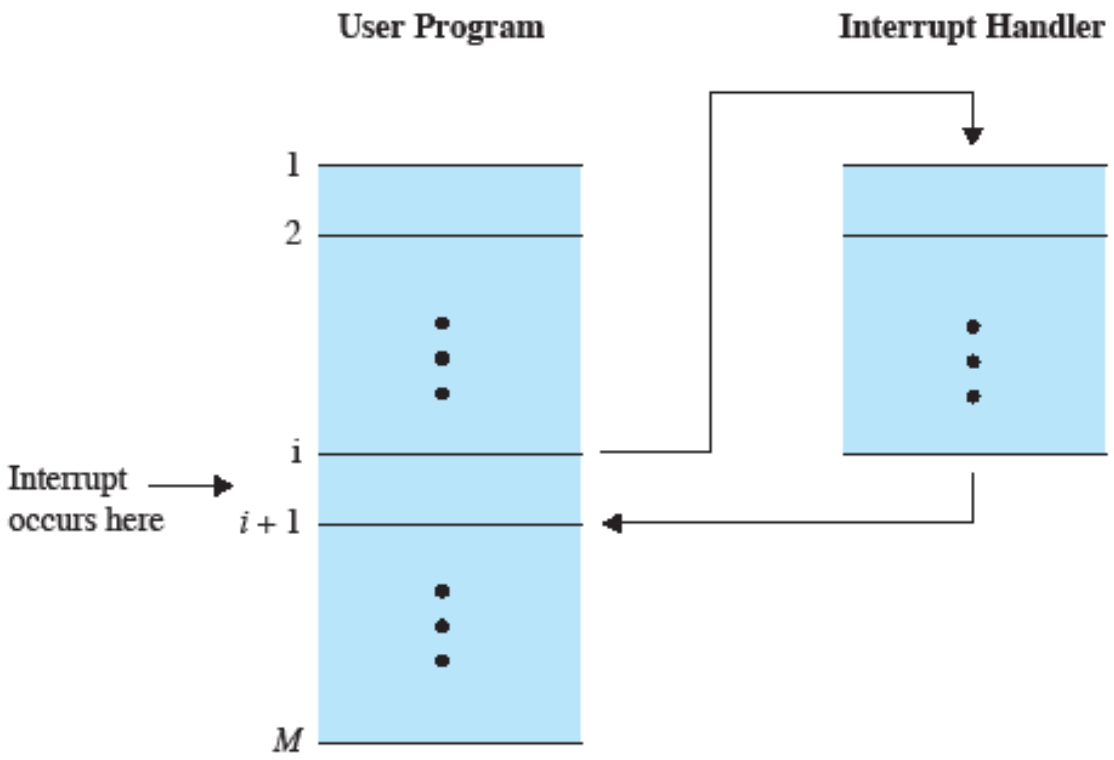


Figure 1.6 Transfer of Control via Interrupts

Instruction Cycle with Interrupts

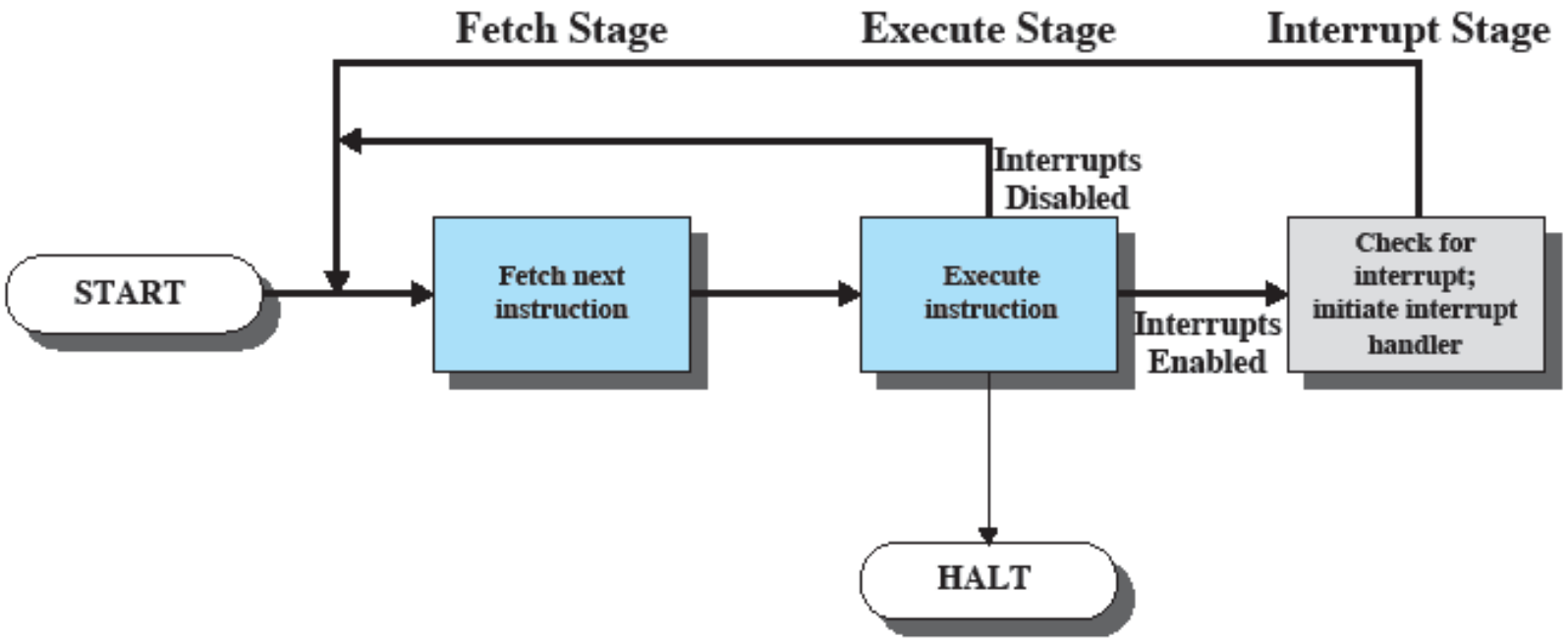


Figure 1.7 Instruction Cycle with Interrupts

Big advantage: no special code or programmer attention required.
Everything happens in hardware.

Program Timing: Short I/O Wait

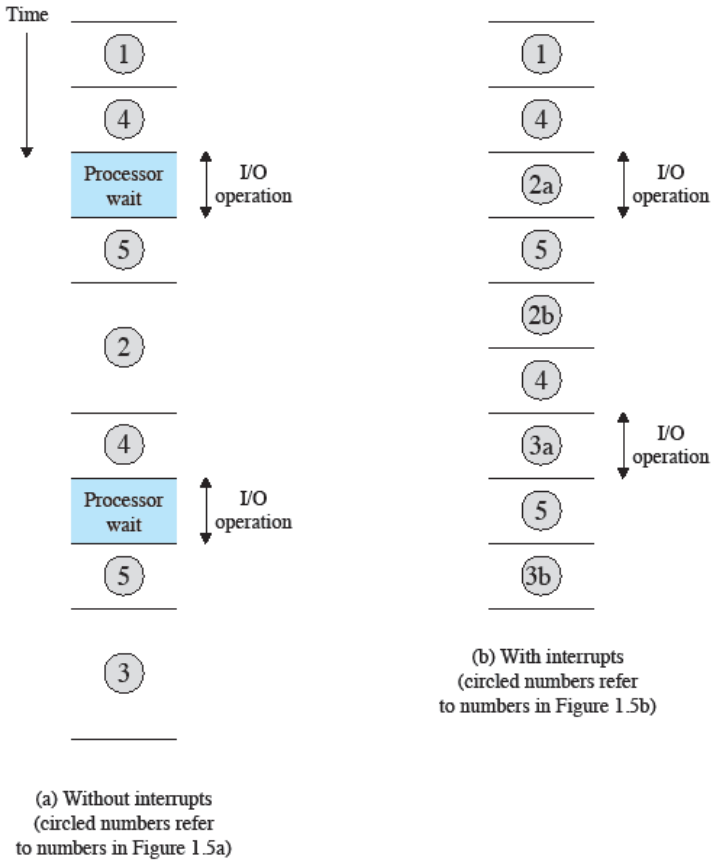


Figure 1.8 Program Timing: Short I/O Wait

Program Timing: Long I/O Wait

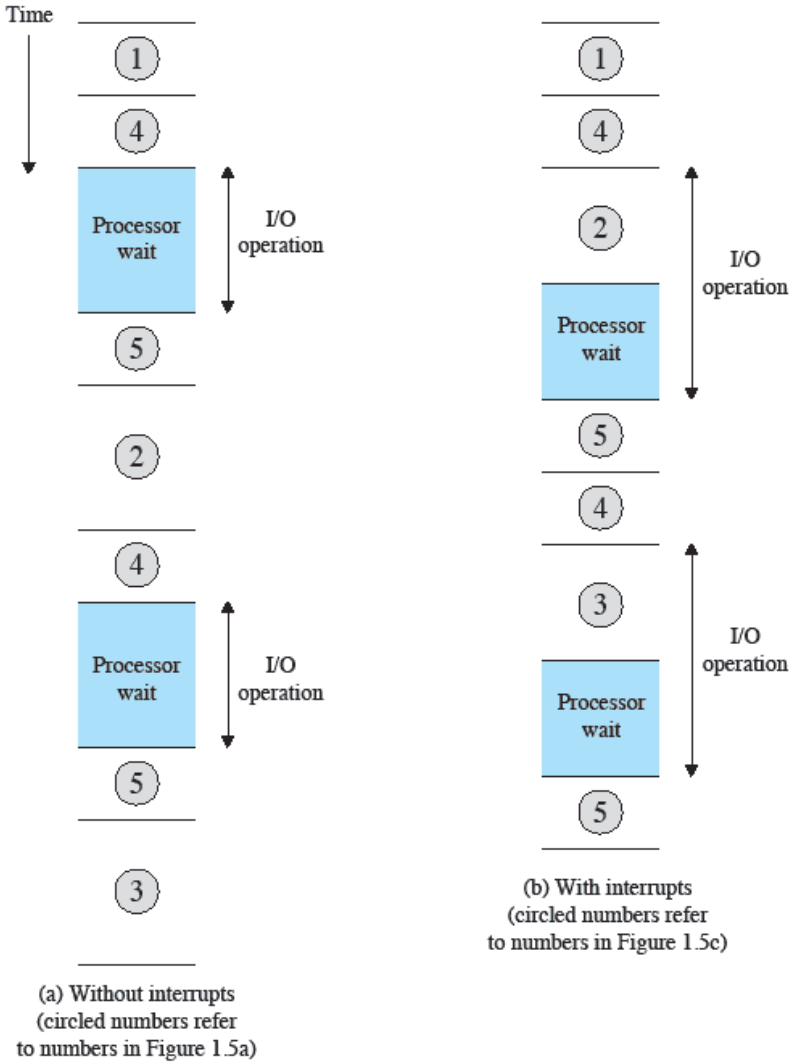
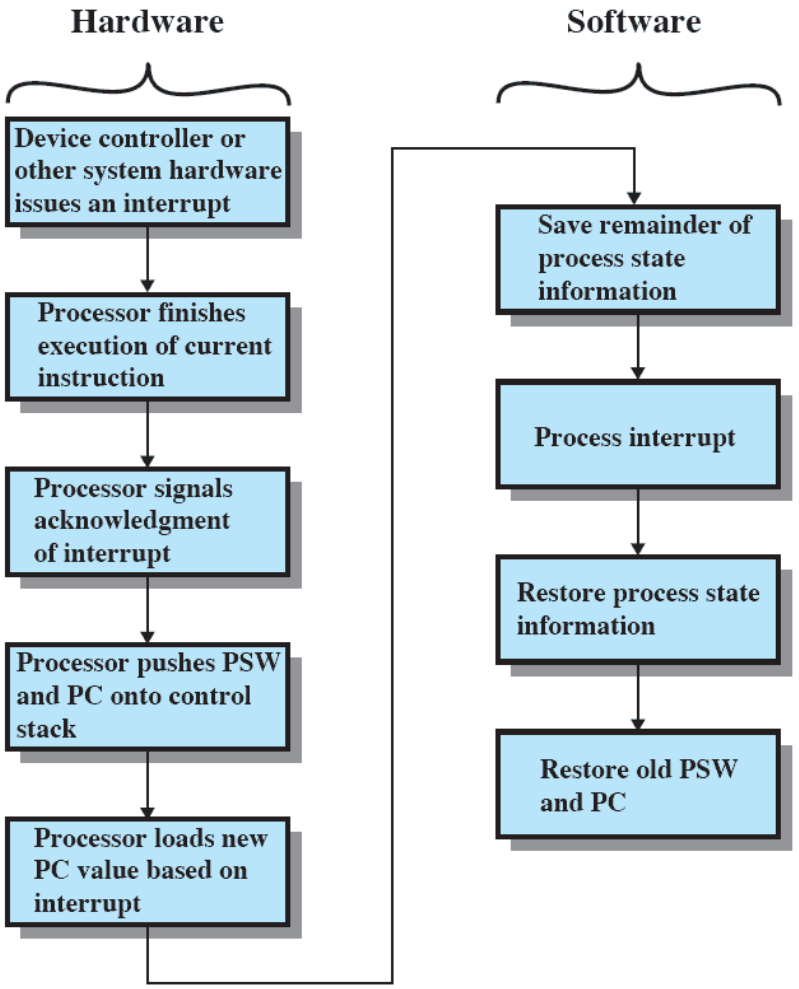


Figure 1.9 Program Timing: Long I/O Wait

Simple Interrupt Processing

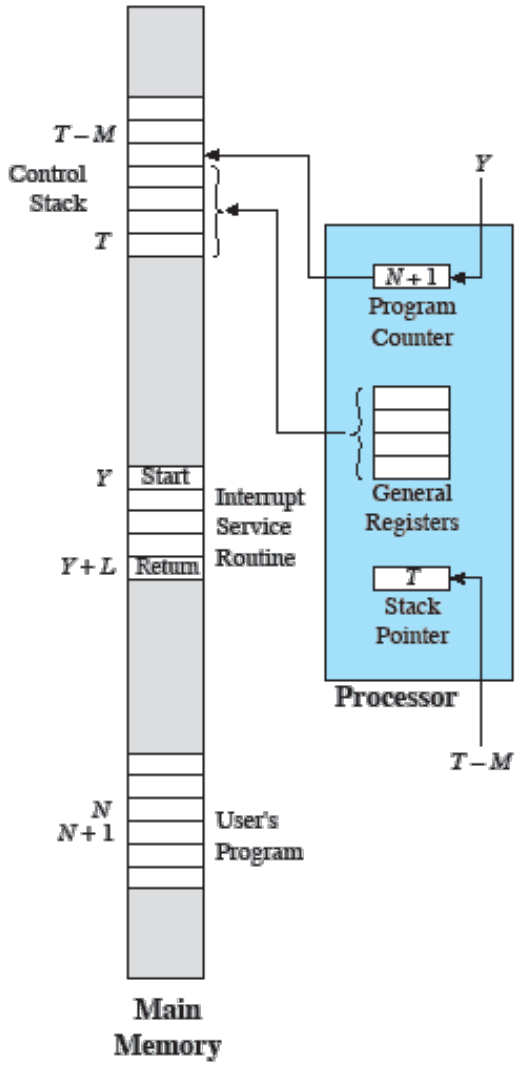


Discuss:
Lean ISR vs heavy ISR.

Figure 1.10 Simple Interrupt Processing

An Interrupt Occurs

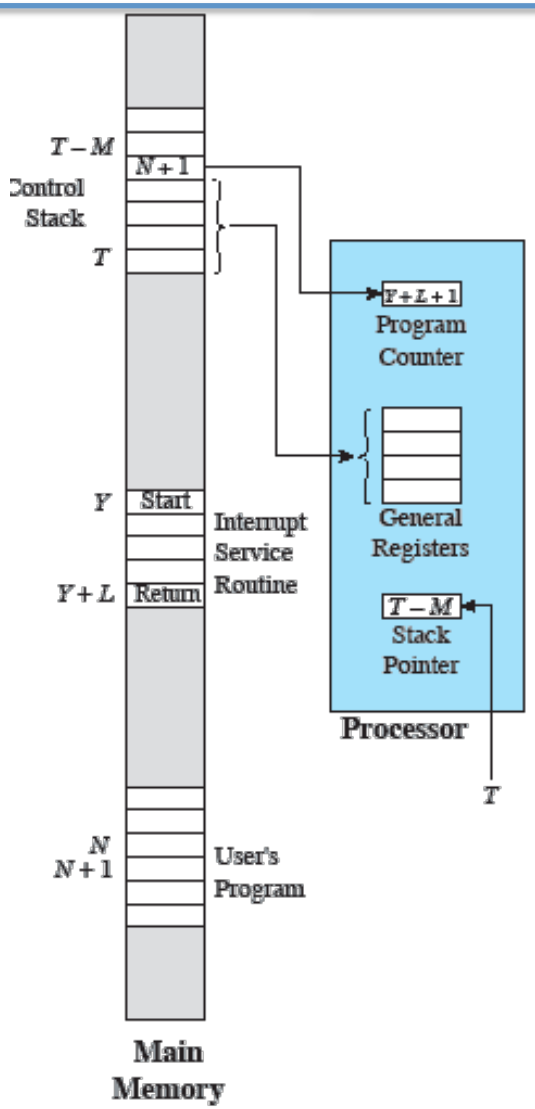
Storing a snapshot.



(a) Interrupt occurs after instruction at location N

Returning from an Interrupt

Restoring from a snapshot.

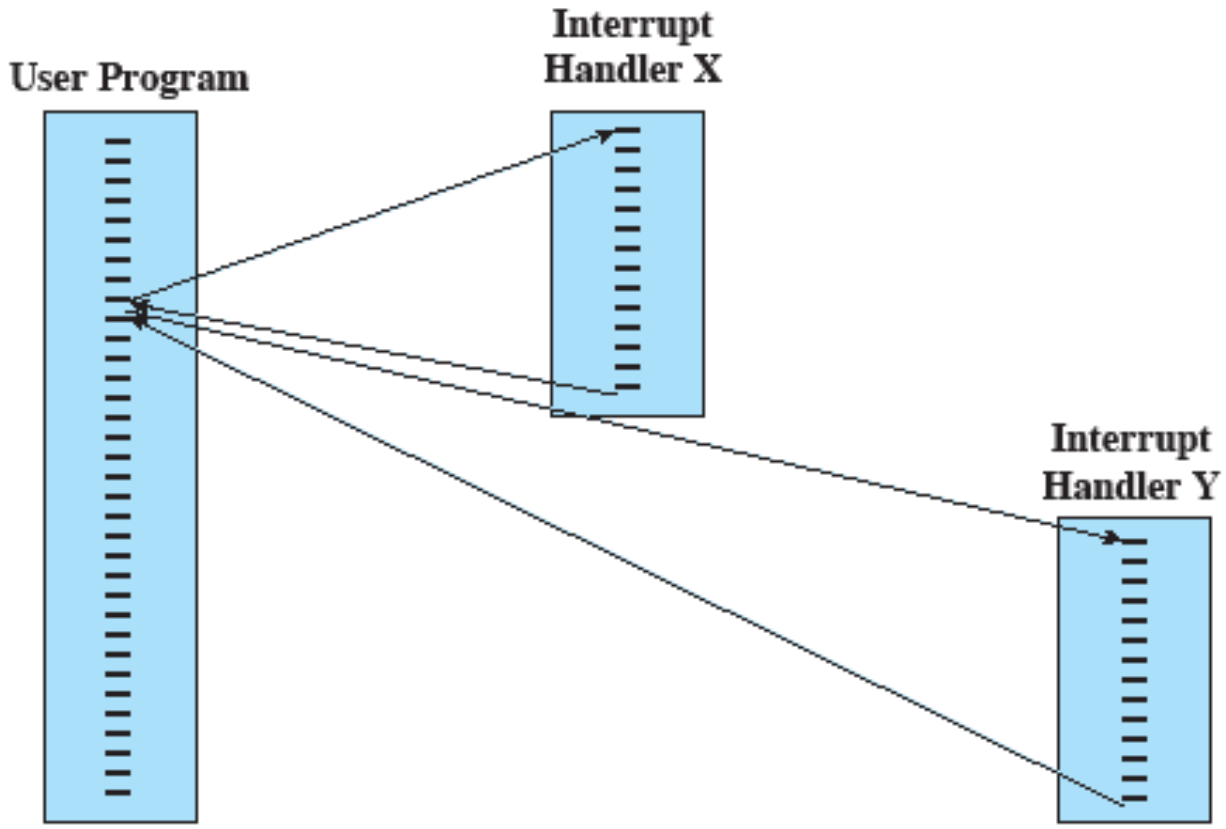


(b) Return from interrupt

Multiple Interrupts?

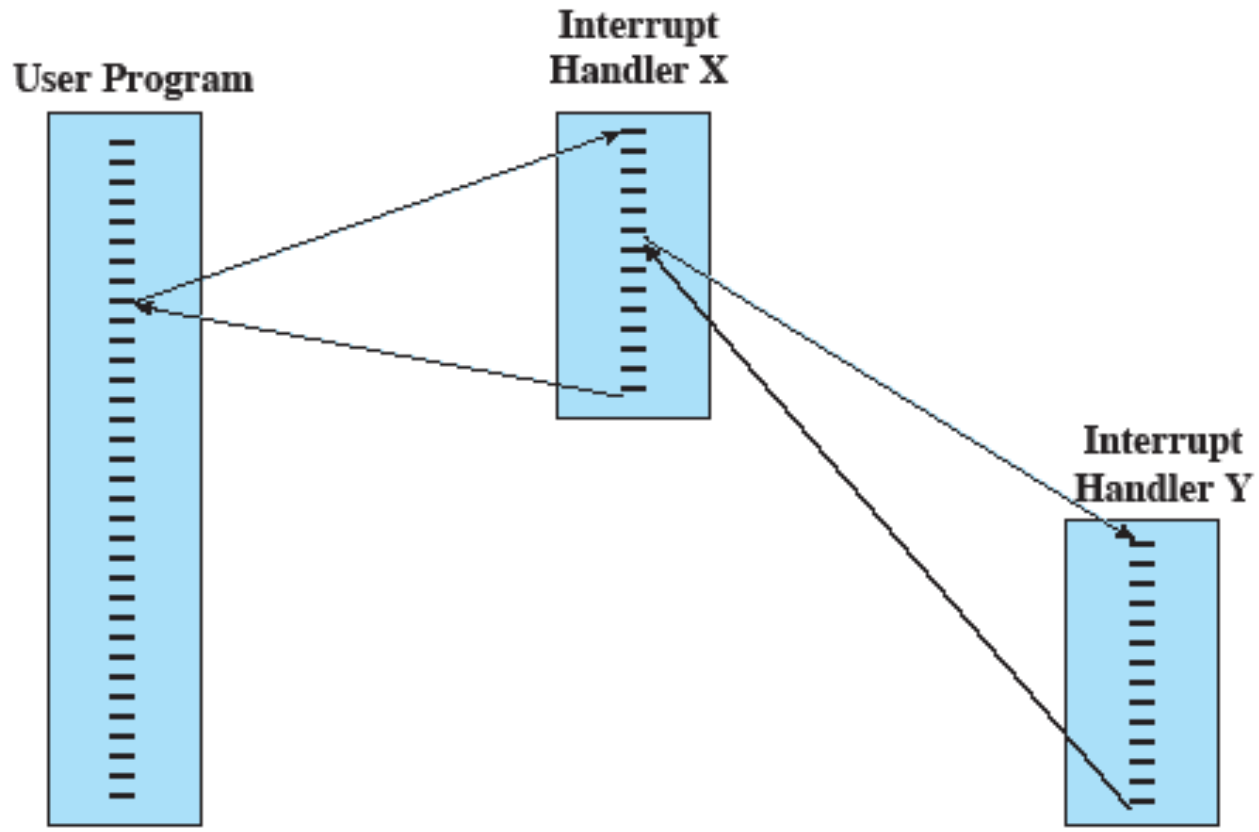
- Multiple I/O, timers, ADC, PWM settings, RS232, CAN, EEPROM, I2C, etc.
- Approach 1:
  - Disable interrupts in ISR
  - No priority information
  - Not useful for time critical elements
- Approach 2:
  - Priorities for ISR
  - How to assign priorities
  - finite number of priority levels

Sequential Interrupt Processing



(a) Sequential interrupt processing

Nested Interrupt Processing



(b) Nested interrupt processing

Multiprogramming

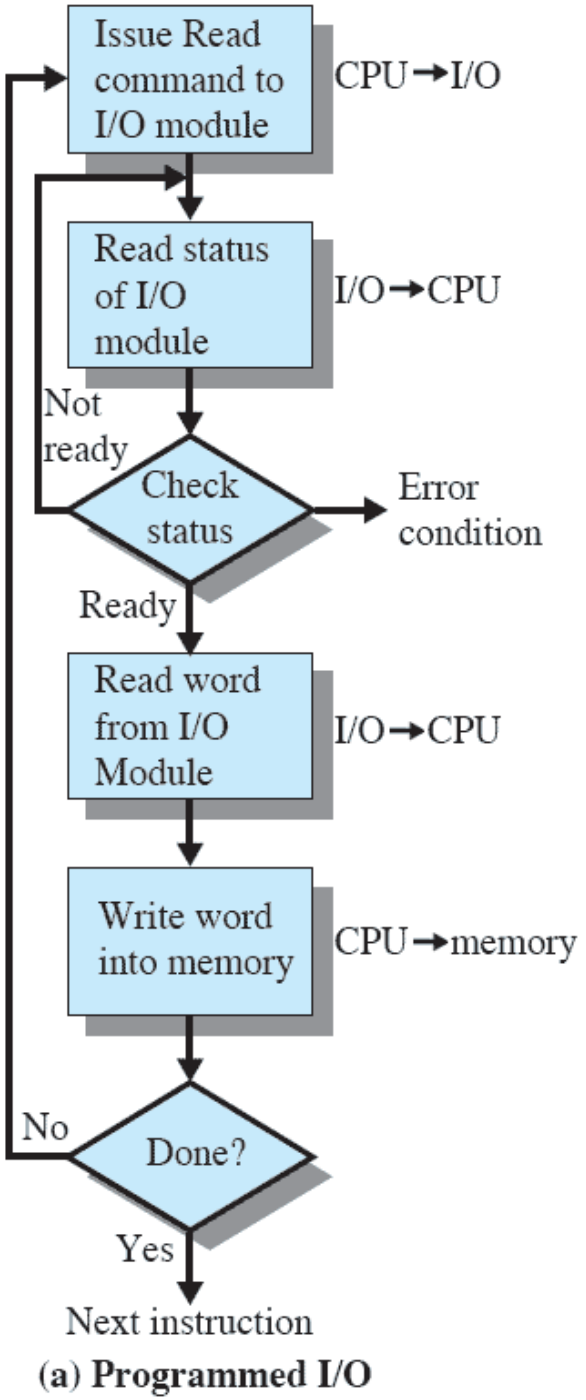
- Long I/O requests still leave the processor idle → run multiple programs
- Processor has more than one program to execute
- The sequence the programs are executed depend on their relative priority and whether they are waiting for I/O
- After an interrupt handler completes, control may not return to the program that was executing at the time of the interrupt

Memory Hierarchy

Self study

Programmed I/O

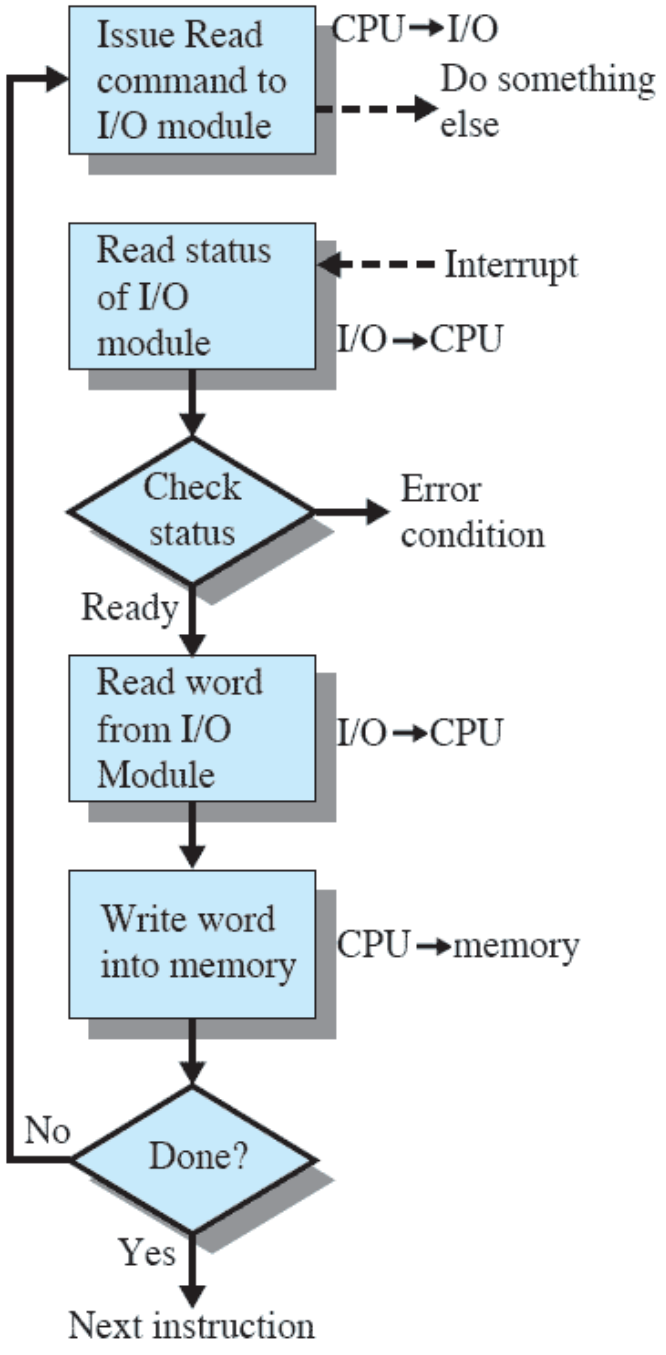
- I/O module performs the action, not the processor
- Sets the appropriate bits in the I/O status register
- No interrupts occur
- Processor checks status until operation is complete
- Called busy waiting, has benefits



(a) Programmed I/O

Interrupt-Driven I/O

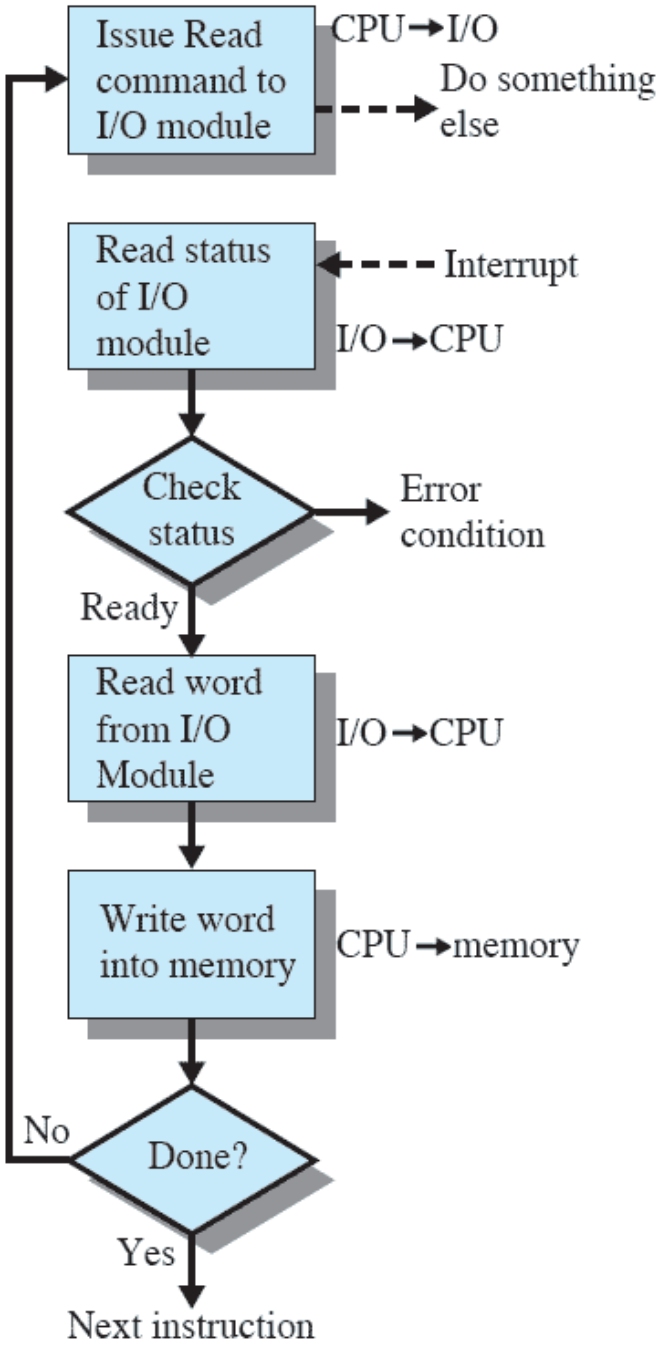
- Processor is interrupted when I/O module ready to exchange data
- Processor saves context o program executing and begins executing interrupt- handler



(b) Interrupt-driven I/O

Interrupt-Driven I/O

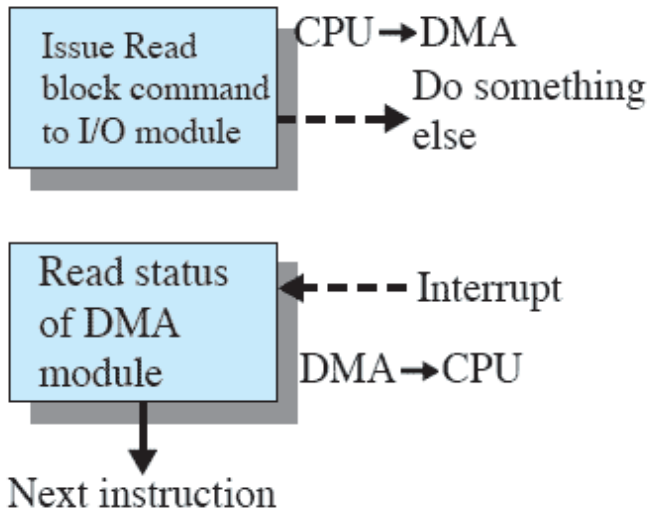
- No needless waiting
- Consumes a lot of processor time compared to DMA, because every word read or written passes through the processor



(b) Interrupt-driven I/O

Direct Memory Access

- Transfers a block of data directly to or from memory
- An interrupt is sent when the transfer is complete
- More efficient
- Not always available (e.g., external peripherals)



(c) Direct memory access