

# ***Computer System Overview***

Athar Mahboob  
MIS & CS Department  
Institute of Business Administration  
[athar@atharmahboob.com](mailto:athar@atharmahboob.com)  
<http://www.atharmahboob.com>

# *Operating System*

- ◆ Exploits the hardware resources of one or more processors
- ◆ Provides a set of services to system users
- ◆ Manages secondary memory and I/O devices

# *Basic Elements of a Computer System*

- ◆ Processor
- ◆ Main Memory
  - ◆ referred to as real memory or primary memory
  - ◆ volatile
- ◆ I/O modules
  - ◆ secondary memory devices
  - ◆ communications equipment
  - ◆ terminals
- ◆ System bus
  - ◆ communication among processors, memory, and I/O modules

# Top-Level Components

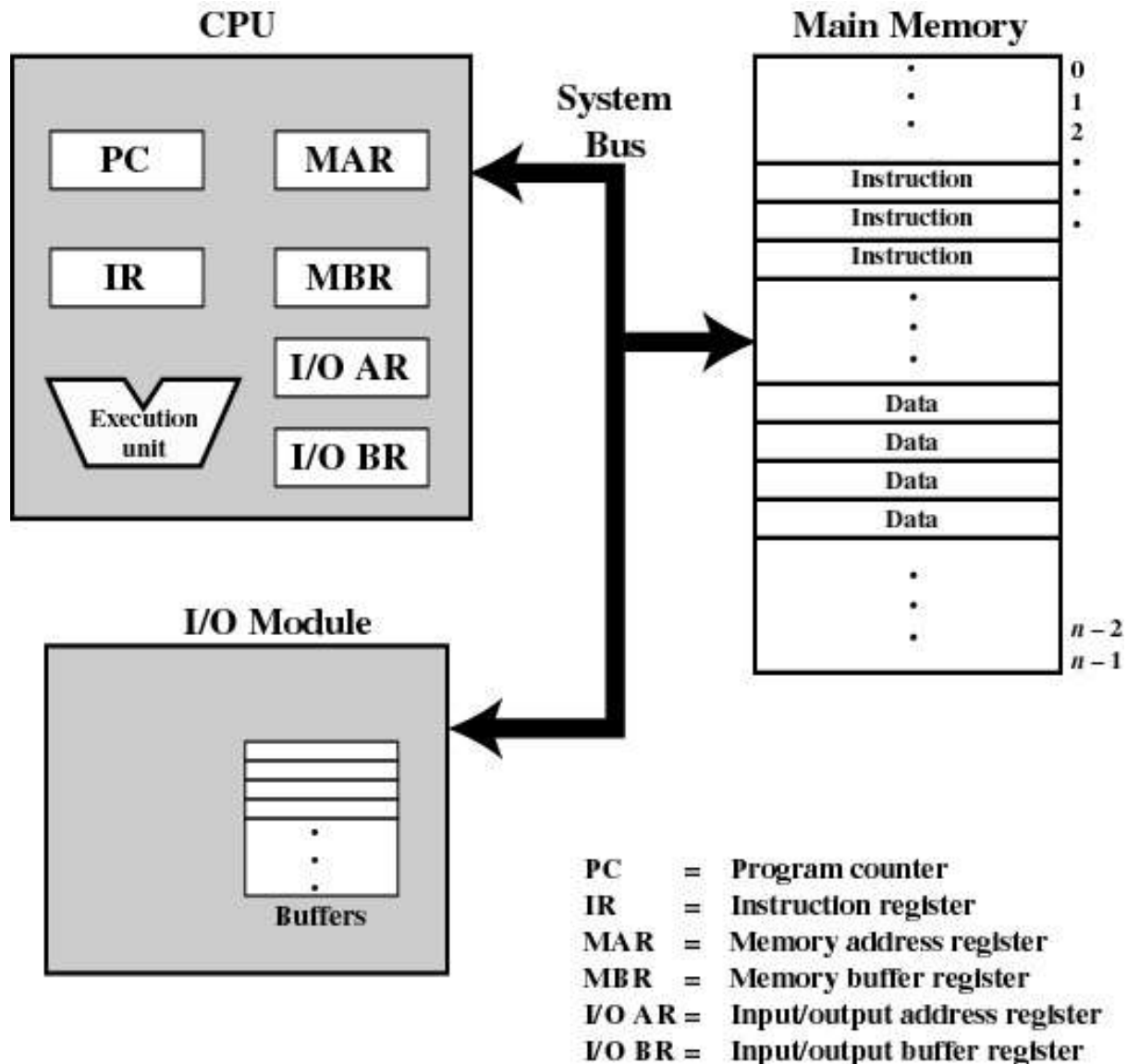


Figure 1.1 Computer Components: Top-Level View

# *Processor Registers*

- ◆ User-visible registers
  - ◆ Enable programmer to minimize main-memory references by optimizing register use
- ◆ Control and status registers
  - ◆ Used by processor to control operating of the processor
  - ◆ Used by operating-system routines to control the execution of programs

# *User-Visible Registers*

- ◆ May be referenced by machine language
- ◆ Available to all programs - application programs and system programs
- ◆ Types of registers
  - ◆ Data
  - ◆ Address
    - ◆ Index
    - ◆ Segment pointer
    - ◆ Stack pointer

# *User-Visible Registers*

- ◆ Address Registers
  - ◆ Index
    - ◆ involves adding an index to a base value to get an 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

# *Control and Status 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)
  - ◆ condition codes
  - ◆ Interrupt enable/disable
  - ◆ Supervisor/user mode

# *Control and Status Registers*

- ◆ Condition Codes or Flags
  - ◆ Bits set by the processor hardware as a result of operations
  - ◆ Can be accessed by a program but not altered
  - ◆ Examples
    - ◆ positive result
    - ◆ negative result
    - ◆ zero
    - ◆ Overflow

# *Instruction Cycle*

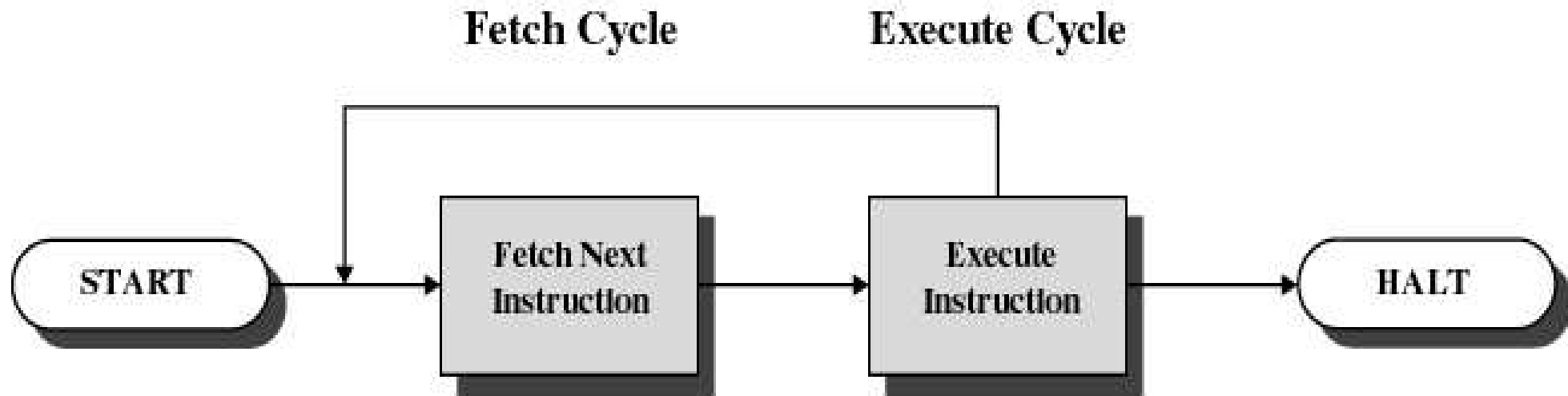


Figure 1.2 Basic Instruction Cycle

# *Instruction Fetch and Execute*

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

# *Instruction Register*

- ◆ Fetched instruction is placed in the instruction register
- ◆ Types of instructions
  - ◆ Processor-memory
    - ◆ transfer data between processor and memory
  - ◆ Processor-I/O
    - ◆ data transferred to or from a peripheral device
  - ◆ Data processing
    - ◆ arithmetic or logic operation on data
  - ◆ Control
    - ◆ alter sequence of execution

# Example of Program Execution

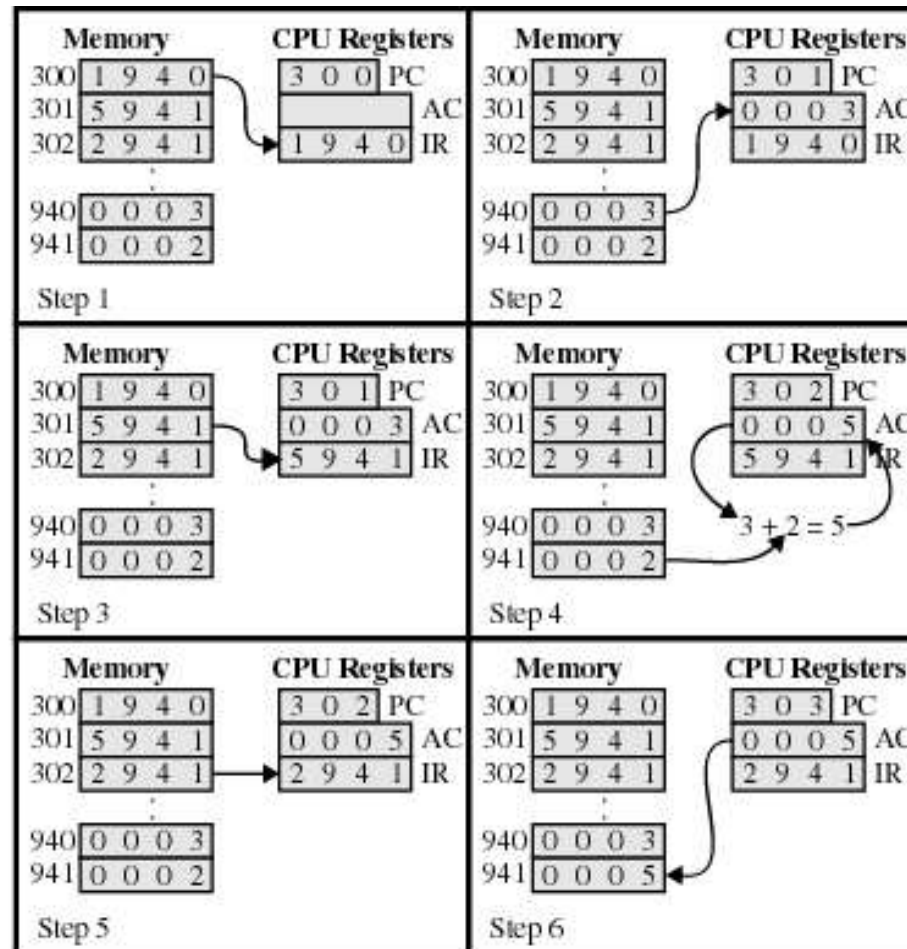


Figure 1.4 Example of Program Execution  
(contents of memory and registers in hexadecimal)

# *Direct Memory Access (DMA)*

- ◆ I/O exchanges occur directly with memory
- ◆ Processor grants I/O module authority to read from or write to memory
- ◆ Relieves the processor responsibility for the exchange
- ◆ Processor is free to do other things

# *Interrupts*

- ◆ An interruption of the normal sequence of execution
- ◆ Improves processing efficiency
- ◆ Allows the processor to execute other instructions while an I/O operation is in progress
- ◆ A suspension of a process caused by an event external to that process and performed in such a way that the process can be resumed

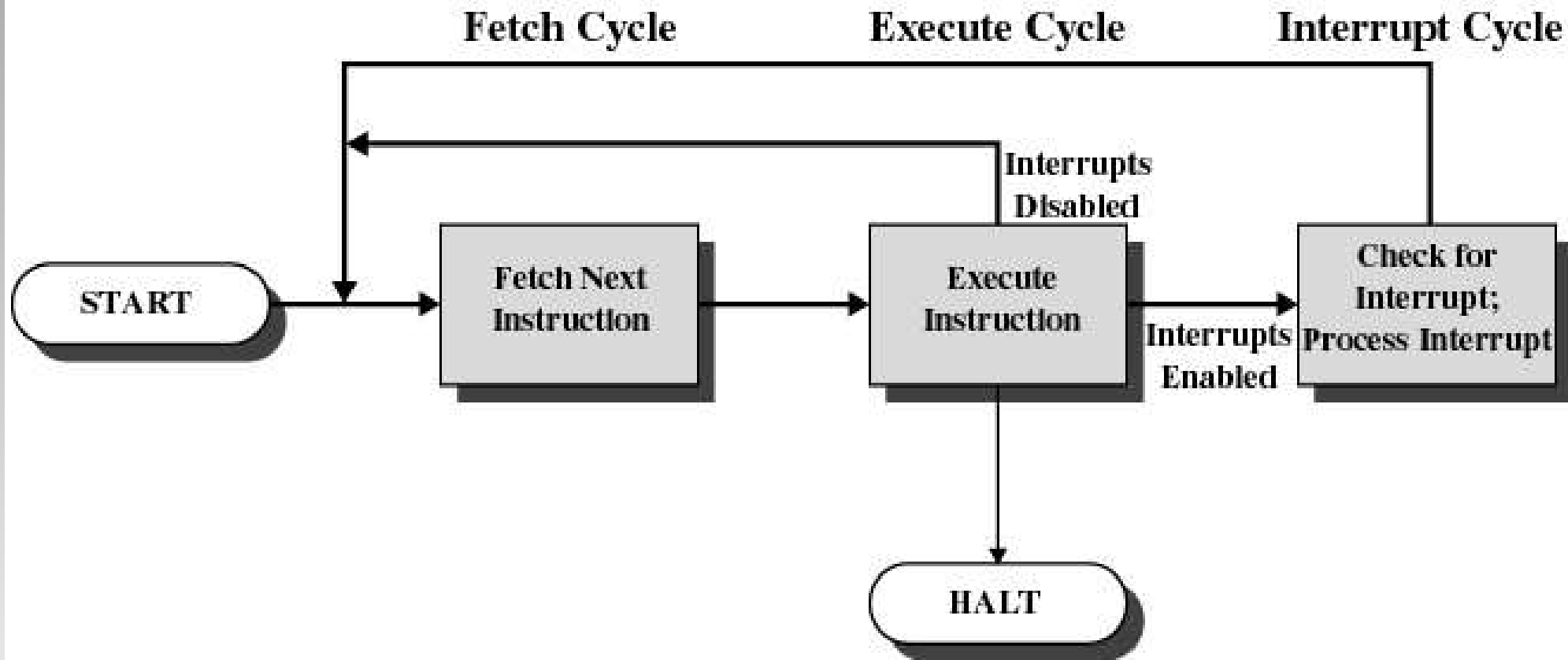
# *Classes of Interrupts*

- ◆ Program
  - ◆ arithmetic overflow
  - ◆ division by zero
  - ◆ execute illegal instruction
  - ◆ reference outside user's memory space
- ◆ Timer
- ◆ I/O
- ◆ Hardware failure

# *Interrupt Handler*

- ◆ A program that determines nature of the interrupt and performs whatever actions are needed
- ◆ Control is transferred to this program
- ◆ Generally part of the operating system

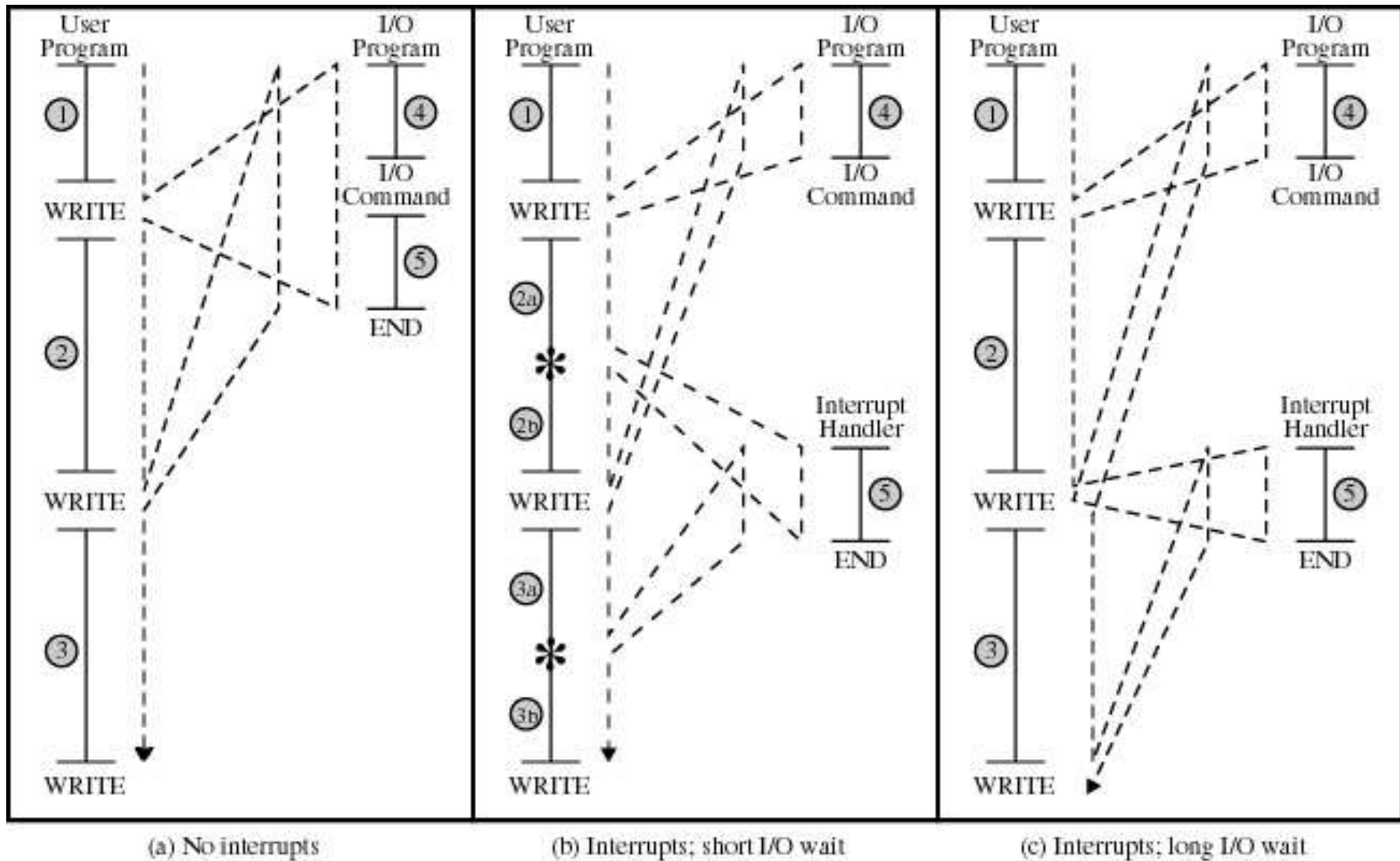
# *Interrupt Cycle*



**Figure 1.7** Instruction Cycle with Interrupts

# *Interrupt Cycle*

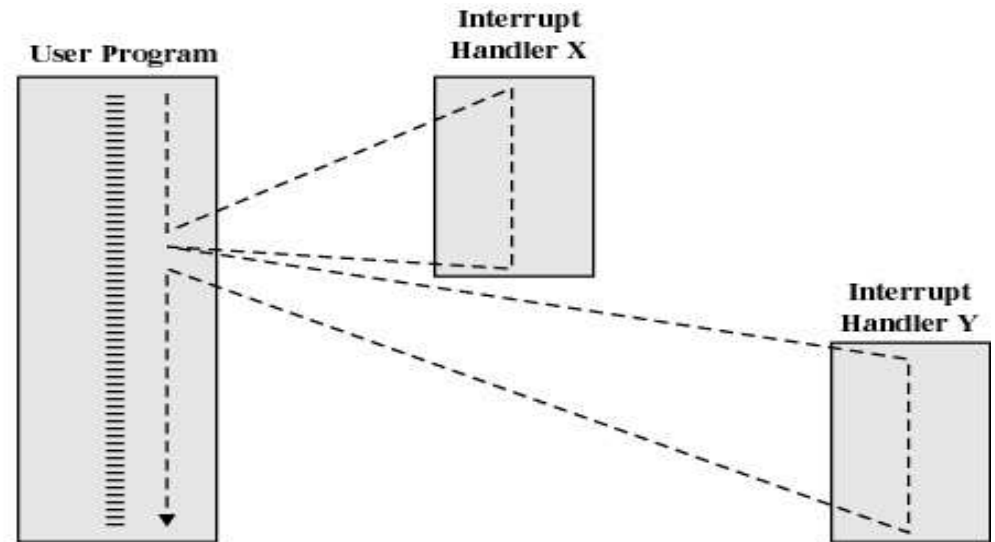
- ◆ Processor checks for interrupts
- ◆ If no interrupts fetch the next instruction for the current program
- ◆ If an interrupt is pending, suspend execution of the current program, and execute the interrupt handler



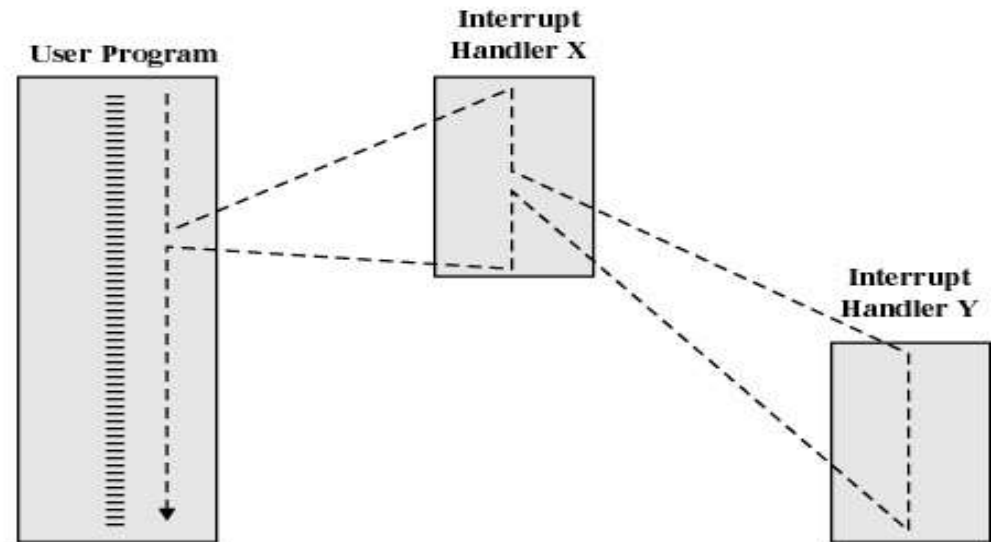
**Figure 1.5 Program Flow of Control Without and With Interrupts**

# Multiple Interrupts

- ◆ Disable interrupts while an interrupt is being processed
- ◆ Processor ignores any new interrupt request signals



(a) Sequential Interrupt processing



(b) Nested Interrupt processing

Figure 1.12 Transfer of Control with Multiple Interrupts

# *Multiple Interrupts Sequential Order*

- ◆ Disable interrupts so processor can complete task
- ◆ Interrupts remain pending until the processor enables interrupts
- ◆ After interrupt handler routine completes, the processor checks for additional interrupts

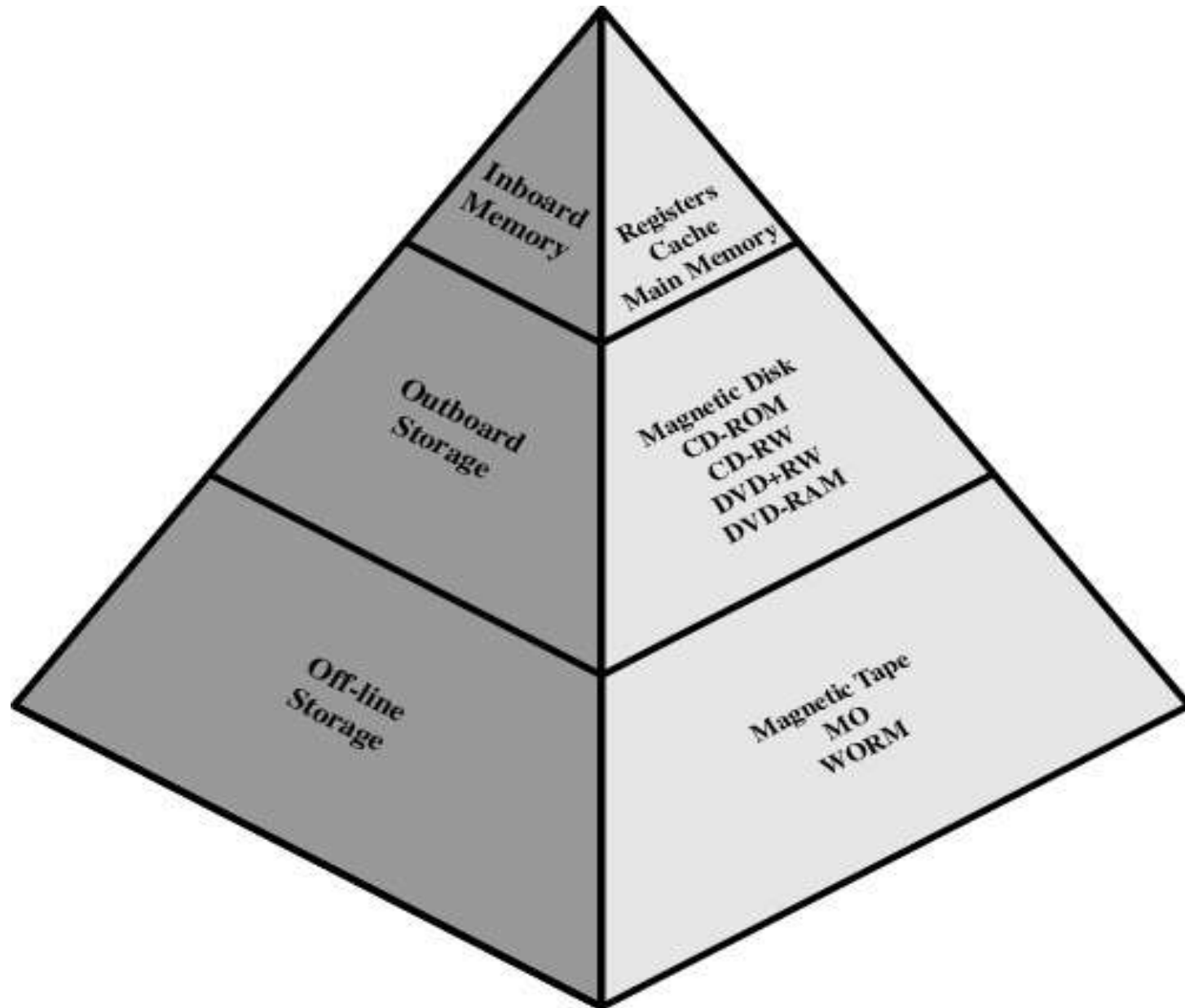
# *Multiple Interrupts Priorities*

- ◆ Higher priority interrupts cause lower-priority interrupts to wait
- ◆ Causes a lower-priority interrupt handler to be interrupted
- ◆ Example when input arrives from communication line, it needs to be absorbed quickly to make room for more input

# *Multiprogramming*

- ◆ 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*



# *Going Down the Hierarchy*

- ◆ Decreasing cost per bit
- ◆ Increasing capacity
- ◆ Increasing access time
- ◆ Decreasing frequency of access of the memory by the processor
  - ◆ locality of reference

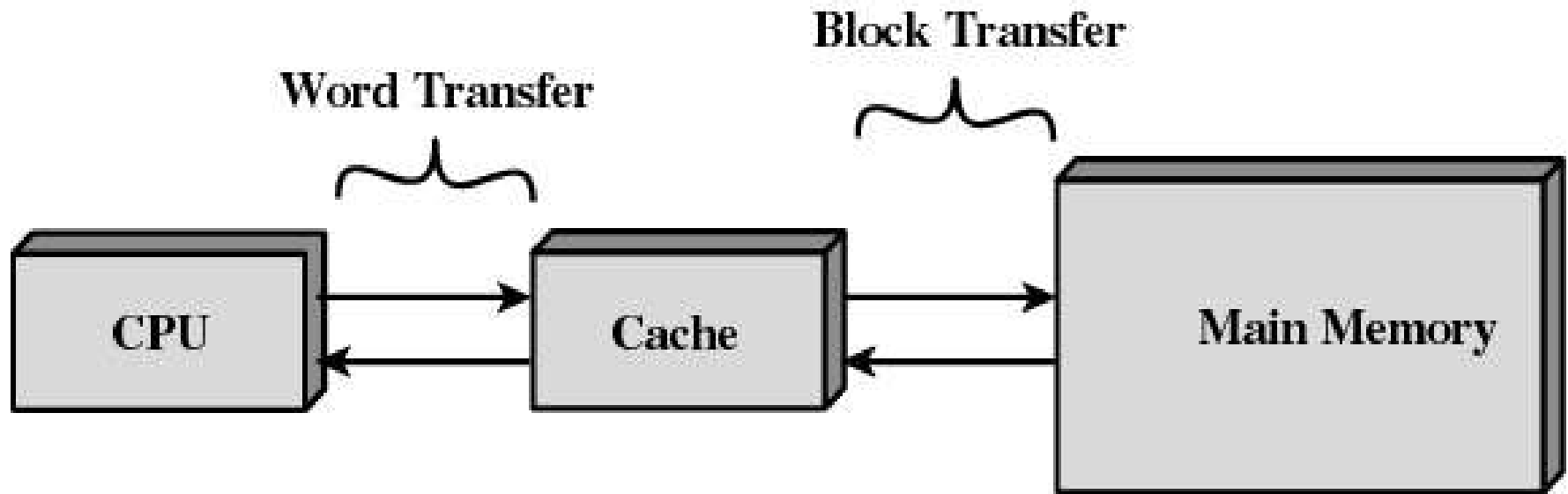
# *Disk Cache*

- ◆ A portion of main memory used as a buffer to temporarily to hold data for the disk
- ◆ Disk writes are clustered
- ◆ Some data written out may be referenced again. The data are retrieved rapidly from the software cache instead of slowly from disk

# *Cache Memory*

- ◆ Invisible to operating system
- ◆ Increase the speed of memory
- ◆ Processor speed is faster than memory speed

# *Cache Memory*



**Figure 1.16 Cache and Main Memory**

# *Cache Memory*

- ◆ Contains a portion of main memory
- ◆ Processor first checks cache
- ◆ If not found in cache, the block of memory containing the needed information is moved to the cache



# *Cache Design*

- ◆ Cache size
  - ◆ small caches have a significant impact on performance
- ◆ Block size
  - ◆ the unit of data exchanged between cache and main memory
  - ◆ hit means the information was found in the cache
  - ◆ larger block size more hits until probability of using newly fetched data becomes less than the probability of reusing data that has been moved out of cache

# *Cache Design*

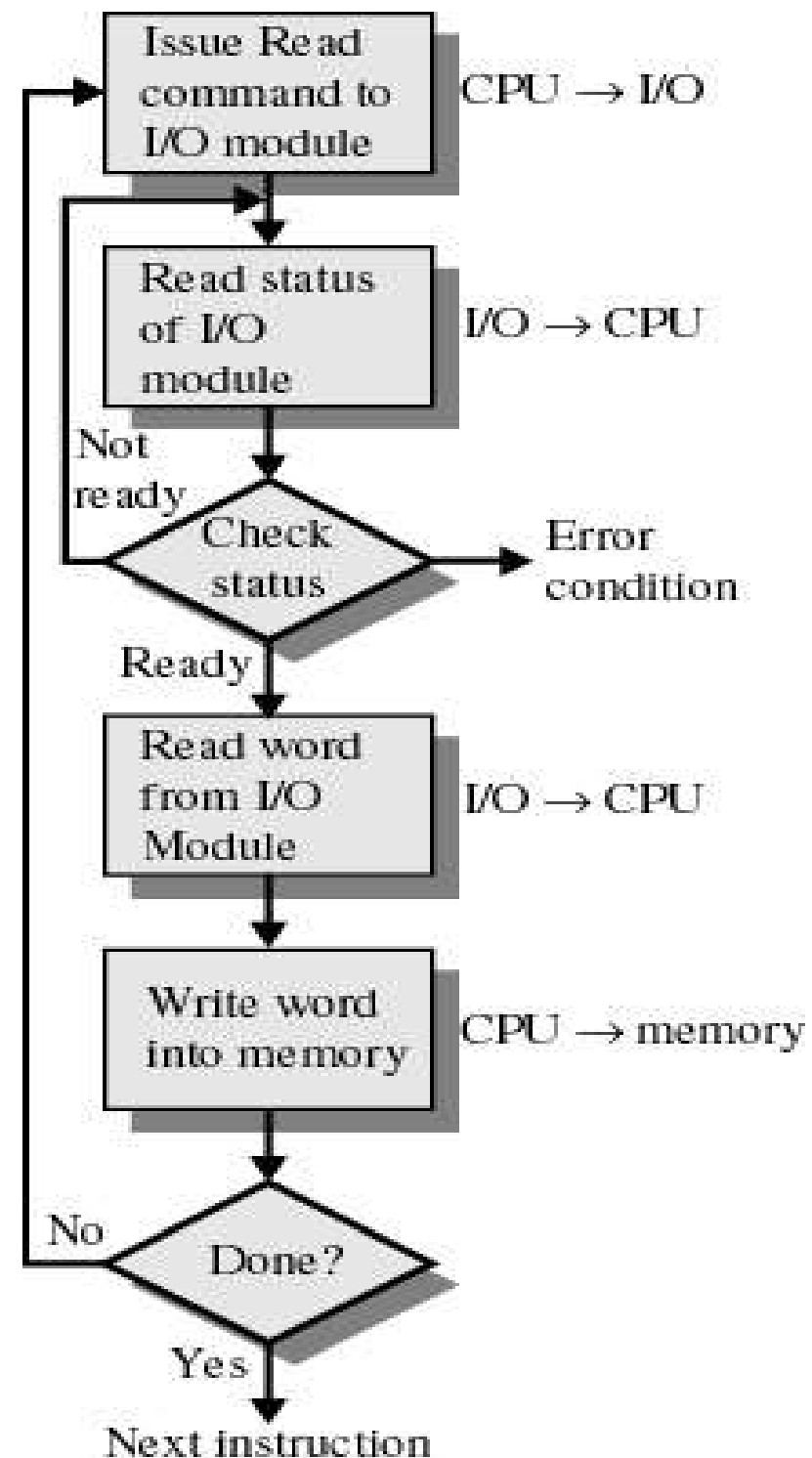
- ◆ Mapping function
  - ◆ determines which cache location the block will occupy
- ◆ Replacement algorithm
  - ◆ determines which block to replace
  - ◆ Least-Recently-Used (LRU) algorithm

# *Cache Design*

- ◆ Write policy
  - ◆ When the memory write operation takes place
  - ◆ Can occur every time block is updated
  - ◆ Can occur only when block is replaced
    - ◆ Minimizes memory operations
    - ◆ Leaves memory in an obsolete state

# Programmed I/O

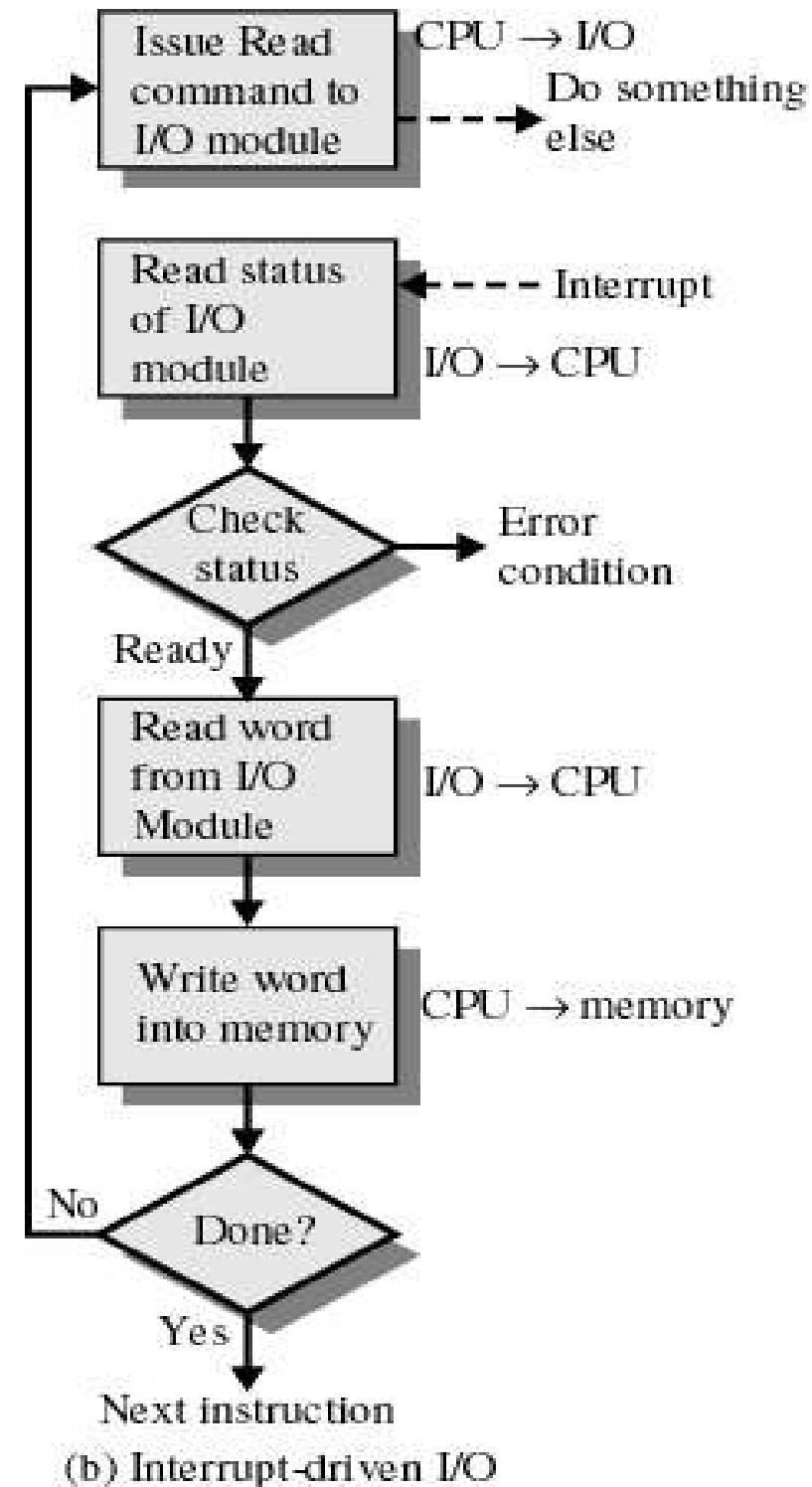
- ◆ I/O module performs the action, not the processor
- ◆ Sets appropriate bits in the I/O status register
- ◆ No interrupts occur
- ◆ Processor checks status until operation is complete



(a) Programmed I/O

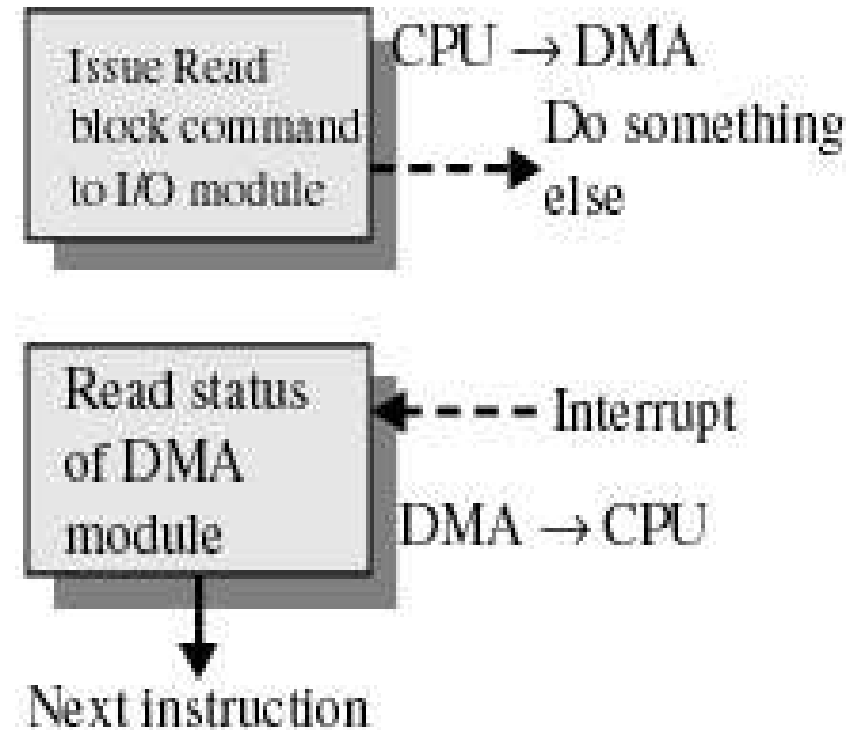
# Interrupt-Driven I/O

- ◆ Processor is interrupted when I/O module ready to exchange data
- ◆ Processor is free to do other work
- ◆ No needless waiting
- ◆ Consumes a lot of processor time because every word read or written passes through the processor



# Direct Memory Access

- ◆ Transfers a block of data directly to or from memory
- ◆ An interrupt is sent when the task is complete
- ◆ The processor is only involved at the beginning and end of the transfer



(c) Direct memory access