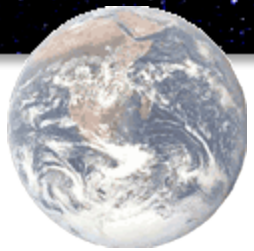
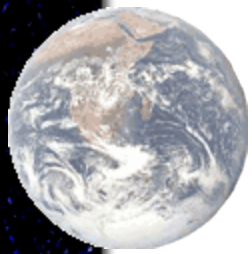
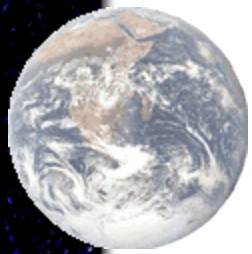


# Course Overview



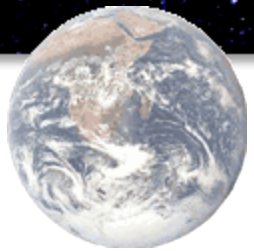
# Pagina & Materiale

- http://wwwusers.di.uniroma1.it/~stefa/Sistemi_Operativi/Home_Page.html
- Libro: W. Stallings, "Operating Systems", 7th ed., Prentice-Hall.
- Meglio in Inglese ☺



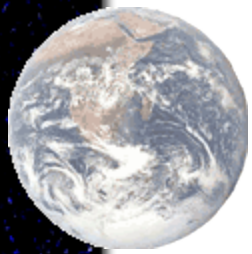
# Esame finale:

- Due Esoneri
- Esercizi di livelli diversi di difficolta'
- Passare gli esoneri = passare l'esame!
- Orale obbligatorio per chi non passa uno o entrambi gli esoneri



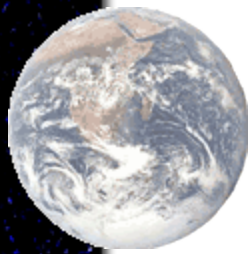
# Computer System Overview

## Chapter 1



# 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

- Processor
- 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, memory, and I/O modules

# Top-Level Components

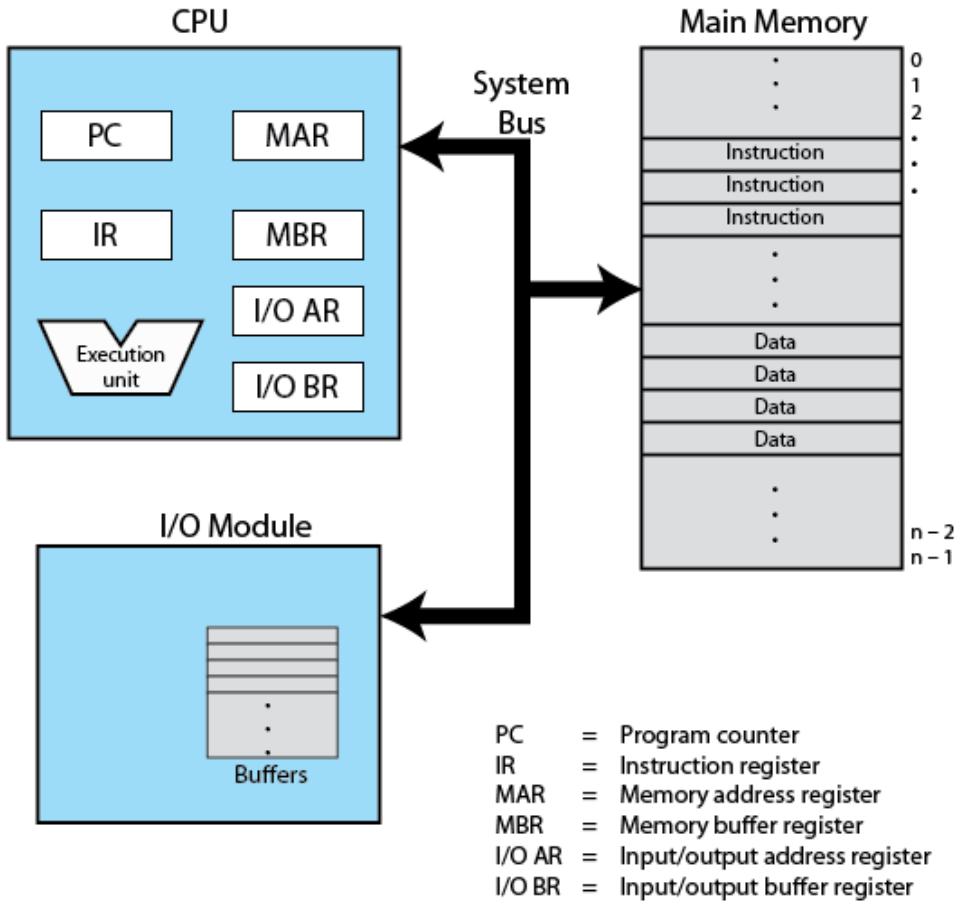
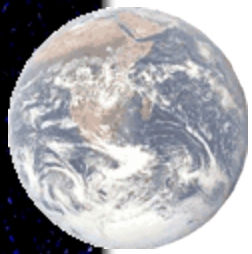
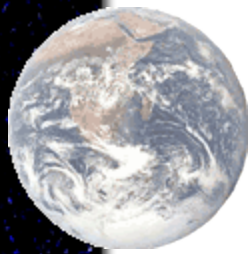



Figure 1.1 Computer Components: Top-Level View



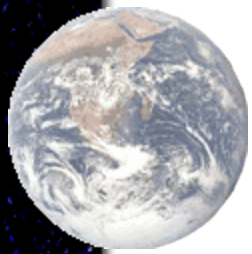
# Processor

- Two internal registers
  - Memory address register (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



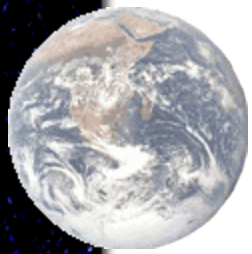
# Processor Registers

- 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 privileged operating-system routines to control the execution of programs



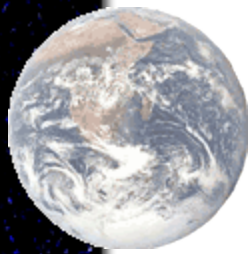
# User-Visible Registers

- 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



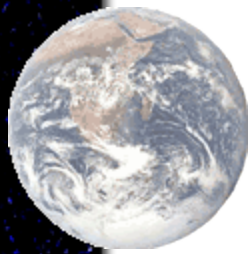
# User-Visible Registers

- 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



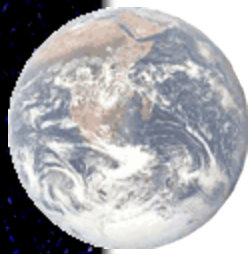
# Control and Status Registers

- 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
  - Examples
    - Positive result
    - Negative result
    - Zero
    - Overflow



# Instruction Execution

- Two steps
  - Processor reads instructions from memory
    - Fetches
  - Processor executes each instruction

# Instruction Cycle

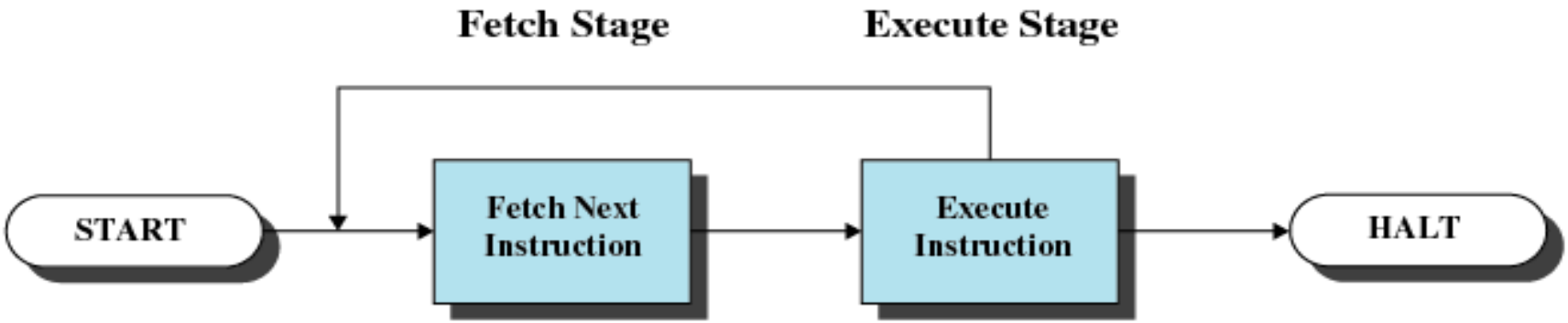
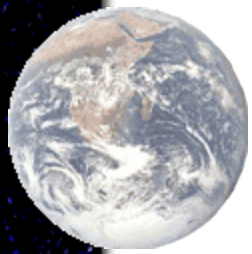
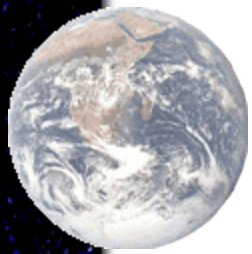



**Figure 1.2 Basic Instruction Cycle**



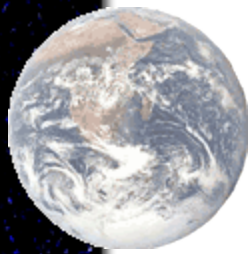
# Instruction Fetch and Execute

- 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



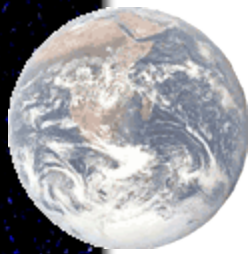
# Instruction Register

- Fetched instruction is placed in the instruction register
- Categories
  - 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



# Interrupts

- Interrupt the normal sequencing of the processor
- Most I/O devices are slower than the processor
  - Processor must pause to wait for device

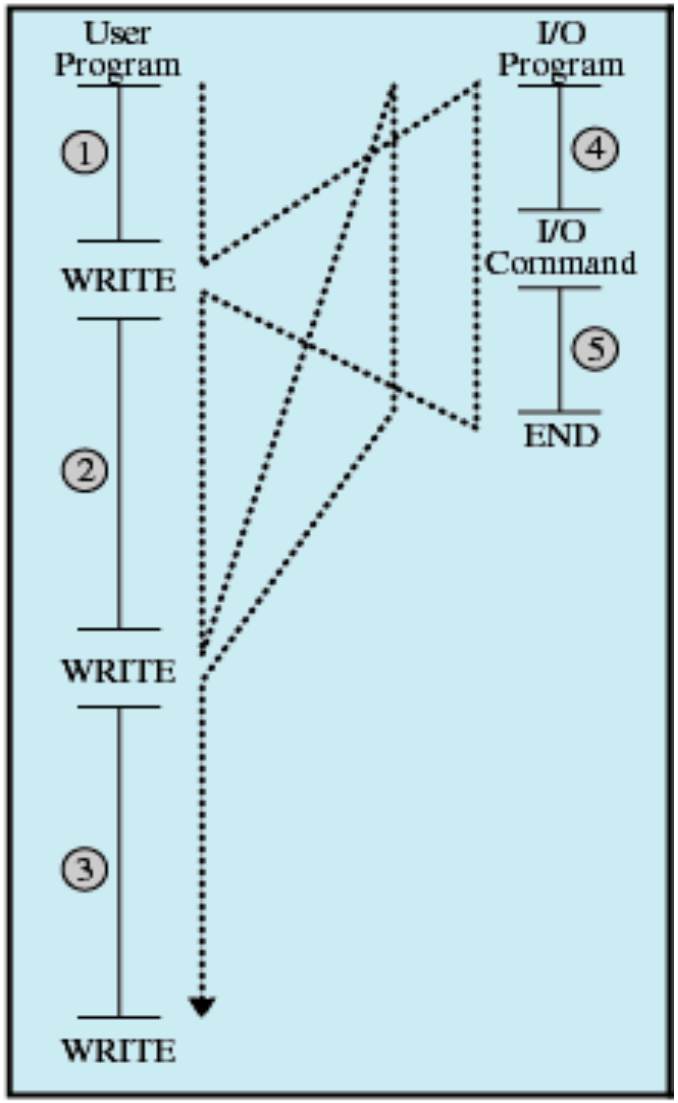


# Classes of Interrupts

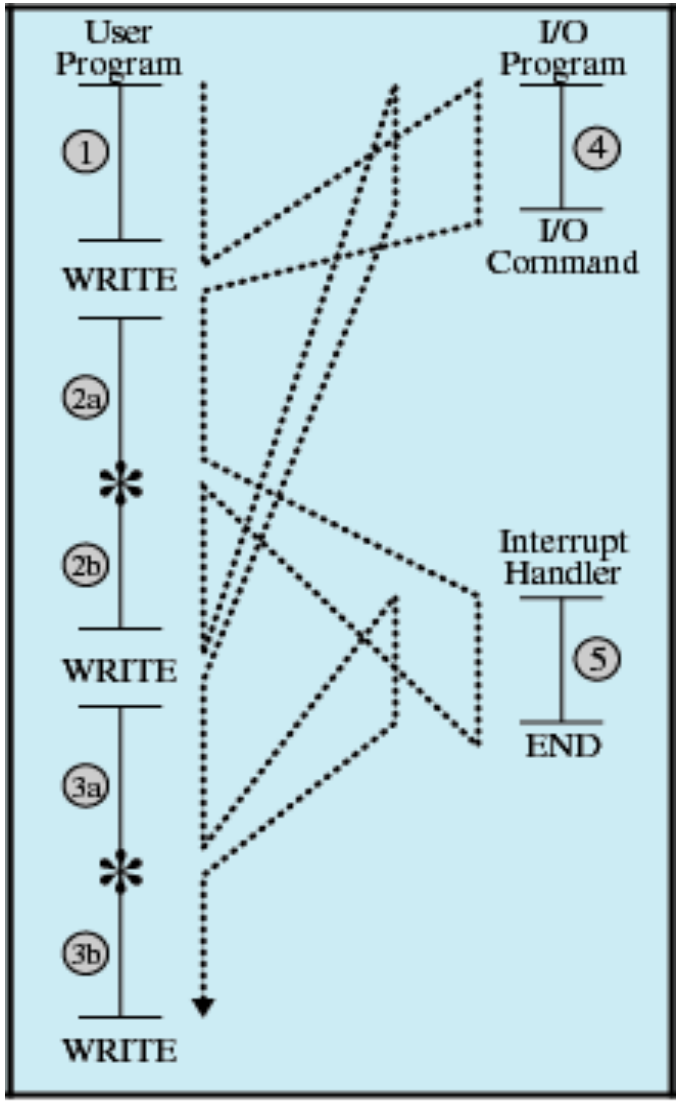
**Table 1.1 Classes of Interrupts**

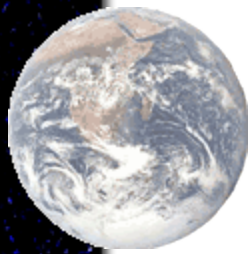
| | |
|---|---|
| **Program** | Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, and reference outside a user's allowed memory space. |
| **Timer** | Generated by a timer 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. |

# Program Flow of Control Without Interrupts



# Program Flow of Control With Interrupts, Short I/O Wait



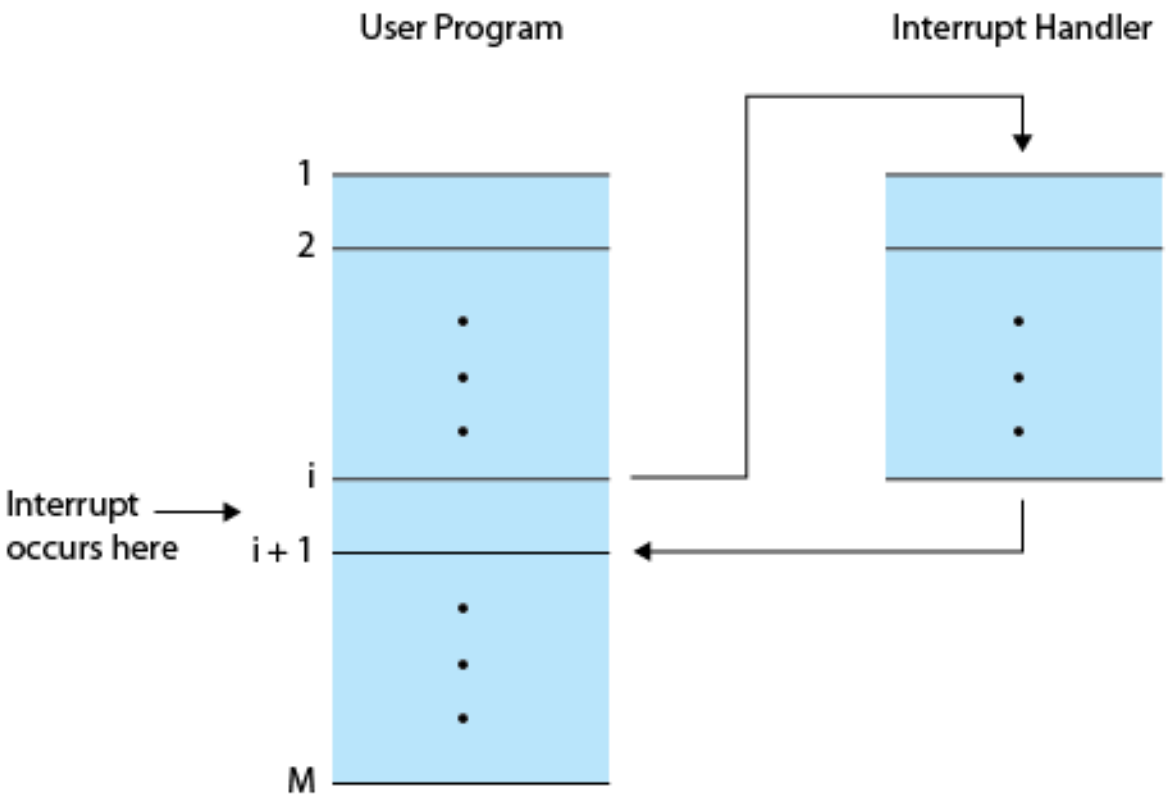


# Interrupt Handler

- Program to service a particular I/O device
- Generally part of the operating system

# Interrupts

- Suspends the normal sequence of execution



# Interrupt Cycle

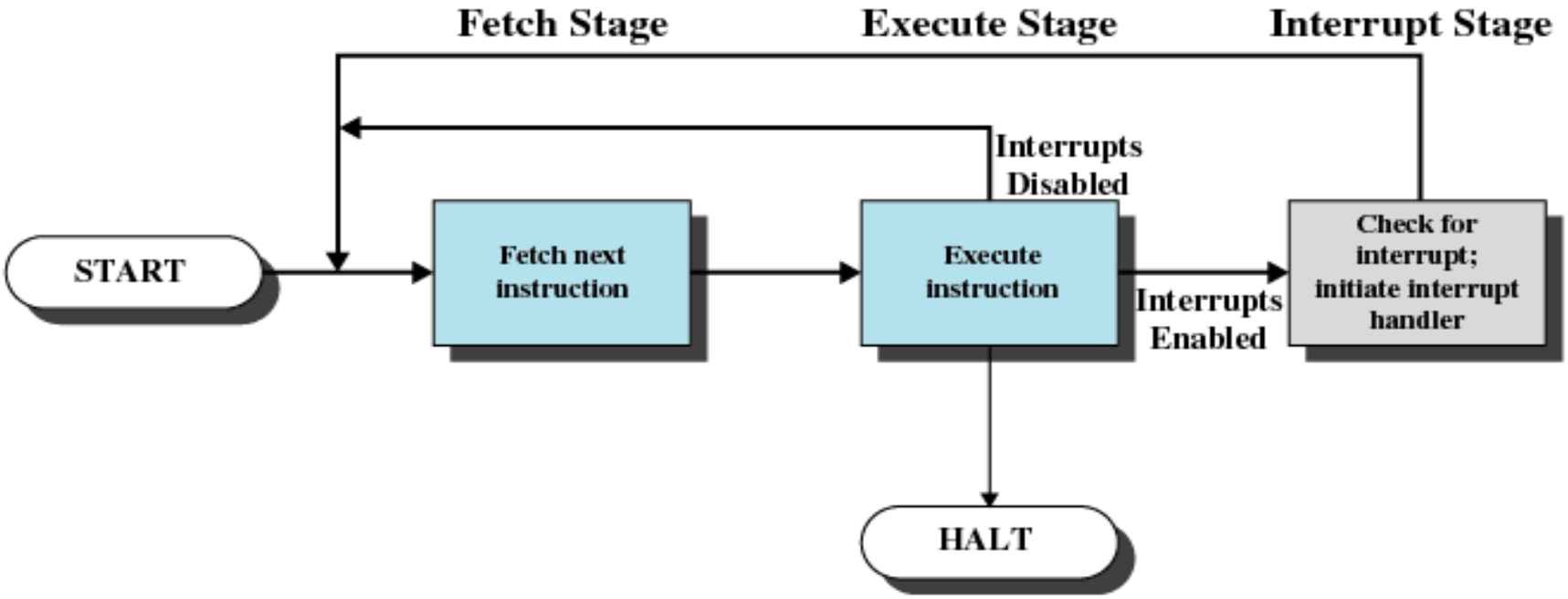
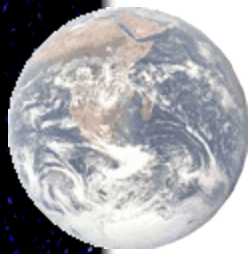


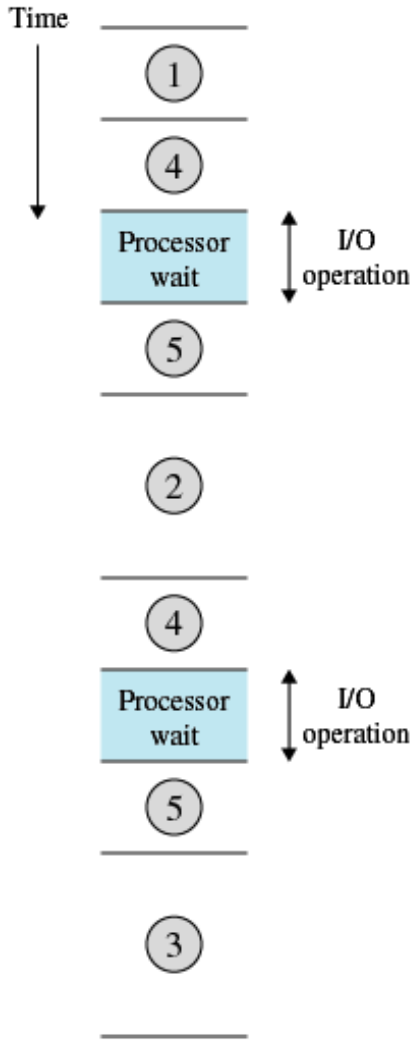

**Figure 1.7 Instruction Cycle with Interrupts**



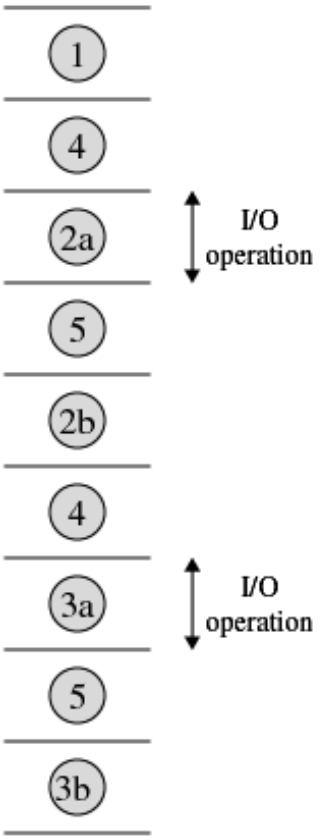
# Interrupt Cycle

- 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 routine

# Timing Diagram: Short I/O Wait

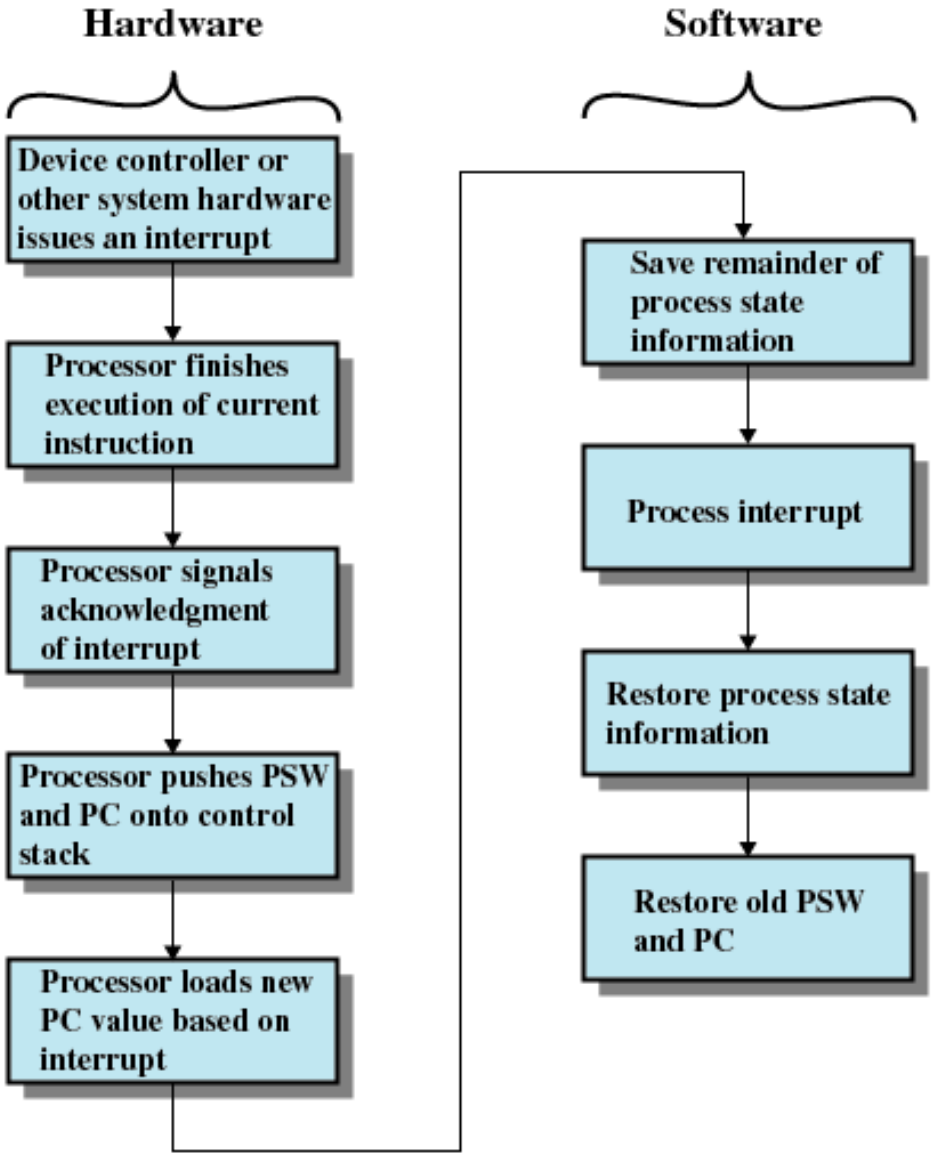



(a) Without interrupts
(circled numbers refer
to numbers in Figure 1.5a)



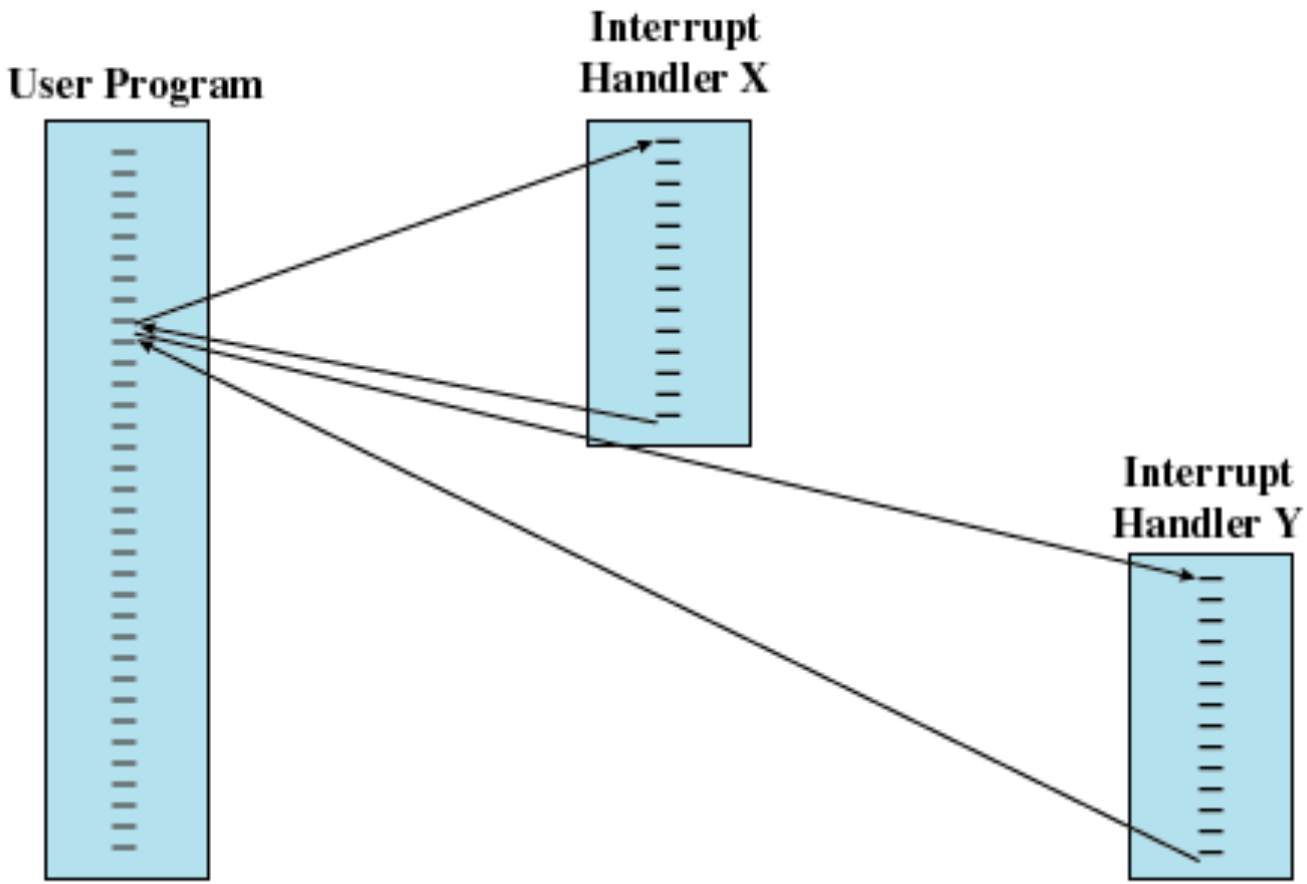

(b) With interrupts
(circled numbers refer
to numbers in Figure 1.5b)

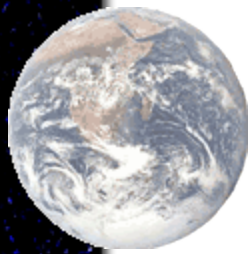
# Simple Interrupt Processing



# Multiple Interrupts

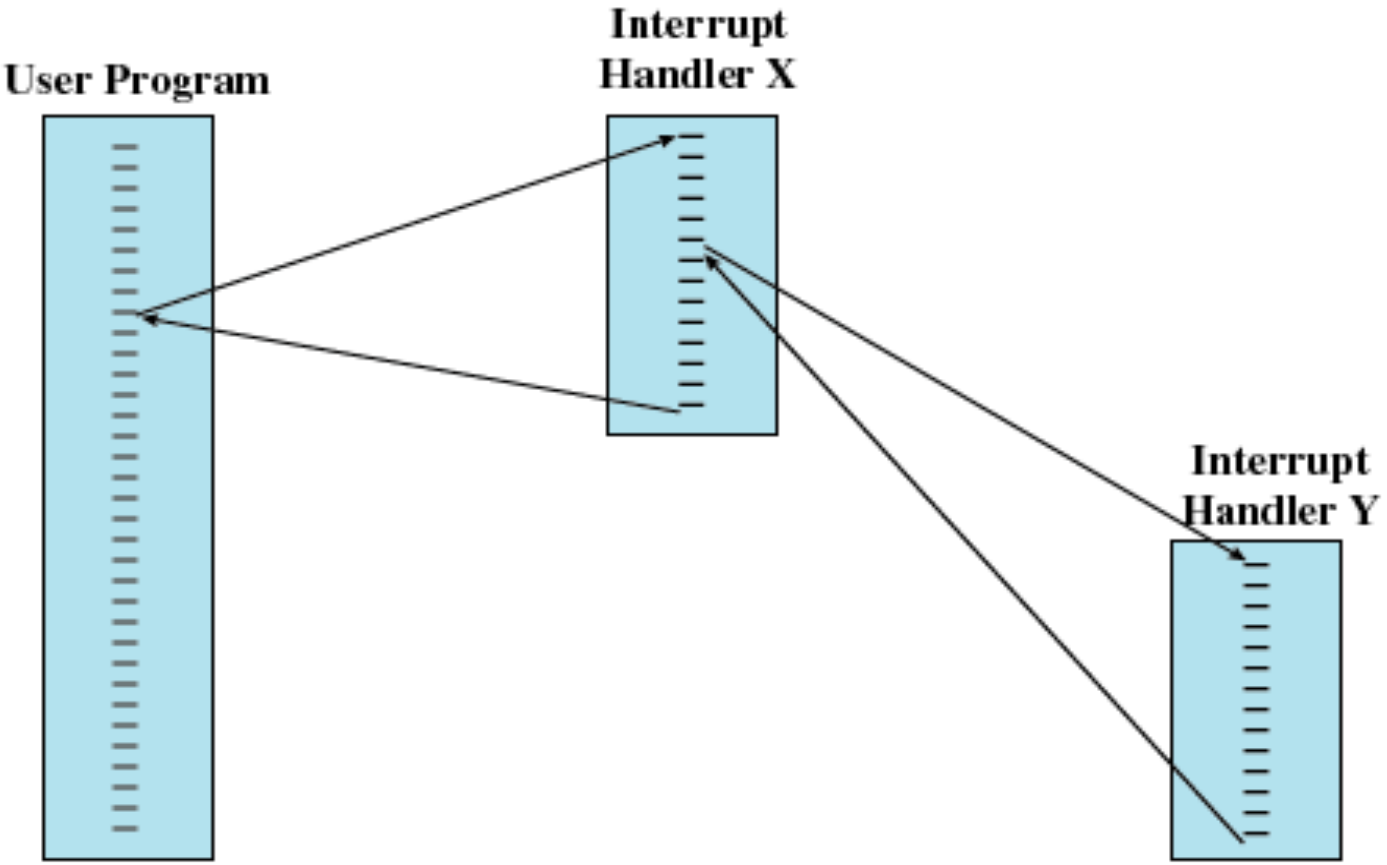
- Disable interrupts while an interrupt is being processed



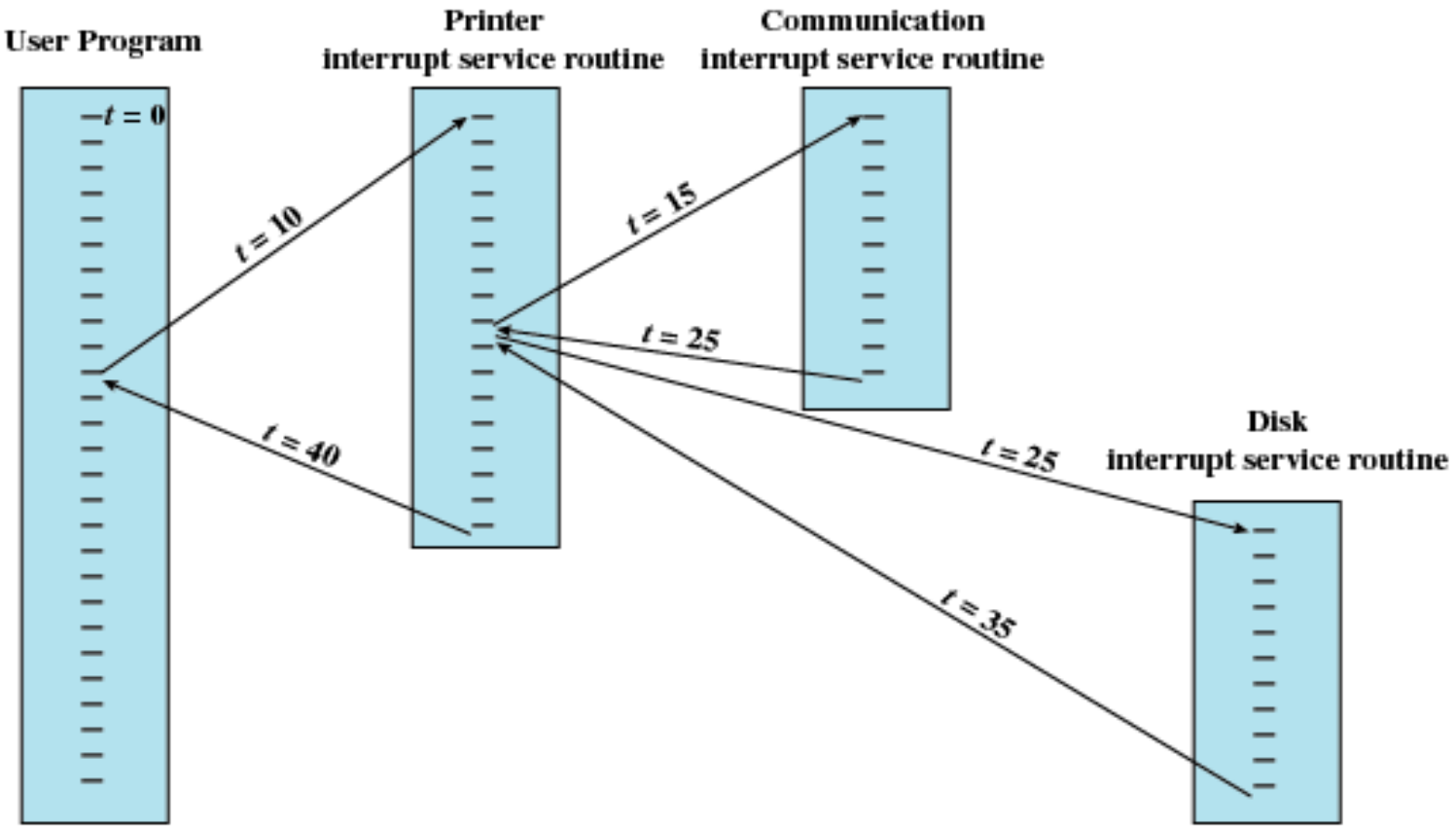


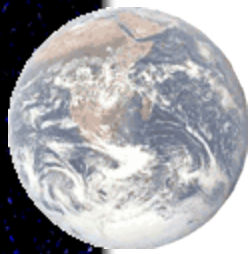
# Multiple Interrupts

- Define priorities for interrupts



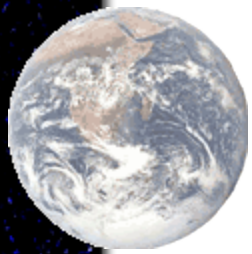
# Multiple Interrupts





# Multiprogramming

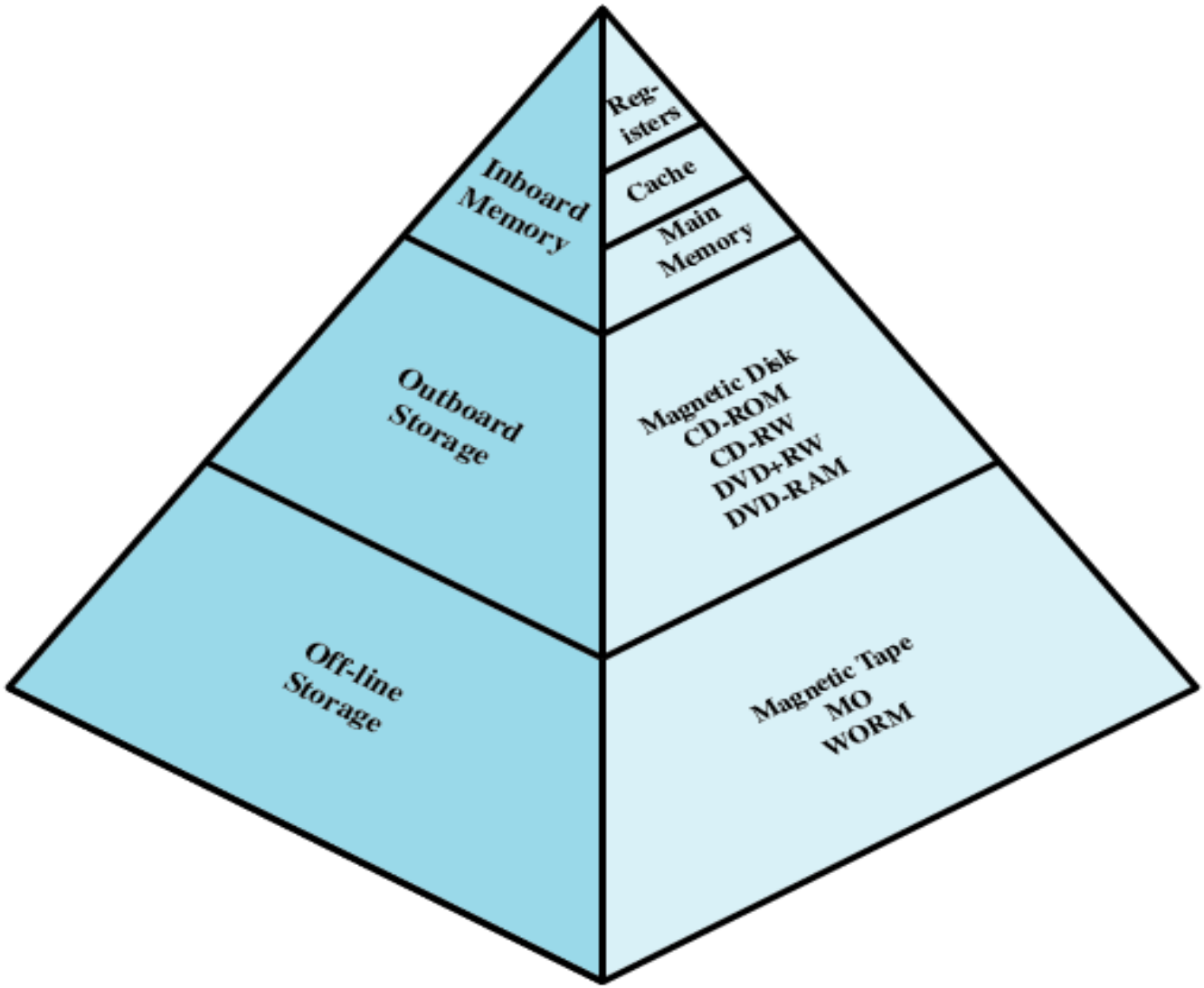
- Processor has more than one program to execute
- The sequence the programs are executed depend on their relative priority and whether they are waiting for I/O
- After an interrupt handler completes, control may not return to the program that was executing at the time of the interrupt

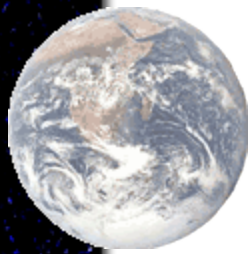


# Memory Hierarchy

- Faster access time, greater cost per bit
- Greater capacity, smaller cost per bit
- Greater capacity, slower access speed

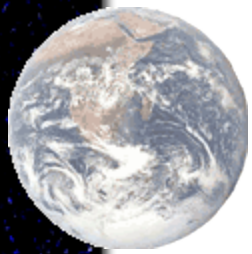
# Memory Hierarchy





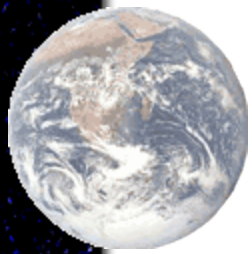
# Going Down the Hierarchy

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



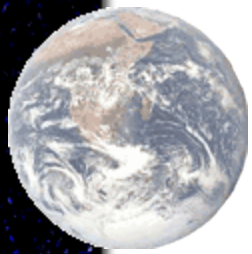
# Secondary Memory

- Nonvolatile
- Auxiliary memory
- Used to store program and data files



# 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
- Exploit the principle of locality

# Cache Memory

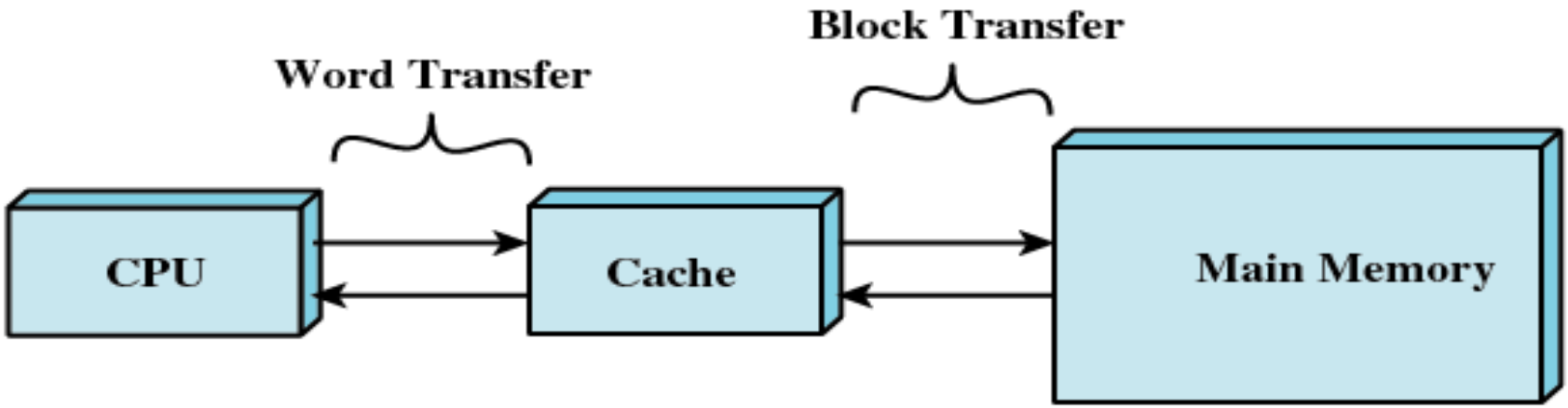
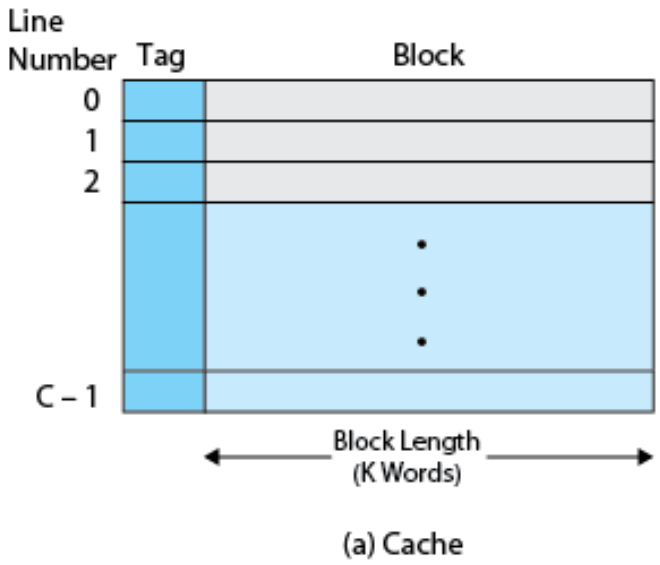
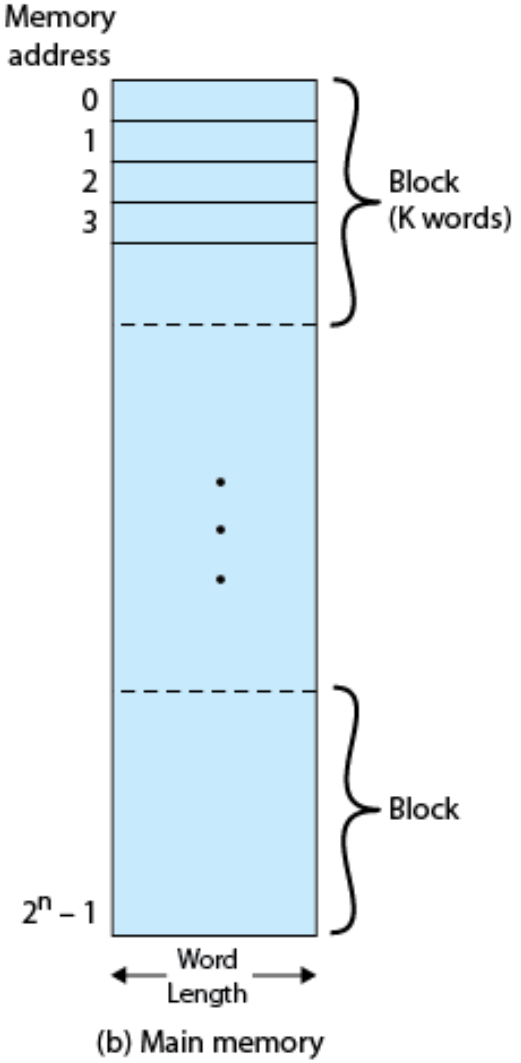



Figure 1.16 Cache and Main Memory

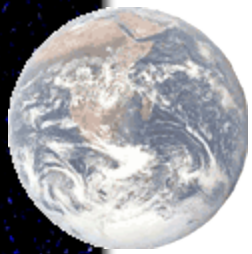
# Cache/Main Memory System




(a) Cache



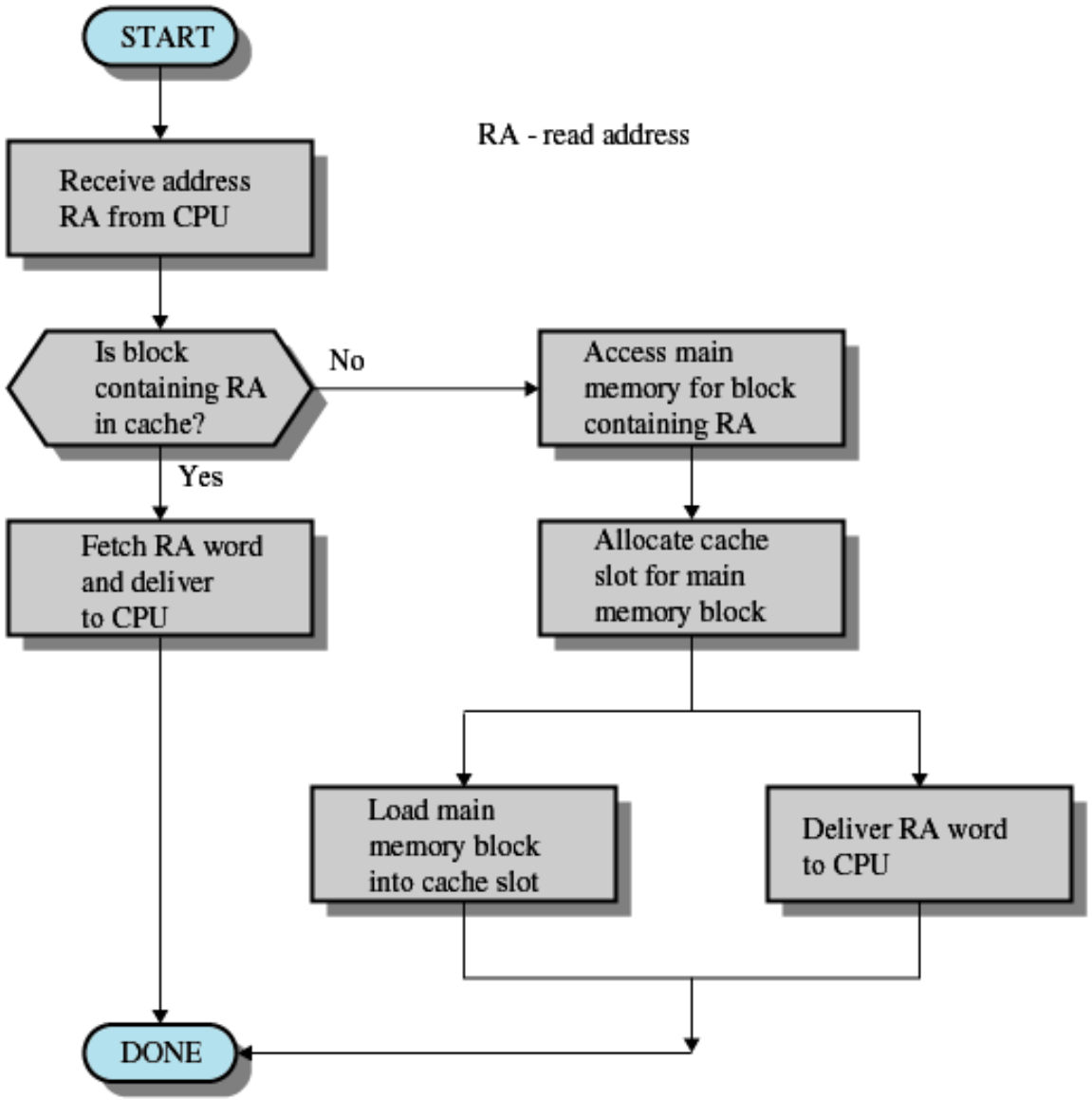

(b) Main memory

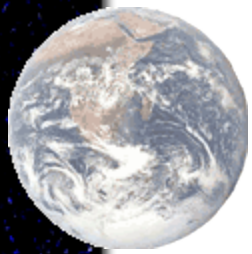


# Cache Memory

- Contains a copy of 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 and delivered to the processor

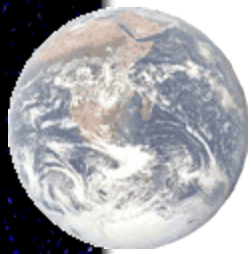
# Cache Read Operation





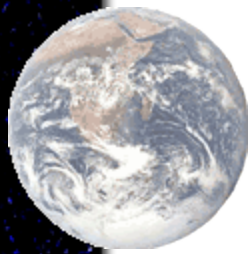
# Cache Design

- Cache size
  - Small caches have a significant impact on performance
- Block size
  - The unit of data exchanged between cache and main memory
  - Larger block size more hits until probability of using newly fetched data becomes less than the probability of reusing data that have to be 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 write operations
    - Leaves main memory in an obsolete state