សាកលវិទ្យាល័យភូមិន្ទភ្នំពេញ

អប់រំ ស្រាវជ្រាវ និង សេវាសង្គម

FE មហាវិទ្យាល័យវិស្វកម្ម

Introduction to Operating Systems

Department of Information Technology Engineering

Lecturer: Kor Sokchea

Administrivia

- **Textbook:** *Operating System Concepts*, 9th Edition, by Silberschatz, Galvin, and Gagne.
- Exams
  - Final Exam: 40%
  - Presentation: 15%
  - Quiz: 10%
  - Programming Assignment: 15%
  - Homework: 10%
  - Attendance: 10%

Course Topics

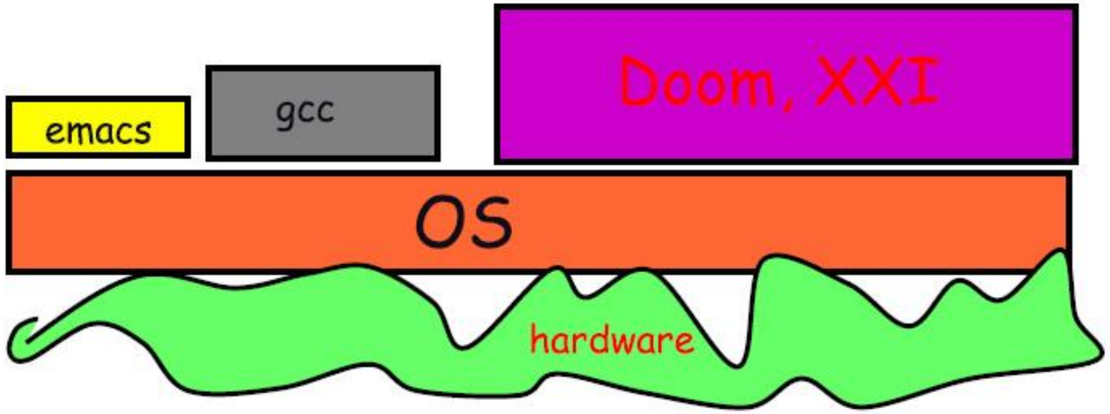
- Threads & Processes
- Concurrency & Synchronization
- Scheduling
- Deadlocks
- Main Memory
- Virtual Memory
- I/O systems
- Disk
- File System
- Protection & Security

Course goals

- **Introduce you to operating system concepts**
  - Hard to use a computer without interacting with OS
  - Understanding the OS enable you to become a more effective programmer
- **Cover important system concepts in general**
  - Computer-System Structure
  - Computer-System Organization
  - Caching
  - Concurrency
  - Memory Management,
  - I/O
  - Protection
- **Prepare you to design your operating system in the future**

What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware
- **Makes hardware useful to the programmer**
- **Provides abstraction for applications**
  - Manages and hides complexity of hardware
  - Accesses hardware through low/level interfaces unavailable to application
- **Provides protection**
  - Prevents one process/user from conflicting with another



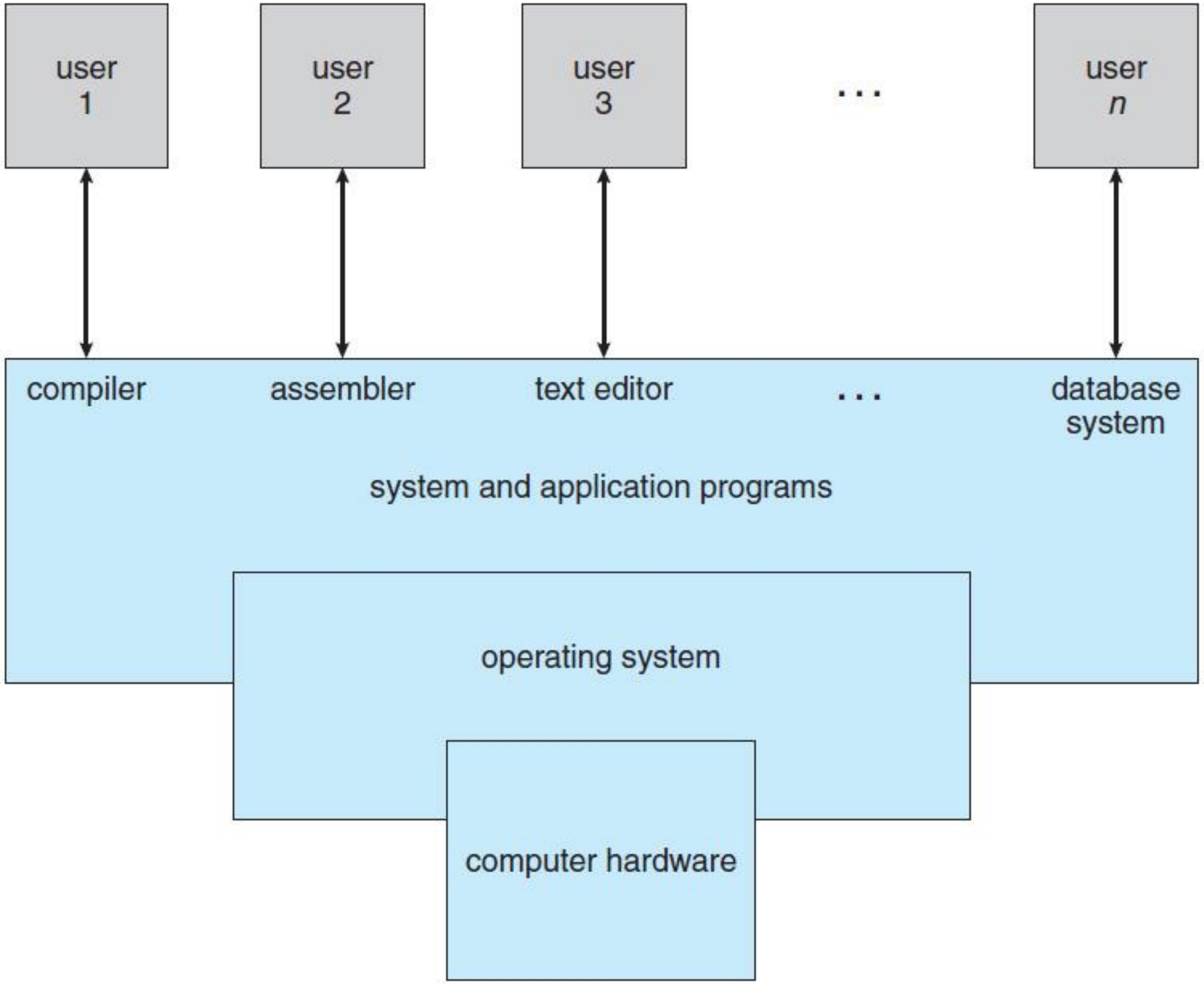
Why study operating systems?

- **Many of you will develop systems utilizing the core concepts in operating systems**
  - Whether you build software or hardware
  - The concepts and design patterns appear at many levels
- **Building applications utilizing operating systems**
  - The better you understand their design and implementation, the better you will make use of them
- **High-performance servers issue**
- **Resource consumption issue**
  - Battery life, radio spectrum, etc
- **Security issue**
- **New "smart" devices need new OS**

Operating Systems at the heart of it all …

- Make the incredible advance in the underlying hardware available to a rapid evolving body of applications
  - Processing, Communications, Storage, Interaction
- The key building blocks
  - Scheduling
  - Concurrency
  - Address spaces
  - Protection, Isolation, Security
  - Networking, distributed systems
  - Persistent storage, transactions, consistency, resilience
  - Interfaces to all devices

Computer System Structure



Computer System Structure

- Computer system can be divided into four components:
  - **Hardware:** provides basic computing resources
    - CPU, memory, I/O devices
  - **Operating system**
    - Controls and coordinates use of hardware among various applications and users
  - **Application programs:** define the ways in which the system resources are used to solve the computing problems of the users
    - Word processors, compilers, web browsers, database systems, video games
  - **Users**
    - People, machines, other computers

What Operating Systems Do

- **User View:**
  - **PC Operating System (OS)** is designed for convenience, **ease of use** and **good performance**
    - Don’t care about **resource utilization**
  - **Mainframe** or **minicomputer OS** is designed to **maximize** *resource utilization*
  - **Workstation OS** is designed to compromise between individual usability and resource utilization
  - Handheld computers are resource poor, optimized for usability and battery life
  - Some computers have little or no user interface, such as embedded computers in devices and automobiles
    - It OS is designed primarily to run without user intervention

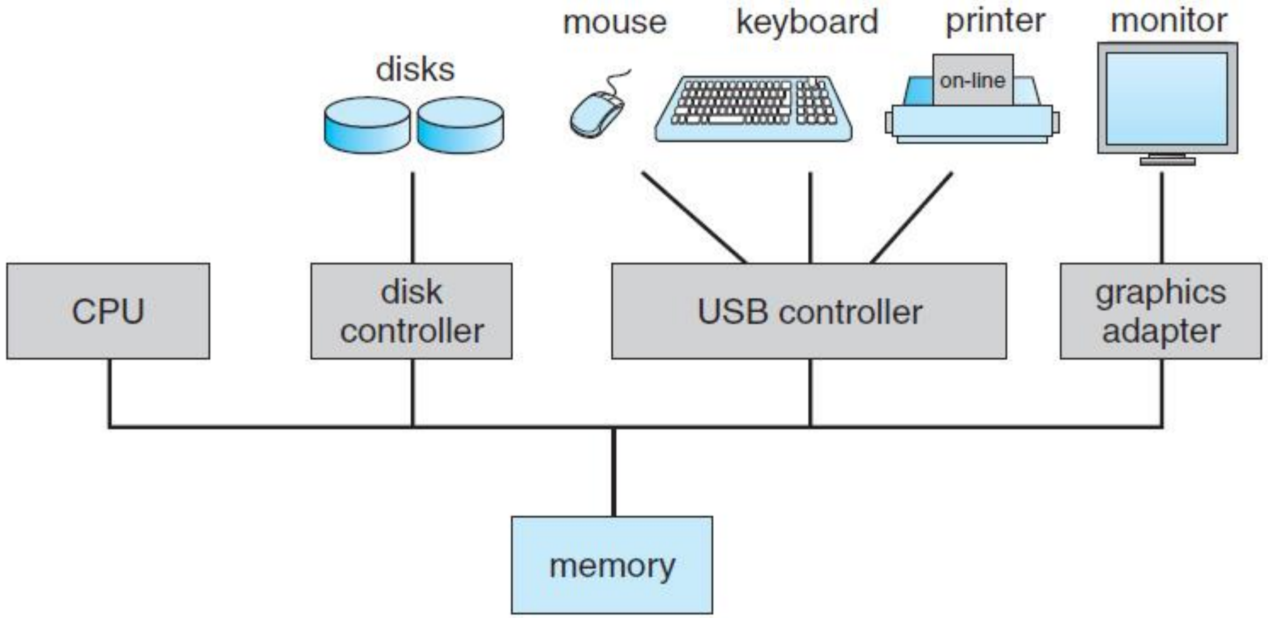
What Operating Systems Do

- **System View:**
  - OS is a **resource allocator**
    - Manages all resources
    - Controls between conflicting requests for efficient and fair resource use
  - OS is a **control program**
    - Controls execution of programs to prevent errors and improper use of computer

Computer-System Organization

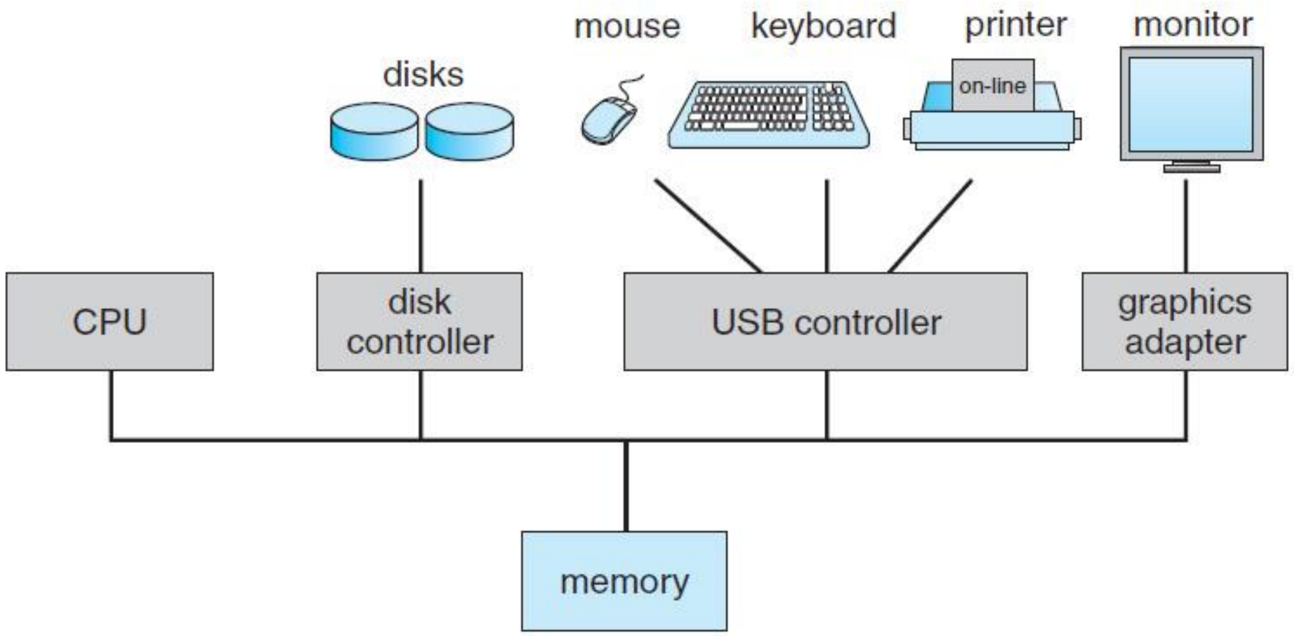
Computer-System Operation:

- A modern general-purpose computer system contains one or more CPUs and device controllers connected through a common bus offering access to shared memory
- Concurrent execution of CPUs and devices competing for memory cycles
- Each device controller is in charge of a specific type of device



Computer Operation

- **Bootstrap program** generally known as **firmware** is loaded at power-up or reboot
  - Typically stored within read-only memory (**ROM**) or electrically erasable programmable read-only memory (**EEPROM**)
  - Initializes all aspects of the system
  - Loads operating system kernel and starts execution by locating the operating system kernel and load it into memory

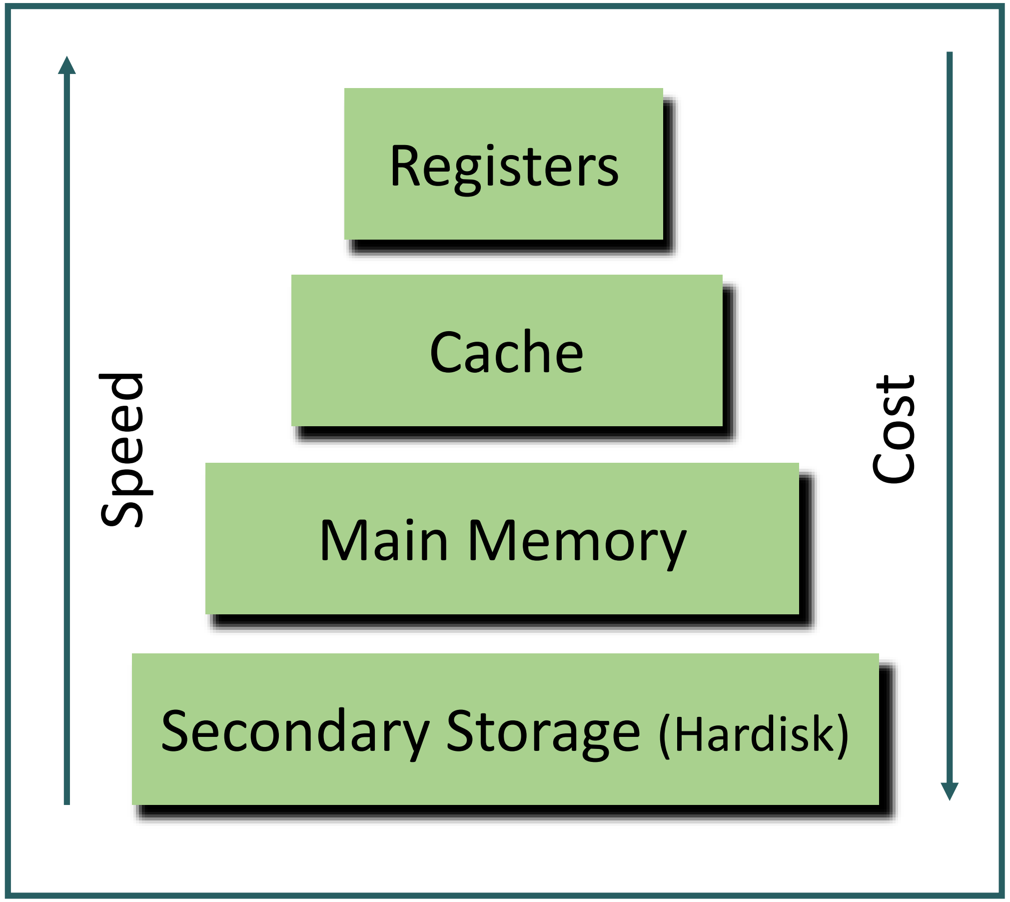


Storage Structure

- **Main memory** – only large storage media that the CPU can access directly
  - Random access
  - Typically volatile
- **Secondary storage** – extension of main memory that provides large nonvolatile storage capacity
- **Hard disks** – rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into tracks, which are subdivided into sectors
  - The disk controller determines the logical interaction between the device and the computer
- **Solid-state disks** – faster than hard disks, nonvolatile
  - Various technologies
  - Becoming more popular

Storage Hierarchy

- Storage systems are organized in hierarchy
  - **Speed**
  - **Cost**
  - **Volatility**
- **Caching**: copying information into faster storage system; main memory can be viewed as a cache for secondary storage
- **Device Driver** for each device controller to manage I/O
  - Provides uniform interface between controller and kernel



I/O Structure

- A general-purpose computer system consists of CPUs and multiple device controllers connecting through a common bus
  - Each device controller is in charge of a specific type of device
- The device controller is responsible for:
  - Moving data between the peripheral devices
  - Maintaining local buffer storage
- Operating systems consist of a **device driver** for each device controller
- **Device driver:**
  - Understands the device controller
  - Provides uniform interface between controller and kernel

How I/O Operation Works

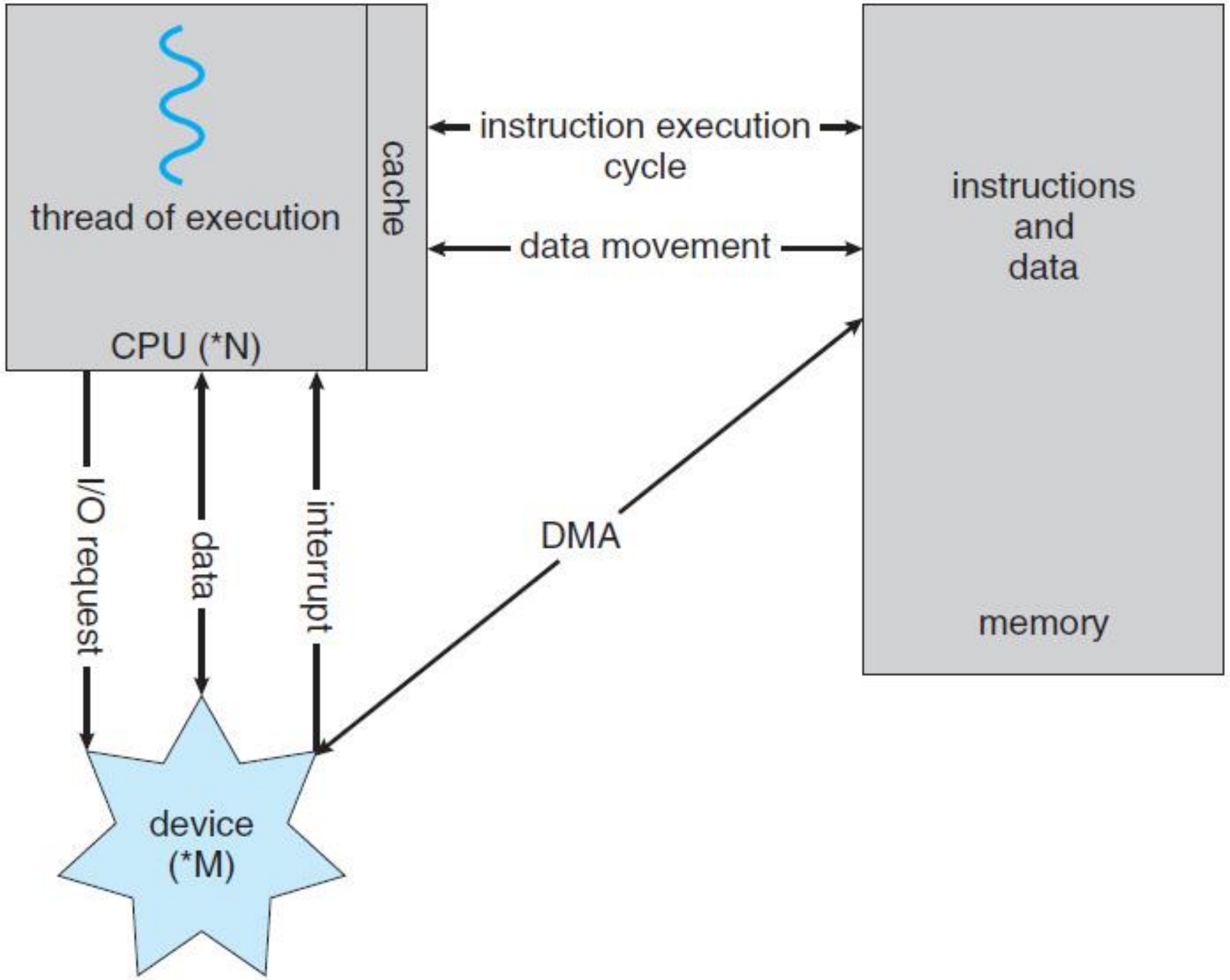
- To start an I/O operation, the device driver loads the appropriate registers within the device controller
- Device controller examines the contents of these registers to determine what action to take
  - Ex: Read a character from the keyboard
- The controller starts the transfer of data from the device to its local buffer
- Once the transfer of data is complete, the device controller informs the device driver via an **interrupt**
- Then, device driver returns control to the operating system
  - Returning the data or a pointer to the data if the operation was a read
- Other operations, the device driver returns status information
- This form of interrupt-driven I/O is bad for moving large amounts of data

Direct Memory Access Structure

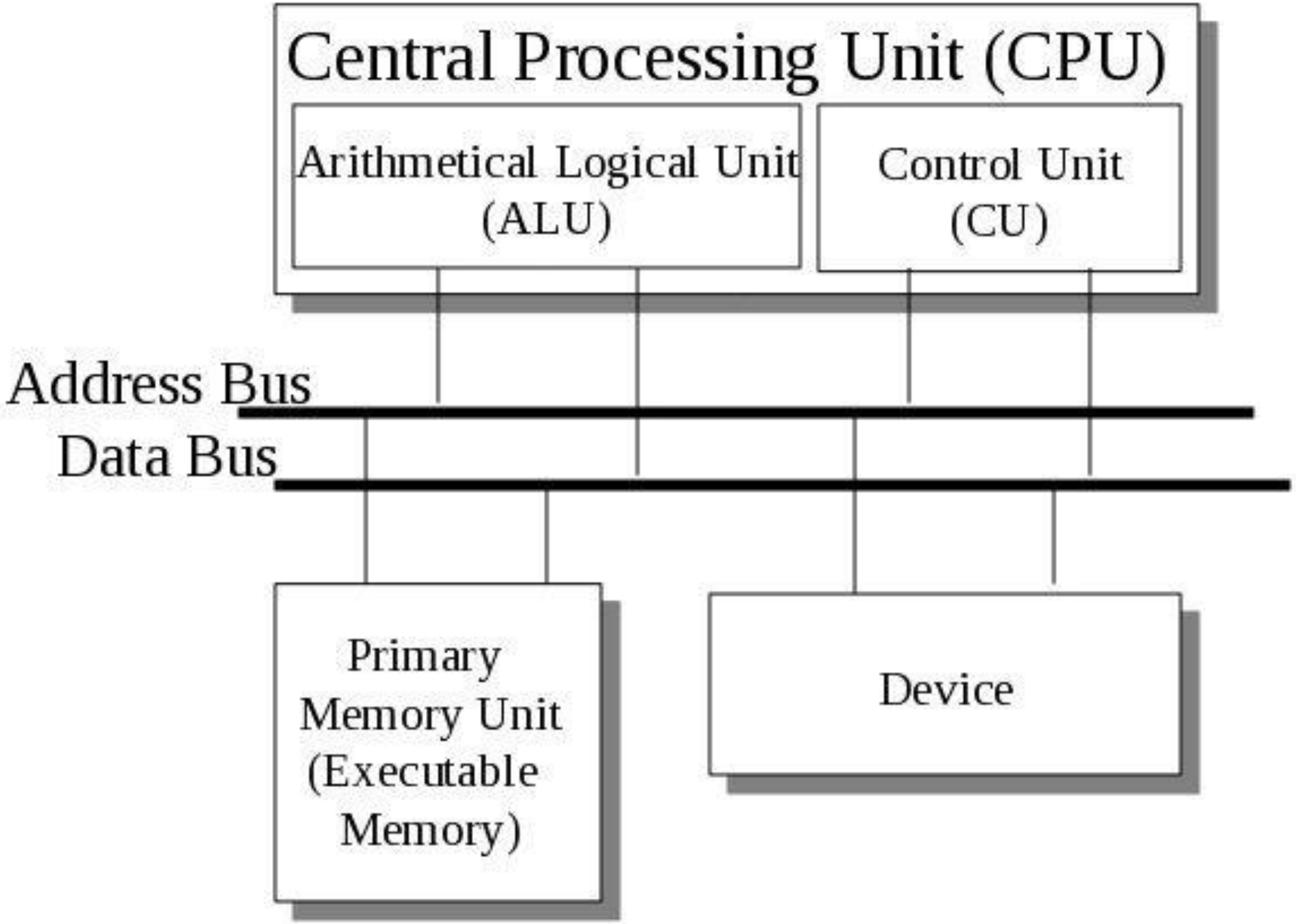
- Used for high-speed I/O devices able to transfer information at close to memory speeds
- Device controller transfers an entire block of data directly to or from its own buffer storage to memory
- Only one interrupt is generated per block, rather than the one interrupt per byte

How a modern computer system works

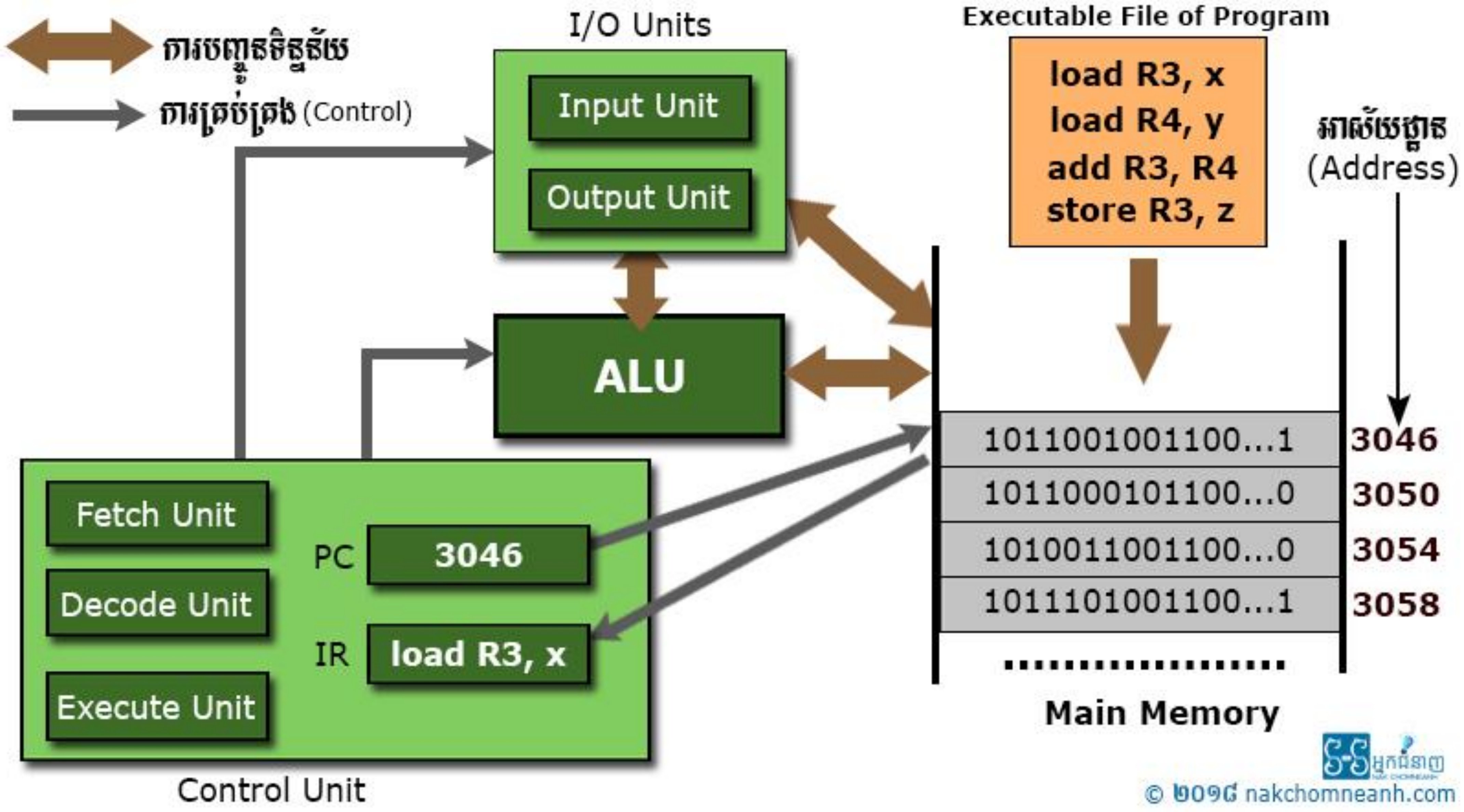
- Data is constantly being moved between the CPU, memory and the various devices
- The CPU uses I/O addresses to direct data to particular devices
- The devices in turn utilize interrupts to notify the CPU and operating system of their needs



The von Neumann Architecture



How does CPU works?

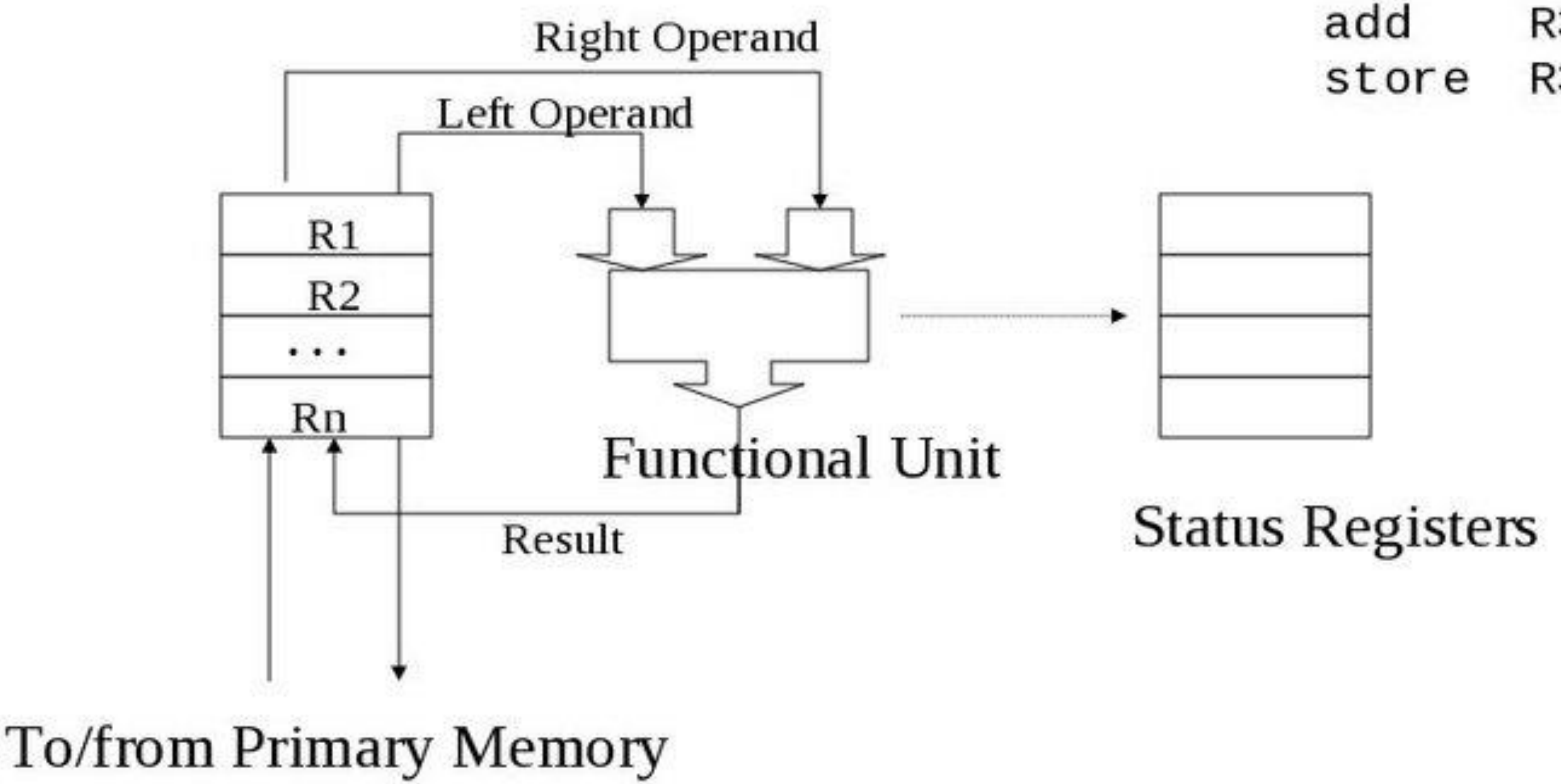


The CPU Control Unit

- **Program Counter (PC)**: is a register in a computer processor that contains the address of the instruction being executed a the current time
- **Instruction Register (IR)**: is the part of a CPU's control unit storing instruction currently being **executed** or **decoded**
- **CPU Operation:**
  - **Fetch:** the first step that involves retrieving an instruction from program memory
  - **Decode:** the instruction is converted into signals that control other parts of the CPU
    - The way in which the instruction is interpreted is defined by the CPU's instruction set architecture
  - **Execute:** is performed after the fetch and decode steps.
    - **ALU** does calculation
    - **I/O** unit loads or stores data between main memory registers

The Arithmetic Logic Unit (ALU)

```
load   R3,b
load   R4,c
add    R3,R4
store  R3,a
```



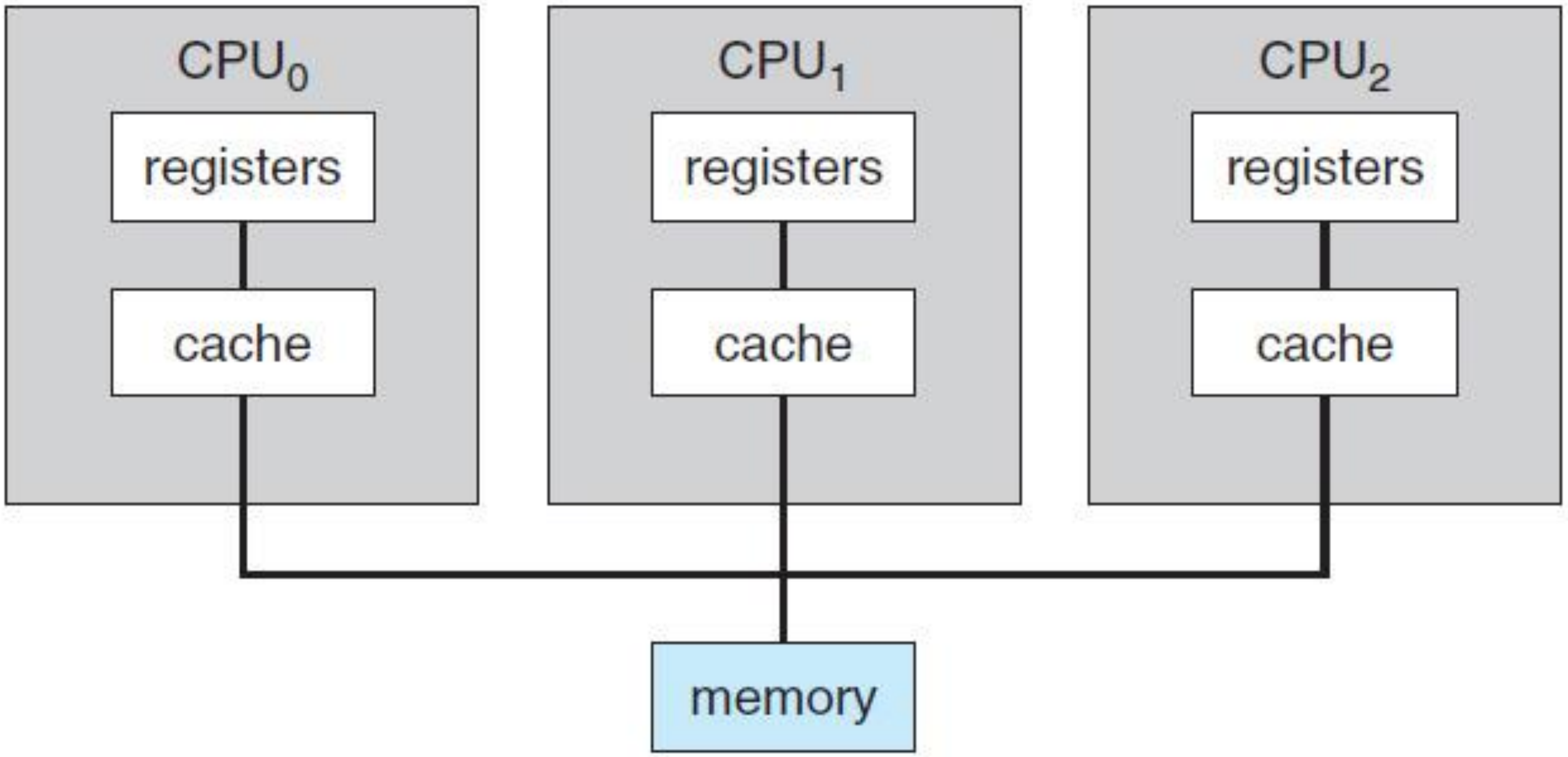
The von Neumann Loop

```
PC = <machine start address>;
IR = memory[PC];
haltFlag = CLEAR;
while(haltFlag not SET) {
    execute(IR);
    PC = PC + sizeof(INSTRUCT);
    IR = memory[PC]; // fetch phase
}
```

Computer-System Architecture

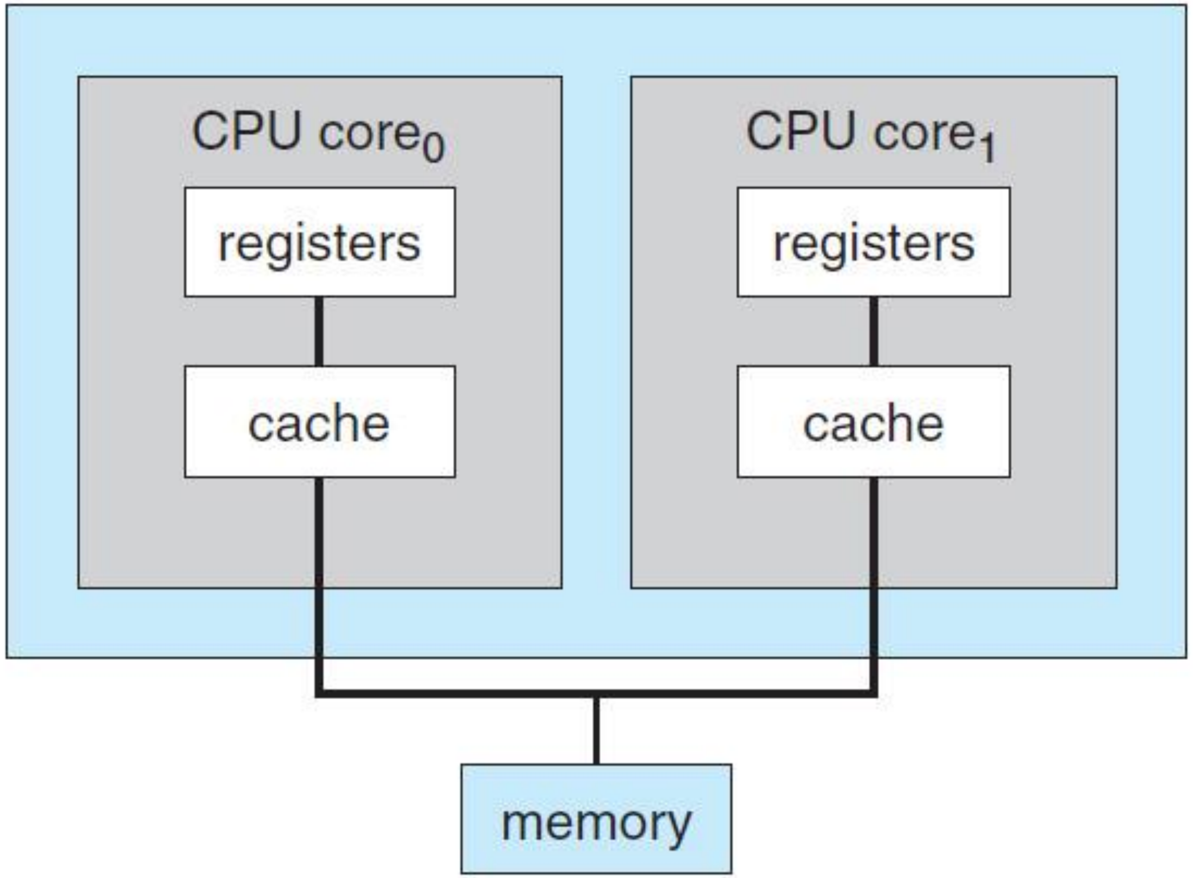
- Most System use a single general-purpose processor
  - Most system have special-purpose processors as well
- **Multiprocessors systems** also known **as parallel systems** or **multicore systems** dominates the computing market
  - Advantages include:
    a. Increased throughput
    b. Economy of scale
    c. Increased reliability – **graceful degradation** or **fault tolerant**
- There are two kinds:
  - **Asymmetric Multiprocessing** – Each processor is assigned a specific task
    - A boss processor control the system
    - The other processors look to the boss for instruction
  - **Symmetric multiprocessing** – Each processor performs all tasks

Symmetric Multiprocessing Architecture



A Dual-Core Design

- Recent CPU design is to include multiple computing cores on a single chip called **multicore**
- They can be more efficient than multiple chips with single cores
  - Because on-chip communication is faster than between-chip communication



Basic of How Operating System Work

- ## Role of Interrupts

  - **Interrupts** are signals sent to the CPU by external devices, normally I/O devices. They tell the CPU to stop its current activities and execute the appropriate part of the operating system

  - There are three types of interrupts:

    1) **Hardware Interrupts:** are generated by hardware devices to signal that they need some attention from the OS

       - **Ex:** They maybe have just received some data (keystrokes) on the keyboard or data on the Ethernet card

    2) **Software Interrupts:** are generated by programs when they want to request a system call to be performed by the operating system

    3) **Traps:** are generate by the CPU itself to indicate that some error or condition occurred for which assistance from the OS is needed

Basic of How Operating System Work

- **CPU Execution Mode**
  - There are two mode of execution:
    - **User mode:** is restricted in that certain instructions cannot be executed, certain registers cannot be accessed, and I/O devices can not be accessed
      - Can only execute a subset of instructions
      - Can only reference a subset of memory locations
    - **Kernel** or **supervisor mode:** Kernel mode has none of restrictions
      - A system call will set the CPU to kernel mode, as will traps and interrupts
      - Can execute all machine instructions
      - Can reference all memory locations

Basic of How Operating System Work

- **CPU Response to Interrupts**

  - A key point towards understanding how operating systems work is to understand what the CPU does when an interrupt occurs

  - The hardware of the CPU does the exact same thing for each interrupt, which is what allows operating systems to take control away from the current running user process

  - The switching of running processes to execute code from the OS kernel is called a **context switch**

  - **Context switch:** switching from running a user level process to the OS kernel and to other user processes before the current process is resumed

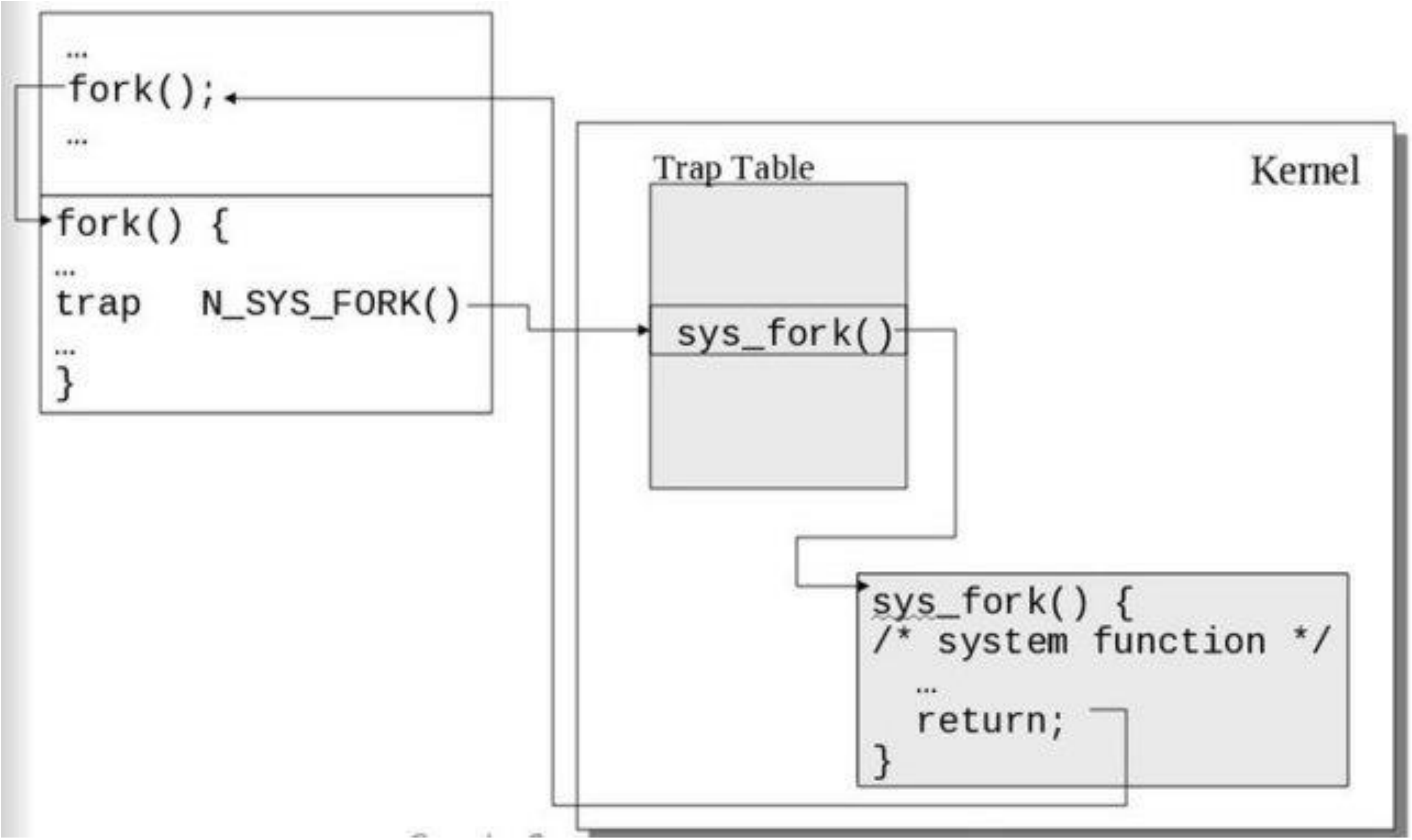
Basic of How Operating System Work

- **CPU Response to Interrupts**
  - CPUs reply on the data contained in a couple of registers to correctly handle interrupts
  - One register holds a pointer to the **process control block** of the current running process
    - This register is set each time a process is loaded into memory
  - The other register holds a table containing pointers to the instructions in the OS kernel for **interrupt handlers** and **system calls**
  - The value in this register and contents of the table are set when the operating system is initialized at boot time

Basic of How Operating System Work

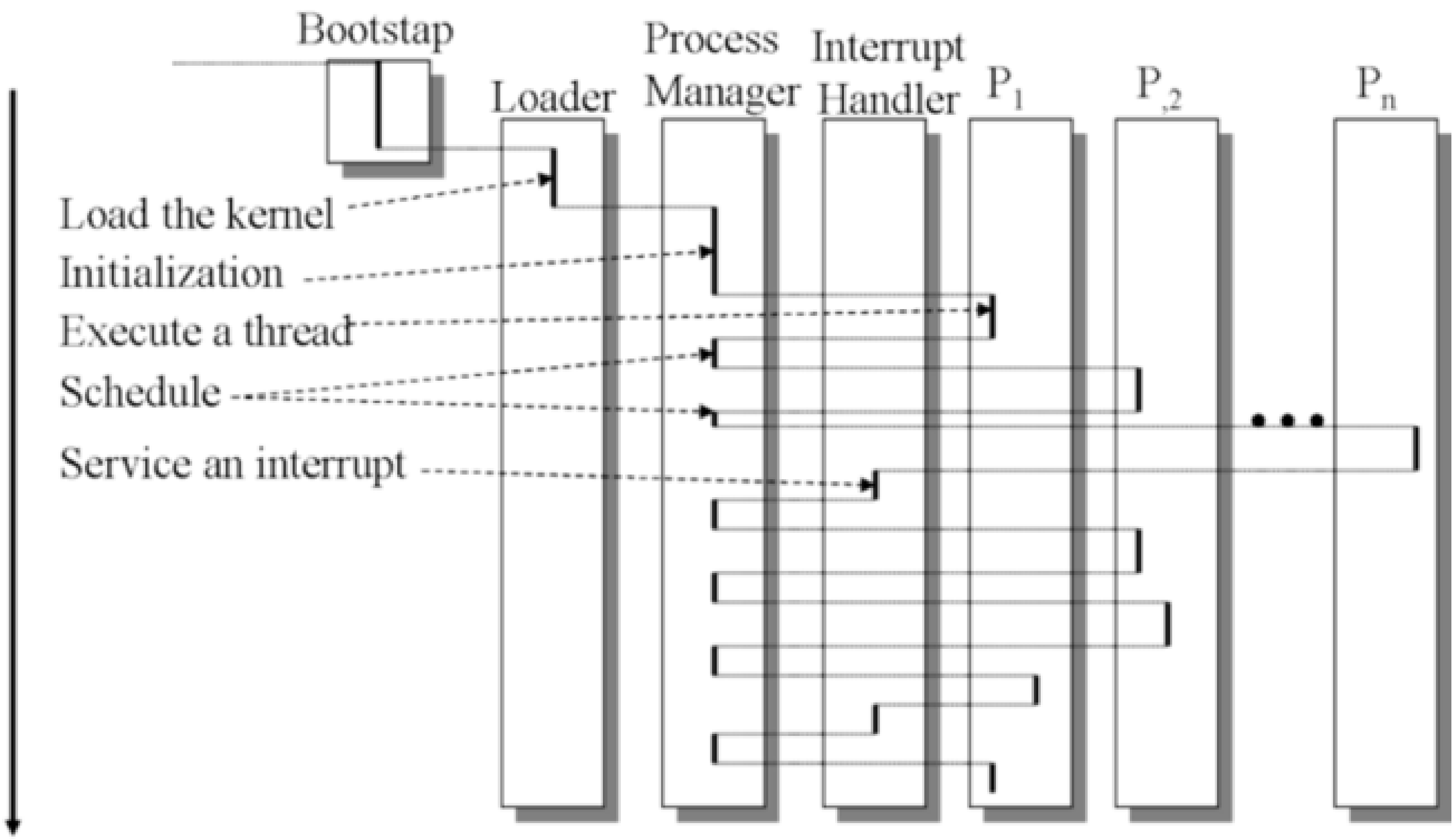
- **CPU Response to Interrupts**
  - The CPU performs the following actions in response to an interrupt:
    1. Using the pointer to the current process control block
       - The state and all register values for the process are saved for use when the process is later restarted
    2. The **CPU mode bit** is switched to **supervisory mode**
    3. Using the pointer to the interrupt handler table and the interrupt vector
       - The location of the kernel code to execute is determined
       - The interrupt vector is the **interrupt request** (**IRQ**) for hardware interrupts and an argument to the interrupt assembly language instruction for software interrupts
    4. Processing is switched to the appropriate portion of the kernel

Basic of How Operating System Work



Software Interrupts

Basic of How Operating System Work



processing switches between user processes and the operating system as hardware and software interrupts are received

Parts of Operating System

- There are 4 broad tasks performed by an operating system:
  - **Process Management**
    - A process is an executing program
    - It consists of code, data, a certain set of resources allocated to it, and one or more threads of execution through the code
    - The OS manages the allocation of resources to these processes, and also provides system calls to manage these processes
  - **Memory Management**
    - Memory must be shared between the OS and an application program
    - The OS must manage the allocation of memory to processes and control the memory management hardware that determines which memory locations a process may access

Parts of Operating System

- There are 4 broad tasks performed by an operating system:
  - **File System Management**
    - Computers process information that must be transmitted, processed, and stored
    - File systems are an abstract organized collection of file system objects
    - The OS provides primitives to manipulate these objects
  - **Device Management**
    - Information is sent through a computer's input and output devices
    - Processes access these devices using the system call interface
    - The OS tries to manage devices in a manner that makes them efficiently shared among all processes requiring them
    - A **system call** is a programming interface to the services provided by the OS, typically written in C/C++

Operating-System Design and Implementation

- **Design Goals**
  - At the highest level, system design is dominated by the choice of hardware and system type
  - Beyond this level, the requirements can be divided into two groups:
    - **User goals:** include convenience, reliability, security, and speed
    - **System goals:** include ease of design, implementation, maintenance, flexibility, and efficiency

Operating-System Design and Implementation

- **Implementation**
  - Operating systems were firstly written in assembly, but nowadays C/C++ is the language commonly used
  - Small blocks of assembly code are still needed, especially related to
    - Low level **I/O** functions in device drivers
    - Turning interrupts on and off
    - The **Test and Set Instruction** for **Synchronization Facilities**
  - Using higher level languages allows the code to be written faster
    - It also makes the OS much easier to port to different hardware platforms