Introduction to Operating Systems
Lect. 1 - Agenda

- Course outline and policies
- Computer-System Organization
- What Operating Systems Do
- Operating-System Structure and operations

General Information:

Instructor : Omar Mansour, Ph.D.

Email : omansour@nyu.edu

Office : 370 Jay street, room 816

Credit Hours : 3

Class sessions : Wednesday, 6-8:30 pm

Class location : online (via Zoom)

Teaching assistants:

David Du (yd1487@nyu.edu)
Zirui Fu (zf715@nyu.edu)
Xinyuan Zhang (xz306@nyu.edu)

Required Text Book:

Silberschatz, Galvin and Gagne, ***Operating System Concepts***, 9th edition, Wiley.

Course Description:

This course covers the functions and organization of operating systems, including process management, memory management, resource allocation, input/output systems, and information protection.

Pre-requisites:

1. C language programming
2. Computer architecture
3. Data structures and algorithms, particularly stacks, queues and linked lists.

Course Objectives:

1. Acquiring fundamental knowledge and proficiency in modern operating system design.
2. Learning how to use utilities provided by modern operating systems in developing reliable applications that can interact with the system and with other local or remote applications.

Grading:

Assignments and quizzes : 40%
Mid-term exam : 30%
Final exam : 30%

Grading range:

Grade letter	Percentage of available points
A	94-100
A-	88-93
B+	82-87
B	76-81
B-	70-75
C+	64-69
C	58-63
C-	50-57

Attendance and participation policy:

- Attendance and participation includes attendance, class participation (e.g. answering questions posed by the instructor), quizzes, in-class assignments, etc.
- If I notice a significant amount of class absence or lack of participation in assignments, quizzes, etc. , I may notify you by email, and it may result in failing or being withdrawn from the course.

Assignments policy:

1. Assignments must be submitted on or before 11:55 pm on the day they are due.
2. Late assignments will not be permitted.
3. Students are required to perform the work pertaining to the assignments **alone.** This includes programming assignments. Students however are encouraged to discuss the concepts pertaining to the course or the assignments with other students or with their teaching assistants, while doing the actual work themselves.
4. Copying of code or answers to homework questions is an act of plagiarism. If the teaching assistant suspects any type of cheating or plagiarism, he/she may ask the student involved for discussing his/her work.

Syllabus (tentative):

Week	Description
1	Introduction
2	Operating systems architecture
3	Processes and operating system data structures
4	Inter-process communications
5	Threads
6	Synchronization
7	Midterm exam
8	Deadlocks
9	Scheduling
10	Memory management
11	Virtual memory
12	Disk Management
13	I/O and file systems
14	Final exam

Academic Honesty:

- Students at NYU are expected to be honest and forthright in their academic endeavors.
- Academic dishonesty includes cheating, unapproved collaboration, coercion, inventing false information or citations, plagiarism, tampering with computers, destroying other people's coursework, lab or studio property, theft of course materials, or other academic misconduct. If you have questions regarding this policy, contact your professor *prior* to submitting the work for evaluation. See your academic catalogue for a full explanation.
- All students must adhere to the NYU Tandon school of engineering's "Student Code of Conduct", https://engineering.nyu.edu/campus-and-community/student-life/office-student-affairs/policies/student-code-conduct

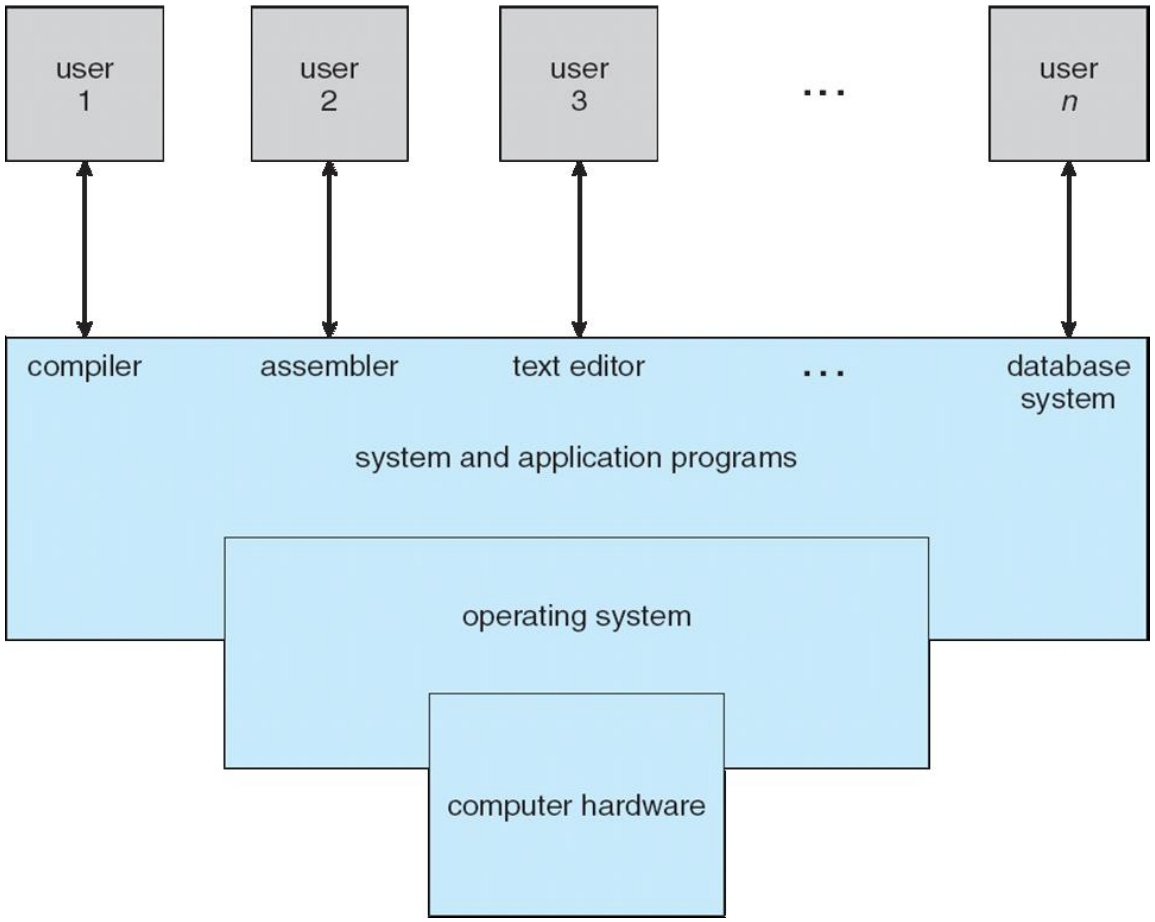
Academic Honesty (cont.):

- All assignments, unless otherwise explicitly listed, are to be done independently. Unless explicit permission from the instructor is provided, no outside sources may be used. If you have any doubts of your sources or their applicability, please contact the instructor as soon as possible.
- Anyone caught cheating in this course will receive a “0” on the assignment/assessment and the professor additionally retains the option of significantly reducing the final grade. If a student is caught a second time in, he/she shall fail the course.

Computer System Structure

- A Computer system can be divided into three components:
  - Hardware – provides basic computing resources
    - CPU, memory, I/O devices
  - Operating system
    - Controls and coordinates use of hardware among various applications and users
  - System and Application programs – Use system resources (e.g. CPU, memory, etc.) to solve the computing problems of the users
    - Example application programs: Word processors, web browsers, database systems, video games
    - Example of system programs: compilers, shell programs, etc.

Components of a Computer System



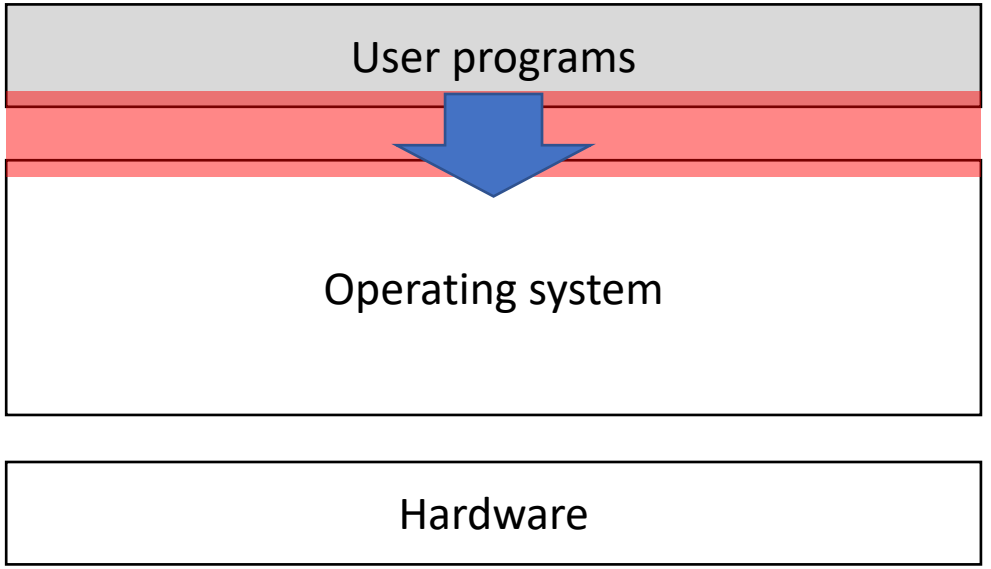
What is an Operating System?

- A program that acts as an intermediary between **application/system programs** and the **computer hardware**
  - Users mainly interact with the computer via the command-line interface (CLI) or Graphics user interface (GUI). Both may be considered **system programs**.
  - Application and system programs may also be referred to as **user programs**, because they run in an unprivileged mode (user mode), whereas the operating system runs in a privileged mode (or kernel mode).

- Operating system goals:
  1. Execute user programs, make solving user problems easier and make the computer system convenient to use.
  2. Use the computer hardware in an efficient and secure manner.

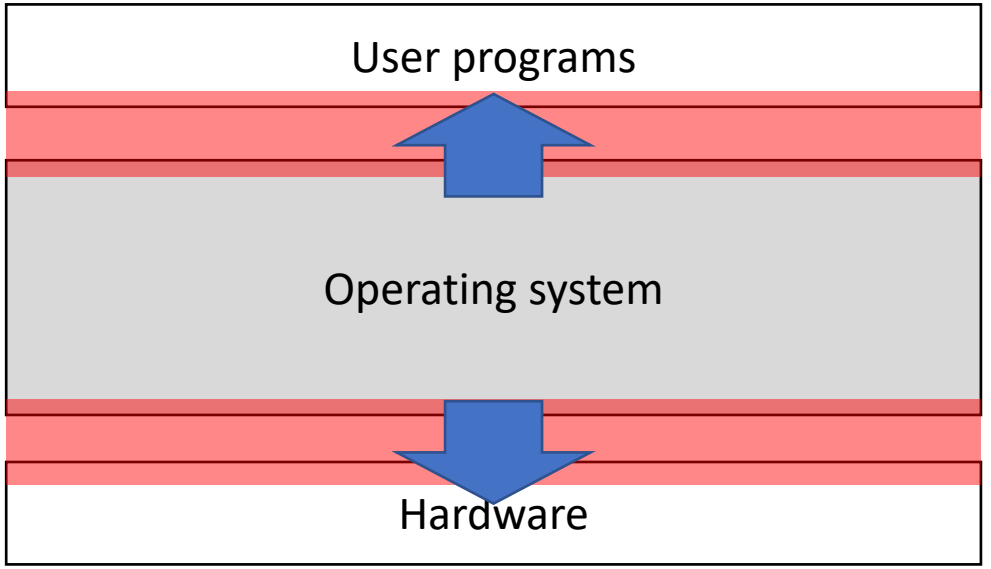
What Operating Systems Do? – user view

- Provides convenience, **ease of use** and **good responsiveness**
  - Don't care about **resource utilization**



What Operating Systems Do? – system view

- OS is a **resource allocator**
  - Manages all resources
  - Decides 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 the computer



What Operating Systems Do?

Various computing platforms:

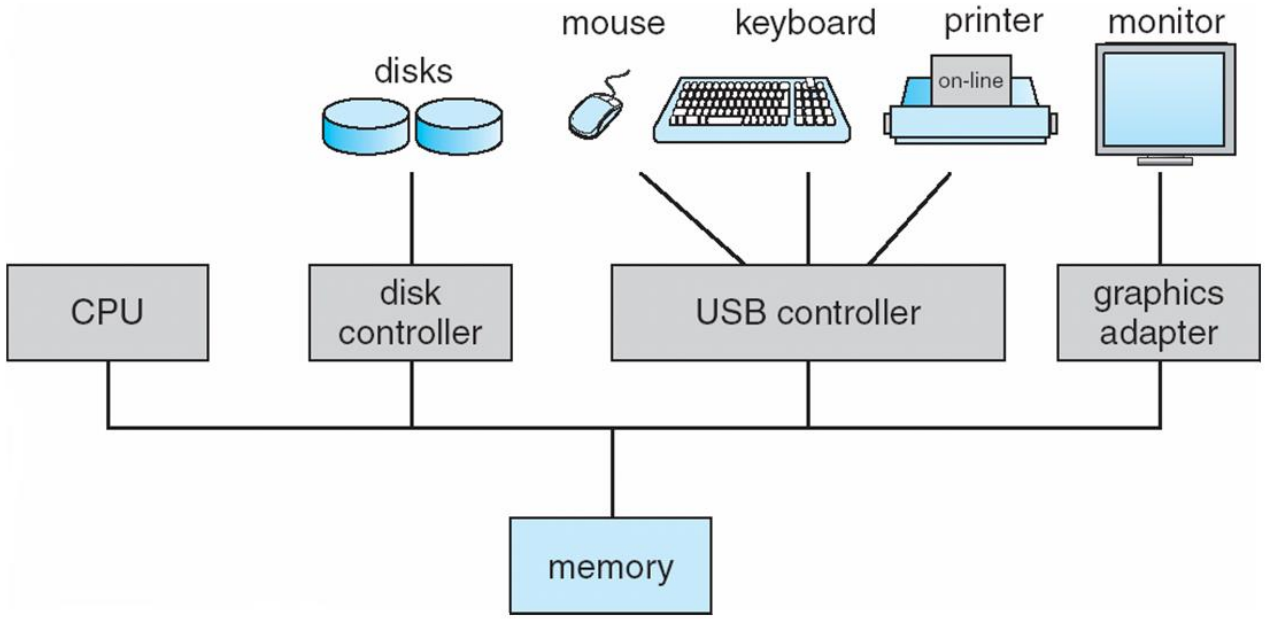
- Users of **dedicated systems** such as **workstations** (desktops or laptops) have dedicated resources but frequently use shared resources from **servers**
- **Servers** (e.g. **mainframe** or **minicomputer**) must allocate hardware resources in a manner that keeps all users happy
- **Handheld or mobile 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

Operating System Definition

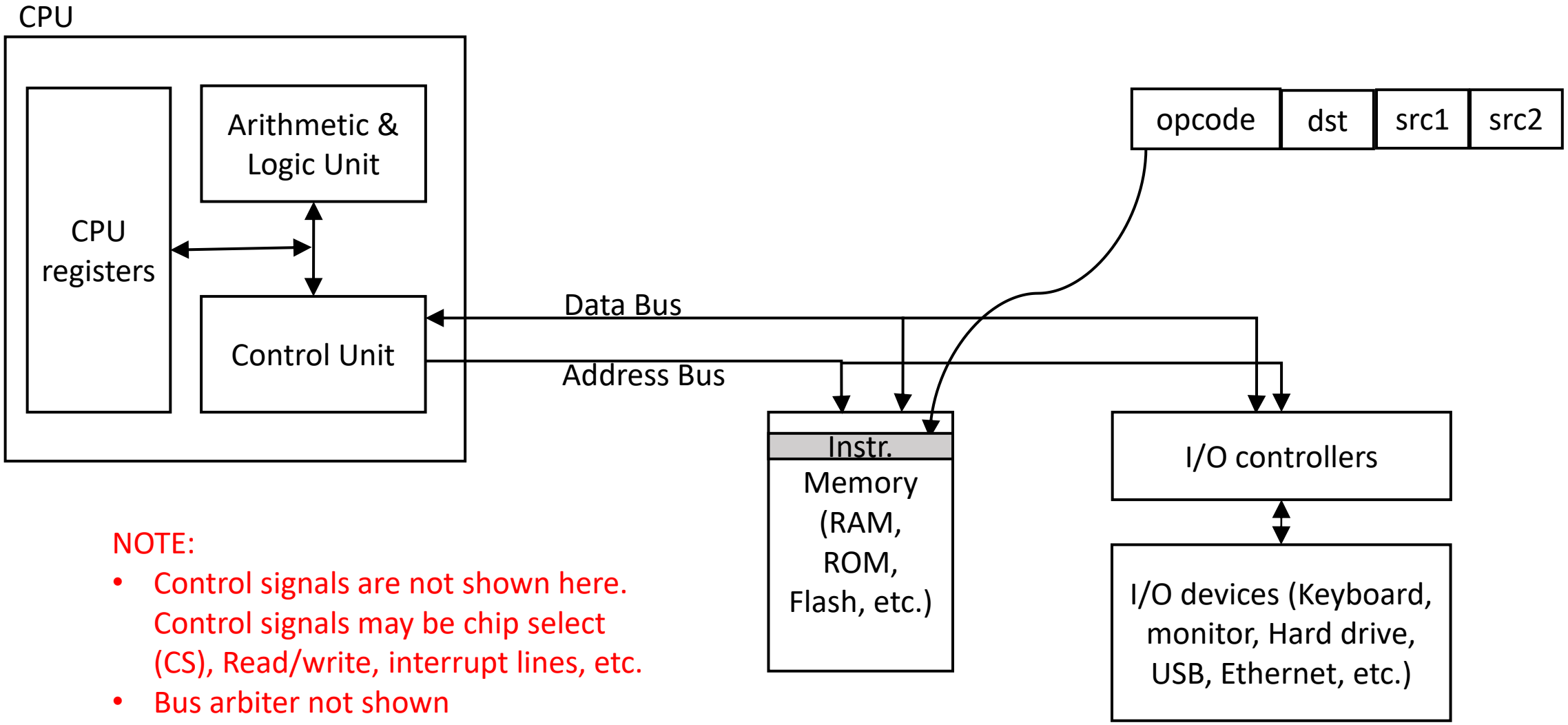
- No universally accepted definition
- “Everything a vendor ships when you order an operating system” is a good approximation
  - But varies wildly
- “The one program **running at all times** on the computer” is the **kernel**. The kernel runs while the CPU is in ***privileged or kernel mode*** (i.e. it can access any hardware resource and any memory location).
- Everything else runs the CPU in ***user mode*** (i.e. cannot freely access all memory locations, or hardware resources) and is either
  - a system program (ships with the operating system) , or
  - an application program.

Computer System Organization

- Computer-system operation
  - A computer system hardware has 3 main components:
    - One or more CPUs.
    - Memory (may be shared amongst CPUs and devices)
    - I/O Device controllers and I/O devices
  - They connect through common bus providing access to shared memory
  - Concurrent execution of CPUs and devices competing for memory cycles.



Computer system organization – The Von Neumann model



NOTE:

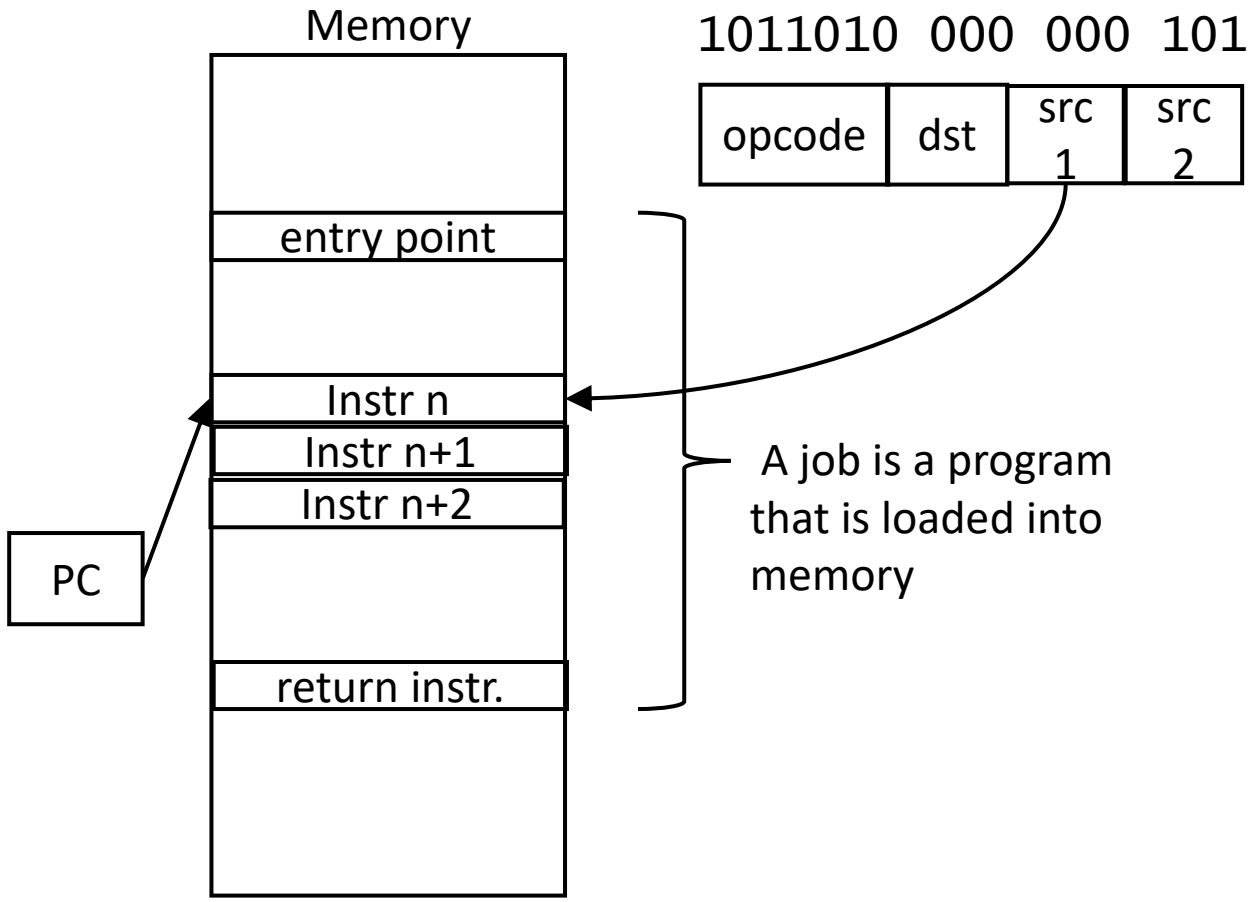
- Control signals are not shown here. Control signals may be chip select (CS), Read/write, interrupt lines, etc.
- Bus arbiter not shown

Central Processing Unit (CPU)

- **Control unit:** directs the overall operation of the computer
  - Keeps track of where the next instruction resides (using the program counter, PC)
  - Issues the signals needed to both read data from and write data to other units in the system (memory or I/O controllers)
  - Executes all instructions by issuing signals to the arithmetic and logic unit.
- **Arithmetic and Logic Unit (ALU):** performs all of the elementary computations, such as addition, subtraction, comparisons, and so on, that a computer provides.
- **CPU registers:** may be used as source operands and as destinations for the result.
- Reduced instruction set architecture (RISC) vs complex instruction set architecture (CISC)

Programs and jobs

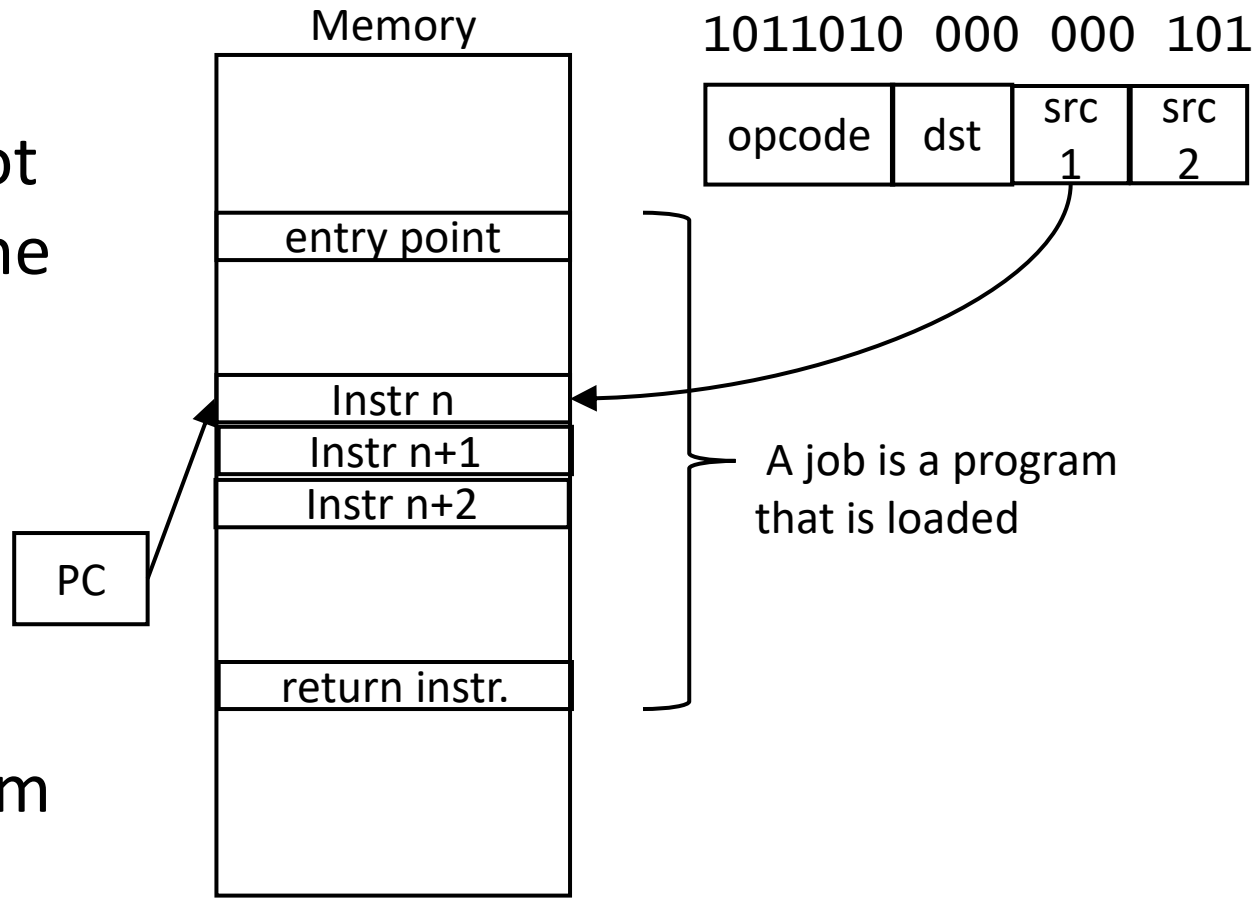
- A **program** is a collection of machine instructions (i.e. code), usually residing in the hard disk.
- A **job** or **process** is a program that is loaded into the system memory (code+data)
- The first instruction to execute is called the entry point (not necessarily at the beginning)



Programs and jobs – cont.

- The last instruction to execute is the **exit** or **return** instruction (not necessarily at the end), where the PC goes back to the initial caller.

- NOTE: The control unit can only fetch instructions from memory, i.e. it cannot directly access data in a hard disk, and thus a program is needed to access the hard disk's I/O registers and access the desired program.



Computer Startup

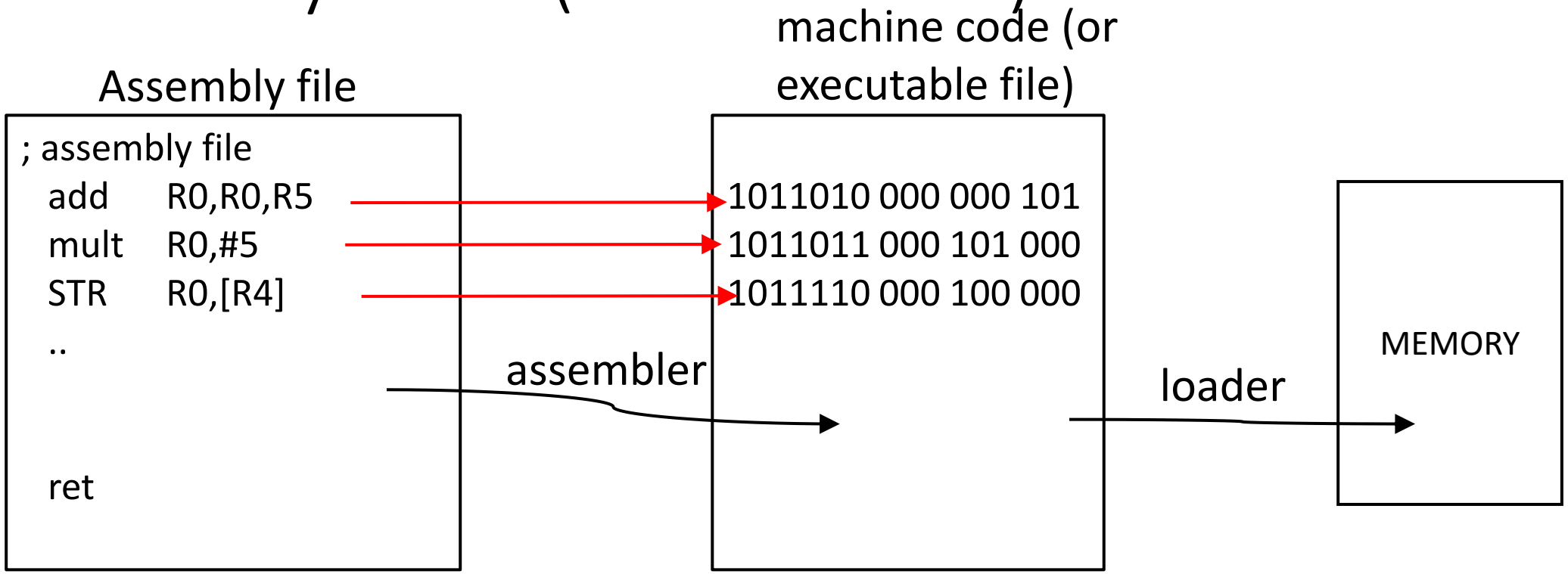
- **bootstrap program** (or bootstrap loader) executes immediately after power-up or reboot:
  - Typically stored in ROM or EPROM, generally known as **firmware**
    - Today, a combination of ROM + flash is most common.
  - Initializes necessary aspects of the system, e.g.:
    - Hardware resources required to read from the disk
    - serial or USB keyboard interface, etc.
  - Loads operating system kernel from non-volatile storage (e.g. hard disk) and starts execution

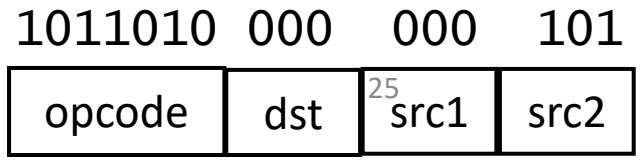
Machine Code

- A program may be stored in a hard-drive when it is not running but must be loaded (by a “loader”) into the main memory in order to be executed by the CPU (and thus becomes a job or process).

- In the early days of computing (1960’s), programs (or jobs) used to be written using machine code (i.e. binary ones and zeros such as 1011010000000101) into punch cards. The punch cards were then loaded into computers to be executed.

Assembly code (source code)





Assembly code

- Each CPU platform has its own assembly language.
- Instead of using machine codes (ones/zeros), **symbolic names** are used for instructions (opcode field) and source/destination registers.
- Operands that **are immediate values** (i.e. not source or dest. Registers) may be coded in binary, decimal, hexadecimal, etc.
- **Labels** may be used to specify certain program locations (addresses) and can be then used within instructions instead of the actual addresses. This is useful in accessing memory locations containing variables or in branch instructions.
- Has facilities to **allocate RAM locations** to certain variables.
- Allow definition of **macros** for ease of programming

C source code

- Machine code and assembly are often called first and second generation languages.
- C and C++ use **English-like statements** and are considered to be third generation languages. They work at much higher level than assembly.
- C/C++ source code may access any specific memory location using **pointers**.
- Code written in C or C++ cannot access **CPU registers**, however most vendors of C and C++ compilers offer facilities to access assembly instructions as if they are function calls, e.g. asm(“ add r2,1”);
- We will use C for most of our assignments.

I/O Operation

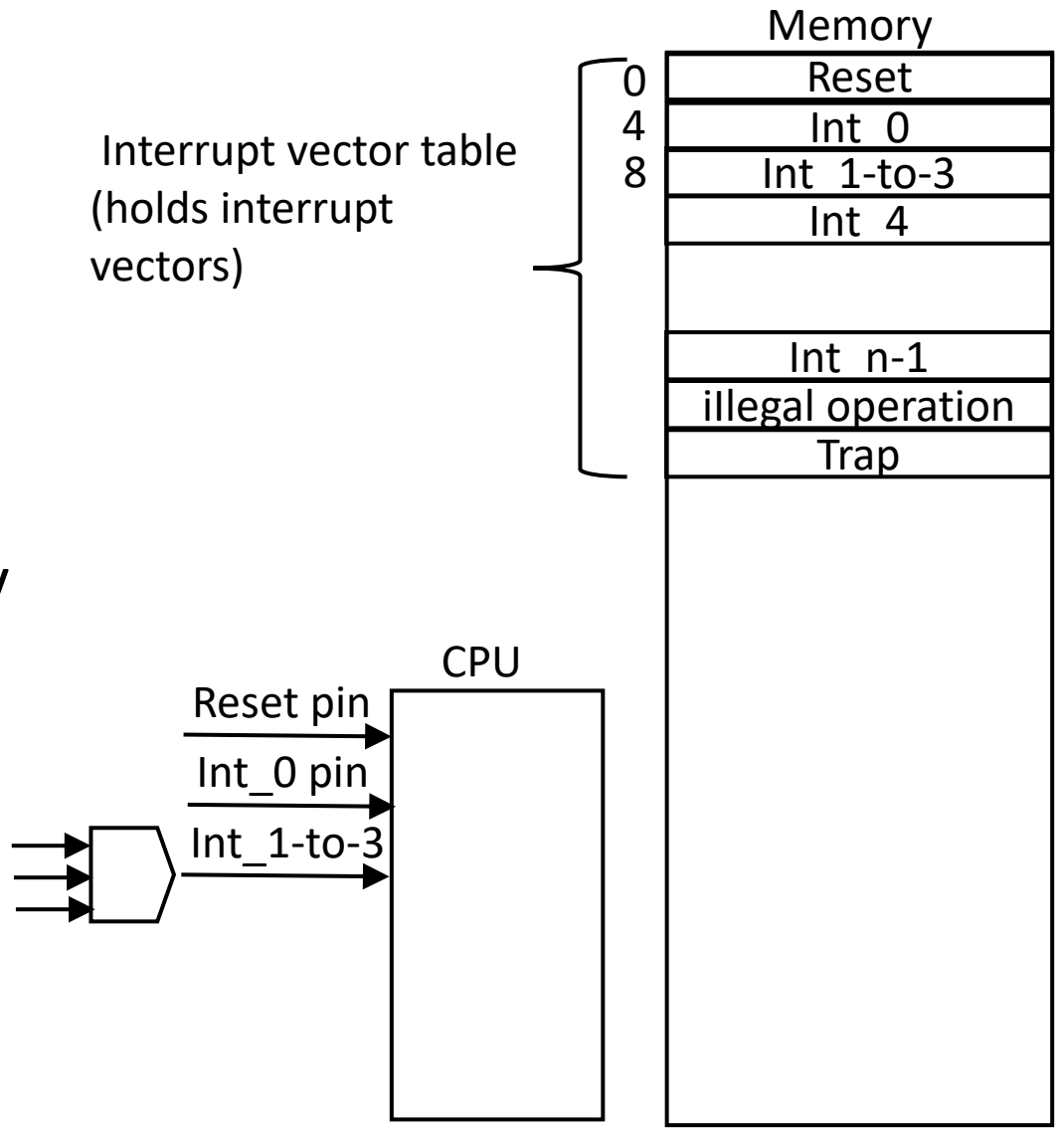
- I/O devices and the CPU can operate or execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer (usually memory mapped)
- CPU communicates with devices by:
  - Writing to control registers (to instruct the controller to do an operation)
  - Reading from status registers
  - Moving data from main memory to local buffers and vise versa.
- The CPU then waits till the operation is done via one of two methods:
  - **Polling:** The CPU continuously reads the status till it detects operation is complete.
  - **Interrupt-driven-I/O:** CPU goes off and executes other tasks. Device controller informs CPU of completion by asserting an interrupt pin.
- A **Device Driver** (part of the kernel software) exists for each device controller to manage I/O
  - Provides uniform interface between controller and rest of the kernel

Interrupts, exceptions and traps

- The terms interrupt and exception are often used interchangeably. Interrupts may occur due to 3 categories of causes:
  - Caused by an I/O controller (**hardware interrupts**). These are usually the result of an event external to the CPU.
  - The software trying to perform an **illegal operation**, e.g.:
    - Divide by zero
    - Accessing a memory region it's not supposed to access
    - An invalid opcode.
  - A **trap** is a software-generated interrupt/exception caused by a user program to request system services.
- When an exception occurs (due to any of the 3 causes), control (i.e. code execution) is transferred to the corresponding interrupt service routine.

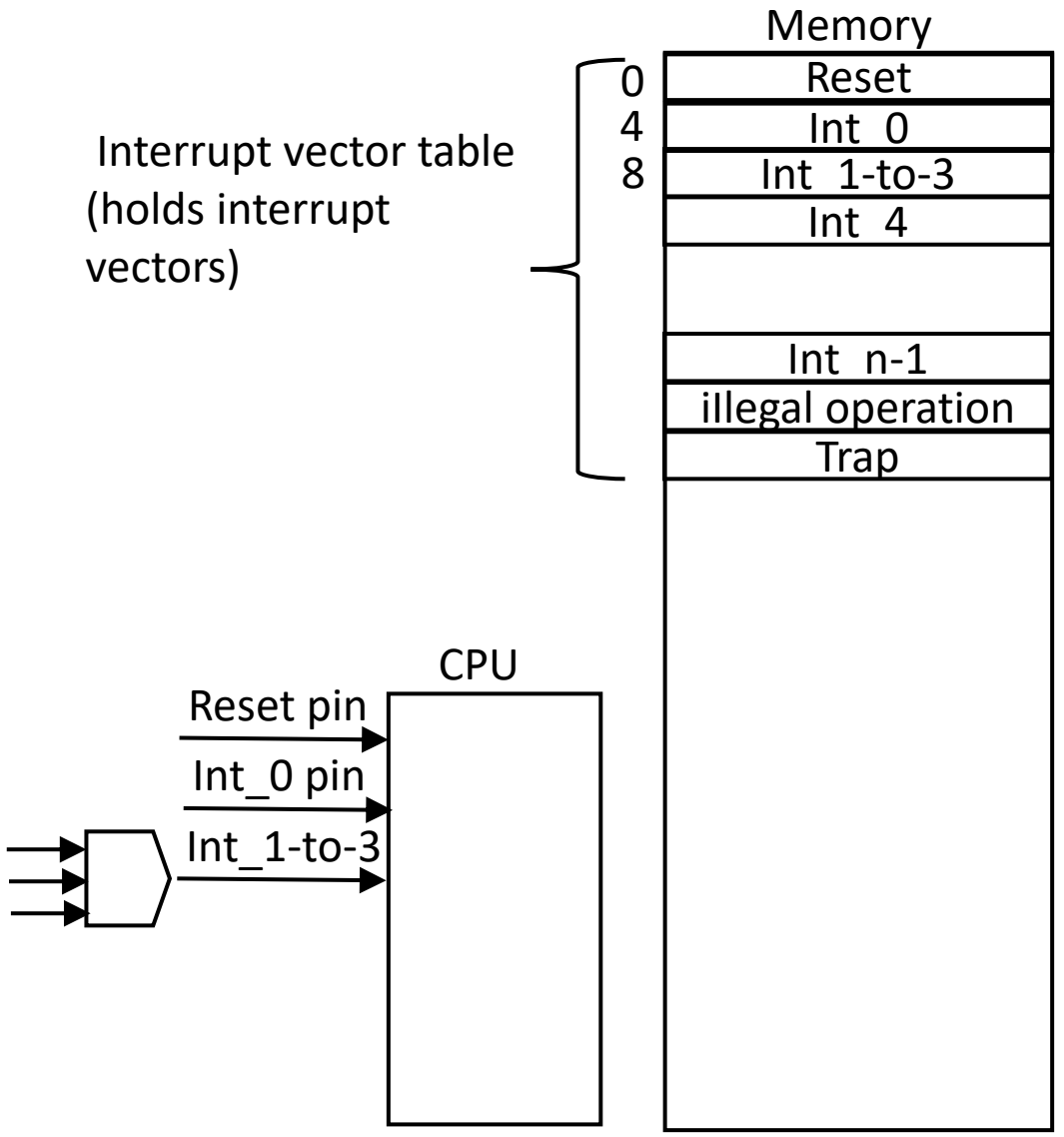
The Interrupt Vector Table

- A table that stores either (processor dependent):
  - Branch instructions to the **interrupt service routines (ISR)** – i.e. routines that can handle interrupts
  - Addresses of entry points for ISRs.
- The interrupt vector table may be located at address 0, or at the end of the memory address space (depending on the CPU configuration).
- It may be remapped during booting the OS kernel (why?)
- Generally, each interrupt line maps to a specific interrupt vector.
  - How many bytes?



The Interrupt Vector Table – cont.

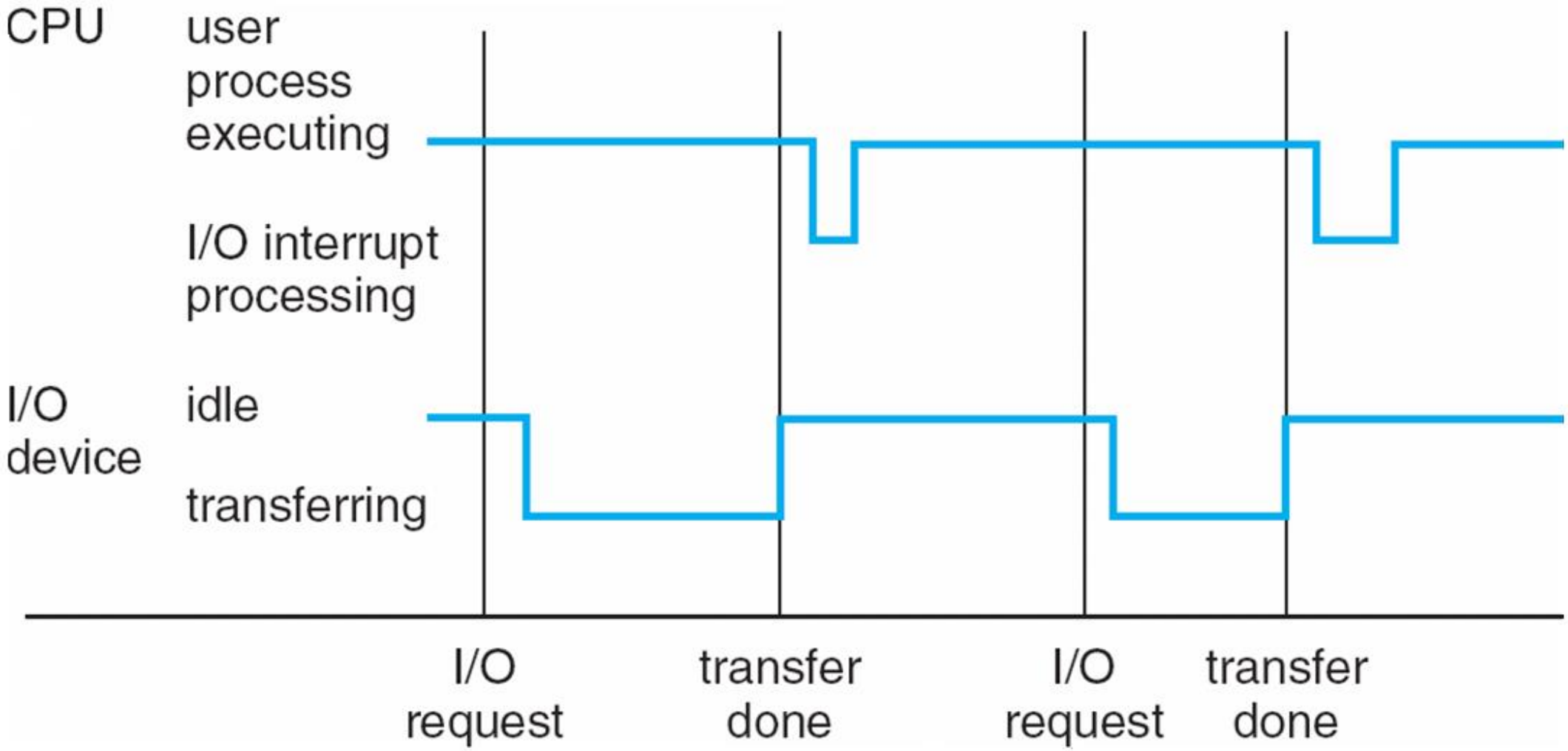
- What is stored in the reset vector?
- There is usually one vector for the trap and a few for illegal operations.
- Multiple interrupt lines may **share** a single interrupt vector (as in interrupt lines 1 to 3 in the example to the right)
- In such a case the ISR would need to detect which particular interrupt pin is responsible and then call the corresponding interrupt handler.



Interrupt Handling

- The interrupt service routine (ISR) resides within the operating system's kernel and it needs to preserve the state of the CPU registers by **storing** them into the main memory.
  - This includes saving the program counter (PC) register.
  - Note that in many CPU platforms some interrupt types may have hardware support, in which the CPU registers are saved automatically. This is often achieved by having multiple banks of CPU registers.
- Prior to the ISR's exit, it then needs to **restore** the CPU registers. The PC needs to be the last register to be restored, why?

Interrupt Timeline



Storage Definitions and Notation Review

The basic unit of computer storage is the **bit**. A bit can contain one of two values, 0 and 1. All other storage in a computer is based on collections of bits. Given enough bits, it is amazing how many things a computer can represent: numbers, letters, images, movies, sounds, documents, and programs, to name a few. A **byte** is 8 bits, and on most computers it is the smallest convenient chunk of storage. For example, most computers don't have an instruction to move a bit but do have one to move a byte. A less common term is **word**, which is a given computer architecture's native unit of data. A word is made up of one or more bytes. For example, a computer that has 64-bit registers and 64-bit memory addressing typically has 64-bit (8-byte) words. A computer executes many operations in its native word size rather than a byte at a time.

Computer storage, along with most computer throughput, is generally measured and manipulated in bytes and collections of bytes.
A **kilobyte**, or **KB**, is 1,024 bytes
a **megabyte**, or **MB**, is $1{,}024^2$ bytes
a **gigabyte**, or **GB**, is $1{,}024^3$ bytes
a **terabyte**, or **TB**, is $1{,}024^4$ bytes
a **petabyte**, or **PB**, is $1{,}024^5$ bytes

Computer manufacturers often round off these numbers and say that a megabyte is 1 million bytes and a gigabyte is 1 billion bytes. Networking measurements are an exception to this general rule; they are given in bits (because networks move data a bit at a time).

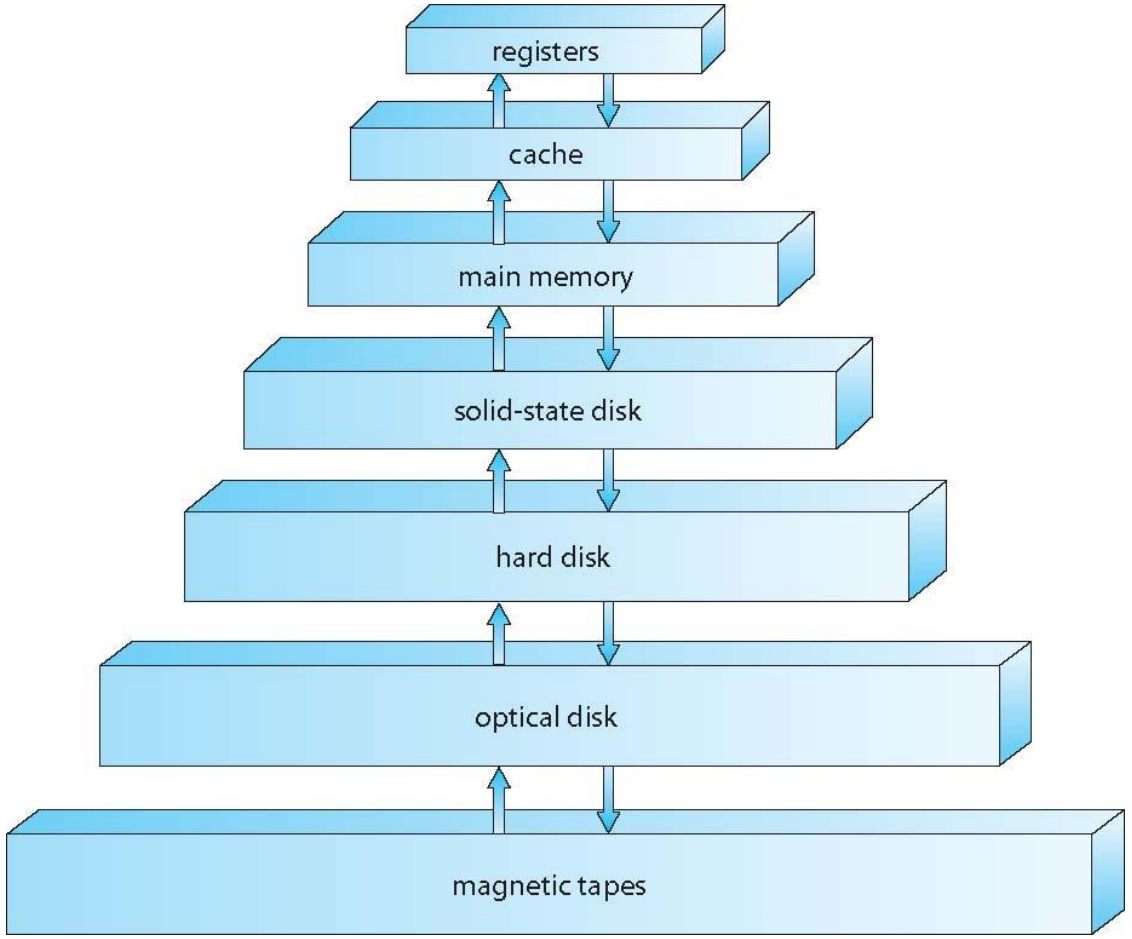
Storage Structure

- Main memory – is the only storage media that the CPU can access **directly**:
  - RAM: random access memory, volatile
  - ROM: read only memory, non-volatile (e.g. ROM, EEPROM or flash)
- Secondary storage – extension of main memory that provides large nonvolatile storage capacity, however, the CPU can only access this memory **indirectly** via a device controller (using its control/status and data interfaces)
  - **Hard disks** – rigid metal or glass platters covered with magnetic recording material.
    - Disk surface is logically divided into tracks, which are subdivided into sectors.
  - **Solid-state disks** – faster than hard disks, also nonvolatile
    - Becoming more popular

Storage Hierarchy

- Storage systems organized in hierarchy
  - Speed
  - Cost

  (usually, the larger the memory, the slower it is)
- **Caching** – As a concept, it means copying information into faster storage system; main memory can be viewed as a cache for secondary storage



Caching

- Important principle, performed at **many levels** in a computer (in hardware, operating system, software)
- Information in use is copied from slower to faster storage temporarily
  - Faster storage (cache) checked first to determine if information is there
    - If it is, information used directly from the cache (cache hit).
    - If not, data copied to cache and used there (cache miss).
- Why is it advantageous to use cache?
- Cache management is an important design problem
  - Cache size (affects speed + cost)
  - Replacement policy (e.g. LIFO, LRU, etc.)

Direct Memory Access Structure

- Used for high-speed I/O devices able to transmit information at close to memory speeds (e.g. gigabit ethernet)
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
- Only one interrupt is generated per block of data, rather than the one interrupt per word or byte.

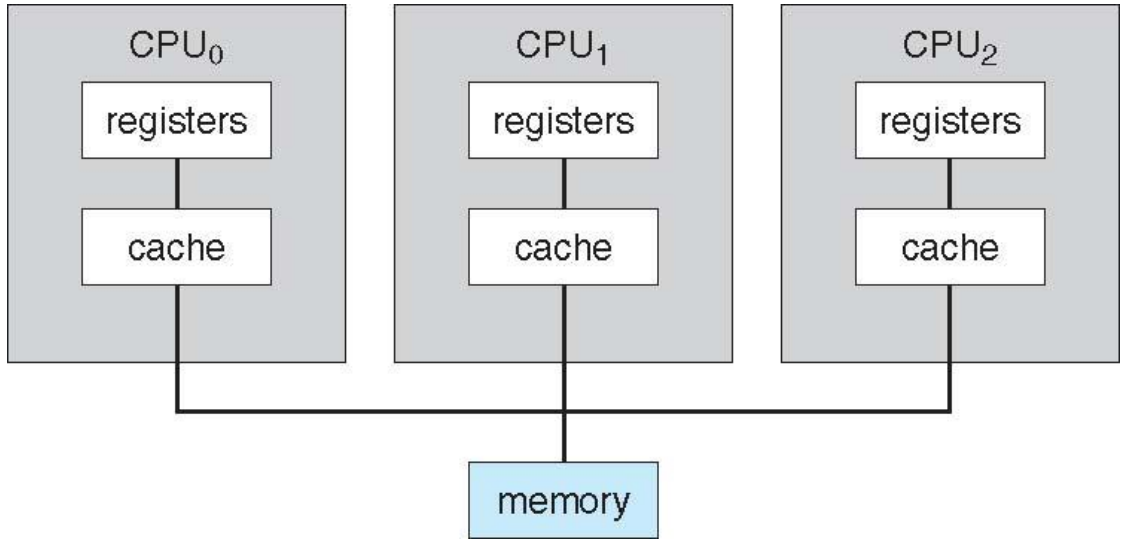
Computer-System Architecture

- Many systems use a **single general-purpose processor**
  - Most systems have special-purpose processors as well (e.g. a GPU or a DSP), but these do not make the system a multiprocessor system.

- **Multiprocessors** systems growing in use and importance
  - Also known as **parallel systems**, **tightly-coupled systems**
  - Advantages include:
    1. **Increased throughput**
    2. **Economy of scale**
    3. **Increased reliability** – graceful degradation or fault tolerance

Computer-System Architecture – cont.

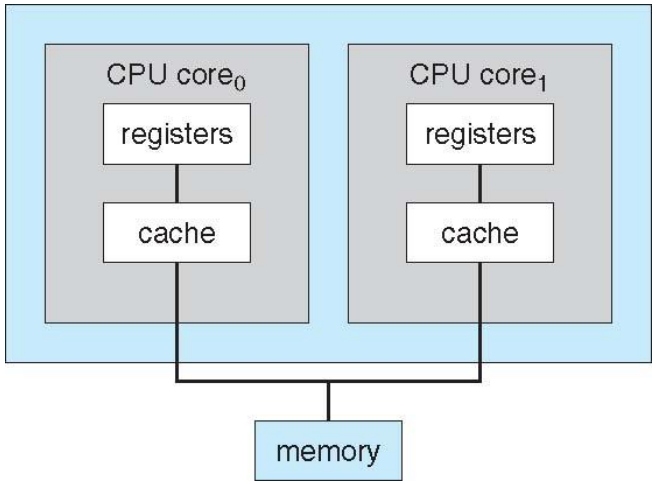
- **Multiprocessors** - Two types:
  1. **Asymmetric Multiprocessing** – processors are not treated as equal.
     - Processors may be dedicated to specific tasks
     - e.g. boss and worker processors
  2. **Symmetric Multiprocessing (SMP)** – all processors are treated equally
     - Single instance of the OS.
     - Each processor is **capable** of performing any task, such as handling interrupts, running the OS kernel, running applications, etc.



Symmetric multiprocessors

Multi-Core Designs

- A **multicore** system may have multiple cores in a single chip, and is thus a multiprocessor system.
- Systems may be built of multiple chips, each with multiple cores.

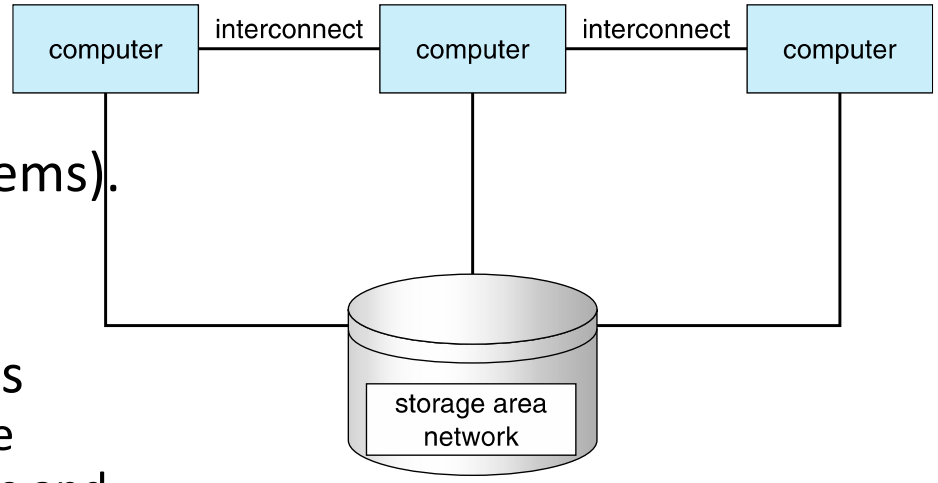


Blade systems

- Blade systems are those containing separate CPU boards, I/O boards and storage (secondary) **in one chassis**.
  - Each CPU board operates completely **independent** of other boards, i.e. an application program must be aware of the presence of the many CPU boards.
  - **Common storage** device, which blade nodes use for booting.
  - Chassis provides power, cooling and interconnect.

Clustered Systems

- Each node has its **own storage** (local), I/O and running its OS independently.
- A utility software is needed for communicating between nodes.
- Do not share memory (in contrast to multiprocessor systems).
- Usually **connected via LAN or SAN**, and sharing common storage
- Provides a **high-availability** service which survives failures
  - **Asymmetric clustering** has one machine in hot-standby mode
  - **Symmetric clustering** has multiple nodes running applications and also monitoring each other (more efficient use of hardware)
- Some clusters are for **high-performance computing (HPC)**
  - Applications must be written to use **parallelization**
- Some have **distributed lock manager** (**DLM**) to avoid conflicting operations on the common storage devices.



Operating System Structure

- **Multiprogramming batch systems**
  - A bit historical
  - Needed for efficiency: Single user cannot keep CPU and I/O devices bu at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has c to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via **job scheduling**
  - When it has to wait (for I/O for example), OS switches to another job
  - Typically used **non-preemptive = cooperative** multitasking

- **Timesharing/interactive systems**
  - Logical extension in which CPU switches jobs so frequently that users c interact with each job while it is running, creating **interactive** computi (**response time** should be < 1 second) -> **preemptive**
  - Each user has at least one program executing in memory ⇨ **process**
  - If several jobs ready to run at the same time ⇨ **CPU scheduling** select one of them
  - If processes don't fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes by partially (not fully) loading them in memory