Concepts of Computer Science

Architectures

Chapter Goals

- Lecture 1:
  - The von Neumann architecture
  - Components of the architecture
  - The Fetch-Decode-Execute cycle
- Lecture 2:
  - Primary and secondary memory
  - Parallelism
- Lecture 3:
  - Heads-up on Low Level programming

- Lecture 1:
  - From theoretical to physical
  - The von Neumann architecture
  - Components of the architecture
  - The Fetch-Decode-Execute cycle

Stored-Program computers

Stored-program

- Programs can be **represented and stored as data**. There is no significant difference between instructions and data, therefore we can **store programs in our main data memory**.

We send information around our computer using a connection called a **bus**. Remember the multiplexer last week?

The von Neumann architecture

- A stored-program computer where instructions and data **cannot be fetched at the same time**, as they use a shared bus.

- This causes a **bottleneck** on the processing as we need to perform fetching of instructions and data in **cycles.**

The von Neumann architecture

- **Control unit**: Directs operations of the processor, using the instruction register and program counter.
- **Arithmetic/Logic unit**: Performs basic arithmetic and logical operations on processor's registers.
- **Memory**: Main data (including program) memory.
- **Bus**: Connection between CPU, memory, and peripherals.

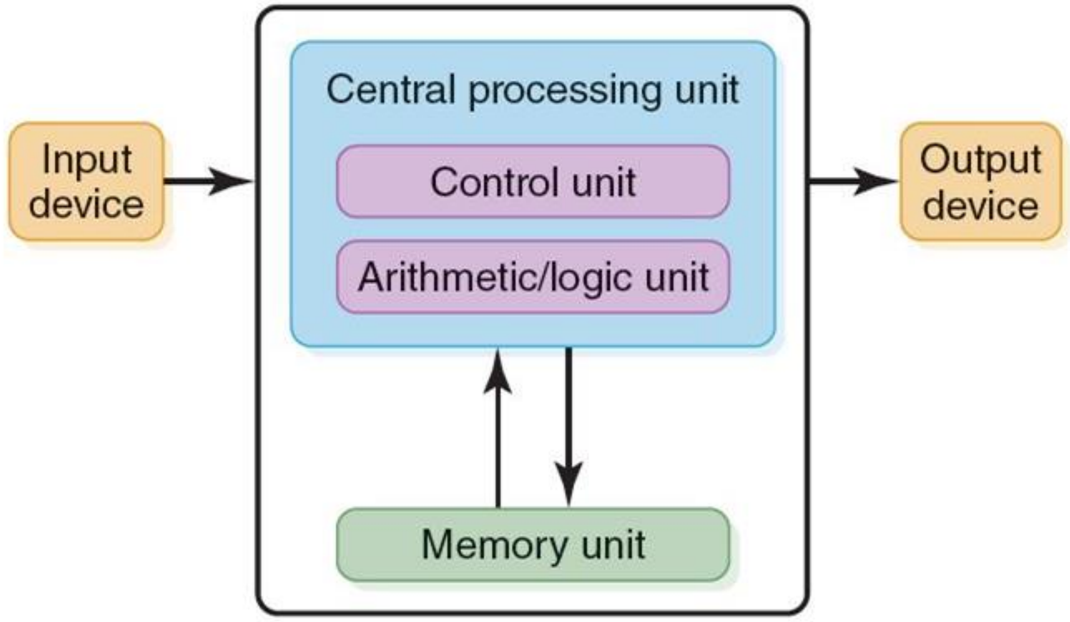


Image from Computer Science Illuminated. Jones and Bartlett

Memory

- Collection of cells which **store information**. Each cell has a **unique physical address**.
- Most computers can address individual bytes of information.
- Often we want to read from / write to these memory locations.

Address	Content
0000 0000	1111 1101
0000 0001	1110 1001
0000 0010	0000 0000
0000 0011	0110 1010
…	…
1111 1101	1010 1111
1111 1110	0100 0110
1111 1111	1110 1101

Memory

- What does the 01000110 mean in memory address 11111110?
- It could be an instruction, a natural number, a signed integer, a character, part of an image…
- No way to really know without providing further context.

Address	Content
0000 0000	1111 1101
0000 0001	1110 1001
0000 0010	0000 0000
0000 0011	0110 1010
…	…
1111 1101	1010 1111
1111 1110	0100 0110
1111 1111	1110 1101

Arithmetic / Logic unit (ALU)

- Performs **basic arithmetic operations**, such as addition and multiplication.
- Performs **logical operations**, such as AND, OR, and NOT.
- Modern ALUs have a small number of small storage units called **registers**, which can be accessed faster than main memory.

Control Unit

- This is the organising force in the computer. It provides **timing signals** and **control signals** which directs the ALU, memory, and I/O devices to respond to instruction.

- It also contains the **Instruction Register** and the **Program Counter.** These are both critical in the execution of instructions.

Control Unit

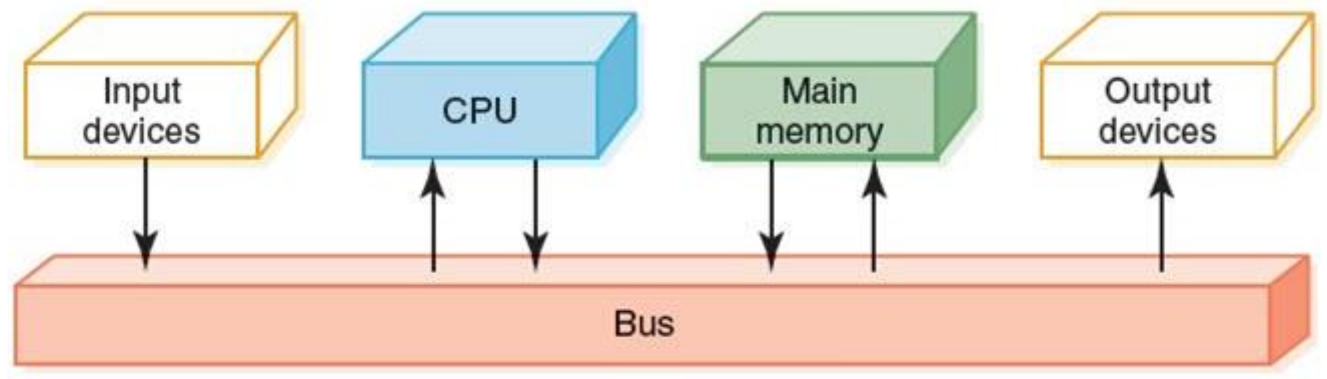
- **Instruction Register**
  - A register which **contains the instruction** that is currently **being executed**.
- **Program Counter**
  - A register which **contains the address of the next instruction** to be executed.
- These work together as part of the Fetch-Decode-Execute cycle to get the computer doing things.

Central Processing Unit (CPU)

- Commonly the control unit and ALU are contained within a singular component, named the **Central Processing Unit**
- The CPU also often contains additional registers and cache memory, with high communication speeds made possible by their close physical proximity.

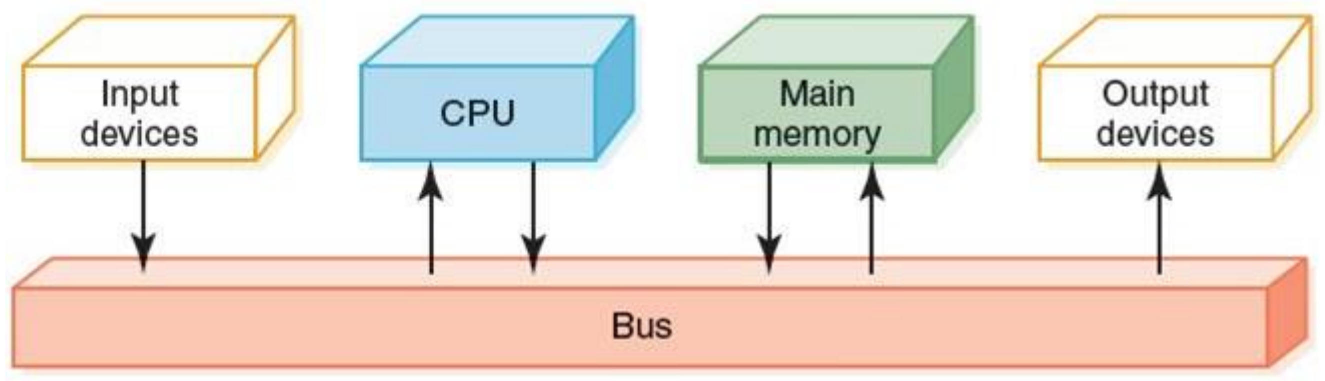
Bus (generic)

- Communication system that transfers data
- Can be between components inside a computer or between computers
- Includes the medium (wires, fibre, etc.) and protocols (rules)



Front-Side Bus

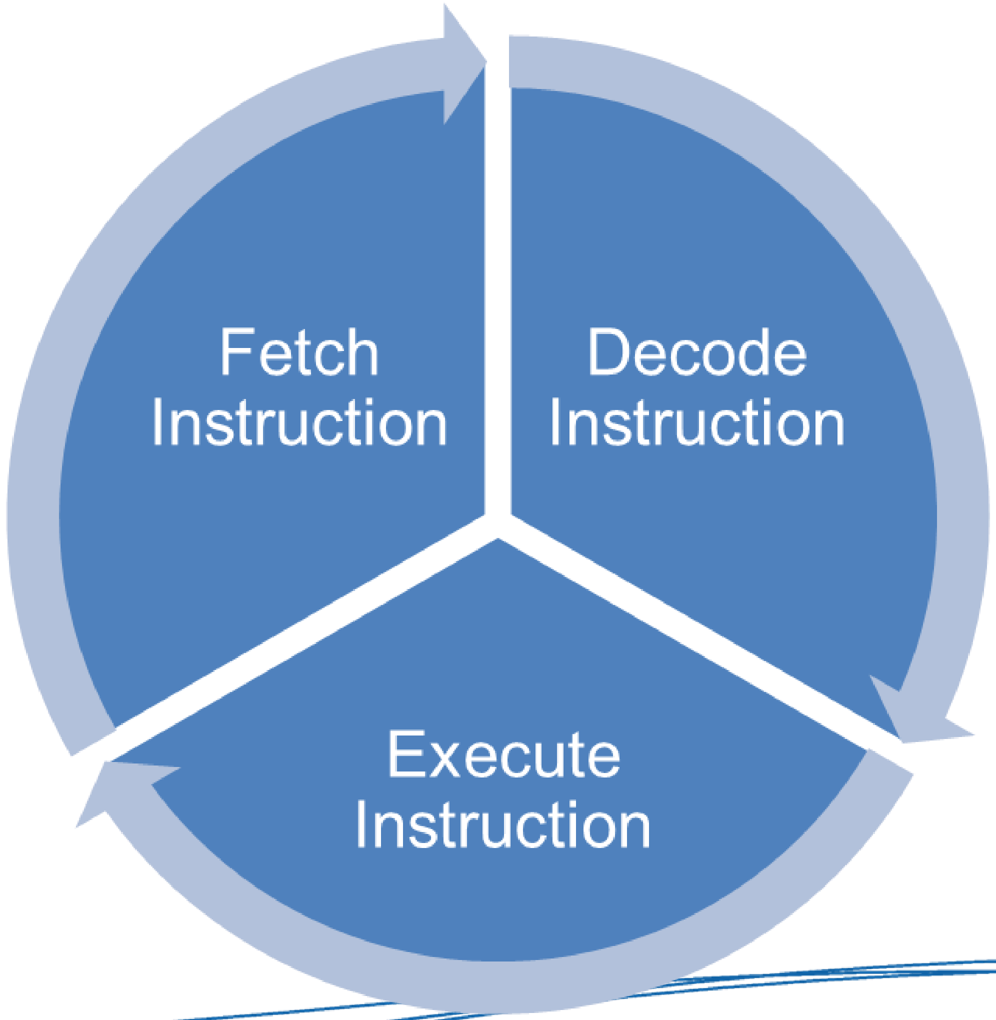
- This is **a dedicated bus** which connects the CPU, main memory, and other internal components (including other buses).
- This usually allows the communication of a **word-size** of information at any given time, controlled by the control unit with its **control signals** and **timing signals**.



The Fetch-Decode-Execute Cycle

- A cycle of CPU actions which allow the computer to perform operations. Sometimes also called the **Instruction Cycle:**
  - **Fetch** the next instruction
  - **Decode** the instruction
  - Get data from memory (if needed)
  - **Execute** the instruction

The Fetch-Decode-Execute Cycle



The Fetch-Decode-Execute Cycle

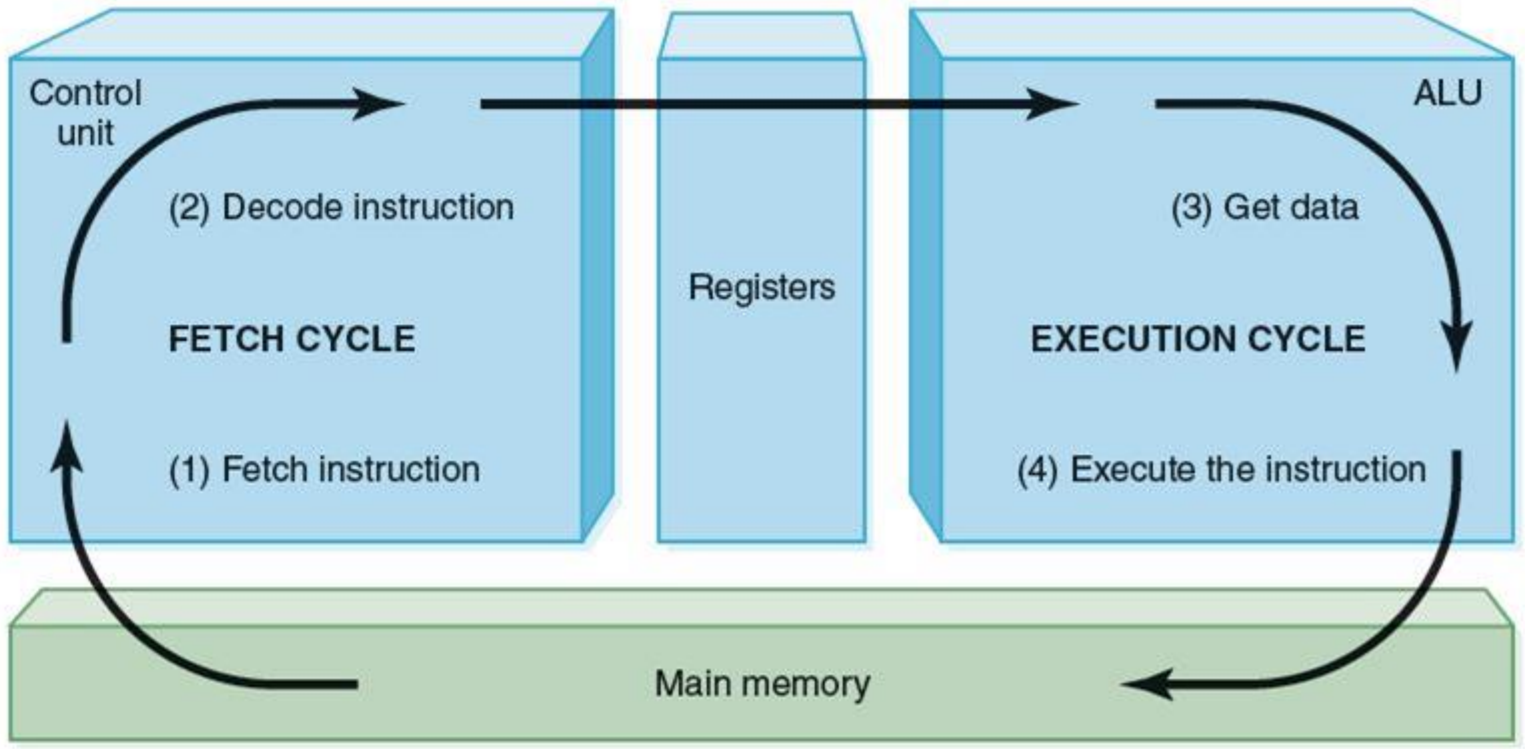
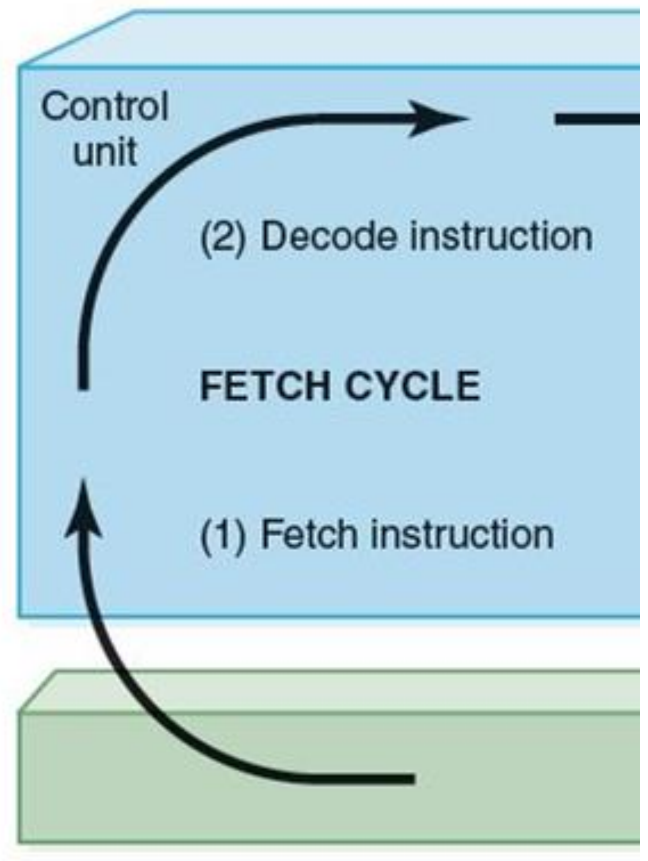


Image from Computer Science Illuminated. Jones and Bartlett

The Fetch-Decode-Execute Cycle

- **Fetch**: Fetch instruction from location pointed to by program counter. Place instruction into instruction register and update program counter to point to the next memory location.

- **Decode**: The instruction (represented as a binary bit string) within instruction register is interpreted into an operation to perform.

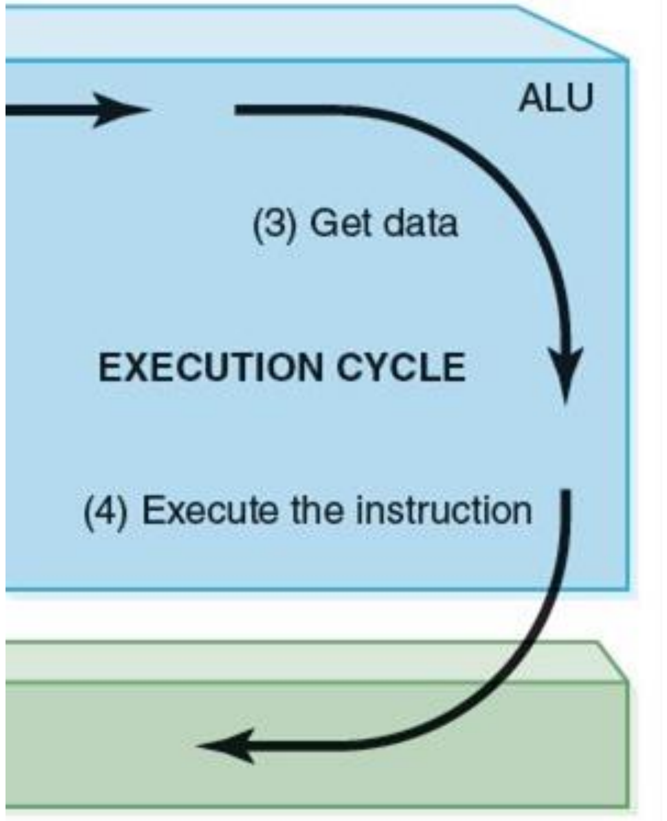


The Fetch-Decode-Execute Cycle

- **Read Memory**: If required, auxiliary data is fetched from main memory to be processed, and then placed into data registers.

- **Execute**: CPU's Control Unit passes the decoded instruction to functional units of the CPU, performing the actions required by the instruction.

- The cycle then repeats, fetching the next instruction.



- Lecture 2:
  - Primary and secondary memory
  - Parallelism

Types of Memory

- **Random Access Memory (RAM)**: Each location can be accessed and changed.
- **Read Only Memory (ROM)**: Each location can be accessed, but not changed/modified.

RAM is volatile, it stores the values with electrical voltage and when this voltage is removed the information is lost.

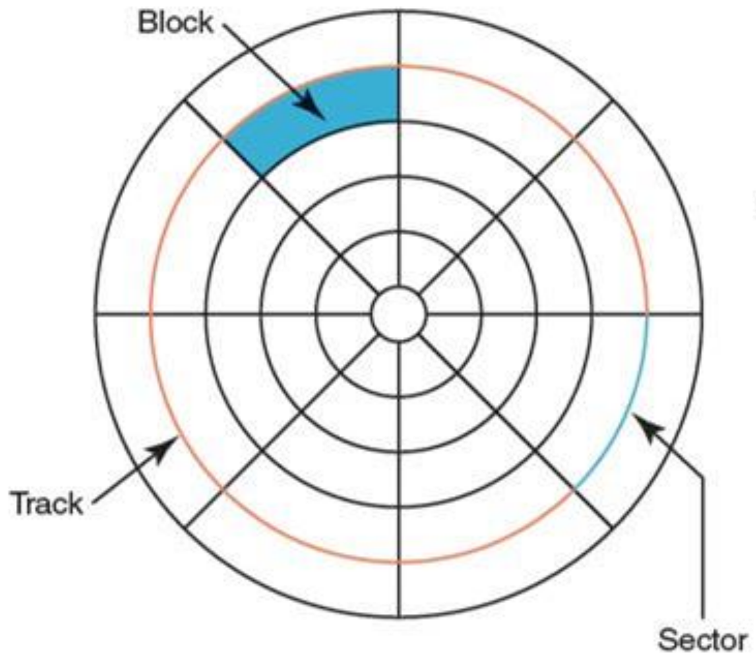
ROM is non volatile, the values are ‘hard coded’ into the address.

Types of Memory

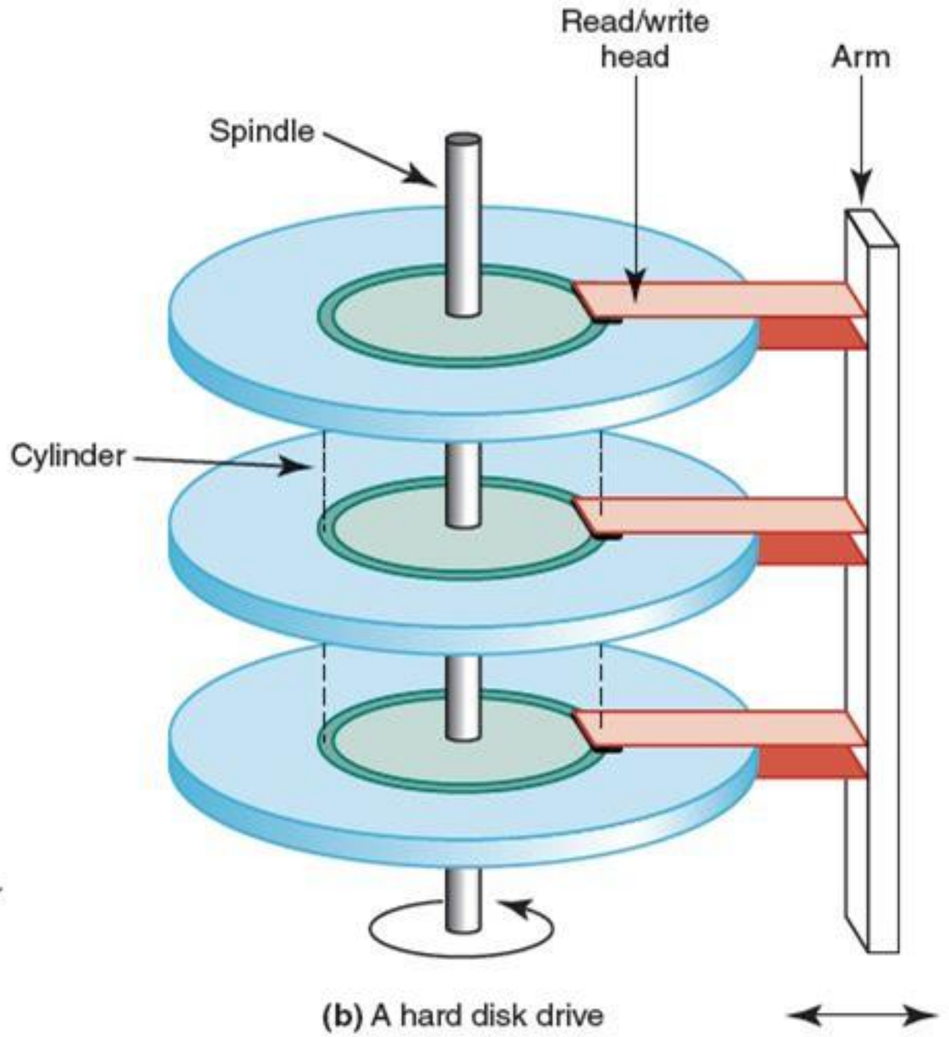
- **Main memory**: Storage location which can be **directly accessed by the CPU**, usually physically located on the computer to allow for extremely fast access to **active program and data** information.

- **Secondary memory**: Storage locations which are used for non-active information stored over longer periods of time. This can include devices such as **magnetic disks**, **optical disks**, and **solid state drives**.

Magnetic Disk Storage



(a) A single disk



(b) A hard disk drive

Magnetic Disk Layout

- **Track:** A physical, circular path on the disk surface. This is where the data is written/read.
- **Sector:** A sub-division of a track, storing a fixed amount of data.
- **Block:** Chunk of data on disk, identified by both track and sector. The minimal storage unit of a disk.
- **Cylinder:** A set of concentric tracks on all disks in a drive.

Magnetic Disk Access

- It takes time to access the information on the physical disk.
- **Seek Time**: time for read/write head to move over correct track.
- **Latency**: time for sector to be rotated under read/write head.
- **Access Time**: time for block to start being read (seek + latency)
- **Transfer Rate**: rate at which data moves from disk to memory.

Optical Disks

- **Compact disk (CD):** A disk that uses a laser to read information stored optically on a plastic-coated disk; data is evenly distributed around spiral track.
- **Digital versatile disk (DVD):** Used for storing audio and video.
- **Blu-ray:** Higher capacity DVD (using a blue light laser) allowing higher resolution video, etc.

Solid State Drives

- Uses **integrated circuits** to store data, rather than the magnetic or optical elements seen previously.

- Typically uses **flash memory** to store the information. This stores the value even when the power is removed, great for secondary storage.

- As it uses logical gates (transistors) to store data, there are no moving parts. This makes them much faster and quieter than mechanical disk drives, and provides impact resistance.

Parallel Computing

- We discussed the Fetch-Decode-Execute cycle, and the limitations of the von Neumann bottleneck.
- In some scenarios it is beneficial to perform some operations **concurrently** (at the same time).
- This is called **parallelism**, and there are four common approaches to this task, depending on the application…

Parallel Computing

- Approaches to parallelism:
  - Bit-level
  - Instruction-level
  - Data-level
  - Task-level

- Modern computers often have multiple **cores** or processing units. Each processor is then able to carry out it’s own instructions.

Bit-level Parallelism

- Individual bits can be processed simultaneously

- For example,
  - If our computer’s **word size** is 8 bits, it would require 2 operations to process a 16-bit value.
  - With a 16-bit word size we could do this task in a single operation.

  - Consider the 8-bit adder from last week. To add 16-bit numbers would require two applications of this adder. But if we had a 16-bit adder then we only need apply it once.

Instruction-level Parallelism

- Some instructions can be executed independently

- This can take two main forms:
  - **Pipelining** is the concept of overlapping instructions which can be **carried out in series**. For example, in the Fetch-Decode-Execute cycle, can we start fetching the next instruction while our current one is decoding?

  - **Superscalar** is the concept of having multiple execution units which allows us to execute **multiple instructions per cycle**, increasing the **bandwidth** of our processor.

Data-level Parallelism

- Application of single instruction to many pieces of data simultaneously. **Single Instruction, Multiple Data** (SIMD)

- For example,
  - Calculating the average grades of all students at the same time.
  - Apply the same instructions to each student record in parallel.

Task-level Parallelism

- Entire tasks can be completed simultaneously, either with the same data or on different data.

- **Multiple instructions, multiple data** (MIMD)

- For example,
  - Different users carrying out different searches on Google