Introduction

A **computer** is a general-purpose electronic device that can be programmed to carry out a set of arithmetic or logical operations automatically. Since a sequence of operations can be readily changed, the computer can solve more than one kind of problem.

Conventionally(بشـكل تقليدي), a computer consists of at least one processing element, typically a central processing unit (CPU), and some form of memory. The processing element carries out arithmetic and logic operations, and a sequencing and control unit can change the order of operations in response to stored information. Peripheral devices allow information to be retrieved from an external source, and the result of operations saved and retrieved.

Mechanical analog computers started appearing in the first century (القرن) and were later used in the medieval era (العصـر) for astronomical calculations. In World War II, mechanical analog computers were used for specialized military applications such as calculating torpedo aiming. During this time the first electronic digital computers were developed. Originally they were the size of a large room, consuming as much power as several hundred modern personal computers (PCs).

Modern computers based on integrated circuits are millions to billions of times more capable than the early machines, and occupy a fraction of the space. Computers are small enough to fit into mobile devices, and mobile computers can be powered by small batteries. Personal computers in their various forms are icons of the Information Age and are generally considered as “computers”. However, the embedded computers found in many devices from MP3 players to fighter aircraft and from electronic toys to industrial robots are the most numerous.

Etymology

The first known use of the word "computer" was in 1613 in a book called ***The Yong Mans Gleanings*** by English writer **Richard Braithwaite**: "I have read the truest computer of Times, and the best Arithmetician that ever breathed, and he reduces thy days into a short number." It referred to a person who carried out calculations, or computations. The word continued with the same meaning until the middle of the 20th century. From the end of the 19th century the word began to take on its more familiar meaning, a machine that carries out computations.

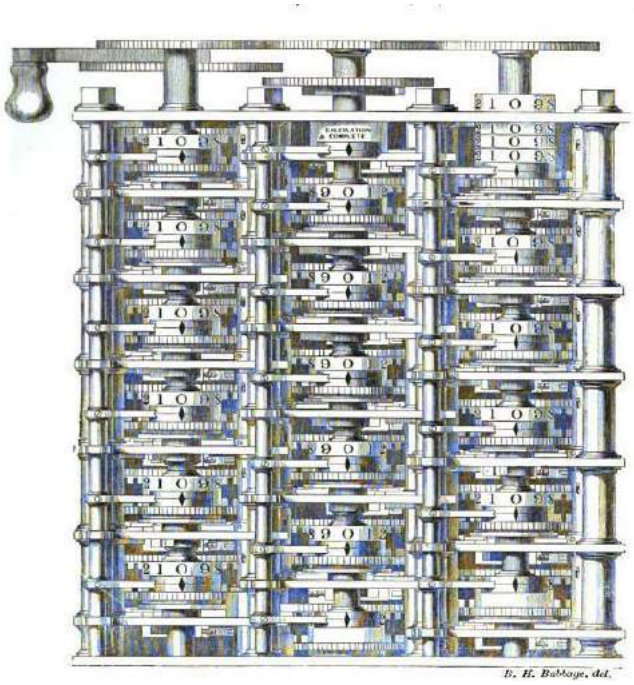
First general-purpose computing device

Charles Babbage, an English mechanical engineer and polymath, originated the concept of a programmable computer. Considered the "father of the computer", he conceptualized and invented the first mechanical computer in the early 19th century. After working on his revolutionary difference engine, designed to aid in navigational calculations, in 1833 he realized that a much more general design, an Analytical Engine, was possible.

The input of programs and data was to be provided to the machine via punched cards, a method being used at the time to direct mechanical looms such as the Jacquard loom. **For output,** the machine would have a printer, a curve plotter and a bell. The machine would also be able to punch numbers onto cards to be read in later. The Engine incorporated an arithmetic logic unit, control flow in the form of conditional branching and loops, and integrated memory, making it the first design for a general-purpose computer that could be described in modern terms as Turing-complete.

The machine was about a century ahead of its time. All the parts for his machine had to be made by hand — this was a major problem for a device with thousands of parts. Eventually, the project was dissolved with the decision of the

British Government to cease funding. Babbage's **failure to** complete the analytical engine can be chiefly attributed to difficulties not only of politics and financing, but also to his desire to develop an increasingly sophisticated computer and to move ahead faster than anyone else could follow. Nevertheless, **his son, Henry Babbage,** completed a simplified version of the analytical engine's computing unit (the mill) in 1888. He gave a successful demonstration of its use in computing tables in 1906.



A portion of Babbage's Difference engine.

Later analog computers

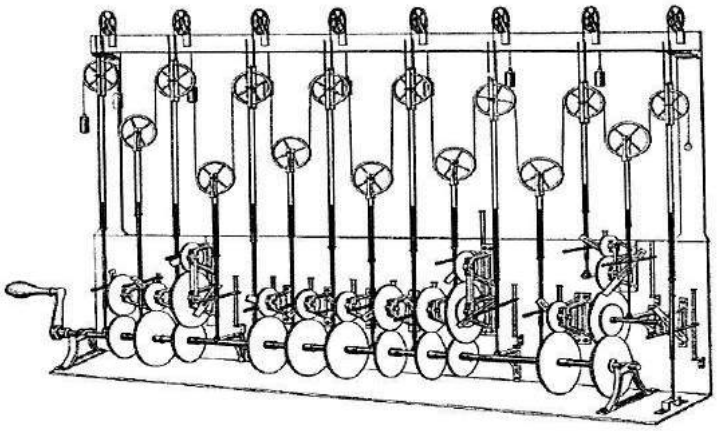
During the first half of the 20th century, many scientific computing needs were met by increasingly sophisticated analog computers, which used a direct mechanical or electrical model of the problem as a basis for computation. However, these were not programmable and generally lacked the versatility and accuracy of modern digital computers.

The first modern analog computer was a tide-predicting machine, invented by Sir William Thomson in 1872. The differential analyzer, a mechanical analog

computer designed to solve differential equations by integration using wheel-and-disc mechanisms, was conceptualized in 1876 by James Thomson, the brother of the more famous Lord Kelvin.

The art of mechanical analog computing reached its zenith with the differential analyzer, built by H. L. Hazen and Vannevar Bush at MIT starting in 1927. This built on the mechanical integrators of James Thomson and the torque amplifiers invented by H. W. Nieman. A dozen of these devices were built before their obsolescence became obvious.

By the 1950s the success of digital electronic computers had spelled the end for most analog computing machines, but analog computers remain in use in some specialized applications such as education (control systems) and aircraft (slide rule).



Sir William Thomson's third tide-predicting machine design, 1879–81

Digital Computers

Everything that a digital computer does is based on one operation: the ability to determine if a switch, or "gate," is open or closed. That is, the computer can recognize only two states in any of its microscopic circuits: on or off, high voltage or low voltage, or-in the case of numbers-0 or 1. The speed at which the computer performs this simple

act, however, is what makes it a marvel of modern technology. Computer speeds are measured in megaHertz, or millions of cycles per second. A computer with a "clock speed" of 10 mHz-a fairly representative speed for a microcomputer-is capable of executing 10 million discrete operations each second. Business microcomputers can perform 15 to 40 million operations per second, and supercomputers used in research and defense applications attain speeds of billions of cycles per second. Digital computer speed and calculating power are further enhanced by the amount of data handled during each cycle. If a computer checks only one switch at a time, that switch can represent only two commands or numbers; thus ON would symbolize one operation or number, and OFF would symbolize another. By checking groups of switches linked as a unit, however, the computer increases the number of operations it can recognize at each cycle. For example, a computer that checks two switches at one time can represent four numbers (0 to 3) or can execute one of four instructions at each cycle, one for each of the following switch patterns: OFF-OFF (0); OFF-ON (1); ON-OFF (2); or ON-ON (3).

Programs

The defining feature of modern computers which distinguishes them from all other machines is that they can be programmed. That is to say that some type of instructions (the program) can be given to the computer, and it will process them. Modern computers based on the von Neumann architecture often have machine code in the form of an imperative programming language.

In practical terms, a computer program may be just a few instructions or extend to many millions of instructions, as do the programs for word processors and web browsers for example. A typical modern computer can execute billions of instructions per second (gigaflops) and rarely makes a mistake over many years of operation. Large computer programs consisting of several million instructions may take teams of programmers years to write, and due to the complexity of the task almost certainly contain errors.

Basic Components of Stored-Program Computers

By stored-program computer, we mean a machine in which the program, as well as the data, are stored in memory, each word of which can be accessed in uniform time. Most of the high-level language programming the reader has done will likely have used this kind of computer implicitly. However, the program that is stored is not high-level language text. If it were, then it would be necessary to constantly parse this text, which would slow down execution immensely. Instead one of two other forms of storage is used: An abstract syntax representation of the program could be stored.

The identifiers in this representation are pre-translated, and the structure is traversed dynamically as needed during execution. This is the **approach** used by an **interpreter for the language. A second approach** is to use a **compiler** for the language. The compiler translates the program into the very low-level language native to the machine, appropriately called **machine language**. The native machine language acts as a least-common-denominator language for the computer. A machine that had to understand, at a native level, many different languages would be prohibitively complex and slow. **Machine language** is rarely programmed directly, since that would involve manipulating bits, with which it is easy to err (أخطأ). Instead, an equivalent symbolic form known as **assembly language** is employed.

Machine code

In most computers, individual instructions are stored as machine code with each instruction being given a unique number (its operation code or opcode for short). The command to add two numbers together would have one opcode; the command to multiply them would have a different opcode, and so on. The simplest computers are able to perform any of a handful of different instructions; the more

complex computers have several hundred to choose from, each with a unique numerical code.

Since the computer's memory is able to store numbers, it can also store the instruction codes. This leads to the important fact that entire programs (which are just lists of these instructions) can be represented as lists of numbers and can themselves be manipulated inside the computer in the same way as numeric data. The fundamental concept of storing programs in the computer's memory alongside the data they operate on is the crux of the von Neumann, or stored program, architecture.

Instead, each basic instruction can be given a short name that is indicative of its function and easy to remember – a mnemonic such as ADD, SUB, MULT or JUMP. These mnemonics are collectively known as a computer's assembly language. Converting programs written in assembly language into something the computer can actually understand (machine language) is usually done by a computer program called an assembler.

Programming language

- **Low Level Language**
- **High Level Language**

Low-level languages

Machine languages and the assembly languages that represent them (collectively termed *low-level programming languages*) tend to be unique to a particular type of computer. For instance, an ARM architecture computer (such as may be found in a PDA or a hand-held videogame) cannot understand the machine language of an Intel Pentium or the AMD Athlon 64 computer that might be in a PC.

High-level languages/Third Generation Language

Though considerably easier than in machine language, writing long programs in assembly language is often difficult and is also error prone. Therefore, most practical programs are written in more abstract high-level programming languages that are able to express the needs of the programmer more conveniently (and thereby help reduce programmer error).

High level languages are usually "compiled" into machine language (or sometimes into assembly language and then into machine language) using another computer program called a compiler. High level languages are less related to the workings of the target computer than assembly language, and more related to the language and structure of the problem(s) to be solved by the final program. It is therefore often possible to use different compilers to translate the same high level language program into the machine language of many different types of computer. This is part of the means by which software like video games may be made available for different computer architectures such as personal computers and various video game consoles.

Components of Computer

A general purpose computer has four main components:

- ✓ **arithmetic logic unit (ALU)**
- ✓ **control unit**
- ✓ **memory, and**
- ✓ **input and output devices (collectively termed I/O).**

These parts are interconnected by buses, often made of groups of wires.
Inside each of these parts are thousands to trillions of small electrical circuits which can be turned off or on by means of an electronic switch. Each circuit represents a bit (binary digit) of information so that when the circuit is on it represents a "1", and when off it represents a "0" (in positive logic representation).

Control unit

The control unit (often called a control system or central controller) manages the computer's various components; it reads and interprets (decodes) the program instructions, transforming them into control signals that activate other parts of the computer. Control systems in advanced computers may change the order of execution of some instructions to improve performance.

A key component common to all CPUs is the program counter, a special memory cell (a register) that keeps track of which location in memory the next instruction is to be read from.

The control system's function is as follows—note that this is a simplified description, and some of these steps may be performed concurrently or in a different order depending on the type of CPU:

1. Read the code for the next instruction from the cell indicated by the program counter.
2. Decode the numerical code for the instruction into a set of commands or signals for each of the other systems.
3. Increment the program counter so it points to the next instruction.
4. Read whatever data the instruction requires from cells in memory (or perhaps from an input device). The location of this required data is typically stored within the instruction code.
5. Provide the necessary data to an ALU or register.
6. If the instruction requires an ALU or specialized hardware to complete, instruct the hardware to perform the requested operation.
7. Write the result from the ALU back to a memory location or to a register or perhaps an output device.
8. Jump back to step (1).

Central Processing unit (CPU)

The control unit, ALU, and registers are collectively known as a central processing unit (CPU). Early CPUs were composed of many separate components but since the mid-1970s CPUs have typically been constructed on a single integrated circuit called a *microprocessor*.

Arithmetic logic unit (ALU)

The ALU is capable of performing two classes of operations: arithmetic and logic.

The set of **arithmetic operations** that a particular ALU supports may be limited to addition and subtraction, or might include multiplication, division, trigonometry functions such as sine, cosine, etc., and square roots. Some can only operate on whole numbers (integers) whilst others use floating point to represent real numbers, albeit with limited precision.

Logic operations involve Boolean logic: AND, OR, XOR, and NOT. These can be useful for creating complicated conditional statements and processing boolean logic.

Memory

A computer's memory can be viewed as a list of cells into which numbers can be placed or read. Each cell has a numbered "address" and can store a single number.

The CPU contains a special set of memory cells called registers that can be read and written to much more rapidly than the main memory area. There are typically between two and one hundred registers depending on the type of CPU. Registers are used for the most frequently needed data items to avoid having to access main memory every time data is needed.

Computer main memory comes in two principal varieties:

- random-access memory or RAM
- read-only memory or ROM

RAM can be read and written to anytime the CPU commands it, but ROM is preloaded with data and software that never changes, therefore the CPU can only read from it.

ROM is typically used to store the computer's initial start-up instructions. In general, the contents of RAM are erased when the power to the computer is turned off, but ROM retains its data indefinitely. In a PC, the ROM contains a specialized program called the BIOS that orchestrates loading the computer's operating system from the hard disk drive into RAM whenever the computer is turned on or reset. In embedded computers, which frequently do not have disk drives, all of the required software may be stored in ROM.

Software stored in ROM is often called firmware(البرنامج الدائم), because it is notionally more like hardware than software. Flash memory blurs the distinction between ROM and RAM, as it retains its data when turned off but is also rewritable. It is typically much slower than conventional ROM and RAM however, so its use is restricted to applications where high speed is unnecessary.

In more sophisticated computers there may be one or more RAM cache memories, which are slower than registers but faster than main memory. Generally computers with this sort of cache are designed to move frequently needed data into the cache automatically, often without the need for any intervention on the programmer's part.

- **Input/output (I/O)**

I/O is the means by which a computer exchanges information with the outside world. Devices that provide input or output to the computer are called peripherals. On a typical personal computer, peripherals include input devices like the keyboard

and mouse, and output devices such as the display and printer. Hard disk drives, floppy disk drives and optical disc drives serve as both input and output devices. Computer networking is another form of I/O. I/O devices are often complex computers in their own right, with their own CPU and memory. A graphics processing unit might contain fifty or more tiny computers that perform the calculations necessary to display 3D graphics. Modern desktop computers contain many smaller computers that assist the main CPU in performing I/O.

Memory System Design

Basic concepts

In this lecture, we introduce a number of fundamental concepts that relate to the memory hierarchy of a computer.

Memory Hierarchy

A typical memory hierarchy starts with a small, expensive, and relatively fast unit, called the **cache**, followed by a larger, less expensive, and relatively slow **main memory** unit. Cache and main memory are built using solid-state semiconductor material (typically CMOS transistors). It is customary to call the fast memory level the **primary memory**. The solid-state memory is followed by larger, less expensive, and far slower magnetic memories that consist typically of the (hard) disk and the tape. It is customary to call the disk the secondary memory, while the tape is conventionally called the tertiary memory. The objective behind designing a memory hierarchy is to have a memory system that performs as if it consists entirely of the fastest unit and whose cost is dominated by the cost of the slowest unit.

The memory hierarchy can be characterized by a number of parameters. Among these parameters are the access type, capacity, cycle time, latency, bandwidth, and cost. The term access refers to the action that physically takes place during a read or write operation. The capacity of a memory level is usually measured in bytes. The cycle time is defined as the time elapsed from the start of a read operation to the start of a subsequent read. The latency is defined as the time interval between the request for information and the access to the first bit of that information. The bandwidth provides a measure of the number of bits per second that can be accessed. The cost of a memory level is usually specified as dollars per megabytes. Figure below depicts a typical memory hierarchy.

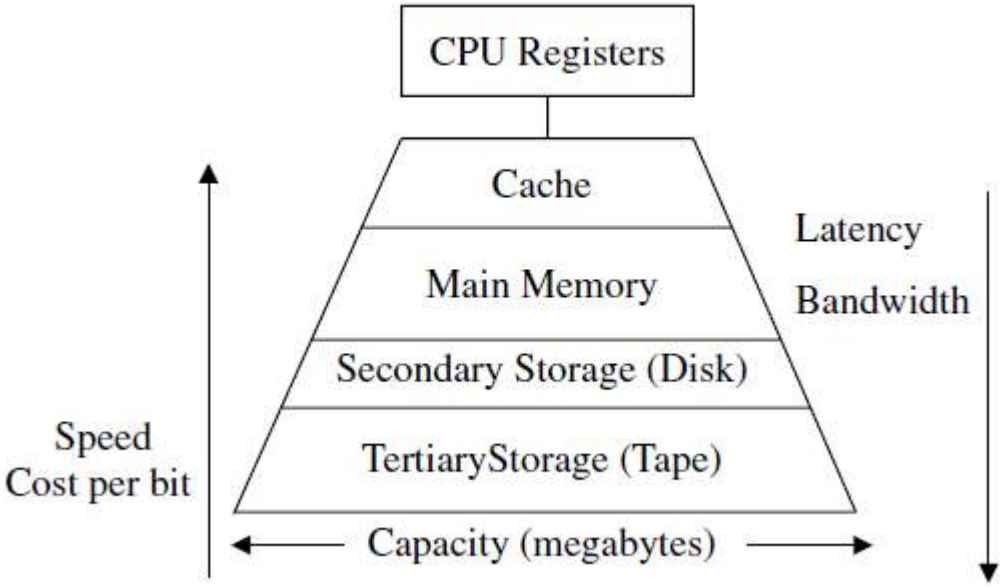


Table below provides typical values of the memory hierarchy parameters.

	Access type	Capacity	Latency	Bandwidth	Cost/MB
CPU registers	Random	64–1024 bytes	1–10 ns	System clock rate	High
Cache memory	Random	8–512 KB	15–20 ns	10–20 MB/s	$500
Main memory	Random	16–512 MB	30–50 ns	1–2 MB/s	$20–50
Disk memory	Direct	1–20 GB	10–30 ms	1–2 MB/s	$0.25
Tape memory	Sequential	1–20 TB	30–10,000 ms	1–2 MB/s	$0.025

The term **random access** refers to the fact that any access to any memory location takes the same fixed amount of time regardless of the actual memory location and/or the sequence of accesses that takes place. **For example**, if a write operation to memory location 100 takes 15 ns and if this operation is followed by a read operation to memory location 3000, then the latter operation will also take 15 ns. This is to be compared to sequential access in which if access to location 100 takes 500 ns, and if a consecutive access to location 101 takes 505 ns, then it is expected that an access to location 300 may take 1500 ns. This is because the

memory has to cycle through locations 100 to 300, with each location requiring 5 ns.

The effectiveness of a memory hierarchy depends on the principle of moving information into the fast memory infrequently and accessing it many times before replacing it with new information. This principle is possible due to a phenomenon called locality of reference; that is, within a given period of time, programs tend to reference a relatively confined area of memory repeatedly. There exist two forms of locality: spatial and temporal locality.

Spatial locality refers to the phenomenon that when a given address has been referenced, it is most likely that addresses near it will be referenced within a short period of time, for example, consecutive instructions in a straight line program. **Temporal locality**, on the other hand, refers to the phenomenon that once a particular memory item has been referenced, it is most likely that it will be referenced next, for example, an instruction in a program loop.

Cash Memory

The idea behind using a cache as the first level of the memory hierarchy is to keep the information expected to be used more frequently by the CPU in the cache (a small high-speed memory that is near the CPU). The end result is that at any given time some active portion of the main memory is duplicated in the cache.

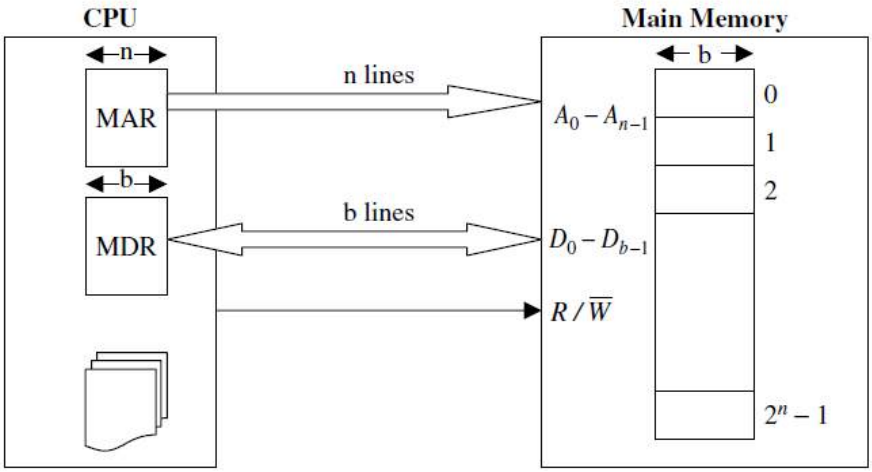
Therefore, when the processor makes a request for a memory reference, the request is first sought in the cache. If the request corresponds to an element that is currently residing in the cache, we call that a **cache hit**. On the other hand, if the request corresponds to an element that is not currently in the cache, we call that a **cache miss**. **A cache hit ratio, hc**, is defined as the probability of finding the requested element in the cache. **A cache miss ratio (1 _ hc)** is defined as the probability of not finding the requested element in the cache.

In the case that the requested element is not found in the cache, then it has to be brought from a subsequent memory level in the memory hierarchy. Assuming that the element exists in the next memory level, that is, the main memory, then it has to be brought and placed in the cache. In expectation that the next requested element will be residing in the neighboring locality of the current requested element (spatial locality), then upon a cache miss what is actually brought to the main memory is a block of elements that contains the requested element.

The advantage of transferring a block from the main memory to the cache will be most visible if it could be possible to transfer such a block using one main memory access time. Such a possibility could be achieved by increasing the rate at which information can be transferred between the main memory and the cache. One possible technique that is used to increase the bandwidth is memory interleaving. To achieve best results, we can assume that the block brought from the main memory to the cache, upon a cache miss, consists of elements that are stored in different memory modules, that is, whereby consecutive memory addresses are stored in successive memory modules.

Main Memory

As the name implies, the main memory provides the main storage for a computer. Figure below shows a typical interface between the main memory and the CPU.



Two CPU registers are used to interface the CPU to the main memory. These are the memory address register (MAR) and the memory data register (MDR). The MDR is used to hold the data to be stored and/or retrieved in/from the memory location whose address is held in the MAR.

It is possible to visualize a typical internal main memory structure as consisting of rows and columns of basic cells. Each cell is capable of storing one bit of information. At any given time, the address decoder activates only one word select line while deactivating the remaining lines. A word select line is used to enable all cells in a row for read or write. Data (bit) lines are used to input or output the contents of cells. Each memory cell is connected to two data lines. A given data line is common to all cells in a given column.

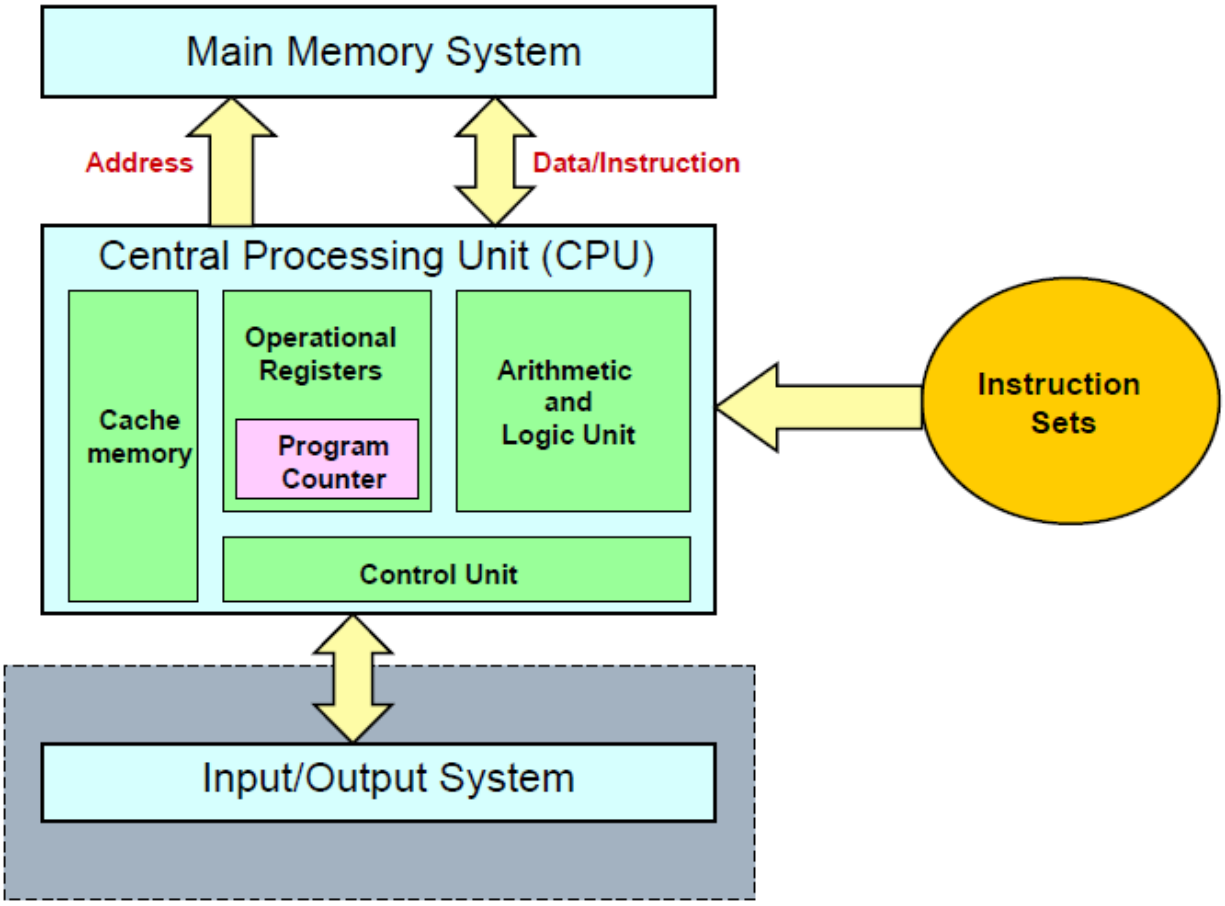
Input / Output Organization

Accessing I/O Devices

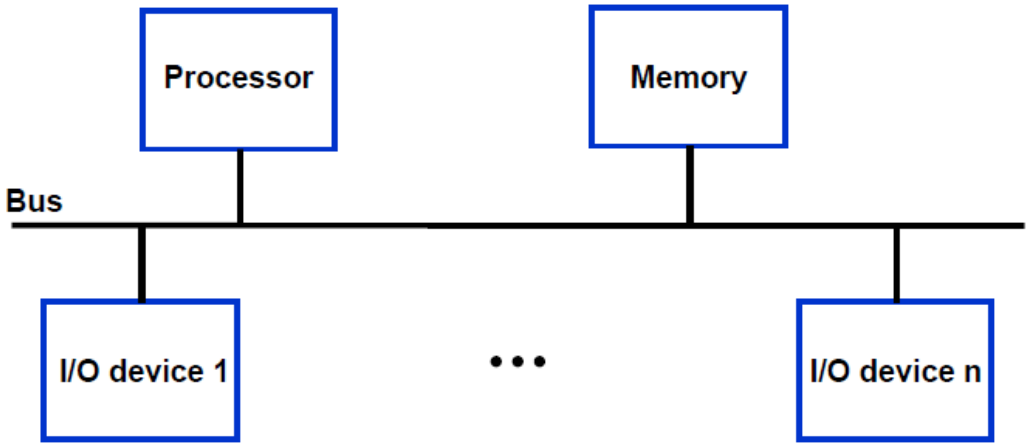
Most modern computers use single bus arrangement for connecting I/O devices to CPU & Memory. The bus enables all the devices connected to it to exchange information.

Bus consists of 3 set of lines : *Address, Data, Control*. Processor places a particular address (unique for an I/O Dev.) on address lines

Device which recognizes this address responds to the commands issued on the Control lines. Processor requests for either Read / Write. The data will be placed on Data lines.



- Single-bus structure
  - The bus enables all the devices connected to it to exchange information
  - Typically, the bus consists of three sets of lines used to carry address, data, and control signals
  - Each I/O device is assigned a unique set of addresses



I/O Mapping

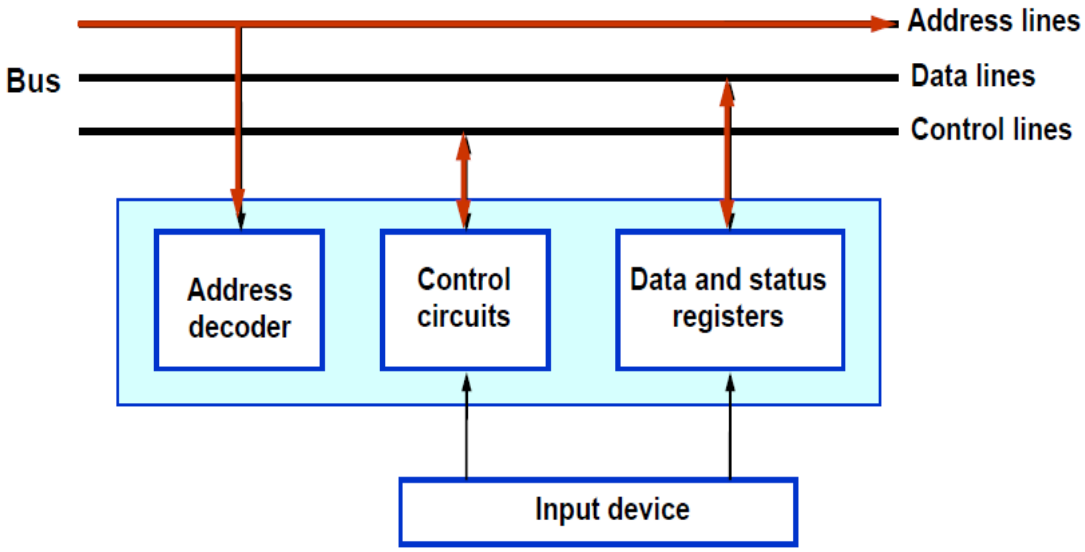
- Memory mapped I/O
  - ✓ Devices and memory share an address space
  - ✓ I/O looks just like memory read/write
  - ✓ No special commands for I/O
- Isolated I/O
  - ✓ Separate address spaces
  - ✓ Need I/O or memory select lines
  - ✓ Special commands for I/O

When I/O devices and the memory share the same address space, the arrangement is called memory mapped I/O. With memory-mapped I/O, any machine instruction that can access memory can be used to transfer data to or from an I/O device Most computer systems use memory-mapped I/O. Some processors have

special IN and OUT instructions to perform I/O transfers When building a computer system based on these processors, the designer has the option of connecting I/O devices to use the special I/O address space or simply incorporating them as part of the memory address space.

I/O Interface for an Input Device

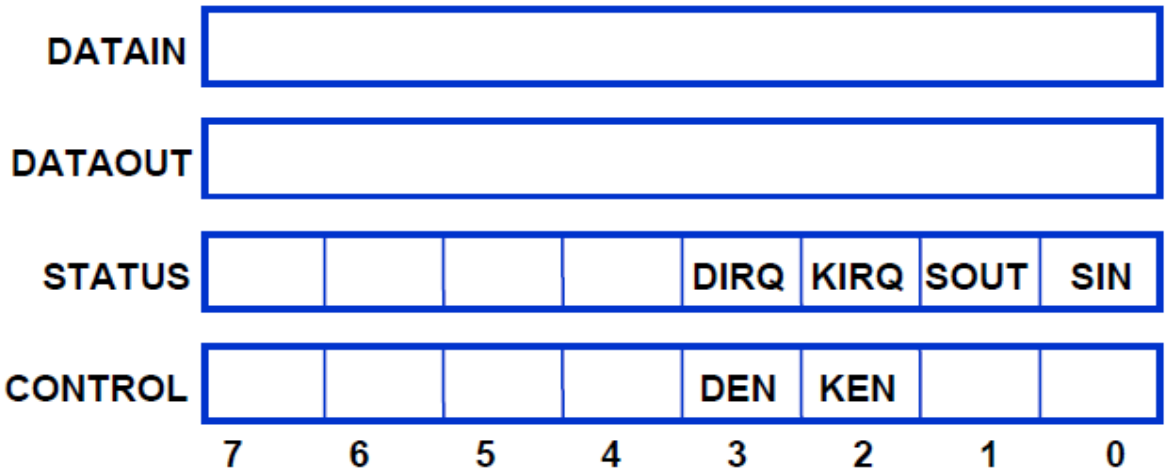
The address decoder, the data and status registers, and the control circuitry required to coordinate I/O transfers constitute the device's *interface circuit.*



Program-Controlled I/O

Consider a simple example of I/O operations involving a keyboard and a display device in a computer system. The four registers shown

below are used in the data transfer operations. The two flags KIRQ and DIRQ in STATUS register are used in conjunction with interrupts.



There are two other commonly used mechanisms for implementing I/O operations: *interrupts* and *direct memory access*

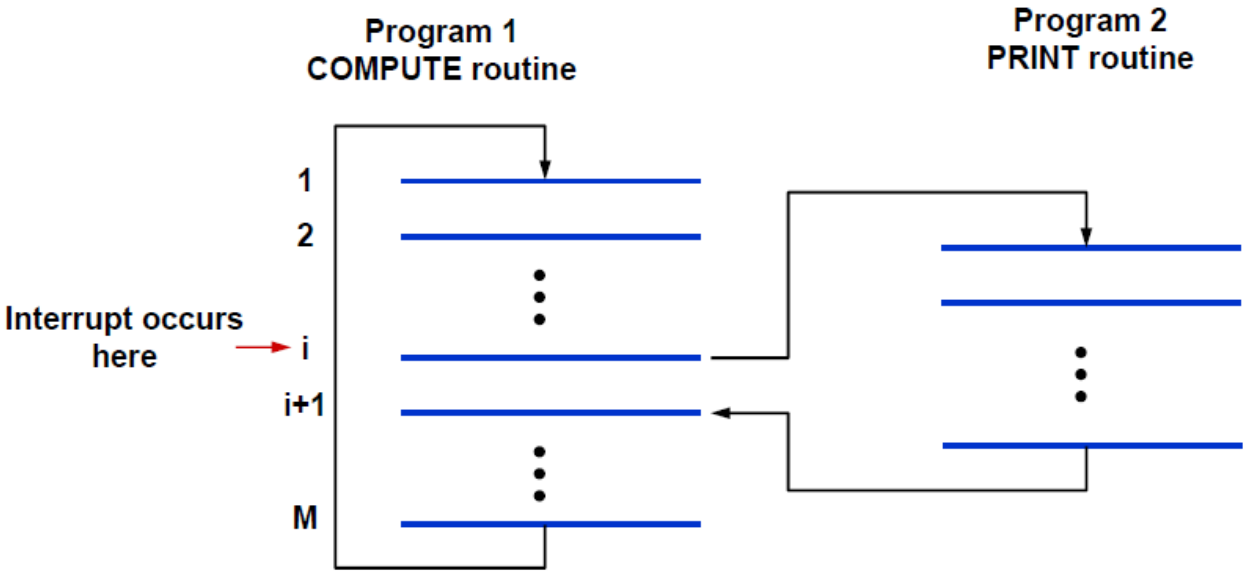
Interrupts: synchronization is achieved by having the I/O device send a special signal over the bus whenever it is ready for a data transfer operation.

Direct memory access: it involves having the device interface transfer data directly to or from the memory.

Interrupts

To avoid the processor being not performing any useful computation, a hardware signal called an *interrupt* to the processor can do it. At least one of the bus control lines, called an *interrupt-request* line, is usually dedicated for this purpose. An *interrupt-service routine* usually is needed and is executed when an interrupt request is issued. On the other hand, the processor must inform the device that its request has been recognized so that it may remove its interrupt-request signal. An *interrupt-acknowledge* signal serves this function.

Example



Interrupt-Service Routine & Subroutine

Treatment of an interrupt-service routine is very similar to that of a subroutine An important departure from the similarity should be noted. A subroutine performs a function required by the program from which it is called.

The interrupt-service routine may not have anything in common with the program being executed at the time the interrupt request is received. In fact, the two programs often belong to different users

Before executing the interrupt-service routine, any information that may be altered during the execution of that routine must be saved. This information must be restored before the interrupted program is resumed.

Interrupt Latency

The information that needs to be saved and restored typically includes the condition code flags and the contents of any registers used by both the interrupted program and the interrupt-service routine.

Saving registers also increases the delay between the time an interrupt request is received and the start of execution of the interrupt-service routine. The delay is called *interrupt latency*.

Typically, the processor saves only the contents of the program counter and the processor status register. Any additional information that needs to be saved must be saved by program instruction at the beginning of the interrupt-service routine and restored at the end of the routine.

Handling Multiple Devices

Handling multiple devices gives rise to a number of questions:

- ✓ How can the processor recognize the device requesting an interrupt?
- ✓ Given that different devices are likely to require different interrupt-service routines, how can the processor obtain the starting address of the appropriate routine in each case?
- ✓ Should a device be allowed to interrupt the processor while another interrupt is being serviced?
- ✓ How should two or more simultaneous interrupt request be handled?

The information needed to determine whether a device is requesting an interrupt is available in its status register. When a device raises an interrupt request, it sets to 1 one of the bits in its status register, which we will call the IRQ bit.

Interrupt Priority

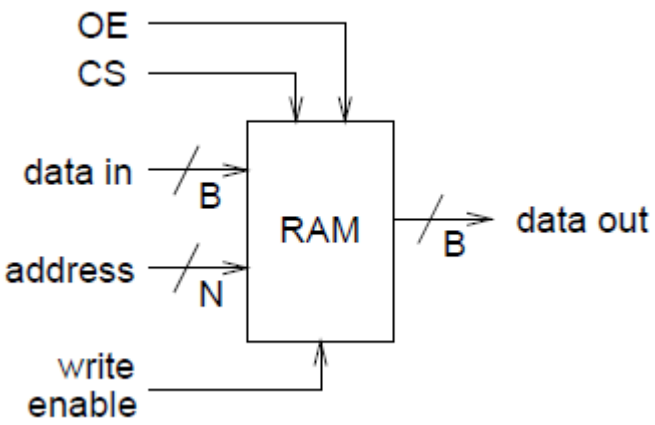
The processor's priority is usually encoded in a few bits of the processor status word. It can be changed by program instructions that write into the program status register (PS). These are privileged instructions, which can be executed only while the processor is running in the supervisor mode.

The processor is in the supervisor mode only when executing operating system routines. It switches to the user mode before beginning to execute application program. An attempt to execute a privileged instruction while in the user mode leads to a special type of interrupt called a privilege exception.

Memory Devices

RAM

A RAM (random-access memory) chip is a memory device that can be written as well as read. A RAM can be described as a sequential circuit in which *N* address inputs select one of 2^N of *B*-bit registers. The following diagram shows the essential pins on a (RAM):



During a *write* operation the CPU selects one set of *B* flip-flops by putting the desired address on the address pins and the data to be latched (stored) into the flip-flops on the data input pins. The write enable pin is used to latch the data into the flip-flops in the same way that a clock is used in a D flip-flop.

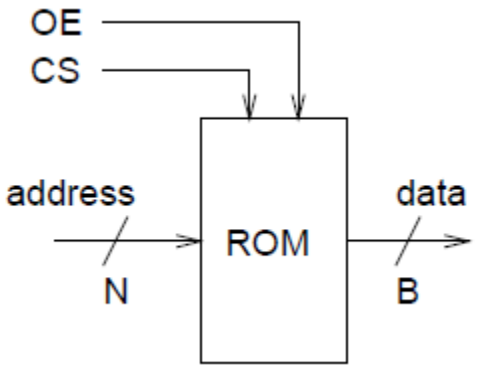
During a *read* operation a particular set of flip-flops is again selected by the address pins and the data previously stored in the flip-flops appears on the data output pins. In many cases a bidirectional data bus is used for both data input and data output.

ROM

The simplest memory IC is a ROM (read-only memory). A ROM can be described as a combinational logic circuit that implements an arbitrary B-bit function of N bits.

The N input bits are known as the address and the B output bits are the data. Such a device is referred to as a 2^N by B memory.

The following diagram shows the input and output pins on a typical read-only memory (ROM):



The address inputs are typically labeled A_0 to A_{N-1} and the data outputs D_0 to D_{B-1}. The CS (*chip select*) and OE (*output enable*) pins must be active for data to appear on the output.

Advantages of ROM

- ✓ Non-volatile in nature
- ✓ These cannot be accidentally changed
- ✓ Cheaper than RAMs
- ✓ Easy to test
- ✓ More Reliable than RAMs
- ✓ These are static and do not require refreshing
- ✓ Its contents are always known and can be verified

Programmed ROM (PROM)

PROM is read-only memory that can be modified only once by a user. The user buys a blank PROM and enters the desired contents using a PROM programmer.

Inside the PROM chip there are small fuses which are burnt open during programming. It can be programmed only once and is not erasable.

EPROM (Erasable and Programmable Read Only Memory)

The EPROM can be erased by exposing it to ultra-violet light for a duration of up to 40 minutes. Usually, a EPROM eraser achieves this function. During programming an electrical charge is trapped in an insulated gate region. The charge is retained for more than ten years because the charge has no leakage path. For erasing this charge, ultra-violet light is passed through a quartz crystal window(lid). This exposure to ultra-violet light dissipates the charge. During normal use the quartz lid is sealed with a sticker.

DIRECT MEMORY ACCESS (DMA)

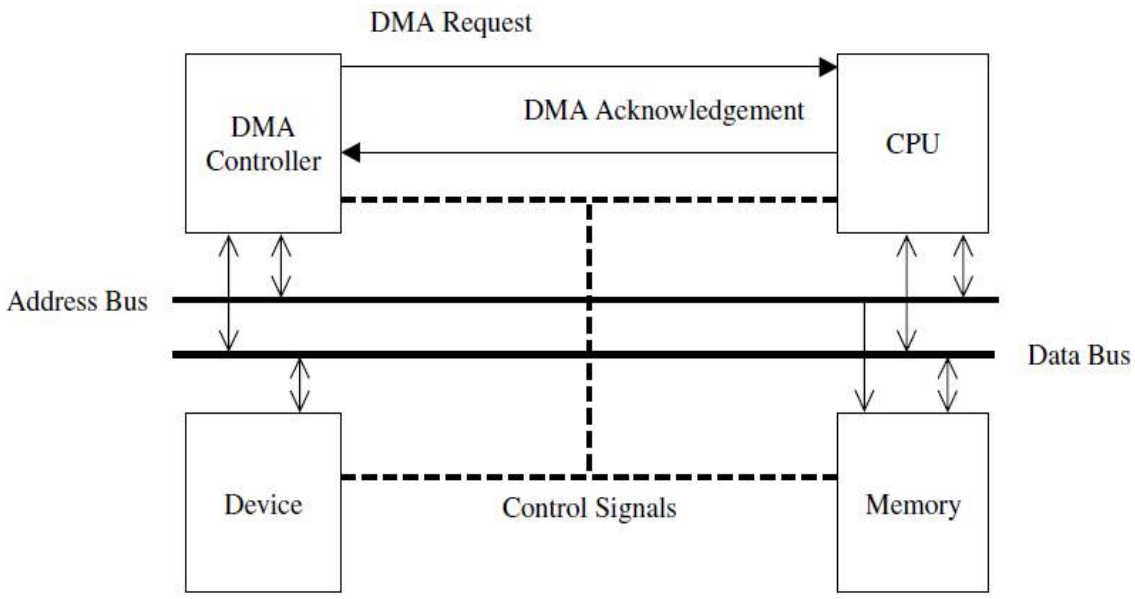
The **main idea** of direct memory access (DMA) is to enable peripheral devices to cut out the "middle man" role of the CPU in data transfer. It allows peripheral devices to transfer data directly from and to memory without the intervention of the CPU. Having peripheral devices access memory directly would allow the CPU to do other work, which would lead to improved performance, especially in the cases of large transfers.

The DMA controller is a piece of hardware that controls one or more peripheral devices. It allows devices to transfer data to or from the system's memory without the help of the processor.

In a typical DMA transfer, some event notifies the DMA controller that data needs to be transferred to or from memory. Both the DMA and CPU use memory bus and only one or the other can use the memory at the same time.

- ✓ The DMA controller then sends a request to the CPU asking its permission to use the bus.
- ✓ The CPU returns an acknowledgment to the DMA controller granting it bus access.
- ✓ The DMA can now take control of the bus to independently conduct memory transfer.

When the transfer is complete the DMA relinquishes its control of the bus to the CPU. Processors that support DMA provide one or more input signals that the bus requester can assert to gain control of the bus and one or more output signals that the CPU asserts to indicate it has relinquished the bus. Figure below shows how the DMA controller shares the CPU's memory bus.



DMA controller shares the CPU's memory bus

The following steps summarize the DMA operations:

1. DMA controller initiates data transfer.
2. Data is moved (increasing the address in memory, and reducing the count of words to be moved).
3. When word count reaches zero, the DMA informs the CPU of the termination by means of an interrupt.
4. The CPU regains access to the memory bus.

A DMA controller may have multiple channels. Each channel has associated with it an address register and a count register. To initiate a data transfer the device driver sets up the DMA channel's address and count registers together with the direction of the data transfer, read or write. While the transfer is taking place, the CPU is free to do other things. When the transfer is complete, the CPU is interrupted.

Direct memory access channels cannot be shared between device drivers. A device driver must be able to determine which DMA channel to use. Some devices have a fixed DMA channel, while others are more flexible, where the device driver can simply pick a free DMA channel to use.

Linux tracks the usage of the DMA channels using a vector of dma_chan data structures (one per DMA channel). The dma_chan data structure contains just two fields, a pointer to a string describing the owner of the DMA channel and a flag indicating if the DMA channel is allocated or not.

8085 Microprocessor Architecture

The 8085 microprocessor is an 8-bit processor available as a 40-pin IC package and uses +5 V for power. It can run at a maximum frequency of 3 MHz. Its data bus width is 8-bit and address bus width is 16-bit, thus it can address 2^{16} = 64 KB of memory.

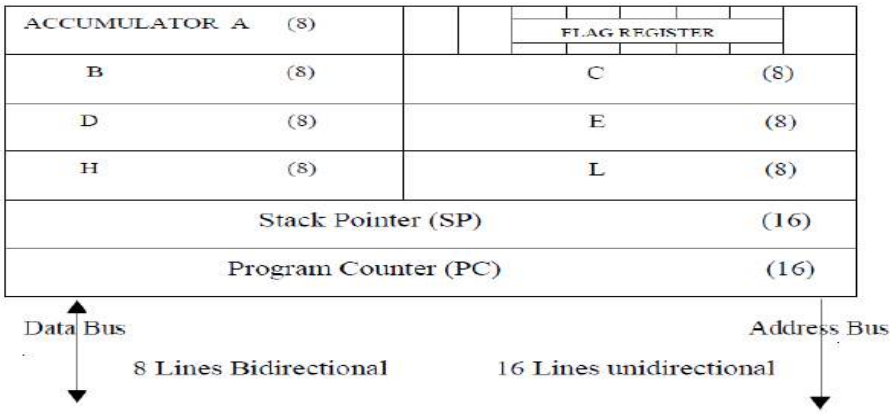
Arithmetic and Logic Unit

The ALU performs the actual numerical and logical operations such as Addition (ADD), Subtraction (SUB), AND, OR etc. It uses data from memory and from Accumulator to perform operations. The results of the arithmetic and logical operations are stored in the accumulator.

Registers

The 8085 includes six registers, one accumulator and one flag register, as shown in Fig. below. In addition, it has two 16-bit registers: stack pointer and program counter. They are briefly described as follows.

The 8085 has six general-purpose registers to store 8-bit data; these are identified as B, C, D, E, H and L. they can be combined as register pairs - BC, DE and HL to perform some 16-bit operations. The programmer can use these registers to store or copy data into the register by using data copy instructions.



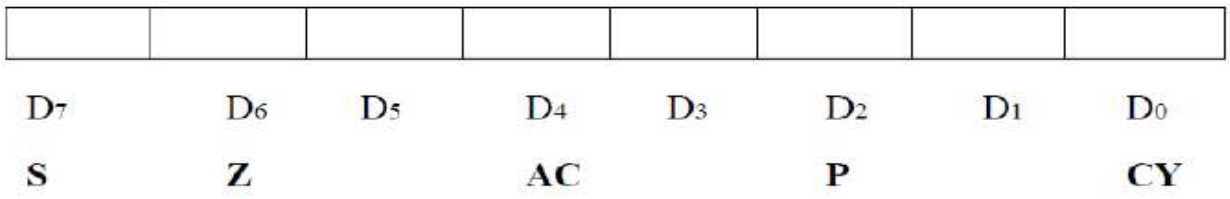
Register organization

Accumulator

The accumulator is an 8-bit register that is a part of ALU. This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A.

Flag register

The ALU includes five flip-flops, which are set or reset after an operation according to data condition of the result in the accumulator and other registers. They are called Zero (Z), Carry (CY), Sign (S), Parity (P) and Auxiliary Carry (AC) flags. Their bit positions in the flag register are shown in Fig. below. The microprocessor uses these flags to test data conditions.



Flag register

For example, after an addition of two numbers, if the result in the accumulator is larger than 8-bit, the flip-flop uses to indicate a carry by setting CY flag to 1. When an arithmetic operation results in zero, Z flag is set to 1. The S flag is just a copy of the bit D7 of the accumulator. A negative number has a 1 in bit D7 and a positive number has a 0 in 2's complement representation. The AC flag is set to 1, when a carry result from bit D3 and passes to bit D4. The P flag is set to 1, when the result in accumulator contains even number of 1s.

Program Counter (PC)

This 16-bit register deals with sequencing the execution of instructions. This register is a memory pointer. The microprocessor uses this register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte is being fetched, the program counter is automatically incremented by one to point to the next memory location.

Stack Pointer (SP)

The stack pointer is also a 16-bit register, used as a memory pointer. It points to a memory location in R/W memory, called stack. The beginning of the stack is defined by loading 16-bit address in the stack pointer.

Instruction Register/Decoder

It is an 8-bit register that temporarily stores the current instruction of a program. Latest instruction sent here from memory prior to execution. Decoder then takes instruction and decodes or interprets the instruction. Decoded instruction then passed to next stage.

Control Unit

Generates signals on data bus, address bus and control bus within microprocessor to carry out the instruction, which has been decoded. Typical buses and their timing are described as follows:

- ***Data Bus*:** Data bus carries data in binary form between microprocessor and other external units such as memory. It is used to transmit data i.e. information, results of arithmetic etc between memory and the microprocessor. Data bus is bidirectional in nature. The data bus width of 8085 microprocessor is 8-bit i.e. 28 combination of binary digits and are typically identified as D0 – D7. Thus size of the data bus determines what arithmetic can be done. If only 8-bit wide then largest number is 11111111

(255 in decimal). Therefore, larger numbers have to be broken down into chunks of 255. This slows microprocessor.

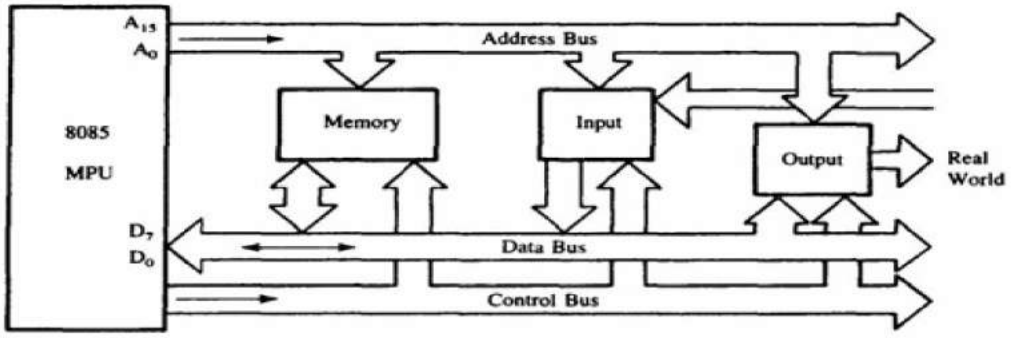
- **Address Bus:** The address bus carries addresses and is one way bus from microprocessor to the memory or other devices. 8085 microprocessor contain 16-bit address bus and are generally identified as A0 - A15. The higher order address lines (A8 – A15) are unidirectional and the lower order lines (A0 – A7) are multiplexed (time-shared) with the eight data bits (D0 – D7) and hence, they are bidirectional.
- **Control Bus:** Control bus are various lines which have specific functions for coordinating and controlling microprocessor operations. The control bus carries control signals partly unidirectional and partly bidirectional.

The schematic representation of the 8085 bus structure is as shown in Fig. below.

The microprocessor performs primarily four operations:

- ✓ Memory Read: Reads data (or instruction) from memory.
- ✓ Memory Write: Writes data (or instruction) into memory.
- ✓ I/O Read: Accepts data from input device.
- ✓ I/O Write: Sends data to output device.

The 8085 processor performs these functions using address bus, data bus and control bus as shown in Fig. below.



The 8085 bus structure

Instruction Set and Execution In 8085

Based on the design of the ALU and decoding unit, the microprocessor manufacturer provides instruction set for every microprocessor. The instruction set consists of both machine code and mnemonics. An instruction is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions that a microprocessor supports is called instruction set. Microprocessor instructions can be classified based on the parameters such functionality, length and operand addressing.

Classification based on functionality:

- ✓ **Data transfer operations:** This group of instructions copies data from source to destination. The content of the source is not altered.
- ✓ **Arithmetic operations:** Instructions of this group perform operations like addition, subtraction, increment & decrement. One of the data used in arithmetic operation is stored in accumulator and the result is also stored in accumulator.
- ✓ **Logical operations:** Logical operations include AND, OR, EXOR, NOT. The operations like AND, OR and EXOR uses two operands, one is stored in accumulator and other can be any register or memory location. The result is stored in accumulator. NOT operation requires single operand, which is stored in accumulator.
- ✓ **Branching operations:** Instructions in this group can be used to transfer program sequence from one memory location to another either conditionally or unconditionally.
- ✓ **Machine control operations:** Instruction in this group control execution of other instructions and control operations like interrupt, halt etc.

Classification based on length:

- ✓ One-byte instructions: Instruction having one byte in machine code. Examples are depicted in Table 1.
- ✓ Two-byte instructions: Instruction having two byte in machine code. Examples are depicted in Table 2
- ✓ Three-byte instructions: Instruction having three byte in machine code. Examples are depicted in Table 3.

Table 1.Examples of one byte instructions

Opcode	Operand	Machine code/Hex code
MOV	A, B	78
ADD	M	86

Table 2 Examples of two byte instructions

Opcode	Operand	Machine code/Hex code	Byte description
MVI	A, 7FH	3E	First byte
		7F	Second byte
ADI	0FH	C6	First byte
		0F	Second byte

Table 3 Examples of three byte instructions

Opcode	Operand	Machine code/Hex code	Byte description
JMP	9050H	C3	First byte
		50	Second byte
		90	Third byte
LDA	8850H	3A	First byte
		50	Second byte
		88	Third byte

Addressing Modes in Instructions:

The process of specifying the data to be operated on by the instruction is called addressing. The various formats for specifying operands are called addressing modes. The 8085 has the following five types of addressing:

- ✓ Immediate addressing
- ✓ Memory direct addressing
- ✓ Register direct addressing
- ✓ Indirect addressing
- ✓ Implicit addressing

- **Immediate Addressing:** In this mode, the operand given in the instruction - a byte or word – transfers to the destination register or memory location.

Ex: MVI A, 9AH

- ✓ The operand is a part of the instruction.
- ✓ The operand is stored in the register mentioned in the instruction.

- **Memory Direct Addressing:** Memory direct addressing moves a byte or word between a memory location and register. The memory location address is given in the instruction.

Ex: LDA 850FH

This instruction is used to load the content of memory address 850FH in the accumulator.

- **Register Direct Addressing:** Register direct addressing transfer a copy of a byte or word from source register to destination register.

Ex: MOV B, C

It copies the content of register C to register B.

- **Indirect Addressing:** Indirect addressing transfers a byte or word between a register and a memory location.

Ex: MOV A, M

Here the data is in the memory location pointed to by the contents of HL pair. The data is moved to the accumulator.

- **Implicit Addressing** In this addressing mode the data itself specifies the data to be operated upon.

Ex: CMA

The instruction complements the content of the accumulator. No specific data or operand is mentioned in the instruction.

Data Communication Principles

Data Communication is one of the most challenging fields today as far as technology development is concerned. **Data, essentially meaning information coded in digital form, that is, 0s and 1s, is needed to be sent from one point to the other either directly or through a network.**

And when many such systems need to share the same information or different information through the same medium, there arises a need for proper organization (rather, "socialization") of the whole network of the systems, so that the whole system works in a cohesive fashion.

Therefore, in order for a proper interaction between the data transmitter (the device needing to commence data communication) and the data receiver (the system which has to receive the data sent by a transmitter) there has to be some set of rules or ("protocols") which all the interested parties must obey.

The requirement above finally paves the way for some ***DATA COMMUNICATION STANDARDS.***

Depending on the requirement of applications, one has to choose the type of communication strategy. **There are basically two major classifications, namely SERIAL and PARALLEL,** each with its variants.

Any data communication standard comprises

- The protocol.
- Signal/data/port specifications for the devices or additional electronic circuitry involved.

What is Serial Communication?

Serial data communication strategies and, standards are used in situations having a limitation of the number of lines that can be spared for communication. This is the primary mode of transfer in long-distance communication.

Serial data communication is the most common ***low-level*** protocol for communicating between two or more devices. Normally, one device is a computer, while the other device can be a modem, a printer, another computer, or a scientific instrument such as an oscilloscope or a function generator.

As the name suggests, the serial port sends and receives bytes of information, rather characters *(used in the other modes of communication)*, in a serial fashion - one bit at a time. These bytes are transmitted using either a ***binary (numerical) format*** or *a* ***text format***.

The most common serial communication system protocols can be studied under the following categories: ***Asynchronous, Synchronous*** *and* ***Bit-Synchronous*** *communication standards.*

Asynchronous Communication and Standards

The Protocol

- This protocol allows bits of information to be transmitted between two devices at an arbitrary point of time.
- The protocol defines that the data, more appropriately a "character" is sent as "frames" which in turn is a collection of bits.
- The start of a frame is identified according to a ***START*** bit(s) and a ***STOP*** bit(s) identifies the end of data frame. Thus, the ***START*** and the ***STOP*** bits are part of the frame being sent or received.
- The protocol assumes that both the transmitter and the receiver are configured in the same way, i.e., follow the same definitions for the start, stop and the actual data bits.
- Both devices, namely, the transmitter and the receiver, need to communicate at an agreed upon data rate ***(baud rate)*** such as **19,200 KB/s or 115,200 KB/s.**

- This protocol has been in use for 15 years and is used to connect PC peripherals such as ***modems*** and the applications include the classic ***Internet dial-up*** modem systems.
- Asynchronous systems allow a number of variations including the number of ***bits in a character*** (5, 6, 7 or 8 bits), the number of ***stops bits*** used (1, 1.5 or 2) and an optional ***parity bit***. Today the most common standard has 8 bit characters, with 1 stop bit and no parity and this is frequently abbreviated as ***'8-1-n'***. A single ***8-bit*** character, therefore, consists of ***10 bits on the line***, i.e., One ***Start*** bit, Eight ***Data*** bits and One ***Stop*** bit (as shown in the figure below).
- Most important observation here is that the individual characters are ***framed*** (unlike all the other standards of serial communication) and ***NO CLOCK*** data is communicated between the two ends.

The Typical Data Format *(known as "FRAME")* for Asynchronous Communication

Interface Specifications for Asynchronous Serial Data Communication

The ***serial port interface*** for connecting two devices is specified by the **TIA** (*Telecommunications Industry Association)* / **EIA-232C** *(Electronic Industries Alliance)*

standard published by the Telecommunications Industry Association; both the physical and electrical characteristics of the interfaces have been detailed in these publications.

RS-232, RS-422, RS-423 and RS-485 are each a recommended standard (RS-XXX) of the Electronic Industry Association (EIA) for asynchronous serial communication and have more recently been rebranded as ***EIA-232, EIA-422, EIA-423 and EIA-485.***

It must be mentioned here that, although, some of the more advanced standards for serial communication like the USB and FIREWIRE are being popularized these days to fill the gap for high-speed, relatively short-run, heavy-data-handling applications, but still, the above four satisfy the needs of all those high-speed and longer run applications found most often in industrial settings for plant-wide security and equipment networking.

RS-232, 423, 422 and 485 specify the **communication system characteristics** of the hardware such as **voltage levels, terminating resistances, cable lengths, etc**. **The standards, however, say nothing about the software protocol or how data is framed, addressed, checked for errors or interpreted.**

THE RS-232

This is the original serial port interface "standard" and it stands for "Recommended Standard Number 232" or more appropriately EIA Recommended Standard 232 is the oldest and the most popular serial communication standard. It was first introduced in 1962 to help ensure connectivity and compatibility across manufacturers for simple serial data communications.

Applications

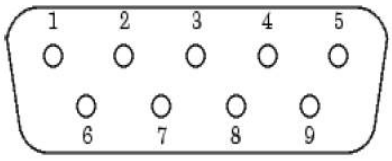
- Peripheral connectivity for PCs (the PC COM port hardware), which can range beyond modems and printers to many different handheld devices and modern scientific instruments.

All the various characteristics and definitions pertaining to this standard can be summarized according to:

- The maximum bit transfer rate capability and cable length.
- Communication Technique: names, electrical characteristics and functions of signals.
- The mechanical connections and pin assignments.

Interfacing of Peripherals Involving the Rs-232 Asynchronous Communication Standards

The RS-232 standard defines the two devices connected with a serial cable as the Data Terminal Equipment (DTE) and Data Circuit-Terminating Equipment (DCE). This terminology reflects the RS-232 origin as a standard for communication between a computer terminal and a modem. Primary communication is accomplished using three pins: the Transmit Data (TD) pin, the Receive Data(RD) pin, and the Ground pin (not shown). Other pins are available for data flow control. The serial port pins and the signal assignments for a typical asynchronous serial communication can be shown in the scheme for a 9-pin male connector (DB9) on the DTE as under:

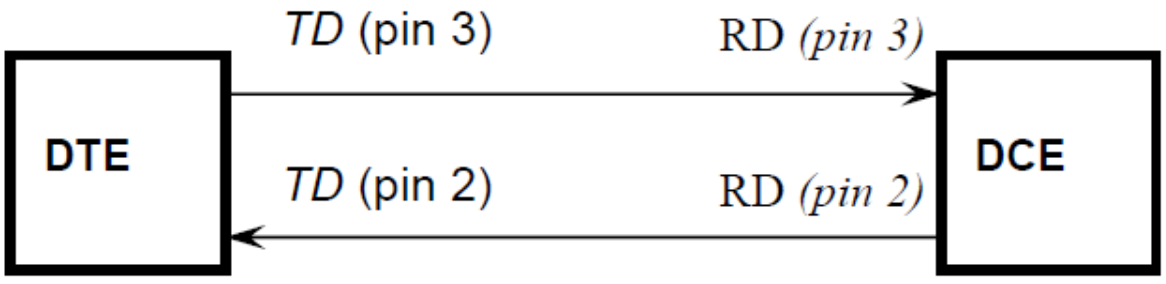


The DB9 male connector

Serial Port Pin and Signal Assignments			
Pin	***Label***	***Signal Name***	***Signal Type***
1	CD	*Carrier Detect*	Control
2	RD	*Received Data*	Data
3	TD	*Transmitted Data*	Data
4	DTR	*Data Terminal Ready*	Control
5	GND	*Signal Ground*	Ground
6	DSR	*Data Set Ready*	Control
7	RTS	*Request to Send*	Control
8	CTS	*Clear to Send*	Control
9	RI	*Ring Indicator*	Control

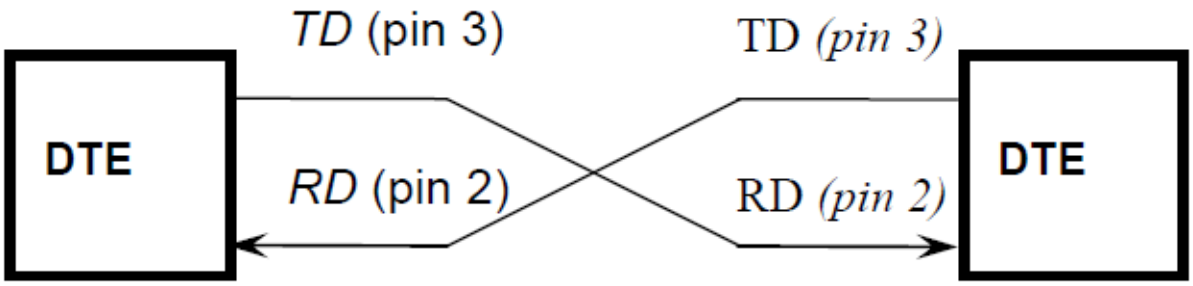
(The RS-232 standard can be referred for a description of the signals and pin assignments used for a 25-pin connector)

Because RS-232 mainly involves connecting a DTE to a DCE, the pin assignments are defined such that ***straight-through cabling*** is used, where pin 1 is connected to pin 1, pin 2 is connected to pin 2, and so on. A DTE to DCE serial connection using the ***Transmit Data (TD)*** pin and the ***Receive Data (RD)*** pin is shown below.



Connecting two DTE's or two DCE's using a straight serial cable, means that the TD pin on each device are connected to each other, and the RD pin on each device are connected to each other. Therefore, to connect two like devices, a ***null***

modem cable has to be used. As shown below, null modem cables crosses the transmit and receive lines in the cable.



Serial ports consist of two signal types: data signals and control signals. To support these signal types, as well as the signal ground, the RS-232 standard defines a 25-pin connection. However, most PC's and UNIX platforms use a 9-pin connection. In fact, only three pins are required for serial port communications: one for receiving data, one for transmitting data, and one for the signal ground.

Throughout this discussion computer is considered a DTE, while peripheral devices such as modems and printers are considered DCE's. Note that many scientific instruments function as DTE's.

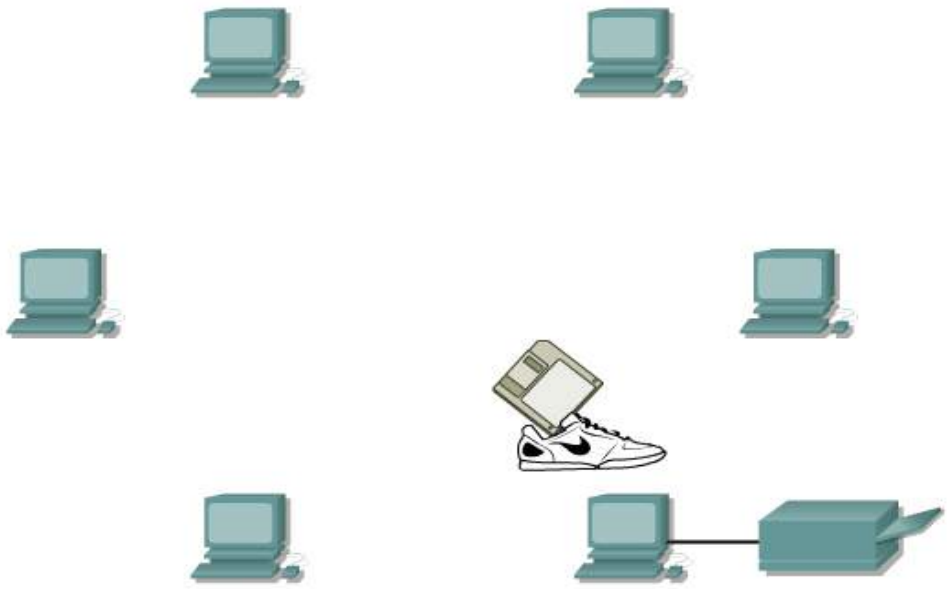
The term "data set" is synonymous with "modem" or "device," while the term "data terminal" is synonymous with "computer."

Network terminology

Data networks

Data networks developed as a result of business applications that were written for microcomputers. The microcomputers were not connected so there was no efficient way to share data among them. It was not efficient or cost-effective for businesses to use floppy disks to share data. Sneaker net created multiple copies of the data. Each time a file was modified it would have to be shared again with all other people who needed that file. If two people modified the file and then tried to share it, one of the sets of changes would be lost. Businesses needed a solution that would successfully address the following three problems:

- How to avoid duplication of equipment and resources
- How to communicate efficiently
- How to set up and manage a network



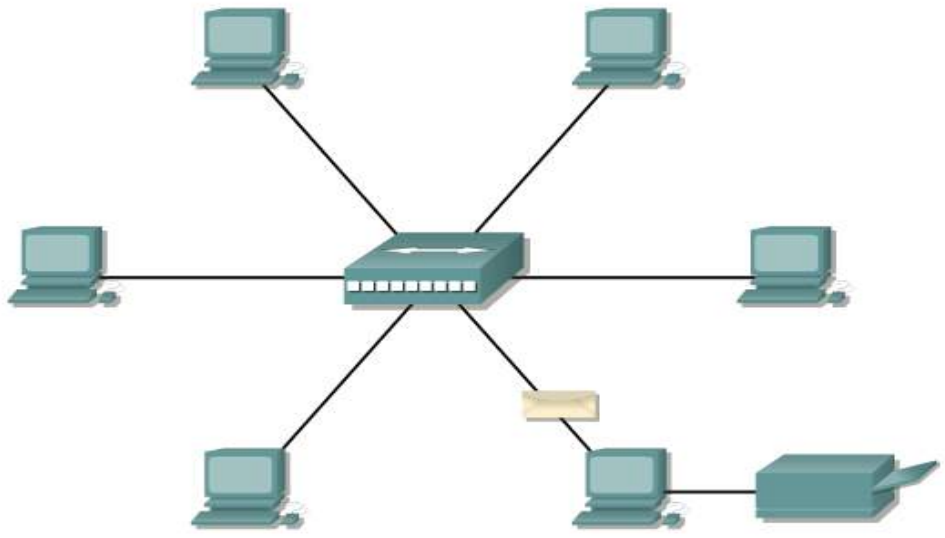
Figure(1) Use floppy disks to share data

Businesses realized that computer networking could increase productivity and save money. Networks were added and expanded almost as rapidly as new network technologies and products were introduced. The early development of

networking was disorganized. However, a tremendous expansion occurred in the early 1980s.

In the mid-1980s, the network technologies that emerged were created with a variety of hardware and software implementations. Each company that created network hardware and software used its own company standards. These individual standards were developed because of competition with other companies. As a result, many of the network technologies were incompatible with each other. It became increasingly difficult for networks that used different specifications to communicate with each other. Network equipment often had to be replaced to implement new technologies.

One early solution was the creation of local-area network (LAN) standards. Figure(2) shows that LAN standards provided an open set of guidelines that companies used to create network hardware and software. As a result, the equipment from different companies became compatible. This allowed for stability in LAN implementations.



Figure(2) Use Network to share data

In a LAN system, each department of the company is a kind of electronic island. As the use of computers in businesses grew, LANs became insufficient.

A new technology was necessary to share information efficiently and quickly within a company and between businesses. The solution was the creation of metropolitan-area networks (MANs) and wide-area networks (WANs) shown in Figure(3). Because WANs could connect user networks over large geographic areas, it was possible for businesses to communicate with each other across great distances. Figure(4) summarizes the relative sizes of LANs and WANs.

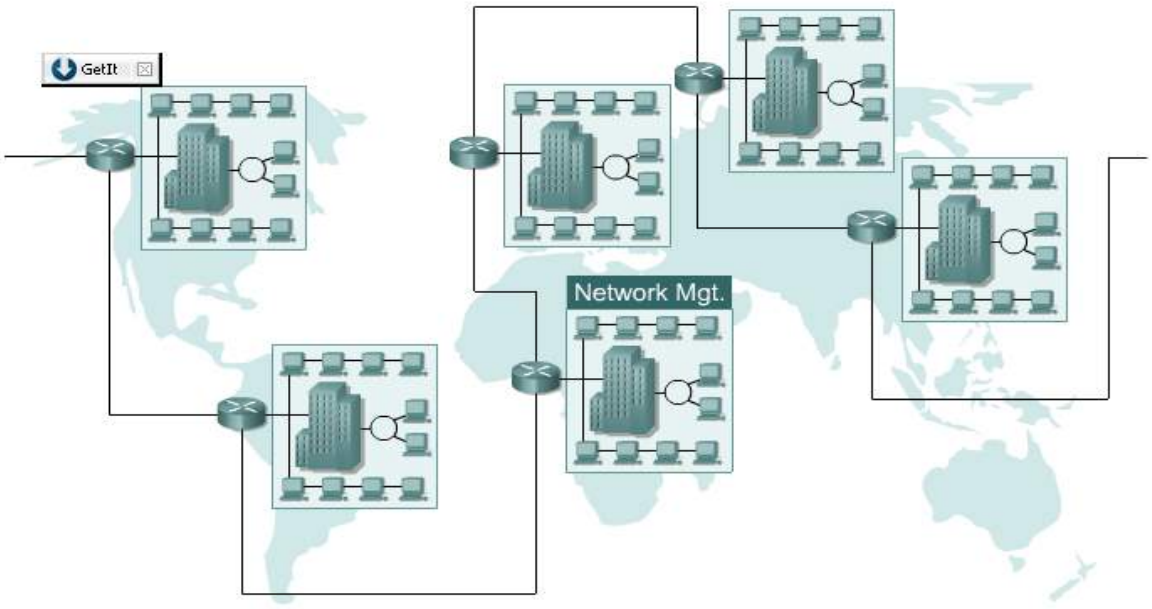


Figure.(3). Wide-Area Networks (WANs)

Distance Between CPUs	Location of CPUs	Name
0.1 m	Printed circuit board Personal data asst.	Motherboard Personal area network (PAN)
1.0 m	Millimeter Mainframe	Computer systems network
10 m	Room	Local area network (LAN) Your classroom
100 m	Building	Local area network (LAN) Your school
1000 m = 1 km	Campus	Local area network (LAN) Stanford University
100,000 m = 100 km	Country	Wide area network (WAN) Cisco Systems, Inc.
1,000,000 m = 1,000 km	Continent	Wide area network (WAN) Africa
10,000,000 m = 10,000 km	Planet	Wide area network (WAN) The Internet
100,000,000 m = 100,000 km	Earth-moon system	Wide area network (WAN) Earth and artificial satellites

Figure(4) The relative sizes of LANs and WANs.

Networking Devices

Equipment that connects directly to a network segment is referred to as a device. These devices are broken up into two classifications. The first classification is end-user devices. End-user devices include computers, printers, scanners, and other devices that provide services directly to the user. The second classification is network devices. Network devices include all the devices that connect the end-user devices together to allow them to communicate.

End-user devices that provide users with a connection to the network are also referred to as hosts shown in Figure(5). These devices allow users to share, create, and obtain information. The host devices can exist without a network, but without the network the host capabilities are greatly reduced. NICs are shown in Figure(6) used to physically connect host devices to the network media. They use this connection to send e-mails, print reports, scan pictures, or access databases.

Figure(5) Host

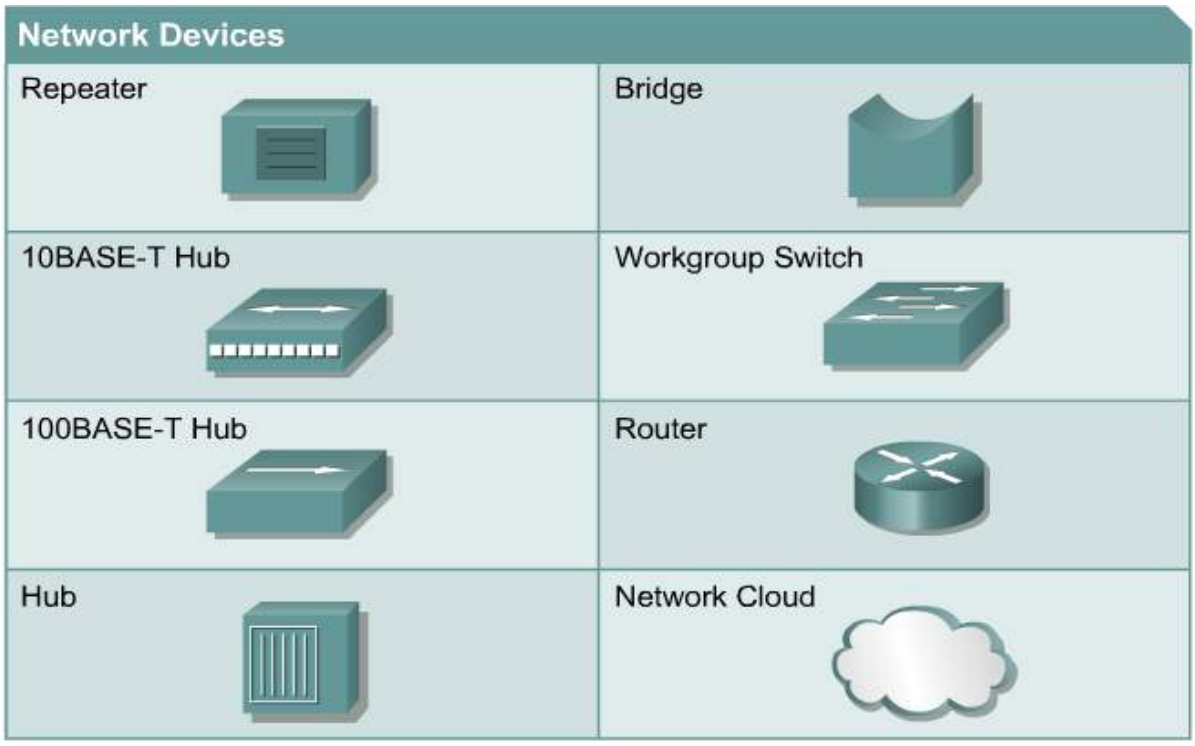
A NIC is a printed circuit board that fits into the expansion slot of a bus on a computer motherboard. It can also be a peripheral device. NICs are sometimes called network adapters. Laptop or notebook computer NICs are usually the size of a PCMCIA card that's shown in Figure(7). Each NIC is identified by a unique code called a Media Access Control (MAC) address. This address is used to control data communication for the host on the network. More about the MAC address will be covered later. As the name implies, the NIC controls host access to the network.

Figure(6) Network Adapter Card (NIC)

Figure(7) Laptop computer NICs (PCMCIA) card

Figure(8) shows network devices are used to extend cable connections, concentrate connections, convert data formats, and manage data transfers. Examples of devices that perform these functions are repeaters, hubs, bridges, switches, and routers. All of the network devices mentioned here are

covered in depth later in the course. For now, a brief overview of networking devices will be provided.



Figure(8) Network devices

A repeater is a network device used to regenerate a signal. Repeaters regenerate analog or digital signals that are distorted by transmission loss due to attenuation. A repeater does not make intelligent decision concerning forwarding packets like a router.

Hubs concentrate connections. In other words, they take a group of hosts and allow the network to see them as a single unit. This is done passively, without any other effect on the data transmission. Active hubs concentrate hosts and also regenerate signals.

Bridges convert network data formats and perform basic data transmission management. Bridges provide connections between LANs. They also check data to determine if it should cross the bridge. This makes each part of the network more efficient.

Workgroup switches add more intelligence to data transfer management. They can determine if data should remain on a LAN and transfer data only to the connection that needs it. Another difference between a bridge and switch is that a switch does not convert data transmission formats.

Routers have all the capabilities listed above. Routers can regenerate signals, concentrate multiple connections, convert data transmission formats, and manage data transfers. They can also connect to a WAN, which allows them to connect LANs that are separated by great distances. None of the other devices can provide this type of connection.

Network Topology

Network topology defines the structure of the network. One part of the topology definition is the physical topology, which is the actual layout of the wire or media. The other part is the logical topology, which defines how the hosts access the media to send data. The physical topologies that are commonly used are as follows:

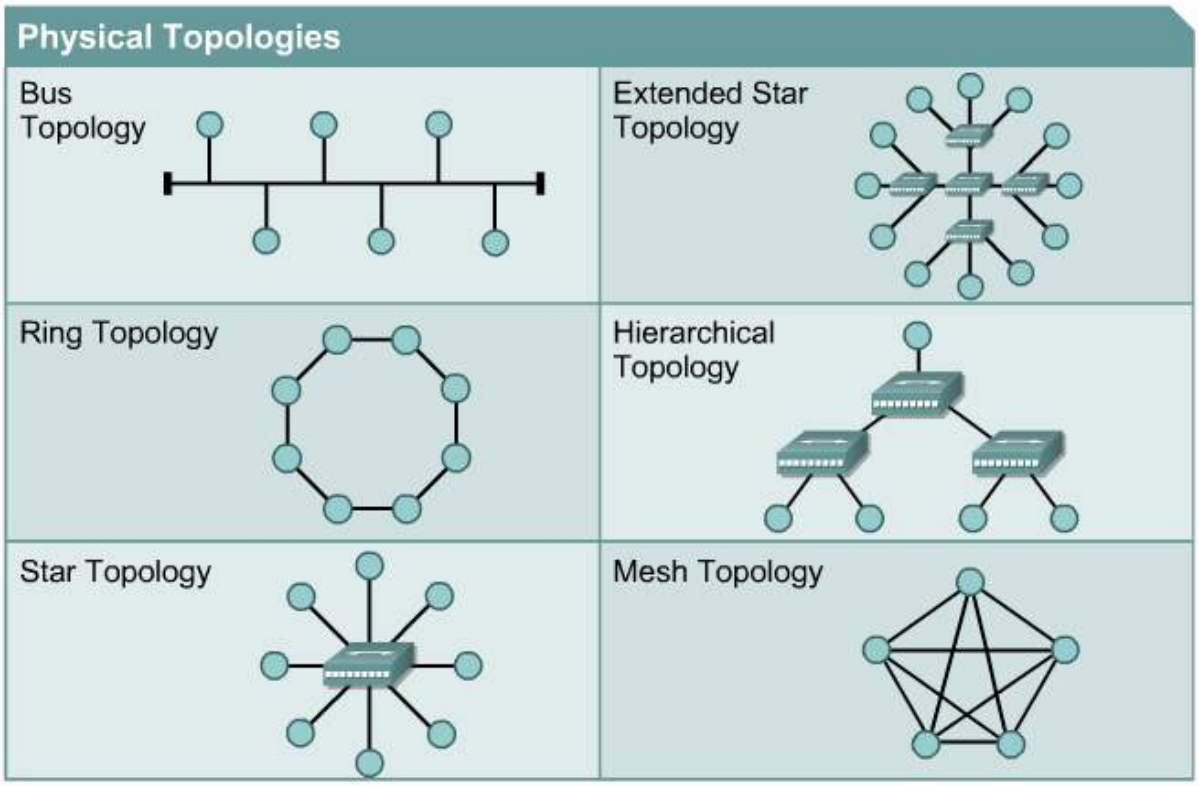
- A bus topology uses a single backbone cable that is terminated at both ends. All the hosts connect directly to this backbone.
- A ring topology connects one host to the next and the last host to the first. This creates a physical ring of cable.
- A star topology connects all cables to a central point.
- An extended star topology links individual stars together by connecting the hubs or switches.
- A hierarchical topology is similar to an extended star. However, instead of linking the hubs or switches together, the system is linked to a computer that controls the traffic on the topology.
- A mesh topology is implemented to provide as much protection as possible from interruption of service. For example, a nuclear power plant might use a

mesh topology in the networked control systems. As seen in the graphic, each host has its own connections to all other hosts. Although the Internet has multiple paths to any one location, it does not adopt the full mesh topology.

The logical topology of a network determines how the hosts communicate across the medium. The two most common types of logical topologies are broadcast and token passing.

The use of a broadcast topology indicates that each host sends its data to all other hosts on the network medium. There is no order that the stations must follow to use the network. It is first come, first serve. Ethernet works this way as will be explained later in the course.

The second logical topology is token passing. In this type of topology, an electronic token is passed sequentially to each host. When a host receives the token, that host can send data on the network. If the host has no data to send, it passes the token to the next host and the process repeats itself. Two examples of networks that use token passing are Token Ring and Fiber Distributed Data Interface (FDDI). A variation of Token Ring and FDDI is Arcnet. Arcnet is token passing on a bus topology.



Figure(9) Physical network topologies

Internet Organization

- How does a network of computers work?
- How is information sent from a source computer and gets to a destination computer?

A good way to understand this is to think in terms of *layers*.

A network is made up of layers, the lowest of which is the **hardware**. All other layers consist of software. Lower layers consist of programs that use the hardware directly. Programs in higher layers invoke the ones in the lower layers, and are easy to use, since they don't require detailed knowledge of how the network operates.

The lowest layer, the hardware, is, of course, necessary. Information can be sent, in bits, on wires and can be created and processed by hardware circuits. Hardware, however, is not completely reliable, and is tedious to use directly. This is where the low software layers come in. Software is necessary to decide how to route the information, and to check and make sure that all the bits of a message have been received correctly.

Computer networks resemble the phone network in a superficial way because of the following:

- They use existing phone lines to send information, and they rent dedicated lines from telephone and telecommunications companies.
- first, information is then moved both ways; finally, the connection is closed.

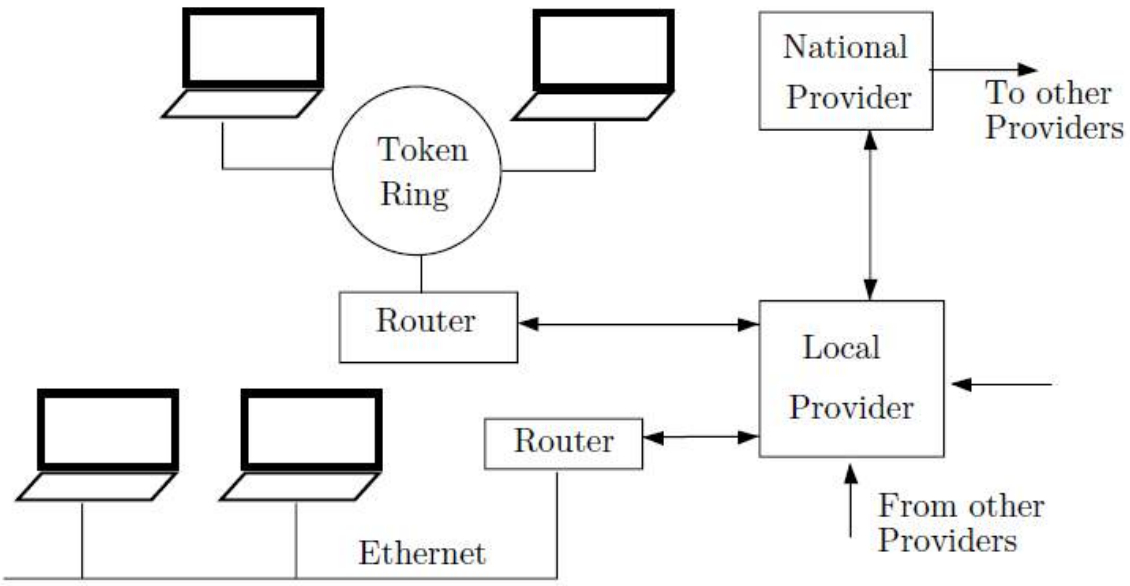
It therefore comes as a surprise to learn that computer networks operate more like the postal service than the phone network.

When *A* calls *B* on the phone, the phone network creates a direct connection between them, and dedicates certain lines to that connection. No one else can use these lines as long as *A* and *B* stay connected. These two users monopolize part of the phone network for a certain duration. On the other hand, when *A* sends a letter to *B*, the

postal service does not dedicate any routes or resources to that letter. It sorts the letter, together with many others, then sends it, by truck or plane, to another post office, where it is sorted again and routed, until the letter arrives at its destination.

We say that the phone system is a ***circuit switched*** **network**, whereas the postal service uses ***packet switching***. Computer networks also use packet switching. A user does not monopolize any part of the network. Rather, any message sent by the user is examined by network software. Depending on its destination address, it is sent to another part of the network where it is examined again, sent on another trip, and so

on. Eventually, it arrives at its destination, or is sent back, if the destination address does not exist.



Internet organization

This structure is illustrated by Figure above. It shows two local networks, a token ring and an Ethernet, connected to a local provider. This may be a computer at a local university or a local computer center.

The connections consist of dedicated lines run between computers called *routers*. A router is a computer performing all the network operations for a local network. A router is normally dedicated to network applications and does not do anything else.

The router at the local network provider is connected to a number of local routers, and to at least one bigger network provider, perhaps a national one. The national network providers are connected with high-speed dedicated *backbone* lines, to complete the network.

Internet: Physical Layout

The key to understanding the physical structure of the Internet is to think of it as a hierarchical structure.

At the bottom of this hierarchy are the internet users (PCs, workstations, mainframes). Each is connected to a local Internet Service Provider (ISP), which is the next higher level in the hierarchy. The local ISPs are in turn connected to regional ISPs (or regional backbone providers), which are in turn connected to national and international backbone providers. These providers are the highest level in the hierarchy and are connected together at interconnect points. New branches and connections can be added to any level.

Internet Protocol (IP)

We continue with the analogy of the post office. In order for a letter to arrive at its destination, it must include the receiver's address at a certain place on the envelope. It should also have the sender's address, in a different place, in case it has to be sent back. Similarly, a message sent on the network must have both the receiver's and sender's addresses, and they must appear at certain places. Since the

addresses are examined by software, they should be numeric, not strings of characters.

The Internet therefore specifies a set of rules that tell users how and where to include addresses in a message. These rules are called the **Internet Protocol,** or IP. A message generated in a computer is sent by it to the router of its local network. The router may be connected to only one other router (at the local provider), in which case it must send all messages to that router. The router at the local provider executes IP programs that examine the address, and decide how to route the message. This process continues, and the message 'hops' from router to router. At a certain point, the message arrives at a national router, which forwards it to a smaller, local provider. From there it is sent to a local network, where the local router finally sends it to the destination computer (or waits for the destination computer to query it, looking for messages).

Transmission Control Protocol (TCP)

Packets sent in a certain order may be sent along different routes, and may arrive out of order. Also, packets may be lost, or may get damaged along the way. Here again the post office is a good analogy. This is why IP is enough in principle, but not in practice. Another protocol, the transmission control protocol (TCP), is used on the Internet to help with packet organization.

TCP software at the sending router breaks a single long message into packets, and adds a serial number and a checksum to each packet. At the receiving router, similar software combines arriving packets according to their numbers, and checks for missing or bad packets.

The combined efforts of IP and TCP create the effect of Internet resources being dedicated to each user when, in reality, all users are treated equally. Note that, in practice, most users don't know anything about IP/TCP, and use software on a

higher layer. A typical program used for Internet communications, such as a web browser, starts with a menu of standard Internet operations, such as Email, Telnet, FTP.

Domain Names

All software used for communications over the Internet uses IP addresses. Human users, however, rarely have to memorize and type such numbers. We normally use text-based addresses like http://www.JohnDoe.com/. Such an address is called a **universal resource locator (URL).** The part **JohnDoe.com is** called the **domain name**.

Here are some reasons for why domain names are important:

1. People find it more convenient to memorize and use text rather than numbers. We find it more natural to use http://www.JohnDoe.com/ instead of an IP number such as 123.234.032.321.

2. IP numbers may change, but internet users prefer to stay with the same domain names.

3. Since an IP address is 32 bits, there can be 2^{32} IP addresses. This is a large number (about 4.3 billion), but the amount of available IP addresses is dwindling fast because of the rapid growth of the Internet.