LECTURE NOTES

ON

COMPUTER ORGANIZATION
(ECE)

III B. Tech V Semester(IARE-R16)

Prepared
by
Mr. N V Krishna Rao ,
Assistant Professor,CSE

Mr. P Anjaiah,
Assistant Professor,CSE

Mr. N Rajasekhar,
Assistant Professor,CSE

Ms. B Vijaya Durga,
Assistant Professor,CSE

ELECTRONICS AND COMMUNICATION ENGINEERING

INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
Dundigal, Hyderabad -500 043

UNIT-I
INTRODUCTION

Computing and computers

Today, almost all of us in the world make use of computers in one way or the other. It finds applications in various fields of engineering, medicine, commercial, research and others. Not only in these sophisticated areas, but also in our daily lives, computers have become indispensable. They are present everywhere, in all the dev ices that we use daily like cars, games, washing machines, microwaves etc. and in day to day computations like banking, reservations, electronic mails, internet and many more.

The word computer is derived from the word compute. Compute means to calculate. The computer was originally defined as a super fast calculator. It had the capacity to solve complex arithmetic and scientific problems at very high speed. But nowadays in addition to handling complex arithmetic computations, computers perform many other tasks like accepting, sorting, selecting, moving, comparing various types of information. They also perform arithmetic and logical operations on alphabetic, numeric and other types of information. This information provided by the user to the computer is data. The information in one form which is presented to the computer is the input information or **input data**.

Information in another form is presented by the computer after performing a process on it. This information is the output information or **output data.**

The set of instructions given to the computer to perform various operations is called as the **computer program.** The process of converting the input data into the required output form with the help of the computer program is called as **data processing.** The computers are therefore also referred to as data processors

Therefore a computer can now be defined as a fast and accurate data processing system that accepts data, performs various operations on the data, has the capability to store the data and produce the results on the basis of detailed step by step instructions given to it. The terms **hardware and software** are almost always used in connection with the computer.

The Hardware:

The hardware is the machinery itself. It is made up of the physical parts or devices of the computer system like the electronic Integrated Circuits (ICs), magnetic storage media and other mechanical devices like input devices, output devices etc. All these various hardware are linked together to form an effective functional unit.The various types of hardware used in the computers, has evolved from vacuum tubes of the first generation to Ultra Large Scale Integrated Circuits of the present generation.

The Software

The computer hardware itself is not capable of doing anything on its own. It has to be given explicit instructions to perform the specific task. The computer program is the one which controls the processing activities of the computer. The computer thus functions according to the instructions written in the program. Software mainly consists of these computer programs, procedures and other documentation used in the operation of a computer system. Software is a collection of programs which utilize and enhance the capability of the hardware

Elements of a computer

Computer is a very effective and efficient machine which performs several activities in few minutes, which otherwise would have taken several days if performed naturally. Besides there would have been a doubt about the accuracy, finish etc. The computer may be faster; more accurate but it cannot compete with human brain. However there are some similarities between the human and the computer which would make the computer more understandable.

Human	Computer
Like human beings has ears, nose, eyes etc.	Computers have input devices such as keyboard, scanner, touch screen, mouse etc to get information.
Like we remember things	Computer also stores information.
We recollect certain information as required.	The computer also retrieves information when times,
We express ourselves by speech, writing etc	Computer expresses through screen, Printouts etc which We call as output.
When we watch, hear, learn certain things and analyze.	with the help of software, computer also can analyze Information and draw conclusions.
The place where we store, analyze,	The computer brain is known as CPU conclude information is known as the brain (Central Processing Unit) where it analyses information.

The computer has storage devices like floppies, hard disks, compact disks to store and retrieve information. However computer does not understand emotions, it does not understand meaning beyond words, it cannot read between the lines like the human. We learn many things unknowingly, certain things knowingly; we call it as upbringing. But computers can learn everything only knowingly. We learn many things on our own, but computer has to be taught to do everything.

The basic parts of computer system are:

- Input Unit
- The Central Processing Unit
- Output Unit

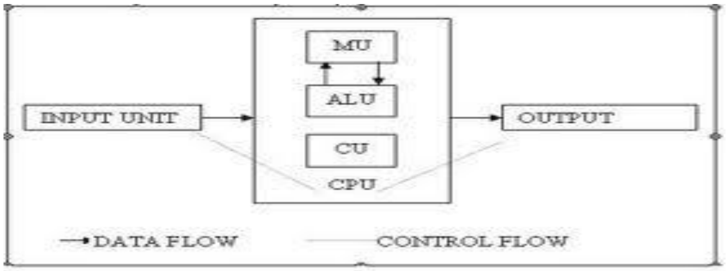


Fig. 1.9 Central Processing Unit.

The Input Unit:

Input devices are the dev ices which are used to feed programs and data to the computer. The input system connects the external environment with the computer system. The input devices are the means of communication between the user and the computer system. Typical input devices include the keyboard, floppy disks, mouse, microphone, light pen, joy stick, magnetic tapes etc. The way in which the data is fed into the computer through each of these devices is different. However, a computer can accept data only in a specific form. Therefore these input devices transform the data fed to them, into a form which can be accepted by the computer. These devices are a means of communication and inter1 station between the user and the computer systems.

Thus the functions of the input unit are :

- accept information (data) and programs.
- convert the data in a form which the computer can accept.
- provide this converted data to the computer for further

processing. The Central Processing Unit:

This is the brain of any computer system. The central processing unit or CPU is made of three parts:

- The control nit.
- The arithmetic logic unit
- Te primary storage unit

The Control Unit:

The Control Unit controls the operations of the entire computer system. The control unit gets the instructions from the programs stored in primary storage unit interprets these instruction an subsequently directs the other units to execute the instructions. Thus it manages and coordinates the entire computer system.

The Arithmetic Logic Unit:

The Arithmetic Logic Unit (ALU) actually executes the instructions and performs all the calculations and decisions. The data is held in the primary storage unit and transferred to the ALU whenever needed. Data can be moved from the primary storage to the arithmetic logic unit a number of times before the entire processing is complete. After the completion, the results are sent to the output storage section and the output devices.

The Primary Storage Unit:

This is also called as Main Memory. Before the actual processing starts the data and the instructions fed to the computer through the input units are stored in this primary storage unit. Similarly, the data which is to be output from the computer system is also temporarily stored in the primary memory. It is also the area where intermediate results of calculations are stored. The main memory has the storage section that holds the computer programs during execution.

Thus the primary unit:

- Stores data and programs during actual processing
- Stores temporary results of intermediate processing
- Stores results of execution temporarily

Output Unit:

The output devices give the results of the process and computations to
the outside world. The output units accept the results produced by the computer, convert them into a human readable form and supply them to the users. The more common output devices are printers, plotters, display screens, magnetic tape drives etc.

Limitations of Computers:

Although the computers of today are highly intelligent and sophisticated they have their own limitations. The computer cannot think on its own, since it does not have its own brain. It can only do what is has been programmed to do. It can execute only those jobs that can be expressed as a finite set of instructions to achieve a specific goal. Each of the steps has to be clearly defined. The computers do not learn from previous experience nor can they arrive at a conclusion without going through all the intermediate steps. However the impact of computers on today’s society in phenomenal and they are today an important part of the society.

HISTORY AND EVALUATION OF COMPUTERS

A Brief History and Evaluation of Computers

History of computers dates back to the 1800s with English mathematician Charles Babbage inventing different machines for automatic calculations. However, history of computing dates back to as ancient as 2,700 BC. While the development and use of **Abacus** around 2700 BC in different world civilizations marked the beginning of computing, innovations such as the **Jacquard Loom** (1805) and **Charles Babbage's “Analytical Engine”** (1834) signified the new age continuation of this development.

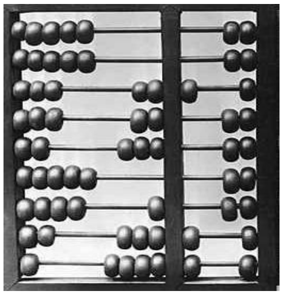
The modern history of computers primarily comprises the development of mechanical, analog and digital computing architectures. During the early days of electronic computing devices, there was much discussion about the relative merits of Analog vs. Digital computers. While Analog Computers use the continuously changeable aspects of physical phenomena such as electrical, mechanical, or hydraulic quantities to model the problem that is being solved, Digital Computers use varying quantities symbolically with their numerical values changing.

As late as the 1960s, mechanical devices, such as the **Merchant Calculator** have widespread application in science and engineering. Until this period, analog computers were routinely used to solve systems of finite difference equations arising. However, in the end, digital computing devices proved to have the power, economics and scalability that were necessary to deal with large scale computations, and found universal acceptance.

Digital computers now dominate the computing world in all areas ranging from the hand calculator to the super computer and are pervasive throughout society.

Therefore, this brief sketch of the development of scientific computing is limited to the area of digital, electronic computers.

The Mechanical Era (1623 - 1945)



Indeed, the history and evolution of computers is quite extraordinary. The history of computers can be traced back to 2700 BC in different civilizations such as Sumerian, Roman and Chinese, which made use of Abacus for mathematical calculations. **Abacus**, a wooden rack holding two horizontal wires with beads strung on them. Numbers are represented using the position of beads on the rack. Fast and simple calculations can be carried out by appropriately placing the beads. In 1620, an English mathematician by the name ***William Oughtred*** invented the slide rule – a calculating device based on the principle of logarithms. It consisted of two graduated scales devised in such a manner that suitable alignment of one scale against the other, made it possible to perform additions, compute products etc. just by inspection.

Blaise Pascal, a French mathematician, is usually credited for building the first digital computer in 1642. He invented the mechanical calculating machine. Numbers were entered in this machine by dialing a series of numbered wheels. Another series of toothed wheels transferred the movements to a dial, which showed the results.

In 1671, ***Gottfried von Leibnitz***, a German mathematician, invented a calculating machine which was able to add and perform multiplications. He invented a special **stepped gear** mechanism for introducing the addend digits, which is still being used.

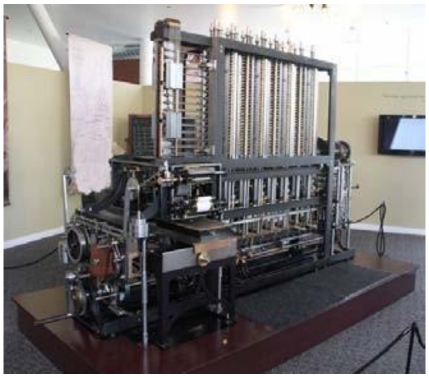
It was only about a century later that ***Thomas of Colmar*** created the first successful mechanical calculator which could add, subtract, multiply, and divide. A lot of improved desktop calculators by various inventors followed, such that by 1890 arrange of improvements like accumulation of partial results, storage of past results, and printing of results were taking place.

The First Computer

Charles Babbage, a professor of mathematics at **Cambridge University**, England, realized that many long calculations usually consisted of a series of actions that were constantly repeated and hence could possibly be automated. By 1822, he designed an automatic calculating machine that he called the '**Difference Engine**'. It was intended to be steam powered and fully automatic (including printing of result tables), commanded by a fixed instruction program. In short, he developed a prototype of a computer which was 100 years ahead of time and is, therefore, considered as the **Father of modern day computers**.

The idea of using machines to solve mathematical problems can be traced at least as far as the early 17th century. Mathematicians who designed and implemented calculators that were capable of addition, subtraction, multiplication, and division included **Wilhelm Schickard**, **Blaise Pascal** and **Gottfried Leibnitz**.

The first multi-purpose, i.e. ***programmable*** computing device was probably **Charles Babbage's Difference Engine**, which was begun in 1823 but never completed. A more ambitious machine was the **Analytical Engine** was designed in 1842, but unfortunately it also was only partially completed by Babbage. Babbage was truly a man ahead of his time: many historians think the major reason he was unable to complete these projects was the fact that the technology of the day was not reliable enough. The first computers were designed by **Charles Babbage** in the mid-1800s, and are sometimes collectively known as the **Babbage Engines**.



The **Difference Engine** was constructed from designs by Charles Babbage. These early computers were never completed during Babbage's lifetime, but their complete designs were preserved. Eventually, one was built in2002.

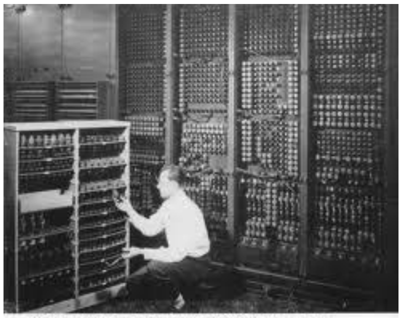
A step towards automated computing was the development of punched cards which were first successfully used by ***Herman Hollerith*** in 1890. He along with ***James Powers*** developed devices that could read information that had been punched into cards, without any human help. This resulted in reduced reading errors, increased workflow and availability of unlimited memory. These advantages were seen by various commercial companies and soon led to the development of improved **punch-card** using computers by companies like **International Business Machines (IBM)** and **Remington**.

Some Well Known First Generation Computers

Mark I

After **World War II** there was a need for advanced calculations. ***Howard A. Aiken*** of **Harvard University**, while working on his doctorate in physics designed a machine that could automatically perform a sequence of arithmetic operations in 1937. He completed this in 1944 and named it **Mark I**. This machine performed a multiplication and division at an average of about four and eleven seconds respectively. The results were printed at a rate of one result per five seconds.

ENIAC



Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

The World War II also produced a large need for computer capacity especially for the military. New weapons were made for which calculating tables and other essential data were needed. In 1942, Professors ***John P. Eckert*** and ***John W. Mauchly*** at the **Moore School of Engineering** of the **University of Pennsylvania**, **USA**, decoded to build a high speed computer to do the job. This was called the **Electronic Numeric Integrator and Calculator (ENIAC)**. It used 18,000 **vacuum tubes**; about 1,800 square feet of floor space, and consumed about 180,000 watts of electrical power. It had punched cards **I/O** and its programs were wired on boards. **ENIAC** is accepted as the first successful high-speed electronic digital computer and was used from 1946 to1955.

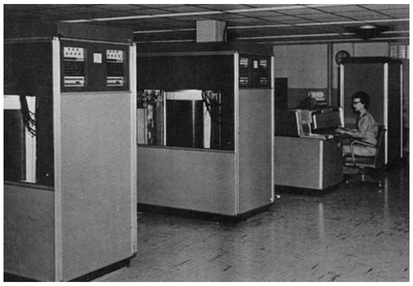
EDVAC

Fascinated by the success of **ENIAC**, ***John Von Neumann***, a mathematician, undertook an abstract study of computation in 1945. In this he aimed to show that a computer should be able to execute any kind of computation by means of a proper programmed control. His ideas, referred to as '**stored program technique**', became essential for future generations of high-speed digital computers and were universally accepted. The basic idea behind the stored program concept was that data as well as instructions can be stored in the computer's memory to enable automatic flow of operations. Between 1947 and 1950, the **More School** personnel and the **Ballistics Research Laboratory** of the **US Army** built a computer named **Electronic Discrete Variable Automatic Computer (EDVAC)**, which was based on **Von Neumann's** concept of stored program.

UNIVAC

The **Universal Automatic Computer (UNIVAC)**, developed in 1951, was the first digital computer to be produced and was installed in the **Census Bureau**. The first generation stored-program computers needed a lot of maintenance. **EDVAC** and **UNIVAC** fell into this group of computers and were the first commercially available computers.

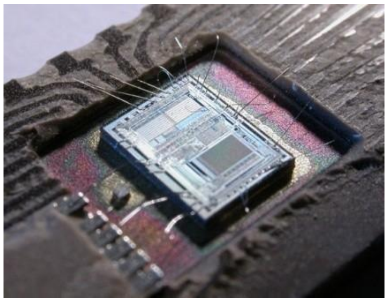
Mid-1950s: Transistor Computers (Second Generation)



The development of **transistors** led to the replacement of vacuum tubes, and resulted in significantly smaller computers. In the beginning, they were less reliable than the **vacuum tubes** they replaced, but they also consumed significantly less power. **IBM 350 RAMAC** used disk drives.

These transistors also led to developments in **computer peripherals**. The first disk drive, the **IBM 350 RAMAC**, was the first of these introduced in 1956.

1960s: The Microchip and the Microprocessor (Third Generation Computers)



The microchip (or integrated circuit) is one of the most important advances in computing technology. Many overlaps in history existed between microchip-based computers and transistor-based computers throughout the 1960s. **Microchips** allowed the manufacturing of smaller computers. The micro chipspurred the production of minicomputers and microcomputers, which were small and inexpensive enough for small businesses and even individuals to own. The microchip also led to the microprocessor, another breakthrough technology that was important in the development of the personal computer.

The first processors were 4-bit, but 8-bit models quickly followed by 1972. 16-bit models were produced in 1973, and 32-bit models soon followed. AT&T Bell Labs created the first fully 32-bit single-chip microprocessor, which used 32-bit buses, 32- bit data paths, and 32-bit addresses, in 1980. The first 64-bit microprocessors were in use in the early 1990s in some markets, though they didn't appear in the PC market until the early2000s.

1970s: Personal Computers (Fourth Generation)

The first personal computers were built in the early 1970s. Most of these were runs, and worked based on small-scale integrated circuits and multi-chip CPUs.

The **Commodore PET** was a personal computer in the 70s. The Altair 8800 was the first popular computer using a single-chip microprocessor. Clones of this machine quickly cropped up, and soon there was an entire market based on the design and architecture of the 8800. It also spawned a club based around hobbyist computer builders, the **Homebrew Computer Club**. 1977 saw the rise of the "**Trinity**" the **Commodore PET**, the **Apple II**, and the **Tandy Corporation's TRS-80**. These three computer models eventually went on to sell millions.

These early PCs had between 4kB and 48kB of **RAM**. The Apple II was the only one with a full-color, graphics-capable display, and eventually became the best-seller among the trinity, with more than 4 million units sold.

1980s-1990s: The Early Notebooks and Laptops

One particularly notable development in the 1980s was the advent of the commercially available portable computer. **Osborne 1** was small and portable enough to transport.

The first of these was the **Osborne 1**, in 1981. It had a tiny 5" monitor and was large and heavy compared to modern laptops (weighing in at 23.5 pounds). Portable computers continued to develop, though, and eventually became streamlined and easily portable, as the notebooks we have today are. These early portable computers were portable only in the most technical sense of the word. Generally, they were anywhere from the size of a large electric typewriter to the size of a suitcase. The Gavilan SC was the first PC to be sold as a "laptop". The first laptop with a flip form factor was produced in 1982, but the first portable computer that was actually marketed as a "laptop" was the Gavilan SC in1983. Early models had monochrome displays, though there were color displays available starting in 1984 (the Commodore SX-64). Laptops grew in popularity as they became smaller and lighter.

2000s: The Rise of Mobile Computing (Present and Beyond) Mobile computing is one of the most recent major milestones in the history of computers. Many smart phones today have higher processor speeds and more memory than desktop PCs had even ten years ago. With phones like the iPhone and the Motorola Droid, it's becoming possible to perform most of the functions once reserved for desktop PCs from anywhere. The Droid is a Smartphone capable of basic computing tasks such as emailing and web browsing.

Mobile computing really had its start in the 1980s with the pocket PCs of the era. These were something like a cross between a calculator, a small home computer and a PDA. During the 1990s, PDAs (Personal Digital Assistant) became popular.

A number of manufacturers had models, including Apple and Palm. The main feature PDAs had that not all pocket PCs had was a touch screen interface. Most basic computing functions can now be done on a Smartphone, such as email, browsing the internet, and uploading photos and videos.

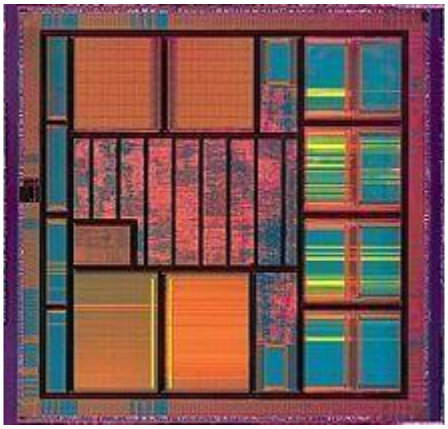
Late 2000s: Notebooks (Artificial Intelligence)

Another recent progression in computing history is the development of notebook computers. Notebooks are smaller and more portable than standard laptops. Some notebooks go as far as to have not only built-in Wi-Fi capabilities, but also built-in mobile broadband connectivity options.

The Asus EEE PC 700 was the first notebook to enter mass production. The first mass-produced notebook was the Asus Eee PC 700, released in 2007. They were originally released in Asia, but were released in the US not long afterward. Other manufacturers quickly followed suit, releasing additional models throughout 2008 and 2009.

VLSI era

Very-large-scale integration (VLSI) is the process of creating an integrated circuit (IC) by combining hundreds of thousands of transistors or devices into a single chip. VLSI began in the 1970s when complex semiconductor and communication technologies were being developed. The microprocessor is a VLSI device. Before the introduction of VLSI technology most ICs had a limited set of functions they could perform. An electronic circuit might consist of a CPU, ROM, RAM and other glue logic. VLSI lets IC designers add all of these into one chip.



VLSI integrated-circuit

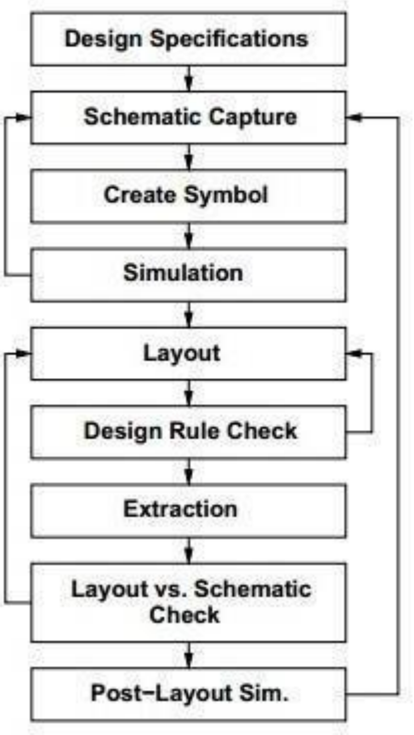
The electronics industry has achieved a phenomenal growth over the last few decades, mainly due to the rapid advances in large scale integration technologies and system design applications. With the advent of very large scale integration (VLSI) designs, the number of applications of integrated circuits (ICs) in high-performance computing, controls, telecommunications, image and video processing, and consumer electronics has been rising at a very fast pace.

The current cutting-edge technologies such as high resolution and low bit-rate video and cellular communications provide the end-users a marvelous amount of applications, processing power and portability. This trend is expected to grow rapidly, with very important implications on VLSI design and systems design.

VLSI Design Flow

The VLSI IC circuits design flow is shown in the figure below. The various levels of design are numbered and the blocks show processes in the design flow.

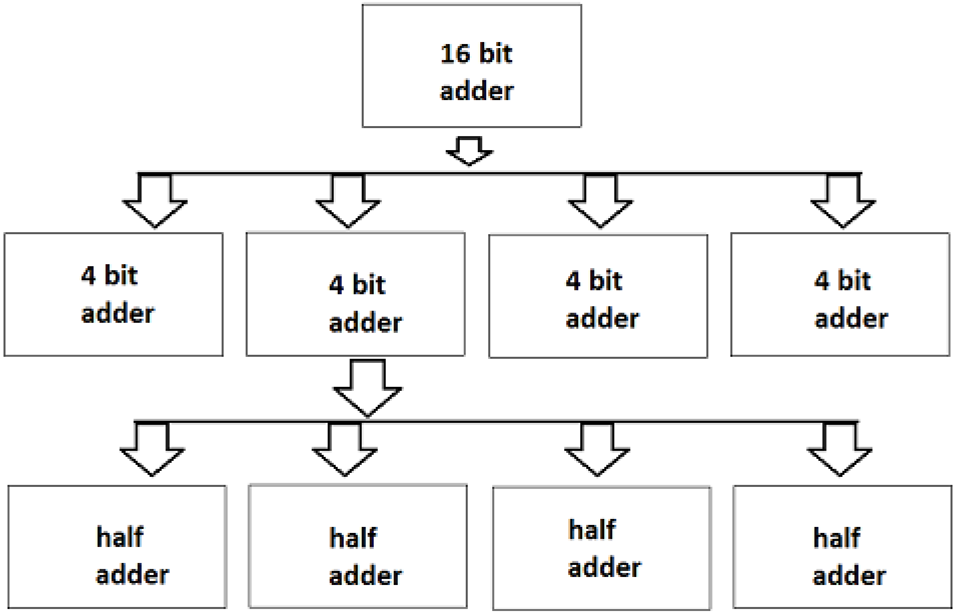
Specifications comes first, they describe abstractly, the functionality, interface, and the architecture of the digital IC circuit to be designed.



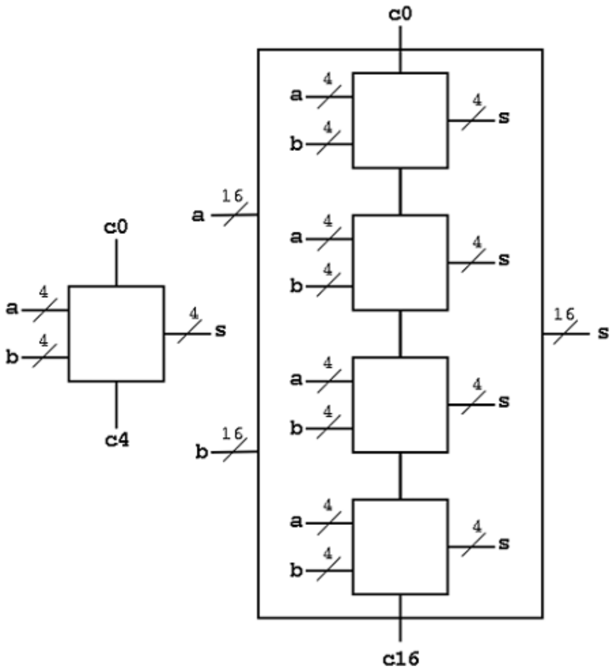
Behavioral description is then created to analyze the design in terms of functionality, performance, compliance to given standards, and other specifications. RTL description is done using HDLs. This RTL description is simulated to test functionality. From here onwards we need the help of EDA tools. RTL description is then converted to a gate-level net list using logic synthesis tools. A gate level net list is a description of the circuit in terms of gates and connections between them, which are made in such a way that they meet the timing, power and area specifications. Finally, a physical layout is made, which will be verified and then sent to fabrication.

Design Hierarchy-Structural

The design hierarchy involves the principle of "Divide and Conquer." It is nothing but dividing the task into smaller tasks until it reaches to its simplest level. This process is most suitable because the last evolution of design has become so simple that its manufacturing becomes easier. We can design the given task into the design flow process's domain (Behavioral, Structural, and Geometrical). To understand this, let’s take an example of designing a 16-bit adder, as shown in the figure below.



Here, the whole chip of 16 bit adder is divided into four modules of 4-bit adders. Further, dividing the 4-bit adder into 1-bit adder or half adder. 1 bit addition is the simplest designing process and its internal circuit is also easy to fabricate on the chip. Now, connecting all the last four adders, we can design a 4-bit adder and moving on, we can design a 16-bit adder.



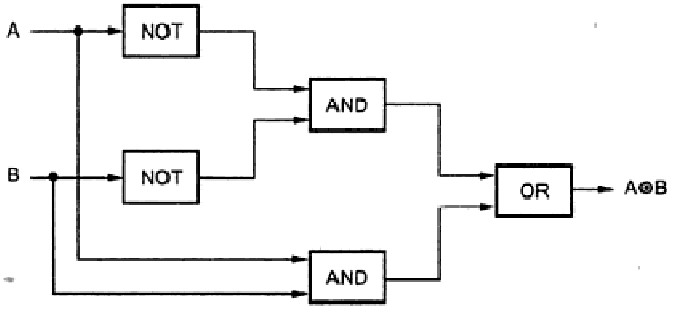
Computer System Design:

A computer is a large and complex system in which system objects are the components of the computer. These components are connected to perform a specific function. The function of such system is determined by the functions of its components and how the components are connected.

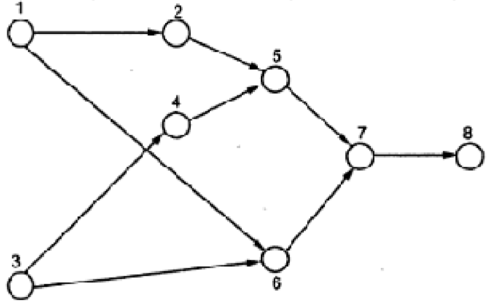
System Representation

We can represent a system using a graph or a block diagram. A computer system is usually represented by a block diagram. A system has its own structure and behavior. The structure and behavior are the two properties of the system. We can define the structure of a system as the abstract graph consisting of its block diagram with no functional information, as shown in Fig. below. As shown in figure, the structure gives the components and their interconnection. A behavioral description, on the other hand, describes the function of each component and thus the function of the system. The behavior of the system may be represented by Boolean function or by truth table in case of logic circuit.

The behavior of logic circuits can also be described by Hardware description language such as VHDL. They can provide precise, technology-independent descriptions of digital circuits at various levels of abstraction, primarily the gate and register levels.



(a) A block diagram representing EX-NOR logic circuit



(b) Structure of a system as an abstract graph

Design Process

For a given system's structure, the task of determining its function or behavior is termed analysis. On the other hand, the problem of determining a system structure that exhibits a given behavior is design or synthesis.

The design process starts with the construction of initial design. In this process, with given a desired range of behavior and set of available components we have to determine a structure (Design).

The next step is to evaluate its cost and performance. The cost and performance should be in the acceptable range. Then we have to confirm that whether the formed structure achieves the desired behavior. If not we have to modify the design to meet the design goals. The Fig. below illustrates the design process.

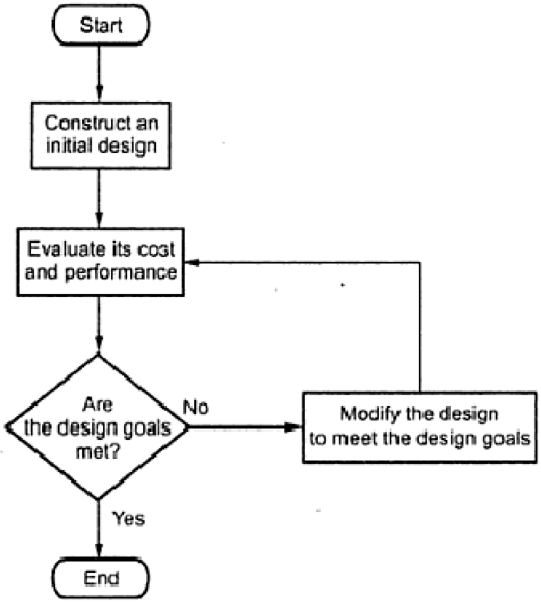

Start
Construct an initial design
Evaluate its cost and performance
Are the design goals met?
No
Modify the design to meet the design goals
Yes
End

Figure Design

Computer-aided Design

The computer-aided design (CAD) tool provides designers with a range of programs to support their design goals. They are used to automate fully or party the more tedious design and evaluate its steps. They contribute mainly in three important ways to the overall design process.

- CAD editors or translators convert design data into forms such as HDL descriptions or schematic diagrams, which can be efficiently processed by the humans, computers or both.
- Simulators create the computer model for the design and can mimic the design's behavior. It helps designer to determine how well the design meets various performance and cost goals.
- Synthesizers derive structures that implement all or part of some design step.

Gate Level Design

Gate level design concerned with processing binary variables: 0 and 1. In this level, the design components are logic gates and flip-flops. Logic gates are memory less elements; however flip-flops are bit storage devices. Using these elements gate level design is used to build the combinational and sequential circuits.

When logic gates are connected together to produce a specified output for certain specified combinations of input variables, with no storage involved, the resulting circuit is called combinational logic. In combinational logic, the output variables are at all times dependent on the combination of input variables.

A combinational circuit consists of input variables, logic gates, and output variables. The logic gates accept signals from the input variables and generate output signals. This process transforms binary information from the given input data to the required output data. Fig. below shows the block diagram of a combinational circuit. As shown in figure, the combinational Circuit accepts n-input binary variables and generates output variables depending on the logical combination of gates.

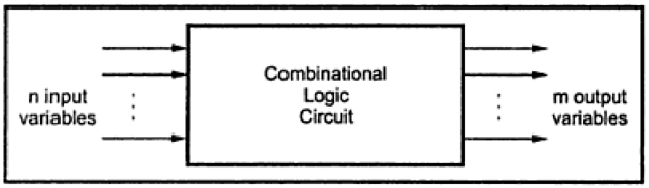


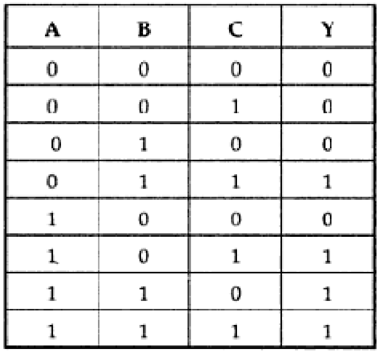
Figure: Block diagram of a combinational circuit

The design of combinational circuits starts from the outline of the problem statement and ends in a logic circuit diagram or a set of Boolean functions from which the logic diagram can be easily obtained. The design procedure of the combinational circuit involves following steps :

1. The problem definition.
2. The determination of number of available input variables and required output variables.
3. Assigning letter symbols to input and output variables.
4. The derivation of truth table indicating the relationships between input and output variables.
5. Obtain simplified Boolean expression for each output.
6. Obtain the logic diagram.

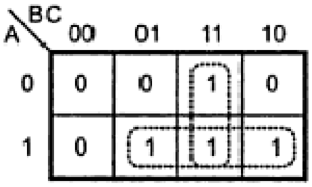
Example: Design a combination logic circuit with three input variables that will produce logic 1 output when more than one input variables are logic 1.

Sol.: Given problem specifies that there are three input variables and one output variable. We assign A, B and C letter symbols to three input variables and assign Y letter symbol to one output variable. The relationship between input variables and output variable can be tabulated as shown in truth table below:



A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

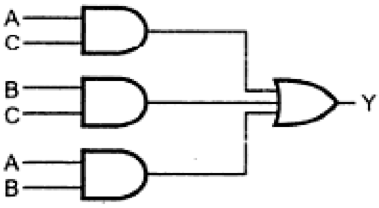
Table : Truth table



A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Now we obtain the simplified Boolean expression for output variable Y using K-map simplification.

Logic Diagram



There are many applications in which digital outputs are required to be generated in accordance with the sequence in which the input signals are received. This requirement cannot be satisfied using a combinational logic system. These applications require outputs to be generated that are not only dependent on the present input conditions but they also depend upon the past history of these inputs. The past history is provided by feedback from the output back to the input.

Fig. below shows the block diagram of sequential circuit. As shown in the figure below, memory elements are connected to the combinational circuit as a feedback path.

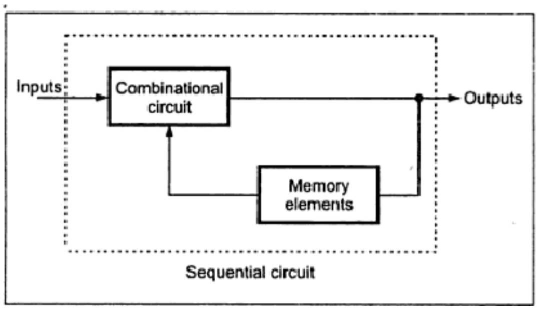


Fig: Block diagram of sequential circuit

The information stored in the memory elements at any given time defines the present state of the sequential circuit. The present state and the external inputs determine the outputs and the next state of the sequential circuit. Thus we can specify the sequential circuit by a time sequence of external inputs, internal states (present states and next states), and outputs.

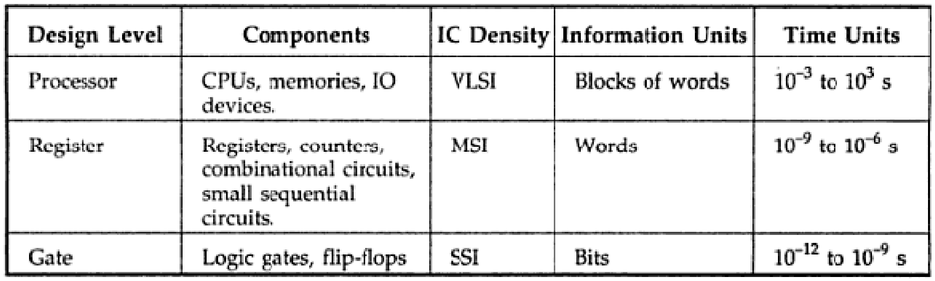
Latches and flip-lops both are bistable elements. These are the basic building blocks of most sequential circuits. The main difference between latches and lip-lops is in the method used for changing their state. We use the name lip-lop for a sequential device that normally samples its inputs and changes its outputs only at times determined by clocking signal. On the other hand, we use the name latch for a sequential device that checks all of its inputs continuously and changes its outputs accordingly at any time independent of a clocking signal. Many times enable signal is provided with the latch. When enable signal is active output changes occur as input changes. But when enable signal is not activated input changes do not affect output. There are various types of flip-lops such as SR flip-flop, D flip-flop, JK flip-flop, T lip-lop and master- slave lip-lop. Each flip-flop has different characteristic equation. Flip-flops and if necessary basic gates are used to design sequential circuits.

Design Levels:

The design of a computer system can be carried at several levels of abstraction. The three such recommended levels are:

- The processor level also called the architecture, behavior, or system level.
- The register level, also called the register-transfer level(RTL).
- The gate level, also called the logic level.

The Table below shows the comparison between these levels.



Design Level	Components	IC Density	Information Units	Time Units
Processor	CPUs, memories, IO devices.	VLSI	Blocks of words	10^{-3} to 10^{3} s
Register	Registers, counters, combinational circuits, small sequential circuits.	MSI	Words	10^{-9} to 10^{-6} s
Gate	Logic gates, flip-flops	SSI	Bits	10^{-12} to 10^{-9} s

Table: Comparison between design levels

To design a complex system usually we have to

- Specify the processor-level structure of the system.
- Specify the register-level structure of each component type identified in step1.
- Specify the gate-level structure of each component type identified in step2.

This design approach is known as top-down design approach and it is extensively used in both hardware and software designs. Well, it is up to the designer to decide whether to design a system using medium scale ICs, small scale ICs or a single IC composed of standard cells. If the system is to be designed using medium-scale ICs or standard cells, then the third step, gate-level design, is no longer needed. In the following section we discuss the register level and processor level design approaches.

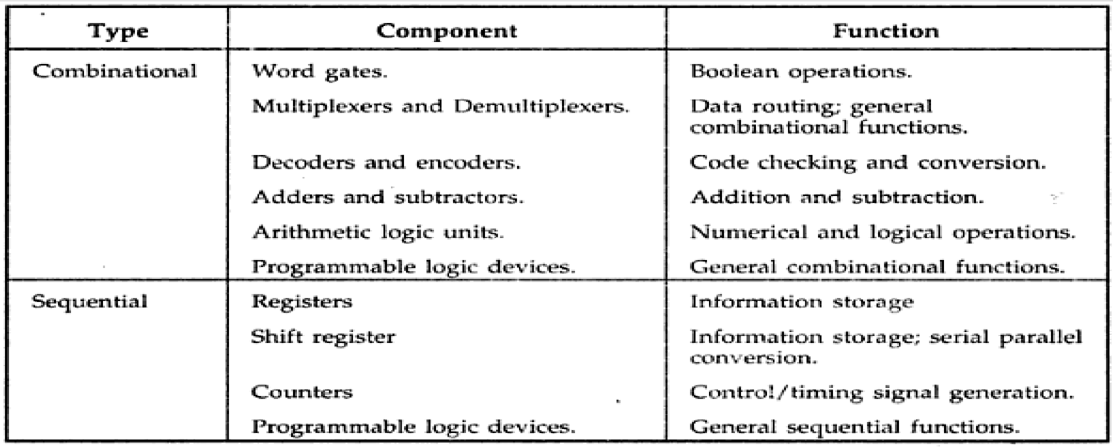
Register Level Design

At the register or register transfer level, related information bits are grouped to form words or vectors.

These words are processed by small combinational or sequential circuits

Register Level Components

The Table below shows the commonly used register level components and their functions. These components are link to form circuits.



Type	Component	Function
Combinational	Word gates.	Boolean operations.
	Multiplexers and Demultiplexers.	Data routing; general combinational functions.
	Decoders and encoders.	Code checking and conversion.
	Adders and subtractors.	Addition and subtraction.
	Arithmetic logic units.	Numerical and logical operations.
	Programmable logic devices.	General combinational functions.
Sequential	Registers	Information storage
	Shift register	Information storage; serial parallel conversion.
	Counters	Control/timing signal generation.
	Programmable logic devices.	General sequential functions.

Fig: Commonly used register level components

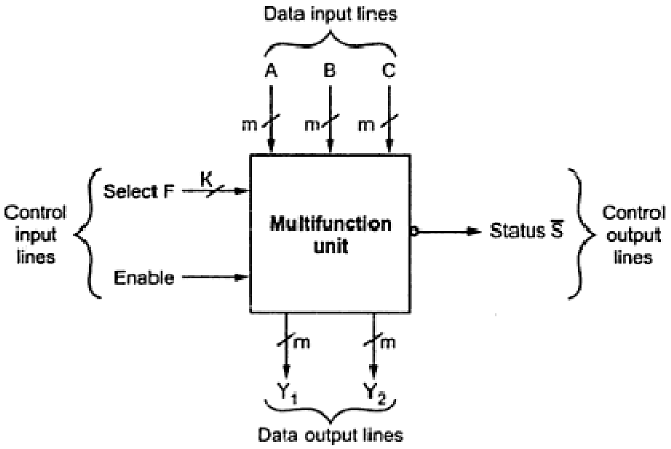


Fig: Generic block representation of a register level component

The Fig. above shows the generic block representation of a register-level component.

- The "/m" on the input lines indicate it is a m-bit input bus.
- A slash'/' with number or letter next to it indicates the multi-bit bus.
- A bubble on the start or end of the line indicates an active low signal; otherwise it is an active high signal.
- The input and output data lines are shown separately.
- Similarly the input and output control lines are also shown separately.
- The input control lines associated with a multifunction block fall into two broad categories: select lines and enable lines. The select lines specify one of several possible operations that the unit is to perform and enable lines specify the time or condition for a selected operation to be performed.
- The output control signals, if any, indicate when or how the unit completes its processing.

Word Gate: Bit-wise logical functions can be performed on the m-bit binary words using word gate operators. Let A = (al, a2,.., a_m) and B = (b1, b2,…, b_m) be the two m-bit words we can perform the bitwise AND operation on them to result another m-bit result, as shown in the Fig. below

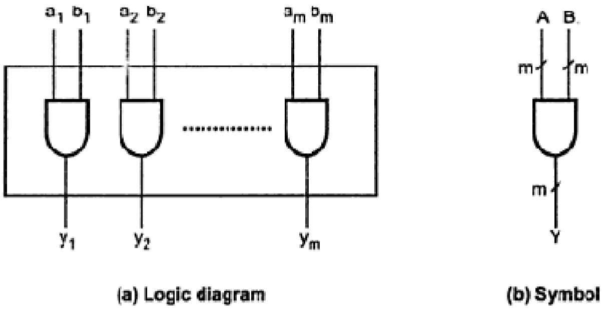


Fig.: Two-Input, m-bit AND word gate

Multiplexers

Multiplexer is a digital switch. It allows digital information from several sources to be routed onto a single output line, as shown in the Fig. below. The basic multiplexer has several data-input lines and a single output line. The selection of a particular input line is controlled by a set of selection lines. Normally, there are 2n input lines and n selection lines whose bit combinations determine which input is selected. Therefore, multiplexer is 'many into one' and it provides the digital equivalent of an analog selector switch.

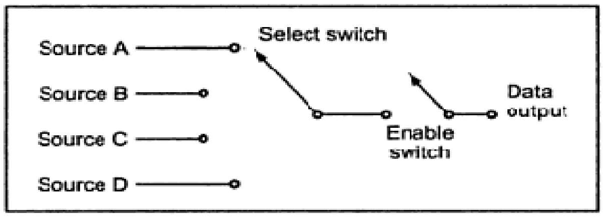


Fig: Analog selector switch

Decoder

A decoder is a multiple-input, multiple-output logic circuit which converts coded inputs into coded outputs, where the input and output codes are different. The input code generally has fewer bits than the output code. Each input code word produces a different output code word, i.e., there is one-to-one mapping from input code words into output code words. This one-to-one mapping can be expressed in a truth table.

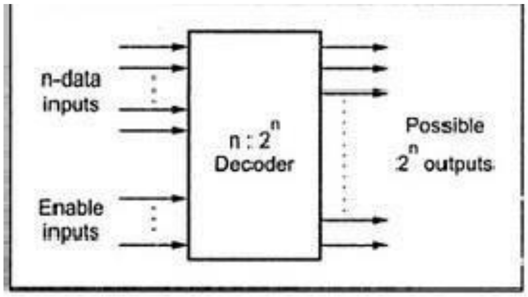


Fig : general structure of decoder

The Fig. above shows the general structure of the decoder circuit. As shown in the Figure, the encoded information is presented as n inputs producing 2" possible outputs. The 2" output values are from 0 through 2" - 1. Sometimes an n-bit binary code is truncated to represent fewer output values than 2n. For example, in the BCD code, the 4-bit combinations 0000 through 1111 are not used. Usually ,a decoder is provided with enable inputs to activate decoded output based on data inputs. When any one enable input is unasserted, all outputs of decoder are disabled.

Encoder

An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has 2" (or fewer) input lines and n output lines. In encoder the output lines generate the binary code corresponding to the input value. The Fig. below shows the general structure of the encoder circuit. As shown in the Fig. below, the decoded information is presented as 2" inputs producing n possible outputs.

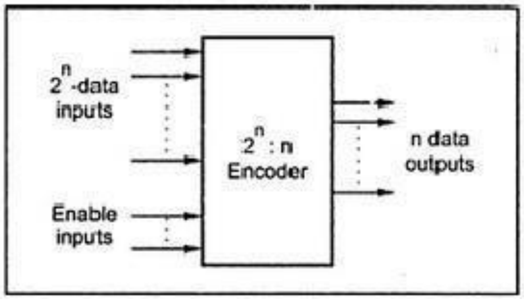
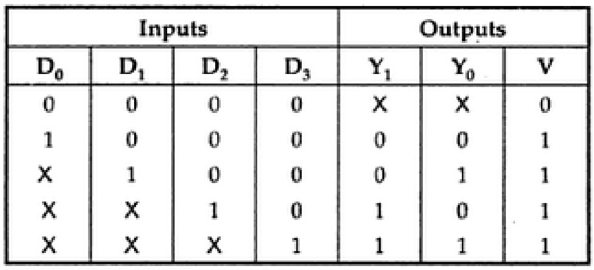


Fig: General structure of encoder

A priority encoder is an encoder circuit that includes the priority function. In priority encoder, if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

Table : shows truth table of 4-bit priority encoder.



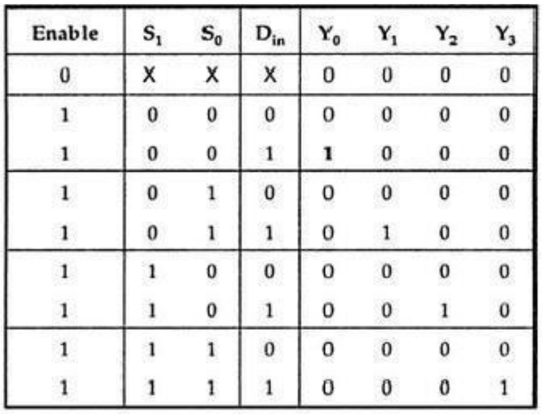
Inputs				Outputs		
D_0	D_1	D_2	D_3	Y_1	Y_0	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Table: Truth table of 4-bit priority encoder

Table above shows D3 input with highest priority and D0 input with lowest priority. When D3 input is high, regardless of other inputs output is 1 1. The D2 has the next priority. Thus, when D3 = 0 and D2 = 1, regardless of other two lower priority input, output is 10. The output for D, is generated only if higher priority inputs are 0, and so on. The output V (a valid output indicator) indicates, one or more of the inputs are equal to 1. If all inputs are 0, V is equal to 0, and the other two outputs (Y, and Y0) of the circuit are not used.

Demultiplexer

A demultiplexer is a circuit that receives information on a single line and transmits this information on one of 2" possible output lines. The selection of specific output line is controlled by the values of n selection lines. Fig. 2.18 shows1:4demultiplexer. The single input variable Dm has a path to all four outputs, but the input information is directed to only one of the output lines.



Enable	S_1	S_0	D_{in}	Y_0	Y_1	Y_2	Y_3
0	X	X	X	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	1	0	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	0
1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	1

Table: Function table for 1:4 demultiplexers

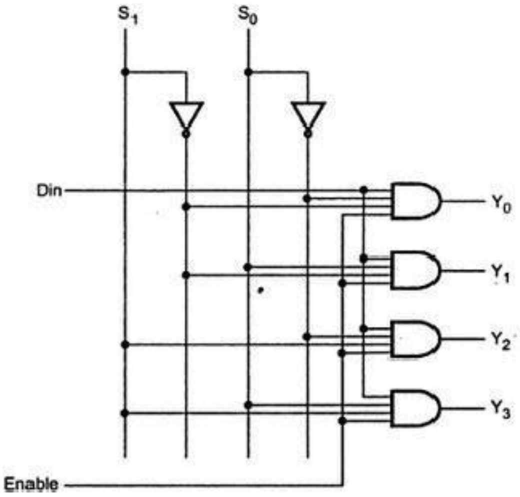


Fig: logic diagram

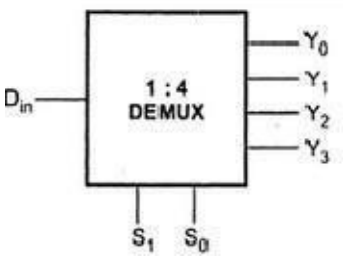
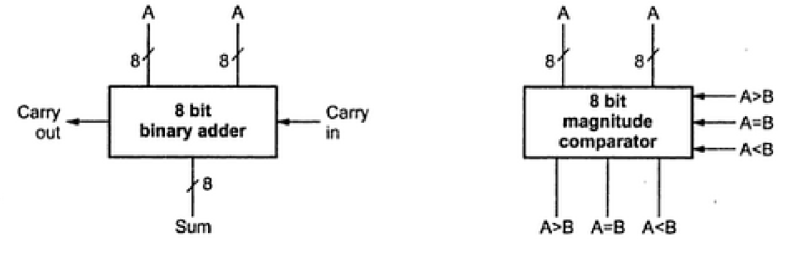


Fig: Block Diagram

Arithmetic Elements

The arithmetic functions such as addition and subtraction of fixed point numbers can be implemented by combinational register-level components. Most forms of fixed-point multiplication and division and essentially all floating-point operations are too complex to be realized by single component at this design level. However, adders and subtractors for fixed- point binary numbers are basic register level components from which we can derive a variety of other arithmetic circuits. The Fig. (a) shows a component that adds two 8-hit data words and an input carry bit; it is called a 8-bit adder. One such component can be cascaded to form an adder to add numbers of arbitrary size.



(a) Symbol for 8-bit binary adder

(b) Symbol for 8-bit magnitude comparator

However, addition time increases with the number size.

Register

A register is a group of flip-flops. A flip-flop can store 1-bit information. So an n-bit register has a group of n lip-lops and is capable of storing any binary information/number containing n-bits. **Buffer Register** Figure below shows the simplest register constructed with four D flip-flops. This register is also called buffer register. Each D-lip-lop is triggered with a common negative edge clock pulse. The input X bits set up the flip-lops for loading. Therefore, when the first negative clock edge arrives, the stored binary information becomes, $Q_AQ_BQ_CQ_D$ = ABCD

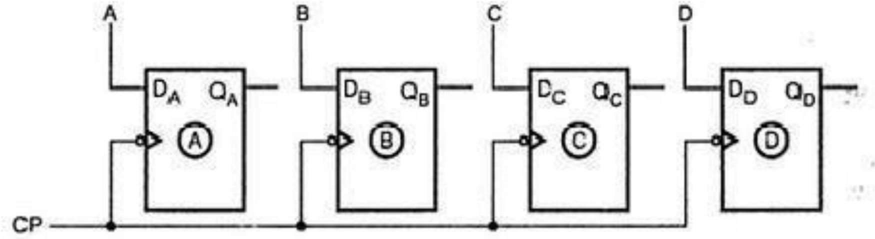


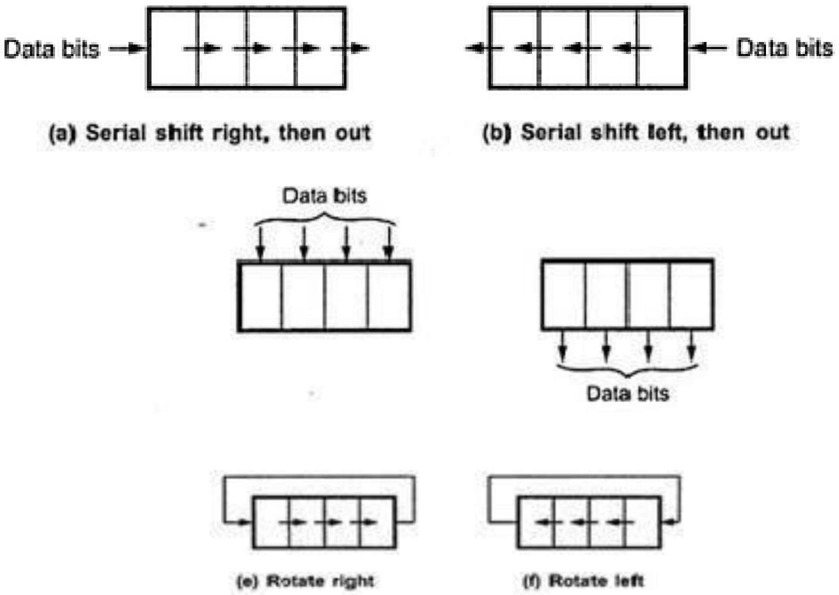
Fig: Buffer register

In this register, four D flip-flops are used. So it can store 4-hit binary information. Thus the number of flip-flop stages in a register determines its total storage capacity.

Shift Registers

The binary information (data) in a register can be moved from stage to stage within the register or into or out of the register upon application of clock pulses. This type of bit movement or shifting is essential for certain arithmetic and logic operations used in microprocessors. This gives rise to a group of registers called 'shift registers'. They are very important in applications involving the storage and transfer of data in a digital system.

Fig below gives the symbolical representation of the different types of data movement in shift register operations.



(a) Serial shift right, then out

(b) Serial shift left, then out

(e) Rotate right

(f) Rotate left

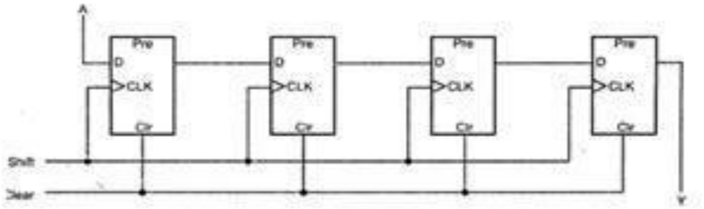


Fig.: Basic data movement In registers

Fig:(a)

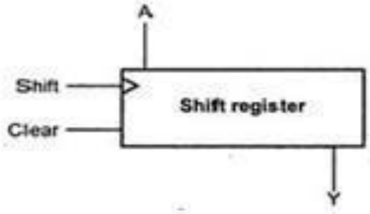


Fig: (b)

Fig: 4-bit right shift register (a) Logic diagram (b) Symbol

The Fig. above shows the register level implementation of right shift register using D flip-flops. A right shift is accomplished by activating the SHIFT enable line connected to the clock input CLK of each flip-flop. In addition to the serial data lines, m input and output lines are often provided to permit parallel data transfers to or from the shift register. Additional control lines are required to select the serial or parallel input modes. The shift register further can be refined to permit both left and right shift operations. The Fig. below shows the shift register with parallel and serial modes along with the right and left shifts operations.

Counters

A register is used solely for storing and shifting data which is in the form of is and/or Os, entered from an external source. It has no specific sequence of states except in certain very specialized applications. A counter is a register capable of counting the number of clock pulses arriving at its clock input. Count represents the number of clock pulses arrived. A specified sequence of states appears as the counter output. This is the main difference between a register and a counter. A specified sequence of states is different for different types of counters.

There are two types of counters, synchronous and asynchronous. In synchronous counter, the common clock input is connected to all of the flip-flops and thus they are clocked simultaneously. In asynchronous counter, commonly called, ripple counters, the first flip-flop is clocked by the external clock pulse and then each successive flip-flop is clocked by the Q or Q output of the previous flip-flop. Therefore in an asynchronous counter, the lip-lops are not clocked simultaneously.

The Fig. below shows the symbol for a Enable (dock) modulo 2" up-down counter. On receiving positive going edge on the Enable (clock signal) input counter increments its count by 1. As it is an n-bit counter, its counting is modulo-2"; that is, the counter's modulus k = 2", and it has 2" states So, St,, S,,, 1The output of counter is an n-bit binary number. The CLEAR input of the counter when activated resets the counter and UP/DOWN input selects the operation of the counter.

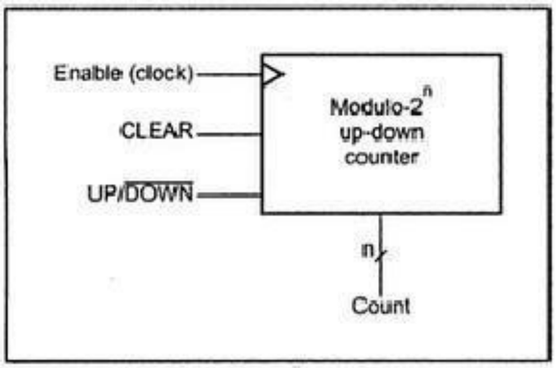


Fig: A modulo 2" up-down counter

Programmable Logic Devices

There are many applications for digital logic where the market is not great enough to develop a special-purpose MSI or LSI chip. This situation has led to the development of programmable logic devices (PLDs) which can be easily configured by the individual user for specialized applications.

Basically, there are three types of PLDs:

- Read Only Memory(ROM)

- Programmable Logic Array(PLA)
- Programmable Array Logic(PAL)

Here, we examine programmable logic devices as a new class of components.

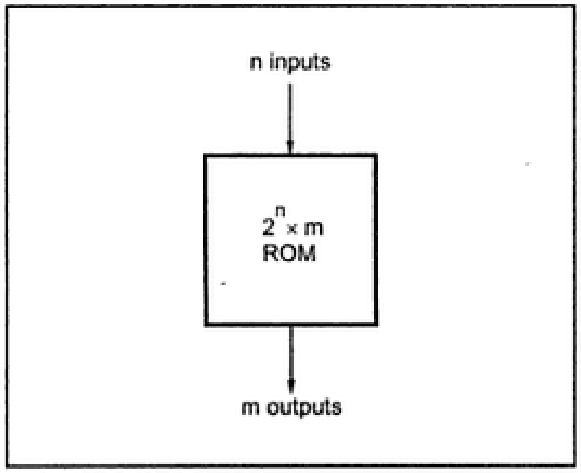


Fig. :Block diagram of ROM

A read only memory(ROM)is a device that includes both the encoder and the OR gates within a single IC package.

The Fig. above shows the block diagram of ROM. It consists of n input lines and m output lines. Each bit combination of the input variables is called an address. Each bit combination that comes out of the output lines is called a word. The number of bits per word is equal to the number of output lines, m. The address specified in binary number denotes one of the minterms of n variables. The number of distinct addresses possible with n input variables is 2°. An output word can be selected by a unique address, and since there are 2" distinct addresses in a ROM, there are 2" distinct words in the ROM. The word available on the output lines at any given time depends on the address value applied to the input lines.

Let us consider 64 x 4 ROM. The ROM consists of 64 words of 4 bits each. This means that there are four output lines and particular word from 64 words presently available on the output lines is determined from the six input lines. There are only six inputs in a 64 x 4 ROM because 2b = 64, and with six variables, we can specify 64 addresses or minterms. For each address input, there is a unique selected word. Thus, if the input address is 000000, word number 0 is selected and applied to the output lines. If the input address is 111111, word number 63 is selected and applied to the output lines.

Programmable Read Only Memory (PROM)

Programmable Read Only Memory (PROM) allows user to store data/program. PROMs use the fuses with material like nichrome and polycrystalline. The user can blow these fuses by passing around 20 to 50 mA of current for the period 5 to 20 us. The blowing of fuses according to the truth table is called programming of ROM. The user can program PROMS with special PROM programmer. The PROM programmer selectively burns the fuses according to the bit pattern to be stored..This process is also known as burning of PROM. The PROMs are one time programmable. Once programmed, the information stored is permanent.

EPROM (Erasable Programmable Read Only Memory)

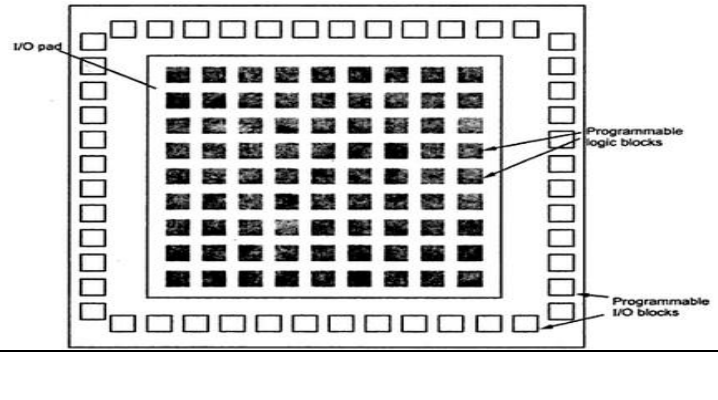
Erasable programmable ROMs use MOS circuitry. They store is and Os as a packet of charge in a buried layer of the IC chip. EPROMs can be programmed by the user with a special EPROM programmer. The important point for now is that we can erase the stored data in the EPROMs by exposing the chip to ultraviolet light through its quartz window for 15 to 20 minutes. In EPROMs, it is not possible to erase selective information; when erased, the entire information is lost. The chip can be reprogrammed. This memory is ideally suitable for product development, experimental projects, and college laboratories, since this chip can be reused many times.

EEPROM (Electrically Erasable Programmable Read Only Memory)

Electrically erasable programmable ROMs also use MOS circuitry very similar to that of EPROM. Data is stored as charge or no charge on an insulated layer or an insulated floating gate in the device. The insulating layer is made very thin (< 200 A). There fore, a voltage as low as 20 to 25V can be used to move charges across the thin barrier in either direction for programming or erasing. EEPROM allows selective erasing at the register level rather than erasing all the information since the information can be changed by using electrical signals. The EEPROM memory also has a special chip erase mode by which entire chip can be erased in 10 ms. This time is quite small as compared to time required to erase EPROM, and it can be erased and reprogrammed with device right in the circuit. However, EEPROMs are most expensive and the least dense ROMs.

Field Programmable Gate Arrays

In mid-1980s, an important class of PLDs was introduced, called field-programmable gate array. The Fig. below shows the general structure of FPGA chip. It consists of a large number of programmable logic blocks surrounded by programmable I/O block. The programmable logic blocks of FPGA are smaller and less capable than a PLD, but an FPGA chip contains a lot more logic blocks to make it more capable. As



shown in the Fig. below, the logic blocks are distributed across the entire chip. These logic blocks can be interconnected with programmable inter connections. As compared to standard gate arrays, the field programmable gate arrays are larger devices. The basic cell structure for FPGA is somewhat complicated than the basic cell structure of standard gate array. The FPLA use read/write memory cell to control the state of each connection. The word field in the name refers to the ability of the gate arrays to be programmed for a specific function by the user instead of by the manufacturer of the device. The word array is used to indicate a series of columns and rows of gates that can be programmed by the end user. Two types of logic cells found in FPGAs are those based on multiplexers and those based on PROM table-lookup memories.

Register Level Design:

At register level design, a set of registers are linked by combinational data transfer and data-processing circuits. A block diagram defines its structure and the set of operations it performs on data words can define its behavior. Each operation can be defined in the form: Cond : Z = f (A1, A2, A3, ... ,AL)

where f is a function to be performed or an instruction to be executed in one clock cycle, and A1, A2, Ay ..., Ak and Z denote data words or the registers that store data. The prefix 'cond' denotes a control condition that must be satisfied (cond = 1) for the indicated operation to take place. Therefore, when 'con& = 1 the function f is computed on AI, A2, A, ... and Ai, and result is stored in the Z data word.

Data and Control

The Fig.(a) below shows simplest register level system. It performs operation Z = A + B. The Fig (b) shows a more complicated system that can perform several different operations. Such a multifunction system can perform only one operation at a time. he operation to be perform is decided by the control signals. Therefore, the multifunction system is partitioned into a data-processing part called a data path, and a controlling part called a control unit. The control unit is responsible for selecting and controlling the actions of the data path.

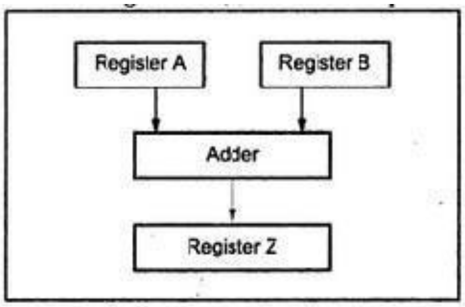


Fig. (a) Simple registers level system

As shown in Fig. (b) below, control unit (CU) selects the operation for the ALU to perform in each clock cycle. It also determines the input operands to apply to the ALU and the destination of its results. The large extension to this multifunction unit is the computer's CPU. The computer's CPU is responsible for the interpretation of instructions and generation of required control signals for the execution of the instructions. This unit of computer called I-unit and the data path unit of computer is called E-unit.

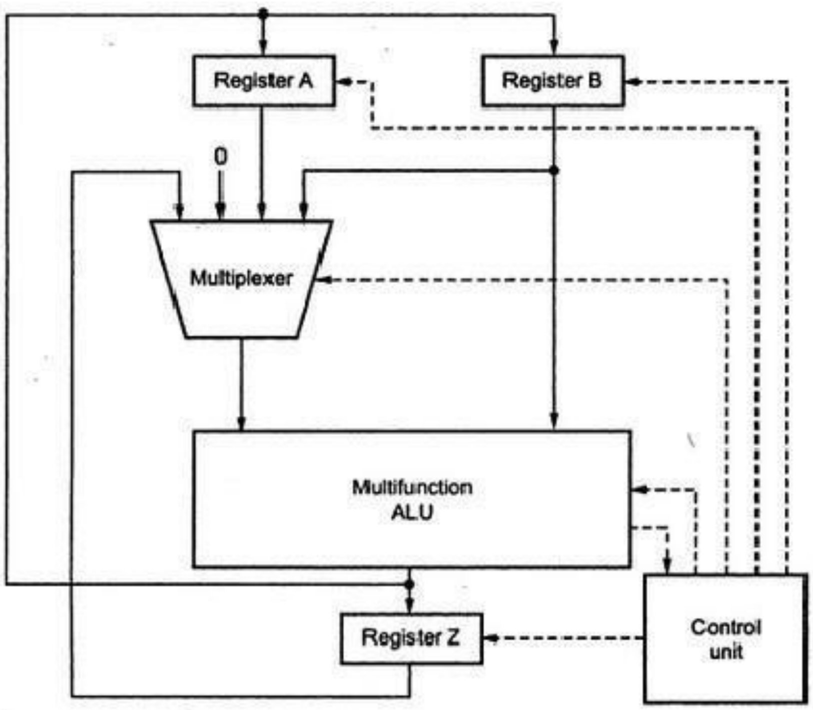
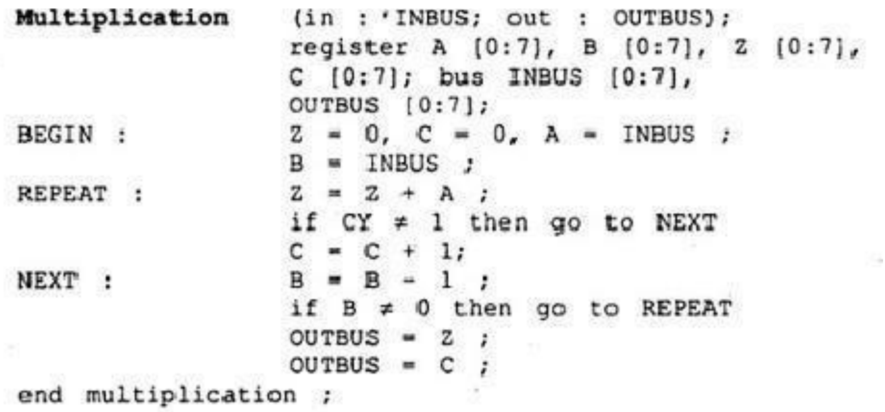


Fig. (b) Multifunction registers level system

A Description Language

The HDL can be used to provide both behavioral and structural description at the register level. For example, if cond = 1 then Z = f (A1, A2, A3, , Ak) ; where f can be any function. For example, Z = A + B or Z = A - B. Here, + represents the adder and - represents the subtractor. The input connections in both the cases from registers A and B are inferred from the fact that A and B are the arguments of + and -, while the output connection from the adder/subtractor is inferred.

Let us see the formal language description of an 8-bit binary multiplier. We know that, the multiplication can be performing in two ways: 1. Repetitive addition 2. Shift and add. Here, our intention is to study the language descriptions, hence we prefer simple method of multiplication, i.e. multiplication by reparative addition.



```
Multiplication    (in : 'INBUS; out : OUTBUS);
                  register A [0:7], B [0:7], Z [0:7],
                  C [0:7]; bus INBUS [0:7],
                  OUTBUS [0:7];
BEGIN :           Z = 0, C = 0, A = INBUS ;
                  B = INBUS ;
REPEAT :          Z = Z + A ;
                  if CY ≠ 1 then go to NEXT
                  C = C + 1;
NEXT :            B = B - 1 ;
                  if B ≠ 0 then go to REPEAT
                  OUTBUS = Z ;
                  OUTBUS = C ;
end multiplication ;
```

In the above program, two 8-bit buses INBUS and OUTBUS form multipliers input and output ports, respectively. The program initializes these buses with statement: bus INBUS (0:7), OUTBUS (0:7) and initialize 8-bit registers A, B, Z and C with statement: register A (0: 7), B (0: 7), Z(: 7) and C (0:). The registers Z and C are initialized to store the lower byte and higher byte of multiplication, respectively. Initially, the result (Z and C) is made 0, and register A and B are loaded with multiplicand and multiplier from the INBUS, respectively. The multiplicand is added repeated for multiplier times and result is stored in the Z and C registers. The carry after addition of lower byte is used to increment the value in the higher byte register, i.e. C register. The final result is then transferred 8 bits at a time to OUTBUS.

Design Techniques

The general approach to the design problem for register level system is as follows:

1. Define the desired behavior of the system by a set of sequences of register-transfer operations, such that each operation can be implemented directly using the available design components. This gives the desired algorithm.
2. Analyze the algorithm, to determine the types of components and the number of each type required for the datapath.
3. Construct a block diagram for datapath using the components identified in step 2. Make the connections between the components so that all data paths implied by algorithm are present and the given performance-cost constraints are met.
4. Analyze algorithm and datapath to identify the control signals needed. Introduce the logic or control points necessary to apply these signals into data path.
    i. Design a control unit for datapath that meets all the requirements of algorithm.
    ii. Check whether the final design operates correctly and meets all performance-cost goals.

A design of algorithm in step 1 is a creative design process. It is similar to writing computer program and depends heavily on the skill and experience of the designer. The second step is to identify the data processing components. It is a straight forward process. However, it becomes complicated when the possibility of sharing components exists. For example, the perform operation Cond: A = A + B, C = C + D; requires two adders, because the operation has to perform in parallel. However, if we use single adder and perform operations serially we can lower the cost by sharing a single adder. Thus: Cond (to): A = A + B Cond (to + 1): C = C + D. The step3 requires defining an interconnection structure that links the component needed by the various parts of algorithm. Identifying the control signals and design of control unit in step 4 and step 5, respectively, is a relatively independent process. The step 6, design verification plays an important role in the development process. The simulation via CAD tool can be used to identify and correct functional errors before new design is committed to hardware.

Processor Level Design:

The processor level which is also called system level is the highest in the hierarchy of computer design. The storage and processing of information are the major objectives of this level. Processing involves execution of programs and processing of data files. The components required for performing these functions are complex. Usually sequential circuits are used which are based on VLSI technology. A slight design theory is necessary at this level of abstraction.

Processor Level Components

The different types of components which ac generally used at this level can be divided mainly into four groups as

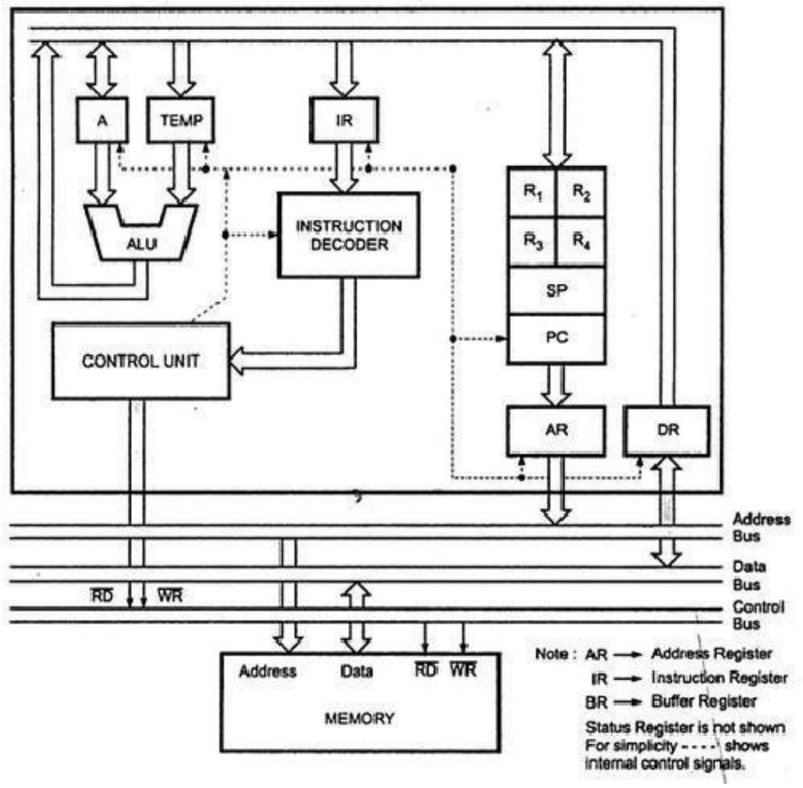
- Processors
- Memories
- I/O devices
- Inter connection networks

Central Processing Unit

The primary function of a central processing unit is to execute sequences of instructions stored in a memory, which is external to the central processing unit. When the functions of the processor are restricted, those processors become more specialized processor such as I/O processor. Most of the times CPUs are microprocessors whose physical implementation is a single VLSI chip. Fig. below shows a typical CPU structure and its connection to memory: The CPU contains different units such as control unit, arithmetic logic unit, register unit, decoding unit which are necessary for the execution of instructions. The sequence of operations involved in processing an instruction constitutes an instruction cycle. This can be subdivided into three major phases: fetch cycle, decode cycle and execute cycle. The address of the next instruction which is to be fetched from memory is in the program counter (PC). During fetch phase CPU

loads this address in address register (AR). This is the register which gives address to the memory. Once the address is available on the address bus, the read command from control unit copies the contents of addressed memory location to the instruction register (IR). During decode phase, the instruction in the IR is decoded by instruction decoder. In the next, i.e. execute phase CPU has to perform a particular set of micro-operation depending on the instruction.

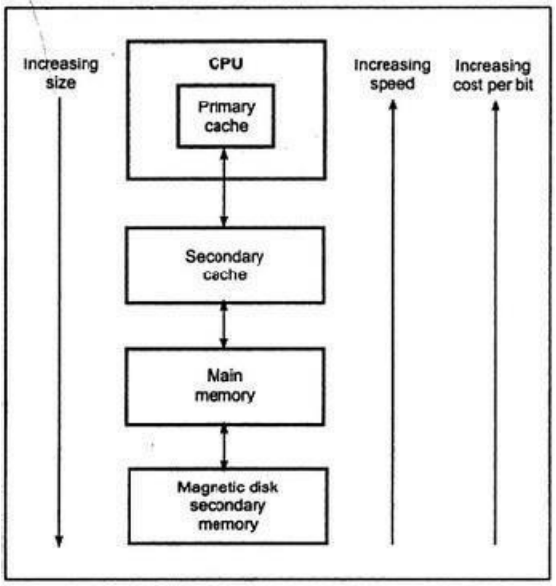
All these operations are synchronized with the help of clock signal. The frequency of this signal is nothing but the operating frequency of CPU. Thus the CPU is a synchronous sequential circuit and its clock period is the computer's basic unit of time.



A
TEMP
IR
ALU
INSTRUCTION DECODER
R1
R2
R3
R4
SP
PC
CONTROL UNIT
AR
DR
Address Bus
Data Bus
Control Bus
RD
WR
Address
Data
RD
WR
MEMORY
Note : AR → Address Register
IR → Instruction Register
BR → Buffer Register
Status Register is not shown
For simplicity - - - - shows
internal control signals.

Fig.: Typical CPU structure

Memories: For the storage of programs and data required by the processors, external memories are necessary. Ideally, computer memory should be fast, large and inexpensive. Unfortunately, it is impossible to meet all the three of these requirements simultaneously. Increased speed and size arc achieved at increased cost. Very fast memory system can be achieved if SRAM chips are used. These chips are expensive and for the cost reason it is impracticable to build a large main memory using SRAM chips. The only alternative is to use 'DRAM chips for large main memories. Processor fetches the code and data from the main memory to execute the program. The DRAMs which form the main memory are slower devices. So it is necessary to insert wait states in memory read/write cycles. This reduces the speed of execution. The solution for this problem is come out with the fact that most of the computer programs work with only small sections of code and data at a particular time. In the memory system small section of SRAM is added along with main memory, referred to as cache memory. The program which is to be executed is loaded in the main memory, but the part of program (code) and data that work at a particular time is usually accessed from the cache memory. This is accomplished by loading the active part of code and data from main memory to cache memory. The cache controller looks after this swapping between main memory and cache memory with the help of DMA controller. The cache memory just discussed is called secondary cache. Recent processors have the built-in cache memory called primary cache. DRAMs along with cache allow main memories in the range of tens of megabytes to be implemented at a reasonable cost, the size and better speed performance. But the size of memory is still small compared to the demands of large programs with voluminous data. A solution is provided by using secondary storage, mainly magnetic disks and magnetic tapes to implement large memory spaces. Very large disks are available at a reasonable price, sacrificing the speed. From the above discussion, we can realize that to make efficient computer system it is not possible to rely on a single memory component, but to employ a memory hierarchy. Using memory hierarchy all of different types of memory units are employed to give efficient computer system. A typical memory hierarchy is illustrated in Fig. above. In summary, we can say that a huge amount of cost-effective storage can be provided by magnetic disks. A large, yet affordable, main memory can be built with DRAM technology along with the cache memory to achieve better speed performance.



I 0 Devices

A computer communicates with outside world by means of input-output (I 0) system. The main function of

I/O system is to transfer information between CPU or memory and the outside world. The important point to be noted here is, I/O devices (peripherals) cannot be connected directly to the system bus. The reasons are discussed here.

i. A variety of peripherals with different methods of operation are available. So it would be impractical to incorporate the necessary logic within the CPU to control a range of devices.

ii. The data transfer rate of peripherals is often much slower than that of the memory or CPU. So it is impracticaltousethehighspeedsystembustocommunicatedirectlywiththeperipherals.

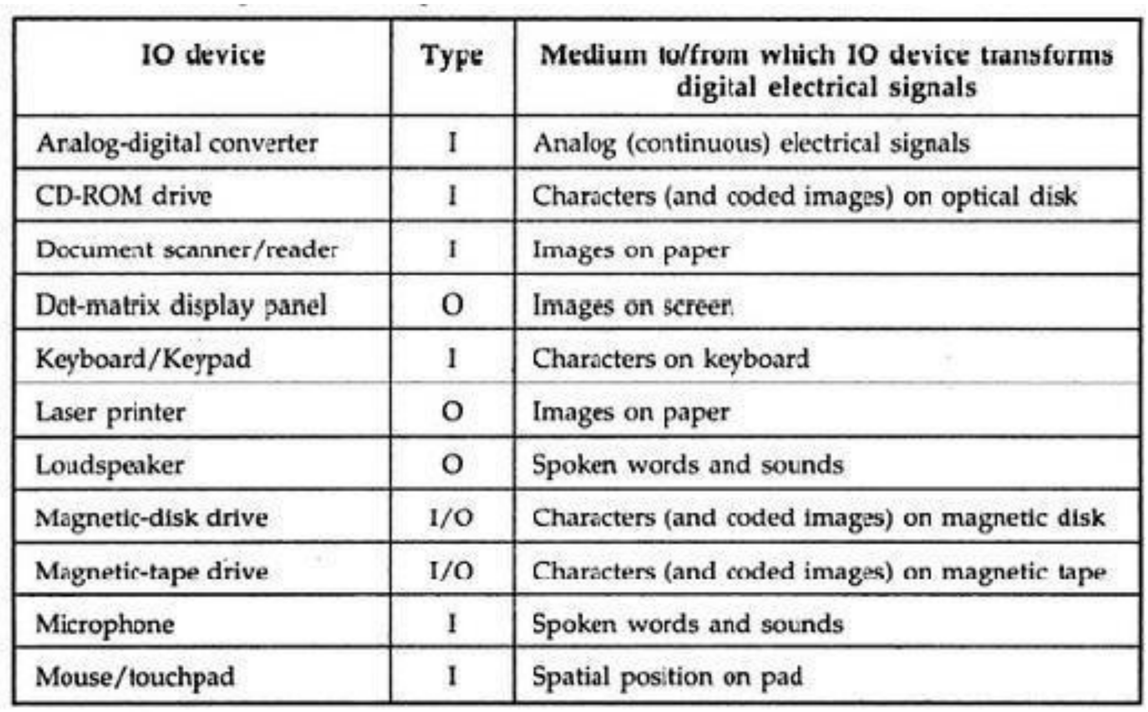
iii. Generally, the peripherals used in a computer system have different data formats and word lengths than that of CPU used in it.

So to overcome all these difficulties, it is necessary to use a module in between system bus and peripherals, called as 1/0 module or I/O system.

This I/O system has two major functions,

- Interface to the CPU and memory via the system bus,
- Interface to one or more I/O devices by tailored data links.

The table below gives list of representative I 0devices.



IO device	Type	Medium to/from which IO device transforms digital electrical signals
Aralog-digital converter	I	Analog (continuous) electrical signals
CD-ROM drive	I	Characters (and coded images) on optical disk
Document scanner/reader	I	Images on paper
Dct-matrix display panel	O	Images on screen
Keyboard/Keypad	I	Characters on keyboard
Laser printer	O	Images on paper
Loudspeaker	O	Spoken words and sounds
Magnetic-disk drive	I/O	Characters (and coded images) on magnetic disk
Magnetic-tape drive	I/O	Characters (and coded images) on magnetic tape
Microphone	I	Spoken words and sounds
Mouse/touchpad	I	Spatial position on pad

Interconnection Networks

The processor level components, CPU, memories, I 0 devices communicate via system bus (address bus, data bus and control bus). In a computer system, when many components are used, communication between these components may be controlled by a subsystem called an interconnection network. Switching network, communications controller, bus controller are the examples of such subsystem. Under the control of interconnection network, dynamic communication paths among the components via the buses can be established. The communication paths are shared by the components, to reduce cost. At any time, communication and hence use of shared bus is possible between any two components. When more than two components request use of the bus, it results in bus contention. The function of the interconnecting network is to resolve such contention. For performing this function, interconnecting network selects one of the requesting devices on some priority basis and connects it to the bus. The remaining requesting devices are kept in a queue.

Some evolutionary steps in the I/O function arc summarized here.

1. In simple microprocessor-controlled devices, a peripheral device is directly controlled by CPU.
2. AcontrolleristhenaddedtoCPUtocontrolperipheraldeviceswithprogrammingfacility.
3. Nowinterruptsareemployedintheconfigurationmentionedinstep2.ThissavestheCPUtime which was required for a polling of I/O device.
4. DMA controller is introduced to give direct access to memory for I/O module.
5. The I/O module is then enhanced to become a processor with a specialized instruction set tailored for I/0. The I/0 processor is capable of executing an I/O program in memory with directions given by CPU. It can execute I/O program without intervention of CPU.
6. The I/O module is further enhanced to have its own local memory. This makes it possible to control a large set of I/O devices with minimal CPU involvement.

In step 5 and step 6 we have seen that the I/O module is capable of executing programs. Such I/0 modules is commonly known as I/0 channels. Generally the communication between processor-level components is asynchronous since they cannot access some unit or bus simultaneously and hence components cannot be synchronized directly by a common clock signal. The following different causes can be stated regarding this synchronization problem.

- The speed of operation of different components varies over a wide range. e.g. CPUs are faster than main memories and main memories are faster than I/O devices.
- The different components work more independently. e.g. execution of different programs by CPUs and10Ps.
- It is practically difficult to allow synchronous transmission of information between components due to large physical distance between them.

Processor-Level Design

While designing any system, it is very much difficult to give a precise description of the desired system behavior. Because of this reason, the processor level design job is critical as compare to register level design. Generally to design at this level, a prototype design of known performance is taken. Then according to the necessity new technologies are added and new performance requirements are achieved.

Performance characteristics

Year by year, the cost of computer systems continues to drop dramatically, while the performance and capacity of those systems continue to rise equally dramatically. In this section we introduce some basic aspects of computer system performance characteristics. The total time needed to execute application programs is the most important measure of computer system performance. In other words we can say that the speed of the computer system is an important characteristic to define the performance of the computer system. The speed of the computer system depends on various factors. Let us discuss those factors.

- ➢ **Hardware**: The speed of the processor used in the computer systems basically decides the speed of the computer system. For example, system having processor Pentium IV runs faster than system having Pentium 1. However, system speed is not only depends on the processor speed, but it is also affected by the supported hardware.

Each processor has its own address bus width and data bus width, internal registers, on chip memory and the instruction set. Higher data bus width allows the transfer of data with more number of bits at a time. For example, data bus width of 64-bit allows 64 bits transfer of data at a time and data bus width of 32-bit allows 32-bits transfer of data at a time. Higher address bus width gives higher addressing capacity. More number of internal registers allow to store partial results and avoid unnecessary memory accesses resulting faster operation. Similarly, on-chip memory allows to store currently executing program module, required data and partial results in CPU itself which can be accessed quickly resulting faster operation. The system speed is also depends on the speed of the secondary memory, the speed of the I/O parts and the speed of data transfer between them.

- **Programming Language** : Now a days, programs are usually written in a high-level languages. These languages require separate compiler to translate programs into machine level language. Therefore, the computer system is affected by the performance of the compiler and hence the language used for the program.

- **Pipelining**: The processor executes the instruction in steps or phrases such as fetch, decode and execute. By overlapping these phases of successive instructions we can achieve a substantial improvement in the performance of the computer system. This technique is known as pipelining.

- **Parallelism**: It is possible to perform transfers to and from secondary memory like storage disk or tapes in parallel with program execution in the processor or with activity in other I/O devices. Such technique is known as parallelism. The most of computer system use this parallelism to the improve the system performance.

- **Types of memory and I 0 devices**: The performance of the computer depends on the type of memory and 10 devices supported by it.

- **Compatibility with other types of computer cost**: The performance of a computer system is also decided by the total cost of the system.

All these performance specifications are considered while designing a new computer system. Even though the new computer design is closely based on a known design, accurate performance prediction of the new system may not be possible. For accurate performance, the understanding of the relation between the structure of a computer and its performance is very important. Using some mathematical analysis, a little amount of useful performance evaluation can be done. For performance evaluation, experiments during the design process are to be performed. For this purpose computer simulation can be used or performance of the copy of the machine under working conditions can be measured.

Prototype Structures

The processor-level design using prototype-structures involve following steps in the design process. First select a prototype design as per the system requirements and adapt it to satisfy the given performance constraints.

1. Determine the performance of proposed system.
2. If the performance is unsatisfactory, design is to be modified. Repeat step1.
3. The above steps are to be continued until the acceptable design is obtained and the desired performance constraints are achieved.

These steps are widely followed for designing a computer system. While designing new systems, the precautions are always taken to remain compatible with existing hardware and software standards. The reason is that when these standards are to be changed, computer owners have to spend money to retrain users and programmers.

Also the well tested software is to be replaced by the modified software. So in the new design of the computer system the drastic changes in the previous design are generally avoided. Because of all these reasons, there is slow evolution of computer architecture.

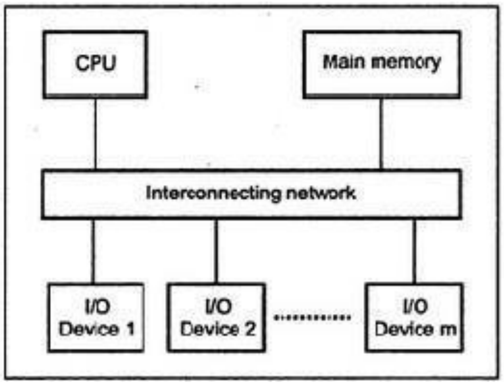


Fig. :Basic computer structure

Figure above shows the structure of first-generation computers. This is the basic computer structure. The second and subsequent generations of computer involve special-purpose 10 processors and cache memory in addition to basic components used within the basic system. This advanced structure is shown on nest page.

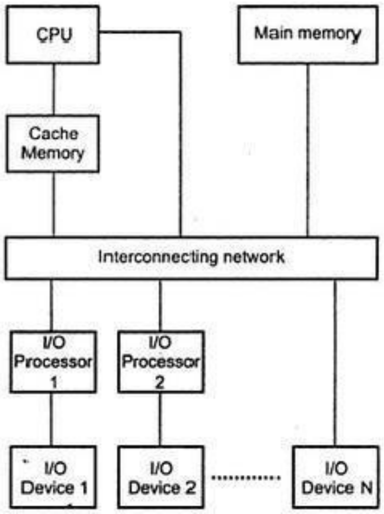


Fig.: Computer structure with 10 processors and cache memory

The more advanced structure involves more than one CPU, i.e. a multiprocessor system. Fig. below gives the computer structure with two CPUs, main memory banks.

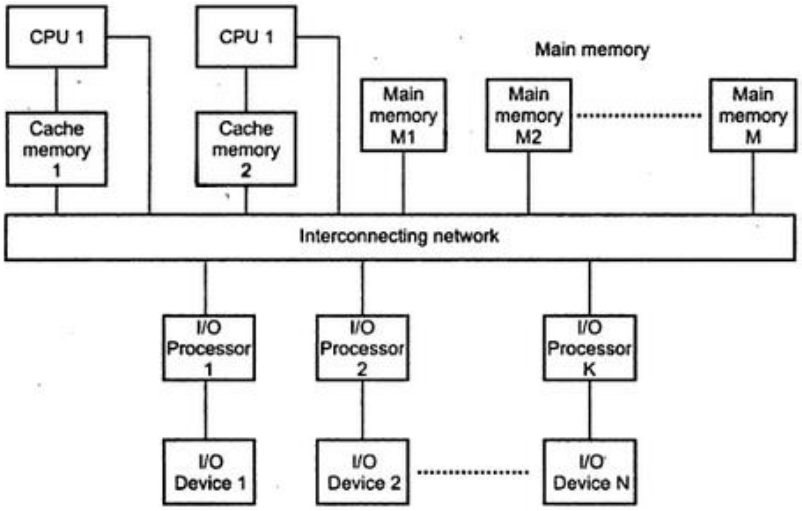


Fig: Computer structure with two CPUs, main memory banks

If we link several copies of the foregoing prototype structures, more complex structures of computer can be obtained. Computer Network is an example of such a structure.

Queuing Models

In this section we will discuss an analytic performance model of a computer system. The model which is discussed here is based on queuing theory. The model which is considered here is, M/M/1 model. The first M indicates the inter arrival time between two successive items requiring service from the server.

The items are served in their order of arrival. i.e. First Come First Service (FCFS) scheduling. The second M indicates service time distribution. I indicate number of service facility centers. Fig. below shows simple queuing model of a computer.

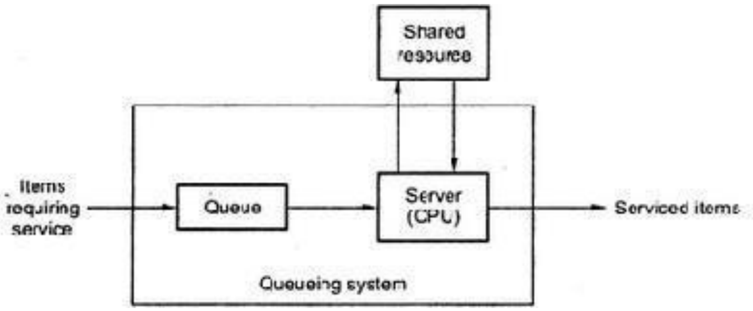


Fig. : Simple queuing model of a computer system

CPU is used as a server. The items or tasks requiring service by the CPU are queued in a memory. One task is processed at a time by the CPU. The tasks from the queue are processed (serviced) by the CPU on FCFS basis. There are different performance parameters which can characterize the steady-state performance of the single-server queuing system.

1. **Traffic Intensity:** It is denoted by p and given by p = X / p. It is the average fraction of time the server is busy. Thus p is nothing but utilization of the server.
2. **Average number of tasks queued in the system**: It includes the number of tasks waiting for service and the number of tasks actually being served. It is also known as mean queue length. Let E(N) be the average number of tasks in the system. Then,

$$E(N) = \sum_{n=0}^{\infty} n \cdot P_n \qquad \dots \text{(i)}$$

Where 13,, is the probability that there are n tasks in the system. It is given by

$$P_n = (1-\rho)\rho^n$$

Substituting in equation (i) gives,

$$\begin{aligned} E(N) &= \sum_{n=0} n(1-\rho)\rho^n = (1-\rho)\rho(1+2\rho+3\rho^2+4\rho^3+\cdots) \\ &= (1-\rho)\cdot\rho\frac{1}{(1-\rho)^2} = \frac{\rho}{1-\rho} = \frac{\lambda/\mu}{1-\lambda/\mu} \qquad \dots (\because \rho = \lambda/\mu) \\ &= \frac{\lambda}{\mu-\lambda} \qquad \dots \text{(ii)} \end{aligned}$$

3. **Average time that tasks spend in the system:** It involves waiting time in queue and actual service time. This is also called Average response time or mean waiting time. Let E (V) be the average time that tasks spend in the system. The quantities E (V) and E(N) are related directly. When average number of tasks are E(N) and tasks enter the system at rate X, then we can write,

$$E(V) = E(N)/\lambda \qquad \dots \text{(iii)}$$

Combining equations (ii) and (iii), we get

$$E(V) = \frac{\lambda}{\mu-\lambda} \times \frac{1}{\lambda}$$

$$E(V) = \frac{1}{\mu-\lambda}$$

4. Average time spent waiting in the queue excluding service time : Let it beE(W).

$$E(W) = E(V) - \frac{1}{\mu}$$

where1/ is the average time required to service at ask.

$$\therefore \qquad E(W) = \frac{1}{\mu-\lambda} - \frac{1}{\mu}$$

$$\therefore \qquad E(W) = \frac{\lambda}{\mu(\mu-\lambda)} \qquad \text{... (iv)}$$

5. Average number of tasks waiting in the queue excluding those being served is denoted by E(Q). The average number of tasks being serviced is X,. Hence subtracting this from E(N) yields E(Q).

$$\therefore \qquad E(Q) = E(N) - \rho = \frac{\lambda}{\mu-\lambda} - \frac{\lambda}{\mu}$$

$$= \frac{\lambda^2}{\mu(\mu-\lambda)} \qquad \text{... (v)}$$

From equation (iv) and (v), we get

$$E(W) = \frac{E(Q)}{\lambda}$$

CPU ORGANIZATION:

Central processing unit (CPU) is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions. In the computer all the all the major components are connected with the help of the system bus. Data bus is used to shuffle data between the various components in a computer system. To differentiate memory locations and I/O devices the system designer assigns a unique memory address to each memory element and I/O device. When the software wants to access some particular memory location or I/O device it places the corresponding address on the address bus. Circuitry associated with the memory or I/O device recognizes this address and instructs the memory or I/O device to read the data from or place data on the data bus. Only the device whose address matches the value on the address bus responds.

The control bus is an eclectic collection of signals that control how the processor communicates with the rest of the system. The read and write control lines control the direction of data on the data bus. When both contain logic one the CPU and memory-I/O are not communicating with one another. If the read line is low (logic zero) the CPU is reading data from memory (that is the system is transferring data from memory to the CPU). If the write line is low the system transfers data from the CPU to memory. The CPU controls the computer. It fetches instructions from memory, supply the address and control signals needed by the memory to access its data. Internally, CPU has three sections as shown in the fig below

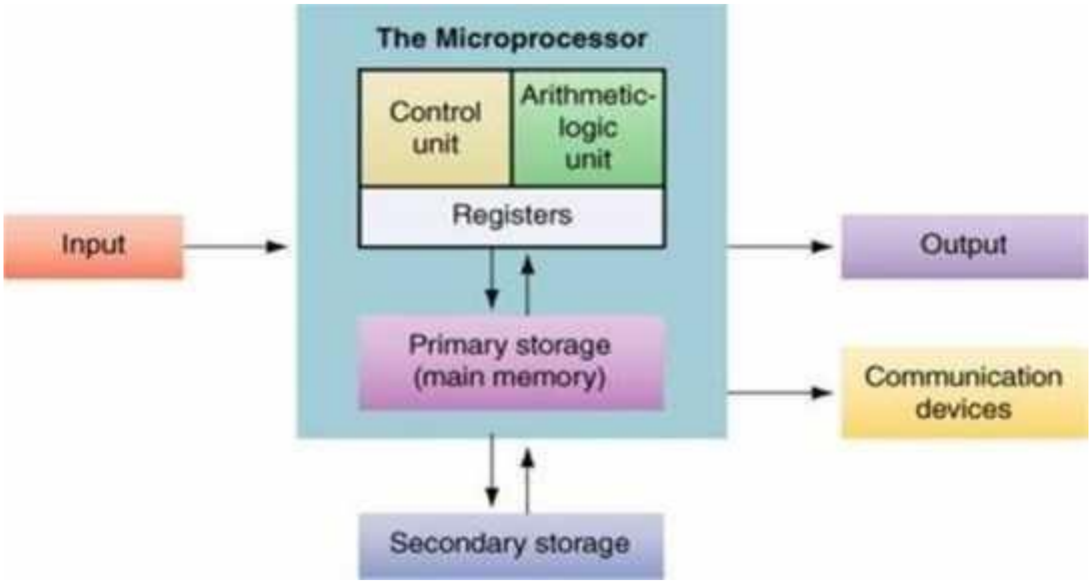


Fig 1.3: CPU Organization

1. The register section, as its name implies, includes a set of registers and a bus or other communication mechanism.
2. The register in a processor's instruction set architecture are found in the section of the CPU.
3. The system address and data buses interact with this section of CPU. The register section also contains other registers that are not directly accessible bytheprogrammer.
4. The fetch portion of the instruction cycle, the processor first outputs the address of the instruction onto the address bus. The processor has a register called the"**programcounter**".
5. The CPU keeps the address of the next instruction to be fetched in this register. Before the CPU outputs the address on to the system bus, it retrieves the address from the program counter register.
6. At the end of the instruction fetch, the CPU reads the instruction code from the system data bus.
7. It stores this value in an internal register, usually called the"**instructionregister**".
8. The arithmetic / logic unit (or) ALU performs most arithmetic and logic operations such as adding and ANDing values. It receives its operands form the register section of the CPU and stores its result back in the registersection.
9. Just as CPU controls the computer, the control unit controls the CPU. The control unit receives some data values from the register unit, which it used to generate the control signals. This code generates the instruction codes & the values of some flag registers.
10. The control unit also generates the signals for the system control bus such as READ,WRITE,IO/M signals.

DATA REPRESENTATION

DATA TYPES

The data types found in the registers of digital computers may be classified as being one of the following categories:

- numbers used in arithmetic computations,
- Letters of the alphabet used in data processing.
- Other discrete symbols used for specific purposes.

- All types of data, except binary numbers, are represented in computer registers in binary form. This is because registers are made up of flip-flops and flip-flops are two-state devises that can store only l's and O's.
- The binary number system is the most natural system to use in a digital computer. But sometimes it is convenient to employ different number systems.

Number systems

- A number system of base, or radix, r is a system that uses distinct symbols for r digits. Numbers are represented by a string of digit symbols.
- To determine the quantity that the number represents, it is necessary to multiply each digit by an integer power of r and then form the sum of all weighted digits. For example, the decimal number system in everyday use employs the radix 10system.
- The 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. The string of digits.
- 724.5 is interpreted to represent the quantity.
- 7 X 102 + 2 X 101 + 4 X 10° + 5 X10-1

The binary number system uses the radix 2. The two digit symbols used are 0 and 1. The string of digits 101101 is interpreted to represent the quantity

$$1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 45$$

To distinguish between different radix numbers, the digits will be enclosed in parentheses and the radix of the number inserted as a subscript. For example, to show the equality between decimal and binary forty-five we will write $(101101)_2 = (45)_{10}$.

The octal (radix 8) and hexadecimal (radix 16) are important in digital computer work. The eight symbols of the octal system are 0, 1, 2, 3, 4, 5, 6, and 7. The 16 symbols of the hexadecimal system are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, 0, E, and F. When used to represent hexadecimal digits, the symbols A, B, C, D, E, F correspond to the decimal numbers 10, 11, 12, 13, 14, 15, respectively.

Octal 736.4 is converted to decimal as follows:

$$\begin{aligned}(736.4)_8 &= 7 \times 8^2 + 3 \times 8^1 + 6 \times 8^0 + 4 \times 8^{-1} \\ &= 7 \times 64 + 3 \times 8 + 6 \times 1 + 4/8 = (478.5)_{10}\end{aligned}$$

The equivalent decimal number of hexadecimal F3 is obtained from the following calculation:

$$(F3)_{16} = F \times 16 + 3 = 15 \times 16 + 3 = (243)_{10}$$

Binary Numeral System - Base-2

Binary numbers uses only 0 and 1 digits.

B denotes binary prefix.

Examples:

101012 = 10101B = 1×24+0×23+1×22+0×21+1×20 = 16+4+1= 21

101112 = 10111B = 1×24+0×23+1×22+1×21+1×20 = 16+4+2+1= 23

1000112 = 100011B = 1×25+0×24+0×23+0×22+1×21+1×20 =32+2+1= 35

Octal Numeral System - Base-8

Octal numbers uses digits from 0..7.

Examples:

278 = 2×81+7×80 = 16+7 = 23

308 = 3×81+0×80 = 24

43078 = 4×83+3×82+0×81+7×80= 2247

Decimal Numeral System - Base-10

Decimal numbers uses digits from 0...9.

These are the regular numbers that we use.

Example:

253810 = 2×103+5×102+3×101+8×100

Hexadecimal Numeral System - Base-16

Hex numbers uses digits from 0...9 and

A...F. H denotes hex prefix.

Examples:

2816 = 28H = 2×161+8×160 = 40

2F16 = 2FH = 2×161+15×160 = 47

BC1216 = BC12H = 11×163+12×162+1×161+2×160= 48146

TYPES OF INSTRUCTIONS

The basic computer has three 16-bit instruction code formats:

1. Memory Reference Instructions
2. Register Reference Instructions
3. Input/output Instructions

MEMORY REFERENCE INSTRUCTIONS

In Memory reference instruction: First 12 bits (0-11) specify an address.

- Next 3 bits specify operation code (oppose).
- Left most bit specify the addressing mode I
- I = 0 for direct address I = 1 for indirect address

Memory Reference Instructions

In Memory reference instruction: first 12 bits (0-11) specify an address.

- The address field is denoted by three axes (in hexadecimal□ notation) and is equivalent to 12-bitaddress. The last mode bit of the instruction represents by symbol I.
- When I = 0, the last four bits of an instruction havea
- Hexadecimal digit equivalent from 0 to 6 since the last bit is zero (0). When I = 1 the last four bits of an instruction have a hexadecimal digit equivalent from 8 to E since the last bit is one (1).

Memory Reference Instructions

	Hexadecimal code		
Symbol	I = 0	I = 1	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	ADD memory word to AC
LDA	2xxx	Axe	LOAD Memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Cxxx	Branch and save return address
ISZ	6xxx	Cxxx	Increment and Skip if zero

Register Reference Instructions

In Register Reference Instruction: First 12 bits (0-11) specify the register operation.

- The next three bits equals to 111 specifyopcode.
- The last mode bit of the instruction is0.
- Therefore, left most 4 bits are always 0111 which is equal to hexadecimal7.

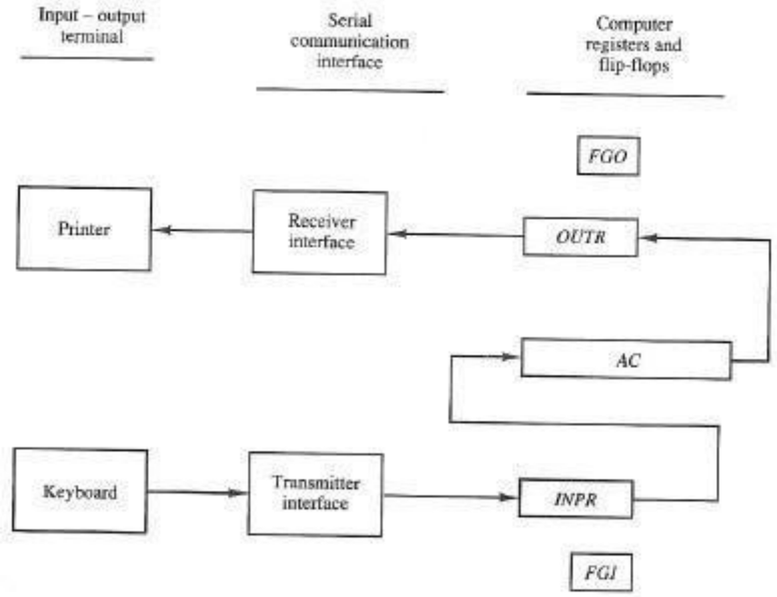
I/O Reference Instructions

In I/O Reference Instruction: First 12 bits (0-11) specify the I/O operation.

- The next three bits equals to 111 specifyopcode.
- The last mode bit of the instruction is1.
- Therefore, left most 4 bits are always 1111 which is equal to hexadecimal.

- Acomputerisuselessunless theinformationiscommunicationbetweenI/Oterminalsandtheprocessor. In a computer, instructions and data stored in memory come from some input device and computational results must be transmitted to the user through some output device.

- The terminal sends and receives serial information. Each quantity of information has 8 bits of an alphanumeric code. Two basic computer resisters INPR and OUTR communicate with acommunication interface.



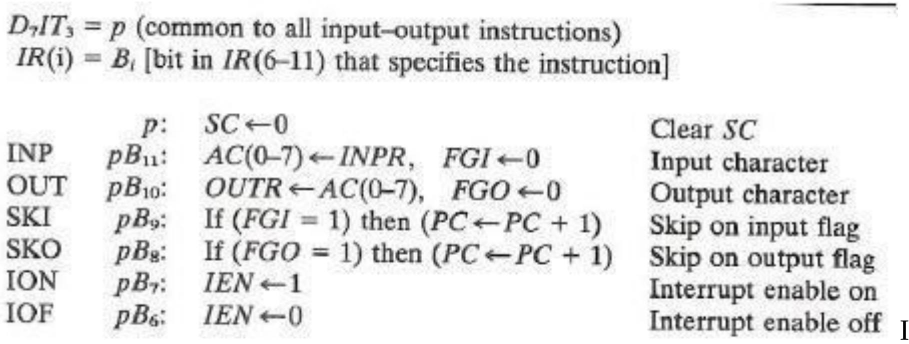
Input-Output Configuration

- The terminals send and receive serial information the 8-bits information is shifted from the keyboard into the input register (INPR). The serial information for the printer is stored in OUTR. These two registers communicateinthecommunicationinterfaceseriallyandwiththeaccumulatorinparallel.
- Thetransmitterinterfacereceives theinformationandtransfertoINPR.Thereceiverinterfacereceives from OUTR and transmits to the printer.
- The input register INPR consists of 8-bits and stores alphanumeric information. FGI is 1-bit controlled flip-flop or flag. A flag bit is set to1.
- When new information is available in input device and cleared to zero and the information is accepted by the computer. Initially, the input flag FGI clear to zero. When a key in a keyboard is pressed,8-bits

alphanumeric code is shifted into INPR is transferred into accumulator and FGI is cleared to zero. The output flag FGO is initially set to one the computer checks FGO and if it is one (1) then the content of accumulator(AC) is transferred into OUTR and FGO is cleared to zero. Once the content of OUTR is transmitted to the printer FGO is again set to 1. Therefore, if FGO=1, then the data from input device can't be transferred into INPR and similarly, if FGO=0, then the content of AC can't be transferred into OUTR.

Input-Output Instruction

I/O instructions are needed to transferring information to and from AC register, for checking the flag bits and for controlling the interrupt facility.



$D_7IT_3 = p$ (common to all input–output instructions)
$IR(i) = B_i$ [bit in $IR(6\text{–}11)$ that specifies the instruction]

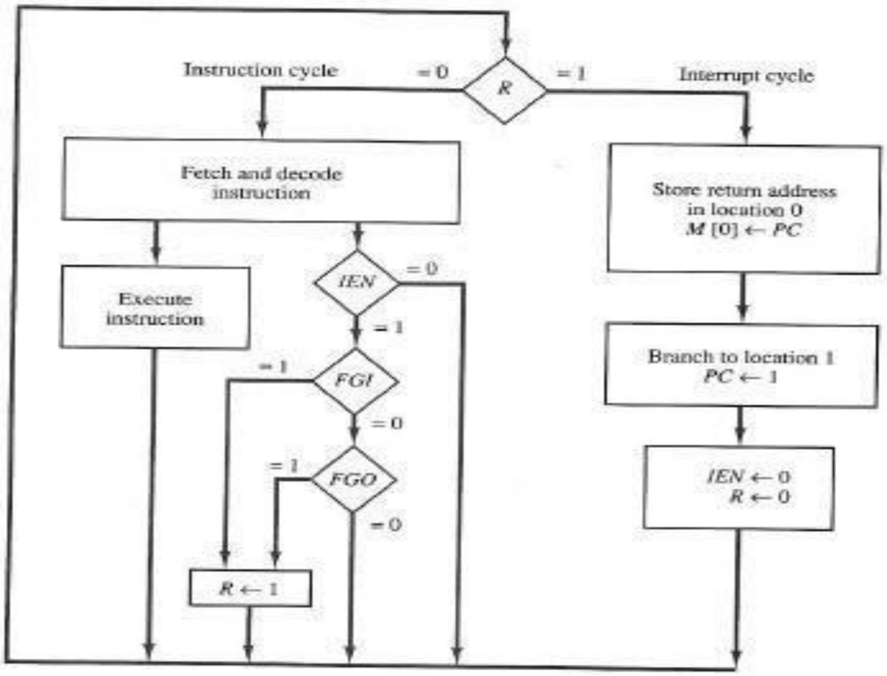
	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0\text{–}7) \leftarrow INPR, \quad FGI \leftarrow 0$	Input character
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{–}7), \quad FGO \leftarrow 0$	Output character
SKI	pB_9:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$	Skip on input flag
SKO	pB_8:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$	Skip on output flag
ION	pB_7:	$IEN \leftarrow 1$	Interrupt enable on
IOF	pB_6:	$IEN \leftarrow 0$	Interrupt enable off

I/O instruction

Program Interrupt

- InputandOutputinteractionswithelectromechanicalperipheraldevicesrequirehugeprocessingtimes compared with CPU processing times. – I/O (milliseconds) versus CPU(nano/microseconds)
- Interrupts permit other CPU instructions to execute while waiting for I/O to complete.
- The I/O interface, instead of the CPU, monitors the I/Odevice.
- WhentheinterfacefoundthattheI/Odeviceis readyfordatatransfer,itgenerates aninterruptrequestto theCPU
- Upon detecting an interrupt, the CPU stops momentarily the task it is doing, branches to the service routine to process the data transfer, and then returns to the task it was performing.

Interrupt Cycle
This is a hardware implementation of a branch and save return address operation.



Flowchart of interrupt cycle

- An alternative to the program controlled procedure is to let the internal device inform the computer when it is ready for data transfer. At The same time, the processor can execute other tasks. This method uses the interrupt facility when the computer is w=executing a program, it doesn't check the flags. But, when the flag is set , the computer is momentarily interrupted from executing the current program and is informed that the flag has been set.
- In this case, the computer momentarily deviates from what it is executing currently and starts the I/Transfers. Once the I/O transfer is complete, the computer returns back to its original job.
- The interrupt enables flip-flop IEN can be set and cleared with two instructions. When IEN is cleared to zero with IOF instruction. Then, the flag can't interrupt the computer, when IEN is set to 1 with ION instruction the computer can be interrupted.

Register transfer operation in interrupt cycle

Register Transfer Statements for Interrupt Cycle

-RF/F←1ifIEN(FGI+FGO)T0’T1’T2’↔T0’T1’T2’(IEN)(FGI+FGO):R←1

The fetch and decode phases of the instruction cycle must be modified: Replace T0,T1,T2 with R'T0,R'T1,R'T2

The interrupt cycle: RT0:*AR←0,TR←PC*

RT1:M[AR]←TR,PC←0

RT2:PC←PC+1,IEN←0,R←0,SC← 0

Complete Computer Description Flowchart

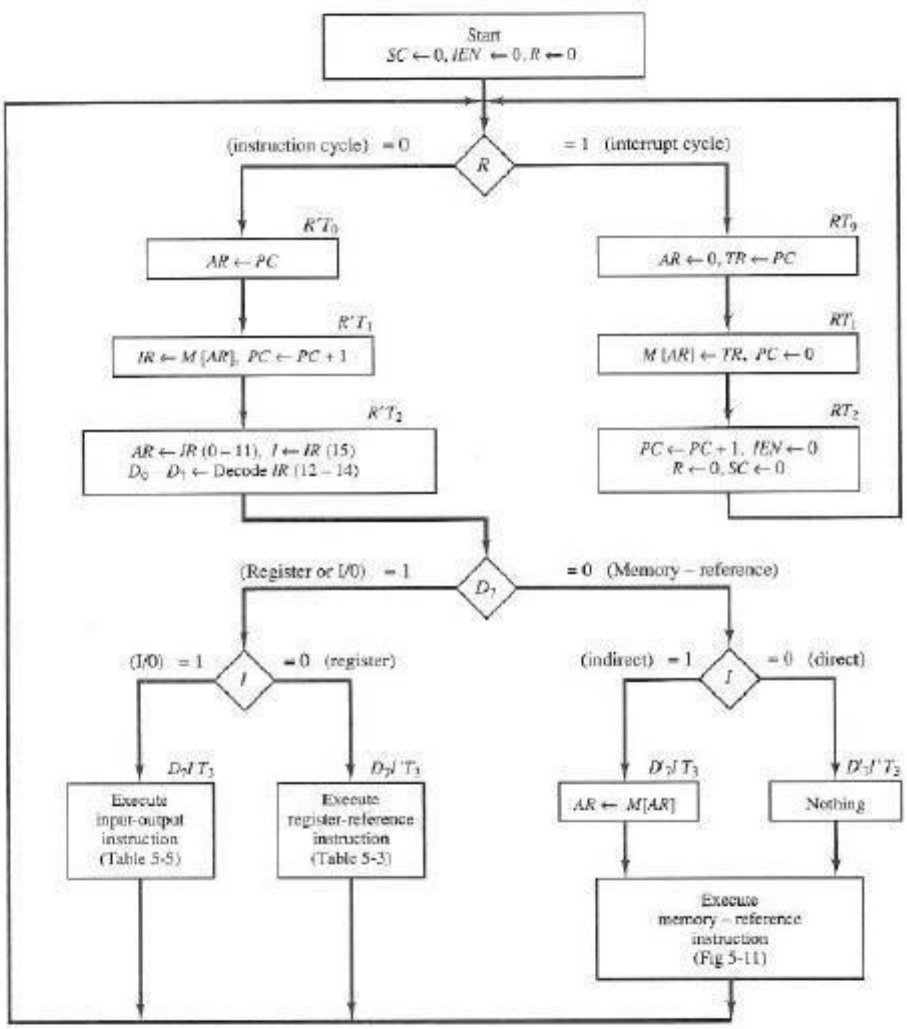
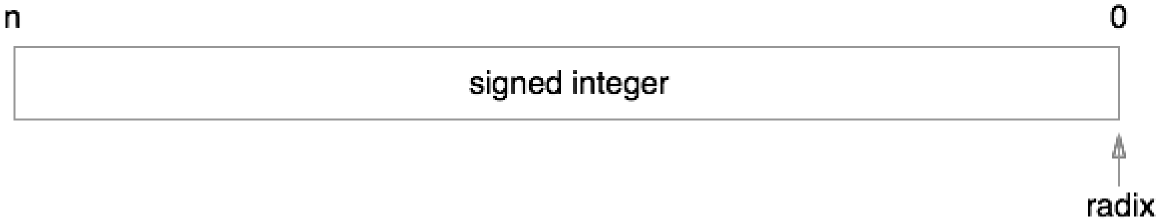


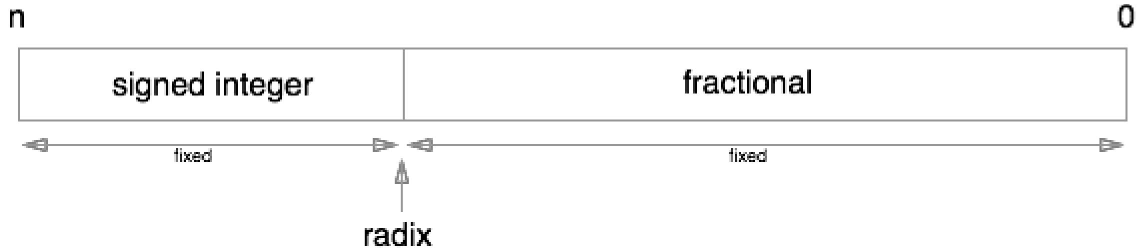
fig. Instruction cycle including Interrupt cycle

Fixed Point Representation

Fixed Point Notation is a representation of our fractional number as it is stored in memory. In Fixed Point Notation, the number is stored as a signed integer in two's complement format.



On top of this, we apply a notional split, locating the radix point (the separator between integer and fractional parts) a fixed number of bits to the left of its notational starting position to the right of the least significant bit. I've illustrated this in the diagram below.



When we interpret the bits of the signed integer stored in memory we reposition the radix point by multiplying the stored integer by a fixed scaling factor. The scaling factor in binary is always 2 raised to a fixed exponent. As the scaling factor is a power of 2 it relocates the radix point some number of places to the left or right of its starting position.

During this conversion there are three directions that the radix point can be moved:

- **The radix point is moved to the right:** This is represented by a scaling factor whose exponent is 1 or more. In this case additional zeros are appended to the right of the least-significant bit and means that the actual number being represented is larger than the binary integer that was stored.
- **The radix point remains where it is:** This is represented by a scaling factor whose exponent is 0 and means that the integer value stored is exactly the same as the integer value being represented.
- **The radix point is moved to the left:** This is represented by a scaling factor whose exponent is negative. This means that the number being represented is smaller than the integer number that was stored and means that the number being represented has a fractional component.

Let’s take a look at a couple of examples.

Examples of Fixed Point Numbers

Lets assume we have an 8-bit signed binary number 00011011_2 that is stored in memory using 8-bits of storage (hence the leading zeros).

In our first scenario, lets also assume this number was stored as a signed fixed-point representation with a scale factor of 2^2.

As our scale factor is greater than 1, when we translated the bits stored in memory into the number we are actually representing, we move the radix point two places to the right. This gives us the number: 1101100_2 (Note the additional zeros that are appended to the right of the least significant bit).

In our second scenario, let us assume that we start off with the same binary number in memory but this time we’ll assume that it is stored as a signed fixed-point representation with a scale factor of 2^{-3}. As the exponent is negative we move the radix point three places to the left. This gives us the number00011.011_2

Advantages and Disadvantages of Fixed Point Representation

The major advantage of using a fixed-point representation is performance. As the value stored in memory is an integer the CPU can take advantage of many of the optimizations that modern computers have to perform integer

Arithmetic without having torelyon additional hardware or software logic. This in turn can lead to increases in performance and when writing your apps, can therefore lead to an improved experience for your users.

However, there is a downside! Fixed Point Representations have a relatively limited range of values that they can represent.

So how do we work out the maximum and minimum numbers that can be stored in a fixed-point representation and determine whether it is suitable for our needs? All we do is take the largest and smallest integer values that can be stored in the given number of bits and multiply that by the scale factor associated with our fixed-point representation. For a given signed binary number using b bits of storage with a scale factor of f the maximum and minimum values that can be store dare:

Minimum: $-2^{b-1}/2^{f}$

Maximum: $(2^{b-1}-1)/2^{f}$

If the number you want to represent fits into this range then things are great. If it doesn't though, you have to look for an alternative! This is where Floating Point Notation comes in.

Floating Point Notation

Floating Point Notation is an alternative to the Fixed Point notation and is the representation that most modern computers use when storing fractional numbers in memory. Floating Point Notation is a way to represent very large or very small numbers precisely using scientific notation in binary. In doing so, Floating Point Representation provides a varying degrees of precision depending on the scale of the numbers that you are using. For example, the level of precision we need when we are talking about the distance between atoms (10^{-10} m) is very different from the precision we need when we're talking about the distance between the earth and the sun (10^{11} m). This is a major benefit and allows a much wider range of numbers to be represented than is possible in Fixed Point Notation.

Floating Point Representation is based on **Scientific Notation**. You may have used Scientific Notation in school. When we use Scientific Notation in decimal (the form you're probably most familiar with), we write numbers in the following form:

$$+/- \text{ mantissa } \times 10^{\text{exponent}}$$

In this form, there is an optional sign indicating whether the overall number is positive or negative, followed by a mantissa (also known as a significant) which is a real (fractional) number which in turn is multiplied by a number base (or radix) raised by an exponent. As we know, in decimal this number base is 10.

Floating Point Representation is essentially Scientific Notation applied to binary numbers. In binary, the only real difference is that the number base is 2 instead of 10. We would therefore write Floating Point Numbers in the following form:

+/- mantissa x 2^{exponent}

Now, you may not have realized it but when we write numbers in scientific notation (whether they be binary or decimal) we can write them in a number of different ways.

In decimal we could write 1.5 x 10^2, 15 x 10^1 and 150 x 10^0 and yet all these numbers have exactly the same value.

This provides flexibility but with this flexibility also comes confusion. To try and address this confusion a common set of rules known as normalized scientific notation are used to define how numbers in scientific notation are normally written.

Normalized Scientific Notation

Normalized Scientific Notation is a nomenclature that standardizes the way we write numbers in scientific notation. In the normalized form we have a single key rule:

"We choose an exponent so that the absolute value of the mantissa remains greater than or equal to 1 but less than the number base."

Let's look at a couple of examples!

If we had the decimal number 50010 and wanted to write it in scientific notation we could write it as either 500 x 10^0 or 50 x 10^1.

In normalized form though, we would apply the rule above and move the radix point so that only a single digit, greater than or equal to 1 and less than (in this case) 10 were to the left of the radix point.

In this case this would mean moving our radix point two places to the left so we had 5.0 x $10^?$.

We would then need to work out our exponent. To get back to our original number we would need to move our radix point two places to the right. Remember what we learnt earlier? If we have to move our radix point to the right to get back to our original number that means the exponent is positive. This gives us: 5.0 x 10^2.

Lets look at a slightly more complicated example, this time in binary.

What if we had the binary number 10.1_2? What would this be in scientific notation? Again we apply the rules: We need to have a mantissa that is greater than or equal to 1 and less than our number base (which this time is 2).

That would mean our mantissa would need to be 1.01 x $2^?$. To get back to our original number we would need to move our radix point 1 place to the right. What does right mean? That means the exponent is positive.

Last example. This time one that is a little more tricky!

Imagine I had the number 0.111_2 and wanted to write it in normalized scientific notation? Again, we apply the rules. We need a mantissa greater than or equal to 1 and less than 2.

That means we want to write our mantissa as 1.11 x $2^?$.

Now, to get back to our original number we would need to move our radix point 1 place to the… left. What did we learn about moving to the left? That means our exponent is negative. That gives us: 1.11 x 2^{-1}.

Instruction formats:

- The basic computer has three instruction code formats, as shown in Fig below. Each format has 16 bits. The operation code (opcode) part of the instruction contains three bits and the meaning of the remaining 13 bits depends on the operation code encountered.
- A memory reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode I. I is equal to 0 for direct address and to 1 for indirect address.
- The register-reference instructions are recognized by the operation code 111 with a 0 in the leftmost bit (bit 15) of the instruction. A register-reference instruction specifies an Operation on or a test of the AC register. An operand from memory is not needed therefore the other 12 bits are used to specify the operation or test to be executed.
- Similarly, an input-output instruction does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction. The remaining12 bits are used to specify the type of input-output operation or test performed. The type of instruction is recognized by the computer control from the four bits in positions 12 through 15 of the instruction. If the three op code bits in positions 12 through 14 are not equal to 111, the instruction is a memory-reference type and the bit in position 15 is taken as the addressing mode I. If the 3-bit op code is equal to 111, control then inspects the bit in position15. If this bit is 0, the instruction is a register-reference type. If the bit is 1, the instruction is an I/O type.

- The instruction for the computer is shown the table below. The symbol designation is a three letter word and represents an abbreviation intended for programmers and users. The hexa decimal code is equal to the equivalent hexadecimal number of the binary code used for the instruction. By using the hexadecimal equivalent we reduced the 16 bits of an instruction code to four digits with each hexadecimal digit being equivalent to four bits.
- A memory-reference instruction has an address part of 12 bits. The address part is denoted by three x's and stand for the three hexadecimal digits corresponding to the 12-bit address. The last bit of the instruction is designated by the symbol I. When I = 0, the last four bits of an instruction have a hexadecimal digit equivalent from 0 to 6 since the last bit is 0. When I = 1, the hexadecimal digit equivalent of the last four bits of the instruction ranges from 8 to E since the last bit is I.
- Register-reference instructions use 16 bits to specify an operation. The leftmost four bits are always 0111, which is equivalent to hexadecimal 7. The other three hexadecimal digits give the binary equivalent of the remaining 12bits.
- The input-output instructions also use all 16 bits to specify an operation. The last four bits are always 1111, equivalent to hexadecimal F.

Symb	Hexadecimal code		Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	ADD memory word to AC
LDA	2xxx	Axx	Load memory word to AC
STA	3xxx	Bxx	Store content of AC in memory
BUN	4xxx	Cxx	Branch Unconditionally
BSA	5xxx	Dxx	Branch and save return address
ISZ	6xxx	Exx	Increment & skip if skip
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC & E
CIL	7040		Circulate left AC & E
INC	7020		Increment AC
SPA	7010		Skip next address if AC is +ve
SNA	7008		Skip next address if AC is -ve
SZA	7004		Skip next address if AC is zero
SZE	7002		Skip next address if E is zero
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

Fig: Basic computer instructions

Instruction types:

Assembly languages instructions are grouped together based on the operation they performed.

- Data transfer instructions
- Data operational instructions
- Program control instructions Data transfer instructions:
- **Load the data from memory into the microprocessor**: These instructions copy data from memory into a microprocessor register
- **Store the data from the microprocessor into the memory**: This is similar to the load data expect data is copied in the opposite direction from a microprocessor register to memory.
- **Move data within the microprocessor**: These operations copies data from one microprocessor register to another. **Input the data to the microprocessor**: The microprocessor inputs the data from the input devices ex: keyboard in to one of its registers.

- **Output the data from the microprocessor**: The microprocessor copies the data from one of the registers to an input device such as digital display of a microwave oven.

1) **Data operational instructions:**

- Dataoperationalinstructionsdomodifytheirdatavalues.Theytypicallyperformsomeoperationsusing one or two data values (operands) and store result.
- Arithmetic instructions make up a large part of data operations instructions. Instructions that add, subtract, multiply, or divide values fall into this category. An instruction that increment or decrement also falls in to this category.
- Logical instructions perform basic logical operations on data. They AND, OR, or XOR two data values or complement a single value.
- Shift operations as their name implies shift the bits of a data values also comes under this category.

2) **Program control instructions:**

- Program control instructions are used to control the flow of a program. Assembly language instructions may include subroutines like in high level language program may have subroutines, procedures, and functions.
- A jump or branch instructions are generally used to go to another part of the program or subroutine.
- A microprocessor can be designed to accept interrupts. An interrupt causes the processor to stop what is doing and start other instructions. Interrupts may be software or hardware.
- One final type of control instructions is halt instruction. This instruction causes a processor to stop executing instructions such as end of a program.

ADDRESSING MODES

Addressing mode is the way of addressing a memory location in instruction. Microcontroller needs data or operands on which the operation is to be performed. The method of specifying source of operand and output of result in an instruction is known as addressing mode.

There are various methods of giving source and destination address in instruction, thus there are various types of Addressing Modes. Here you will find the different types of Addressing Modes that are supported in Micro Controller 8051. Types of Addressing Modes are explained below:

Also Read: Introduction to Microcontroller 8051

Types Of Addressing Modes:

Following are the types of Addressing Modes:

- Register Addressing Mode
- Direct Addressing Mode
- Register Indirect Addressing Mode
- Immediate Addressing Mode
- Index Addressing Mode
- Also Read: How Microcontroller Works

Explanation:

- **Register Addressing Mode:** In this addressing mode, the source of data or destination of result is Register. In this type of addressing mode the name of the register is given in the instruction where the data to be read or result is to be stored.

Example: ADD A, R5 (The instruction will do the addition of data in Accumulator with data in register R5)

For example,

```
MOV DX, TAX_RATE ; Register in first operand

MOVCOUNT, CX         ; Register in second operand

MOVEAX,EBX           ; Both the operands are in registers
```

- **Direct Addressing Mode:** In this type of Addressing Mode, the address of data to be read is directly given in the instruction. In case, for storing result the address given in instruction is used to store the result.

Example: MOV A, 46H (This instruction will move the contents of memory location 46H to Accumulator)

For example,

```
BYTE_VALUEDB150         ; A byte value is defined

WORD_VALUEDW300         ; A word value isdefined

ADD BYTE_VALUE,65       ; An immediate operand 65 is added

MOV AX,45H              ; Immediate constant 45H is transferred to AX
```

- **Mode Register Indirect Addressing:** In Register Indirect Addressing Mode, as its name suggests the data is read or stored in register indirectly. That is, we provide the register in the instruction, in which the address of the other register is stored or which points to other register where data is stored or to be stored.

Example: MOV A, @R0 (This instruction will move the data to accumulator from the register whose address is stored in register R0).

Also Read: Architecture of 8051

- **Immediate Addressing Mode:** In Immediate Addressing Mode, the data immediately follows the instruction. This means that data to be used is already given in the instruction itself.

Example: MOV A, #25H (This instruction will move the data 25H to Accumulator. The #sign show that preceding term is data, not the address.)

- **Index Addressing Mode:** Offset is added to the base index register to form the effective address if the memory location. This Addressing Mode is used for reading lookup tables in Program Memory. The AddressoftheexactlocationofthetableisformedbyaddingtheAccumulatorDatatothebasepointer.

Example: MOVC, @A+DPTR (This instruction will move the data from the memory to Accumulator; the address is made by adding the contents of Accumulator and Data Pointer.

Floating Point Arithmetic

Floating point arithmetic derives its name from something that happens when you use exponential notation. Consider the number 123: it can be written using exponential notation as:

1. $1.23 *10^2$
2. $12.3 *10^1$
3. $123 *10^0$
4. $.123 * 10^3$
5. $1230 * 10^{-1}$etc.

All of these representations of the number 123 are numerically equivalent. The differ only in their "**normalization**": where the decimal point appears in the first number. In each case, the number before the multiplication operator ("*") represents the significant figures in the number (which distinguish it from other numbers with the same normalization and exponent); we will call this number the "significand" (also called the "mantissa" in other texts, which call the exponent the "characteristic").
Notice how the decimal point "floats" within the number as the exponent is changed. This phenomenon gives floating point numbers their name. Only two of the representations of the number 123 above are in any kind of standard form. The first representation, $1.23 * 10^2$, is in a form called "scientific notation", and is distinguished by the normalization of the significand: in scientific notation, the significand is always a number greater than or equal to 1 and less than 10. Standard computer normalization for floating point numbers follows the fourth form in the list above: the significand is greater than or equal to .1, and is always less than1.
Of course, in a binary computer, all numbers are stored in base 2 instead of base 10; for this reason, the normalization of a binary floating point number simply requires that there be no leading zeroes after the binary point (just as the decimal point separates the 10^0 place from the 10^{-1} place, the binary point separates the 2^0 place from the 2^{-1} place). We will continue to use the decimal number system for our numerical examples, but the impact of the computer's use of the binary number system will be felt as we discuss the way those numbers are stored in the computer.

Floating Point Formats:

Over the years, floating point formats in computers have not exactly been standardized. While the IEEE (Institute of Electrical and Electronics Engineers) has developed standards in this area, they have not been universally adopted. This is due in large part to the issue of "backwards compatibility": when a hardware manufacturer designs a new computer chip, they usually design it so that programs which ran on their old chips will continue to run in the same way on the new one. Since there was no standardization in floating point formats when the first floating point processing chips (often called "coprocessors" or "FPU"s: "Floating Point Units") were designed, there was no rush among computer designers to conform to the IEEE floating point standards (although the situation has improved with time).

For this reason, we will discuss both the IEEE standards as well as the floating point formats implemented in the very common Intel chips (such as the 80387, 80486 and the Pentium series). Each of these formats has a name like "single precision" or "double precision", and specifies the numbers of bits which are used to store both the exponent and the significand. We will defined the notion of "precision" in the following way: if the significand is stored in n bits, it can represent a decimal number between 0 and $2^n - 1$ (since a significand is stored as an unsigned integer). If we find the largest number "m" such that $10^m - 1$ is less than or equal to $2^n - 1$, m will be the precision. Consider the following:

$2^4 - 1 = 15$	$10^1 - 1 = 9$
$2^8 - 1 = 255$	$10^2 - 1 = 99$
$2^{12} - 1 = 4{,}095$	$10^3 - 1 = 999$
$2^{16} - 1 = 65{,}535$	$10^4 - 1 = 9{,}999$
$2^{20} - 1 = 1{,}048{,}575$	$10^6 - 1 = 999{,}999$

From the last example, it is easy to see that a 20 bit significand provides just over 6 decimal digits of precision. In the other examples, there is more precision than we have indicated. For example, a 16 bit significand is certainly sufficient to represent many decimal numbers with more than 4 digits; however, not all 5 digit decimal numbers can be represented in 16 bits, and so the precision of a 16 bit significand is said to be "> 4" (but less than 5). Some texts attempt to more accurately describe the precision using fractions, but we do not feel the need to do so.

The following table describes the IEEE standard formats as well as those used in common Intel processors:

Precision	**Sign**	**Exponent**	**Significand**	**Total Length**	**Decimal digits**
	(# of bits)	**(# of bits)**	**(# of bits)**	**(in bits)**	**of precision**
IEEE / Intel single	1	8	23	32	> 6
IEEE single extended	1	>= 11	>= 32	>= 44	> 9
IEEE / Intel double	1	11	52	64	> 15
IEEE double extended	1	>= 15	>= 64	>= 80	> 19
Intel internal	1	15	64	80	> 19

UNIT-II
DATA PATH DESIGN

Fixed point arithmetic:

Floating point (FP) representations of decimal numbers are essential to scientific computation using *scientific notation*. The standard for floating point representation is the IEEE 754 Standard. In a computer, there is a tradeoff between range and precision - given a fixed number of binary digits (bits), precision can vary inversely with range. In this section, we overview decimal to FP conversion, MIPS FP instructions, and how registers are used for FP computations.

We have seen that an n-bit register can represent unsigned integers in the range 0 to 2^n-1, as well as signed integers in the range -2^{n-1} to -2^{n-1}-1. However, there are very large numbers (e.g., $3.15576 \cdot 10^{23}$), very small numbers (e.g., 10^{-25}), rational numbers with repeated digits (e.g., 2/3 = 0.666666...), irrationals such as $2^{1/2}$, and transcendental numbers such as e = 2.718..., all of which need to be represented in computers for scientific computation to be supported.

We call the manipulation of these types of numbers *floating point arithmetic* because the decimal point is not fixed (as for integers). In C, such variables are declared as the float data type.

Scientific Notation and FP Representation:

Scientific notation has the following configuration:

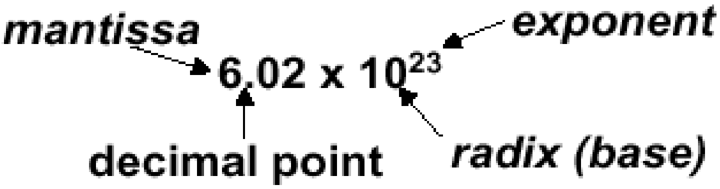


Figure – Scientific Notation

and can be in *normalized form* (mantissa has exactly one digit to the left of the decimal point, e.g., $2.3425 \cdot 10^{-19}$) or *non-normalized form*. Binary scientfic notation has the following configuration, which corresponds to the decimal forms.

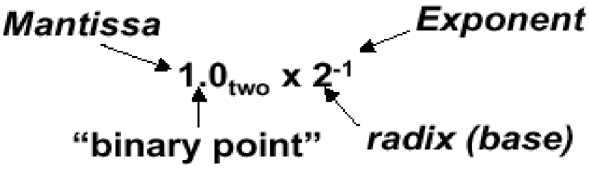


Figure – Binary Scientific Notation

Assume that we have the following *normal format* for scientific notation in Boolean numbers: $+1.xxxxxxx_2 \cdot w^{yyyyy}$,$_2$where "xxxxxxx" denotes the *significand* and "yyyyy" denotes the *exponent* and weassume that the number has sign S. This implies the following 32-bit representation for FP numbers:

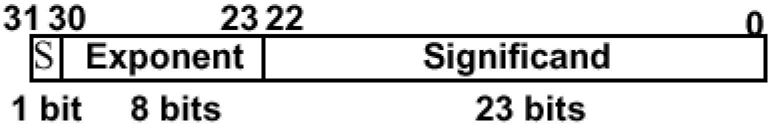


Figure – 32-Bit representation for FP numbers

Overflow and Underflow:

In FP, overflow and underflow are slightly different than in integer numbers. FP overflow (underflow) refers to the positive (negative) exponent being too large for the number of bits alloted to it. This problem can be somewhat ameliorated by the use of *double precision*, whose format is shown as follows:

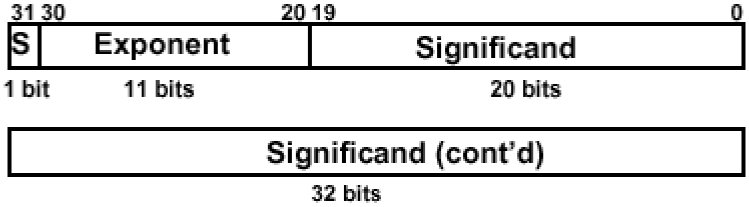


Figure – 32-Bit representation for FP numbers

Here, two 32-bit words are combined to support an 11-bit signed exponent and a 52-bit significand. This representation is declared in C using the double data type, and can support numbers with exponents ranging from - 308_{10} to 308_{10}. The primary advantage is greater precision in the mantissa. The following chart illustrates specific types of overflow and underflow encountered in standard FP representation:

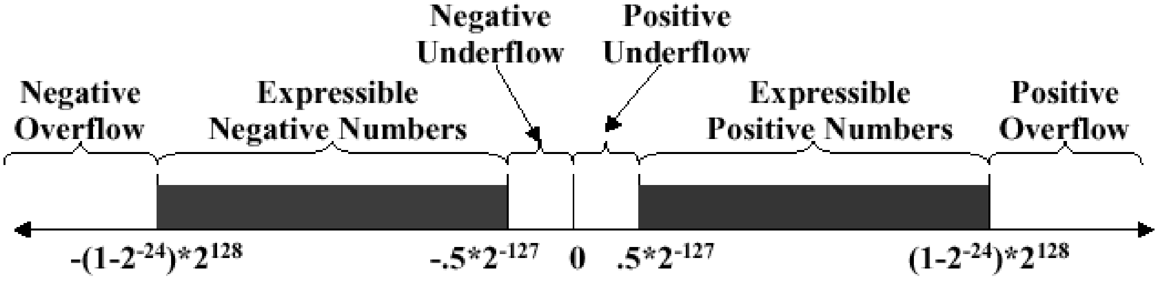


Figure – Standard FP representation

IEEE 754 Standard:

Both single- and double-precision FP representations are supported by the IEEE 754 Standard, which is used in the vast majority of computers since its publication in 1980. IEEE 754 facilitates the porting of FP programs, and ensures minimum standards of quality for FP computer arithmetic. The result is a signed representation - the sign bit is 1 if the FP number represented by IEEE754 is negative. Otherwise, the sign is zero. A leading value of 1 in the significand is implicit for normalized numbers.

Thus, the significand, which always has a value between zero and one, occupies 23 + 1 bits in single-precision FP and 52 + 1 bits in double precision. Zero is represented by a zero significand and a zero exponent - there is no leading value of one in the significand. The IEEE 754 representation is thus computed as:

FP number = $(-1)^S \cdot (1 + \text{Significand}) \cdot 2^{\text{Exponent}}$.

As a parenthetical note, the significand can be translated into decimal values via the following expansion:

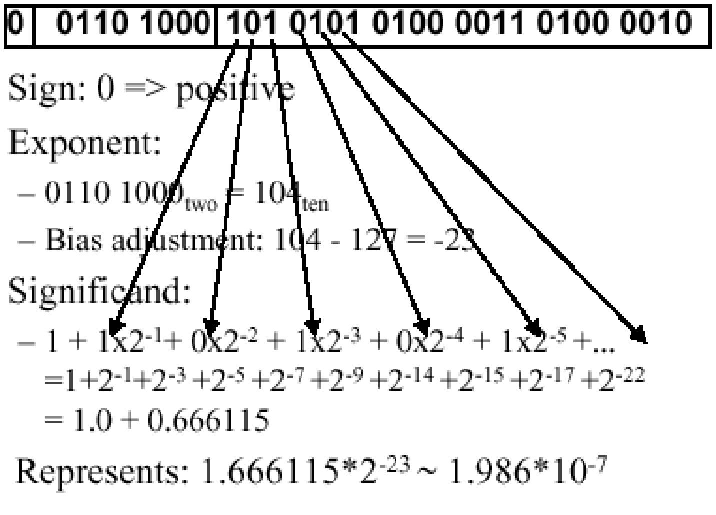
$$1.1001 = (1x2^{0}) + (1x2^{-1}) + (0x2^{-2}) + (0x2^{-3}) + (1x2^{-4})$$

With IEEE 754, it is possible to manipulate FP numbers without having special-purpose FP hardware. For example, consider the sorting of FP numbers. IEEE 754 facilitates breaking FP numbers up into three parts (sign, significant, exponent). The numbers to be sorted are ordered first according to sign (negative < positive), second according to exponent (larger exponent => larger number), and third according to significand (when one has at least two numbers with the same exponents).

Another issue of interest in IEEE 754 is *biased notation* for exponents. Observe that twos complement notation does not work for exponents: the largest negative (positive) exponent is $00000001_2(11111111_2)$. Thus, we must add a *bias term* to the exponent to center the range of exponents on the bias number, which is then equated to zero. The bias term is 127 (1023) for the IEEE 754 single-precision (double-precision) representation. This implies that FP number = $(-1)^S \cdot (1 + \text{Significand}) \cdot 2^{(\text{Exponent-Bias})}$.

As a result, we have the following example of binary to decimal floating point conversion:

Figure – Binary to Decimal floating point conversion



0	0110 1000	101 0101 0100 0011 0100 0010

- Sign: 0 => positive
- Exponent:
  – $0110\ 1000_{two} = 104_{ten}$
  – Bias adjustment: 104 - 127 = -23
- Significand:
  – $1 + 1x2^{-1} + 0x2^{-2} + 1x2^{-3} + 0x2^{-4} + 1x2^{-5} + \ldots$
  $= 1 + 2^{-1} + 2^{-3} + 2^{-5} + 2^{-7} + 2^{-9} + 2^{-14} + 2^{-15} + 2^{-17} + 2^{-22}$
  $= 1.0 + 0.666115$
- Represents: $1.666115*2^{-23} \sim 1.986*10^{-7}$

FP Arithmetic:

Applying mathematical operations to real numbers implies that some error will occur due to the floating point representation. This is due to the fact that FP addition and subtraction are not associative, because the FP representation is only an approximation to a real number.

Example 1. Using decimal numbers for clarity, let $x = -1.5 \cdot 10^{38}$, $y = 1.5 \cdot 10^{38}$, and $z = 1.0$. With floating point representation, we have:

$x + (y + z) = -1.5 \cdot 10^{38} + (1.5 \cdot 10^{38} + 1.0) = 0.0$

and

$(x + y) + z = (-1.5 \cdot 10^{38} + 1.5 \cdot 10^{38}) + 1.0 = 1.0$

The difference occurs because the value 1.0 cannot be distinguished in the significand of $1.5 \cdot 10^{38}$ due to insufficient precision (number of digits) of the significand in the FP representation of these numbers (IEEE 754 assumed).

The preceding example leads to several implementation issues in FP arithmetic. Firstly, *rounding* occurs when performing math on real numbers, due to lack of sufficient precision. For example, when multiplying two N-bit numbers, a 2N-bit product results. Since only the upper N bits of the 2N bit product are retained, the lower N bits are *truncated*. This is also called *rounding toward zero*.

Another type of rounding is called *rounding to infinity*. Here, if rounding toward +infinity, then we always round up. For example, 2.001 is rounded up to 3, -2.001 is rounded up to 2. Conversely, if rounding toward -infinity, then we always round down. For example, 1.999 is rounded down to 1, -1.999 is rounded down to -2. There is a more familiar technique, for example, where 3.7 is rounded to 4, and 3.1 is rounded to 3. In this case, we resolve rounding from *n*.5 to the nearest even number, e.g., 3.5 is rounded to 4, and -2.5 is rounded to2.

A second implementation issue in FP arithmetic is addition and subtraction of numbers that have nonzero significands and exponents. Unlike integer addition, we can't just add the significands. Instead, one must:

1. Denormalize the operands and shift one of the operands to make the exponents of both numbers equal (we denote the exponent by E).
2. Add or subtract the significands to get the resulting significand.
3. Normalize the resulting significand and change E to reflect any shifts incurred by normalization. Wewillreviewseveralapproachestofloatingpointoperationsin MIPSinthefollowingsection.

There are many different criteria's to check when considering the **"best"** scheduling algorithm, they are:

CPU Utilization:

To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time(Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.

Throughput:

It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

Turnaround Time:

It is the amount of time taken to execute a particular process, i.e. The interval from time of submission of the process to the time of completion of the process (Wall clock time).

Waiting Time:

The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

Load Average:

It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.

Response Time:

Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution(final response).

In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

Addition, Subtraction:

Arithmetic instructions in digital computers manipulate data to produce results necessary for the solutions of computational problems. These instructions perform arithmetic calculations and are responsible for the bulk of activity involved in processing data in a computer. The four basic arithmetic operations are addition, subtraction, multiplication and division.

From these four basic operations, it is possible to formulate other arithmetic functions and solve problems by means of numerical analysis methods. An arithmetic processor is the part of a processor unit that executes arithmetic operations.

An arithmetic instruction may specify binary or decimal data, and in each case, the data may be in fixed-point or floating point form. Negative numbers may be in signed magnitude or signed compliment representation. Fixed point numbers may represent integers or fractions. The addition and subtraction algorithm for data represented in signed magnitude and again data represented in signed-2’s complement. It is important to realize that the adopted representation for negative numbers refers to the representation of numbers in the register before and after the execution of the arithmetic operations.

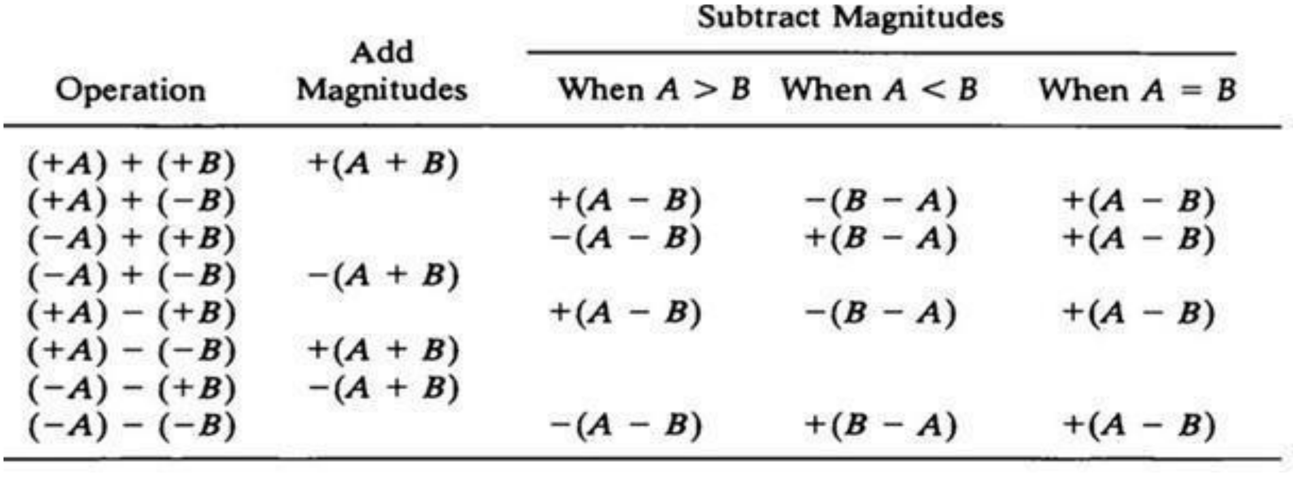
Addition and Subtraction with Signed-magnitude Data:

The representation of numbers in signed-magnitude is familiar because it is used in everyday arithmetic calculation. The procedure for adding or subtracting two signed binary numbers with paper and pencils simple and straight-forward. A review of this procedure will be helpful for deriving the hardware algorithm. We designated the magnitude of the two numbers by A and B. when the signed numbers are added or subtracted, we find that there are eight different conditions to consider, depending on the sign of the numbers and the operation performed. These conditions are listed in the first column of the table below. The other column in the table shows the actual operation to be performed with the magnitude of the numbers.

The last column is needed to prevent negative zero. In other words, when two equal numbers are subtracted, the result should be +0 not -0.

The algorithms for addition and subtraction are derived from the table and can be stated as follows (the words inside parentheses should be used for the subtraction algorithm).

Addition (subtraction) algorithm: when the signs of A and B are identical (different), add the two magnitude and attach the sign of A to the result. When the sign of A and B are different (identical), compare the magnitudes.



Operation	Add Magnitudes	Subtract Magnitudes		
		When $A > B$	When $A < B$	When $A = B$
$(+A) + (+B)$	$+(A + B)$			
$(+A) + (-B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(-A) + (+B)$		$-(A - B)$	$+(B - A)$	$+(A - B)$
$(-A) + (-B)$	$-(A + B)$			
$(+A) - (+B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(+A) - (-B)$	$+(A + B)$			
$(-A) - (+B)$	$-(A + B)$			
$(-A) - (-B)$		$-(A - B)$	$+(B - A)$	$+(A - B)$

Figure: Table for Addition and Subtraction of Signed-Magnitude Numbers

Hardware implementation:

To implement the two arithmetic operations with hardware, it is first necessary that the two numbers be stored in registers. Let A and B be two registers that hold the magnitude of the numbers, and A_s and B_s be two flip-flops that hold the corresponding signs. The results of the operation may be transferred to a third register however, a saving achieved if the result is transferred into A and A's. Thus A and A_s together from an accumulator register.

Consider now the hardware implementation of the algorithms above. First, a parallel adder is needed to perform the micro operation A+B. second, comparator circuit is needed to establish if A>B, A=B, or A<B. third, two parallel subtractor circuits are needed to perform the micro operation A-B and B-A. The sign relationship can be determined from an exclusive-OR gate with A_s andB_s as inputs.

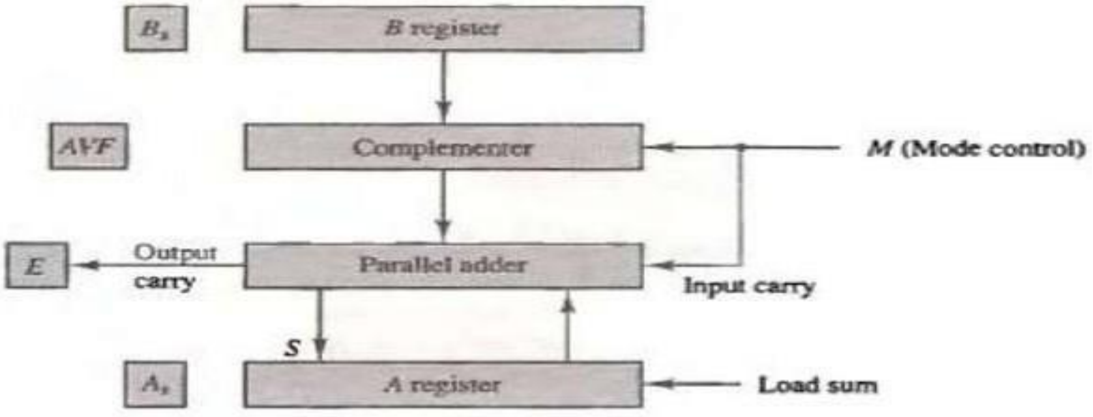


Figure: Hardware for Signed-Magnitude Addition and Subtraction.

The output carry is transferred to flip-flop E. The complementer consists of exclusive-OR gates and the parallel adder consists of full adder circuit.

Hardware algorithm:

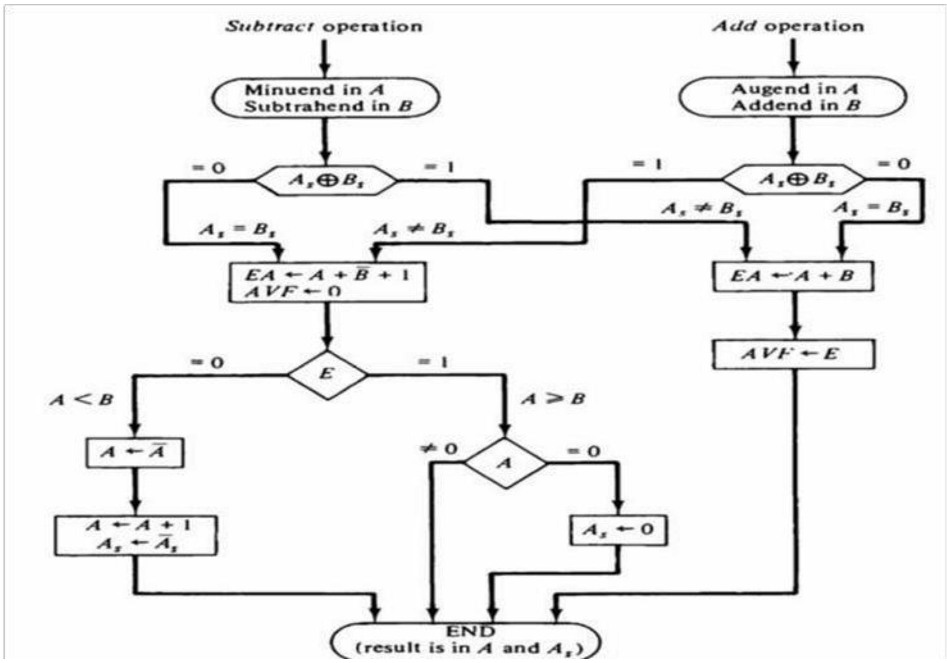


Figure: Hardware Algorithm.

The two signs A_s and B_s are compared by an exclusive-OR gate. For an add operation, identical signs dictate that the magnitudes be added, for subtract operation different signs dictate that the magnitudes be added. The magnitudes are added with a micro operation E AßA+B. Where E A is a register that combines E and A. For A 0 indicates that A<B, for this case it is necessary to take the 2’s compliment of the value in A .this operation can be done with one micro operationAßĀ+1.

However, we assume that A register as circuits for micro operation compliment and increment, so the 2’s compliment is obtain from these two micro operations… The value in AVF provides an overflow indication. The final value of E is immaterial.

Addition and Subtraction with signed2’s complement data:

The left most bit of binary number represents the sign bit; 0 for positive and 1 for negative. If the sign bit is 1, the entire the entire number is represented in 2’s compliment form. The addition of two numbers in signed-2’s complement form consists of adding the number with the sign bits treated the same as the other bits of the number. A carry out of the sign bit position is discarded. The subtraction consists of first taking the 2’s compliment of the subtrahend and then adding it to the minu end When two numbers of n digits each are added and the sum occupies n+1 Digits, we say that an overflow occurred. When the two carriers are applied to an exclusive-OR gate, the overflow is detected when the output of the gate is equal to 1.

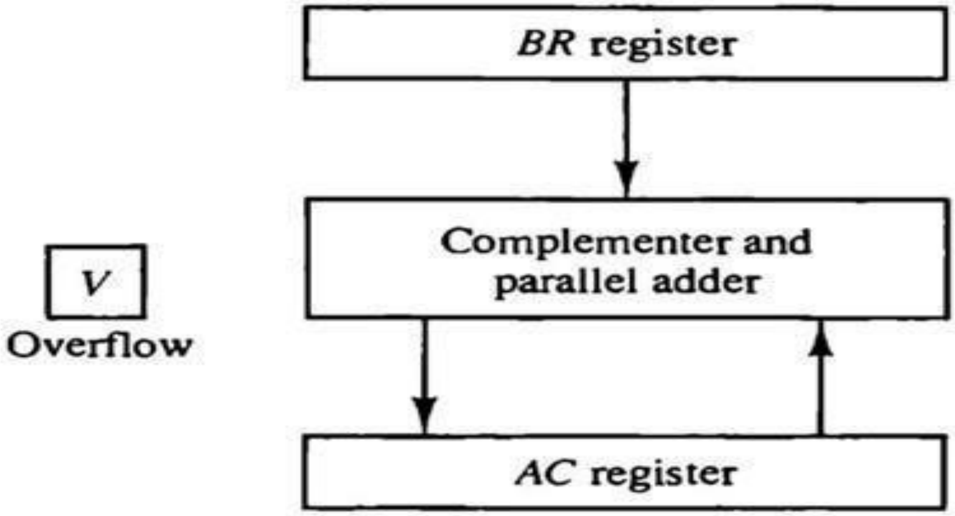


Figure:--Hardware for Signed 2's Compliant Addition and Subtraction

The left most bit in AC and BR represents the sign bits of the numbers. The over flow flip-flops V is set to 1 if there is an overflow. The outputs carry in this case.is discarded.

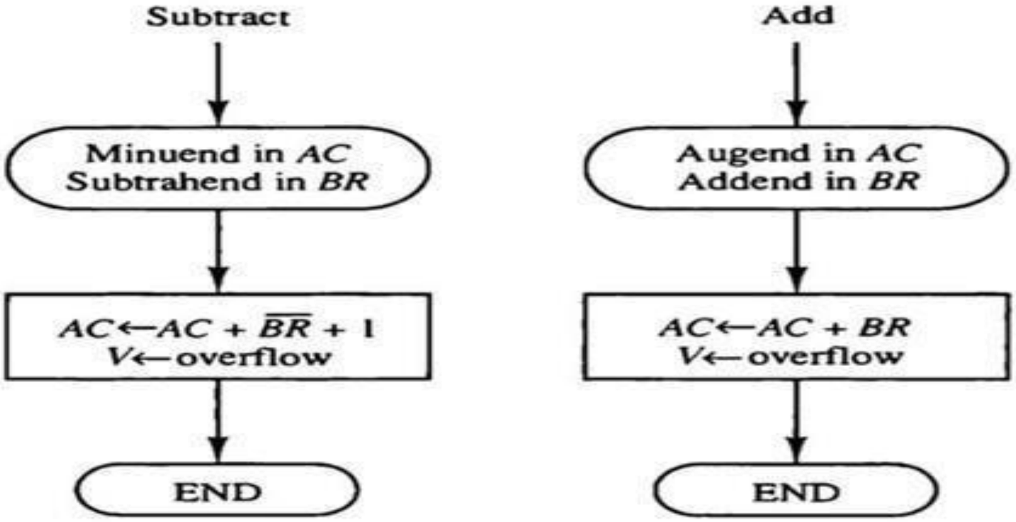


Figure:--Algorithm for Adding and Subtracting numbers in Signed 2's Compliment representation.

The sum is obtained by adding the contents of AC and BR(including their sign bits). The overflow bit V is set to 1 if the ex-OR of the last two carries is 1, and it is cleared to 0 otherwise.

Multiplication and Division:

Reading Assignments and Exercises:

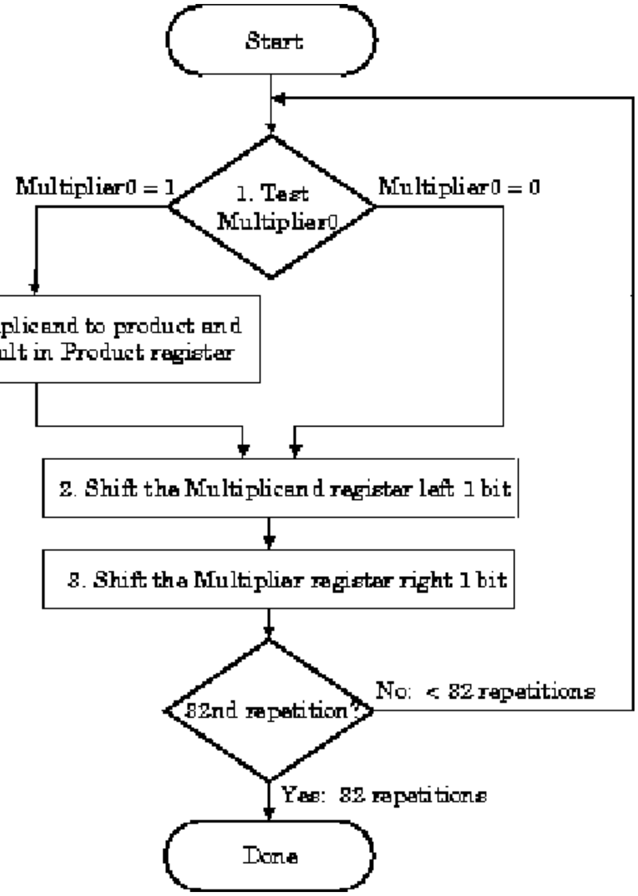
Multiplication is more complicated than addition, being implemented by shifting as well as addition. Because of the partial products involved in most multiplication algorithms, more time and more circuit area is required to compute, allocate, and sum the partial products to obtain the multiplication result.

Multiplier Design:

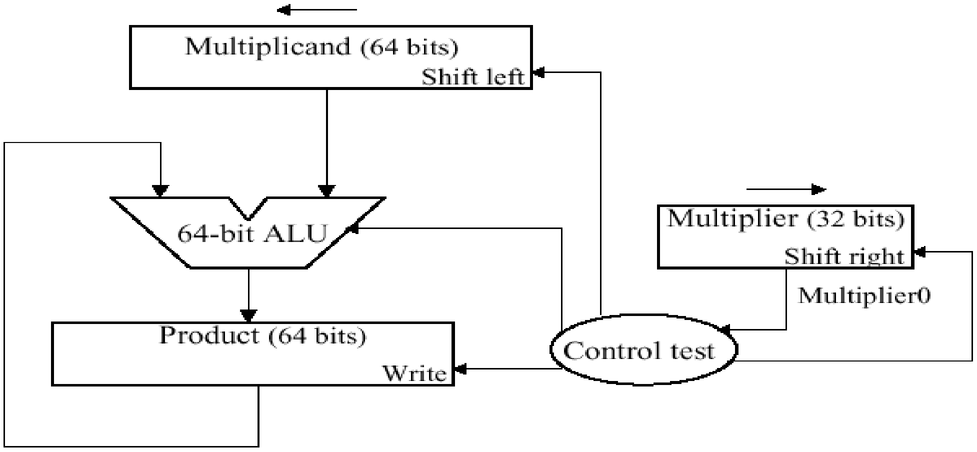
We herein discuss three versions of the multiplier design based on the pencil-and-paper algorithm for multiplication that we all learned in grade school, which operates on Boolean numbers, as follows:

```
Multiplicand: 0010 # Stored in register r1
Multiplier: x 1101 # Stored in register r2
--------------------
Partial Prod    0010     # No shift for LSB of Multiplier
      "      "     0000      # 1-bit shift of zeroes (can omit)
      "     "     0010      # 2-bit shift for bit 2 of Multiplier
      " “ 0010 # 3-bit shift for bit 3 of Multiplier
        -------------------- # Zero-fill the partial products and add
        PRODUCT 0011010 # Sum of all partial products -> r3
```

A flowchart of this algorithm, adapted for multiplication of 32-bit numbers, is shown in Figure below; together with a schematic representation of a simple ALU circuit that implements this version of the algorithm. Here, the multiplier and the multiplicand are shifted relative to each other, which is more efficient than shifting the partial products alone.


Start
1. Test Multiplier0
Multiplier0 = 1
Multiplier0 = 0
1a. Add multiplicand to product and place the result in Product register
2. Shift the Multiplicand register left 1 bit
3. Shift the Multiplier register right 1 bit
32nd repetition?
No: < 32 repetitions
Yes: 32 repetitions
Done

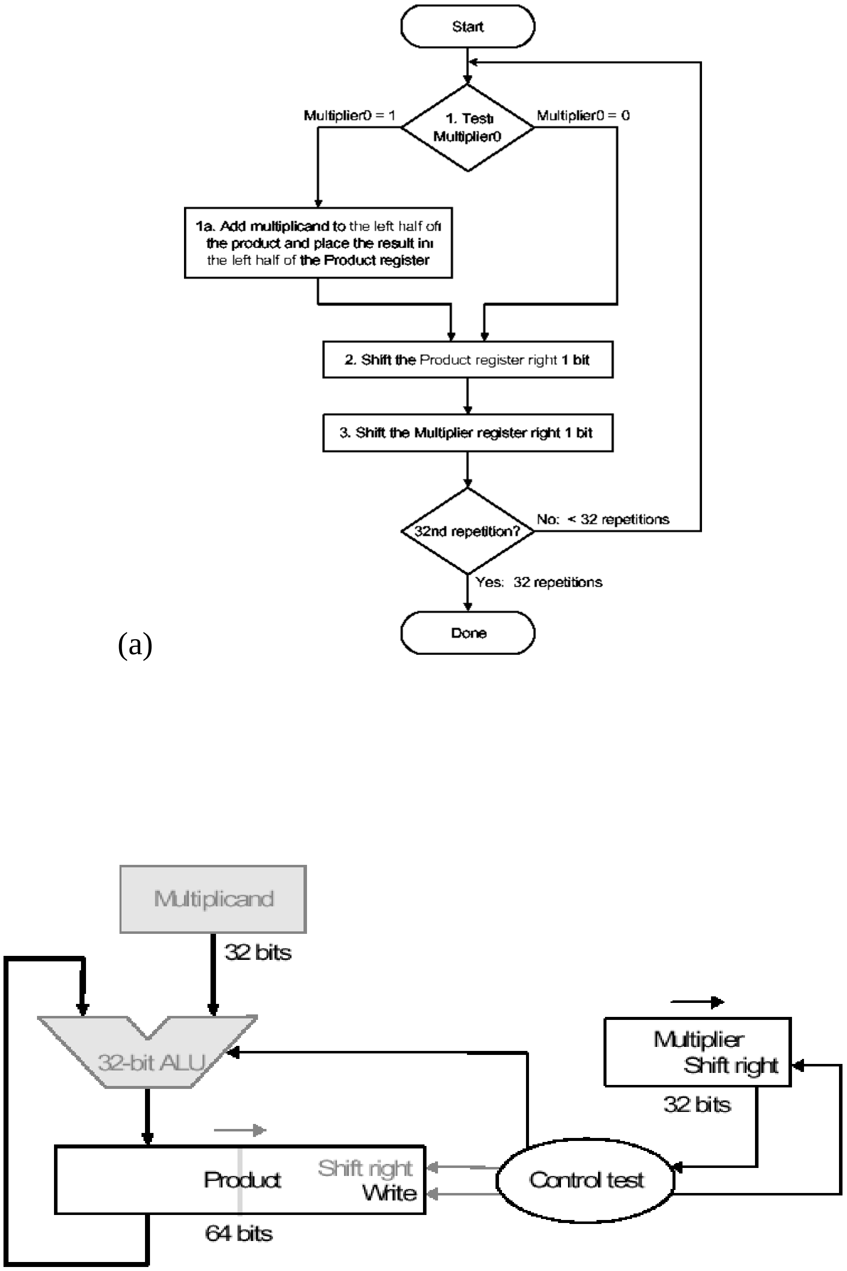
(a)



(b)

Figure: Pencil-and-paper multiplication of 32-bit Boolean number representations: (a) algorithm, and (b) simple ALU circuitry

The second version of this algorithm is shown in Figure. Here, the product is shifted with respect to the multiplier, and the multiplicand is shifted after the product register has been shifted. A 64-bit register is used to store both the multiplicand and the product.



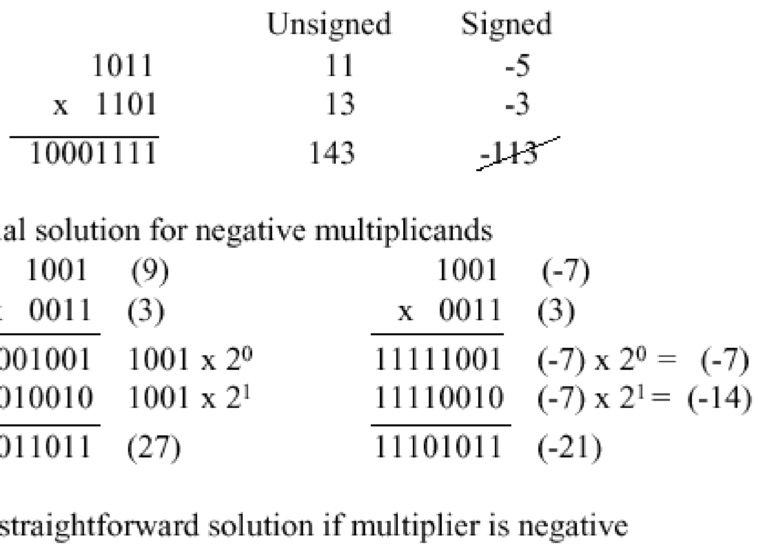
(a)

(b)

Figure: Second version of pencil-and-paper multiplication of 32-bit Boolean number representations: (a) algorithm, and (b) schematic diagram of ALU circuitry.

Thus, we have the following shift-and-add scheme for multiplication:

The preceding algorithms and circuitry does not hold for signed multiplication, since the bits of the multiplier no longer correspond to shifts of the multiplicand. The following example is illustrative:



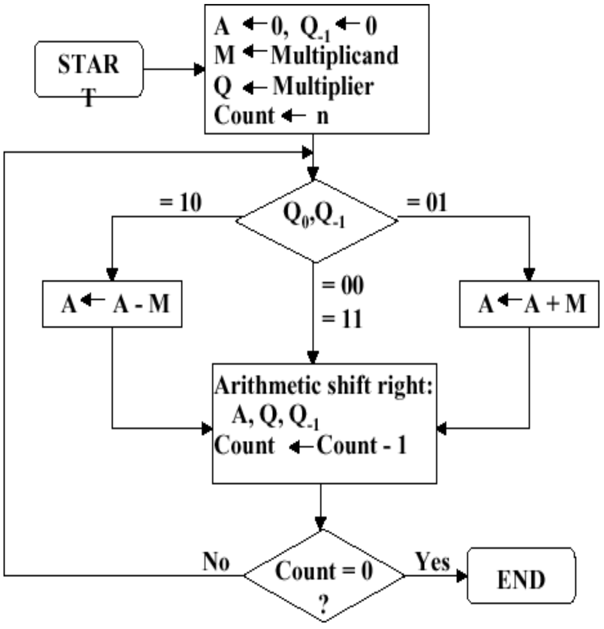
	Unsigned	Signed
1011	11	-5
x 1101	13	-3
10001111	143	~~-113~~

- Partial solution for negative multiplicands

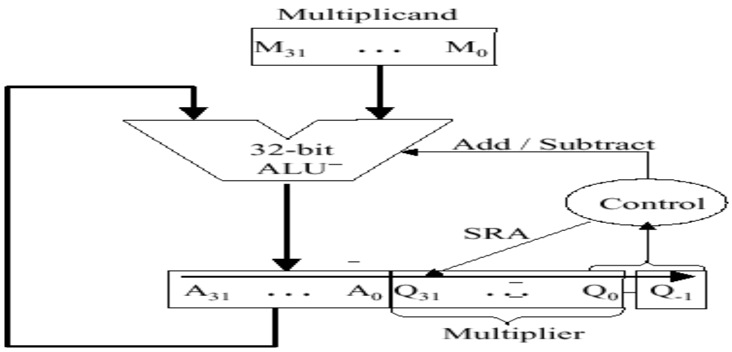
1001	(9)	1001	(-7)
x 0011	(3)	x 0011	(3)
00001001	1001 x 2^0	11111001	(-7) x 2^0 = (-7)
00010010	1001 x 2^1	11110010	(-7) x 2^1 = (-14)
00011011	(27)	11101011	(-21)

- No straightforward solution if multiplier is negative

A solution to this problem is Booth's Algorithm, whose flowchart and corresponding schematic hardware diagram are shown in Figure. Here, the examination of the multiplier is performed with look ahead toward the next bit. Depending on the bit configuration, the multiplicand is positively or negatively signed, and the multiplier is shifted or un shifted.



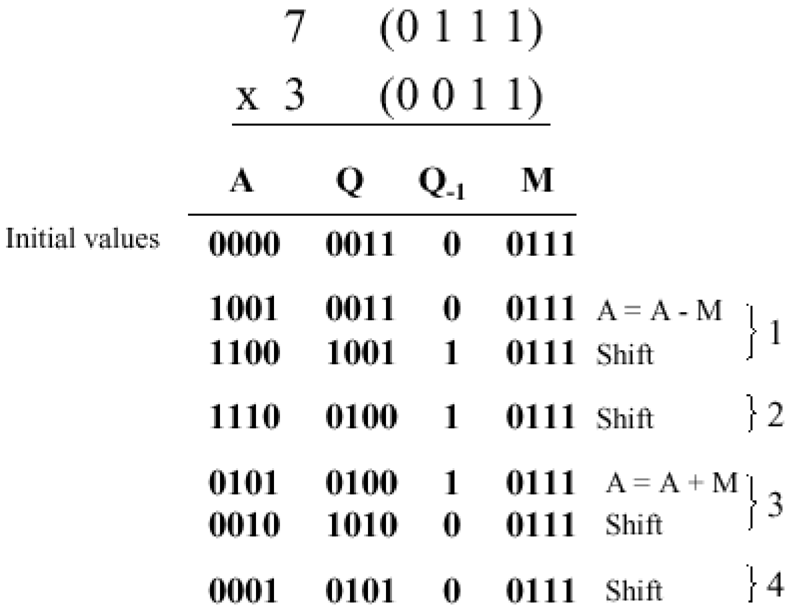
(a)



(b)

Figure. Booth's procedure for multiplication of 32-bit Boolean number representations: (a) algorithm, and (b) schematic diagram of ALU circuitry.

Observe that Booth's algorithm requires only the addition of a subtraction step and the comparison operations for the two-bit codes, versus the one-bit comparison in the preceding three algorithms. An example of Booth's algorithm follows:



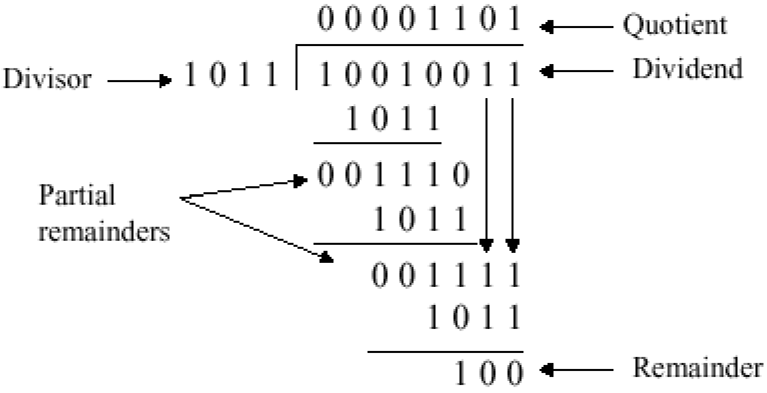
$$\begin{array}{rl} 7 & (0\,1\,1\,1) \\ \text{x }\ 3 & (0\,0\,1\,1) \\ \hline \end{array}$$

	A	Q	Q_{-1}	M		
Initial values	0000	0011	0	0111		
	1001	0011	0	0111	A = A - M	1
	1100	1001	1	0111	Shift	
	1110	0100	1	0111	Shift	2
	0101	0100	1	0111	A = A + M	3
	0010	1010	0	0111	Shift	
	0001	0101	0	0111	Shift	4

Here N = 4 iterations of the loop are required to produce a product from two N = 4 digit operands. Four shifts and two subtractions are required. From the analysis of the algorithm shown in Figure 3.18a, it is easily seen that the maximum work for multiplying two N-bit numbers is given by **O**(N) shift and addition operations. From this, the worst-case computation time can be computed given CPI for the shift and addition instructions, as well as cycle time of the ALU.

Design of Arithmetic Division Hardware:

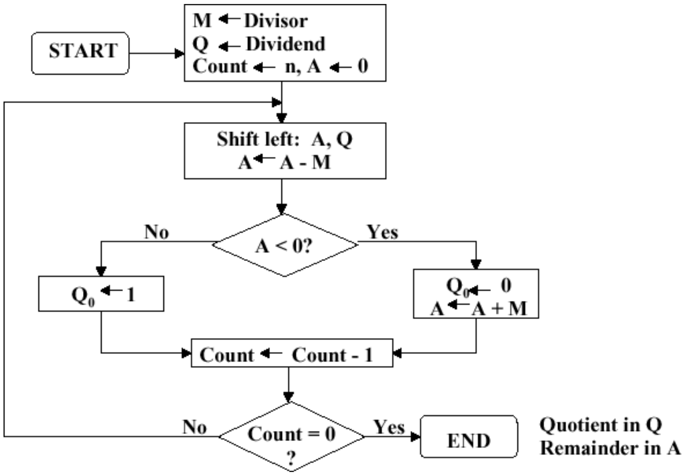
Division is a similar operation to multiplication, especially when implemented using a procedure similar to the algorithm shown in Figure 3.18a. For example, consider the pencil-and-paper method for dividing the byte 10010011 by the nybble 1011:



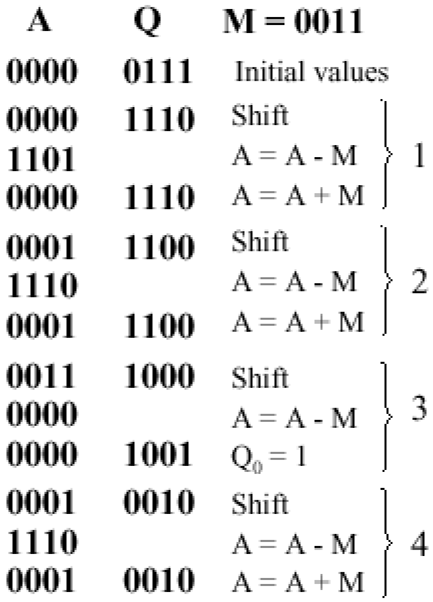
The governing equation is as follows:

$$\text{Dividend} = \text{Quotient} \cdot \text{Divisor} + \text{Remainder}.$$

Unsigned Division. The unsigned division algorithm that is similar to Booth's algorithm is shown in Figure a, with an example shown in Figure b. The ALU schematic diagram in given in Figure c. The analysis of the algorithm and circuit is very similar to the preceding discussion of Booth's algorithm.

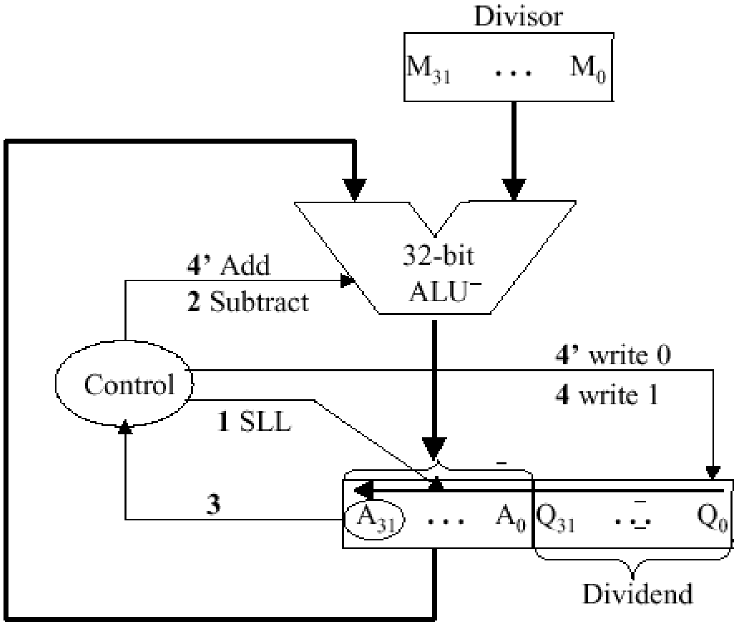


(a)



A	Q	M = 0011	
0000	0111	Initial values	
0000	1110	Shift	1
1101		A = A - M	
0000	1110	A = A + M	
0001	1100	Shift	2
1110		A = A - M	
0001	1100	A = A + M	
0011	1000	Shift	3
0000		A = A - M	
0000	1001	$Q_0 = 1$	
0001	0010	Shift	4
1110		A = A - M	
0001	0010	A = A + M	

(b)



(c)

Figure . Division of 32-bit Boolean number representations: (a) algorithm, (b) example using division of the unsigned integer 7 by the unsigned integer 3, and (c) schematic diagram of ALU circuitry.

Signed Division

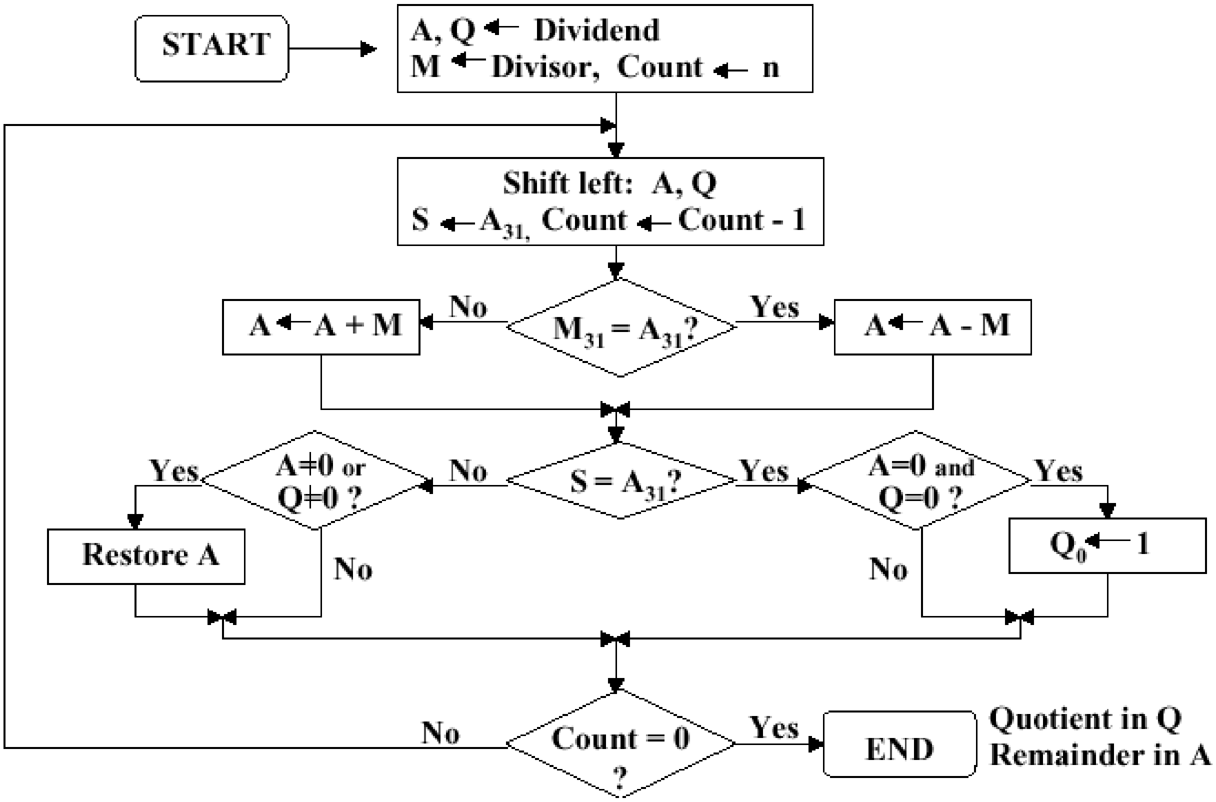
With signed division, we negate the quotient if the signs of the divisor and dividend disagree. The remainder and the divident must have the same signs. The governing equation is as follows:

$$\text{Remainder} = \text{Divident} - (\text{Quotient} \cdot \text{Divisor}) \,,$$

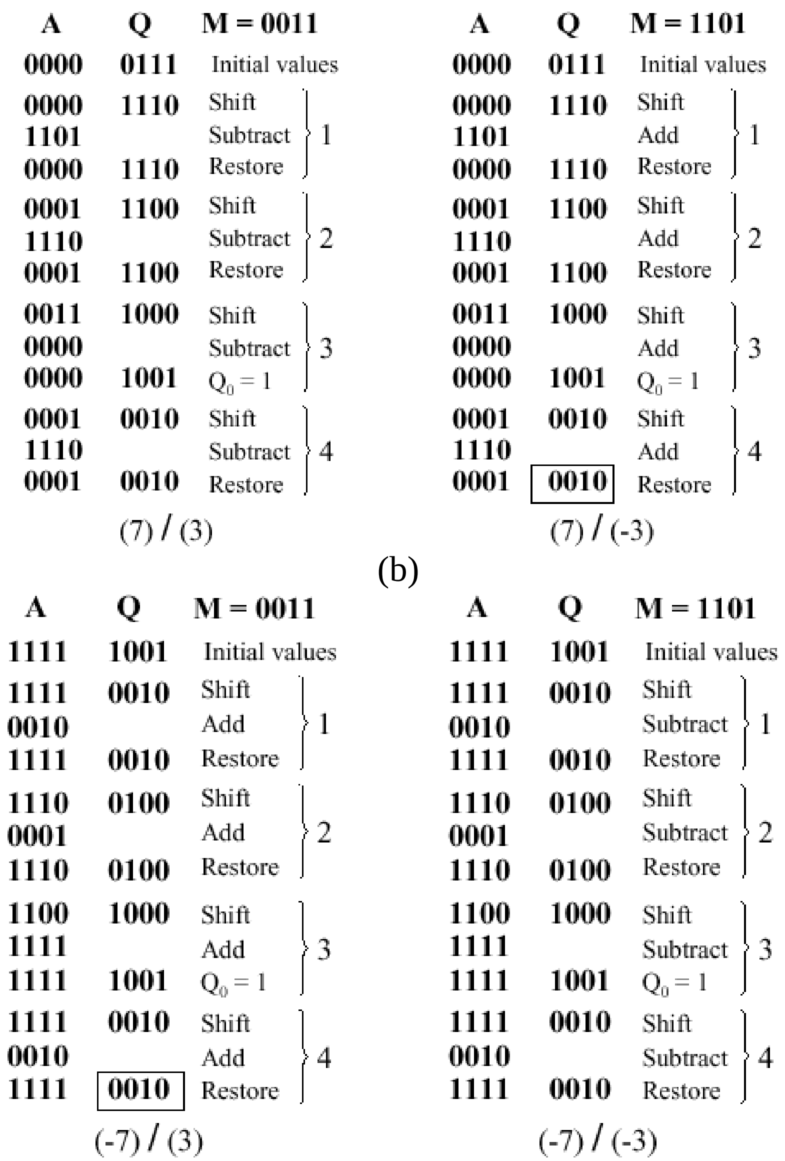
and the following four cases apply:

(+7) / (+3): Q = 2; R = 1
(-7) / (+3): Q = -2; R = -1
(+7) / (-3): Q = -2; R = 1
(-7) / (-3): Q = 2; R = -1

We present the preceding division algorithm, revised for signed numbers, as shown in Figure a. Four examples, corresponding to each of the four preceding sign permutations, are given in Figure b and c.



(a)



A	Q	M = 0011	
0000	0111	Initial values	
0000	1110	Shift	1
1101		Subtract	
0000	1110	Restore	
0001	1100	Shift	2
1110		Subtract	
0001	1100	Restore	
0011	1000	Shift	3
0000		Subtract	
0000	1001	$Q_0 = 1$	
0001	0010	Shift	4
1110		Subtract	
0001	0010	Restore	

(7) / (3)

A	Q	M = 1101	
0000	0111	Initial values	
0000	1110	Shift	1
1101		Add	
0000	1110	Restore	
0001	1100	Shift	2
1110		Add	
0001	1100	Restore	
0011	1000	Shift	3
0000		Add	
0000	1001	$Q_0 = 1$	
0001	0010	Shift	4
1110		Add	
0001	$\boxed{0010}$	Restore	

(7) / (-3)

(b)

A	Q	M = 0011	
1111	1001	Initial values	
1111	0010	Shift	1
0010		Add	
1111	0010	Restore	
1110	0100	Shift	2
0001		Add	
1110	0100	Restore	
1100	1000	Shift	3
1111		Add	
1111	1001	$Q_0 = 1$	
1111	0010	Shift	4
0010		Add	
1111	$\boxed{0010}$	Restore	

(-7) / (3)

A	Q	M = 1101	
1111	1001	Initial values	
1111	0010	Shift	1
0010		Subtract	
1111	0010	Restore	
1110	0100	Shift	2
0001		Subtract	
1110	0100	Restore	
1100	1000	Shift	3
1111		Subtract	
1111	1001	$Q_0 = 1$	
1111	0010	Shift	4
0010		Subtract	
1111	0010	Restore	

(-7) / (-3)

(c)

Figure. Division of 32-bit Boolean number representations: (a) algorithm, and (b,c) examples using division of +7 or -7 by the integer +3 or -3;

Division in MIPS

MIPS supports multiplication and division using existing hardware, primarily the ALU and shifter. MIPS needs one extra hardware component - a 64-bit register able to support sll and sra instructions. The upper (high) 32 bits of the register contains the remainder resulting from division. This is moved into a register in the MIPS register stack (e.g., $t0) by themfhi command. The lower 32 bits of the 64-bit register contains the quotient resulting from division. This is moved into a register in the MIPS register stack by the mflo command.

In MIPS assembly language code, signed division is supported by the div instruction and unsigned division, by the divu instruction. MIPS hardware does not check for division by zero. Thus, divide-by-zero exception must be detected and handled in system software. A similar comment holds for overflow or underflow resulting from division.

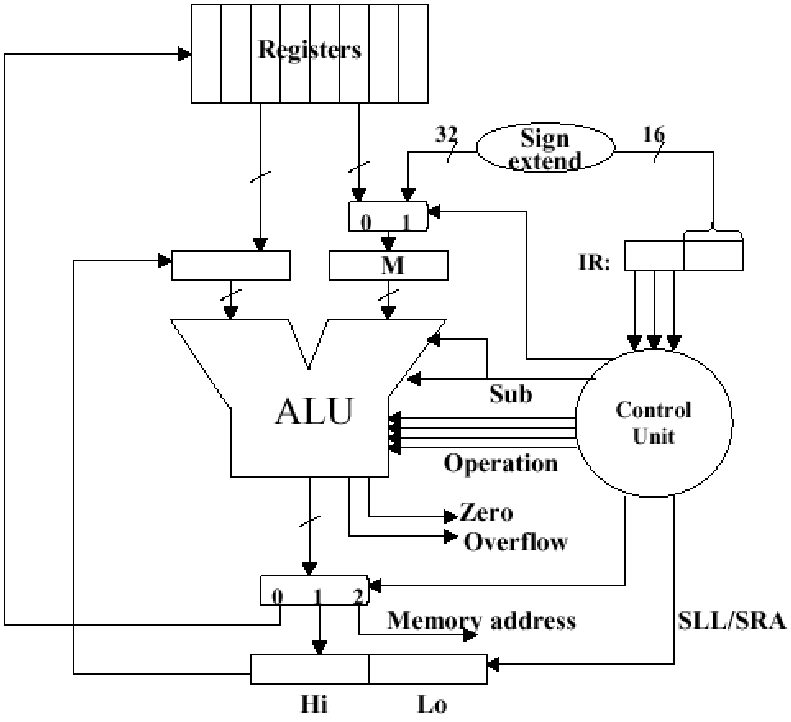


Figure: MIPS ALU supporting the integer arithmetic operations (+,-,x,/)

Combinational ALU's:

An ALU is the fundamental unit of any computing system.

Understanding how an ALU is designed and how it works is essential to building any advanced logic circuits.

Using this knowledge and experience, we can move on to designing more complex integrated circuits.

The ALU is the "heart" of a processor—you could say that everything else in the CPU is there to support the ALU.

Typical Schematic Symbol of an ALU:

A and B: the inputs to the ALU (aka operands)

R: Output or Result

F: Code or Instruction from the control Unit (aka as op-code)

D: Output status; it indicates cases

Such as:

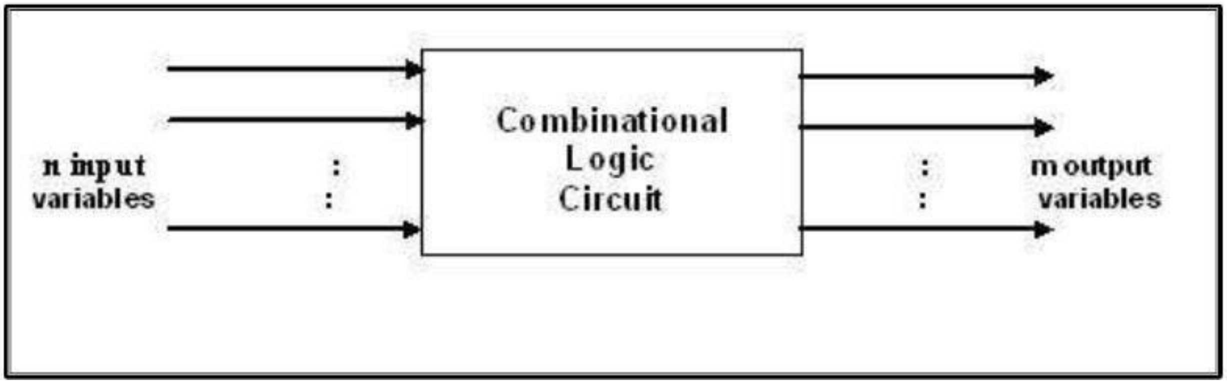
- Carry-in

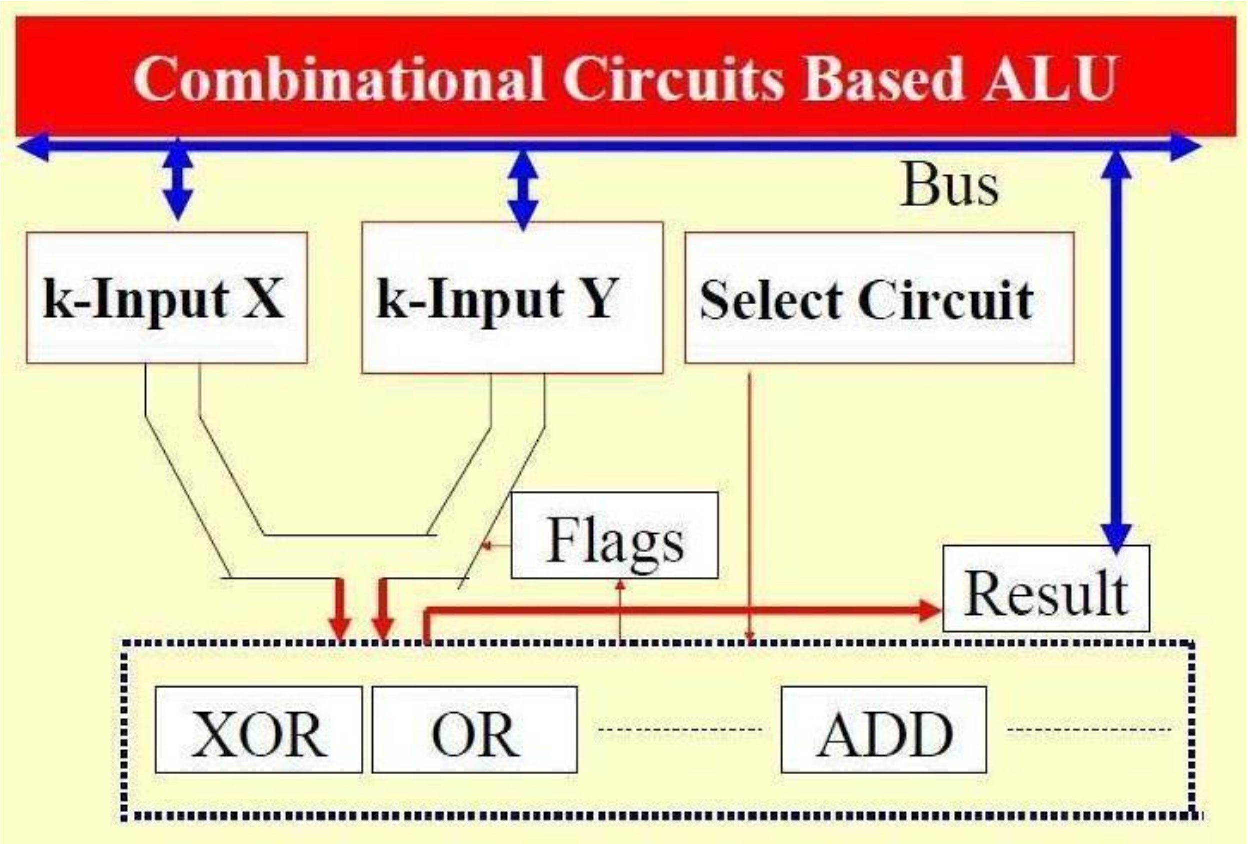
- Carry-out,
- Overflow,
- Division-by-zero
- And . ..

Combinational Circuits:

1. Combinational Circuits are made of logic gates.
2. Doesn’t contain memory element, that’s why they can’t store any information.
3. Value of present output is determined by present input.
4. Examples of combinational circuits are half adders, full adders, sub tractor set c.

Block Diagram of Combinational Circuits:




Combinational Circuits Based ALU

Bus

k-Input X

k-Input Y

Select Circuit

Flags

Result

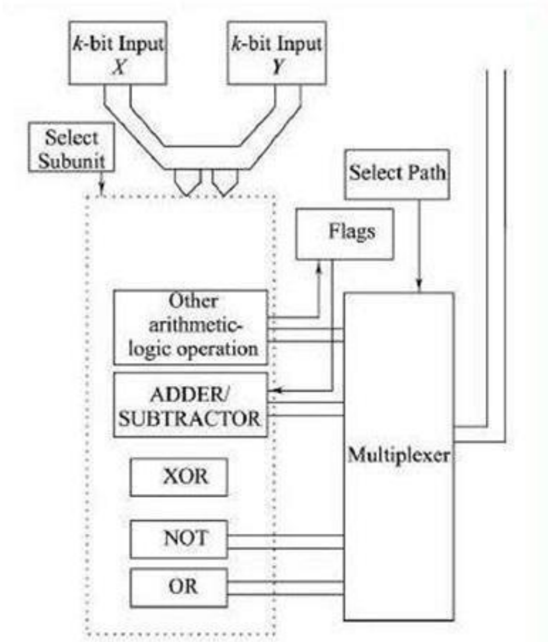
XOR

OR

ADD

Figure: Combinational Circuits Based ALU

An ALU using Combinational Circuits:



Examples of Combinational Circuits:

1. Multiplexer
2. Demultiplexer
3. Encoder
4. Decoder
5. Half Adder
6. Full Adder

Multiplexer:

A multiplexer is a combinational circuit where binary information from one of many input lines is selected and directs it to a single output line.

Demultiplexer:

Demultiplexing is the reverse process of multiplexing; i.e., a Demultiplexer is a combinational circuit that receives information on a single line and transmits this information on one of 2n possible output lines.

Encoder:

An encoder is a combinational circuit that Produces the reverse function from that of a Decoder.

Decoder:

Decoder is a combinational logic circuit that receives coded information on n input lines and feeds them to maximum of 2n unique output lines after conversion

Half Adder:

A half-adder is a combinational circuit that performs the addition of two bits.

Full Adder:

This type of adder is a little more difficult to implement than a half-adder.

The main difference between a half-adder and a full adder is that the full-adder has three inputs and two outputs.

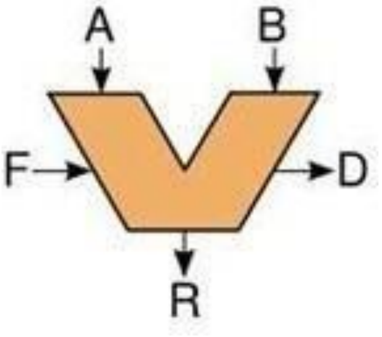
Design Procedure for Combinational Circuits:

1. This procedure involves the following steps:
2. The problem is stated.
3. The number of available input variables and output variables is determined.

4. The input and output variables are assigned letter symbols.
5. Truth table is drawn
6. Boolean function for output is obtained.
7. The logic diagram is drawn.

Design Procedure for Combinational Circuits:

1. To determine the output functions as algebraic expressions.
2. It is the reverse process of design procedure.
3. Logic diagram of the circuit is given.
4. Obtain the truth table from the diagram.
5. Obtain Boolean function from the Truth Table for output.



Sequential ALU's

An ALU is the fundamental unit of any computing system.
Understanding how an ALU is designed and how it works is essential to building any advanced logic circuits.
Using this knowledge and experience, we can move on to designing more complex integrated circuits.
The ALU is the "heart" of a processor—you could say that everything else in the CPU is there to support the ALU.

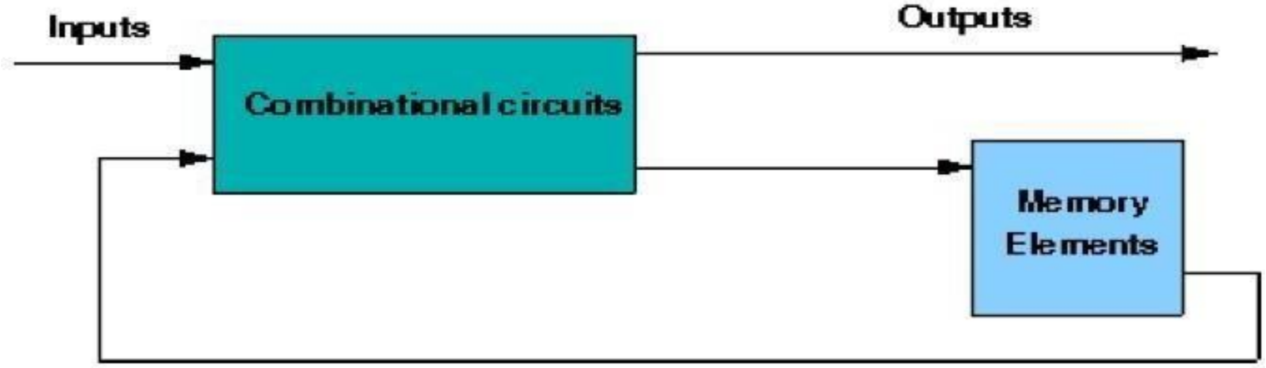
Typical Schematic Symbol of an ALU:
A and B: the inputs to the ALU (aka operands)
R: Output or Result
F: Code or Instruction from the Control Unit (aka asop-code)
D: Output status; it indicates cases
Such as:

- Carry-in
- Carry-out,
- Overflow,
- Division-by-zero
- And . . .

Sequential Logic Circuits:

1) Made up of combinational circuits and memory elements.
2) These memory elements are devices capable of storing ONE-BIT information.
3) Output depends on input and previous state.
4) Examples of sequential circuits are flip flops, counters, shift registers.

Block Diagram of Sequential Circuits:



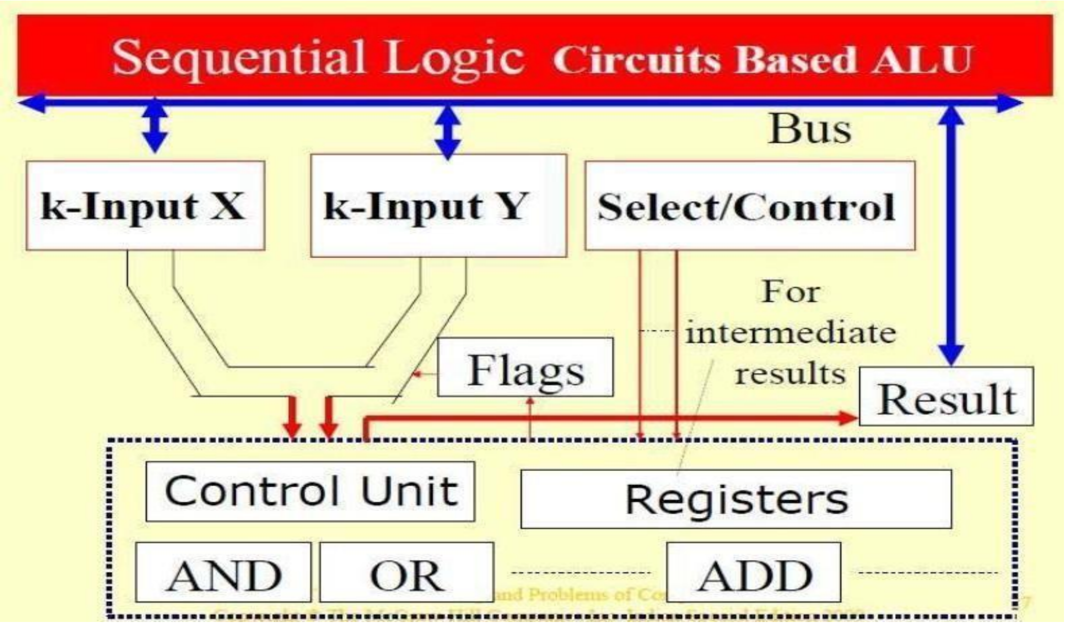
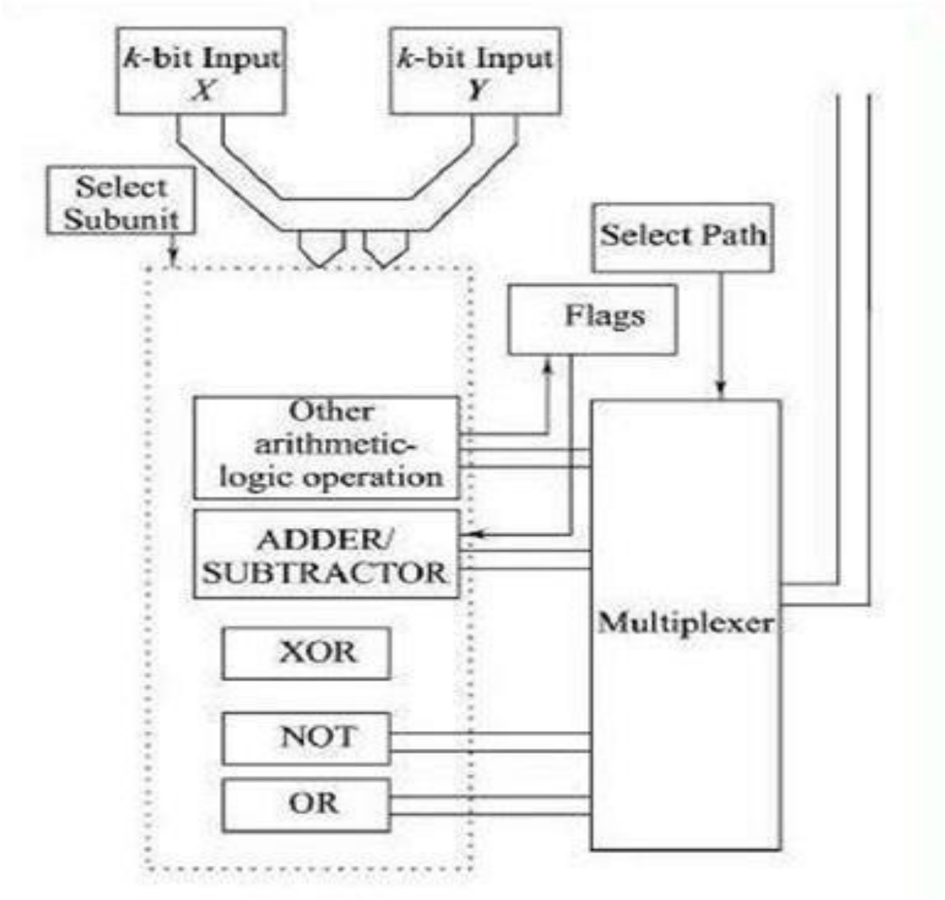


Figure: Sequential Circuits Based ALU

An ALU using Sequential Circuits:



Examples of Sequential Circuits:

1) Flip-Flops
    a. JK Flip-Flop
    b. RS Flip-Flop
    c. PR Flip-Flop
    d. D Flip-Flop
2) Registers
3) Counters.

Flip-Flops:

- Flip-Flops are the basic building blocks of sequential circuits.
- A flip-flop is a binary cell which can store a bit of information.
- A basic function of flip-flop is storage, which means memory. A flip-flop (FF) is capable of storing 1 (one) bit of binary data.
- It has two stable states either '1' or '0'. A flip-flop maintains any one of the two stable states which can be treated as zero or one depending on presence and absence of output signals.

Registers and Counters:

A circuit with flip-flops is considered a sequential circuit even in the absence of combinational logic. Circuits that include flip-flops are usually classified by the function they perform.

Two such circuits are registers and counters:

Registers-

a. It is a group of flip-flops.
b. Its basic function is to hold information within a digital system so as to make it available to the logic units during the computing process.

Counters:-

a. It is essentially a register that goes through a predetermined sequence of states.

Types of Sequential Circuits:

Sequential circuits are of two types:

1. Synchronous Sequential Circuits
2. Asynchronous Sequential Circuits

Synchronous Sequential Circuits:

- In synchronous sequential circuits, the state of the device changes only at discrete times in response to a clock Pulse.
- In a synchronous circuit, an electronic oscillator called a clock generates a sequence of repetitive pulses called the clock signal which is distributed to all the memory elements in the circuit

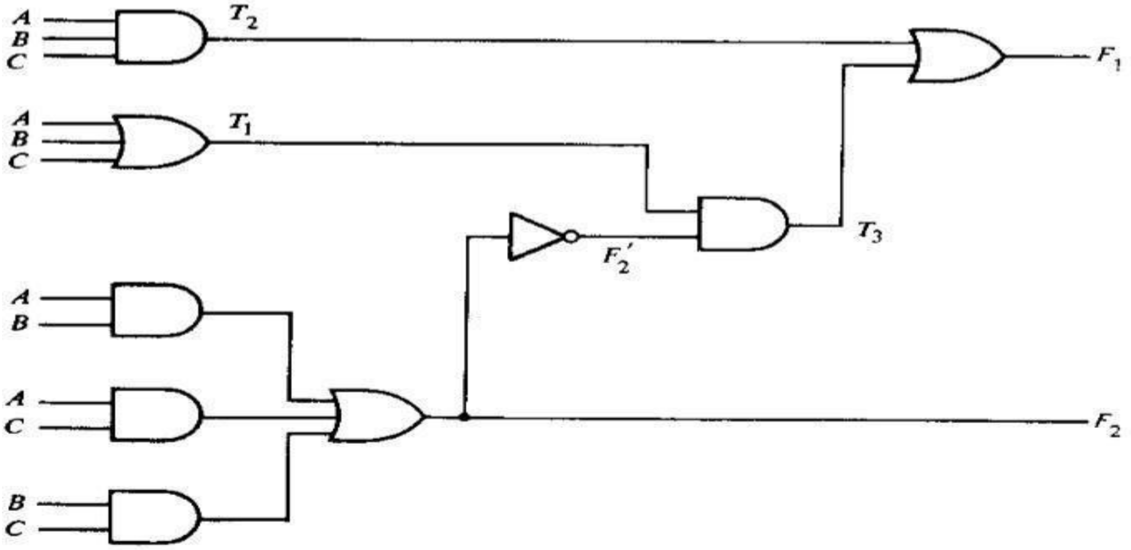
Asynchronous Sequential Circuits:

- Asynchronous circuit is not synchronized by a clock signal; the outputs of the circuit change directly in response to changes in Inputs.
- The advantage of asynchronous logic is that it can be faster than synchronous logic, because the circuit doesn't have to wait for a clock signal to process inputs.
- The speed of the device is potentially limited only by the propagation delays of the logic gates used.

Design Procedure for Sequential Circuits:
This process involves the following steps:

- Make a state table based on the problem statement. The table should show the present states, inputs, next states and outputs. (It may be easier to find a state diagram first, and then convert that to a table)
- Assign binary codes to the states in the state table, if you haven't already. If you have n states, your binary codes will have at least [log2 n] digits, and your circuit will have at least [log2 n] flip-flops
- For each flip-flop and each row of your state table, find the flip-flop input values that are needed to generate the next state from the present state. You can use flip-flop excitation tables here.
- Find simplified equations for the flip-flop inputs and the outputs.
- Build the circuit!

Logic Diagram for Analysis Example:



The circuit has 3 inputs A, B, C and 2 outputs F1, F2:

The Boolean function for outputs is:

- T1=A+B+C
- T2=ABC
- T3=F2'T1

Outputs functions for gates are:

- F1=T3+T2
- F2=AB+AC+BC

Substituting and Simplifying, we get:

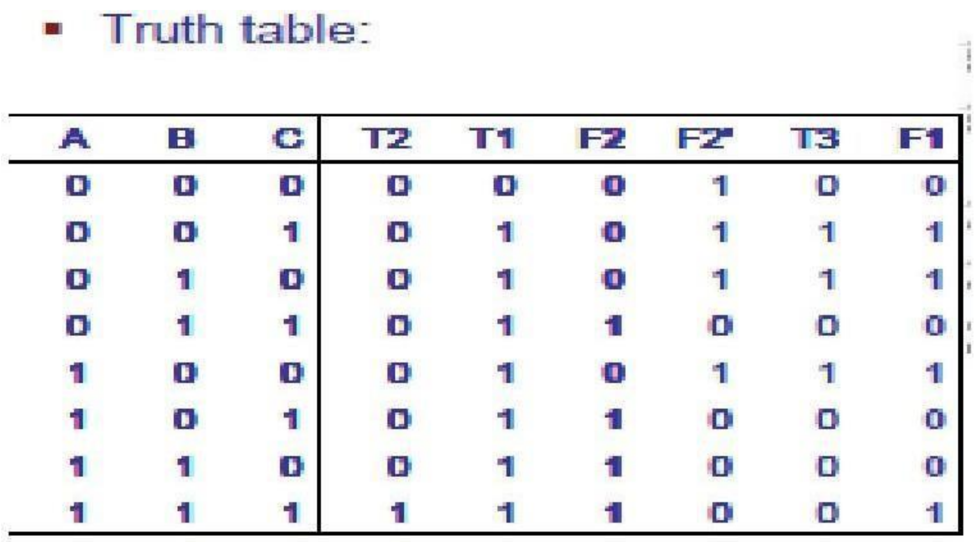
$$F_1 = T_3 + T_2 = F_2'T_1 + ABC = (AB + AC + BC)'(A + B + C) + ABC$$

$$= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC$$

$$= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC$$

$$= A'BC' + A'B'C + AB'C' + ABC$$

Truth table drawn from the logic diagram:



- Truth table:

A	B	C	T2	T1	F2	F2'	T3	F1
0	0	0	0	0	0	1	0	0
0	0	1	0	1	0	1	1	1
0	1	0	0	1	0	1	1	1
0	1	1	0	1	1	0	0	0
1	0	0	0	1	0	1	1	1
1	0	1	0	1	1	0	0	0
1	1	0	0	1	1	0	0	0
1	1	1	1	1	1	0	0	1

Boolean functions obtained for output are:

- F2=AB+AC+BC
- F1=A’BC’+A’B’C+AB’C’+ABC

Carry look ahead adder:

Motivation behind Carry Look-Ahead Adder:

In ripple carry adders, for each adder block, the two bits that are to be added are available instantly. However, each adder block waits for the carry to arrive from its previous block. So, it is not possible to generate the sum and carry of any block until the input carry is known.
The ith block waits for the i- 1th block to produce its carry. So there will be a considerable time delay which is carry propagation delay.

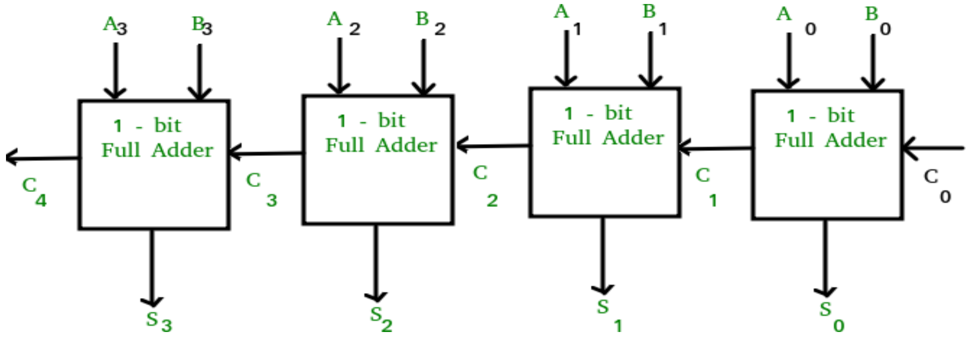


Figure – Digital Logic

Consider the above 4-bit ripple carry adder. The sum S4 is produced by the corresponding full adder as soon as the input signals are applied to it. But the carry input C4 is not available on its final steady state value until carry C3 is available at its steady state value. Similarly C3 depends on C2 and C2 on C1. Therefore, though the carry must propagate to all the stages in order that output S3 and carry C4 settle their final steady-state value. The propagation time is equal to the propagation delay of each adder block, multiplied by the number of adder blocks in the circuit. For example, if each full adder stage has a propagation delay of 20 nanoseconds, then S3 will reach its final correct value after 60 (20 × 3) nanoseconds. The situation gets worse, if we extend the number of stages for adding more number of bits.

Carry Look-ahead Adder:

A carry look-ahead adder reduces the propagation delay by introducing more complex hardware. In this design, the ripple carry design is suitably transformed such that the carry logic over fixed groups of bits of the adder is reduced to two-level logic. Let us discuss the design in detail.

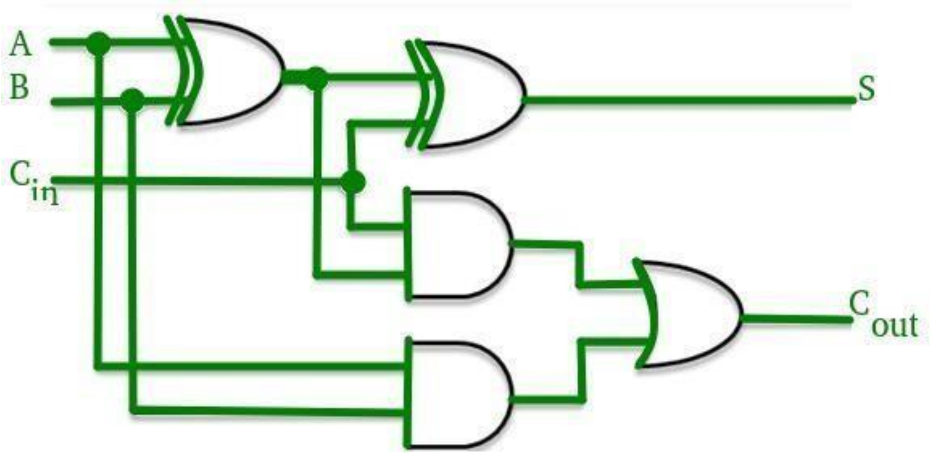
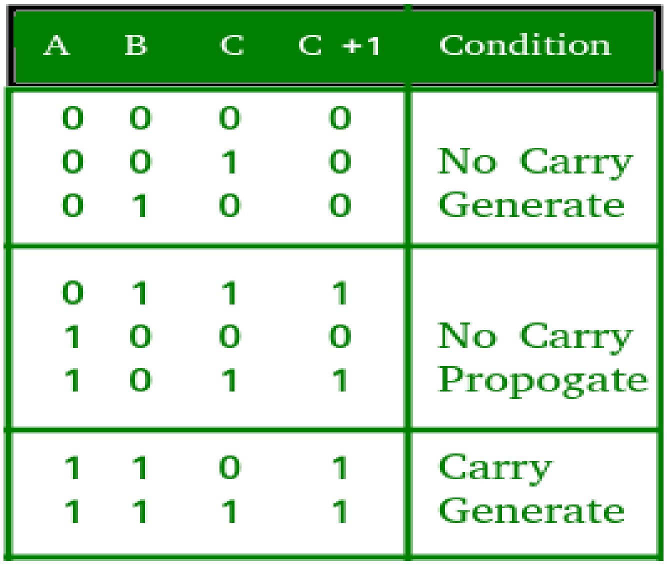
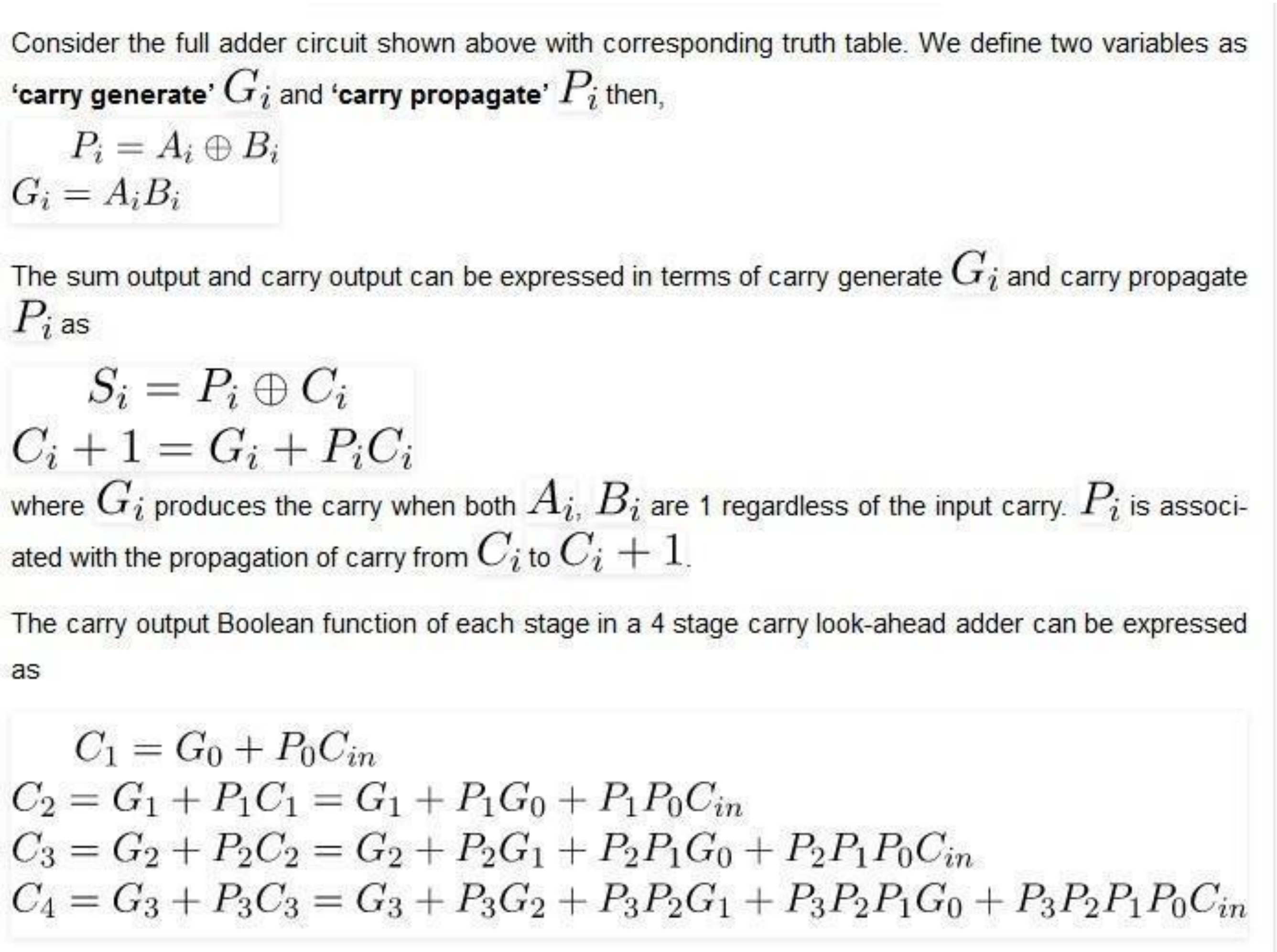


Figure - Design.



A	B	C	C +1	Condition
0	0	0	0	No Carry Generate
0	0	1	0	
0	1	0	0	
0	1	1	1	No Carry Propogate
1	0	0	0	
1	0	1	1	
1	1	0	1	Carry Generate
1	1	1	1	

Figure – Truth table.



Consider the full adder circuit shown above with corresponding truth table. We define two variables as **'carry generate'** G_i and **'carry propagate'** P_i then,

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$

The sum output and carry output can be expressed in terms of carry generate G_i and carry propagate P_i as

$$S_i = P_i \oplus C_i$$
$$C_i + 1 = G_i + P_i C_i$$

where G_i produces the carry when both A_i, B_i are 1 regardless of the input carry. P_i is associated with the propagation of carry from C_i to $C_i + 1$.

The carry output Boolean function of each stage in a 4 stage carry look-ahead adder can be expressed as

$$C_1 = G_0 + P_0 C_{in}$$
$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_{in}$$
$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in}$$
$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$$

Generally, we perform many mathematical operations in our daily life such as addition, subtraction, multiplication, division, and so on. Let us consider the multiplication process that can be performed in different methods. Different types of algorithms can be used to perform multiplication like grid multiplication method, long multiplication, lattice multiplication, peasant or binary multiplication, and soon.

Binary multiplication is usually performed in digital electronics by using an electronic circuit called as binary multiplier. These binary multipliers are implemented using different computer arithmetic techniques. Booth multiplier that works based on booth algorithm is one of the most frequently used binary multipliers.

Booth Algorithm

A Booth multiplication algorithm or Booth algorithm was named after the inventor Andrew Donald Booth. It can be defined as an algorithm or method of multiplying binary numbers in two’s complement notation. It is a simple method to multiply binary numbers in which multiplication is performed with repeated addition operations by following the booth algorithm. Again this booth algorithm for multiplication operation is further modified and hence, named as modified booth algorithm.

Modified Booth's Algorithm:

Booth multiplication algorithm consists of three major steps as shown in the structure of booth algorithm figure that includes generation of partial product called as recoding, reducing the partial product in two rows, and addition that gives final product. For a better understanding of modified booth algorithm & for multiplication, we must know about each block of booth algorithm for multiplication process.

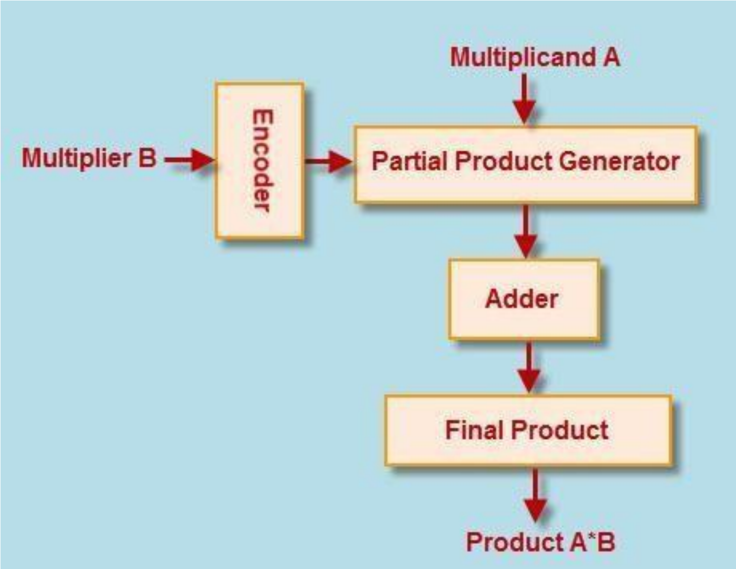


Figure - Modified Booth Algorithm.

.

Modified Booth Algorithm Encoder:

This modified booth multiplier is used to perform high-speed multiplications using modified booth algorithm. This modified booth multiplier’s computation time and the logarithm of the word length of operands are proportional to each other. We can reduce half the number of partial product. Radix-4 booth algorithm used here increases the speed of multiplier and reduces the area of multiplier circuit. In this algorithm, every second column is taken and multiplied by 0 or +1 or +2 or -1 or -2 instead of multiplying with 0 or 1 after shifting and adding of every column of the booth multiplier. Thus, half of the partial product can be reduced using this booth algorithm. Based on the multiplier bits, the process of encoding the multiplicand is performed by radix-4 booth encoder. The overlapping is used for comparing three bits at a time. This grouping is started from least significant bit (LSB), in which only two bits of the booth multiplier are used by the first block and a zero is assumed as third bit as shown in the figure.

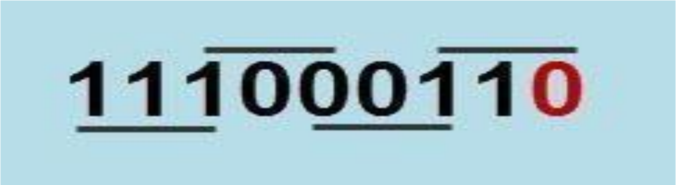
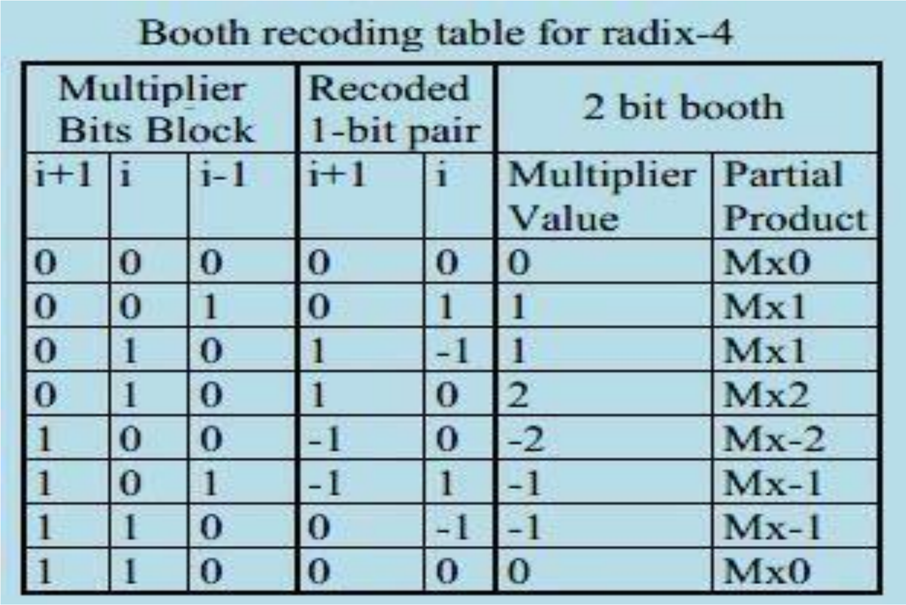


Figure - Bit Pairing as per Booth Recoding

The figure shows the functional operation of the radix-4 booth encoder that consists of eight different types of states. The outcomes or multiplication of multiplicand with 0, -1, and -2 are consecutively obtained during these eight states.



Booth recoding table for radix-4

Multiplier Bits Block			Recoded 1-bit pair		2 bit booth	
i+1	i	i-1	i+1	i	Multiplier Value	Partial Product
0	0	0	0	0	0	Mx0
0	0	1	0	1	1	Mx1
0	1	0	1	-1	1	Mx1
0	1	0	1	0	2	Mx2
1	0	0	-1	0	-2	Mx-2
1	0	1	-1	1	-1	Mx-1
1	1	0	0	-1	-1	Mx-1
1	1	0	0	0	0	Mx0

Figure - Booth Recoding Table for Radix-4

The steps given below represent the radix-4 booth algorithm:

- Extend the sign bit 1 position if necessary to ensure that n iseven.
- Append a 0 to the right of the least significant bit of the boothmultiplier.
- According to the value of each vector, each partial product will be 0, +y, -y, +2y or-2y.

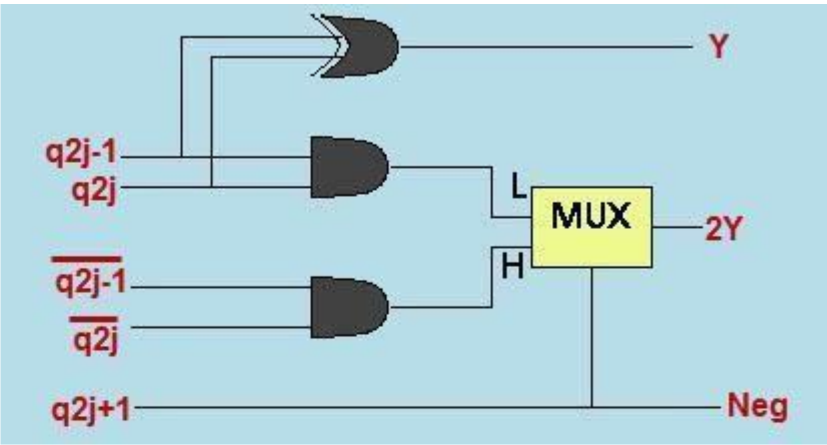


Figure - Booth's Encoder.

Modified booth multiplier's (Z) digits can be defined with the following equation:

The figure shows the modified booth algorithm encoder circuit. Now, the product of any digit of Z with multiplicand Y may be -2y, -y, 0, y, 2y. But, by performing left shift operation at partial products generation stage, 2y may be generated. By taking 1's complement to this 2y, negation is done, and then one is added in appropriate 4-2 compressor. One booth encoder shown in the figure generates three output signals by taking three consecutive bit inputs so as to represent all five possibilities -2X, -X, 0, X,2X.

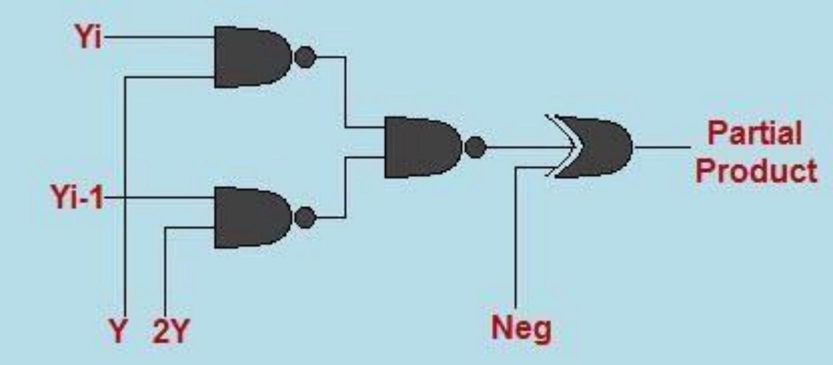
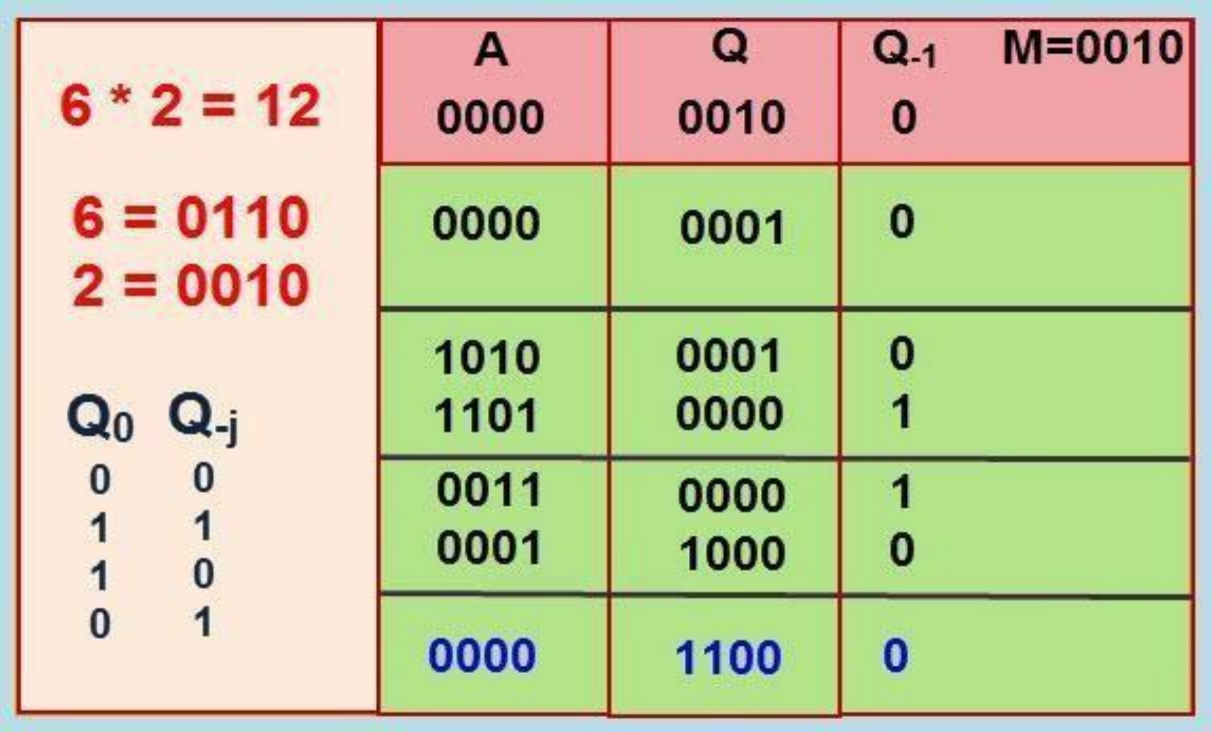


Figure - Partial Product Generator.

If we take the partial product as -2y, -y, 0, y, 2y then, we have to modify the general partial product generator. Now, every partial product point consists of two inputs (consecutive bits) from multiplicand and, based on the requirement, the output will be generated and its complements also generated in case if required. The figure shows the partial product generator circuit. The 2's complement is taken for negative values of y. There are different types of adders such as conventional adders, ripple-carry adders, carry-look-ahead adders, and carry select adders. The carry select adders (CSLA) and carry-look-ahead adders are considered as fastest adders and are frequently used. The multiplication of y is done by after performing shift operation on y – that is – y is shifted to the left by one bit. Hence, to design n-bit parallel multipliers only n2 partial products are generated by using booth algorithm. Thus, the propagation delay to run circuit, complexity of the circuit, and power consumption can be reduced. A simple practical example to understand modified booth algorithm is shown in the figure below.



6 * 2 = 12

6 = 0110
2 = 0010

Q_0	Q_{-j}
0	0
1	1
1	0
0	1

A	Q	Q_{-1}	M=0010
0000	0010	0	
0000	0001	0	
1010 1101	0001 0000	0 1	
0011 0001	0000 1000	1 0	
0000	1100	0	

Figure -Practical Multiplication Example using Modified Booth Algorithm.

Robertson Algorithm

Recall that the `pencil-and-paper' algorithm is in that each product term (obtained by multiplying each bit of the multiplier to the multiplicand) has to be saved till all such product terms are obtained. In machine implementations, it is desirable to add all such product terms to form the partial product. Also, instead of shifting the product terms to the left, the partial product is shifted to the right before the addition takes place. In other words, if Pi is the partial product after i steps and if Y is the multiplicand and X is the multiplier, the and the process repeats

$$P_i \quad P_i + x_j \quad Y$$

and

$$P_{i+1} \quad P_i \quad 2^{1}$$

,…

Note that the multiplication of signed magnitude numbers requires a straightforward extension of the unsigned case. The magnitude part of the product can be computed just as in the unsigned magnitude case. The sign p0 of the product P is computed from the signs of X and Y as p0 x0y0.

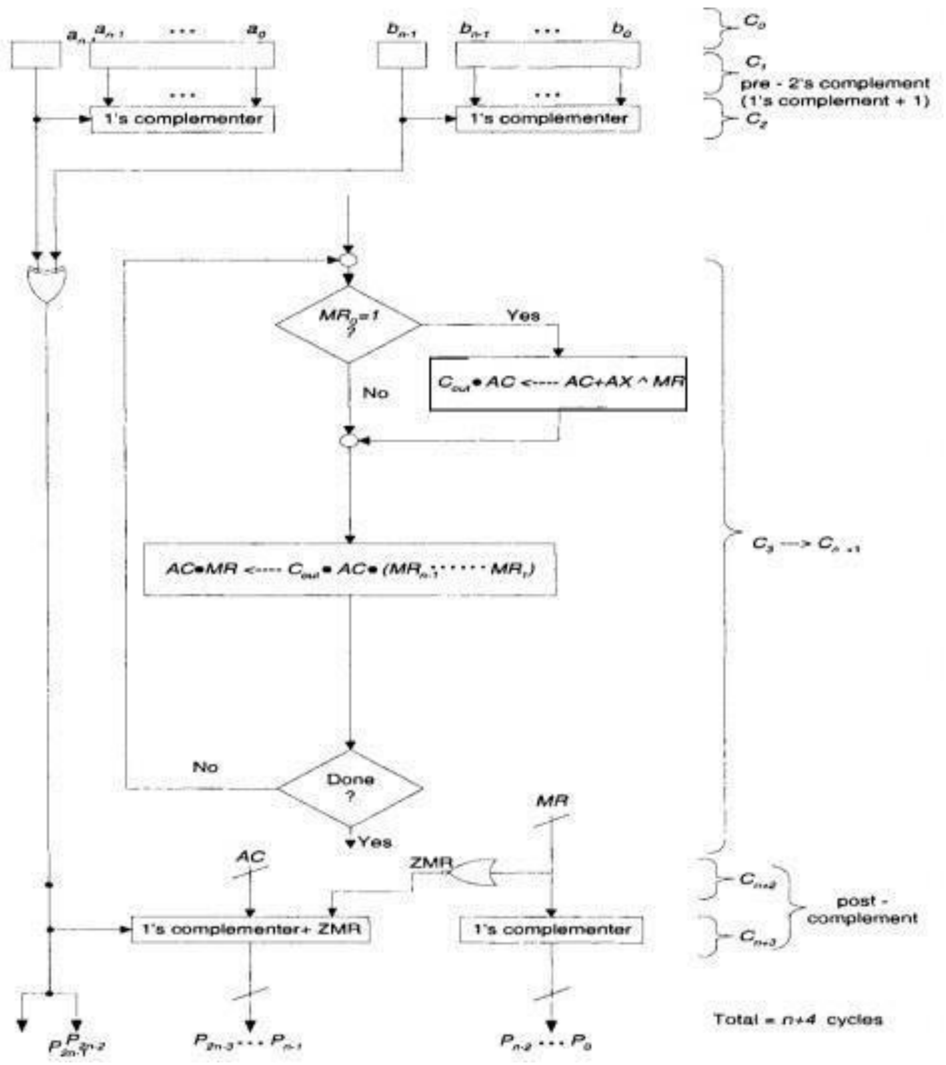


Figure – Two's complement Multiplication - Robertson's Algorithm.

Consider the case that we want to multiply two 8 bit numbers X = x0x1:::x7 and Y = y0y1:::y7. Depending on the sign of the two operands X and Y , there are 4 cases to be considered : x0 = y0 = 0, that is, both X and Y are positive. Hence, multiplication of these numbers is similar to the multiplication of unsigned numbers. In other words, the product P is computed in a series of add-and-shift steps of the form.

$$P_i \quad P_i + x_j \; Y$$
$$P_{i+1} \quad P_i \; 2^{1}$$

Note that all the partial product are non-negative. Hence, leading 0s are introduced during right shift of the partial product. x0 = 0; y0 = 1, that is, X is positive and Y is negative. In this case, the partial product is positive and hence leading 0s are shifted into the partial product until the first 1 in X is encountered. Multiplication of Y by this 1, and addition to the result causes the partial product to be negative, from which point on leading 1s are shifted in (rather than 0s). X0 = 1; y0 = 1, that is, both X and Y are negative. Once again, leading 1s are shifted into the partial product once the first 1 in X is encountered. Also, sinceX is negative, the correction step (subtraction as the last step) is also performed.

Booth's Algorithm

Booth algorithm gives a procedure for **multiplying binary integers** in signed 2's complement representation **in efficient way**, i.e., less number of additions/subtractions required. It operates on the fact that strings of 0's in the multiplier require no addition but just shifting and a string of 1's in the multiplier from bit weight 2^k to weight 2^m can be treated as 2^ (k+1) to 2^m.As in all multiplication schemes, booth algorithm requires examination **of the multiplier bits** and shifting of the partial product. Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial product, or left unchanged according to following rules:

1. The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1's in the multiplier
2. The multiplier is added to the partial product upon encountering the first 0 (provided that there was a previous '1') in a string of 0's in the multiplier.
3. The partial product does not change when the multiplier bit is identical to the previous multiplier bit.

Hardware Implementation of Booths Algorithm – The hardware implementation of booth algorithm requires the register configuration.

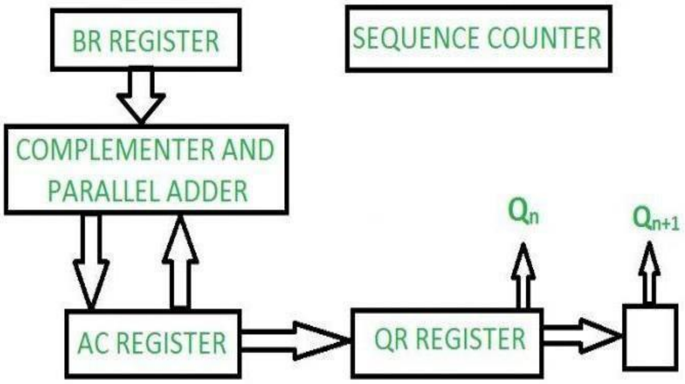


Figure – Register configuration.

We name the register as A, B and Q, AC, BR and QR respectively. Qn designates the least significant bit of multiplier in the register QR. An extra flip-flop Qn+1is appended to QR to facilitate a double inspection of the multiplier. Qn+1 is appended to QR to facilitate a double inspection of the multiplier. The flowchart for booth algorithm is shown below.

Booth's Algorithm Flowchart –

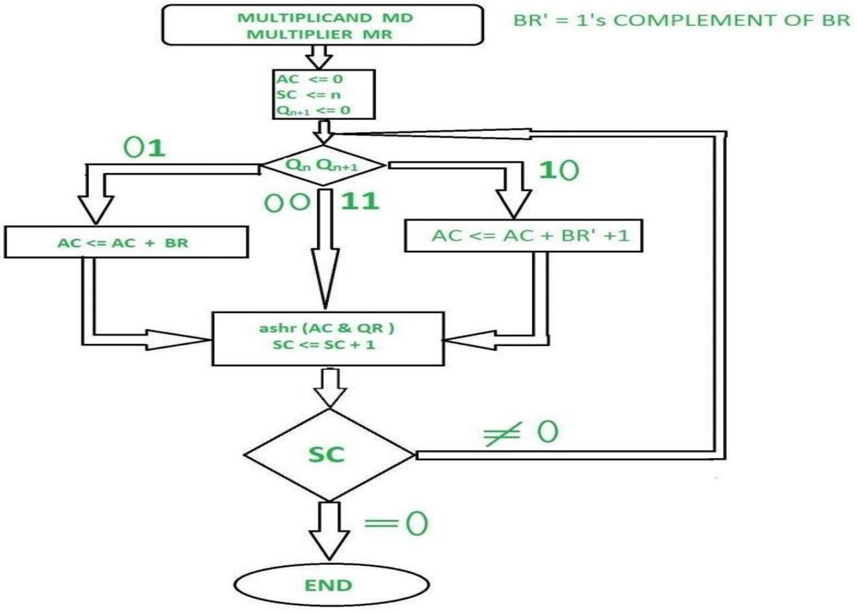


Figure – Booth's Algorithm Flowchart

AC and the appended bit Qn+1 are initially cleared to 0 and the sequence SC is set to a number n equal to the number of bits in the multiplier.

When the two bits are equal, the partial product does not change. An overflow cannot occur because the addition and subtraction of the multiplicand follow each other. As a consequence, the 2 numbers that are added always have a opposite signs, a condition that excludes an overflow. The next step is to shift right the partial product and the multiplier (including Qn+1). This is an arithmetic shift right (ashr) operation which AC and QR ti the right and leaves the sign bit in AC unchanged. The sequence counter is decremented and the computational loop is repeated times.

Example – A numerical example of booth's algorithm is shown below for n = 4. It shows the step by step multiplication of -5 and -7.

MD = -5 = 1011, MD = 1011, MD'+1 = 0101

MR = -7 = 1001

The explanation of first step is as follows: Qn+1

AC = 0000, MR = 1001, Qn+1 = 0, SC = 4

Qn Qn+1 = 10

So, we do AC + (MD)'+1, which gives AC = 0101

On right shifting AC and MR, we get

AC = 0010, MR = 1100 and Qn+1 = 1

OPERATION	AC	MR	Qn+1	SC
	0000	1001	0	4
AC + MD' + 1	0101	1001	0	
ASHR	0010	1100	1	3
AC + MD	1101	1100	1	
ASHR	1110	1110	0	2
ASHR	1111	0111	0	1
AC + MD' + 1	0010	0011	1	0

Product is calculated as follows:

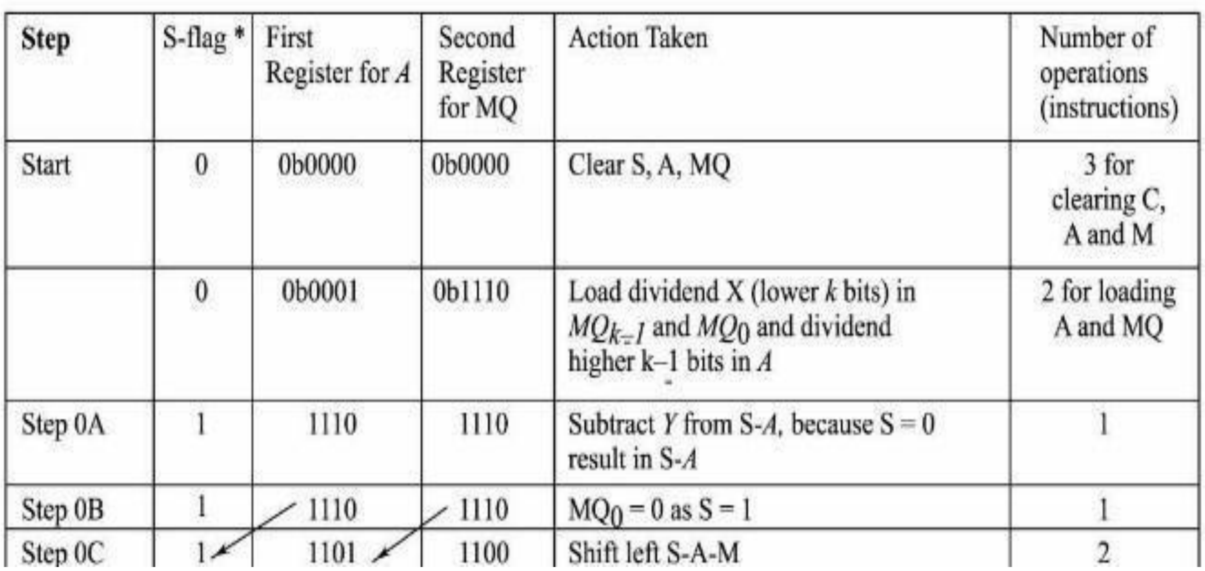
Product = AC MR
Product = 0010 0011 = 35

Division using Non-restoring Algorithm:

- Assume-- that there is an accumulator and MQ register, each of k–bits.
- MQ0, (lbs. of MQ) bit gives the quotient, which is saved after a subtraction or addition.
- Total number of additions or subtractions are k -only and total number of shifts = k plus one addition for restoring remainder if needed.
- Assume ─that X register has (2k − 1) bit for dividend and Y has the k –bit divisor.
- Assume ─ a sign-bit S shows the sign.

1. Load (upper half k − 1 bits of the dividend X) into accumulator k -bit A and load dividend X (lower half bits into the lower k bits at quotient register.
   - Reset sign S=0.
   - Subtract the Kbits divisor Y from S-A (1 plus Kbits) and assign MQ0as perS.

2. If sign of A, S= 0, shift S plus 2k-bit register pair A-MQ left and subtract the k-bits divisor Y rom S-A (1 plus k-bits);
   - Assign MQ0as perS.
3. Repeat step 2 again till the total number of operations =k.
4. If at the last step, the sign of Ain S= 1, then add Y into S-A to leave the correct remainder into A and also assign MQ0as per S, else do-nothing.
5. A has the remainder and MQ has the quotient



Step	S-flag *	First Register for *A*	Second Register for MQ	Action Taken	Number of operations (instructions)
Start	0	0b0000	0b0000	Clear S, A, MQ	3 for clearing C, A and M
	0	0b0001	0b1110	Load dividend X (lower *k* bits) in MQ_{k-1} and MQ_0 and dividend higher k–1 bits in *A*	2 for loading A and MQ
Step 0A	1	1110	1110	Subtract *Y* from S-*A*, because S = 0 result in S-*A*	1
Step 0B	1	1110	1110	MQ_0 = 0 as S = 1	1
Step 0C	1	1101	1100	Shift left S-A-M	2

Coprocessor

History Of Co-Processor:

Co-processor for floating point arithmetic first appeared in desktop computers in 1970s. The coprocessors become common in 1980s and into the early 1990s. Early 8_Bit and 16 Bit processor uses software to carry out the floating point arithmetic operations. Math co-processor was popular purchase for users of computer-aided design (CAD) software and scientific and engineering calculations.

Operation Performed by Coprocessor

- Floating point arithmetic
- Graphic & Signal processing.
- String processing.
- Encryption
- Coprocessors are Unable to fetch the code from the memory so they work under the controlof main processor.

Architecture of 8087:

INTEL 8087

- Numeric Processor.
- Packed in 40 pin ceramic DIP package.
- Available in 5 MHz, 8MHz, 10MHz versions compatible with 8086, 8088, 80186,80188.
- It adds 68 new instructions to the instruction set of8086.

How it works

- The 8087 instruction may lie interleaved in the 8086 program, but it is the task of 8086 to identify the 8087 instructions from the program, send it to 8087 for further execution & after the completion of execution cycle the result may be referred back to CPU.
- Operation of 8087 does not require any software support from the system software or operating system.

Architecture of 8087

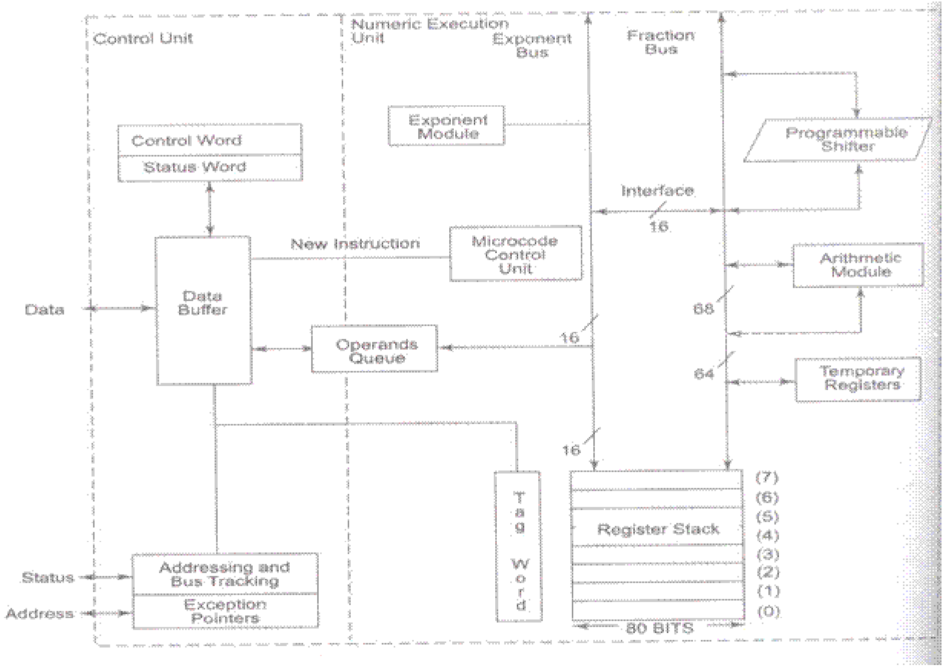


Figure – Architecture of 8087 Explanation

Two major sections:

1) Control unit

2) Numeric Execution unit

Control Unit

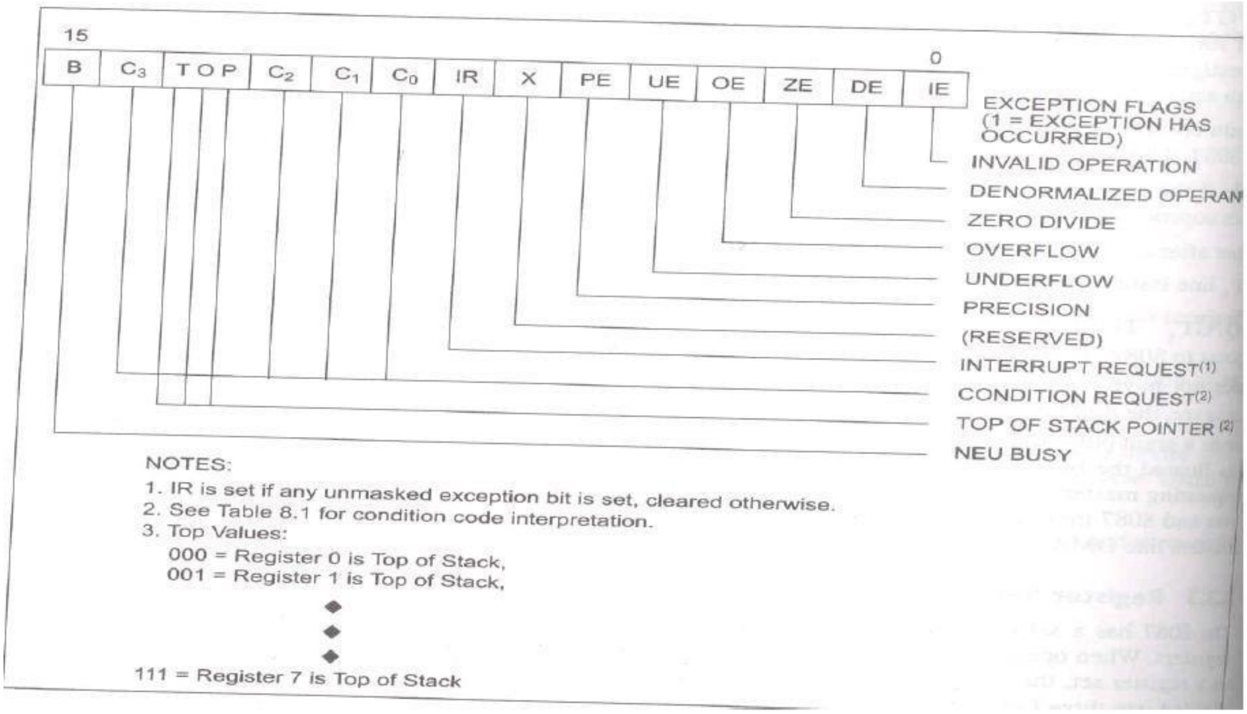
- Function:
- It interfaces the coprocessor to the microprocessor – system data bus.
- Monitors the instruction stream. If the instruction is an Escape (coprocessor) instruction, the coprocessor executes it; if not the microprocessor executes it.
- It receives, decodes instructions, read and write memory operands and executes the8087 instruction

Numeric Execution Unit (NEU)

Functions:

- Execute all the numeric processor instructions.
- It has 8 register (80 bit) stacks that hold the operands for arithmetic instructions & the result.
- Instruction either addresses data in specific stack data – register or uses push and pop mechanism to store or retrieve data.

Control Word Register of 8087



NOTES:
1. IR is set if any unmasked exception bit is set, cleared otherwise.
2. See Table 8.1 for condition code interpretation.
3. Top Values:
   000 = Register 0 is Top of Stack,
   001 = Register 1 is Top of Stack,
   ◆
   ◆
   ◆
   111 = Register 7 is Top of Stack

Fig. 8.13 Status word of 8087

Coprocessor Control Instructions

The coprocessor has control instructions for initialization, exception handling, and task switching.
All control instructions have two forms.

Coprocessor Control Instructions

FINIT/FNINIT

Performs a reset (**initialize**) operation on the arithmetic coprocessor. The coprocessor operates with a Closure of projective (unsigned infinity), rounds to the nearest or even, and uses extended precision when reset or initialized also sets register 0 as the top of the stack.

FSETPM

Changes the coprocessor to the **protected addressing mode** used when the microprocessor is protected mode Protected mode can only be exited by a hardware reset In 80386-Pentium 4, with a change to the control register

FLDCW

Loads the control register with the word addressed by the operand.

FSTCW
Stores the control register into the word-sized memory operand.

FSTSW AX
Copies the contents of the control register to the AX register. not available to 8087

FCLEX
Clears the error flags in the status register and also the busy flag.

Graphics Coprocessor

Noun a high-speed display adapter that is dedicated to graphics operations such as line drawing and plotting. A coprocessor utilized to accelerate the displaying of graphics, significantly speeding up the updating of the images on a screen, and freeing the CPU to take care of other tasks. A graphics coprocessor maybe incorporated into a graphics accelerator, or may be part of a separate subsystem. Also called graphics processor.

Nano Programming

Generally In micro programmed processors, an instruction fetched from memory is interpreted by a micro program stored in a single control memory CM; whereas in other micro programmed processors, the micro instructions are not directly used by the decoder to generate control signals. This is achieved by the use of a second control memory called a Nano control memory (nCM). So now there are two levels of control memories, a higher level control memory is known as micro control memory (µCM) and a lower level control memory is known as Nano control memory (nCM). This is shown in Figure. Thus a microinstruction is in primary control-store memory, it then has the control signals generated for each microinstruction using a secondary control store memory The output word from the secondary memory is called Nano instruction. The µCM stores micro instructions whereas nCM stores nano instructions. The decoder uses Nano instructions from nCM to generate control signals. Thus Nano programming gives an alternative strategy to generate control signals. The process of generation of control signals using nano instructions is shown in Figure

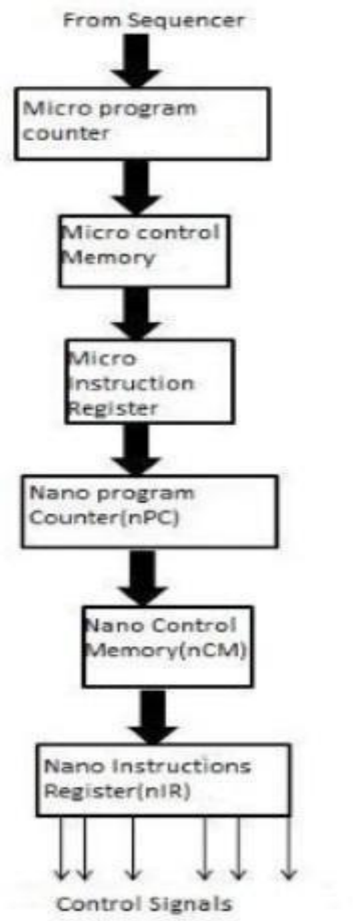


Figure - Nano Programming.

Nano instruction addresses are generated by a nano program counter and nano instructions are placed in a register nIR. The next address of nIR is directly obtained. The next address is generated by either incrementing the nano program counter or loading it from external source (branch field or address from micro instruction opcode)

Advantages of Nano programming

1. Reduces total size of required control memory

In two level control design technique, the total control memory size S2can be calculated as

$S2=HmxWm+HnxWn$

Where $H-mn$ represents the number of words in the high level memory

Wm

represents the size of word in the high level memory

Hn represents the number of words in the low level memory

Wn represents the size of word in the low level memory

Usually, the micro programs are vertically organized so Hm is large and Wm is small. In Nano programming, we have a highly parallel horizontal organization, which makes Wn large and Hn is small. This gives the compatible size for single level control unit as *S*1=*Hmx* Wn which is larger than *S*2. The reduced size of control memory reduces the total chip area.

2. Greater design flexibility

Because of two level memories organization more design flexibility exists between instructions and hardware.

Disadvantage of Nano programming

1. Increased memory access time:

The main disadvantage of the two level memory approaches is the loss of speed due to the extra memory access required for Nano control memory.

Modified booth‘s Algorithm

Generally, we perform many mathematical operations in our daily life such as addition, subtraction, multiplication, division, and so on. Let us consider the multiplication process that can be performed in different methods. Different types of algorithms can be used to perform multiplication like grid multiplication method, long multiplication, lattice multiplication, peasant or binary multiplication, and soon.

Binary multiplication is usually performed in digital electronics by using an electronic circuit called as binary multiplier. These binary multipliers are implemented using different computer arithmetic techniques. Booth multiplier that works based on booth algorithm is one of the most frequently used binary multipliers.

Booth Algorithm

A Booth multiplication algorithm or Booth algorithm was named after the inventor Andrew Donald Booth. It can be defined as an algorithm or method of multiplying binary numbers in two’s complement notation. It is a simple method to multiply binary numbers in which multiplication is performed with repeated addition operations by following the booth algorithm. Again this booth algorithm for multiplication operation is further modified and hence, named as modified booth algorithm.

Modified Booth Algorithm

Booth multiplication algorithm consists of three major steps as shown in the structure of booth algorithm figure that includes generation of partial product called as recoding, reducing the partial product in two rows, and addition that gives final product. For a better understanding of modified booth algorithm & for multiplication, we must know about each block of booth algorithm for multiplication process.

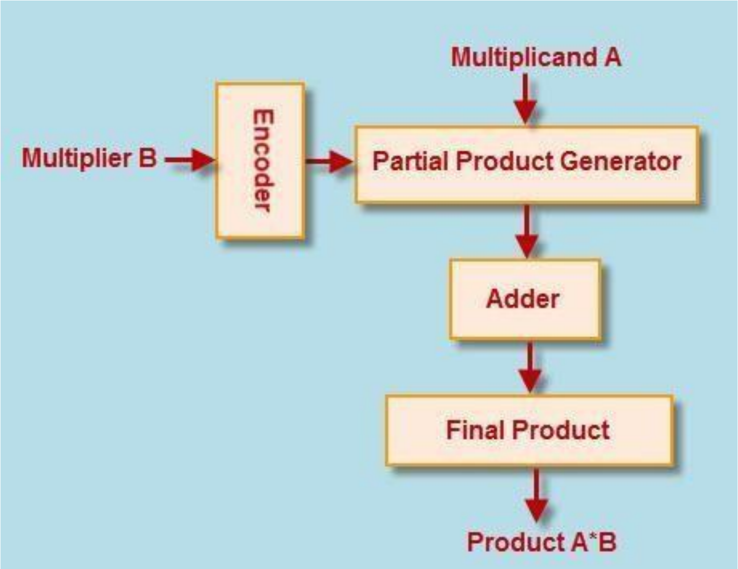


Figure - Modified Booth Algorithm.

.

Modified Booth Algorithm Encoder

This modified booth multiplier is used to perform high-speed multiplications using modified booth algorithm. This modified booth multiplier's computation time and the logarithm of the word length of operands are proportional to each other. We can reduce half the number of partial product. Radix-4 booth algorithm used here increases the speed of multiplier and reduces the area of multiplier circuit. In this algorithm, every second column is taken and multiplied by 0 or +1 or +2 or -1 or -2 instead of multiplying with 0 or 1 after shifting and adding of every column of the booth multiplier. Thus, half of the partial product can be reduced using this booth algorithm. Based on the multiplier bits, the process of encoding the multiplicand is performed by radix-4 booth encoder.

The overlapping is used for comparing three bits at a time. This grouping is started from least significant bit (LSB), in which only two bits of the booth multiplier are used by the first block and a zero is assumed as third bit as shown in the figure.

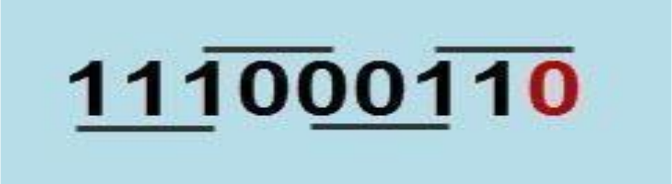
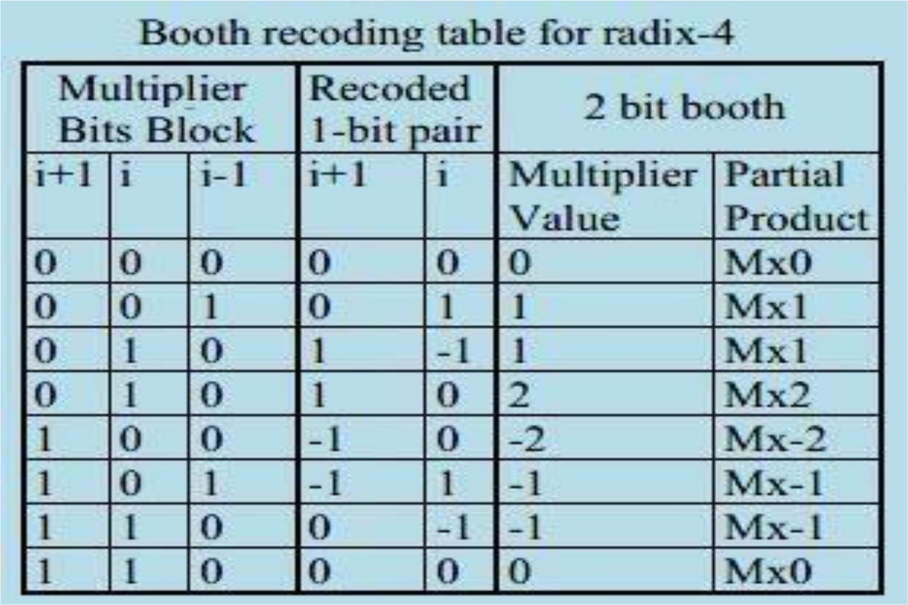


Figure - Bit Pairing as per Booth Recoding

The figure shows the functional operation of the radix-4 booth encoder that consists of eight different types of states. The outcomes or multiplication of multiplicand with 0, -1, and -2 are consecutively obtained during these eight states.



Booth recoding table for radix-4

Multiplier Bits Block			Recoded 1-bit pair		2 bit booth	
i+1	i	i-1	i+1	i	Multiplier Value	Partial Product
0	0	0	0	0	0	Mx0
0	0	1	0	1	1	Mx1
0	1	0	1	-1	1	Mx1
0	1	0	1	0	2	Mx2
1	0	0	-1	0	-2	Mx-2
1	0	1	-1	1	-1	Mx-1
1	1	0	0	-1	-1	Mx-1
1	1	0	0	0	0	Mx0

Figure - Booth Recoding Table for Radix-4

The steps given below represent the radix-4 booth algorithm:

- Extend the sign bit 1 position if necessary to ensure that n is even.
- Append a 0 to the right of the least significant bit of the booth multiplier.
- According to the value of each vector, each partial product will be 0, +y, -y, +2y or-2y.

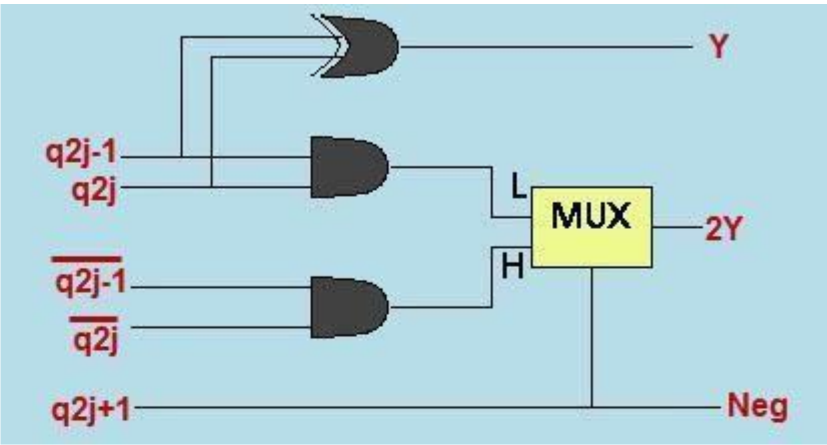


Figure - Booth's Encoder.

Modified booth multiplier’s (Z) digits can be defined with the following equation:

The figure shows the modified booth algorithm encoder circuit. Now, the product of any digit of Z with multiplicand Y may be -2y, -y, 0, y, 2y. But, by performing left shift operation at partial products generation stage, 2y may be generated. By taking 1’s complement to this 2y, negation is done, and then one is added in appropriate 4-2 compressor. One booth encoder shown in the figure generates three output signals by taking three consecutive bit inputs so as to represent all five possibilities -2X, -X, 0, X,2X.

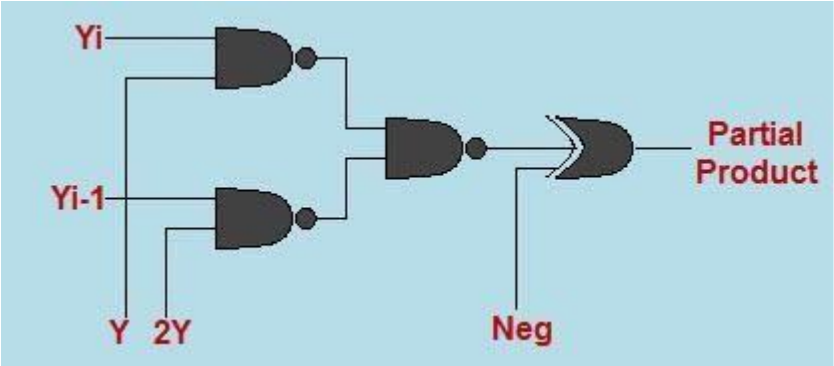
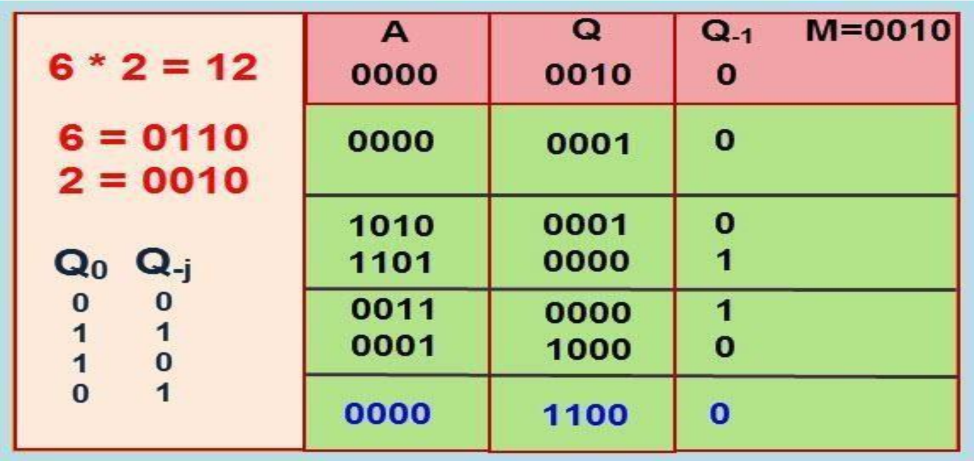


Figure - Partial Product Generator.

If we take the partial product as -2y, -y, 0, y, 2y then, we have to modify the general partial product generator. Now, every partial product point consists of two inputs (consecutive bits) from multiplicand and, based on the requirement, the output will be generated and its complements also generated in case if required. The figure shows the partial product generator circuit.

The 2’s complement is taken for negative values of y. There are different types of adders such as conventional adders, ripple-carry adders, carry-look-ahead adders, and carry select adders. The carry select adders (CSLA) and carry-look-ahead adders are considered as fastest adders and are frequently used. The multiplication of y is done by after performing shift operation on y – that is – y is shifted to the left by one bit. Hence, to design n-bit parallel multipliers only n2 partial products are generated by using booth algorithm. Thus, the propagation delay to run circuit, complexity of the circuit, and power consumption can be reduced. A simple practical example to understand modified booth algorithm is shown in the figure below.



6 * 2 = 12

6 = 0110
2 = 0010

Q_0	Q_{-j}
0	0
1	1
1	0
0	1

A	Q	Q_{-1}	M=0010
0000	0010	0	
0000	0001	0	
1010 1101	0001 0000	0 1	
0011 0001	0000 1000	1 0	
0000	1100	0	

Figure -Practical Multiplication Example using Modified Booth Algorithm.

UNIT-III
CONTROL DESIGN

Hardwired control:

There are two major types of control organization: hardwired control and micro programmed control. In the hardwired organization, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits. It has the advantage that it can be optimized to produce a fast mode of operation. In the micro programmed organization, the control information is stored in a control memory.

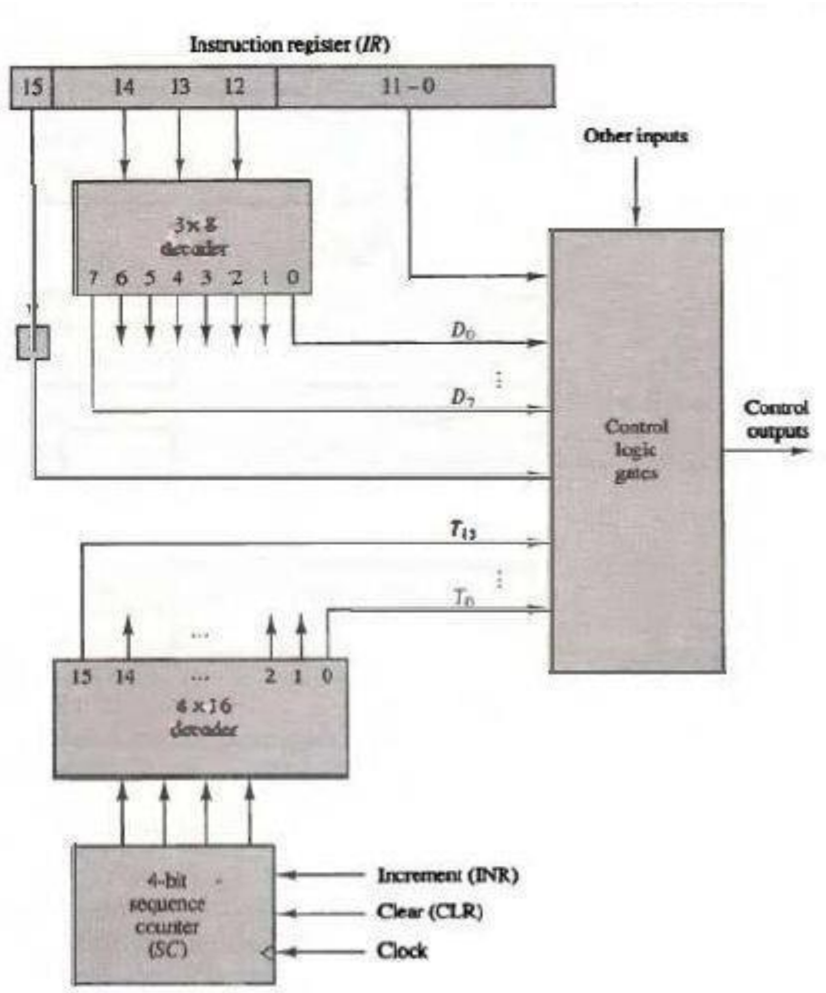
The control memory is programmed to initiate the required sequence of micro operations. A hardwired control, as the name implies, requires changes in the wiring among the various components if the design has to be modified or changed. In the micro programmed control, any required changes or modifications can be done by updating the micro program in control memory.

The block diagram of the control unit is shown in Fig. 5-6. It consists of two decoders, a sequence counter, and a number of control logic gates. An instruction read from memory is placed in the instruction register (IR). The position of this register in the common bus system is indicated. The instruction register is shown again in Fig. 5-6, where it is divided into three parts: the I bit, the operation code, and bits 0 through1.

The operation code in bits 12 through 14 are decoded with a 3 x 8 decoder. The eight outputs of the decoder are designated by the symbols D0 through D7• The subscripted decimal number is equivalent to the binary value of the corresponding operation code. Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I. Bits 0 through 11 are applied to the control logic gates. The 4-bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T0 through T15• The internal logic of the control gates will be derived later when we consider the design of the computer in detail. The sequence counter SC can be incremented or cleared synchronously (see the counter of Fig. 2-11). Most of the time, the counter is incremented to provide the sequence of timing signals out of the 4 x 16 decoder. Once in awhile, the counter is cleared to 0, causing the next active timing signal to be To. As an example, consider the case where SC is incremented to provide timing signals T0, Tv T2, T3, and T4 in sequence.

At time T4, SC is cleared to 0 if decoder output D3 is active. This is expressed symbolically by the statement D3T4: SC <- 0 The timing diagram of Fig. 5-7 shows the time relationship of the control signals. The sequence counter SC responds to the positive transition of the clock. Initially, the CLR input of SC is active. The first positive transition of the

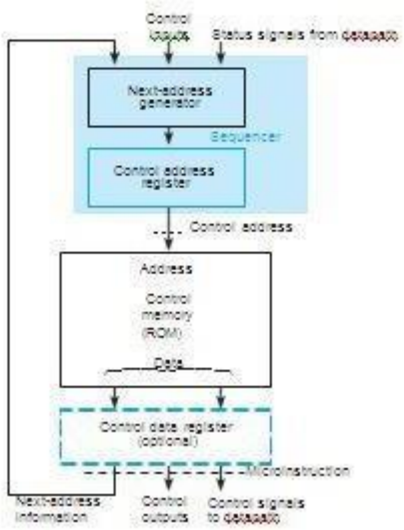
Clock clears SC to 0, which in term activates the timing signal T0 out of the decoder. T0isactiveduringonedockcyde. The positive dock translation abele T0 in the diagram will trigger only those registers whose control inputs are connected to timing signal To. SC is incremented with every positive dod< transition, unless its O.R input is active. This produces the sequence of timing signals To. T,, T:z, T,.. T., and so on, as shown in the diagram. (Note the relationship between the timing sigN) and its corresponding positive dod< transition.)If SC is not cleared, the timing signals wiD continue witli T, T., up to T15 and back toT.



Micro programmed Control:

A control unit with its binary control values stored as words in memory is called a *micro programmed control*. Each word in the control memory contains a *microin- struction* that specifies one or more micro operations for the system. A sequence of microinstructions constitutes a *micro program*. The latter is often fixed at the time of the system design and so is usually stored in ROM. Microprogramming involves placing some representation for combinations of values of control variables in words of ROM for use by the rest of the control logic via successive read operations. The contents of a word in ROM at a given address specify the micro operations to be performed for both the data path and the control unit. A micro program can also be stored in RAM. In this case, it is loaded initially at system startup from the computer console or fromsomeformofnonvolatilestorage,suchasamagneticdisk.WitheitherROMorRAM,thememoryin the control unit is called *control memory*; if RAM is used, the memory is referred to as *writable control memory*.

Figure 1 shows the general configuration of a micro programmed control. The control memory is assumed to be a ROM within which all control information is permanently stored. The *control address register* (*CAR*) specifies the address of the microinstruction. The *control data register (CDR)*, which is optional, may hold the microinstruction currently being executed by the data path and the control unit.



One of the functions of the control word is to determine the address of the next microinstruction to be executed. This microinstruction may be the next one in sequence, or it may be located somewhere else in the control memory. Therefore, one or more bits that specify how to determine the address of the next microin- struction must be present in the current microinstruction. The next address may also be a function of status and external control inputs. While a microinstruction is being executed, the next-address generator produces the next address. This address is transferred to the CAR on the next clock pulse and is used to read the next microinstruction to be executed from ROM. Thus, the microinstructions contain bits for activating microoperations in the data path and bits that specify the sequence of microinstructions executed.

The next-address generator, in combination with the CAR, is sometimes called a microprogram sequencer, as it determines the sequence of instructions that is read from control memory. The address of the next microinstruction can be specified in several ways, depending on the sequencer inputs. Typical functions of a microprogram sequencer are incrementing the CAR by one and loading the CAR. Possible sources for the load operation include an address from control memory, tion.

The CDR holds the present microinstruction while the next address is being computed and the next microinstruction is being read from memory. The CDR breaks up a long combinational delay path through the control memory and the data path. Insertion of this register is just like inserting a pipeline platform, as in Section 7-11; it allows the system to use a higher clock frequency and hence per- form processing faster. The inclusion of a CDR in a system, however, complicates the sequencing of microinstructions, particularly when decision making based on status bits is involved. Hence, for simplicity, we omit the CDR and take the micro- instructions directly from the ROM outputs. The ROM operates as a combinational circuit, with the address as the input and the corresponding microinstruction as the output. The contents of the specified word in ROM remain on the output lines of the ROM as long as the address value is applied to the inputs. No read/write signal is needed, as it is with RAM. Each clock pulse executes the microoperations specified by the microinstruction and also transfers a new address to the CAR, which, in this case, is the only component in the control that receives clock pulses and stores state information. The next-address generator and the control memory are combinational circuits. Thus, the state of the control unit is given by the contents of the CAR.

The status bits enter the next-address generator and affect the determination of the next state. Unless the status bits bypass the control unit and directly control the microoperations being executed in the data path, they can do no more than select the next micro operation by affecting the address generated by the next- address generator. This has a profound effect on the structure of the ASM charts for micro programmed controls. The sequential circuits must be Moore-type sequential circuits, and as a consequence, conditional output boxes are not permit- ted in the ASM charts. This often means that more states will be required in the ASM for a given hardware algorithm. An ASM chart for the binary multiplier, developed under the restriction that the system contain no conditional output boxes, is given in Figure 2. Compared to the ASM chart in Figure 8-7 in the text, this chart has two more states, INIT and ADD, that have been added where originally conditional output boxes were used. Besides being a Moore-type circuit, this ASM has only single decision boxes determining the sequencing between states. Although next-state decisions based on multiple values are possible, they are often excluded in simpler next-address generator designs.

Pipe line control:

Suppose you wanted to make an automobile from scratch. You might gather up the raw materials, form the metal into recognizable shapes, cast some of the metal into an engine block, connect up fuel lines, wires, etc., to eventually (one would hope) make a workable automobile. To do this, you would need many skills - all the skills of the artisans that make autos, and management skills in addition to being an electrician and a metallurgist. This would not be an efficient way to make a car, but would definitely provide many challenges.

That is the way a multi cycle data path works - it is designed to do everything - input, output, and computation (recall the fetch-decode-execute sequence). We need to ask ourselves if this is really the best way to compute efficiently, especially when we consider the complexity of control for large (CISC) systems or even smaller RISC processors.

Fortunately, our analogy with car-making is not so far-fetched, and can actually help us arrive at a more efficient processor design. Consider the modern way of making cars - on an *assembly line*. Here, there is an orderly flow of parts down a conveyor belt, and the parts are processed by different stations (also called *segments* of the assembly line). Each segment does one thing, over and over. The segments are coordinated to exploit the *sequentiality* inherent in the automobile assembly process. The work gets done more smoothly (because of the orderly flow of input parts and output results), more efficiently (because each assembler at each segment of the pipeline does his or her task at what one hopes is maximum efficiency), and more reliably because there is greater consistencyinonetaskbeingdonerepetitively(providedtheassemblylineisdesignedcorrectly).

A similar analogy exists for computers. Instead of a multi cycle data path with its complex control system that walks, talks, cries, and computes - let us suppose that we could build an *assembly line for computing*. Such objects actually exist, and they are called *pipeline processors*. They have sequentially-arranged stages or segments, each of which perform a specific task in a fixed amount of time. Data flows through these pipelines like cars through an assembly line.

Pipeline Data path Design and Implementation

The work involved in an instruction can be partitioned into steps labeled IF (Instruction Fetch), ID (Instruction Decode and data fetch), EX (ALU operations or R-format execution), MEM (Memory operations), and WB (Write-Back to register file). We next discuss how this sequence of steps can be implemented in terms of MIPS instructions.

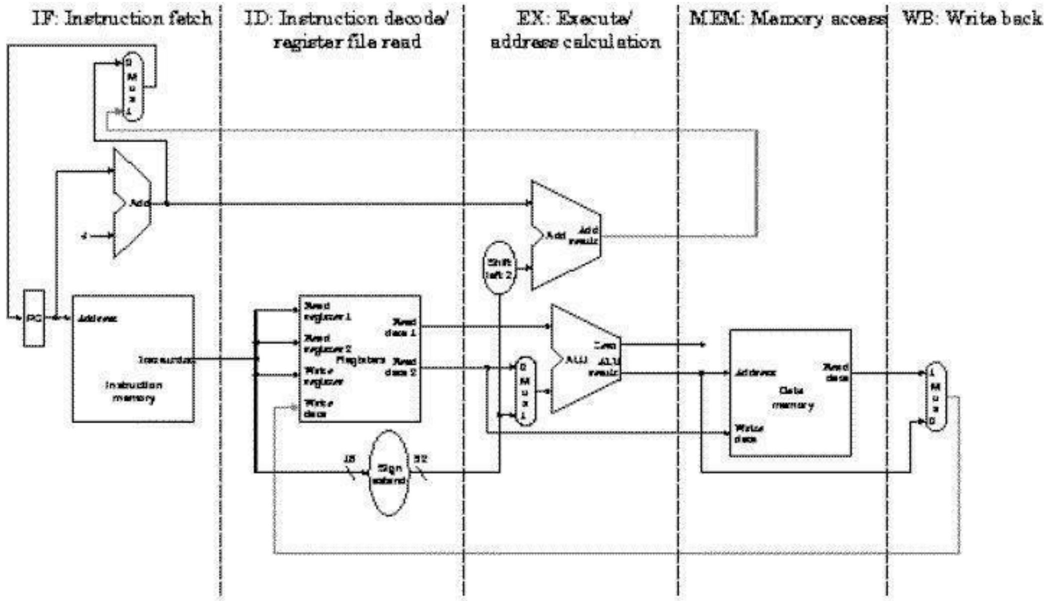
MIPS Instructions and Pipelining

In order to implement MIPS instructions effectively on a pipeline processor, we must ensure that the instructions are the same length (simplicity favors regularity) for easy IF and ID, similar to the multi cycle datapath. We also need to have few but consistent instruction formats, to avoid deciphering variable formats during IF and ID, which would prohibitively increase pipeline segment complexity for those tasks. Thus, the register indices should be in the same place in each instruction. In practice, this means that the rd, rs, and rt fields of the MIPS instruction must not change location across all MIPS pipeline instructions.

Additionally, we want to have instruction decoding and reading of the register contents occur at the same time, which is supported by the datapath architecture that we have designed thus far. Observe that we have memory address computation in the lw and sw instructions only, and that these are the only instructions in our five-instruction MIPS subset that perform memory operations. As before, we assume that operands are aligned in memory, for straightforward access.

Data path partitioning for Pipelining

Recall the single-cycle data path, which can be partitioned (subdivided) into functional units as shown in Figure 5.2. Because the single-cycle data path contains separate Instruction Memory and Data Memory units, this allows us to directly implement in hardware the IF-ID-EX-MEM-WB representation of the MIPS instruction sequence. Observe that several control lines have been added, for example, to route data from the ALU output (or memory output) to the register file for writing. Also, there are again three ALUs, one for ALU op, another for JTA computation, and a third for adding PC+4 to compute the address of the next instruction.



Pipeline Control Issues and Hardware:

Observe that there is nothing to control during instruction fetch and decode (IF and ID). Thus, we can begin our control activities (initialization of control signals) during ID, since control will only be exerted during EX, MEM, and WB stages of the pipeline. Recalling that the various stages of control and buffer circuitry between the pipeline stages are labelled IF/ID, ID/EX, EX/MEM, and MEM/WB, we have the propagation of control

Here, the following stages perform work as specified:

- IF/ID: Initializes control by passing the rs, rd, and rt fields of the instruction, together with the opcode and funct fields, to the control circuitry.
- ID/EX: Buffers control for the EX, MEM, and WB stages, while executing control for the EX stage. Control decides what operands will be input to the ALU, what ALU operation will be performed, and whether or not a branch is to be taken based on the ALU Zerooutput.
- EX/MEM: Buffers control for the MEM and WB stages, while executing control for the MEM stage. The control lines are set for memory read or write, as well as for data selection for memory write. This stage of control also contains the branch control logic.
- MEM/WB: Buffers and executes control for the WB stage, and selects the value to be written into the register file.

Overview of Hazards

Pipeline processors have several problems associated with controlling smooth, efficient execution of instructions on the pipeline. These problems are generally called *hazards*, and include the following three types:

- **Structural Hazards** occur when different instructions collide while trying to access the same piece of hardware in the same segment of a pipeline. This type of hazard can be alleviated by having redundant hardware for the segments wherein the collision occurs. Occasionally, it is possible to insert stalls or reorder instructions to omit this type of hazard.
- **Data Hazards** occur when an instruction depends on the result of a previous instruction still in the pipeline, which result has not yet been computed. The simplest remedy inserts stalls in the execution sequence, which reduces the pipeline's efficiency. The solution to data dependencies is twofold. First, one can *forward* the ALU result to the write back or data fetch stages. Second, in selected instances, it is possible to restructure the code to eliminate some data dependencies. Forwarding paths are shown as thin blue or red lines in Figure5.4.
- **Control Hazards** can result from branch instructions. Here, the branch target address might not be ready in time for the branch to be taken, which results in *stalls* (dead segments) in the pipeline that have to be inserted as local wait events, until processing can resume after the branch target is executed. Control hazards can be mitigated through accurate branch prediction (which is difficult), and by *delayed branch* strategies.

We next examine hazards in detail, and discuss several techniques for eliminating or relieving hazards.

Nano Programming:

Generally In micro programmed processors, an instruction fetched from memory is interpreted by a micro program stored in a single control memory CM; whereas in other micro programmed processors, the micro instructions are not directly used by the decoder to generate control signals. This is achieved by the use of a second control memory called a Nano control memory (nCM). So now there are two levels of control memories, a higher level control memory is known as micro control memory (μCM) and a lower level control memory is known as Nano control memory (nCM). This is shown in Figure. Thus a microinstruction is in primary control-store memory, it then has the control signals generated for each microinstruction using a secondary control store memory The output word from the secondary memory is called Nano instruction. The μCM stores micro instructions whereas nCM stores nano instructions. The decoder uses Nano instructions from nCM to generate control signals. Thus Nano programming gives an alternative strategy to generate control signals. The process of generation of control signals using nano instructions is shown in Figure

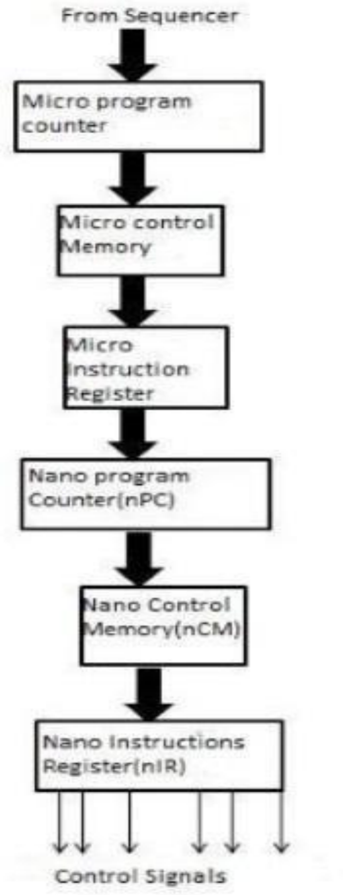


Figure - Nano Programming.

Nano instruction addresses are generated by a nano program counter and nano instructions are placed in a register nIR. The next address of nIR is directly obtained. The next address is generated by either incrementing the nano program counter or loading it from external source (branch field or address from micro instruction opcode)

Advantages of Nano programming

1. Reduces total size of required control memory

In two level control design technique, the total control memory size S2can be calculated as

$S2=HmxWm+HnxWn$

Where $H-mn$ represents the number of words in the high level memory

Wm

represents the size of word in the high level memory

Hn represents the number of words in the low level memory

Wn represents the size of word in the low level memory

Usually, the micro programs are vertically organized so Hm is large and Wm is small. In Nano programming, we have a highly parallel horizontal organization, which makes Wn large and Hnis small. This gives the compatible size for single level control unit as $S1=Hmx$ Wn which is larger than $S2$. The reduced size of control memory reduces the total chip area.

2. Greater design flexibility

Because of two level memories organization more design flexibility exists between instructions and hardware.

Disadvantage of Nano programming

1. Increased memory access time:

The main disadvantage of the two level memory approaches is the loss of speed due to the extra memory access required for Nano control memory.

Superscalar Processor

What is superscalar processor ?

Why Superscalar?

Organization of superscalar processor.

Instruction dispatch.

Reservation station.

Reservation station: Centralized vs distributed.

Recorder buffer.

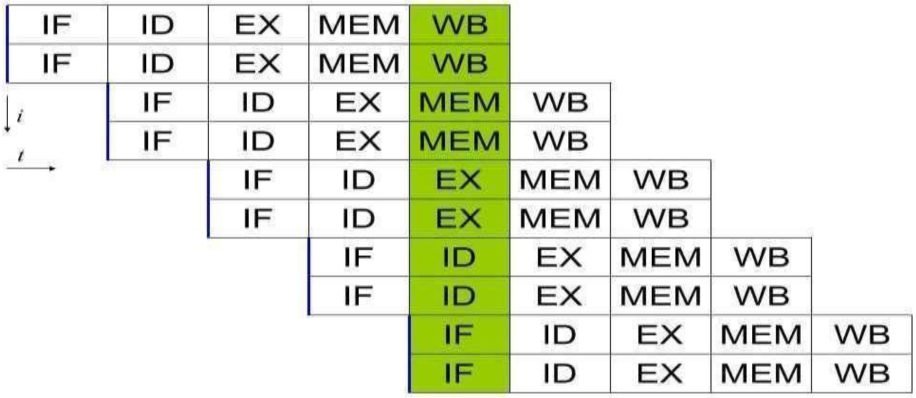
Instruction completion and Retire.

Limitations of superscalar processor.

Superscalar processor:

A superscalar processor is a CPU that implements a form of parallelism called instruction-level parallelism within a single processor.

Simple superscalar pipeline:



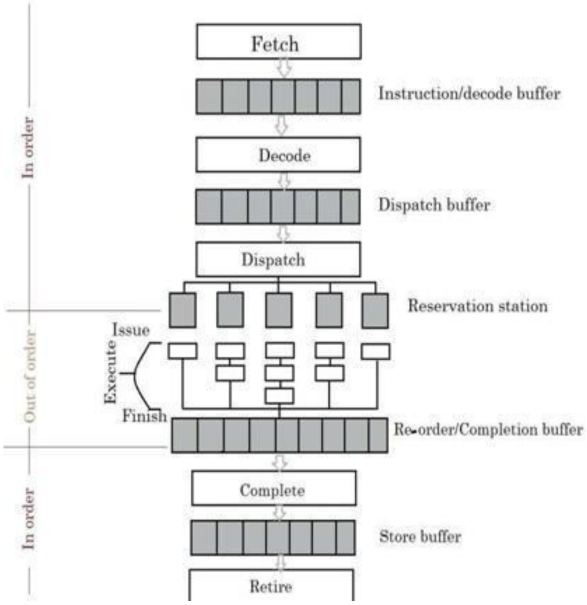
IF	ID	EX	MEM	WB				
IF	ID	EX	MEM	WB				
	IF	ID	EX	MEM	WB			
	IF	ID	EX	MEM	WB			
		IF	ID	EX	MEM	WB		
		IF	ID	EX	MEM	WB		
			IF	ID	EX	MEM	WB	
			IF	ID	EX	MEM	WB	
				IF	ID	EX	MEM	WB
				IF	ID	EX	MEM	WB

By fetching and dispatching two instructions at a time, a maximum of two instructions per cycle can be completed. (IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register write back, i = Instruction number, t = Clock cycle [i.e., time]).

Why superscalar:

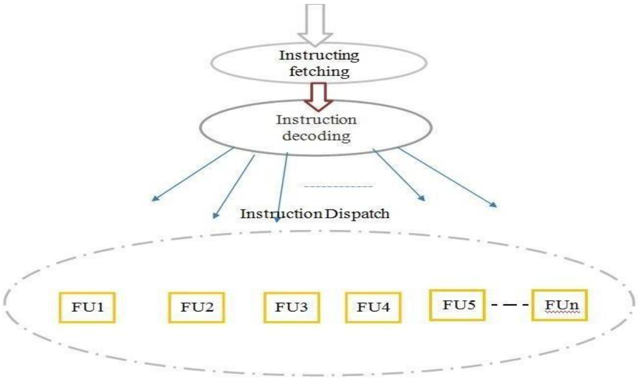
- Most operations are on scalar quantities.
- Improve these operations to get an overall improvement.
- Superscalar processor executes multiple independent instructions in parallel.

Superscalar Organization:



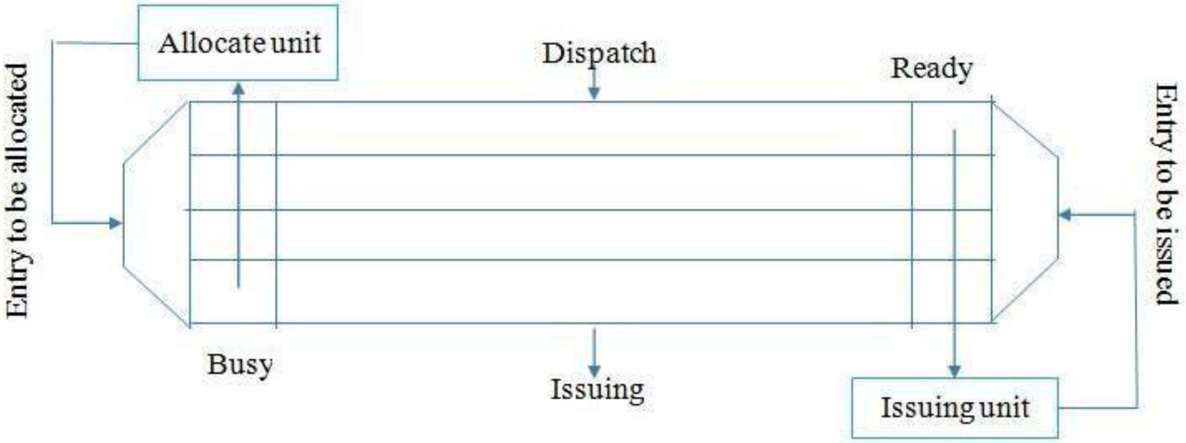
Instruction Dispatch:

Route decoded instructions to appropriate functional units.

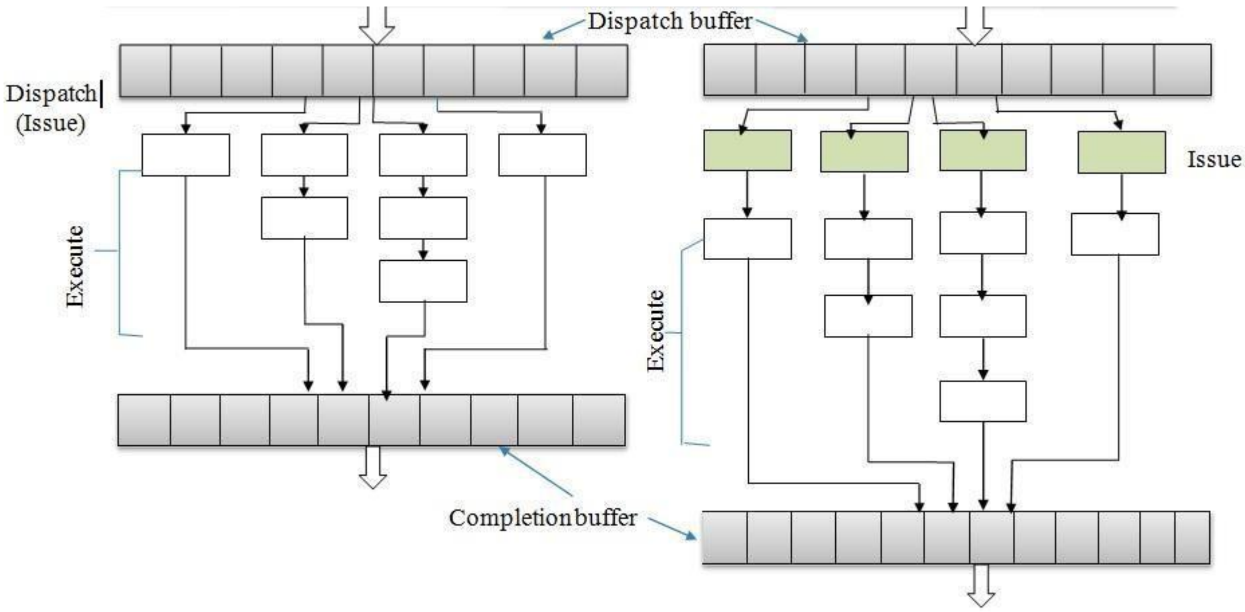


Reservation Station:

- Reservation station decouple instruction decoding and instruction execution.
- Main task: Dispatching -- Waiting--Issuing



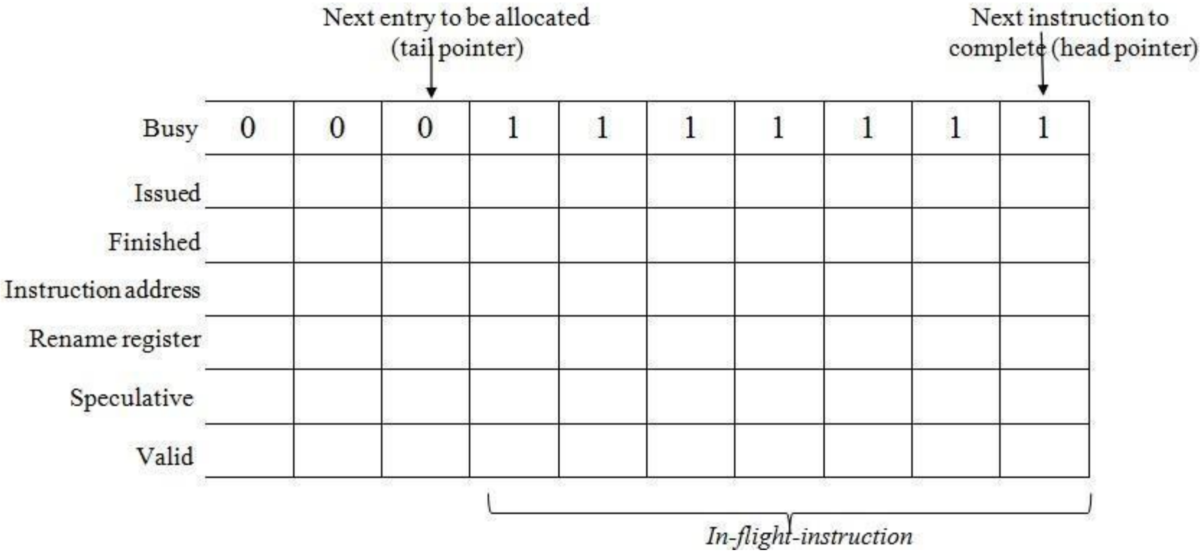
Reservation station: Centralized Vs Distributed



Fig(a): Centralize reservation station (Intel P6)
Fig(b): Distributed reservation station (Power PC 620)

Reorder Buffer:

- Contain all *in–flight* instruction.
- Includes instruction in RS + instruction executing in FUs + instruction which are finished execution but waiting to be completed in program order.
- Only finished and non-speculative instructions can be completed.



Busy	0	0	0	1	1	1	1	1	1	1
Issued										
Finished										
Instruction address										
Rename register										
Speculative										
Valid										

Instruction completion and Retire:

- Completion – finish the execution and update the machine state.
- Retire - update the memory.

- A store may complete by writing to store buffer, but it retire only when the data is written into the memory.
- When an interrupt occurs, stop fetching new instructions and finish the execution of all-in-flight instructions.
- When an exception occurs, the result of the completion may no longer be valid.

Limitation of superscalar processor:

1. Instruction-fetch inefficiencies caused by both branch delays and instruction misalignment
2. Not worthwhile to explore highly- concurrent execution hardware, rather, it is more appropriate to explore economical execution hardware.
3. Degree of intrinsic parallelism in the instruction stream (instructions requiring the same computational resources from theCPU).
4. Complexity and time cost of the dispatcher and associated dependency checking logic.
5. Branch instruction processing.

UNIT-IV
MEMORY ORGANIZATION

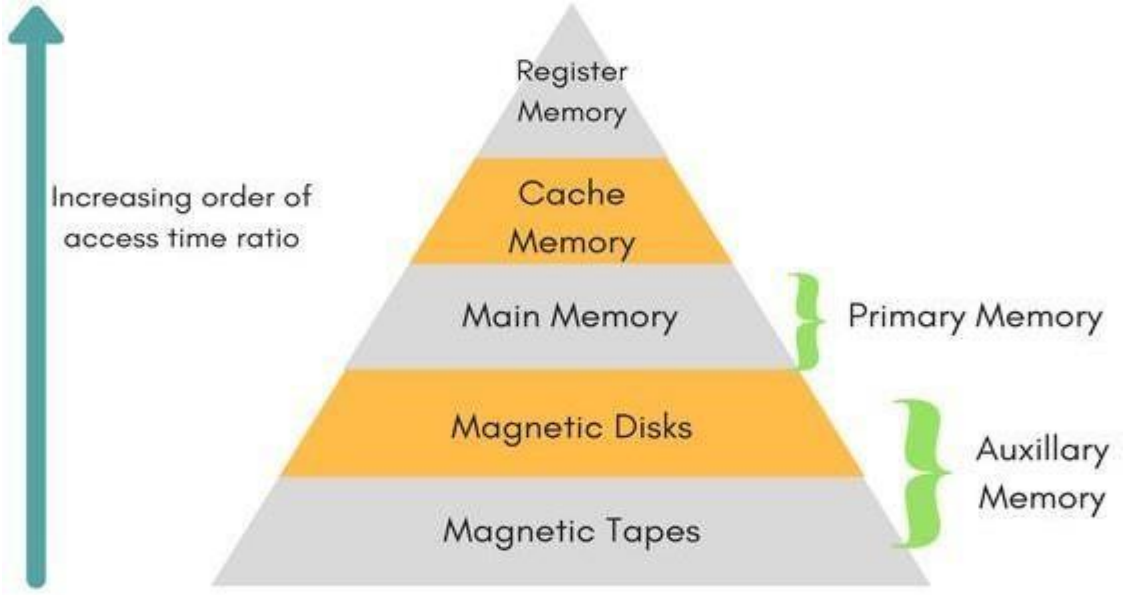
Introduction to Memory:

A memory unit is the collection of storage units or devices together. The memory unit stores the binary information in the form of bits.

Generally, memory/storage is classified into 2 categories:

1. **Volatile Memory**: This loses its data, when power is switchedoff.
2. **Non-Volatile Memory**: This is a permanent storage and does not lose any data when power is switched off.

Memory Hierarchy:



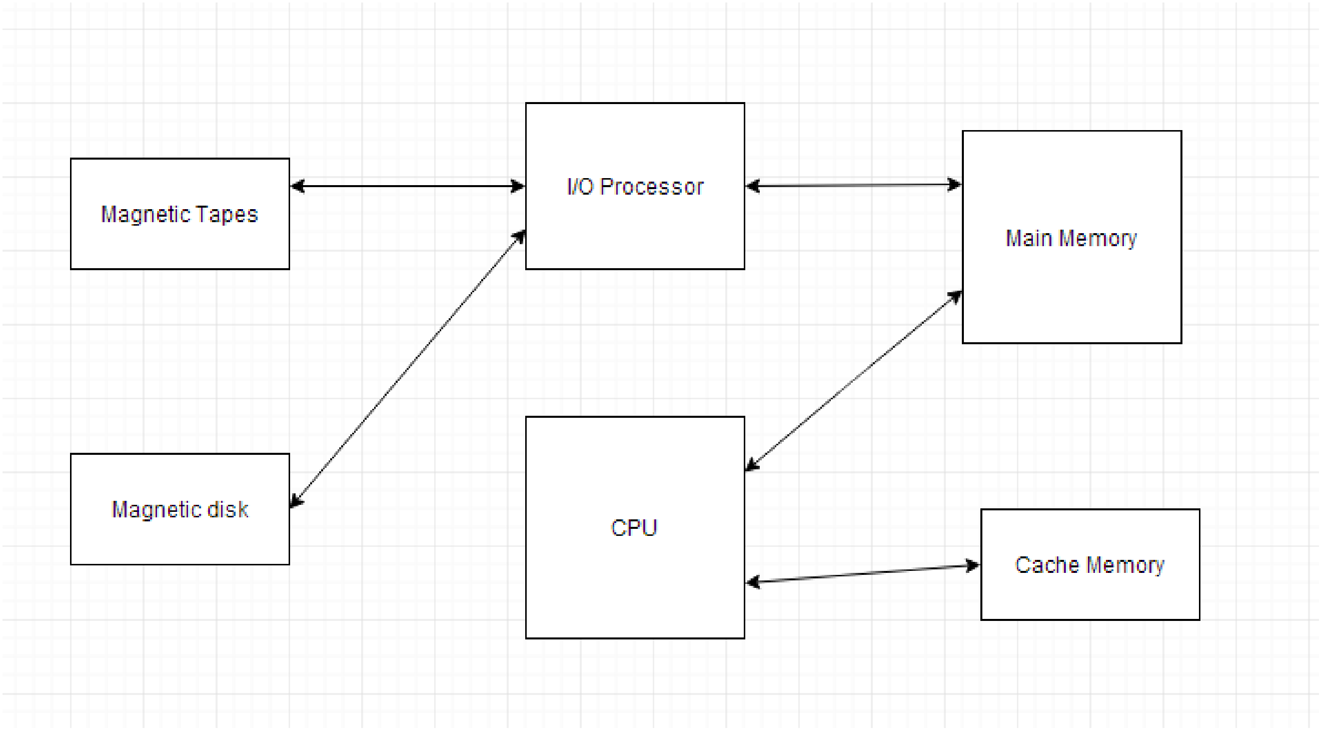
The total memory capacity of a computer can be visualized by hierarchy of components. The memory hierarchy system consists of all storage devices contained in a computer system from the slow Auxiliary Memory to fast Main Memory and to smaller Cache memory.

Auxiliary Memory:-

Auxiliary memory access time is generally **1000 times** that of the main memory, hence it is at the bottom of the hierarchy.

Main Memory:-

The **main memory** occupies the central position because it is equipped to communicate directly with the CPU and with auxiliary memory devices through Input/output processor (I/O). When the program not residing in main memory is needed by the CPU, they are brought in from auxiliary memory. Programs not currently needed in main memory are transferred into auxiliary memory to provide space in main memory for other programs that are currently in use. The **cache memory** is used to store program data which is currently being executed in the CPU. Approximate access time ratio between cache memory and main memory is about **1 to7~10**



Memory Access Methods:-

Each memory is a collection of numerous memory locations. To access data from any memory, first it must be located and then the data is read from the memory location. Following are the methods to access information from memory locations:

1. **Random Access**: Main memories are random access memories, in which each memory location has a unique address. Using this unique address any memory location can be reached in the same amount of time in any order.
2. **Sequential Access:** This methods allows memory access in a sequence or in order.
3. **Direct Access:** In this mode, information is stored in tracks, with each track having a separate read/write head.

Main Memory:-

The memory unit that communicates directly within the CPU, Auxillary memory and Cache memory, is called main memory. It is the central storage unit of the computer system. It is a large and fast memory used to store data during computer operations. Main memory is made up of **RAM** and **ROM**, with RAM integrated circuit chips holing the major share.

1. **RAM:** Random Access Memory
    a) **DRAM**: Dynamic RAM, is made of capacitors and transistors, and must be refreshed every 10~100 ms. It is slower and cheaper than SRAM.
    b) **SRAM**: Static RAM, has a six transistor circuit in each cell and retains data, until powered off.
    c) **NVRAM**: Non-Volatile RAM, retains its data, even when turned off. Example: Flash memory.

2. **ROM**: Read Only Memory, is non-volatile and is more like a permanent storage for information. It also stores the **bootstrap loader** program, to load and start the operating system when computer is turned on.

    Types of ROM

    a) PROM(ProgrammableROM)
    b) EPROM(Erasable PROM)and
    c) EEPROM(Electrically ErasablePROM)

Auxiliary Memory:-

Devices that provide backup storage are called auxiliary memory. **For example:** Magnetic disks and tapes are commonly used auxiliary devices. Other devices used as auxiliary memory are magnetic drums, magnetic bubble memory and optical disks. It is not directly accessible to the CPU, and is accessed using the Input/Output channels.

Cache Memory:-

The data or contents of the main memory that are used again and again by CPU, are stored in the cache memory so that we can easily access that data in shorter time. Whenever the CPU needs to access memory, it first checks the cache memory. If the data is not found in cache memory then the CPU moves onto the main memory. It also transfers block of recent data into the cache and keeps on deleting the old data in cache to accomodate the new one.

Hit Ratio

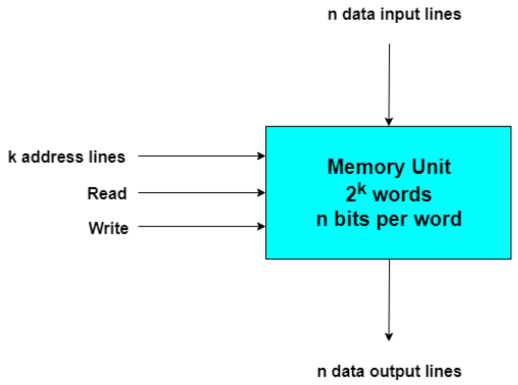
The performance of cache memory is measured in terms of a quantity called **hit ratio**. When the CPU refers to memory and finds the word in cache it is said to produce a **hit**. If the word is not found in cache, it is in main memory then it counts as a **miss**. The ratio of the number of hits to the total CPU references to memory is called hit ratio.

```
Hit Ratio = Hit/(Hit + Miss)
```

Random Access Memory:-

In random-access memory(RAM) the memory cells can be accessed for information transfer from any desired random location. That is, the process of locating a word in memory is the same and requires an equal amount of time no matter where the cells are located physically in memory. Communication between a memory and its environment is achieved through data input and output lines, address selection lines, and control lines that specify the direction of transfer.

A block diagram of a RAM unit is shown below:



The **n** data input lines provide the information to be stored in memory, and the **n** data output lines supply the information coming out of particular word chosen among the **2^k**available inside the memory. The two control inputs specify the direction of transfer desired.

Write and Read Operations:-

The two operations that a random access memory can perform are the **write** and **read** operations. The write signal specifies a transfer-in operation and the read signal specifies a transfer-out operation. On accepting one of these control signals. The internal circuits inside the memory provide the desired function. The steps that must be taken for the purpose of transferring a new word to be stored into memory are as follows:

1. Apply the **binary address** of the desired word into the addresslines.
2. Apply the **data bits** that must be stored in memory into the data inputlines.
3. Activate the **write**input.

The memory unit will then take the bits presently available in the input data lines and store them in the specified by the address lines. The steps that must be taken for the purpose of transferring a stored word out of memory are as follows:

1. Apply the **binary address** of the desired word into the addresslines.
2. Activate the **read** input.

The memory unit will then take the bits from the word that has been selected by the address and apply them into the output data lines. The content of the selected word does not change after reading.

Serial Access Memories:-

Sequential access is a process used for retrieving data from a storage device. It is also known as serial access. In sequential access, the storage device moves through all information up to the point it is attempting to read or write. An example of sequential access drive is a tape drive where the drive moves the tape forward or backward until the destination is reached. Sequential access memory can also be called "storage system." The data is stored and read in a sequential fixed order. Sequential access is the type of memory mostly used for permanent storage, whereas, random access memory is used for temporary storage.

Serial Access Devices:-

Old recording media such as CDs, DVDs, and magnetic tapes are examples of sequential access memory drives. Hard drive is also an example of sequential access memory. Examples of random access memory include memory chips and flash memory (such as memory sticks or memory cards).

Difference between Sequential Access and Random Access:-

Comparing sequential versus random disk operations helps to assess systems efficiency. Accessing data sequentially is faster than random operations, because it involves more search functions. The search operation is performed by the right disk cylinder. It occurs when the disk head positions itself to access the data requested for. More ever, random access delivers a lower rate of output. If the disk access is random, it is advisable to pay attention and monitor for the emergence of any bottleneck. For workloads of either random or sequential input/output, it is advisable to use drives with faster rotational speeds. For workloads that are predominantly random input/output, it is advisable to use a drive with faster search time.

Disadvantages of Sequential Access:-

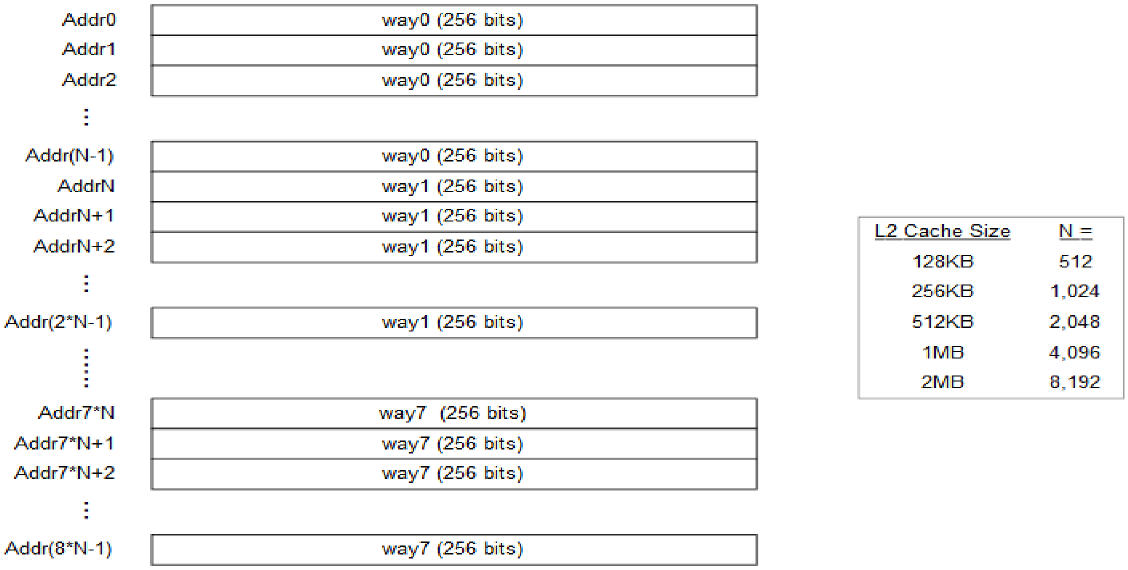
The number of records that are affected when updating a file refers to its hit rate. Let us consider a file with 5000 records; if there is a delete or an update operation affecting only 50 records, then the hit rate is very low. If there are 4500 records that are affected by update or delete operations, then the hit rate is high. Sequential access is found to be slow when the hit rate is low. It is due to the fact that sequential access has to search all the records in a particular order. Moreover, sequential files are executed in a batched transaction to overcome the problem of low hit rate.

RAM interfaces:-

Data RAM:-

The data RAM shown below is organized as 8 ways 256-bit wide contiguous memories. It supports the following accesses:

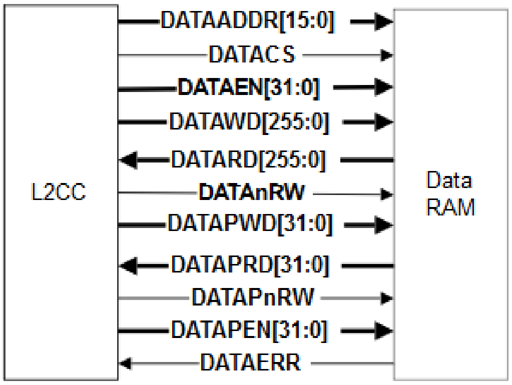
1. 8 word data reads
2. n * 8 bits data writes with byte enables controls
3. 8 word data writes for linefills.



Addr0	way0 (256 bits)
Addr1	way0 (256 bits)
Addr2	way0 (256 bits)
⋮	
Addr(N-1)	way0 (256 bits)
AddrN	way1 (256 bits)
AddrN+1	way1 (256 bits)
AddrN+2	way1 (256 bits)
⋮	
Addr(2*N-1)	way1 (256 bits)
⋮	
Addr7*N	way7 (256 bits)
Addr7*N+1	way7 (256 bits)
Addr7*N+2	way7 (256 bits)
⋮	
Addr(8*N-1)	way7 (256 bits)

L2 Cache Size	N =
128KB	512
256KB	1,024
512KB	2,048
1MB	4,096
2MB	8,192

Data RAM organization:-

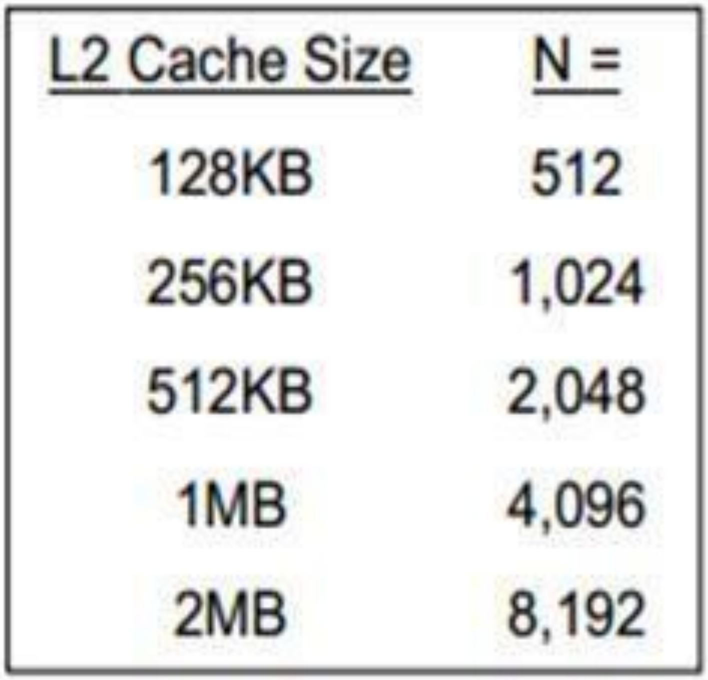


Dirty RAM:-

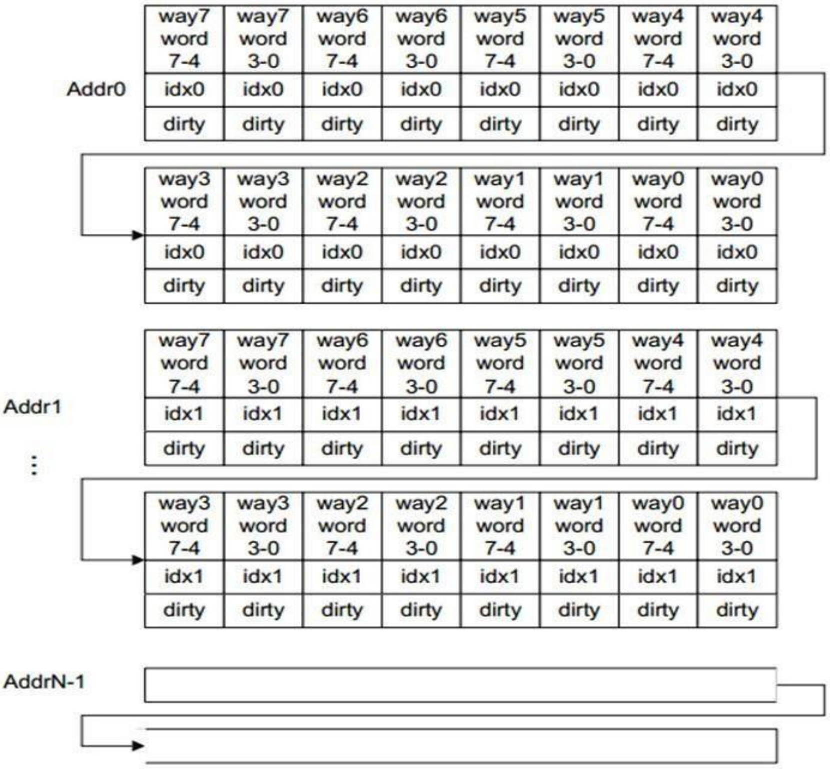
The dirty RAM shown below is organized as a 16-bit wide memory, 2 bits per 8-word cache line. The dirty RAM address is the same as the tag RAM address bus. It supports the following accesses:

- 16 bit dirty reads for write-back eviction on alinefill.
- 16 bit dirty reads for cache maintenance operations.
- 1 or 2 bit dirty writes for writes and allocations.

Dirty RAM organization:-



L2 Cache Size	N =
128KB	512
256KB	1,024
512KB	2,048
1MB	4,096
2MB	8,192



way7 word 7-4	way7 word 3-0	way6 word 7-4	way6 word 3-0	way5 word 7-4	way5 word 3-0	way4 word 7-4	way4 word 3-0
idx0	idx0	idx0	idx0	idx0	idx0	idx0	idx0
dirty	dirty	dirty	dirty	dirty	dirty	dirty	dirty

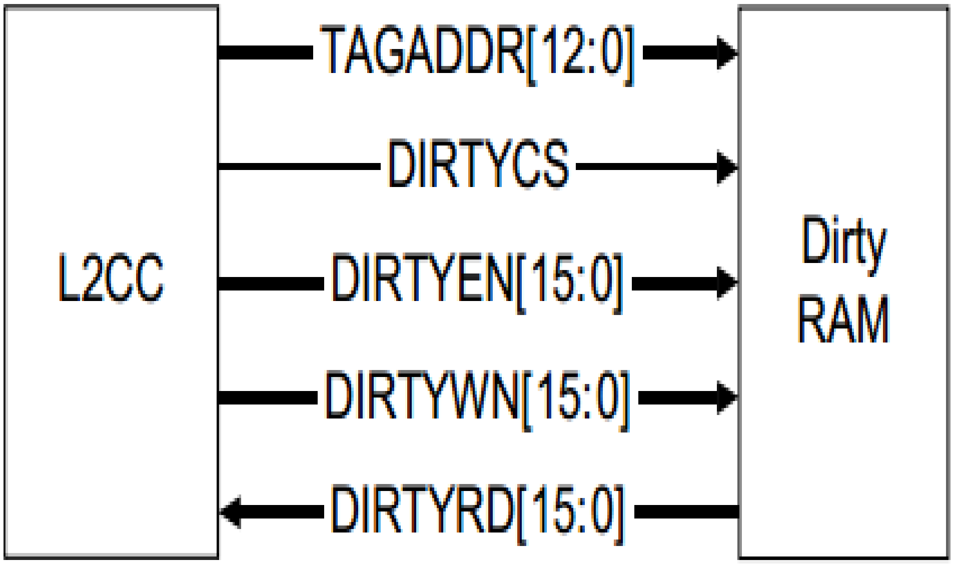
way3 word 7-4	way3 word 3-0	way2 word 7-4	way2 word 3-0	way1 word 7-4	way1 word 3-0	way0 word 7-4	way0 word 3-0
idx0	idx0	idx0	idx0	idx0	idx0	idx0	idx0
dirty	dirty	dirty	dirty	dirty	dirty	dirty	dirty

way7 word 7-4	way7 word 3-0	way6 word 7-4	way6 word 3-0	way5 word 7-4	way5 word 3-0	way4 word 7-4	way4 word 3-0
idx1	idx1	idx1	idx1	idx1	idx1	idx1	idx1
dirty	dirty	dirty	dirty	dirty	dirty	dirty	dirty

way3 word 7-4	way3 word 3-0	way2 word 7-4	way2 word 3-0	way1 word 7-4	way1 word 3-0	way0 word 7-4	way0 word 3-0
idx1	idx1	idx1	idx1	idx1	idx1	idx1	idx1
dirty	dirty	dirty	dirty	dirty	dirty	dirty	dirty

The above figure shows the dirty RAM connectivity.

Dirty RAM connectivity:-



Tag RAM:-

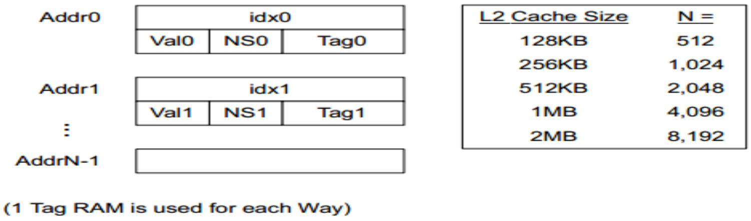
There is one tag RAM for each way of the L2 cache. A tag RAM is organized as a 21-bit wide memory. 18 bits are dedicated to address tag, 1 bit for security information, 1 bit for valid information, and optionally 1 bit for parity. The tag RAM address bus is also the address bus for the dirty RAM. The tag RAM support the following accesses:

- 20-bit tag reads for Tag lookup
- 20-bit tag writes for allocations.

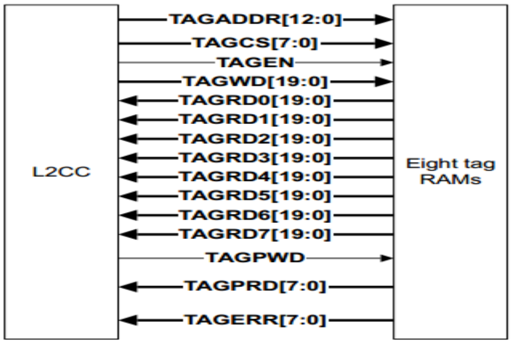
The NS bit takes the value of 1 for NS data, and 0 for secure data.

Note:-You require a 21-bit wide memory to support the parity option.

Tag RAM organization:-

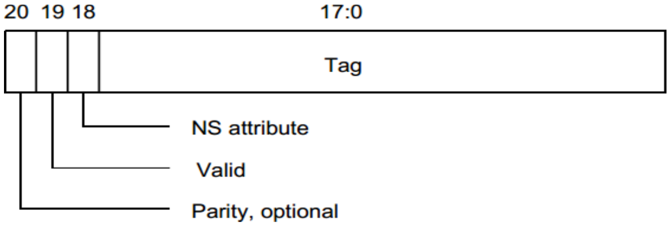


L2 Cache Size	N =
128KB	512
256KB	1,024
512KB	2,048
1MB	4,096
2MB	8,192



Cache lookup:-

The tag RAM format:



Tag RAMformat:-

Each line is marked as secure or NS depending on the value of the **AWPROT[1]** or **ARPROT[1]** value on the original transaction. The security setting of the access, **AWPROT[1]** or **ARPROT[1]**, is used for Cache Lookup and compared with the NS attribute in the Tag. The tag RAM contains a field to hold the NS attribute bit corresponding for each cache line. This is required so that the NS attribute bit for all cache ways is compared to generate the cache hit.

Note

- The cache is not automatically flushed when the processor changes security state.
- If an access is performed, and has an **AWPROT[1]/ARPROT[1]** value of 1'b1, then the NS attribute must be HIGH. Cache lookups are performed on lines marked as NS, the NS cache line attribute = 1, according to *Physical Address*(PA).
- If any access is performed in secure state, and the transaction has an **AWPROT[1]/ARPROT[1]** value of 1'b0), then the NS attribute must be LOW. Cache lookups are performed on lines marked as secure (NS cachelineattribute=0)accordingtoPA.AsecureaccessonlyhitsontagswithasecureNSattribute.

RAM sizes:-

The below table shows the different sizes of RAM.

L2 cache size	Data RAM	Tag RAM	Dirty RAM
128KB	1 × (256 + 32) × (ways × 512)	Ways × (20 + 1) × 512	1 × (2 × ways) × 512
256KB	1 × (256 + 32) × (ways × 1024)	Ways × (19 + 1) × 1,024	1 × (2 × ways) × 1,024
512KB	1 × (256 + 32) × (ways × 2048)	Ways × (18 + 1) × 2,048	1 × (2 × ways) × 2,048
1MB	1 × (256 + 32) × (ways × 4096)	Ways × (17 + 1) × 4,096	1 × (2 × ways) × 4,096
2MB	1 × (256 + 32) × (ways × 8192)	Ways × (16 + 1) × 8,192	1x (2 × ways) × 8,192

Note:-

1. The format for RAM sizes are:
    - Number of RAM × (width + parity) × number of address location.
2. The dirty ram does not have parity. Width for the tag RAM consists of Valid + NS +address.

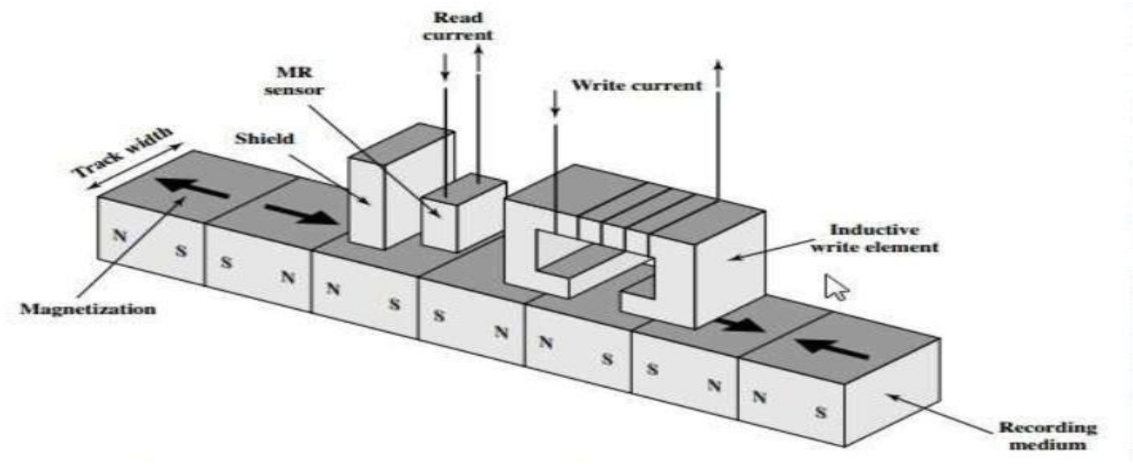
Magnetic Surface Recording:

A disk is a circular platter constructed of nonmagnetic material, called the substrate, coated with a magnetizable material. Traditionally, the substrate has been an aluminum or aluminum alloy material. More recently, glass substrates have been introduced. The glass substrate has a number of benefits, including the following:

1. Improvement in the uniformity of the magnetic film surface to increase diskreliability;
2. A significant reduction in overall surface defects to help reduce readwrite errors;
3. Ability to support lower fly heights (described subsequently);
4. Better stiffness to reduce disk dynamics;and
5. Greater ability to withstand shock and damage

Magnetic Read and Write Memory:-

Magnetic disks remain the most important component of external memory. Both removable and fixed, or hard, disks are used in systems ranging from personal computers to mainframes and supercomputers. Data are recorded on and later retrieved from the disk via a conducting coil named the head. In many systems, there are two heads, a read head and a write head. During a read or write operation, the head is stationary while the platter rotates be neat hit.



Inductive write/Magetoresistive Read Head

The write mechanism exploits the fact that electricity flowing through a coil produces a magnetic field. Electric pulses are sent to the write head, and the resulting magnetic patterns are recorded on the surface below, with different patterns for positive and negative currents. The traditional read mechanism exploits the fact that a magnetic field moving relative to a coil produces an electrical current in the coil. When the surface of the disk passes under the head, it generates a current of the same polarity as the one already recorded. The structure of the head for reading is in this case essentially the same as for writing and therefore the same head can be used for both. Such single heads are used in floppy disk systems and in older rigid disk systems. The read head consists of a partially shielded magneto resistive (MR) sensor. The MR material has an electrical resistance that depends on the direction of the magnetization of the medium moving under it.

Optical Memories:

Optical memories are used for large, storage of data. These devices provide the option of variety of data storage. These can save up to 20 GB of information. The data or information is read or written using a laser beam. Due to its low cost and high data storage capacity these memories are being freely used. Apart from low cost these memories have long life. But the problem is that of low access time.

Some Examples of Optical Memory:

CD-ROM: CD ROM or Compact-Disk Read Only Memory are optical storage device which can be easily read by computer but not written. CD-ROMs are stamped by the vendor, and once stamped, they cannot be erased and filled with new data. To read a CD, CD-ROM player is needed. All D-ROMs conform to a standard size and format, so any type of CD-ROM can be loaded into any CD-ROM player. In addition, CD-ROM players are capable of playing audio CDs, which share the same technology. CD-ROMs are particularly well-suited to information that requires large storage capacity This includes large software applications that support colour, graphics. sound and especially video.

Advantages of CD ROM:

1. Storage capacity is high.
2. Data storage cost per bit is reasonable.
3. Easy to carry.
4. Can store variety of data.

Disadvantages of CD ROM:-

1. CD ROMs are read only.
2. Access time is more than hard disk.

WORM:

WORM or Write Once Read Many or CD-R or CD-Record able are a kind of optical device which provides the user the liberty to write once on the CD R. The user can write on the disk using the CD R disk drive unit. But this data or information cannot be overwritten or changed. CD R does not allow re-writing though reading can be done many times.

Advantages of WORM:-

1. Storage capacity is high.
2. Can be recorded once.
3. Reliable.
4. Runs longer.
5. Access time is good.

Disadvantages or limitations of WORM:-

1. Can be written only once.

Erasable Optical Disk:-

Erasable Optical Disks are also called CD RW or CD rewritable. It gives the user the liberty of erasing data already written by burning the microscopic point on the disk surface. The disk can be reused.

Advantages of CD RW:-

Storage capacity is very high.

1. Reliability is high.
2. Runs longer.
3. Easy to rewrite.

Limitations of CD RW:-

- Access time is high.

DVD-ROM, DVD-R and DVD-RAM:-

DVD or Digital Versatile Disk is another form of optical storage. These are higher in capacity than the CDs. Pre-recorded DVDs are mass-produced using molding machines that physically stamp data onto the DVD. Such disks are known as DVD-ROM, because data can only be read and not written nor erased. DVD Rs are the blank record able DVDs which can be recorded once using optical disk recording technologies by using DVD recorders and then function as a DVD-ROM. DVD-ROM. Re writable DVDs DVD-RAM can be recorded.

Multilevel memories:

Memory Hierarchy

Memory System Organization

No matter how big the main memory, how we can organize effectively the memory system in order to store more information than it can hold. The traditional solution to storing a great deal of data is a memory hierarchy.

Major design objective of any memory system:

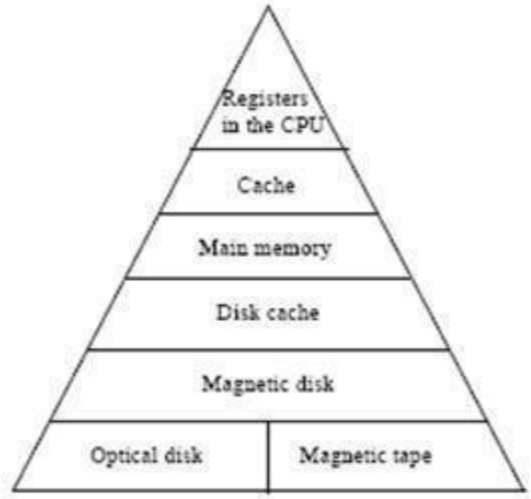
1. To provide adequate storage capacity
2. An acceptable level of performance
3. At a reasonable cost

Four interrelated ways to meet this goal

1. Use a hierarchy of storage devices.
2. Develop automatic space allocation methods for efficient use of the memory.
3. Through the use of virtual memory techniques, free the user from memory management tasks.
4. Design the memory and its related interconnection structure so that the processes.

Multilevel Memories Organization:-

Three key characteristics increase for a memory hierarchy. They are the access time, the storage capacity and the cost. The memory hierarchy is illustrated in figure 9.1.

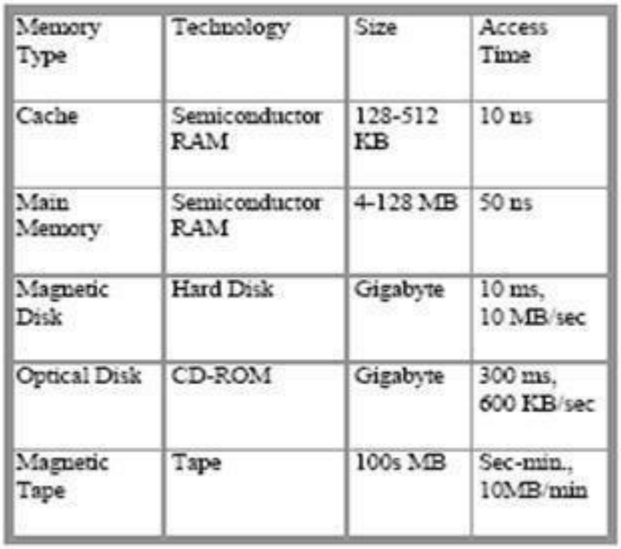


The memory hierarchy

We can see the memory hierarchy with six levels. At the top there are CPU registers, which can be accessed at full CPU speed. Next commes the cache memory, which is currently on order of 32 KByte to a few Mbyte. The main memory is next, with size currently ranging from 16 MB for entry-level systems to tens of Gigabytes. After that come magnetic disks, the current work horse for permanent storage. Finally we have magnetic tape and optical disks for archival storage.

Basis of the memory hierarchy

1. Registers internal to the CPU for temporary data storage (small in number but very fast)
2. External storage for data and programs (relatively large and fast)
3. External permanent storage (much larger and much slower)



Memory Type	Technology	Size	Access Time
Cache	Semiconductor RAM	128-512 KB	10 ns
Main Memory	Semiconductor RAM	4-128 MB	50 ns
Magnetic Disk	Hard Disk	Gigabyte	10 ms, 10 MB/sec
Optical Disk	CD-ROM	Gigabyte	300 ms, 600 KB/sec
Magnetic Tape	Tape	100s MB	Sec-min., 10MB/min

Typical Memory Parameters

Characteristics of the memory hierarchy

1. Consists of distinct “levels” of memory components
2. Each level characterized by its size, access time, and cost per bit
3. Each increasing level in the hierarchy consists of modules of larger capacity, slower access time, and lower cost/bit

Memory Performance

Goal of the memory hierarchy. Try to match the processor speed with the rate of information transfer from the lowest element in the hierarchy.

The memory hierarchy speed up the memory performance.

The memory hierarchy works because of locality of reference.

– Memory references made by the processor, for both instructions and data, tend to cluster together

+ Instruction loops, subroutines

+ Data arrays, tables

– Keep these clusters in high speed memory to reduce the average delay in accessing data

– Over time, the clusters being referenced will change -- memory management must deal with this

- Performance of a two level memory

Example: Suppose that the processor has access to two level of memory:

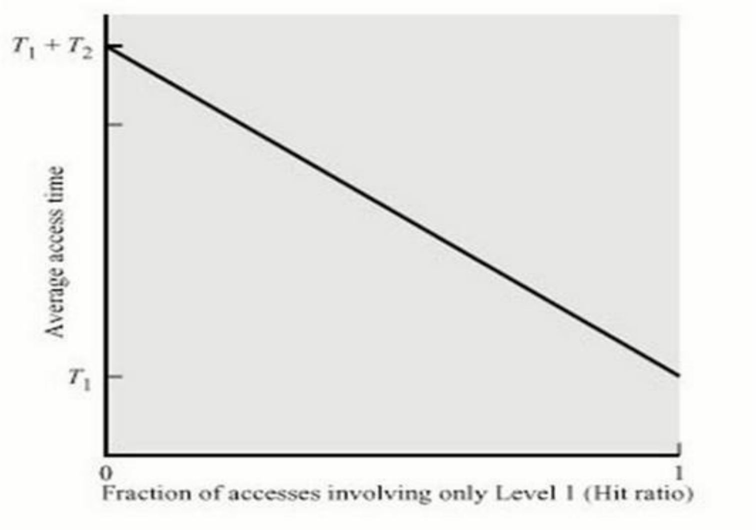
– Two-level memory system

– Level 1 access time of 1us

– Level 2 access time of 10us

– Ave access time = H(1) + (1-H)(10)ns

where: H is a fraction of all memory access that are found in the faster memory (e.g cache)



Performance of a two level memory

Cache & virtualmemory:-

Cache memory:-

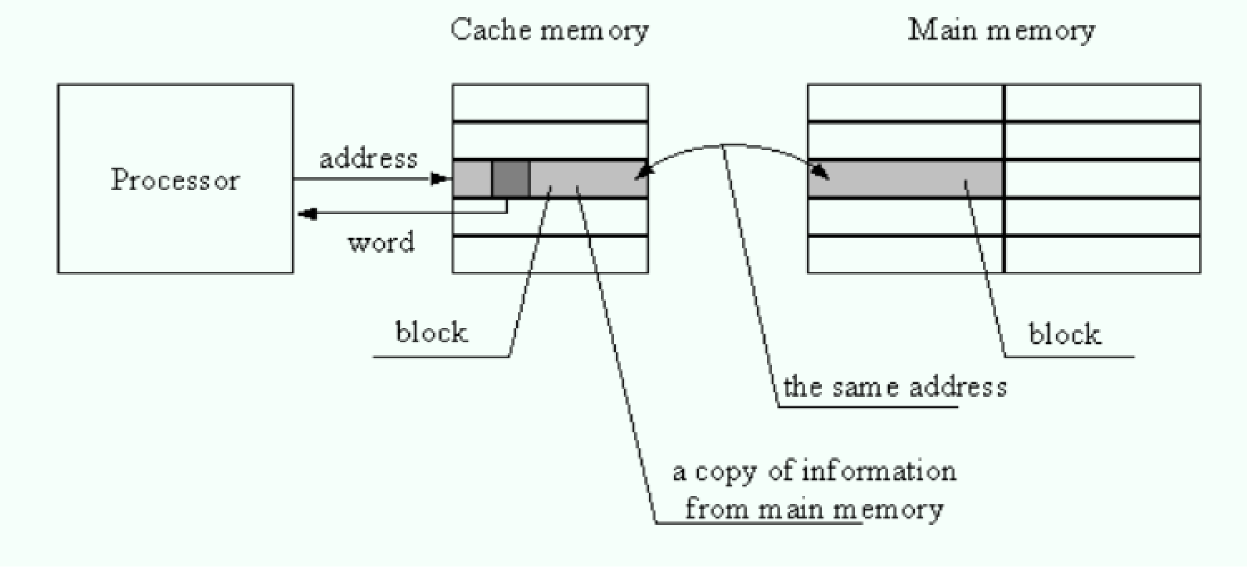
A cache memory is a fast random access memory where the computer hardware stores copies of information currently used by programs (data and instructions), loaded from the main memory. The cache has a significantly shorter access time than the main memory due to the applied faster but more expensive implementation technology. The cache has a limited volume that also results from the properties of the applied technology. If information fetched to the cache memory is used again, the access time to it will be much shorter than in the case if this information were stored in the main memory and the program will execute faster.

Time efficiency of using cache memories results from the locality of access to data that is observed during program execution.

We observe here time and space locality:

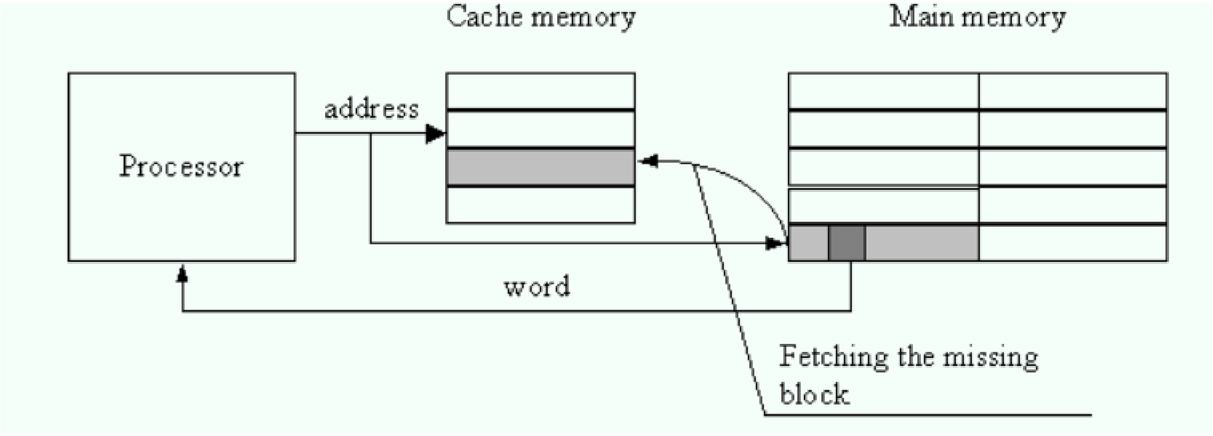
1. **Time locality** consists in a tendency to use many times the same instructions and data in programs during neighbouring time intervals,
2. **Space locality** is atendencytostoreinstructionsanddatausedinaprograminshortdistancesoftime under neighbouring addresses in the main memory.

Due to these localities, the information loaded to the cache memory is used several times and the execution time of programs is much reduced. Cache can be implemented as a multi-level memory. Contemporary computers usually have two levels of caches. In older computer models, a cache memory was installed outside a processor (in separate integrated circuits than the processor itself). The access to it was organized over the processor external system bus. In today's computers, the first level of the cache memory is installed in the same integrated circuit as the processor. It significantly speeds up processor's co-operation with the cache. Some microprocessors have the second level of cache memory placed also in the processor's integrated circuit. The volume of the first level cache memory is from several thousands to several tens of thousands of bytes. The second level cache memory has volume of several hundred thousand bytes. A cache memory is maintained by a special processor subsystem called cache controller. If there is a cache memory in a computer system, then at each access to a main memory address in order to fetch data or instructions, processor hardware sends the address first to the cache memory. The cache control unit checks if the requested information resides in the cache. If so, we have a "hit" and the requested information is fetched from the cache. The actions concerned with a read with a hit are shown in the figure below.



Read implementation in cache memory on hit

If the requested information does not reside in the cache, we have a "miss" and the necessary information is fetchedfromthemainmemorytothecacheandtotherequestingprocessorunit.Theinformationisnotcopiedin the cache as single words but as a larger block of a fixed volume. Together with information block, a part of the address of the beginning of the block is always copied into the cache. This part of the address is next used at readout during identification of the proper information block. The actions executed in a cache memory on "miss" are shown below.



Read implementation in cache memory on miss

To simplify the explanations, we have assumed a single level of cache memory below. If there are two cache levels, then on "miss" at the first level, the address is transferred in a hardwired way to the cache at the second level. If at this level a "hit" happens, the block that contains the requested word is fetched from the second level cache to the first level cache. If a "miss" occurs also at the second cache level, the blocks containing the requested word are fetched to the cache memories at both levels. The size of the cache block at the first level is from 8 to several tens of bytes (a number must be a power of 2). The size of the block in the second level cache is many times larger than the size of the block at the firstlevel.

The cache memory can be connected in different ways to the processor and the main memory:

- as an additional subsystem connected to the system bus that connects the processor with the main memory,
- as a subsystem that intermediates between the processor and the main memory,
- as a separate subsystem connected with the processor, in parallel regarding the main memory. The third solution is applied the most frequently.

We will now discuss different kinds of information organization in cache memories.

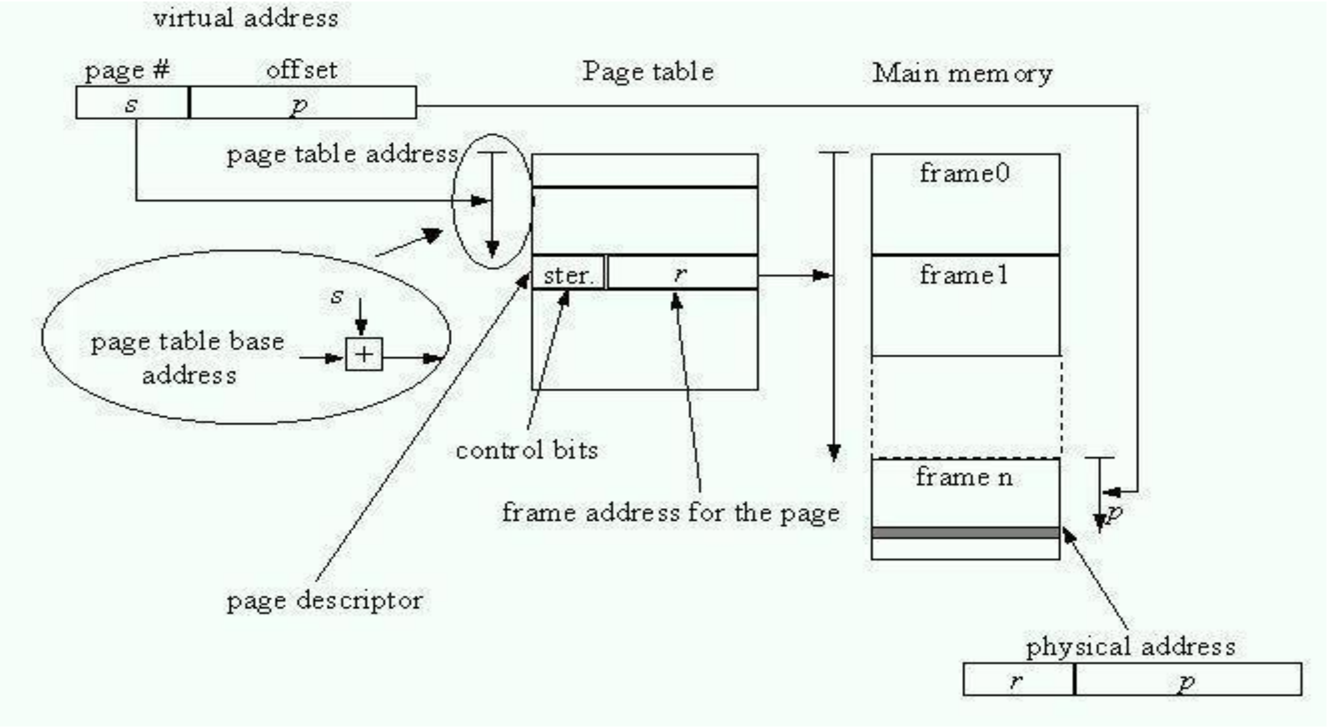
There are three basic methods used for mapping of information fetched from the main memory to the cache memory:

1. associative mapping
2. direct mapping
3. set-associative mapping.

In today's computers, caches and main memories are byte-addressed, so we will refer to byte-addressed organization in the sections on cache memories that follow.

Virtual memory organization:-

In early computers, freedom of programming was seriously restricted by a limited volume of main memory comparing program sizes. Small main memory volume was making large programs execution very troublesome and did not enable flexible maintenance of memory space in the case of many co-existing programs. It was very uncomfortable, since programmers were forced to spend much time on designing a correct scheme for data and code distribution among the main memory and auxiliary store. The solution to this problem was supplied by introduction of the virtual memory concept. This concept was introduced at the beginning of years 1970 under the name of **one-level storage** in the British computer called Atlas. Only much later, together with application of this idea in computers of the IBM Series 370, the term **virtual memory** was introduced. Virtual memory provides a computer programmer with an addressing space many times larger than the physically available addressing space of the main memory. Data and instructions are placed in this space with the use of **virtual addresses**, which can be treated as artificial in some way. In the reality, data and instructions are stored both in the main memory and in the auxiliary memory (usually disk memory). It is done under supervision of the virtual memory control system that governs real current placement of data determined by virtual addresses. This system automatically (i.e. without any programmer's actions) fetches to the main memory data and instructions requested by currently executed programs. The general organization scheme of the virtual memory is shown in the figure below.



General scheme of the virtual memory:-

Virtual memory address space is divided into fragments that have pre-determined sizes and identifiers that are consecutive numbers of these fragments in the set of fragments of the virtual memory. The sequences of virtual memory locations that correspond to these fragments are called **pages** or **segments**, depending on the type of the virtual memory applied. A **virtual memory address** is composed of the number of the respective fragment of the virtual memory address space and the word or byte number in the given fragment.

We distinguish the following solutions for contemporary virtualmemory systems:

- **paged (virtual)memory**
- **segmented (virtual)memory**
- **segmented (virtual) memory with paging**.

When accessing data stored under a virtual address, the virtual address has to be converted into a physical memory address by the use of **address translation**. Before translation, the virtual memory system checks if the segment or the page, which contains the requested word or byte, resides in the main memory. It is done by tests of page or segments descriptors in respective tables residing in the main memory. If the test result is negative, a physical address sub-space in the main memory is assigned to the requested page or segment and next it is loaded into this address sub-space from the auxiliary store. Next, the virtual memory system up-dates the page or segment descriptions in the descriptor tables and opens access to the requested word or byte for the processor instruction, which has emitted the virtual address.

The virtual memory control system is implemented today as partially hardware and software system. Accessing descriptor tables and virtual to physical address translation is done by computer hardware. Fetching missing pages or segments and up-dating their descriptors is done by the operating system, which, however, is strongly supported by special memory management hardware. This hardware usually constitutes a special functional unit for virtual memory management and special functional blocks designed to perform calculations concerned with virtual address translation.

Memory allocation:-

Memory is the processes by which information is encoded, stored and retrieved. Encoding allow information that is from the outside world to reach our senses in the forms of chemical and physical stimuli. Memory allocation is a process by which computer programs and services are assigned with physical or virtual memory space. Memory allocation is the process of reserving a partial or complete portion of computer memory for the execution of programs and processes. Memory allocation is achieved through a process known as memory management.

Memory allocation is primarily a computer hardware operation but is managed through operating system and software applications. Memory allocation process is quite similar in physical and virtual memory management.

Programs and services are assigned with a specific memory as per their requirements when they are executed. Once the program has finished its operation or is idle, the memory is released and allocated to another program or merged within the primary memory.

Memory allocation has two core types;

- Static Memory Allocation: The program is allocated memory at compile time.
- Dynamic Memory Allocation: The programs are allocated with memory at runtime.

Static memory allocation:-

In static memory allocation, size of the memory may be required for the calculation that must be define before loading and executing the program.

Dynamic memory allocation:

There are two methods which are used for dynamic memory allocation:

1. Non-Preemptive Allocation
2. Preemptive Allocation

Non Preemptive allocation:-

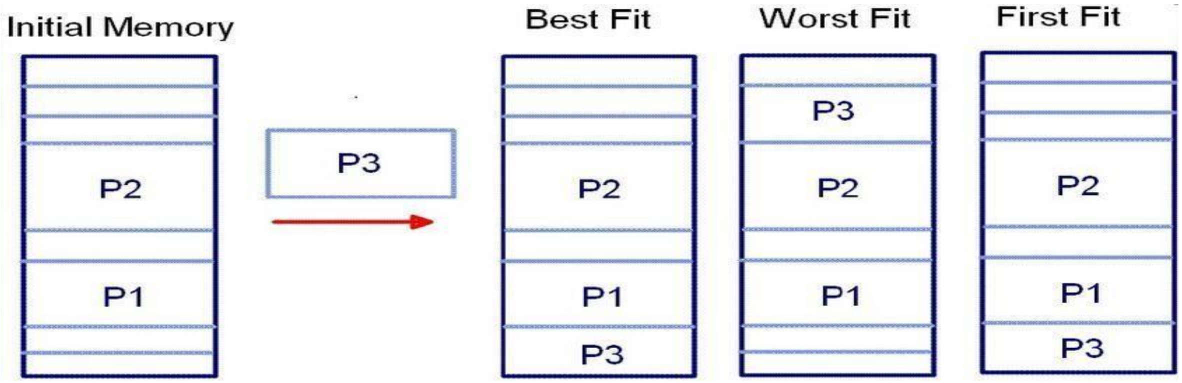
Consider M1 as a main memory and M2 as secondary memory and a block K of n words is to be transferred from M2 to M1. For such memory allocation it is necessary to find or create an available reason of n or more words to accommodate K. This process is known as non preemptive allocation.

First Fit

In this algorithm, searching is started either at the beginning of the memory or where the previous first search ended.

Best fit

In this algorithm, all free memory blocks are searched and smallest free memory block which is large enough to accommodate desired block K is used to allocate K.



Preemptive allocation:-

Non preemptive allocation can’t make efficient use of memory in all situation. Due scattered memory blocks larger free memory blocks may not be available. Much more efficient us of the available memory space is possible if the occupied space can be re allocated to make room for incoming blocks by a method called as Compaction.

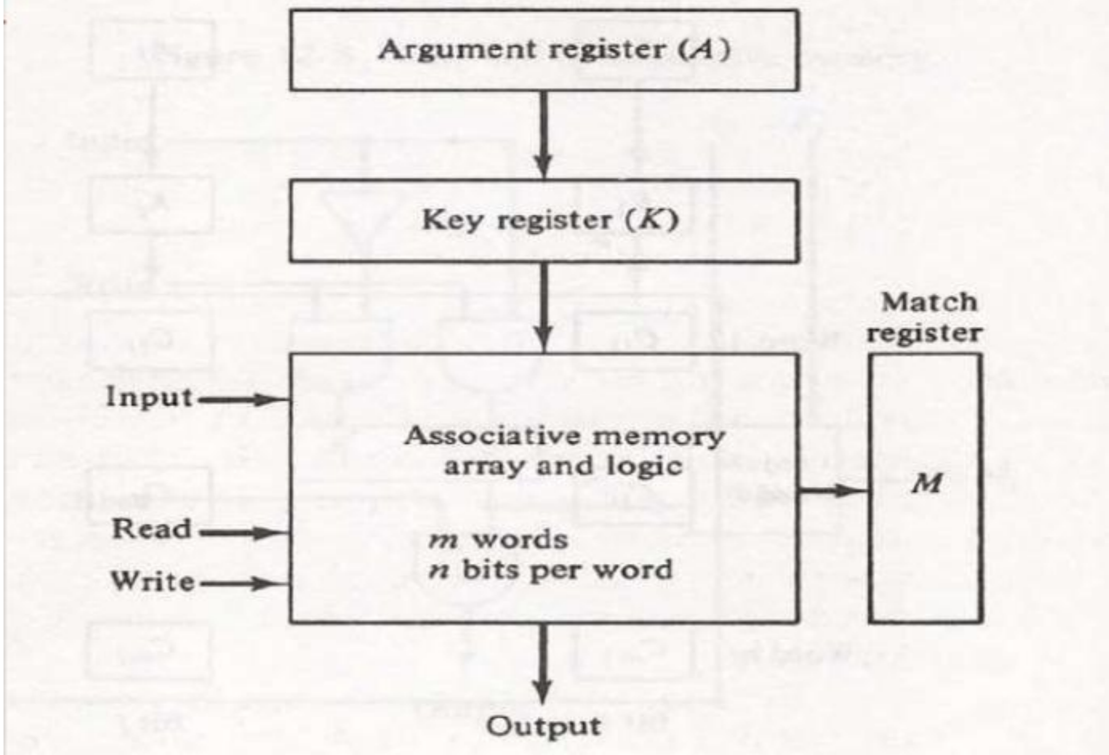
Associative Memory:-

A memory unit accessed by content is called associative memory or content addressable memory(CAM) or associative storage or associative array. Memory is capable of finding empty unused location to store the word.

To search particular data in memory, data is read from certain address and compared if the match is not found content of the next address is accessed and compared. This goes on until required data is found. The number of access depend on the location of data and efficiency of searching algorithm.

Hardware Organization of Associative Memory

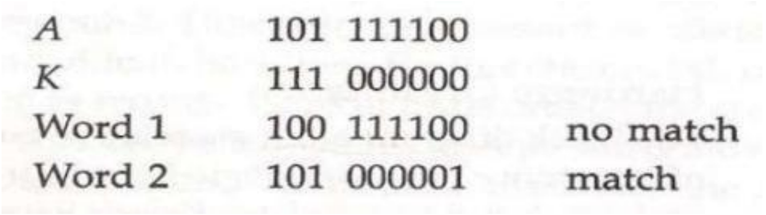
Figure Block diagram of associative memory.



Hardware Organization

Associative Memory is organized in such a way.

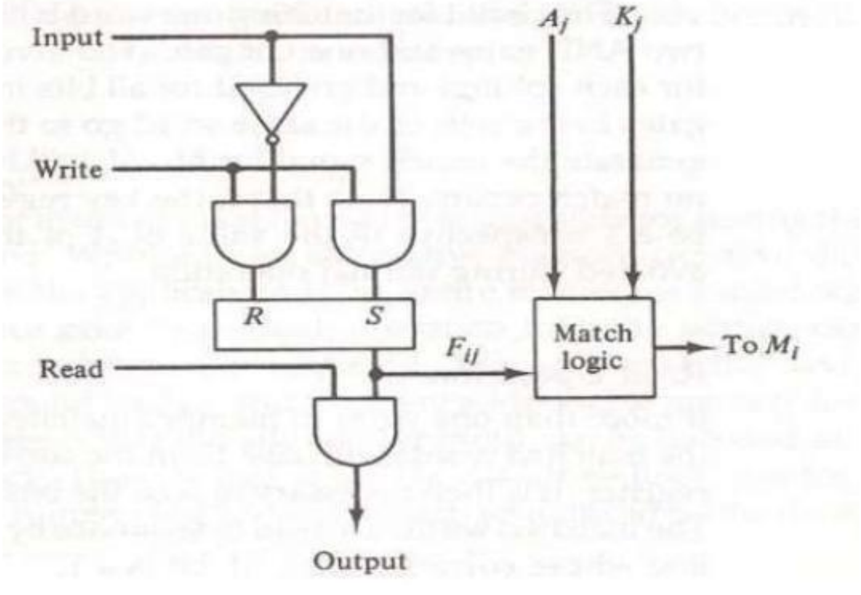
Argument register(A) : It contains the word to be searched. It has n bits(one for each bit of the word).
Key Register(K) : This specifies which part of the argument word needs to be compared with words in memory. If all bits in register are 1, The entire word should be compared. Otherwise, only the bits having k bit set to 1 will be compared.
Associative memory array : It contains the words which are to be compared with the argument word.
Match Register(M): It has m bits, one bit corresponding to each word in the memory array. After the matching process, the bits corresponding to matching words in match register are set to 1.
Key register provide the mask for choosing the particular field in A register. The entire content of A register is compared if key register content all 1. Otherwise only bit that have 1 in key register are 0compared. If the compared data is matched corresponding bits in the match register are set. Reading is accomplished by sequential access in memory for those words whose bit a reset.



A	101 111100	
K	111 000000	
Word 1	100 111100	no match
Word 2	101 000001	match

Match Logic:-

Figure One cell of associative memory.



Let us include key register. If Kj=0 then there is no need to compare Aj and Fij.

1. Only when Kj=1, comparison isneeded.
2. This achieved by ORing each term withKj.

$$M_i = (x_1 + K_1')(x_2 + K_2')(x_3 + K_3') \cdots (x_n + K_n')$$

Read Operation:-

When a word is to be read from an associative memory, the contents of the word, or a prt of the word is specified.

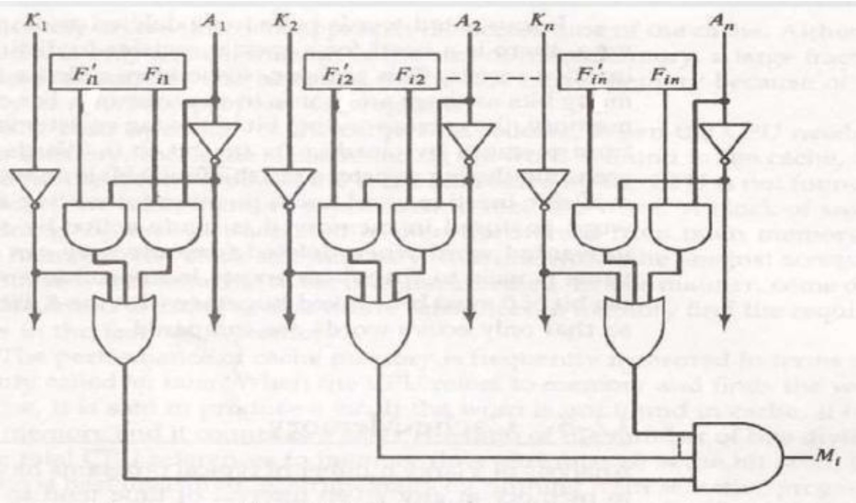


Figure Match logic for one word of associative memory.

Write operations:-

If the entire memory is loaded with new information at once prior to search operation then writing can be done by addressing each location in sequence. Tag register contain as many bits as there are words in memory. It contain 1 for active word and 0 for inactive word. If the word is to be inserted, tag register is scanned until 0 is found and word is written at that position and bit is change to1.

Advantages:-

This is suitable for parallel searches. It is also used where search time needs to be short.

1. Associative memory is often used to speed up databases, in neural networks and in the page tables used by the virtual memory of modern computers.
2. CAM-design challenge is to reduce power consumption associated with the large amount of parallel active circuitry, without sacrificing speed or memory density.

Disadvantages:-

1. An associative memory is more expensive than a random access memory because each cell must have an extra storage capability as well as logic circuits for matching its content with an external argument.
2. Usually associative memories are used in applications where the search time is very critical and must be very short.

UNIT-V
SYSTEM ORGANIZATION

Communication methods

Basic Concepts:

Bus is a group of wires that connects different components of the computer. It is used for transmitting data, control signal and memory address from one component to another. A bus can be 8 bit, 16 bit, 32 bit and 64 bit. A 32 bit bus can transmit 32 bit information at a time. A bus can be internal or external.

Types of bus:

- **Databus**

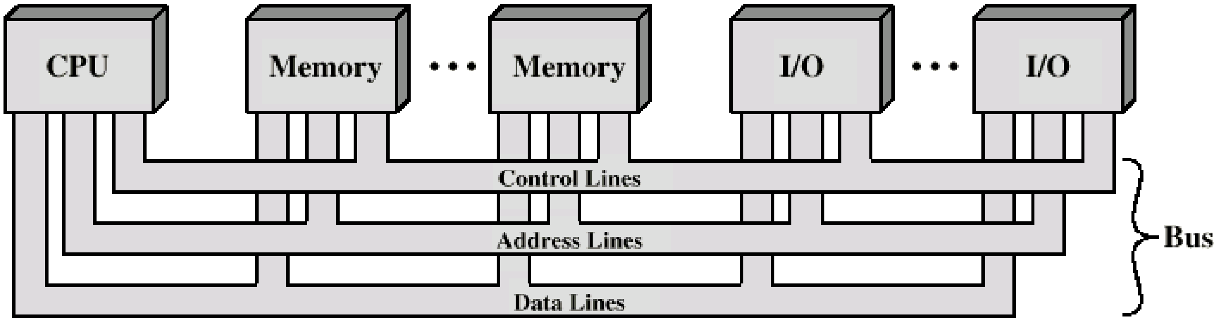
Data bus carries data from on component to another. It is uni-directional for input and output devices and bi-directional for memory and CPU.

- **Controlbus**

Control bus carries control signal. CU of CPU uses control signal for controlling all the components. It is uni-directional from CPU to all other components.

- **Addressbus**

Address bus carries memory address. A memory address is a numerical value used for identifying a memory location. Computer performs all its task through the memory address. CU of CPU sends memory address to all the components. So, address bus is also uni-directional from CPU to all other components.



Bus Interconnection Scheme

Long distance communication

The equipment for long distance communication is very important. Communication takes place either through heliograph signaling using a flashing mirror, electrical lamps, spot lights, colored disks, etc., or it is accomplished electrically by radio or by wires within the confined areas of the space station.

In communicating with the ground, use of heliograph signaling has the disadvantage of being unreliable because itdependsonthereceivingstationontheEarthbeingcloudless.Therefore,thespacestationhasatitsdisposal large radio equipment that makes possible both telegraph and telephone communications with the ground at any time. Overcoming a relatively significant distance as well as the shielding effect exerted by the atmosphere on radio waves (Heaviside layer),* are successful here (after selecting an appropriate direction of radiation) by using shorter, directed waves and sufficiently high transmission power, because conditions for this transmission are favorable since electric energy can be generated in almost any quantities by means of the solar power plant and because the construction of any type of antenna presents no serious problems as a result of the existing weightlessness.

Computer Network | Types of area networks – LAN, MAN and WAN

The **Network** allows computers to **connect and communicate** with different computers via any medium. LAN, MAN and WAN are the three major types of the network designed to operate over the area they cover. There are some similarities and dissimilarities between them. One of the major differences is the geographical area they cover, i.e. **LAN** covers the smallest area; **MAN** covers an area larger than LAN and **WAN** comprises the largest of all.

There are other types of Computer Networks also, like :

- PAN (Personal Area Network)
- SAN (Storage Area Network)
- EPN (Enterprise Private Network)
- VPN (Virtual Private Network)

Local Area Network (LAN) –

LAN or Local Area Network connects network devices in such a way that personal computer and workstations can share data, tools and programs. The group of computers and devices are connected together by a switch, or stack of switches, using a private addressing scheme as defined by the TCP/IP protocol. Private addresses are unique in relation to other computers on the local network. Routers are found at the boundary of a LAN, connecting them to the larger WAN.

Data transmits at a very fast rate as the number of computers linked are limited. By definition, the connections must be high speed and relatively inexpensive hardware (Such as hubs, network adapters and Ethernet cables). LANs cover smaller geographical area (Size is limited to a few kilometers) and are privately owned. One can use it for an office building, home, hospital, schools, etc. LAN is easy to design and maintain. A Communication medium used for LAN has twisted pair cables and coaxial cables. It covers a short distance, and so the error and noise are minimized.

Early LAN's had data rates in the 4 to 16 Mbps range. Today, speeds are normally 100 or 1000 Mbps. Propagation delay is very short in a LAN. The smallest LAN may only use two computers, while larger LANs can accommodate thousands of computers. A LAN typically relies mostly on wired connections for increased speed and security, but wireless connections can also be part of a LAN. The fault tolerance of a LAN is more and there is less congestion in this network. For example : A bunch of students playing Counter Strike in the same room (without internet).

Metropolitan Area Network (MAN) –

MAN or Metropolitan area Network covers a larger area than that of a LAN and smaller area as compared to WAN. It connects two or more computers that are apart but resides in the same or different cities. It covers a large geographical area and may serve as anISP(InternetServiceProvider).MANisdesignedforcustomerswho need a high-speed connectivity. Speeds of MAN ranges in terms of Mbps. It's hard to design and maintain a Metropolitan AreaNetwork.

The fault tolerance of a MAN is less and also there is more congestion in the network. It is costly and may or may not be owned by a single organization. The data transfer rate and the propagation delay of MAN is moderate. Devices used for transmission of data through MAN are: Modem and Wire/Cable. Examples of a MAN are the part of the telephone company network that can provide a high-speed DSL line to the customer or the cable TV network in a city.

Wide Area Network (WAN) –

WAN or Wide Area Network is a computer network that extends over a large geographical area, although it might be confined within the bounds of a state or country. A WAN could be a connection of LAN connecting to other LAN's via telephone lines and radio waves and may be limited to an enterprise (a corporation or an organization) or accessible to the public. The technology is high speed and relatively expensive.

There are two types of WAN: Switched WAN and Point-to-Point WAN. WAN is difficult to design and maintain. Similar to a MAN, the fault tolerance of a WAN is less and there is more congestion in the network. A Communication medium used for WAN is PSTN or Satellite Link. Due to long distance transmission, the noise and error tend to be more in WAN.

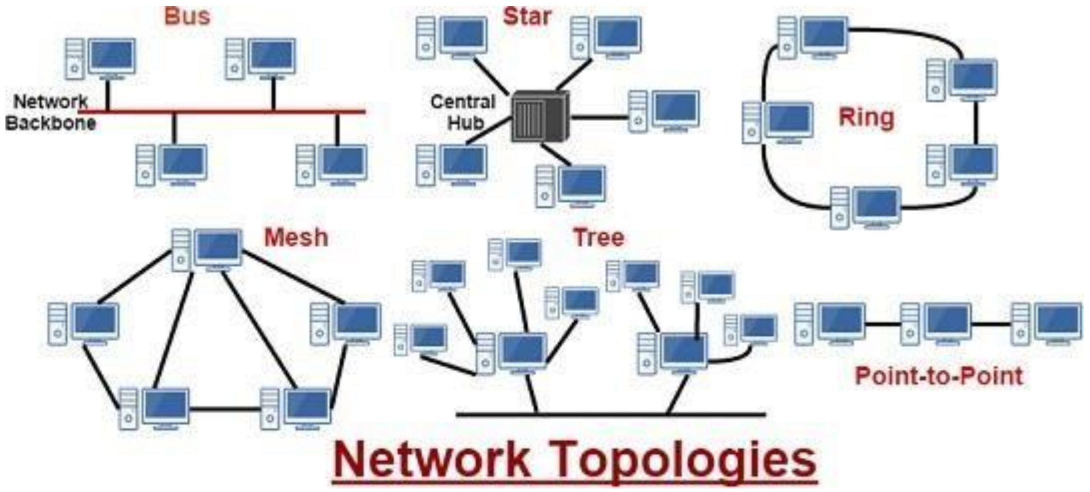
WAN's data rate is slow about a 10th LAN's speed, since it involves increased distance and increased number of servers and terminals etc. Speeds of WAN ranges from few kilobits per second (Kbps) to megabits per second (Mbps). Propagation delay is one of the biggest problems faced here. Devices used for transmission of data through WAN are: Optic wires, Microwaves and Satellites. Example of a Switched WAN is the asynchronous transfer mode (ATM) network and Point-to-Point WAN is dial-up line that connects a home computer to the Internet.

BusControl

Network Topology

Network Topology is an arrangement of nodes, nodes are basically components of network. This arrangement define the network layout or the structure of the network in both the ways physically and logically which enable communication between components of network. Nodes are connected to each other in different ways that we are going to discuss further in types of network topologies and communication take place. Physical layout of topology define the layout of nodes, cables and workstations in the network whereas logical topology determine how the information flow and communication done between the components of the network .So now we are going to discuss the types of Network topologies and how they connect computer network in any organization.

Types of Network Topologies



Point-to-Point Topology

Point-to-Point is one of the simplest topology and basic model of conventional telephony that provide permanent link between two endpoints. The purpose of this topology is to establish communication between two endpoints. Example of this topology is children’s tin can telephone that associated with the two endpoints. This topology is easy to understand, use circuit-switching and packet-switching technology for the information flow and can be dropped when not needed by the user anymore.

Bus Topology

Bus is a type of network topology in which all the devices are connected to a main single cable called the bus. In other words we can say a line connected to devices in the network. From bus only all the devices are connected to each other and share resources and file. This network is simple to maintain because if one node fails to fail then rest can communicate with the others. Advantage of bus topology is easy to connect a computer to a linear bus which require less wired cable because a single cable wire is required. But one of the major disadvantage of this topology is once the main cable get fail then it will broke down the whole network. This network is easy to expand by adding additional nodes to thebus.

Ring Topology

Ring is a type of network topology in which all the devices are connected in a closed loop and share data and resources to their adjacent nodes. Data can be shared with only two devices connected adjacent to a single node and flow of data has to be done in one followed direction only. Each node can receive data or can send data to the destination place. Basically this topology follow the token passing technology where each token contain data, sender and receiver information. Advantage of this topology is this is very organized and each node have equal access resource right. But the major drawback is if one device get fail it will break down the whole network.

Star Topology

Star is a type of network topology in which all the devices are connected to central hub through it communication take place between devices. A central hub can be a router or switch and all the devices are connected to central device with point-to-point connection. Central device act as a junction which manage and control all the data that pass from it before going to intended destination. Advantages of using this topology is easy to install such network. If one device is failed no need to disturb the whole network while remove that device from the network. Easy to detect faults and remove that part. But it also has some of the disadvantages like it require more cable as compared to linear bus topology and if central device fail then also disabled the attached nodes.

Tree Topology

Tree is a type of network topology that integrates the advantages of both bus and star topology. Basically it connect one star network with the another by the help of bus network and as many other star networks. Advantages of this technology is expansion is easy to add other devices, error detection and correction is easy, if one segment is damaged it doesn’t affect another device or segment. But this affect the maintenance of the network by adding additional nodes to the network and scalability depends on type of cable used.

Mesh Topology

Mesh is a type of topology in which each of the network node interconnected with one another and each node sends signal as well as relay data from other nodes. A true mesh is the one in which each node in the network connect to every node in the network which is also termed as full mesh or fully connected topology. But to establish such network is very expensive but still it is used in wireless networks. The techniques used by this network is Flooding or Routing. A device of network can transmit data to different devices at the same time. If one device get damage it doesn’t affect another device or sub-network in the same network. But cost is very high and set-up of this network is difficult to maintain, also network administration is difficult.

Bus interfacing

Consider a computer system using different interface standards. Let us look in to Processor bus and Peripheral Component Interconnect (PCI) bus. These two buses are interconnected by a circuit called bridge. It is a bridge between processor bus and PCI bus. An example of a computer system using different interface standards is shown in figure 4.38. The three major standard I/O interfaces discussed here are:

– PCI (Peripheral Component Interconnect)
– SCSI (Small Computer System Interface)
– USB (Universal Serial Bus)

PCI (Peripheral Component Interconnect)

The topics discussed under PCI are: Data Transfer, Use of a PCI bus in a computer system, A read operation on the PCI bus, Device configuration and Other electrical characteristics. Use of a PCI bus in a computer system is shown in figure 4.39 as a representation.

Host, main memory and PCI bridge are connected to disk, printer and Ethernet interface through PCI bus. At any given time, one device is the bus master. It has the right to initiate data transfers by issuing read and write commands. A master is called an initiator in PCI terminology. This is either processor or DMA controller. The addressed device that responds to read and write commands is called a target. A complete transfer operation on the bus, involving an address and a burst of data, is called a transaction. Device configuration is also discussed.

SCSI Bus

It is a standard bus defined by the American National Standards Institute (ANSI). A controller connected to a SCSI bus is an initiator or a target. The processor sends a command to the SCSI controller, which causes the following sequence of events to take place:

- The SCSI controller contends for control of the bus(initiator).
- When the initiator wins the arbitration process, it selects the target controller and hands over control of the bus to it.
- The target starts an output operation. The initiator sends a command specifying the required read operation.
- The target sends a message to the initiator indicating that it will temporarily suspends the connection between them. Then it releases the bus. The target controller sends a command to the disk drive to move the read head to the first sector involved in the requested read operation.
- The target transfers the contents of the data buffer to the initiator and then suspends the connection again.

- The target controller sends a command to the disk drive to perform another seek operation.
- As the initiator controller receives the data, it stores them into the main memory using the DMA approach.
- The SCSI controllers ends an interrupt to the processor to inform it that the requested operation has been completed.

The bus signals, arbitration, selection, information transfer and reselection are the topics discussed in addition to the above.

Universal Serial Bus (USB)

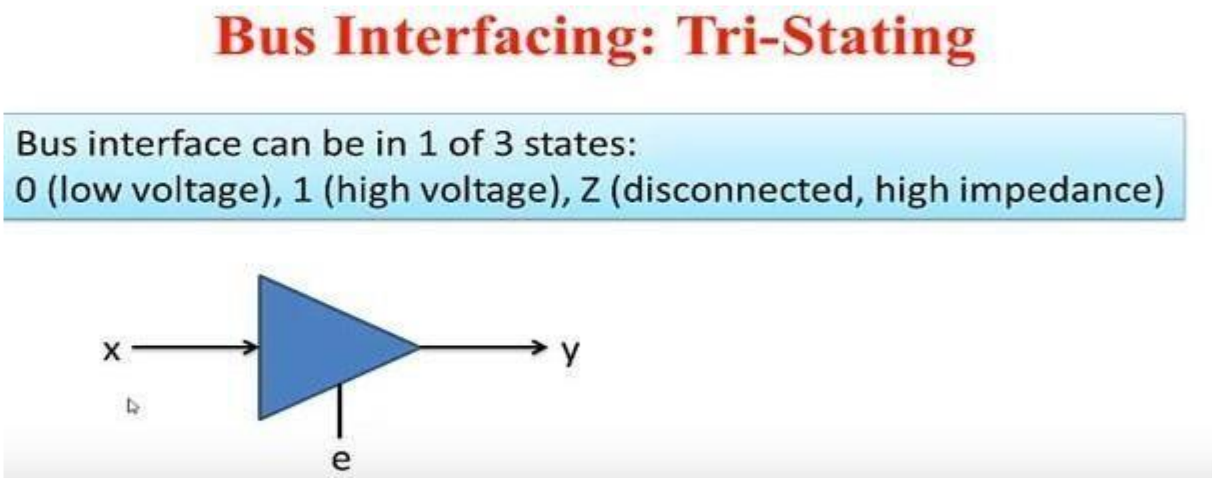
The USB has been designed to meet several key objectives such as:

- Provide a simple, low-cost and easy to use interconnection system that overcomes the difficulties due to the limited number of I/O ports available on a computer
- Accommodate a wide range of data transfer characteristics for I/O devices, including telephone and Internet connections
- Enhance user convenience through a "plug-and-play" mode of operation Port Limitation Here to add new ports, a user must open the computer box to gain access to the internal expansion bus and install a new interface card. The user may also need to know how to configure the device and the software. And also it is to make it possible to add many devices to a computer system at any time, without opening the computer box.

Device Characteristics

The kinds of devices that may be connected to a computer cover a wide range of functionality - speed, volume and timing constraints. A variety of simple devices attached to a computer generate data in different asynchronous mode. A signal must be sampled quickly enough to track its highest-frequency components.

Plug-and-play Whenever a device is introduced, do not turn the computer off/restart to connect/disconnect a device. The system should detect the existence of this new device automatically, identify the appropriate device-driver software and any other facilities needed to service that device, and establish the appropriate addresses and logical connections to enable them to communicate.



Bus Interfacing: Tri-Stating

Bus interface can be in 1 of 3 states:
0 (low voltage), 1 (high voltage), Z (disconnected, high impedance)

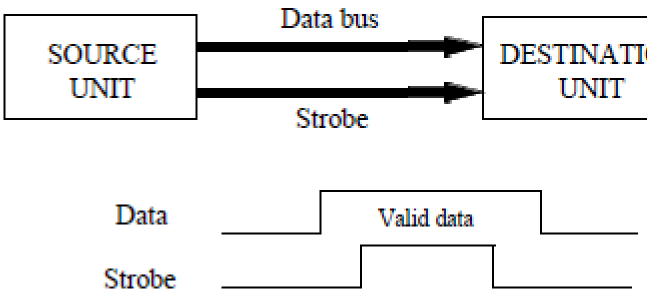
Asynchronous data transfer

Strobe: a pulse supplied to indicate the time at which data is being transmitted

Handshaking: a control signal is transmitted along with the data; another signal is sent by the receiver

Asynchronous data communication is used when transmitting data between two devices which cannot share a common clock. For example, data transmitted via modem would be sent asynchronously, whereas data transmitted within a single computer would be transmitted synchronously. A strobe is a pulse used by one computer to alert the other that it will send data. In some systems, the receiving unit sends an acknowledge signal back to the sender. This is handshaking.

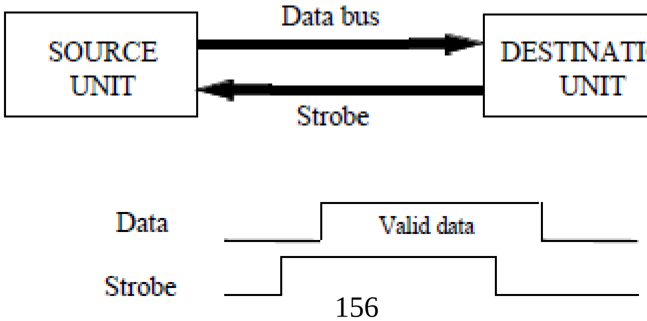
Source-initiated strobe



There are four general methods for transmitting data asynchronously, depending on whether the source or destination initiates the transfer and on whether or not handshaking is used. In source-initiated strobe, the source first places data onto the data bus. After giving it some time to settle down, it activates the strobe signal. This alerts the destination device that data is ready and the destination device reads in the data. After some set amount of time, the source unit de-asserts the strobe and then removes data from the bus. Some devices may load in data on the falling edge of the strobe, so the order of the last two operations is important.

Note that the source device never receives confirmation that the data was received successfully. That is a tradeoff inherent in this type of transfer. The advantage gained is that hardware requirements are reduced. This type of transfer could be useful for such operations as updating remote LED displays.

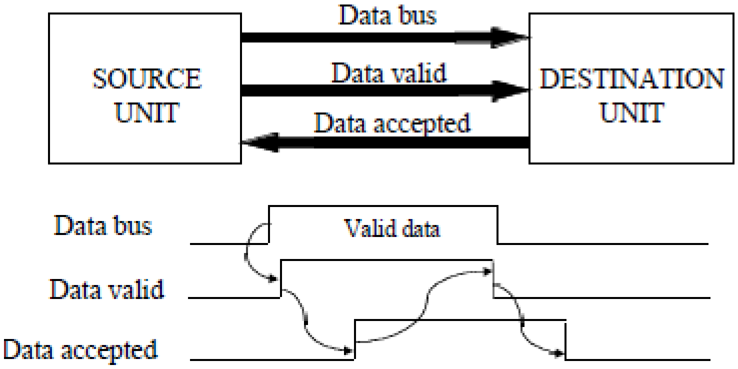
Destination-initiated strobe



Destination-initiated strobe is similar to source-initiated strobe except, as its name implies, the destination device initiates the data transfer by asserting the strobe. This causes data to be placed on the data bus; the destination device then reads in the data. After some preset amount of time, it releases the strobe, which signals the source unit to stop sending valid data. A circuit that uses this form of transmission might have a tri-state buffer as part of the source unit. The data from the source unit is input to the trisate buffer. The output of the buffer is connected to the data bus. The strobe is connected to the buffer enable.

As with the source-initiated strobe, there is no guarantee that the data read in by the destination device is valid. For instance, if the source device was turned off, the strobe would still be initiated, all zeroes would be read in and the strobe would be de-asserted. The destination device cannot tell if the all-zero data was read in from an active device or a powered-down device.

Source-initiated transfer using handshaking

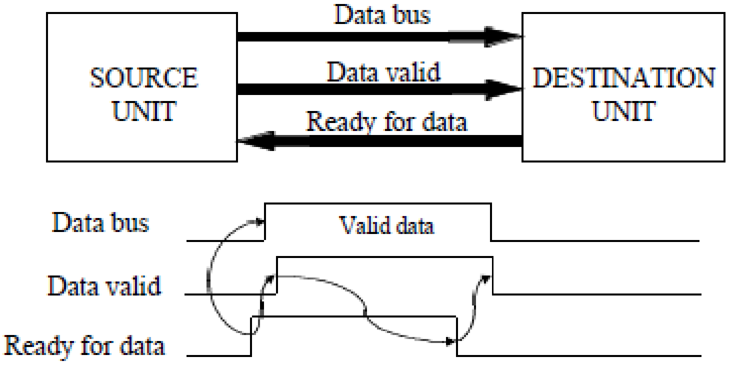


Handshaking removes the uncertainty in asynchronous data transfers by providing confirmation that data was read in successfully. The tradeoff is the increased hardware requirements. In source-initiated data transfer using handshaking, the source unit places valid data onto the data bus and then asserts a data valid signal. So far this is the same as before, with the data valid signal replacing the strobe.

But now something different happens. The data valid signal does not stay high for some predetermined amount of time. Instead it stays high until the destination unit tells it to stop. The destination unit reads in the data and then asserts a data accepted signal. This tells the source that the data has been accepted and to remove it from the bus. It de-asserts the data valid signal and removes data from the bus. The destination unit acknowledges this by de-asserting its data accepted signal, which in turn tells the source device that it may now initiate another data transfer.

One thing not noted here is what to do if the destination device never accepts the data (for example, if it is turned off). The source unit might include a timer that will check for this condition and report an error when it occurs.

Destination-initiated transfer using handshaking



Destination-initiated transfer using handshaking is also similar to its non-handshaking counterpart. The ready for data signal replaces the strobe and is asserted by the destination device to request data. The source device responds by placing data onto the bus and then, after some predetermined amount of time to allow for settling, it asserts the data valid signal. This tells the destination device that it can now read in the data. It does so and signals the source device by de-asserting the ready for data signal. This is followed by the source unit removing the data from the data bus and resetting the data valid signal. This notifies the destination unit that it can now request another data transfer.

Both source- and destination-initiated transfer using handshaking are similar to the process used by the Basic Computer.

Synchronous serial transmission

1. Share a common clock (short distance)
2. Separate clocks with synchronization signals (long distance)
3. Bits are transmitted continuously

Serial data transmission may be synchronous or asynchronous. In synchronous transmission, the source and destination units share a common clock. This is useful when data must be transmitted within a computer or over very short distances for which both devices may access a single clock.

Data can also be transmitted over longer distances synchronously. To do this, both the source and destination units must also send synchronization signals periodically to maintain calibration between the twoclocks.

Unlike asynchronous data transmission, in which the transmission line is used only when sending data, synchronous data transmission requires that transmission occurs continuously. When no data is to be sent, bits must still be transmitted to maintain synchronization.

Bus Arbitration

° The arbitration procedure comes into picture whenever there are more than one processors requesting the services of bus.

° Because only one unit may at a time be able to transmit successfully over the bus, there is some selection mechanism is required to maintain such transfers. This mechanism is called as Bus Arbitration.

° Bus arbitration decides which component will use the bus among various competing requests.

° A selection mechanism must be based on fairness or priority basis.

° Various methods are available that can be roughly classified as either centralized or distributed.

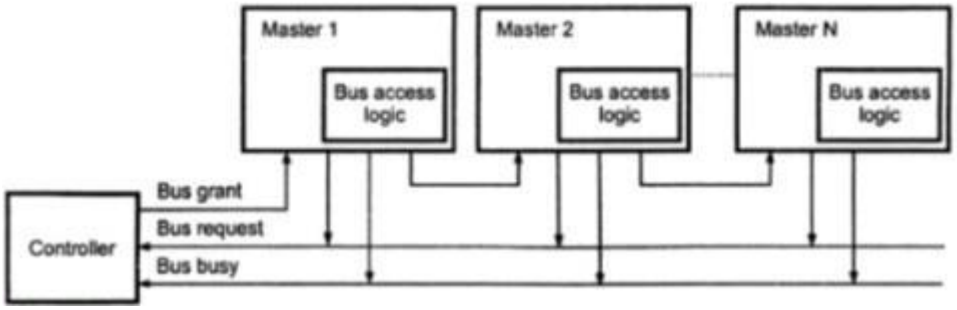
- **1. Centralized Arbitration**
  - In centralized bus arbitration, a single bus arbiter performs the required arbitration. Thebus arbiter may be the processor or a separate controller connected to thebus.
  - There are three different arbitration schemes that use the centralized bus arbitration approach. There schemes are:

a. Daisy chaining

b. Polling method

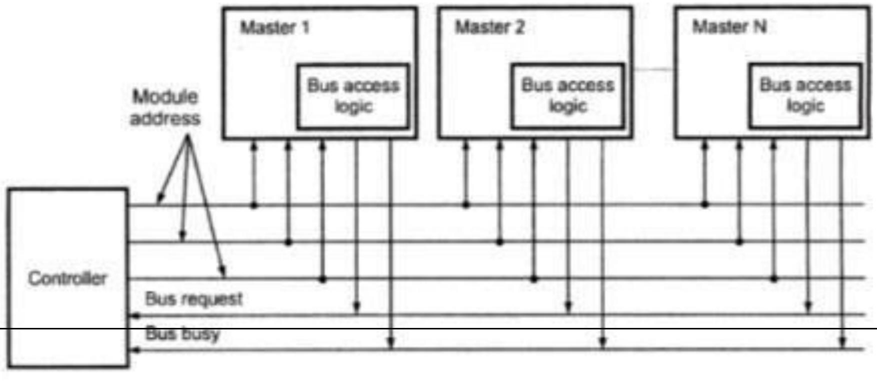
c. Independent request

- **a) Daisy chaining**
  - The system connections for Daisy chaining method are shown in fig below.



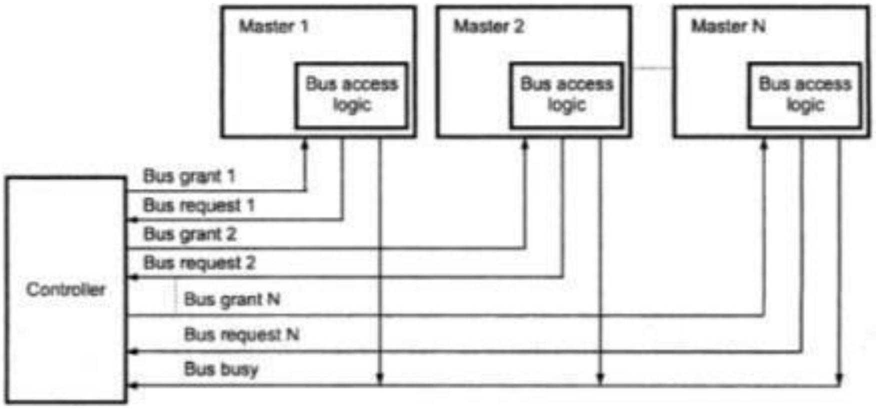
- It is simple and cheaper method. All masters make use of the same line for bus request.
- In response to the bus request the controller sends a bus grant if the bus is free.
- The bus grant signal serially propagates through each master until it encounters the first one that is requesting access to the bus. This master blocks the propagation of the bus grant signal, activities the busy line and gains control of the bus.
- Therefore any other requesting module will not receive the grant signal and hence cannot get the bus access.

b) Polling method



- The system connections for polling method are shown in figure above.
- In this the controller is used to generate the addresses for the master. Number of address line required depends on the number of master connected in the system.
- For example, if there are 8 masters connected in the system, at least three address lines are required.
- In response to the bus request controller generates a sequence of master address. When the requesting master recognizes its address, it activated the busy line ad begins to use thebus.

c) Independent request



- The figure below shows the system connections for the independent request scheme.
- In this scheme each master has a separate pair of bus request and bus grant lines and each pair has a priority assigned to it.
- The built in priority decoder within the controller selects the highest priority request and asserts the corresponding bus grant signal.

2. Distributed Arbitration

- In distributed arbitration, all devices participate in the selection of the next bus master.
- In this scheme each device on the bus is assigned a4-bit identification number.
- The number of devices connected on the bus when one or more devices request for the control of bus, they assert the start-arbitration signal and place their 4-bit ID numbers on arbitration lines, ARB0 through ARB3.
- These four arbitration lines are all open-collector. Therefore, more than one device can place their 4-bit ID number to indicate that they need to control of bus. If one device puts 1 on the bus line and another device puts 0 on the same bus line, the bus line status will be 0. Device reads the status of all lines through inverters buffers so device reads bus status 0as logic 1. Scheme the device having highest ID number has highest priority.
- When two or more devices place their ID number on bus lines then it is necessary to identify the highest IDnumberonbuslinesthenitisnecessarytoidentifythehighestIDnumberfromthe status of busline.

Consider that two devices A and B, having ID number 1 and 6, respectively are requesting the use of the bus.

- Device A puts the bit pattern 0001, and device B puts the bit pattern 0110. With this combination the status of bus-line will be 1000; however because of inverter buffers code seen by both devices is0111.
- Each device compares the code formed on the arbitration line to its own ID, starting from the most significant bit. If it finds the difference at any bit position, it disables its drives at that bit position and for all lower-order bits.
- It does so by placing a 0 at the input of their drive. In our example, device detects a different on line ARB2 and hence it disables its drives on line ARB2, ARB1 and ARB0. This causes the code on the arbitration lines to change to 0110. This means that device B has won the race.
- The decentralized arbitration offers high reliability because operation of the bus is not dependent on any single device.

IO System:

Accessing I/O Devices

I/O interface

Input/output mechanism

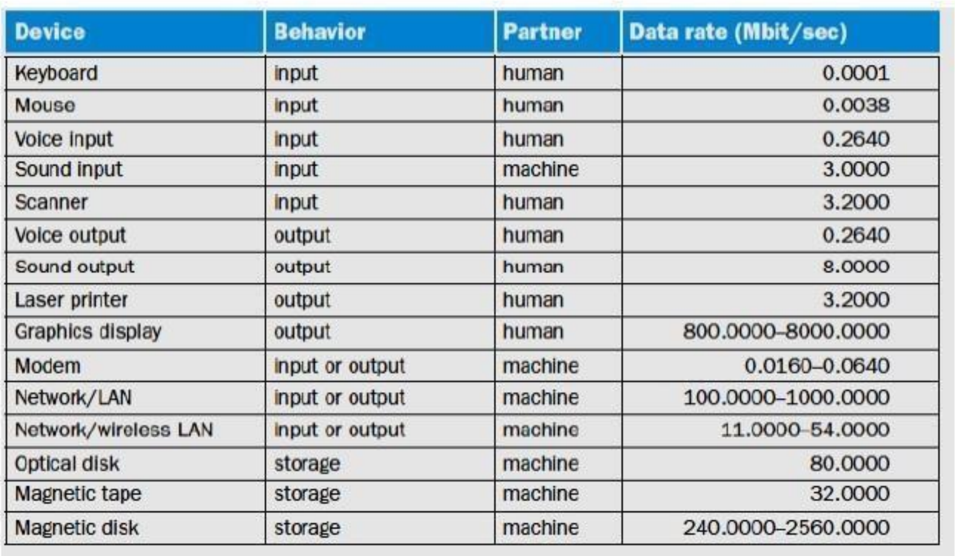
- Memory-mappedI/O
- ProgrammedI/O
- Interrupts
- Direct Memory Access

• Buses

- Synchronous Bus
- Asynchronous Bus

IO in Computer Organization and Operating System:

1. ProgrammedI/O
2. Interrupts
3. DMA (Direct memoryAccess



Device	Behavior	Partner	Data rate (Mbit/sec)
Keyboard	input	human	0.0001
Mouse	input	human	0.0038
Voice input	input	human	0.2640
Sound input	input	machine	3.0000
Scanner	input	human	3.2000
Voice output	output	human	0.2640
Sound output	output	human	8.0000
Laser printer	output	human	3.2000
Graphics display	output	human	800.0000–8000.0000
Modem	input or output	machine	0.0160–0.0640
Network/LAN	input or output	machine	100.0000–1000.0000
Network/wireless LAN	input or output	machine	11.0000–54.0000
Optical disk	storage	machine	80.0000
Magnetic tape	storage	machine	32.0000
Magnetic disk	storage	machine	240.0000–2560.0000

An IO interface consists of the circuitry required to connect an input/ output device to a computer system.

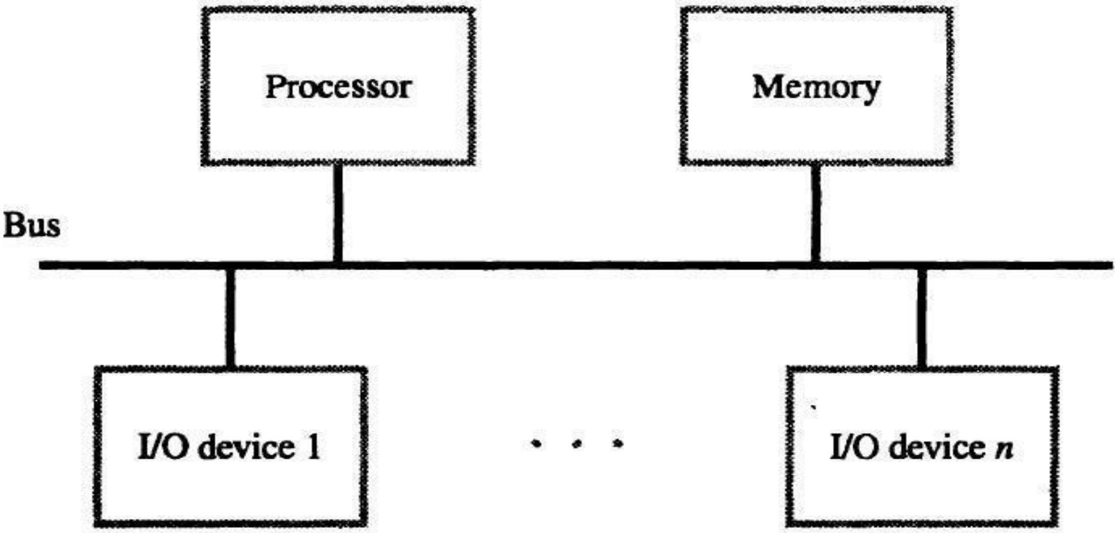


Figure – Bus Connection IO devices to Processor and memory.

A bus is a shared communication link, which uses one set of wires to connect multiple subsystems.

The two major advantages of the bus organization are versatility and low cost.

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, and Control.*
- Processor places a particular address (unique for an I/O Dev.) on addresslines.
- 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.

Hardware to connect I/O devices to bus:

Interface Circuit

- Address Decoder
- Control Circuits
- Data registers
- Status registers

The Registers in I/O Interface – buffer and control.

Flags in Status Registers, like SIN, SOUT

Data Registers, like Data-IN, Data-OUT

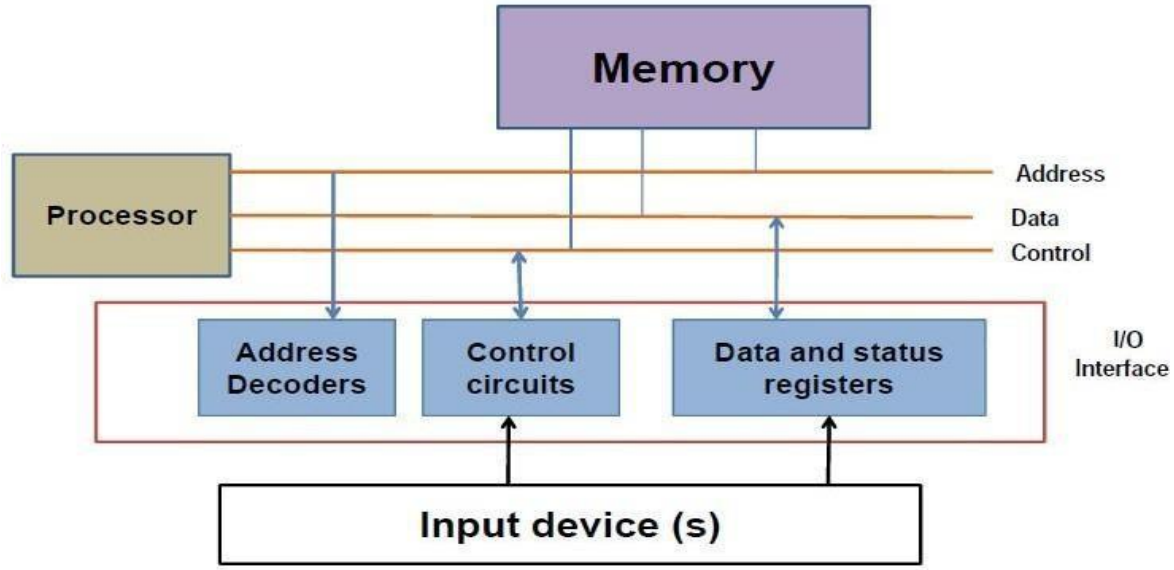


Figure – IO interface for an input device:

Input Output mechanism:

- Memory mappedI/O
- ProgrammedI/O
- Interrupts
- DMA (Direct memory Access)

Memory mapped I/O:

I/O devices and the memoryshare the same address space, the arrangement is called *Memory-mappedI/O*.

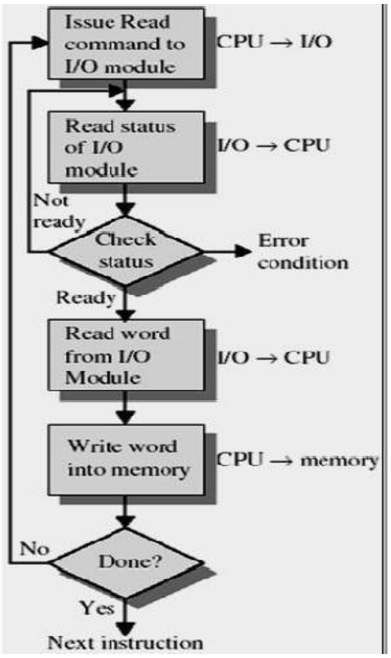
In Memory-mappedI/Oportions of address space are assigned to I/O devices and reads and writes to those addresses are interpreted as commands to the I/Odevice.

“DATAIN” is the address of the input buffer associated with the keyboard.

- Move DATAIN,R0

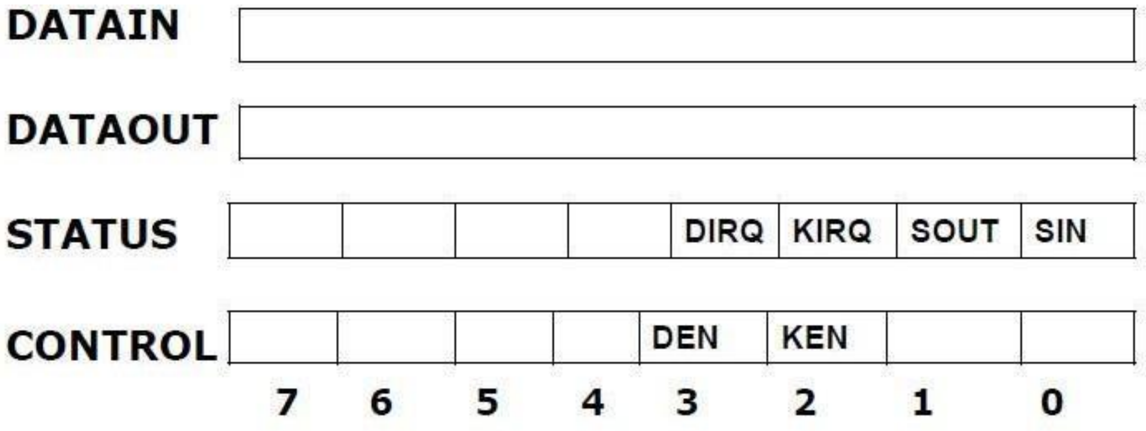
Reads the data from DATAIN and stores them into processor register R0;

- Move R0,DATAOUT



Sends the contents of register R0 to location DATAOUT.

Option of special I/O address space or incorporate as a part of memory address space (address bus is same always).



DATAIN								
DATAOUT								
STATUS					DIRQ	KIRQ	SOUT	SIN
CONTROL					DEN	KEN		
	7	6	5	4	3	2	1	0

Figure – IO operation involving keyboard and display devices.

Registers: DATAIN, DATAOUT, STATUS, CONTROL

Flags: SIN, SOUT - Provides status information for keyboard and display unit
KIRQ, DIRQ – Keyboard, Display Interrupt request bits.
DEN, KEN –Keyboard, Display Enable bits

Programmed I/O
• CPU has direct control over I/O
– Sensing status
– Read/write commands
– Transferring data
• CPU waits for I/O module to
Complete operation
• Wastes CPU time

- In this case, use dedicated I/O instructions in the processor. These I/O instructions can specify both the device number and the command word (or the location of the command word in memory).
- The processor communicates the device address via a set of wires normally included as part of the I/O bus. The actual command can be transmitted over the data lines in the bus. (Example - IntelIA-32).
- By making the I/O instructions illegal to execute when not in kernel or supervisor mode, user programs can be prevented from accessing the devices directly.
- The process of periodically checking status bits to see if it is time for the next I/O operation is called polling. Polling is the simplest way for an I/O device to communicate with the processor.
- The I/O device simply puts the information in a Status register, and the processor must come and get the information.
- The processor is totally in control and does all the work.

Interrupts:

When I/O Device is ready, it sends the INTERRUPT signal to processor via a dedicated controller line.
• Using interrupt we are ideally eliminating WAIT period.

I/O
INTERRUPT
Processor

Figure – Interrupts.

• In response to the interrupt, the processor executes the Interrupt Service Routine(ISR).
• All the registers, flags, program counter values are saved by the processor before running ISR

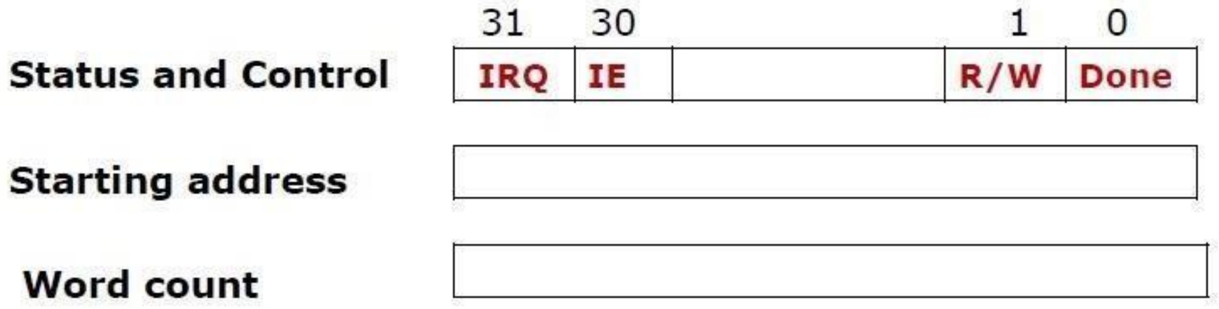
• The time required to save status & restore contribute to execution overhead □ “Interrupt Latency”.

Direct Memory Access (DMA):

For I/O transfer, Processor determines the status of I/O devices, by
– Polling
– Waiting for Interrupt signal
• Considerable overhead is incurred in above I/O transfer processing
• To transfer large blocks of data at high Speed, between EXTERNAL devices & Main Memory, DMA approach is often used
• DMA controller allows data transfer directly between I/O device and Memory, with minimal intervention of processor.

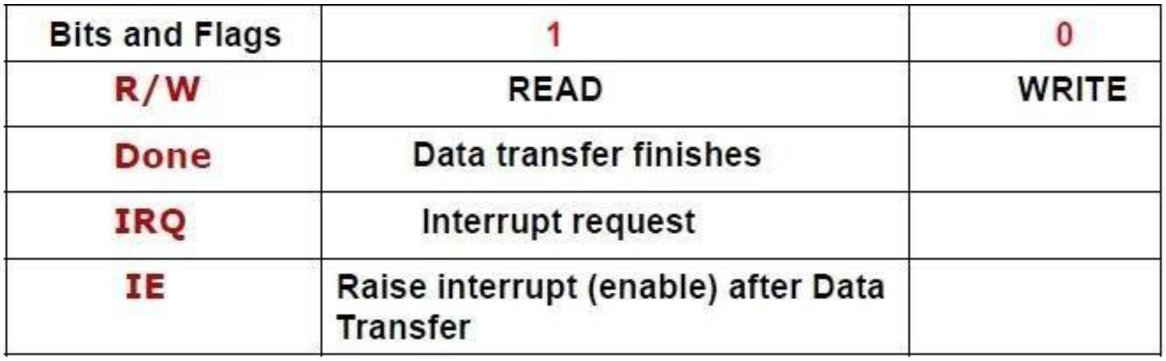
DMA controller acts as a Processor, but it is controlled by CPU
• To initiate transfer of a block of words, the processor sends the following data to controller
– The starting address of the memory block
– The word count
– Control to specify the mode of transfer such as read or write
– A control to start the DMA transfer

DMA controller performs the requested I/O operation and sends a interrupt to the processor upon completion.



In DMA interface

- First register stores the starting address.
- Second register stores Word count.
- Third register contains status and control flags.



Bits and Flags	1	0
R/W	READ	WRITE
Done	Data transfer finishes	
IRQ	Interrupt request	
IE	Raise interrupt (enable) after Data Transfer	

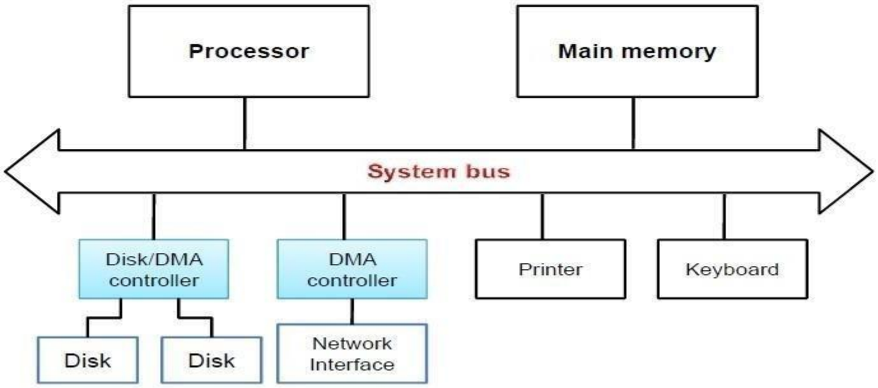


Figure:--Use of DMA Controller in a computer system

IO Interface Circuits:

An I/O interface consists of the circuitry required to connect an input/ output device to a computer system.

On one side of the interface have bus signals for address, data and control.

On the other hand data path with its associated controls to transfer data between the interface and the input/ output device.

This side is called a port. It can be classified as

1. Serial port
2. Parallel port

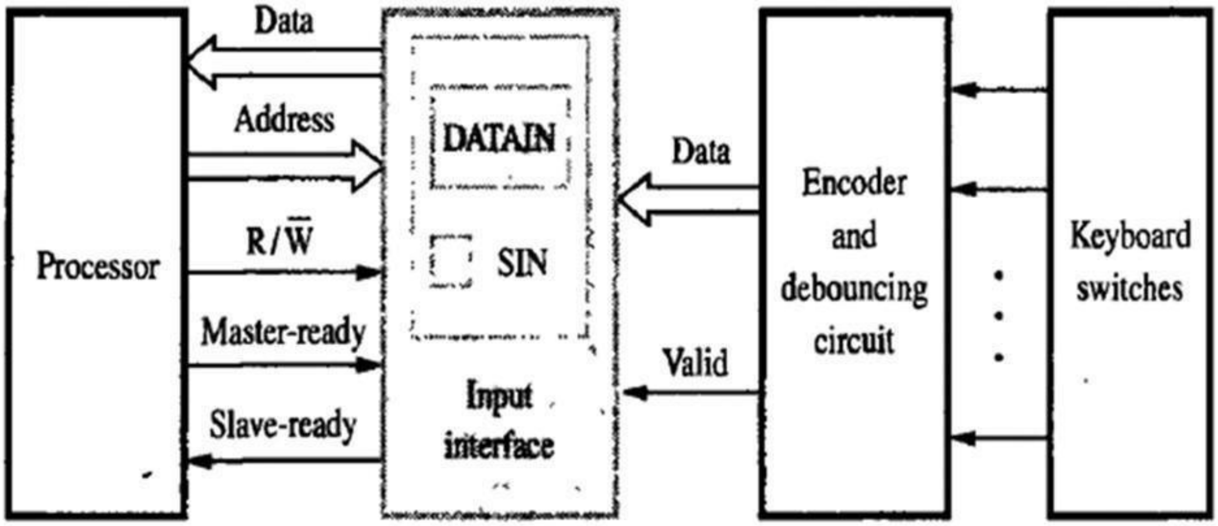


Figure – Keyboard to Processor Connections Interface

Parallel Port:

- It transfers data simultaneously to (or) from the device.
- It uses multiple pin connector.
- Circuit is simple.
- Parallel port is used to send (or) receive data having group of bits(8 bits or 16 bits)simultaneously.
- Parallel ports are classified as input port and output port. Input port – used to receive the data
- Output port- used to send the data.
- It transfers data simultaneously to (or) from the device. It uses multiple pin connector.
- Circuit is simple.

Serial port:

- It transmits and receives data one bit at a time.
- For long distance, it is convenient and cost effective.
- It is used to transmit/ receive data serially. i.e one at a time.
- A key feature of an interface circuit in serial port is that it is capable of communicating in a bit-serial on the device side and in a bit-parallel on the bus side.

I/O interface:

1. It provides a storage buffer for at least one word of data.

2. Contains status flags that can be accessed by the processor to determine whether the buffer is full(or) empty.
3. Address decoding circuitry to determine when it is being addressed by the processor.
4. Performs any format conversion that may be necessary to transfer data between the bus and the input/ output device such as parallel – serial conversion in the case of a serial port.

Input port:

- Commonly used i/p device is a keyboard.
- A key is pressed , corresponding signal alters and encoder circuits generates ASCII code for the corresponding key.

Output port:

- The output port contains a data register DATA OUT and a status flag SOUT.
- SOUT set to 1- when output device ready to accept another character. When it is cleared to 0 then the processor load the data.

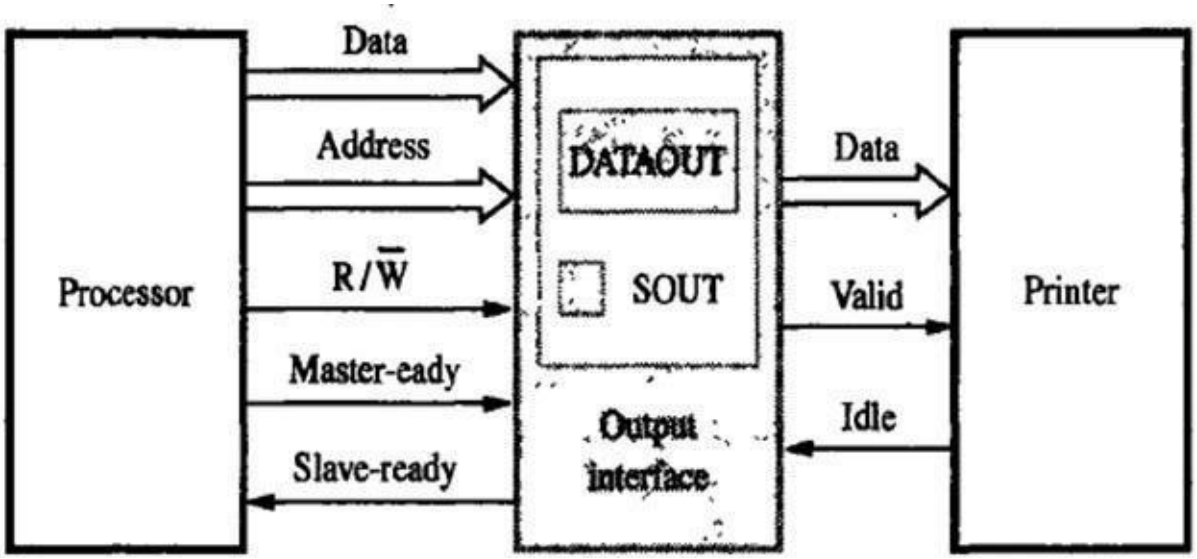
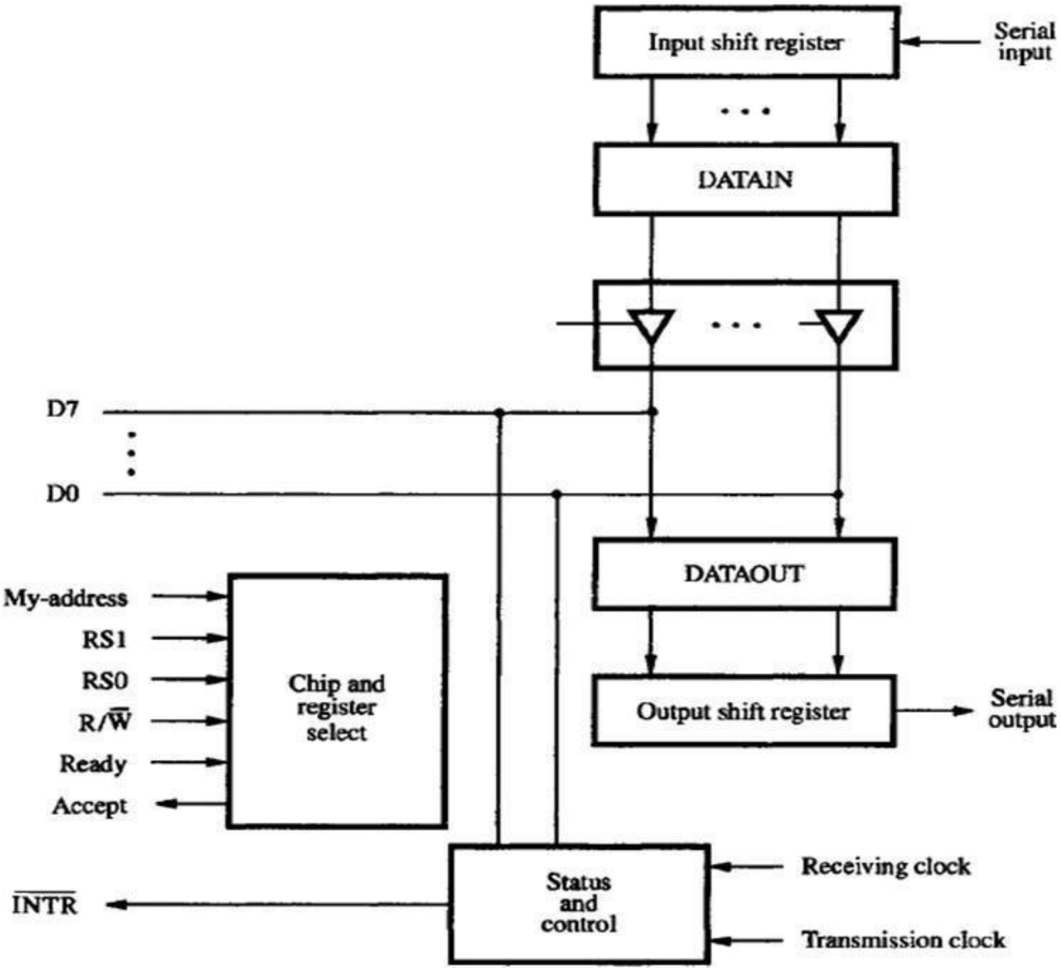


Figure – Printer to Processor Connection

Serial port:

- It is used to transmit/ receive data serially. i.e one at a time.
- A key feature of an interface circuit in serial port is that it is capable of communicating in a bit-serial on the device side and in a bit-parallel on the bus side.
- The input shift register accepts bi-serial input from input/output device.
- When all 8 bits of data have been received, the contents of this shift register are loaded in parallel into the DATAIN register.
- Similarly, output data in the DATAOUT register are loaded into the output shift register.
- The status flag SIN -1 new data loaded in DATAINSIN0- processor read the data of DATAIN

- SOUT 0 –processor writes new data into DATA OUT
- SOUT 1- DATAOUT to output shift register.



Handshaking:

The handshaking method solves the problem of strobe method by introducing a second control signal that provides a reply to the unit that initiates the transfer.

Principle of Handshaking:
The basic principle of the two-wire handshaking method of data transfer is as follow:
One control line is in the same direction as the data flows in the bus from the source to destination. It is used by source unit to inform the destination unit whether there a valid data in the bus. The other control line is in the other direction from the destination to the source. It is used by the destination unit to inform the source whether it can accept the data. The sequence of control during the transfer depends on the unit that initiates the transfer.

Source Initiated Transfer using Handshaking:

The sequence of events shows four possible states that the system can be at any given time. The source unit initiates the transfer by placing the data on the bus and enabling its **data valid** signal. The **data accepted** signal is activated by the destination unit after it accepts the data from the bus. The source unit then disables its **data accepted signal** and the system goes into its initial state.

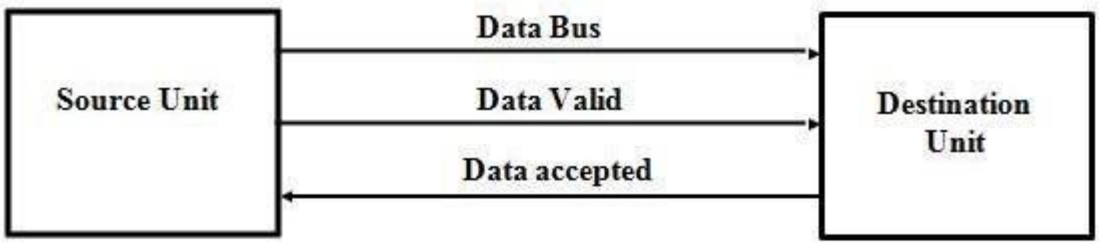


Figure – Block Diagram

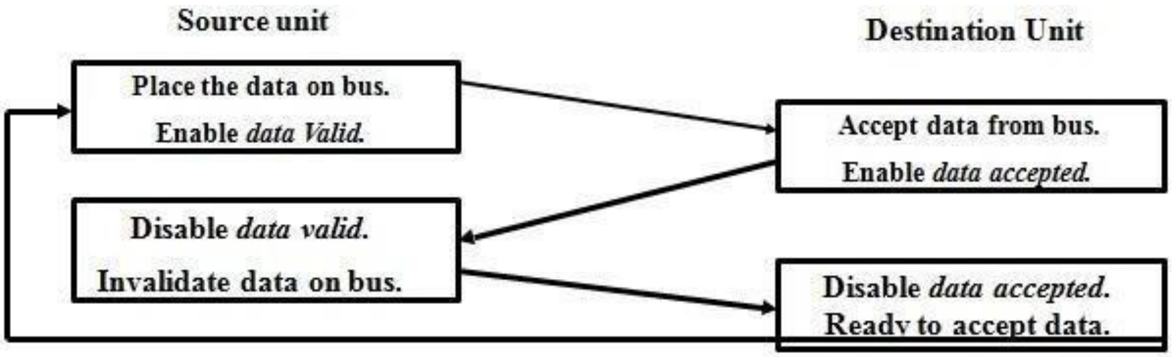
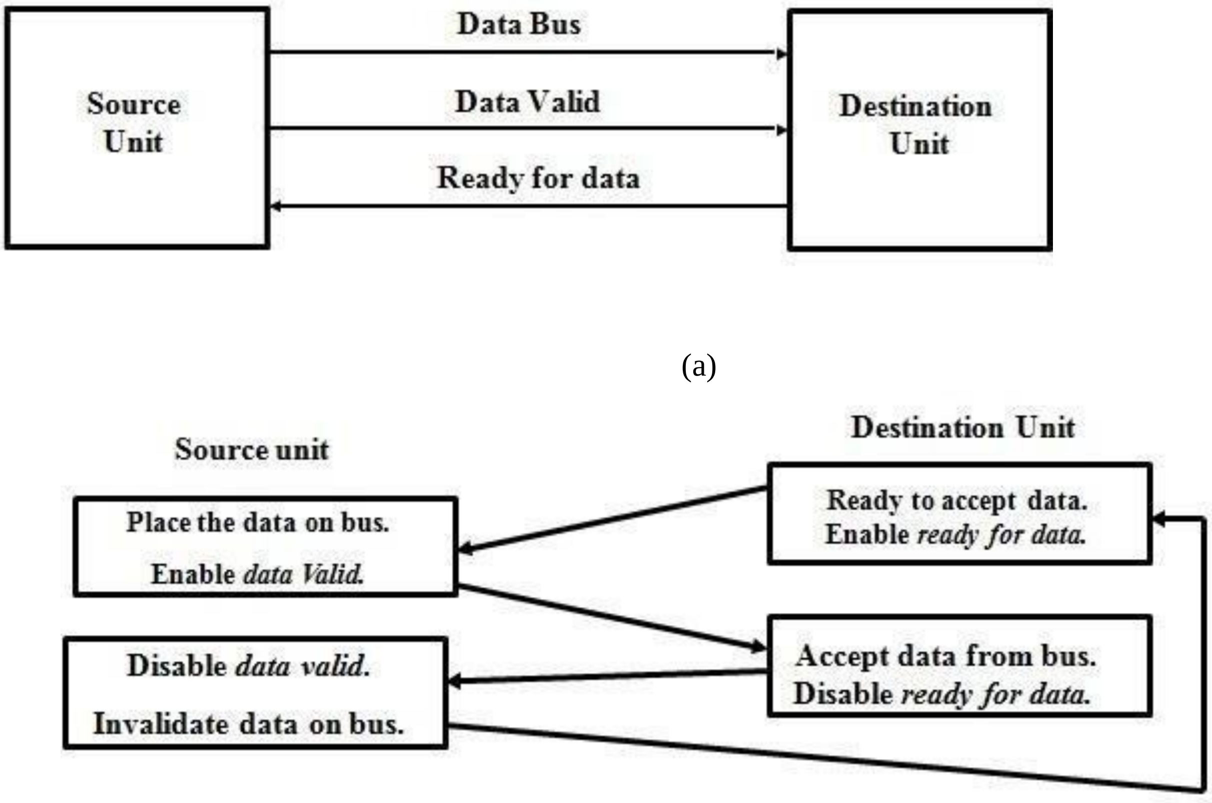


Figure – Sequence of Events

Destination Initiated Transfer Using Handshaking:

The name of the signal generated by the destination unit has been changed to **ready for data** to reflects its new meaning. The source unit in this case does not place data on the bus until after it receives the **ready for data** signal from the destination unit. From there on, the handshaking procedure follows the same pattern as in the source initiated case.

The only difference between the Source initiated and the Destination initiated transfer is in their choice of initial sate.



(a)

(b)

Figure – Destination initiated transfer using Handshaking
(a) Block Diagram and (b)Sequence of Events

Advantage of the Handshaking method:

- The Handshaking scheme provides degree of flexibility and reliability because the successful completion of data transfer relies on active participation by both units.
- If any of one unit is faulty, the data transfer will not be completed. Such an error can be detected by means of a ***Timeout mechanism*** which provides an alarm if the data is not completed with in time.

IOP organization:-

Input/output Processor:-

An input-output processor (IOP) is a processor with direct memory access capability. In this, the computer system is divided into a memory unit and number of processors. Each IOP controls and manage the input-output tasks. The IOP is similar to CPU except that it handles only the details of I/O processing. The IOP can fetch and execute its own instructions. These IOP instructions are designed to manage I/O transfers only.

Block Diagram of IOP:-

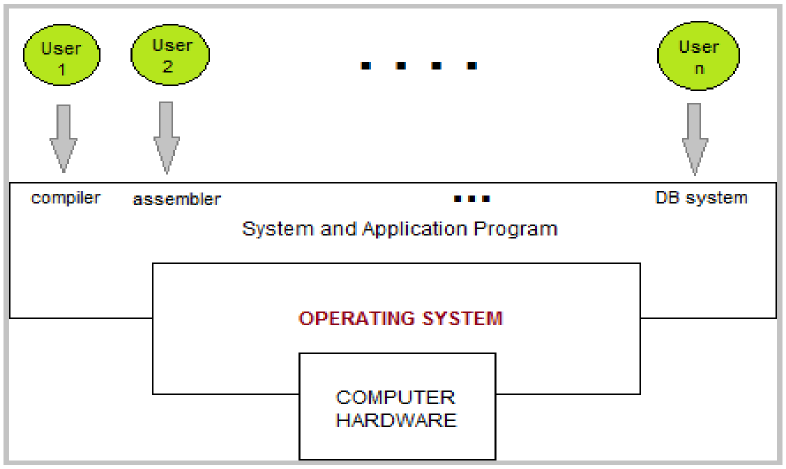
Below is a block diagram of a computer along with various I/O Processors. The memory unit occupies the central position and can communicate with each processor. The CPU processes the data required for solving the computational tasks. The IOP provides a path for transfer of data between peripherals and memory. The CPU assigns the task of initiating the I/O program. The IOP operates independent from CPU and transfer data between peripherals and memory.

The communication between the IOP and the devices is similar to the program control method of transfer. And the communication with the memory is similar to the direct memory access method. In large scale computers, each processor is independent of other processors and any processor can initiate the operation. The CPU can act as master and the IOP act as slave processor. The CPU assigns the task of initiating operations but it is the IOP, who executes the instructions, and not the CPU. CPU instructions provide operations to start an I/O transfer. The IOP asks for CPU through interrupt. Instructions that are read from memory by an IOP are also called *commands* to distinguish them from instructions that are read by CPU. Commands are prepared by programmers and are stored in memory. Command words make the program for IOP. CPU informs the IOP where to find thedata.

OPERATING SYSTEM:-

A computer system has many resources (hardware and software), which may be require to complete a task. The commonly required resources are input/output devices, memory, file storage space, CPU etc. The operating system acts as a manager of the above resources and allocates them to specific programs and users, whenever necessary to perform a particular task. Therefore operating system is the resource manager i.e. it can manage the resource of a computer system internally. The resources are processor, memory, files, and I/O devices. In simple terms, an operating system is the interface between the user and themachine.

Four Components of a Computer System

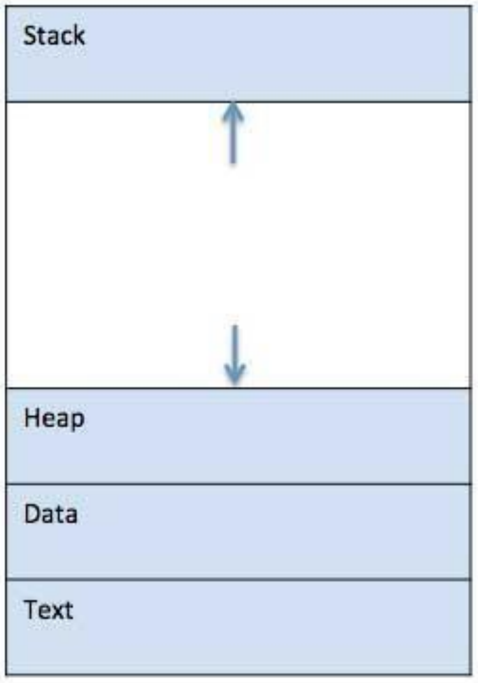


Functions of Operating System

1. Operating system boots the computer.
2. Operating system performs basic computer tasks e.g. managing the various peripheral devices e.g. mouse, keyboard.
3. Operating system provides a user interface, e.g. command line, graphical user interface(GUI).
4. Operating system handles system resources such as computer's memory and sharing of the central processing unit (CPU) time by various applications or peripheral devices.
5. Operating system provides file management which refers to the way that the operating system manipulates stores, retrieves and saves data.
6. Error Handling is done by the operating system. It takes preventive measures whenever required to avoid errors.

Process

A process is basically a program in execution. The execution of a process must progress in a sequential fashion. A process is defined as an entity which represents the basic unit of work to be implemented in the system. To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program. When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data. The following image shows a simplified layout of a process inside main memory−



S. No.	Component	Description
1	**Stack**	The process Stack contains the temporary data such as method/function parameters, return address and local variables.
2	**Heap**	This is dynamically allocated memory to a process during its run time.
3	**Text**	This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.
4	**Data**	This section contains the global and static variables.

Program

A program is a piece of code which may be a single line or millions of lines. A computer program is usually written by a computer programmer in a programming language. For example, here is a simple program written in C programming language −

```
#include

<stdio.h>int

main() {
  printf("Hello, World! \n");
  return 0; }
```

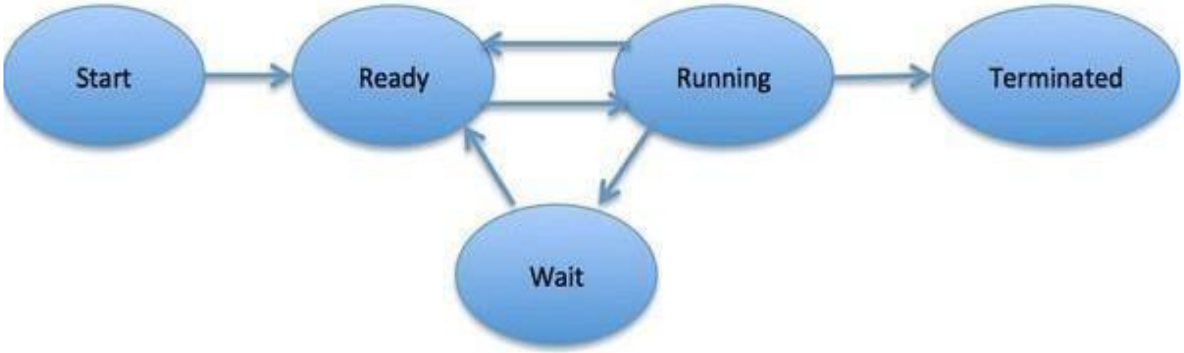
A computer program is a collection of instructions that performs a specific task when executed by a computer. When we compare a program with a process, we can conclude that a process is a dynamic instance of a computer program. A part of a computer program that performs a well-defined task is known as an **algorithm**. A collection of computer programs, libraries and related data are referred to as software.

Process Life Cycle:-

When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized.

In general, a process can have one of the following five states at a time.

S. No.	Component	Description
1	**Start**	This is the initial state when a process is first started/created.
2	**Ready**	The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start** state or while running it by but interrupted by the scheduler to assign CPU to someother process.
3	**Running**	Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.
4	**Waiting**	Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.
5	**Terminated or Exit**	Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.

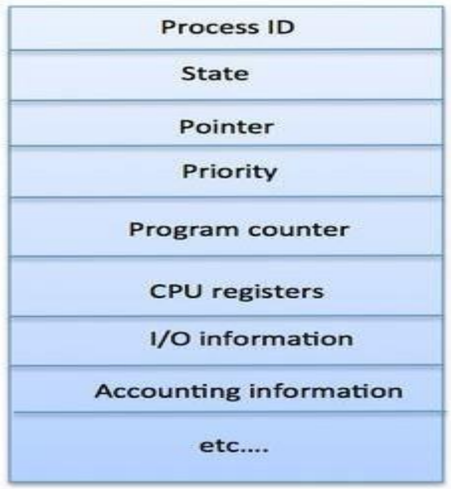


Process Control Block (PCB)

A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process as listed below in the table

S. No.	Component	Description
1	**Process State**	The current state of the process i.e., whether it is ready, running, waiting, or whatever.
2	**Process privileges**	This is required to allow/disallow access to system resources.
3	**Process ID**	Unique identification for each of the process in the operating system.
4	**Pointer**	A pointer to parent process.
5	**Program Counter**	Program Counter is a pointer to the address of the next instruction to be executed for this process.

The architecture of a PCB is completelydependentonOperatingSystemandmaycontaindifferentinformation in different operating systems. Here is a simplified diagram of a PCB



Process ID

State

Pointer

Priority

Program counter

CPU registers

I/O information

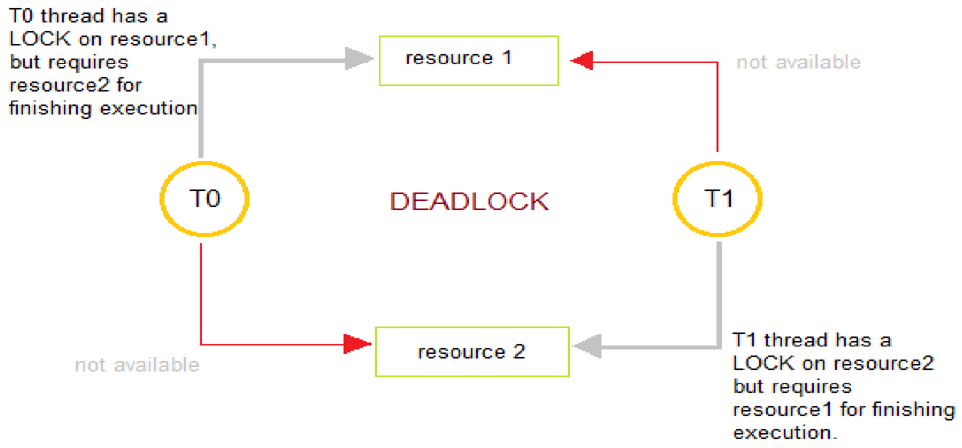
Accounting information

etc....

The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.

Deadlock:-

Deadlocks are a set of blocked processes each holding a resource and waiting to acquire a resource held by another process.


T0 thread has a LOCK on resource1, but requires resource2 for finishing execution
resource 1
not available
T0
DEADLOCK
T1
not available
resource 2
T1 thread has a LOCK on resource2 but requires resource1 for finishing execution.

How to avoid Deadlocks

Deadlocks can be avoided by avoiding at least one of the four conditions, because all this four conditions are required simultaneously to cause deadlock.

1. **Mutual Exclusion**

   Resources shared such as read-only files do not lead to deadlocks but resources, such as printers and tape drives, requires exclusive access by a single process.

2. **Hold and Wait**

   In this condition processes must be prevented from holding one or more resources while simultaneously waiting for one or more others.

3. **No Preemption**

   Preemption of process resource allocations can avoid the condition of deadlocks, where ever possible.

4. **Circular Wait**

   Circular wait can be avoided if we number all resources, and require that processes request resources only in strictly increasing(or decreasing) order.

Handling Deadlock

The above points focus on preventing deadlocks. But what to do once a deadlock has occured. Following three strategies can be used to remove deadlock after its occurrence.

1. **Preemption**

   We can take a resource from one process and give it to other. This will resolve the deadlock situation,but sometimes it does causes problems.

2. **Rollback**

   In situations where deadlock is a real possibility, the system can periodically make a record of the state of each process and when deadlock occurs, roll everything back to the last checkpoint, and restart, but allocating resources differently so that deadlock does not occur.

3. **Kill one or more processes**

   This is the simplest way, but it works.

Multiprocessors:-

A multiprocessor system is an interconnection of two or more CPU's with memory and input- output equipment. Multiprocessors system are classified as multiple instruction stream, multiple data stream systems(MIMD). There exists a distinction between multiprocessor and multi computers that though both support concurrent operations. In multi computers several autonomous computers are connected through a network and they may or may not communicate but in a multiprocessor system there is a single OS Control that provides interaction between processors and all the components of the system to cooperate in the solution of the problem. VLSI circuit technology has reduced the cost of the computers to such a low Level that the concept of applying multiple processors to meet system performance requirements has become an attractive design possibility.

Benefits of Multiprocessing:-

1. Multiprocessing increases the reliability of the system so that a failure or error in one part has limited effect on the rest of the system. If a fault causes one processor to fail, a second processor can be assigned to perform the functions of the disabled one.
2. Improved System performance. System derives high performance from the fact that computations can proceed in parallel in one of the two ways:
   a) Multiple independent jobs can be made to operate in parallel.
   b) A single job can be partitioned into multiple parallel tasks. This can be achieved in two ways:
      i). The user explicitly declares that the tasks of the program be executed in parallel.
      ii). The compiler provided with multiprocessor s/w that can automatically detect parallelism in program. Actually it checks for Data dependency.

Coupling of processors:-

Tightly Coupled System/Shared Memory:-

1. Tasks and/or processors communicate in a highly synchronized fashion
2. Communicates through a common global shared memory
3. Shared memory system doesn’t preclude each processor from having its own local memory(cache memory)

Loosely Coupled System/Distributed Memory:-

1. Tasks or processors do not communicate in a synchronized fashion.
2. Communicates by message passing packets consisting of an address, the data content, and some error detection code.
3. Overhead for data exchange is high.
4. Distributed memory system.

Memory

Shared (Global) Memory:-

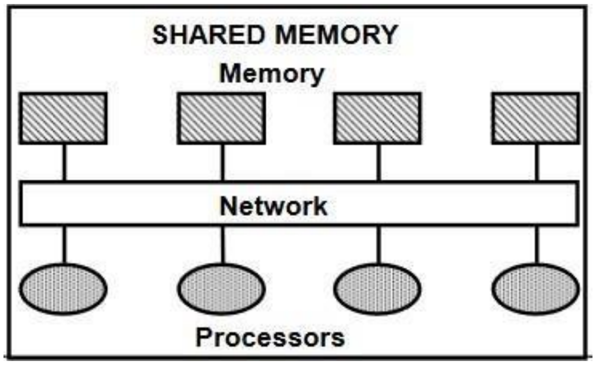
1. A Global Memory Space accessible by all processors.
2. Processors may also have some local memory.

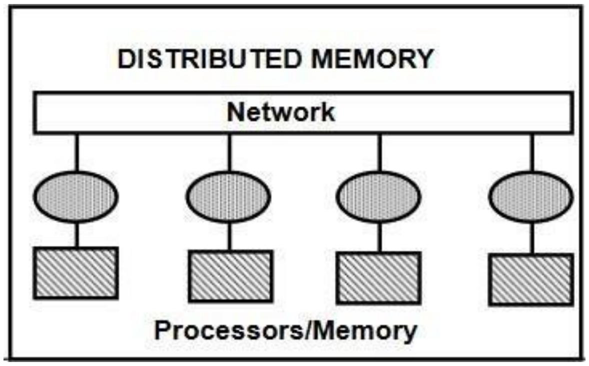
Distributed (Local, Message-Passing) Memory:-

1. All memory units are associated with processors
2. To retrieve information from another processor's memory a message must be sent.

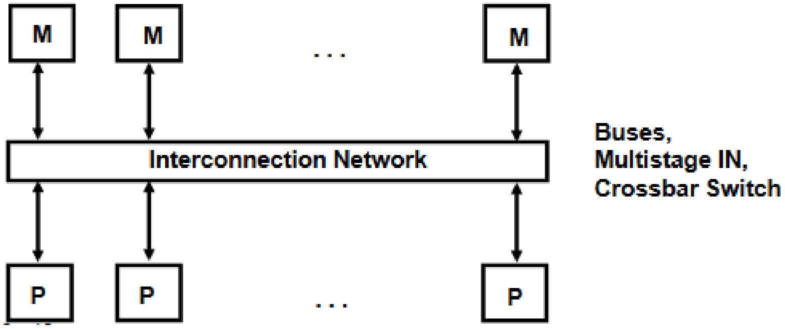
Uniform Memory:-

1. All processors take the same time to reach all memory locations Non uniform (NUMA)Memory.
2. Memory access is not uniform.





Shared Memory Multiprocessors:-

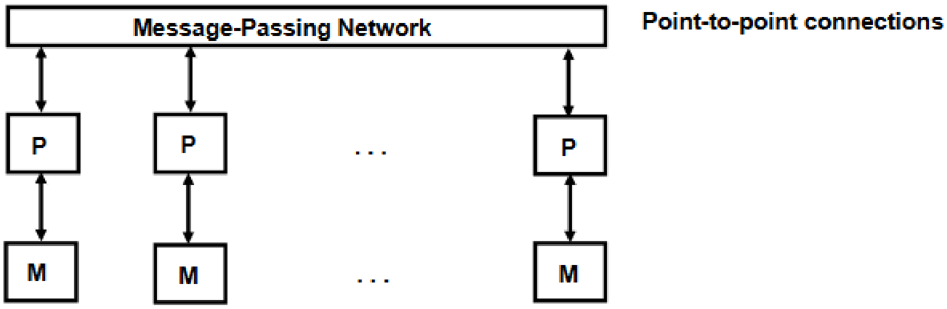


All processors have equally direct access to one large memory address space.

Disadvantage:-

1. Memory access latency.
2. Hot spot problem.

Message Passing Multiprocessors:-



In Message Passing Multiprocessors, all the computers are interconnected to each other. Each processor has its own memory and communicate via message-passing.

Disadvantage:-

1. Communication is over headed.
2. It is hard to programming.

Fault tolerance:-

Fault-tolerant technology is a capability of a computer system, electronic system or network to deliver uninterrupted service, despite one or more of its components failing. Fault tolerance also resolves potential service interruptions related to software or logic errors. The purpose is to prevent catastrophic failure that could result from a single point of failure. VMware vSphere 6 Fault Tolerance is a branded, continuous data availability architecture that exactly replicates a VMware virtual machine on an alternate physical host if the main host server fails. Fault-tolerant systems are designed to compensate for multiple failures. Such systems automatically detect a failure of the computer processor unit, I/O subsystem, memory cards, motherboard, power supply or network components. The failure point is identified, and a backup component or procedure immediately takes its place with no loss of service. To ensure fault tolerance, enterprises need to purchase an inventory of formatted computer equipment and a secondary uninterruptible power supply device. The goal is to prevent the crash of key systems and networks, focusing on issues related to uptime and downtime. Fault tolerance can be provided with software embedded in hardware, or by some combination of the two.

In a software implementation, the operating system (OS) provides an interface that allows a programmer to checkpoint critical data at predetermined points within a transaction. In a hardware implementation (for example, with Stratus and its Virtual Operating System), the programmer does not need to be aware of the fault-tolerant capabilities of the machine. At a hardware level, fault tolerance is achieved by duplexing each hardware component. Disks are mirrored. Multiple processors are lock stepped together and their outputs are compared for correctness. When an anomaly occurs, the faulty component is determined and taken out of service, but the machine continues to function as usual.

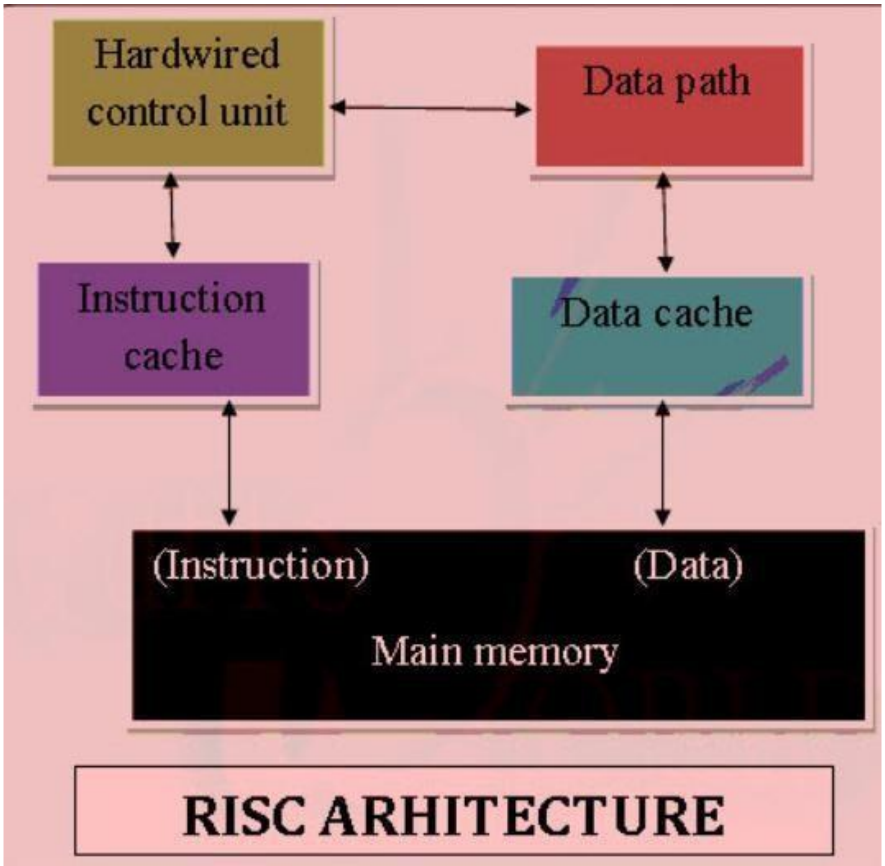
Fault tolerance vs. high availability:-

Fault tolerance is closely associated with maintaining business continuity via highly available computer systems and networks. Fault-tolerant environments are defined as those that restore service instantaneously following a service outage, whereas a high-availability environment strives for five nines of operational service.

In a high-availability cluster, sets of independent servers are coupled loosely together to guarantee system-wide sharing of critical data and resources. The clusters monitor each other's health and provide fault recovery to ensure applications remain available. Conversely, a fault-tolerant cluster consists of multiple physical systems that share a single copy of a computer's OS. Software commands issued by one system are also executed on the other system. The tradeoff between fault tolerance and high availability is cost. Systems with integrated fault tolerance incur a higher cost due to the inclusion of additional hardware.

RISC processors:-

RISC (Reduced Instruction Set Computer) is used in portable devices due to its power efficiency. For Example, Apple iPod and Nintendo DS. RISC is a type of microprocessor architecture that uses highly-optimized set of instructions. RISC does the opposite, reducing the cycles per instruction at the cost of the number of instructions per program Pipelining is one of the unique feature of RISC. It is performed by overlapping the execution of several instructions in a pipeline fashion. It has a high performance advantage over CISC.

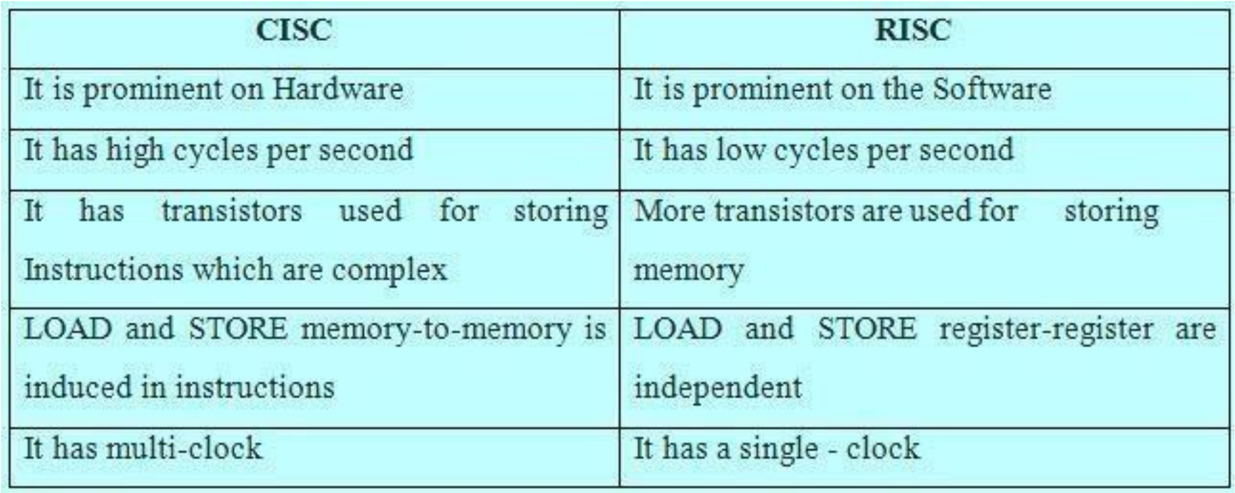


RISC processors take simple instructions and are executed within a clock cycle

RISC ARCHITECTURE CHARACTERISTICS:-

1. Simple Instructions are used in RISC architecture.
2. RISC helps and supports few simple data types and synthesize complex data types.
3. RISC utilizes simple addressing modes and fixed length instructions for pipelining.
4. RISC permits any register to use in any context.
5. One Cycle Execution Time
6. The amount of work that a computer can perform is reduced by separating “LOAD” and “STORE” instructions.
7. RISC contains Large Number of Registers in order to prevent various number of interactions with memory.
8. In RISC, Pipelining is easy as the execution of all instructions will be done in a uniform interval of time i.e. one click.
9. In RISC, more RAM is required to store assembly level instructions.
10. Reduced instructions need a less number of transistors in RISC.
11. RISC uses Harvard memory model means it is Harvard Architecture.
12. A compiler is used to perform the conversion operation means to convert a high-level language statement into the code of its form.

RISC & CISC Comparison:-



CISC	RISC
It is prominent on Hardware	It is prominent on the Software
It has high cycles per second	It has low cycles per second
It has transistors used for storing Instructions which are complex	More transistors are used for storing memory
LOAD and STORE memory-to-memory is induced in instructions	LOAD and STORE register-register are independent
It has multi-clock	It has a single - clock

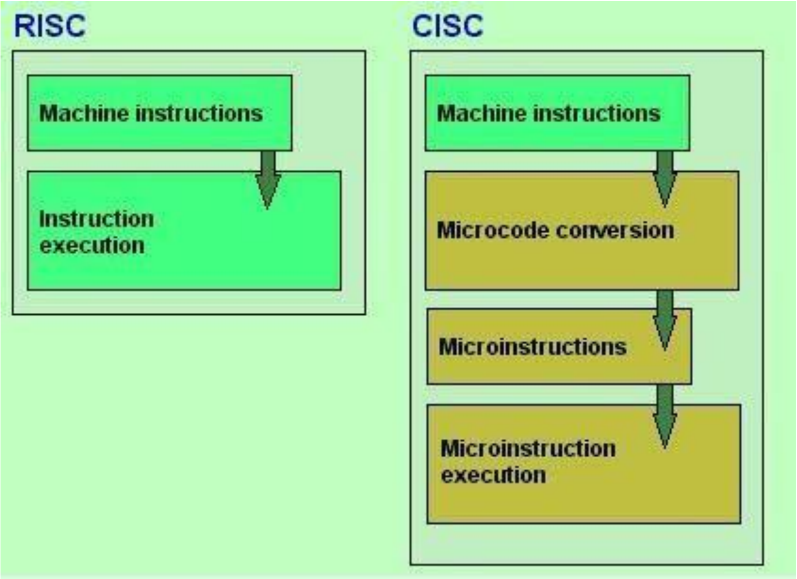
Comparison between CISC & RISC:-

MUL instruction is divided into three instructions

1. “LOAD” – moves data from the memory bank to a register.
2. “PROD” – finds product of two operands located within the registers.

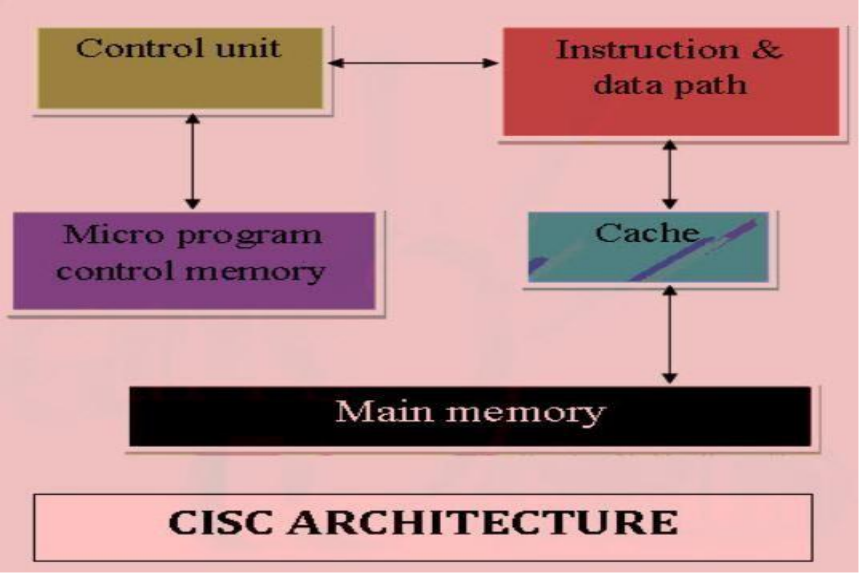
3. "STORE" – moves data from a register to the memory banks.

The main difference between RISC and CISC is the number of instructions and its complexity.



CISC Processors:

The CISC approach attempts to minimize the number of instructions per program, sacrificing the number of cycles per instruction. Computers based on the CISC architecture are designed to decrease the memory cost. Because, the large programs need more storage, thus increasing the memory cost and large memory becomes more expensive. To solve these problems, the number of instructions per program can be reduced by embedding the number of operations in a single instruction, thereby making the instructions more complex.



1. MUL loads two values from the memory into separate registers in CISC.
2. CISC uses minimum possible instructions by implementing hardware and executes operations.
3. Instruction Set Architecture is a medium to permit communication between the programmer and the hardware. Data execution part, copying of data, deleting or editing is the user commands used in the microprocessor and with this microprocessor the Instruction set architecture is operated.

The main keywords used in the above Instruction Set Architecture are as below

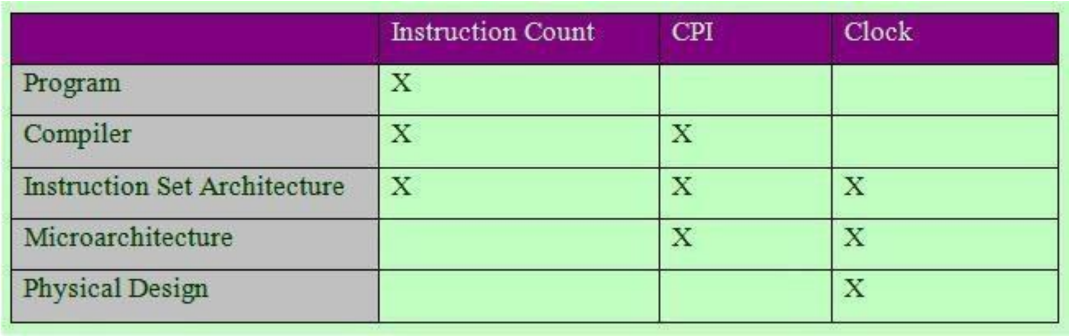
Instruction Set: Group of instructions given to execute the program and they direct the computer by manipulating the data. Instructions are in the form – Opcode (operational code) and Operand. Where, opcode is the instruction applied to load and store data, etc. The operand is a memory register where instruction applied.

Addressing Modes: Addressing modes are the manner in the data is accessed. Depending upon the type of instruction applied, addressing modes are of various types such as direct mode where straight data is accessed or indirect mode where the location of the data is accessed. Processors having identical ISA may be very different in organization. Processors with identical ISA and nearly identical organization is still not nearly identical.

CPU performance is given by the fundamental law

$$CPU\,Time = \frac{Seconds}{Program} = \frac{Instructions}{Program} X \frac{Cycles}{Instructions} X \frac{Seconds}{Cycle}$$

Thus, CPU performance is dependent upon Instruction Count, CPI (Cycles per instruction) and Clock cycle time. And all three are affected by the instruction set architecture.



	Instruction Count	CPI	Clock
Program	X		
Compiler	X	X	
Instruction Set Architecture	X	X	X
Microarchitecture		X	X
Physical Design			X

Instruction Count of the CPU

This underlines the importance of the instruction set architecture. There are two prevalent instruction set architectures

Examples of CISC PROCESSORS

1. **IBM 370/168:–** It was introduced in the year 1970. CISC design is a 32 bit processor and four 64-bit floating point registers.
2. **VAX 11/780:–** CISC design is a 32-bit processor and it supports many numbers of addressing modes and machine instructions which is from Digital Equipment Corporation.
3. **Intel 80486:–** It was launched in the year 1989 and it is a CISC processor, which has instructions varying lengths from 1 to 11 and it will have 235instructions.

CHARACTERISTICS OF CISC ARCHITECTURE:-

1. Instruction-decoding logic will be Complex.
2. One instruction is required to support multiple addressing modes.
3. Less chip space is enough for general purpose registers for the instructions that are 0operated directly on memory.
4. Various CISC designs are set up two special registers for the stack pointer, handling interrupts, etc.
5. MUL is referred to as a “complex instruction” and requires the programmer for storing functions.

Superscalar and vector processor:-

A **Scalar processor** is a normal processor, which works on simple instruction at a time, which operates on single data items. But in today's world, this technique will prove to be highly inefficient, as the overall processing of instructions will be very slow.

Vector(Array) Processing:-

There is a class of computational problems that are beyond the capabilities of a conventional computer. These problems require vast number of computations on multiple data items, that will take a conventional computer(with scalar processor) days or even weeks to complete.

Such complex instructions, which operates on multiple data at the same time, requires a better way of instruction execution, which was achieved by Vector processors.

Scalar CPUs can manipulate one or two data items at a time, which is not very efficient. Also, simple instructions like **ADD A to B, and store into C** are not practically efficient. Addresses are used to point to the memory location where the data to be operated will be found, which leads to added overhead of data lookup. So until the data is found, the CPU would be sitting ideal, which is a big performance issue. Hence, the concept of **Instruction Pipeline** comes into picture, in which the instruction passes through several sub-units in turn. These sub-units perform various independent functions, **for example**: the **first** one decodes the instruction, the **second** sub-unit fetches the data and the **third** sub-unit performs the math itself. Therefore, while the data is fetched for one instruction, CPU does not sit idle, it rather works on decoding the next instruction set, ending up working like an assembly line. Vector processor, not only use Instruction pipeline, but it also pipelines the data, working on multiple data at the same time. A normal scalar processor instruction would be ADD A, B, which leads to addition of two operands, but what if we can instruct the processor to ADD a group of numbers(from 0 to n memory location) to another group of numbers(lets say, n to k memory location). This can be achieved by vector processors.

In vector processor a single instruction, can ask for multiple data operations, which saves time, as instruction is decoded once, and then it keeps on operating on different data items.

Applications of Vector Processors:-

Computer with vector processing capabilities are in demand in specialized applications. The following are some areas where vector processing is used:

1. Petroleum exploration.
2. Medical diagnosis.
3. Data analysis.
4. Weather forecasting.
5. Aerodynamics and space flight simulations.
6. Image processing.
7. Artificial intelligence.

Superscalar Processors:-

It was first invented in 1987. It is a machine which is designed to improve the performance of the scalar processor. In most applications, most of the operations are on scalar quantities. Superscalar approach produces the high performance general purpose processors. The main principle of superscalar approach is that it executes instructions independently in different pipelines. As we already know, that Instruction pipelining leads to parallel processing thereby speeding up the processing of instructions. In Superscalar processor, multiple such pipelines are introduced for different operations, which further improves parallel processing.

There are multiple functional units each of which is implemented as a pipeline. Each pipeline consists of multiple stages to handle multiple instructions at a time which support parallel execution of instructions. It increases the throughput because the CPU can execute multiple instructions per clock cycle. Thus, superscalar processors are much faster than scalar processors.

A **scalar processor** works on one or two data items, while the **vector processor** works with multiple data items. A **superscalar processor** is a combination of both. Each instruction processes one data item, but there are multiple execution units within each CPU thus multiple instructions can be processing separate data items concurrently.

While a superscalar CPU is also pipelined, there are two different performance enhancement techniques. It is possible to have a non-pipelined superscalar CPU or pipelined non-superscalar CPU. The superscalar technique is associated with some characteristics, these are:

1. Instructions are issued from a sequential instruction stream.
2. CPU must dynamically check for data dependencies.
3. Should accept multiple instructions per clock cycle.

Vector processor:-

Vector (Array) Processor and its Types:-

Array processors are also known as multiprocessors or vector processors. They perform computations on large arrays of data. Thus, they are used to improve the performance of the computer.

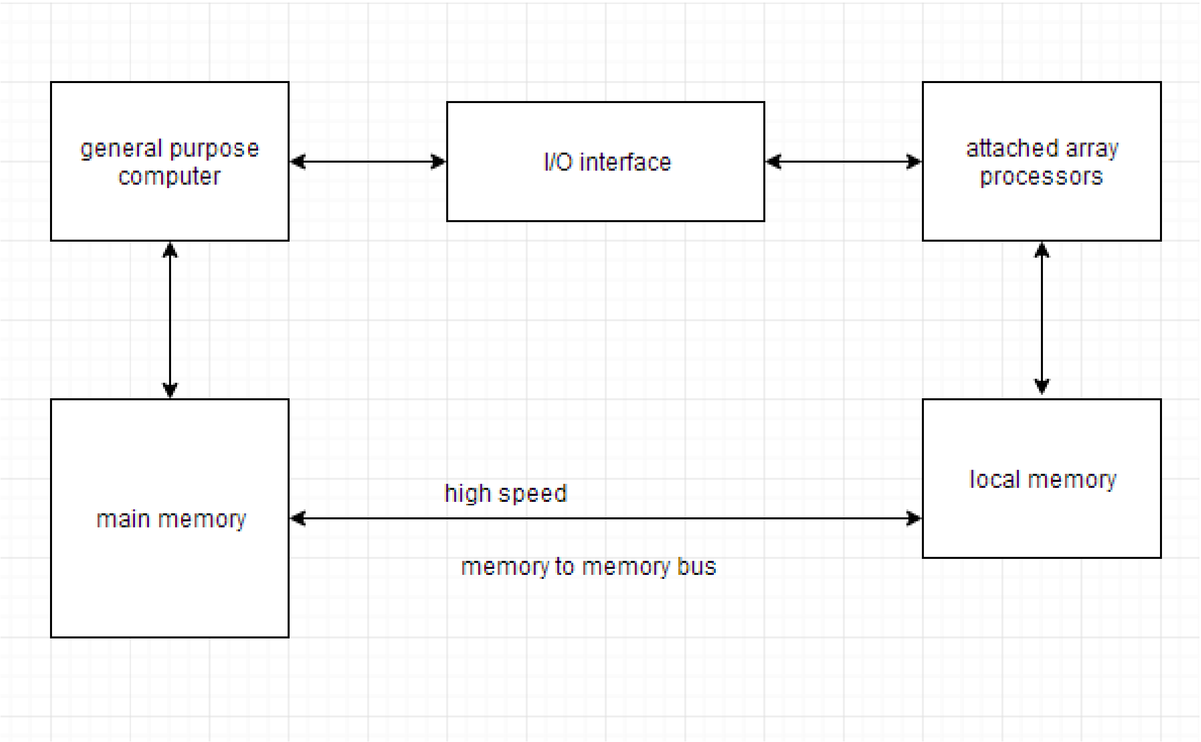
Types of Array Processors:-

There are basically two types of array processors:

1. Attached Array Processors
2. SIMD Array Processors

Attached Array Processors:-

An attached array processor is a processor which is attached to a general purpose computer and its purpose is to enhance and improve the performance of that computer in numerical computational tasks. It achieves high performance by means of parallel processing with multiple functional units.



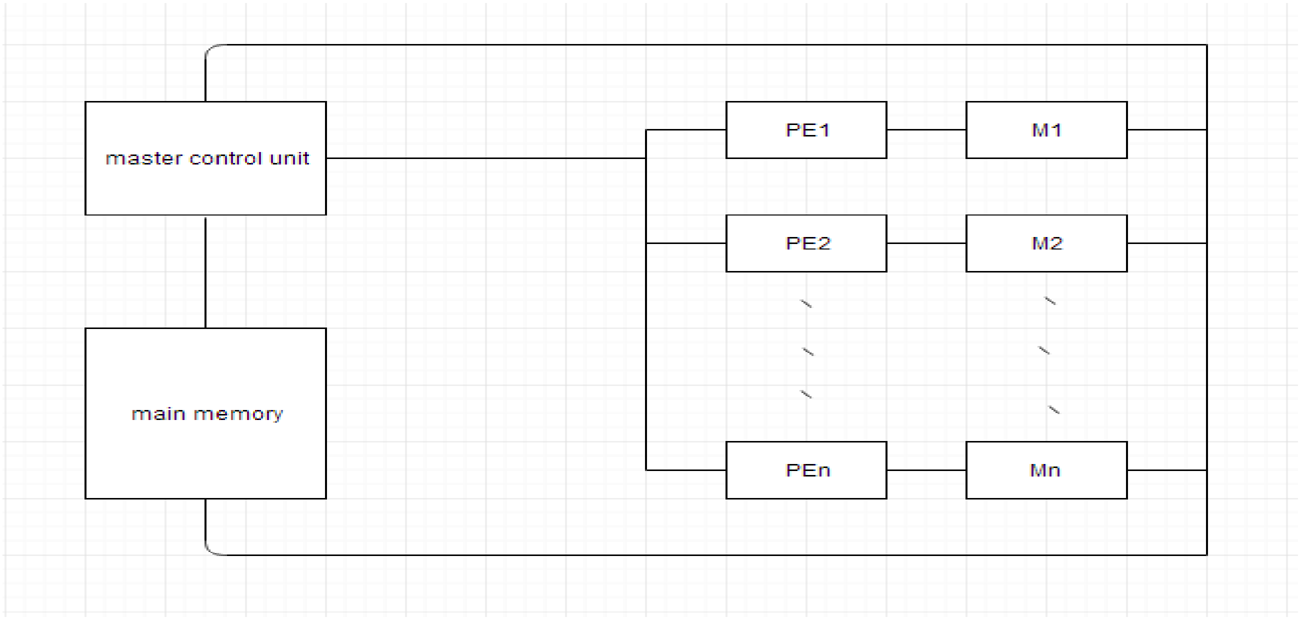
SIMD Array Processors:-

SIMD is the organization of a single computer containing multiple processors operating in parallel. The processing units are made to operate under the control of a common control unit, thus providing a single instruction stream and multiple data streams.

A general block diagram of an array processor is shown below. It contains a set of identical processing elements (PE's), each of which is having a local memory M. Each processor element includes an ALU and registers. The master control unit controls all the operations of the processor elements. It also decodes the instructions and determines how the instruction is to be executed.

The main memory is used for storing the program. The control unit is responsible for fetching the instructions. Vector instructions are send to all PE's simultaneously and results are returned to the memory.

The best known SIMD array processor is the ILLIAC IV computer developed by the Burroughs corps. SIMD processors are highly specialized computers. They are only suitable for numerical problems that can be expressed in vector or matrix form and they are not suitable for other types of computations.



Use of the Array Processor:-

1. Array processors increases the overall instruction processing speed.
2. As most of the Array processors operates asynchronously from the host CPU, hence it improves the overall capacity of the system.
3. Array Processors has its own local memory, hence providing extra memory for systems with low memory.