m3

COMPUTER ORGANIZATION

UNIT-1

→ To have a Thorough understanding of the basic structure and operation of a digital computer

Digital computers

→ The digital computer is a digital system that Performs various computational tasks.

→ The word digital implies information in the Computer is represented by variables that take a limited numbers of discrete values.

→ The first electronic digital Computers developed in the late 1940s, were used Primarily for numerical computations.

→ From the application the term digital computer has emerged. In Practice, digital computers function more reliably if only two States are used.

→ Because human logic tends to binary (true (or) False, Yes (or) No statements)

→ Digital Computers use the binary Number System, which has two digits 0 and 1.

→ A Binary digit is called a bit.

→ By using various coding techniques, groups of bits Can be made to represent not only binary numbers but also other discrete Symbols.

→ such as decimal digits (or) letters of the al

→ The group of bits are used to develop complex sets of instructions for performing various types of computations.

→ Decimal numbers to employ the base 10 system Binary numbers can be found by expanding it into a power series with a base of 2.

For example Binary number 1001011

Quantity that can be converted to a decimal number by multiplying each bit by the base 2

$$1\times2^6+0\times2^5+0\times2^4+1\times2^3+0\times2^2+1\times2^1+1\times2^0=75$$

→ A computer system is sometimes subdivided into two functional entities: hardware and software.

→ The Hardware of the computer consists of all the electronic components and electromechanical devices that comprise the entity of the device.

→ computer software consists of the instruction tasks.

→ A sequence of instructions for the computer is called a Program.

→ A computer system is composed of its hardware and the system software available for its use.

→ The system software of a computer consists of a collection of programs whose purpose is to make more effective use of the computer.

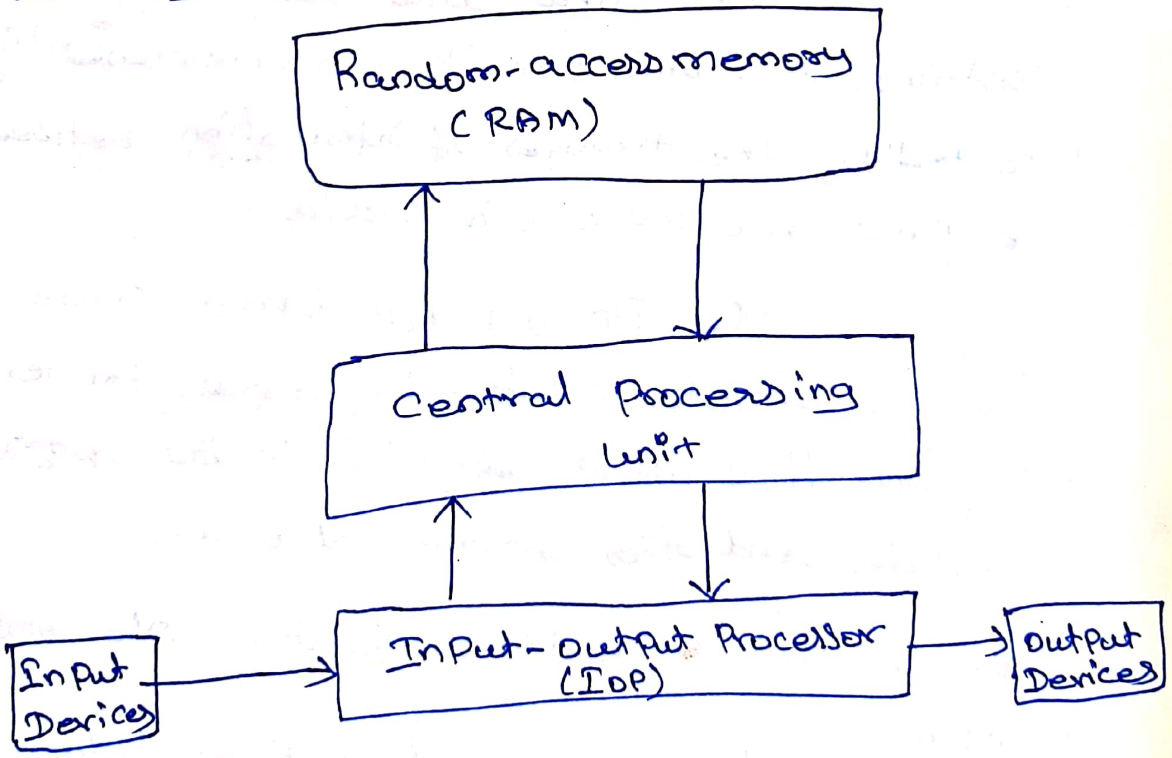
→ The Programs included is a systems Software Package of the computer. as the operating systems.

For ex :-

high level language Program uses to solve particular data Processing needs is an application Program, but the compiler that Program translates the high level language Program to machine language is a system program.

→ The customer who buys a computer system would need, in addition to the hardware, any available software for effective operation of the computer.

BLOCK Diagram of a digital Computer



Computer hardware

The hardware of the computer is usually divided into three major parts, as

CPU → contains an arithmetic and logic unit for manipulating data, a number of registers for storing data and control circuits for fetching and executing instructions.

The memory of a computer contains storage for instructions and data. It is called a random-access Memory (RAM) because the CPU can access any location in memory at random and retrieve in binary information within a fixed interval of time.

The Input and output processor (IOP) contains electronic circuits for communicating and controlling the transfer of information between the computer and the outside world.

The i/p and o/p devices connected to the computer include Keyboards, Printers, terminals, Magnetic disk drives ~~connected to the computer~~ include and other communication devices.

Basic operations of Hardware operations of a computer system.

→ Sometimes considered from three different Points of view,

→ Computer organization → Computer design
→ Computer architecture.

COMPUTER TYPES

→Let us first define the term digital computer or simplyComputer.
→ In the Simplest terms, a contemporary computer is a fast electronic calculating machine that acepts digitized input information, Processes it according to a list of internally Stored Instructions, and Produces the resulting output information.
→ The list of instructions is called a computer Program and the internal Storage is called Computer memory.

Many types of computers exist that differ widely in size, cost, computational Power and intended use.

The most common computer is the Personnal computer, Which has found wide use in homes, Schools and buissness offices. It is the most Common form of desktop computers.

Desktop computers have Processing and Storage units visual display and audio output units, and a Keyboard that can all be located easily on a home or office desk.

The Storage media include hard disks, CD-ROMS, and ^dis Ketter.

Portable notebook computers are a compact version of the Personal computer with all of these Components Packaged into a single unit the size of a thin briefcase.

* workstations with high resolution graphics input output capability, although still retaining the dimensions of desktop computers, have significantly more computational power than personal computers

* workstations are often used in engineering applications, especially for interactive design work.

Beyond workstations, a range of large and very powerful computer systems exist that are called enterprise systems and servers at the low end of the range, and supercomputers at the high end.

Enterprise systems, or mainframes, are used for business data processing in medium to large corporations that require much more computing power and storage capacity than workstations can provide.

Servers contain sizable database and storage units and are capable of handling large volume of requests to access the data.

In many cases servers are widely accessible to the education, buissness and personal uses communities.

The requests and responses are usually transported over Internet communication facilities.

Indeed, the Internet and its associated servers have become dominant worldwide source of all types of information.

→ The Internet communication facilities consists of a Complex structure of high-speed fiber-optic backbone links interconnected with broadcast cable and telephone connections to Schools, businesses and homes.

→ Super computers are ~~used~~ used for the large scale numerical calculations required in applications such as weather forecasting and aircraft design and simulation.

→ In enterprise systems, servers and super computers, the functional units, including multiple processors, may consists of a number of seperate and often large units.

→ when it dealing with Computer hardware it is Customary to distinguish between is referred to as Computer organization, Computer design and Computer architecture.

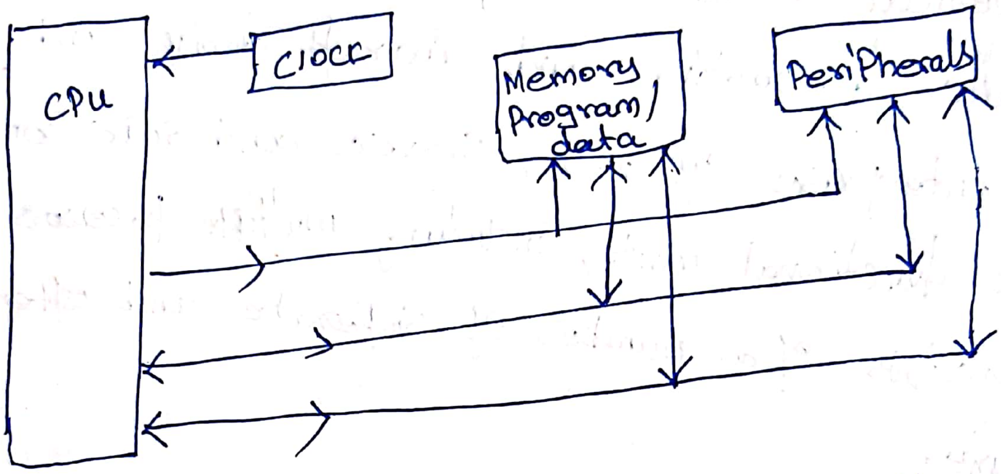
Computer architecture:-

→ It includes the ~~[illegible]~~ information, formats, the Instruction set and techniques for addressing memory.

→ The architectural design of a computer system is concerned with the specifications of the various functional modules such as processors and memories, and structuring them together into a computer system.

VON NEUMAN ARCHITECTURE

→ In 1946, John Von Neuman developed the first computer architecture that allowed the computer to be programmed by codes residing in memory.



In this program instructions were stored in ~~memory~~ ROM.

The Von Neumann architecture is mostly widely used in majority of microprocessors.

In a computer with Von Neumann architecture, the CPU can be either reading an instruction or reading/writing data from/to the memory. Both cannot occur at the same time since the instruction and data use the same signal pathways and memory.

The Von Neumann architecture consists of three buses

1. Databus
2. Address bus
3. Control bus.

The Data bus

Transports data between CPU and its Peripherals. It is bidirectional. The cpu can read or write data in the Peripherals.

The Address bus

The cpu uses the address bus to indicate which peripherals it wants to access and within each peripheral which specific register. The address bus is unidirectional.

The CPU always writes the address, which it read by the Peripherals.

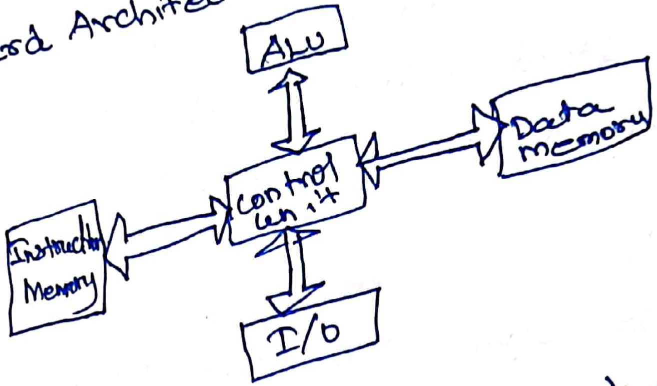
Control bus

The bus carries signals that are used to manage and synchronize the exchange between the CPU and its Peripherals, as well as that indicates if the cpu wants to read or write the Peripheral.

The main characteristics of the Von Neumann Architecture is that it only Possesses 1 Bus System.

The Same Bus carries all the information exchanged between the cpu and the Peripherals including the instruction Codes as well as the Data Processed by the CPU.

Harvard Architecture



→ The Harvard architecture is a computer architecture with physically seperate storage and signal pathways of instructions and data.

→ The term orginated from the Harvard Mark I ~~reply~~ relay based computer, which stored instructions on punched tape (24 bits wide) and data in electromechanical counters

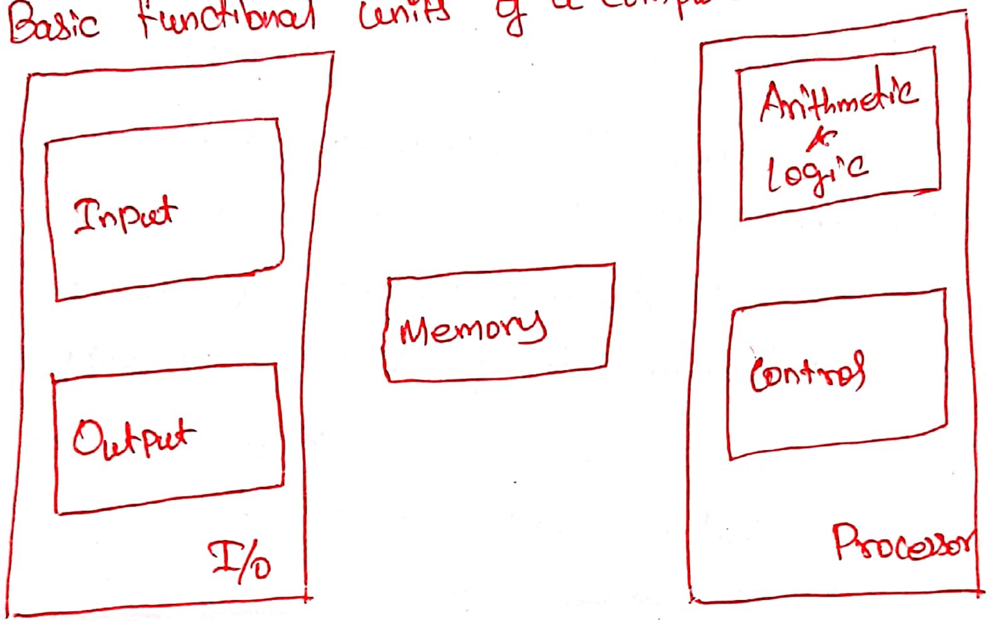
These early machines had data storage entirely contained with in the central processing unit, and provides no access to the instruction storage as data. Programs needed to be loaded by an operator; the processor could not initialize itself.

Today most processors implement such seperate signal pathways for performance reasons, but actually implement a modified Harvard architecture, so they can support tasks like loading a program from disk storage and then executing it.

FUNCTIONAL UNITS

A computer consists of five functionally independent main parts : input, memory, arithmetic and logic unit, output and control units.

Basic functional units of a computer



The input unit accepts the coded information from human operators, from electromechanical devices such as keyboards, or from other computers over digital communication lines.

The information received is either stored in the computer's memory for later reference or immediately used by the arithmetic and logic circuitry to perform the desired operations.

The processing steps are determined by a program ~~the~~ stored in the memory.

→ Finally, the results are sent back to the outside world through the output unit.

→ All of these actions are coordinated by the control unit.

→ These connections, which can be made in several ways, are discussed throughout

→ We refer the Arithmetic and logic circuits, in conjunction with the main control circuits, as the Processor, and input and output equipment is often collectively referred to as the Input-output (I/o) unit

→ To categorize this information as either Instruction or data. Instructions, or machine Instructions are explicit commands that

* Govern the transfer of information within a computer as well as between the computer and its I/o devices.

* specify the arithmetic logic operations to be Performed.

→ A List of instructions that Performs a task is called a Program. usually the Program is stored in the memory.

→ The Processor then fetches the Instructions that makeup the Program from the memory, one after another, and Performs the desired operations.

→ The computer is completely controlled by the stored Program, except for Possible external Interruption by an operator or by I/o devices connected to the machine.

→ The term data, however is often used to mean any digital information. Within this definition of data, an entire program.

An example ↳ This is the task of compiling a highlevel language program into a list of machine instructions constituting a machine language program, called the object program.

The source program is the input data to the compiler program which translates the source program into a machine language program.

Alphanumeric characters are also expressed in terms of binary codes.

Several coding schemes have been developed
Two of the mostly widely used schemes are

ASCII (American Standard Code for Information Interchange) which represented 7-bit code.

EBCDIC - (Extended binary-coded Decimal Interchange code) eightbits are used to denote a character

INPUT UNIT :-

Computers accept coded information through input units, which read the data.

The most well-known input device is Keyboard.

Scanned by CamScanner

* whenever a key is pressed, the correst letter or digit is automatically ~~transmitted~~ translated into its corresponding binary code and transmitted over a cable to either the memory or the processor.

Many other kinds of input devices are available, including joysticks, trackballs, and Mouses.

These are often used as graphic input devices in conjunction with displays.

Microphones can be used to capture audio input which is then sampled and converted into digital codes for storage and processing.

Memory unit

The function of the Memory unit is to store programs and data. These are two classes of storage, called Primary and secondary.

→ Primary storage is a fast memory that operates at electronic speeds

→ Programs must be stored in the memory while they are being executed.

* The memory contains a large number of semiconductor ~~devices~~ storage cells, each capable of storing one bit of information.

These cells are rarely read or written as individual cells but instead are processed in groups of fixed size called words.

The number of bits in each word is often reffered to as the word length of the computer.

Typical word lengths range from 16 to 64 bits. The capacity of the memory in one factor that characterizes the size of a computer.

Small machines have only a few tens of millions of words.

Medium & large machines normally have many tens or hundreds of millions of words.

~~Dat~~

Instructions and data can be written into the memory or read out under the control of the processor.

* Memory in which any location can be reached in a short and fixed amou of time after specifying its address called RAM.

* The time required to access one word is called the memory access time. This time is fixed ~~to access one~~ independent of the location of the word being accessed.

* It typically ranges from a few Nanoseconds (ns) to 100 ns for Modern RAM units.

* memory hierarchy of three or four levels of Semiconductor RAM units with different speeds and sizes.

* The small, fast RAM units called caches.

* The largest and slowest unit is referred to as ~~as~~ the main memory.

Although primary storage is essential, it tends to be expensive.

Thus additional cheaper, Secondary cheaper is used when large amounts of data and many programs have to be stored.

Scanned by CamScanner

* A wide Selection of Secondary storage devices is available, including magnetic disks and tapes and optical disks (CD-ROMS)

ARITHMETIC AND LOGIC UNIT:-

* Many computer operations in ALU of the processor.

* Consider example:- Suppose two numbers located in the memory are to be added. They are brought into the processor,

* and the actual addition is carried out by the ALU.

* Any other ALU operation, for example Multiplication, division or comparison of numbers, by bringing the required operands into the processor.

* They are stored in high speed Storage elements called registers.

* Each register can store one word of data.

* The Control and the arithmetic and logic units are many times faster than other devices connected to a computer system. This enables a single processor to control as devices such as Keyboards, displays, magnetic and optical disks sensors and mechanical controllers

OUTPUT UNIT:-

* The output unit is the counterpart of the input unit.

* Its function is to send processed results to the outside world.

* The most familiar device is Printer.

* Printers employ mechanical impact head, ink jet streams, or photocopying techniques, as in laser printers, to perform the printing.

* It is possible to produce printers capable of printing as many as 10,000 lines per minute.

* Mechanical device but it still very slow compared to the electronic speed ~~[illegible]~~ of a processor unit.

* Some units such as graphic displays provides both an output function and an input function.

CONTROL UNIT:-

The memory, arithmetic and logic, and input and output units store and process information and perform input and output operations. The operations of these units must be coordinated in some way. This is the task of the control unit.

* I/o transfers, consisting of input and output operations are controlled by the instructions of I/o Programs. The devices involved and the information to be transferred.

* The actual timing signals the transfers are generated by the control circuits.

* Data transfers between processor and the memory are also controlled by the control unit through timing signals.

→ A large set of control lines (wires) carries the signals used for timing and synchronization of events in all units.

The operation of a computer.

* The computer accepts information in the form of programs and data through an input unit and stores it in the memory.

* Information stored in the memory is fetched, under program control, into an arithmetic and logic unit, where it is processed.

* Processed information leaves the computer through an output unit.

* All activities inside the machine are directed by the control unit.

Basic Operational Concepts

* The activity in a computer is gove
by instructions.

* To Perform a given task, an appropriate Program consists of a list of instructions is stored in the memory.

* Instructions are brought from the memory into the Processor, which executes the Specified operations

* Data to be used as operands are also stored in the memory.

A typical Instruction may be

Add LOCA, R0

This Instruction adds the operand at memory location LOCA to the Operand in a register in the Processor, R0 and Places the sum into Register R0.

The original contents of Location LOCA are preserved, whereas those of R0 are over written.

This instruction requires the Performance of several steps.

First the Instruction is fetched from the memory into the Processor.

* Next the operand at LOCA is fetched and added to the contents of R0. Finally, the resulting sum is stored in Register R0.

* The Preceding Add Instruction combines a memory access operation with an ALU operation. In many Modern computers, these two types of operations are performed by Separate instructions for performance reasons.

* The effect of the above instruction can be realized by two Instruction queue.

```
Load  LOCA, R1
Add   R1, R0
```

* The first of these Instructions transfers the contents of memory location LOCA into processor Register R1.

* The second Instruction adds the content of Register R1 and R0 and places the sum into R0, This destroys previous contents of Reg R1, R0 whereas original contents of memory locA are preserved.

* Transfers between the memory and the processor are started by sending the address of the memory location to be accessed to the memory unit and issuing the appropriate control signals.

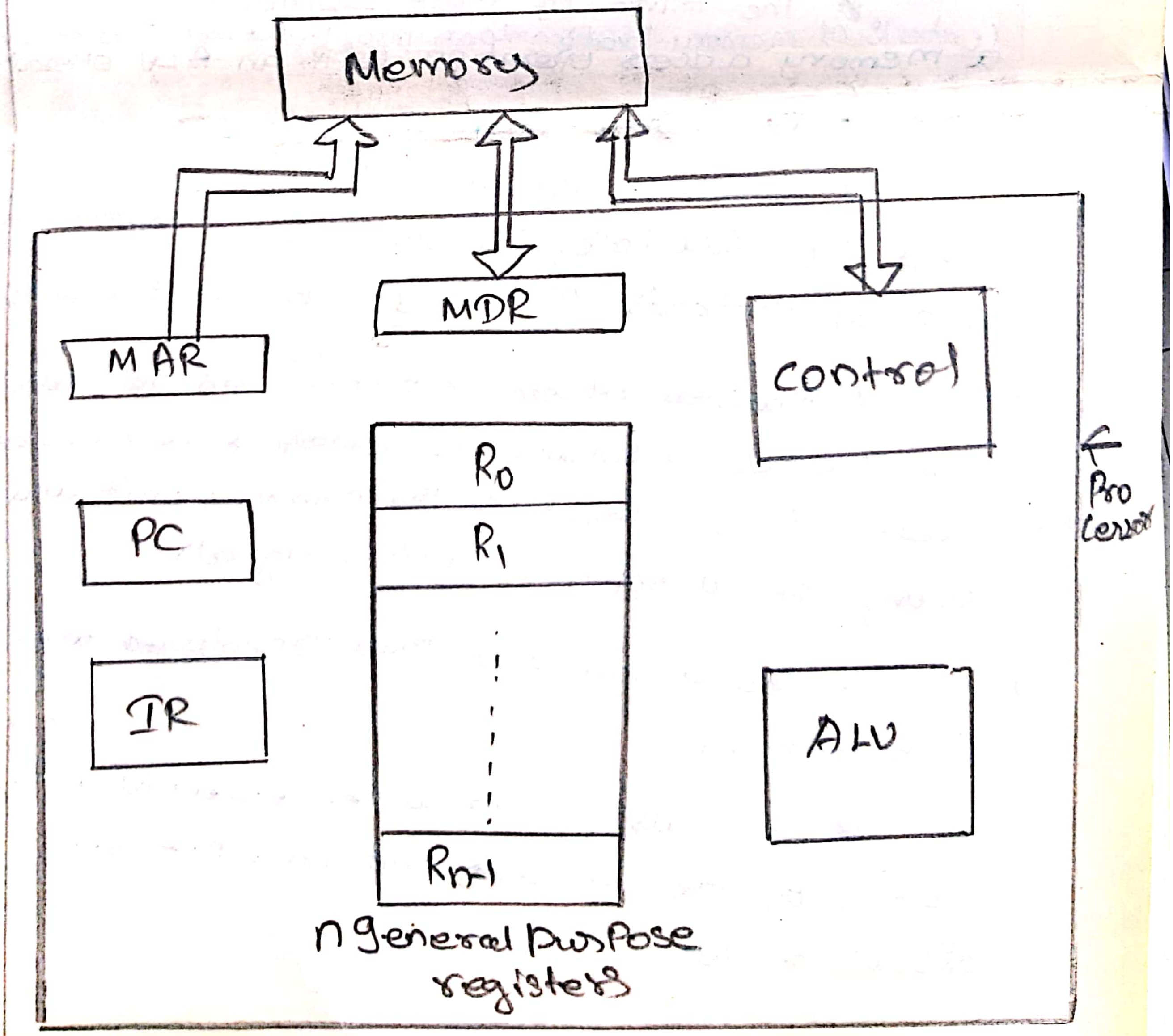
* The datas are then transferred to or from the memory.

* It also shows a few essential operational details of the processor that have not been discussed yet.

* In addition to the ALU and the Control circuitry, the Processor contains a number of registers used for several different purposes.

* The Instruction register (IR) holds the Instructions that is currently being executed.

* Its output is available to the control circuits which generates the timing signals that control the various processing elements involved in executing the Instruction.



* The Program Counter (PC) is another specialized register. It keeps track of the execution of a program.

* It contains the memory address of the next instruction to be fetched and executed.

* During the execution of an instruction, the contents of the PC are updated to correspond to the address of the next instruction to be executed.

* The PC points to the next instruction that is to be fetched from the memory.

* n- general purpose registers R_0 through R_{n-1}

* Finally two registers facilitate the communication with the memory. These are the memory address register (MAR) and the memory data register (MDR).

* the MAR holds the address of the location to be accessed. The MDR consists the data to be written into or read out of the address location.

Scanned by CamScanner

Bus S

* Let us consider some typical operati

* Execution of the program starts when th PC is set to point to the first Instruction of the Program.

* The contents of the PC are transferred to the MAR and a Read control signal is sent to the memory.

* After time required to access the memory elapses, the addressed word is read out of the memory and loaded into the MDR. Next the contents of the MDR are transferred to the IR.

* At this point, the instruction is ready to be decoded and executed.

* If the Instruction involves an operation to be performed by the ALU, it is necessary to obtain the required operands.

* If an operand resides in the memory (it could also be in a general purpose register in the Processor), it has to be fetched by sending its address to the MAR and initiating a Read cycle.

* When the operand has been read from the memory into the MDR, it is transferred from the MDR to the ALU.

BUS STRUCTURES

* we have discussed the function of Individual Parts of a computer.

* To form an operating system, these Parts must be connected in some organized way.

* There are many ways of doing this. we consider simplest and most common of these here.

* To acheive a reasonable speed of operation, a computer must be organized so that all its units can handle one full word of data at a given time.

* when a word of data is transferred between units, all its bits are transferred in parallel, that is.

* The bits are transferred simultaneously over many wires, or lines, one bit per line.

* A group of lines that serves as a connecting path of several devices is called a ~~bu~~ bus.

* In addition to the lines that carry the data, the bus must have lines for address and control purposes.

* The simplest way to interconnect functional units is to use a single bus.

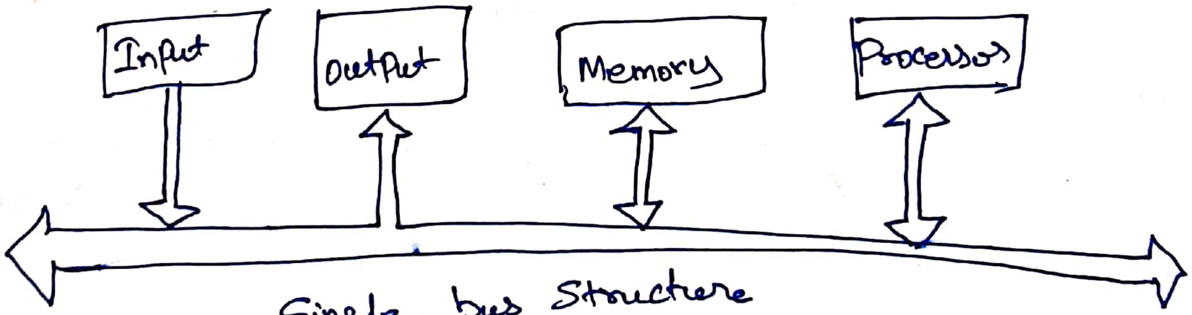
* All units are connected to this

* Because the bus can be use for only one transfer at a time, only two units can entired actively use the bus at any given time.

* Bus control lines are used to arbitrate multiple request for use of the bus.

* The main virtue of the single bus structure is its low cost and its flexibility for attaching Peripheral devices.

* Systems that contain multiple buses acheive more concurrency in operations by allowing two or more transfers to be carried out at the same time.



Single bus Structure

* This leads to better Performance but at an increased cost.

* The devices connected to a bus vary widely in their Speed of operations. Some electromechanical devices such as keyboards and Printers, are relatively slow.

* Others like magnetic or optical disks are considerablly faster.

* Memory and processor units operates at electronic speeds, making them the fastes parts of a computer.

* Because all these devices must communicate with each other over a bus, an efficient transfer mechanism that is not constrained by the slow devices that can be used to smooth out the differences in timing among processors, memories, and external devices is necessary.

* A common approach is to include buffer registers with the devices to hold the information during transfers.

* To illustrate this technique, consider the transfer of an encoded character from a processor to a character printer.

* The processor sends the character over the bus to the printer buffer.

* since the buffer is an electronic register, this transfer requires relatively little time.

* once the buffer is loaded, the printer can start printing without further intervention by the processor.

* The bus and the Processor are no longer needed and can be released for other activity.

* The Printer continues Printing the character in its buffer and is not available for further transfers until the Process is completed

* Thus, buffer registers Smooth out timing timing differences among Processors Memories and I/o devices, They prevent a high speed Processor from being locked to a slow I/o device during a Sequence of data transfers.

* this allows the Processor to switch rapidly from one device to another, interweaving its Processing activity with data transfers involving several I/o devices.

Register Transfer language

* A digital System is an interconnection of digital hardware modules that accomplish a specific information - Processing task.

* Digital Systems Vary in size and complexity from a few Integrated circuits to a complex of interconnected and Interfacing digital Computers.

* Digital System design invariably uses a modular approach.

* The modules are constructed from such digital components as registers, decoders, arithmetic elements, and control logic.

* The various modules are Interconnected with common data and control Paths to form a digital Computer System.

MICRO operation :-

* Digital modules are best defined by the registers they contain and the operations that are Performed on the data Stored in them.

* The operations executed on data Stored in registers are called microoperations.

* A micro operations is an elementary Operation Performed on the information stored in one or more registers.

* The result of the operation may replace the Previous binary information of a register or may be transferred to another register

Examples

Shift, count, clear. and load.

* The registers that implement micro operations for examples, a counter with Parallel load is capable of Per-forming the microoperations increment and load.

* A bidirectional Shift register is capable of Performing the Shift right and shift left micro operations

→ The Internal hardware Organization of a digital computer is best defined by specifying.

1. The set of registers it contains and their function.
2. The sequence of microoperations Performed on the binary information Stored in the registers.
3. The control that initiates the sequence of microoperation

* It is Possible to Specify the sequence of microoperations in a computer by explaining every operations in words, but this Procedure usually involves a lengthy descriptive explanation.

* It is more convenient to adopt a ~~similar~~ Suitable symbology to describe the sequence of transfers between registers and the various arithmetic and logic microoperations associated with the transfers.

* The use of symbols instead of a narrative explanation provides an organized and concise manner for listing the microoperations sequence in registers and the control functions that initiate them.

The symbolic notation used to describe the micro operation transfer among registers is called a register transfer language.

The term "register transfer" implies the availability of hardware logic circuits that can perform a stated microoperation and transfer ~~as~~ to the same or the result of the operation another register.

The word "language" is borrowed from programmers, who apply this term to programming languages.

A programing languages is a procedure for writing symbols to specify a given computational process.

Similarly a natural ~~or~~ language such as English is a system for expressing in symbolic form the micro operation sequences among the registers of a digital module.

* It is a convenient tool for describing Internal organization of digital Computers in consis and Precise manner. It can also be use to facilitate the design Process of digital Systems.

The register transfer language adopted here is believe to be as Simple as Possible, So it Should not take very long to memorize.

We will Proceed to define Symbols for various types of microoperations, and at the same time, describe associated hardware that can implement the Stated microoperations.

To specify the register transfers, the microoperations, and the control functions that describe the Internal hardware organization of digital computers.

Symbology in use can easily be learned once this language has become familiar, for most of the differences between register transfer languages consists of variation in detail rather than in overall Purpose.

REGISTER TRANSFER

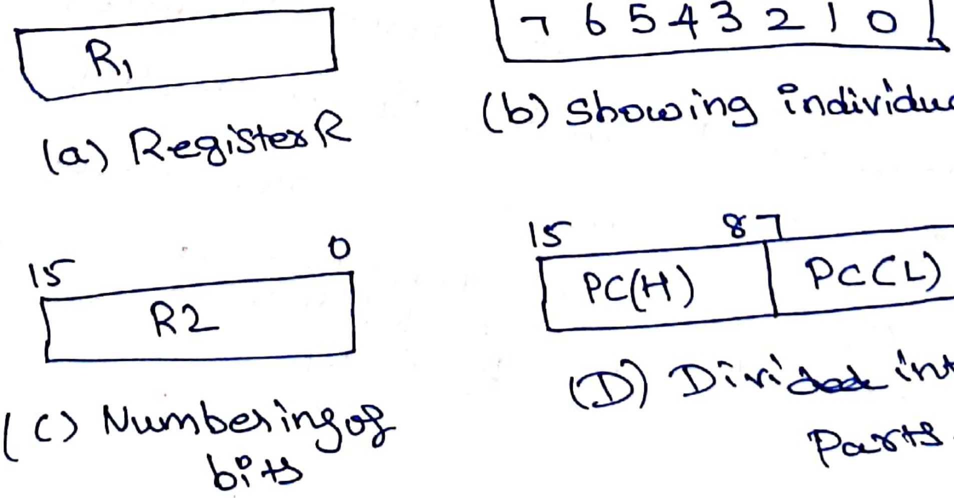
* Computer registers are designated by capital letters (sometimes followed by numerals) to denote the function of the register.

* For example, the register that holds an address for the memory unit is usually called a memory address register and is designated by the name MAR.

* other designation for registers are PC (for Program counter), IR (for instruction register), and R1 (for Processor register). The individual flip flops in an n-bit register are numbered in sequence from 0 through n-1,

* starting from 0 in the right most position and increasing the numbers toward the left.

Block Diagram of register



(a) Register R

(b) Showing individual bits

(c) Numbering of bits

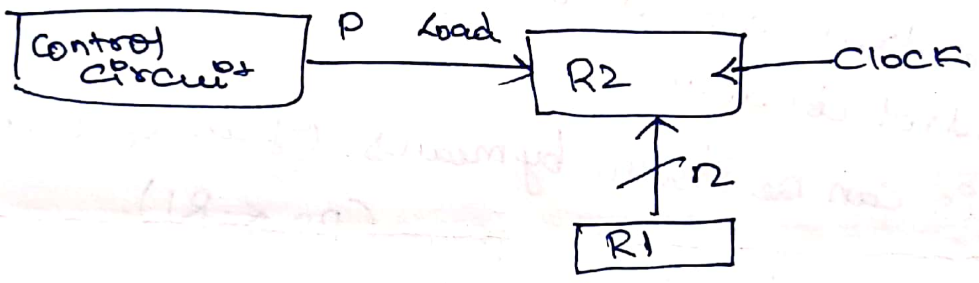
(D) Divided into two Parts.

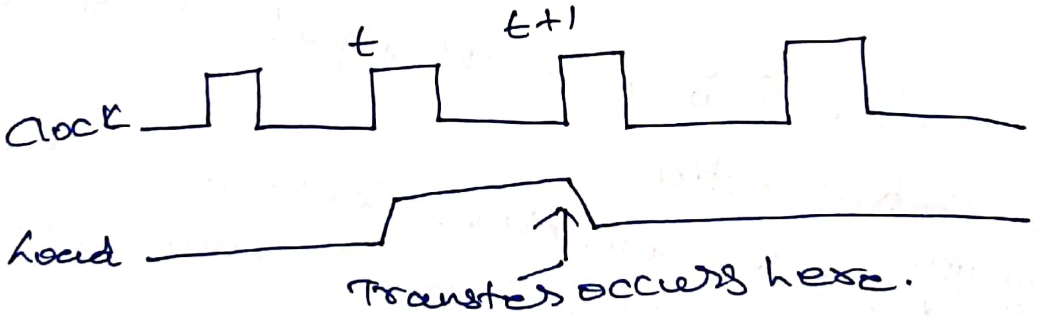
Scanned by CamScanner

* The Control Condition is terminated with a co
It symbolizes the requirement that the transfer
function operation be executed by the hardware
only if P=1

* Every statement written in a register
transfer notation implies a hardware construction
for implementing the transfer.

Transfer from R1 to R2 when P=1





Timing Diagram.

* The n outputs of register R1
are connected to the n inputs of register R

* The letter n will be used to indicate
any number of bits for the register.

* It will be replaced by an
actual number when the length of the Registe
is known.

* Register R2 has a load input that is activated by the control variable P. It is assumed that the control variable is synchronized with the same clock as the one applied to the register.

* In the timing diagram, P is activated in the control section by the rising edge of a clock pulse at time t.

* The next positive transition of the clock at time t+1 finds the load input active and the data inputs of R2 are then loaded into the register in parallel.

* P may go back to 0 at time t+1; otherwise, the transfer will occur with every clock pulse transition while P remains active.

* Note that the clock is not included as a variable in the register transfer statements.

* It is assumed that all transfers occur during a clock edge transition.

* Even though the control condition such as P becomes active just after time t, the actual transfer does not occur until the register is triggered by the next positive transition of the clock at time t+1.

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	MAR, R2
Parentheses()	Denotes a Part of a register	R2(0-7), R2(L)
Arrow ←	Denotes transfer of Information	$R_2 \leftarrow R_1$
Comma,	Seperates two microoperations	$R_2 \leftarrow R1$, $R1 \leftarrow R_2$

* Registers are denoted by capital letters, and numerals may follow the letters.

* Parentheses are used to denote a part of a register by specifying the range of bits or by giving a symbol name to a Portion of a register.

* The arrow denotes a transfer of information and the direction of transfer. A comma is used to separate two or more operations that are executed at the same time.

The Statement

T: R2 ← R1 , R1 ← R2

Denotes an operation that exchanges the contents of two registers during one common clock Pulse provided that T=1. this simultaneous operation is possible with registers that have edge triggered flip-flops.

Scanned by CamScanner

PERFORMANCE

* The most important measure of the Performance of a computer is how quickly it can execute Programs.

* The Speed with which a computer executes programs is affected by the design of its hardware and its machine language Instructions.

* Because programs are usually written in a high level language, Performance is also affected by the compiler that translates Programs into machine language.

* For best Performance, it is necessary to design the compiler, the machine instruction set, and the hardware in a coordinated way.

* How the operating system overlaps Processing, disk transfers and printing for several Programs to make the best Possible use of the resources available

* The total time ~~to~~ required to execute a ~~program~~ program is ~~to~~ $t_s - t_0$.

* This elapsed time is a measure of the performance of the entire computer system.

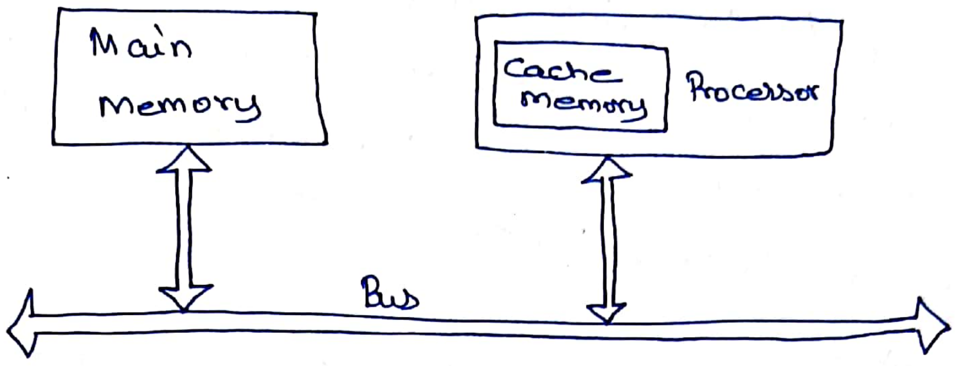
* It is affected by the speed of the processor, the disk and the printer.

* To discuss the performance of the processor we should consider only the period during which the processor is active.

* We will refer to sum of these periods as the processor time needed to execute the program.

* We will identify some of the key parameters that affect the processor time in which the relevant issues are discussed.

* Just the elapsed time for the execution of a program depends on all units in a computer system, the processor time depends on the hardware involved in the execution of individual machine instructions.



The Processor cache.

At the start of execution, all Proglam Instructions and required data are stored in the main memory.

As execution proceeds, instructions are fetched one by one over the bus into the processor, and a copy is placed in the cache.

When the execution of the instruction calls for data located in the main memory, the data are fetched and a copy is placed in the cache.

* If the same instruction or data item is needed a second time, it is read directly from the cache.

* The processor and a relatively small cache memory can be fabricated on a single Integrated circuit chip.

* The Internal speed of performing the basic steps of instruction processing of such chips is very high and is considerably faster than the speed at which instruction and data can be fetched from the main memory.

* A program will be executed faster if the movement of instruction data between the main memory and the processor is minimized, which is acheived by using the cache.

forexample
Suppose a numbers of instructions are executed repeatedly over a short period of time, as happens in a program loop.
If these instructions are available in the cache. they can be fetched quickly during the period of repeated use.

Processorclock

1. Processor circuits are controlled by a timing signal called a clock.
2. The clock defines regular time intervals, called clock cycles.
3. To execute a machine instructions, the processor divides the action to be performed into a sequence of basic steps, such that each step can be completed in one clock cycle.
4. The length P of one clock cycle is an important parameter that affects the processor performance.
5. Its Inverse is the clock rate, $R = 1/P$, which is measured in cycles per second.
6. Processors used in today's personal computers and workstations have clock rates that range from a few hundred million to a over a billion cycles per second.

* In electrical engineering the term "cycles per second" in hertz (Hz), the term

million → denoted by the Prefix Mega(m)

Billion → prefix Giga (G)

Hence 500 million cycles per second is 500 (MHz), 1250 million cycles → 1.25 GHz

2 or 0.8 Clock Periods is nanoseconds.

BASIC PERFORMANCE Equation:-
Where each basic step is completed in one clockcycle. If the Clockrate is R cycles per second the program execution time is given by

$$T = \frac{N \times S}{R}$$

T → Parameter for an application program.
Individual values of the Parameters N, S or R.
To acheive high Performance the computer designes must seek ways to reduce the value of T means reducing N and S, and Increasing R

N is reduced → Source Program is compiled into fewer machine Instructions.

S is reduced → Steps to perform of the execution of instructions is overlapped.

* using high frequency clock ↑ the value of R, the means basic time Required.

PIPELINING AND SUPER SCALAR OPERATION

Instructions are executed one after another, the value of S is the total number of basic steps, or clock cycles, required to execute an instruction.

A substantial Improvement in performance can be acheived by overlapping the successive instructions, using a technique called Pipelining.

Add R1, R2, R3

which adds the contents of registers R1 and R2, and places the sum into R3.

The contents of R1 and R_2 are first transferred into the inputs of the ALU.

After add operation is performed the sum is transferred to R3.

Then if that instruction also uses the ALU, its operands can be transferred to the ALU inputs at the same time.

The result of the Add instruction is being transferred to R3.

The execution proceeds at the rate of one instruction completed in each clock cycle.

Individual Instructions still require several clock cycles to complete.

* Multiple instruction pipelines are implemented in the processor. The creating parallel paths through which different instruction can be executed in parallel. Several instruction in every clock cycle. This mode of operation is called superscalar execution

CLOCK RATE

* first improving IC technology makes logic circuits faster, which reduces the time to complete a basic step.

* The clock period, P to be reduced and clock rate R to be increased.

* second reducing the amount of processing done in one basic step also make it possible to reduce the clock period, P. If the actions have to be performed by an instruction remains the same, the numbers of basic steps needed may increase.

INSTRUCTION SET :- CISC AND RISC

→ simple Instructions requires a small number of basic steps to execute.

→ complex Instructions involve a large steps.

→ The relative merits of processors with simple Instructions and processor with more complex instruc The former are called (RISC). and the latter are referred to (CISC)

COMPILER :-

* A compiler translates a high level language program into a sequence of machine instruction.

* To reduce N, we need to suitable machine instruction set and compile makes good use of it.

* An optimizing compiler takes advantage of various features to the target processor to reduce the product N×S.

* The compiler may rearrange program instructions to acheive better performance.

* The ultimate objective is to reduce the total number of clock cycles needed to perform a required programming task.

PERFORMANCE MEASUREMENT:-

* Computer designers use performance estimate to evaluate the effectiveness of new featu

* A non profit organization called System Performance evaluation corporation (SPEC) selects and publishes representative application programs for different application domains. together with best results for many commercially available computers.

* for general purpose computers a suite of benchmark programs was selected in 1989. It was modified somewhat and published in 1995 and again in 2000.

$$\text{SPEC rating} = \frac{\text{Running time on the reference computer}}{\text{Running time on the computer under test}}$$

Bus and Memory transfers

* A digital computer has many registers, and paths must be Provided to transfer information from one register to another.

* The numbers of wires will be excessive if seperate lines are used between each register and all other registers in the system.

* A more efficient scheme for transfering Information between registers in a multiple register configuration is a common Bus System.

* A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at ~~a~~ time.

* Control signals which determine which register is selected by the bus during each particular register transfer.

* one way of constructing a common bus system is with multiplexers. The multiplexers select the source register whose binary informan is then placed on the bus.

* Each registes has four bits numberred
0 through 3. The bus consists of four 4X1
multiplexers each having four data input,
0 through 3, and two selection inputs, S, and S0.

* In order not to complicate the diagram
with 16 lines crossing each other, we use labels
to show the connections from the outputs of
the registers to the inputs of the multiplexers.

* For example, output 1 of register A is
connected to input 0 of MUX 1 because
this input is labeled A1.

* The significan position in each
register are connected to the data
input of one multiplexer to form one line of
the bus

Thus MUX 0 multiplexes the four 0 bits of the registers, MUX 1 ~~multi~~ multiplexes the four 1 bits of the registers, and similarly for the other two bits.

Function Table for Bus

S_1	S_0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

In general, a bus system will multiplex K registers of n bits each to produce an n-line common bus.

when the bus is includes in the statement, the register transfer is symolized

Bus ← C · R1 ← Bus

The content of ~~a~~ register C is placed on the bus, and the content of the bus is loaded into Register R1 by activating its load control input.

If the bus is known to exist in the system, it may convenient just to show the direct transfer.

R1 ← C

from this statement the designer knows which control signals must be activated to produce the transfer through the bus.

Three state Bus buffers.

* The Bus system can be constructed with three state gates instead of multiplexers

* A three state gate is a digital circuit that exhibits three states

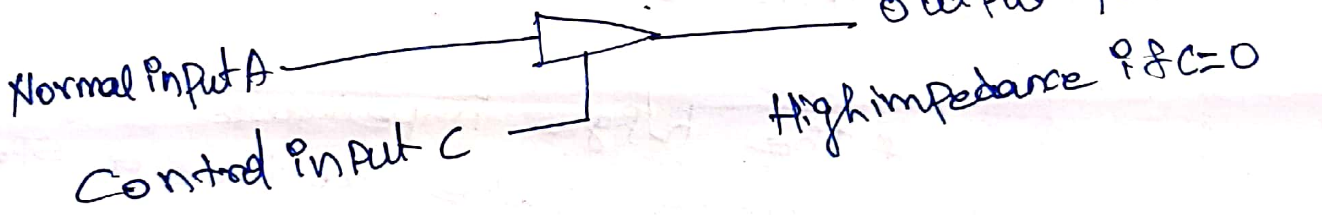
* Two of the states are signals equivalent to logic 1 and 0 as in a conventional gate.

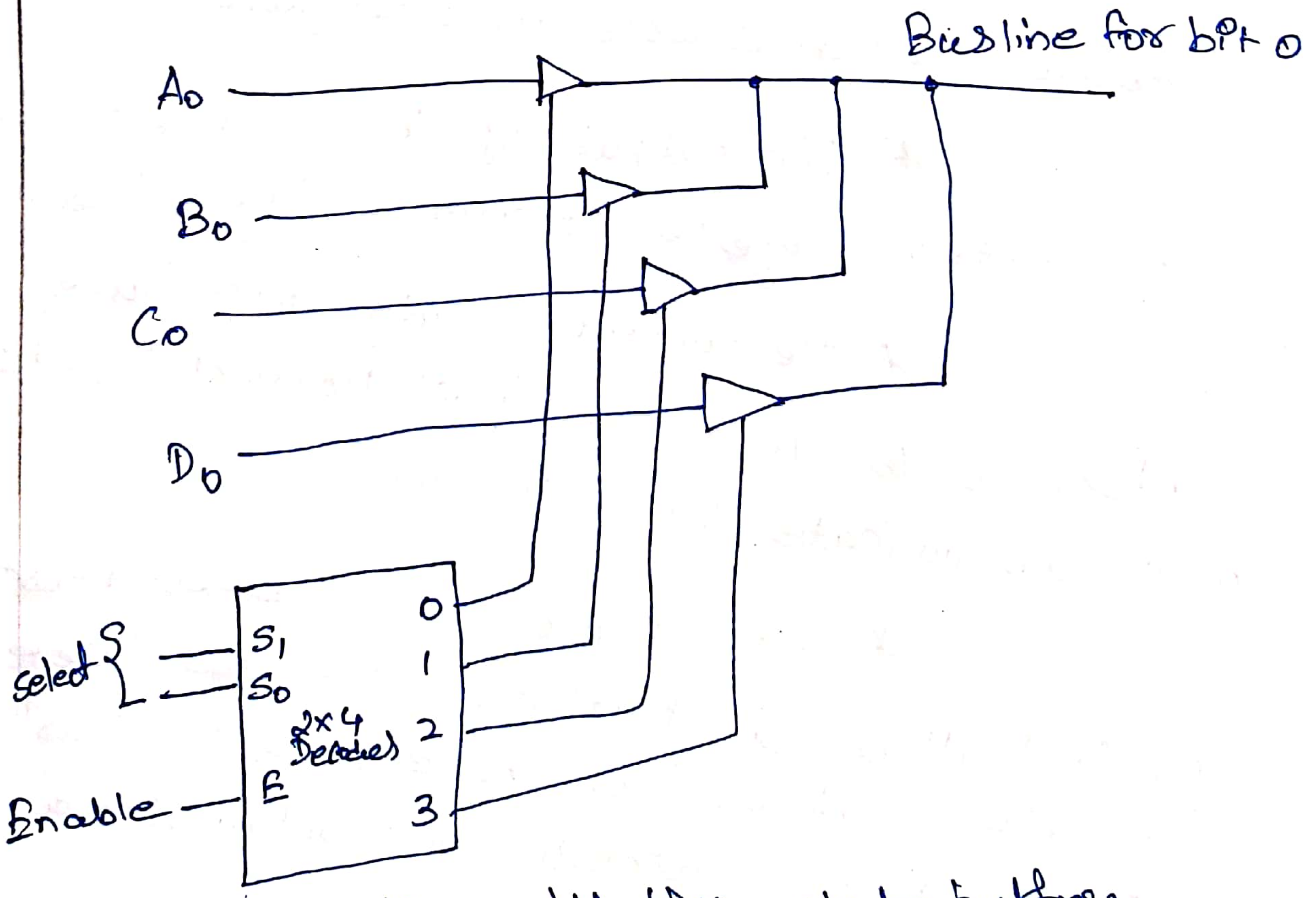
③

* The third state is a high impedance state. The high impedance state behaves like an open circuit, which means that the output is disconnected and does not have logic significance.

* Three state gates may perform any conventional logic AND or NAND. However the most commonly used in the design of a bus system is the buffer gate.

Graphic symbols for three state buffer





Bus line with three state buffers.

* It is distinguished from a normal buffer by having both a normal input and a control input.

* The Control input determines the output state. When the control input ~~is equal~~ ~~[illegible]~~ is equal to 1, the output is enabled and the gate behaves like any conventional buffer, with the output ~~equal~~ equal to the normal i/p

* when the control input is 0, the output is disabled and the gate goes to a high impedance state, regardless of the value ~~of~~ in the normal input.

* The high impedance state of a three state gate provides a special feature not available in other gates.

* The output of four buffers are connected together to form a single bus line.

* the control inputs to the buffer determine which of the four normal inputs will communicate with the bus line.

* The connected buffers must be controlled so that only one ~~the~~ three state buffer has access to the busline while all other buffers are maintained in a high impedance state

* when the enable input of the decoder is 0, all of its four outputs are 0, and the bus line is in a high impedance state because all four buffers are disabled.

* when the enable input is active, one of the three state buffers will be active depending on the binary value in the select inputs of the decoder.

* Each group of four buffers receives one significant bit from the four registers.

* Each common output produces one of the lines for the common bus for a total of n lines.

Memory transfer :-

The transfer of information from a memory word to the outside environment is called a read operation.

The transfer of new information is to be stored into the memory is called a write operation.

The memory word will be be symbolized by M. The particular memory word among the many available is selected by the memory address during the transfer.

* consider a memory unit that receives the address from a register, called the address register symbolized by AR.

DR → Data register.

The read operation can be stated as follows:

Read: DR ← M[AR]

This ~~tea~~ causes a transfer of information into DR from the memory word M selected by the address in AR.

Assume that the input data are in register R1 and the address is in AR. The write operation can be stated as follows.

Write: M[AR] ← R1

This causes a transfer of Information from R1 into the memory word M selected by the address in AR.

ARITHMETIC MICROOPERATIONS

A microoperation is an elementary operation performed with the data stored in registers. The microoperations most often encountered in digital computer are classified into four categories.

1. Register transfer micropperations transfer binary information from one register to another.
2. Arithmetic microoperations perform arithmetic operation on numeric data stored in registers.
3. Logic microoperations perform bit manipulation operations on non numeric data stored in registers.
4. Shift micro operations perform shift operations on data stored in registers.

* This type of microoperation does not change the information content when the binary information moves from the source register to the destination register.

* The other three types of microoperations change the information cotent during the transfer.

* In this section we introduce a set of arithmetic microoperations

* The basic arithmetic microoperations are addition, subtraction, increment, decrement and shift.

* Arithmetic shifts are explained later in conjunction with the shift microoperations.

...microoperations are symbolized

Add micro operation.

The arithmetic microoperation defined by the statement specifies an add microoperations.

$$R3 \leftarrow R1 + R2$$

* It states that the contents of register R1 are added to the contents of register R2 and the sum transferred to register R3.

* To implement this statement with hardware we need three registers and the digital component that performs the addition operation.

Subtract Microoperation.

Subtract is most often implemented through complementation and addition. Instead of using the minus operator, we can specify the subtraction by the following statement

$$R_3 \leftarrow R1 + \overline{R2} + 1$$

$\overline{R2}$ is the symbol for the one's complement of R2. Adding 1 to the 1's complement too produces the 2's complement.

Adding the contents of R1 to the 2's complement of R2 is equivalent to R1 - R2.

Arithmetic microoperations

Symbolic designation	Description.
$R3 \leftarrow R1 + R2$	Contents of R1 plus R2 transferred to R3
$R3 \leftarrow R1 - R2$	Contents of R1 minus R2 transferred to R3
$R2 \leftarrow \overline{R2}$	Complement the contents of R2 (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement the contents of R2 (Negate)
$R3 \leftarrow R1 + \overline{R2} + 1$	R1 plus the 2's complement of R2 (Subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of R1 by one
$R1 \leftarrow R1 - 1$	Decrement " " R1 " "

* The ~~as~~ inc & dec microoperations are symbolized by Plus one and Minus - one operations, respectively. These microoperations are implemented with a combinational circuit or with a binary up-down counter.

* The arithmetic operations Multiply and Divide are not listed. These two operations are valid arithmetic operations but are not included in the basic set of microoperations.

* The only place where these operations can be considered as microoperations is in a digital system, where they are implemented by means of a combinational circuit.

* In such a case, the signals that perform the signals that perform these these operations the signals that perform these operation through gates, and the result of the operation can be transferred into a destination register by a clock pulse as soon as output signal propagates through combinational circuit.

* In most computers, the multiplication operation is implemented with a sequence of add and shift microoperations.

* Division is implemented with a sequence of ~~sub~~ subtract and shift microoperations.

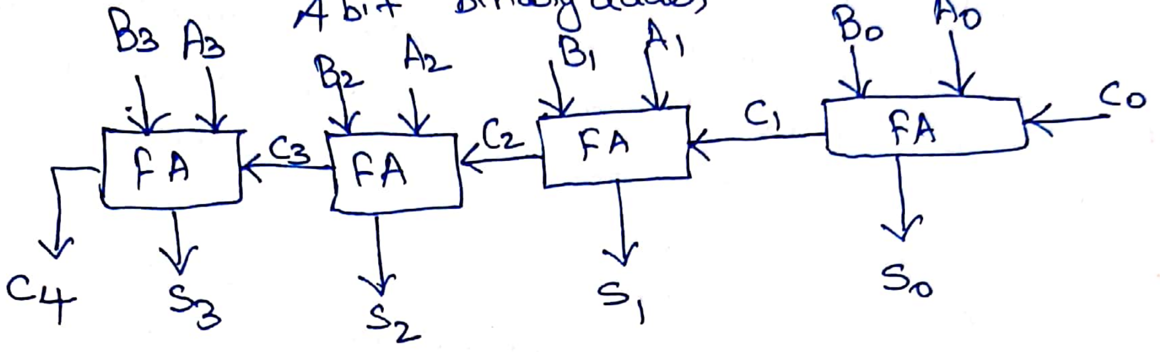
* To specify the hardware in such a case requires list of statement use basic microoperations. add, subtract and shift.

Binary adder

To Implement the add microoperation with hardware, we need the registers that hold the data and the digital component that Performs the arithmetic addition.

* The digital circuit that forms the arithmetic sum of two bits and a previous carry is called a full adder.

4 bit binary adder



* The digital circuit that generates arithmetic sum of two binary numbers of any length is called a binary adder.

* The binary adder adder is constructe with fulladders circuits connected in cascade, with the output carry from one full adder connected to the input carry of the next full adder.

* The augend bits of A and addend bits of B are designated by subscript numbers from right to left with subscript 0 denoting the low order bit.

* The carries are connected in a chain through the full adders.

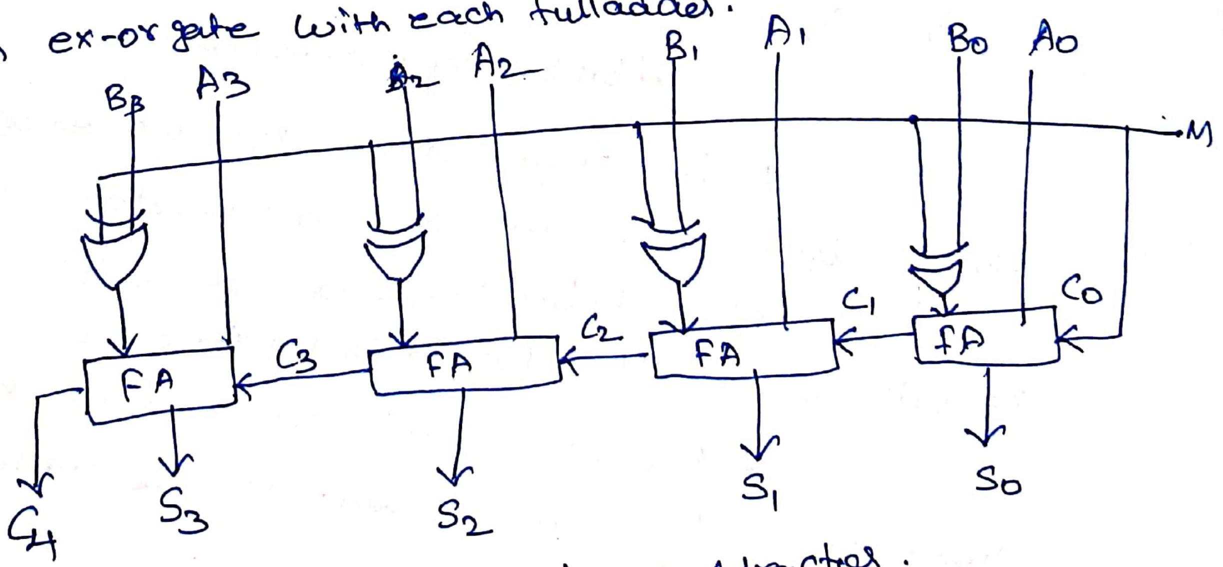
* The input carry to the binary adder is C_0 and the output carry is C_4.

* The S outputs of full adders generate the required sum bits.

* An n bit binary adder requires n full adders.
* The output carry from each full adder is connected to the input carry to the next higher order full adder.
* The n data bits for A inputs come frome one register (such as R1), and the n data inputs for the B inputs come from another register (such as R2).
* The sum can be transferred to a third register or to one of the source register replacing its previous content. (R1 or R2)

Binary adder-subtractor.

* Remember that the subtraction A-B can be done by taking the 2's complement of B and adding it to A.
* The addition and subtraction operations can be combined into one common circuit by including an ex-or gate with each fulladder.



4 bit adder subtractor.

* when $M=0$ The circuit is an adder and when $M=1$ the circuit becomes Subtractor.

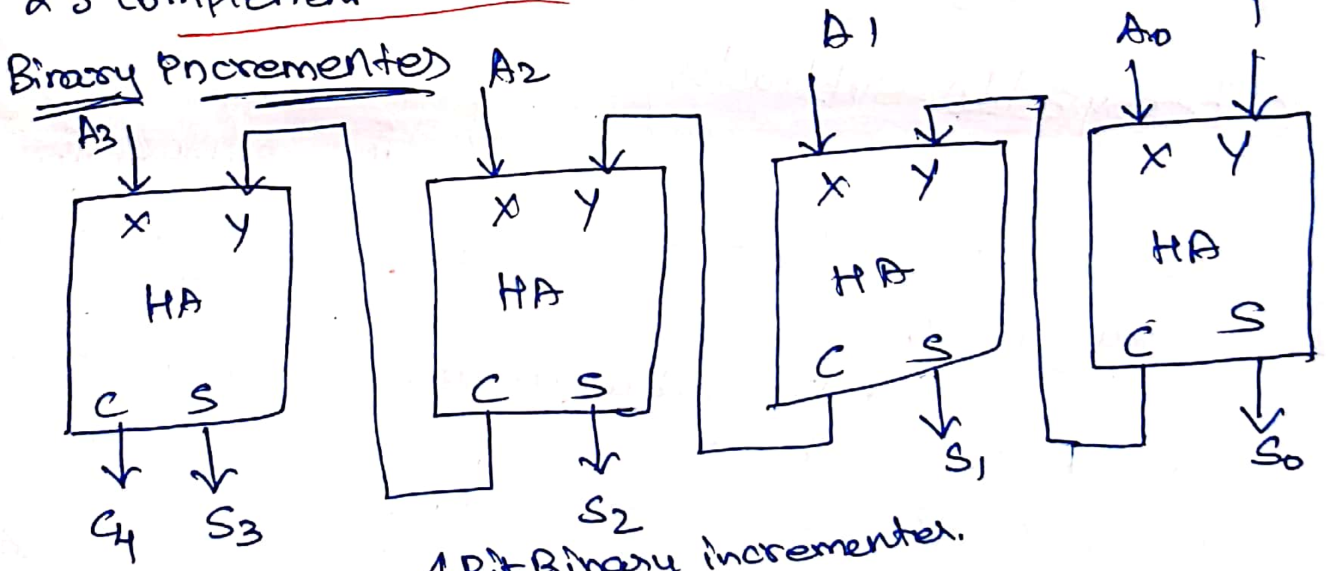
* Each Ex-or Gate receives input M and one of the inputs of B

* when $M=0$, we have $B \oplus 0 = B$. The full adder receive the value of B the input carry is 0, and the circuit Performs A plus B.

* when $M = \phi$ we have $B \oplus 1 = B'$ and $C_0=1$ The B inputs are all complemented and a added through the input Carry.

* The circuit Performs the operation A plus B 2's complement of B.

Binary incrementer



4 Bit Binary incrementer.

* The Incrementer micro operation adds one to a number in a register. for example if a 4-bit register has a binary value 0110 it will go to 0111 after it is incremented.

* Every time count enable is active, the clock pulse transition increments the content of the register by one.

* one of the inputs of the least significant half adder (HA) is connected to logic-1 and the other input is connected to LSB of the number to be incremented.

* The output carry from one half adder is is connected to one of the inputs of the next higher-order half adder.

* The output carry C_4 will be 1 only after incremented binary 1111

Arithmetic circuit.

The basic component of an arithmetic circuit is parallel adder. By controlling the data inputs to the adder, it possiple of differ operation

* The other two data inputs are connected to logic-0 and logic-1.

Logic-0 is fixed voltage value (0 volts for TTL Integrated circuits

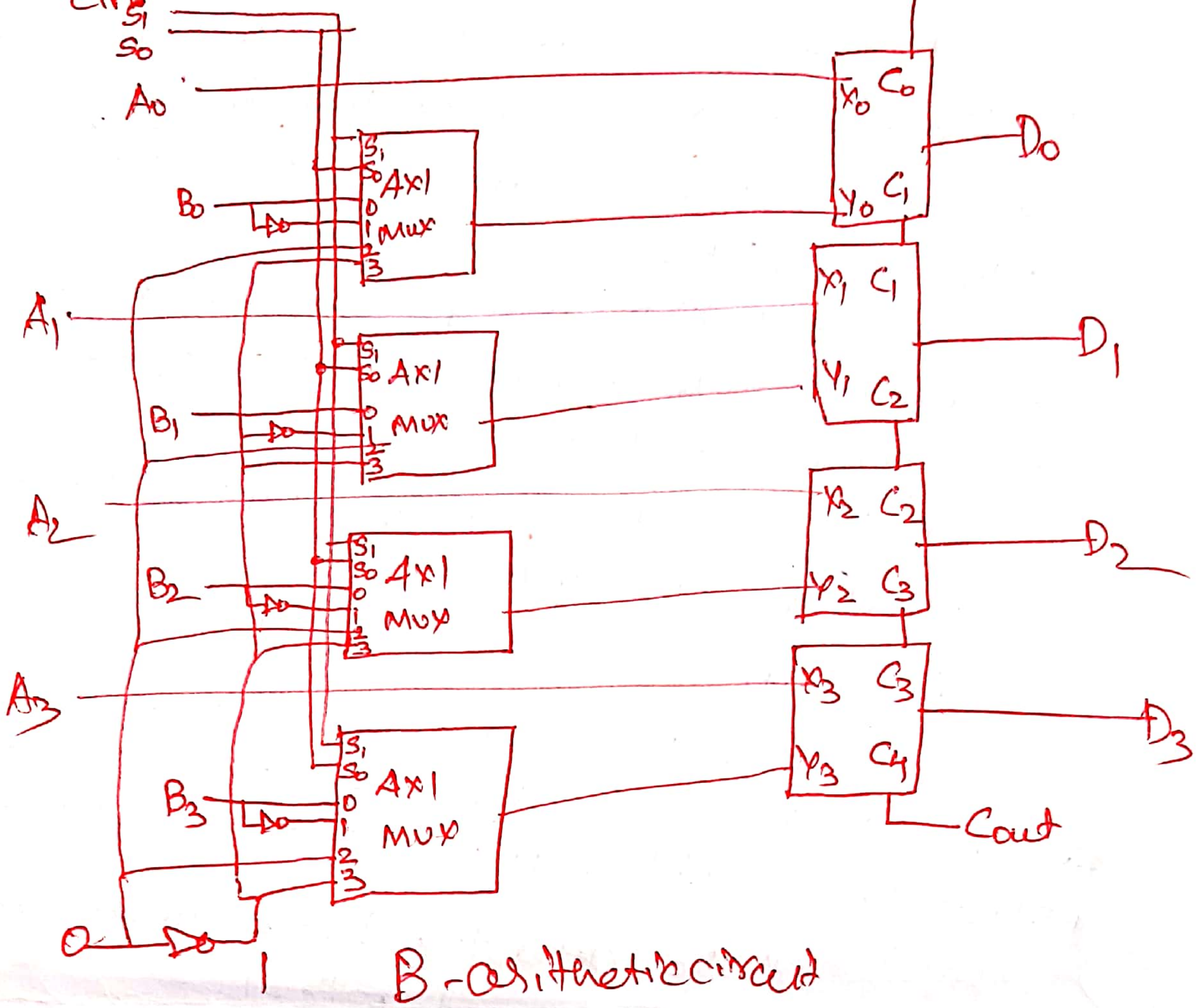
Logic-1 signal can be generated through an Inverter whose ~~logic~~ input is 0.

The output of the binary adder is calculated from the following arithmetic sum.

$$D = A + Y + C_{in}$$

Arithmetic circuit function table

Select			Input	Output	Microoperations
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$	Subtract ~~with Sd~~ with Borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	Subtract ~~with borrow~~
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A



B-arithetic circuit

Addition

# when $S_1S_0 = 00$ → The value B is aplied Y inputs of the adds. If Cin = 0 the

O/P $D = A+B$

If $Cin = 1 = A+B+1$ Both cases Perform the add microopeation by

Subtraction

when $S_1S_0 = 01$ → The complement of B is applied Y inputs to the adder.

If $Cin = 1$ $D = A+\bar{B}$ ~~[illegible]~~ $+1$. This provides A plus the 2's Complement of B provide is equivalent $A - B$

$Cin = 0$ then $D = A + \bar{B}$ is equivalent to subtract with borrow is $A - B - 1$

Increment

When $S_1 S_0 = 10$ the i/p B are neglected all 0's are inserted The O/P becomes $D = A+0+Cin$.

$D = A$

when $C = 0$

& $D = A+1$ when $Cin = 1$ is incremented by one

Dec $S_1 S_0 = 11$ $D = A-1$ when $Cin = 0$ / $Cin = 1$

$D = A-1+1 = A$

SHIFT MICRO OPERATIONS

* Shift Microoperations are used for serial transfer of data.

* They are also used in conjunction with arithmetic, logic and other data processing operations.

* The Contents of a register Can be Shifted to the left or the right

* At the Same time that the bits are shifted, The first flipflop receives its binary information from the serial input.

* During a shift left operation the serial input transfers a bit into the ~~left~~ right most Position.

* During shif right operation the serial Input transfers a bit into the left most Position.

* The Information transferred through the Serial input Determines the type of shift ~~transfers~~ a bit into ~~the leftmost position.~~

~~* The Information~~

* There are three types of shifts: logical Circular, and arithmetic

Logical Shift :-

A logical shift is one that transfers 0 through the Serial input. We will adopt the symbols shl and shr for logical shift left and shift right microoperations

for ex :- $R1 \leftarrow Shl\ R1$

$R2 \leftarrow Shr\ R2$

are two microoperations that Specify a 1-bit shift to the left of the Content of register R1 and a 1-bit Shift to the right of the Content of Register R2

* The register symbol must be the same on both sides of the arrow.

* The bit transferred to the end position through the serial input is assumed tobe 0 during a logical shift.

CIRCULAR SHIFT:

* The circular shift (also known as a rotate operation) circulates the bits of the register around the two ends without loss of information.

* This is accomplished by connecting the serial output of the shift register to its serial input.

* we will assume the symbols cil and cir for the circular shift left and right respectively.

Shift microoperations

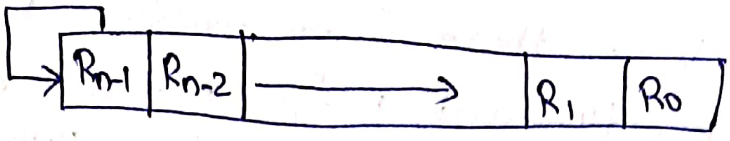
Symbolic designation	Description
R ← shl R	Shift left register R
R ← shr R	Shift right register R
R ← cil R	Circular shift left register R
R ← cir R	circular shift right register R
R ← ashl R	Arithmetic shift left R
R ← ashr R	Arithmetic shift right R

ARITHMETIC SHIFT:-

* An arithmetic shift is a microoperation that shifts a signed binary number to the left or right.

* An arithmetic shift left multiplies a signed binary number by 2.

* An arithmetic shift right divides the number by 2. Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same.



Signbit

Arithmetic shift right

* When it is multiplied or divided by 2. The leftmost bit in a register holds the sign bit, and the remaining bits hold the numbers.

* The sign bit is 0 is positive and 1 for negative
* Negative Numbers are in 2's complement form.
* Bit R_{n-1} is the left most position holds the sign bit.
* R_{n-2} is the MSB of the number and R_0 is the LSB
* The arithmetic shift right leaves the sign bit unchanged and shifts the number (including the sign bit) to the right.
* Thus R_{n-1} remains the same, R_{n-2} receives the bit from R_{n-1} (is lost and replaced by the bit from R_{n-2}) and so on the other bits in the register. R_0 is lost.

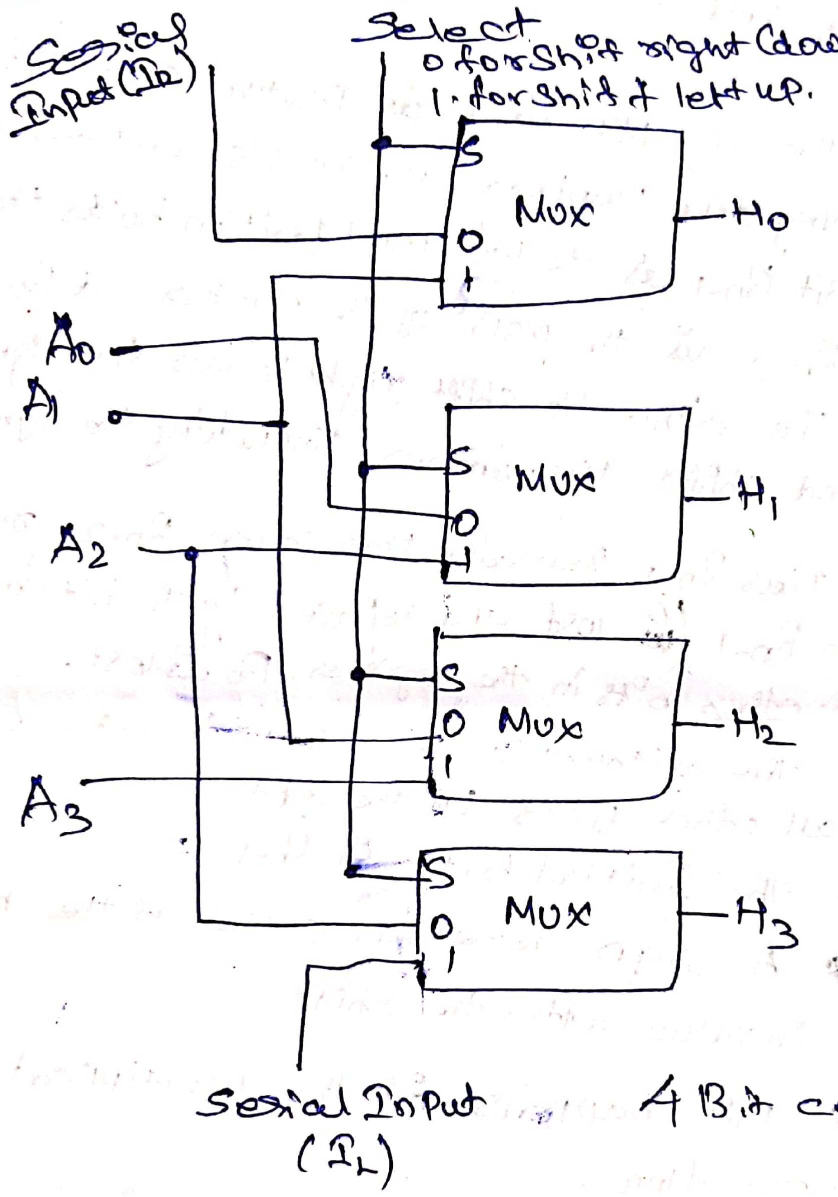
* The arithmetic shift left inserts a 0 into R_0, and shift all other bits to the left
* The Initial bit of R_{n-1} L
* A sign reversal occurs if the bit in R_{n-1} changes in value after the shift.
* This happens if the multiplication by 2 causes an overflow.
* An overflow occurs after an arithmetic shift left if initially, before the shift, R_{n-1} is not equal to R_{n-2}

* An overflow flip flop V_s can be used to detect an arithmetic shift left overflow.

$$V_s = R_{n-1} \oplus R_{n-2}$$

* If $V_s = 0$ there is no overflow, but if $V_s = 1$ there is an overflow and a sign reversal after the shift. V_s must be transferred into the overflow flipflop with the same clock pulse that shifts the register.

HARDWARE IMPLEMENTATION:-



Functional table

Select S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

4 Bit combinational circuit shifter

* A Possible choice for a shift unit would be a bidirectional shift register with Parallel load.

* Information can be transferred to the registers in parallel and then ~~Specified~~ Shifted to the right or left.

* In this type of configuration, a clock pulse is needed for loading the data into the register, and another pulse is needed to initiate the shift.

* In a processor unit with many registers it is more efficient to implement the shift operation with a combinational circuit.

* In this way the content of a register that has to be shifted is first placed onto a common bus whose output is connected to the combinational shifter, and the shifted number is then loaded back into the register.

* This requires only one clock pulse for loading the shifted value into the Register.

SHIFTER

* The 4 bit shifter has four data input, A_0 through A_3 and four data outputs, H_0 through H_3. There are two serial inputs, one for shift left (I_L) and the other for shift right (I_L)

* When the selection input $S=0$ the input data are shifted right.

* When the $S=1$, the input data are shifted left.

* A shifter with n data inputs and outputs requires n multiplexers.

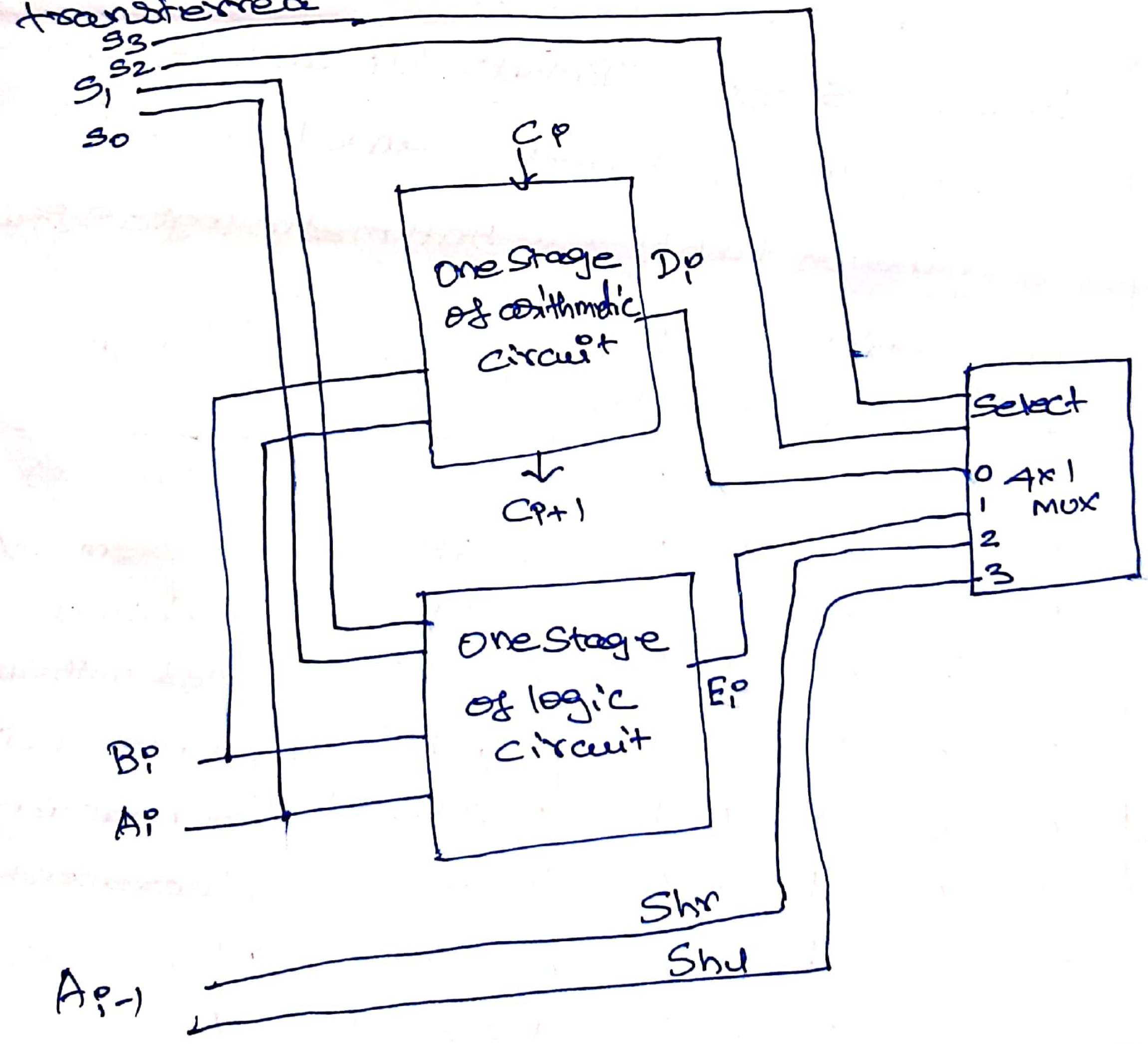
* The two serial inputs can be controlled by another multiplexer to provide three possible types of shifts.
[illegible]

ARITHMETIC LOGIC SHIFT UNIT :

Instead of having individual registers performing the microoperations directly computer system employs a number of storage registers connected to a common operational unit called an arithmetic logic unit, abbreviated ALU.

To perform a microoperation, the contents of specified registers are placed in the inputs of the common ALU.

The ALU performs an operation and the result of the operation is then transferred to a destination register



One stage of arithmetic logic shift unit

* The ALU is a combinational circuit so that the entire register transfer operation from the source registers through the ALU and into the destination register can be performed during one clock pulse period.

* The shift micro operations are often performed in a seperate clock, but sometimes the shift unit is made part of the ALU.

* The arithmetic, logic and shift circuits introduced in previous sections can be combined into the ALU with common selection variables.

* The subscript i designates a typical stage. Inputs A_i and B_i are applied of both the arithmetic and logic units.

Function table for Arithmetic logic Shift unit

Operation Select					Operation	Function
S_3	S_2	S_1	S_0	C_{in}		
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A+1$	Increment A
0	0	0	1	0	$F = A+B$	Addition
0	0	0	1	1	$F = A+B+1$	Add with carry
0	0	1	0	0	$F = A+\overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A+\overline{B}+1$	Subtraction
0	0	1	1	0	$F = A-1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	x	$F = A \wedge B$	AND
0	1	0	1	x	$F = A \vee B$	OR

0	1	1	0	x	$F = A \oplus B$	XOR
0	1	1	1	x	$F = \bar{A}$	Complement A
1	0	x	x	x	F = Shr A	Shift right A into F
1	1	x	x	x	F = Shl A	Shift left A into F

* A particular microoperation is selected with inputs S_1 and S_0. A 4x1 multiplexer at the output chooses between an arithmetic output in E_i and a logic output in H_i.

* The data in the multiplexer are selected with inputs S_3 and S_2.

* The other two data inputs to the multiplexer receive inputs A_{i-1} for the shift right operation and A_{i+1} for the shift left operation.

* the output carry C_{i+1} of a given arithmetic stage must be connected to the input carry C_i of the next stage in sequence.

* Input carry to the first stage is the input carry C_{in} which provides a selection variable for arithmetic operations.

* It provides eight arithmetic operation, four logic operations, and two shift operations.

* Each operations is selected with the five variables S_3, S_2, S_1, S_0, and C_{in}.

* The input carry C_{in} is used for selecting an arithmetic operations only.

* The first eight are arithmetic operations and are selected with $S_3 S_2 = 00$. The next four are logic operations are selected with $S_3 S_2 = 01$.

* The i/p carry has no effect during the logic operations and is marked with don't care x's

* The last two operations are shift operations and are selected with $S_3 S_2 = 10$ and 11.

* The other three selection inputs have no effect on the shift.

Logic microoperations

* Logic microoperations specify binary operations for strings of bits stored in registers.

* These operations consider each bit of the register separately and treat them as binary variables.

* For example, exor microoperations with, the contents of two Registers R1 and R2 is Symbolized by the Statement.

$$P: R1 \leftarrow R1 \oplus R2$$

* It specifies a logic microoperation to be executed on the individual bits of the registers Provided that the control variable P=1.

* As a numerical example, assume that each register has four bits. Let the content of R1 be 1010 and the content of R2 be 1100.

Ex-or microoperations

1010	content of R1
1100	content of R2
0110	Content of R1 after P=1

The content of R1 after the execution of the microoperation, is equal to the bit by bit ex-OR operation on Pairs of bits in R2 and Previous values of R1.

The logic microoperations are seldom used in scientific computations, but they are very useful for bit manipulation of binary data and for making logical decisions.

• It can be used to change bit values, delete a group of bits, or insert new bit values into a register.

Selective set :-

The selective set operation sets to 1 the bits in register A where there are corresponding 1's in register B

```
 1010   A before
 1100   B (logic operand)
 ----
 1110   A after
```

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

from the truth table we note that the bits of A after the operation logical OR operation of bits in B and previous value of A.

Therefore, the OR microoperation can be used to selectively set bits of a register

Selective - complement :-

The selective complement operation complements bits in A where there are corresponding 1's in B.

```
 1010   A before
 1100   B (logic operand)
 ----
 0110   A (after)
```

This example again can serve

Truth tables for 16 functions of two variables

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

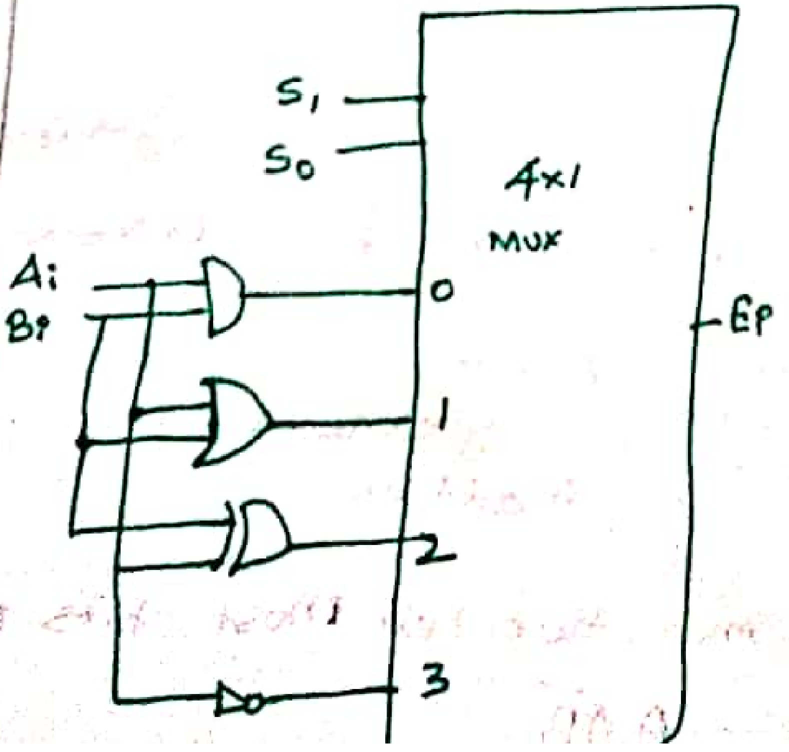
Hardware Implementation

The hardware Implementation of logic microoperations requires that logic gates be inserted for each bit or Pair of bits in the registers to Perform the required logic function.

Although there are 16 logic microoperations, most computers use only four - AND, OR, XOR (Ex-OR), and complement.

<u>Logic circuit</u>

It consists of four gates and a Multiplexer. Each of the four logic operations. is generated through a gate that Performs the required logic



S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \bar{A}$	Complement

Function table

Specific Symbols : This Symbols will be adopted for the logic microoperations OR, AND and Complement, to distinguish them from the corresponding symbols

$V \rightarrow$ OR Microoperation $\wedge$ – AND

The complement microoperation is the same as the 1's complement and use a bar on top. We will never use it to symbolize an OR microoperation. For example in the statement

$P+Q : R_1 \leftarrow R_2 + R_3, \quad R_4 \leftarrow R_5 \vee R_6$

the + between P and Q is an OR operation between two binary variable of a control function.

List of Logic Microoperations

Sixteen logic Microoperations

Booleanfunction	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Ex-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x+y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Ex NOR
$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
	$F \leftarrow$ all 1's	Set to all 1's

UNIT-1 BASIC STRUCTURE OF COMPUTERS

INTRODUCTION

A **Computer** is a programmable machine.

• The two principal characteristics of a computer are:

• It responds to a specific set of instructions in a well-defined manner.

• It can execute a prerecorded list of instructions (a program).

• Modern computers are electronic and digital.

• The actual machinery wires, transistors, and circuits is called hardware.

The instructions and data are called software.

• All general-purpose computers require the following hardware components:

• **Memory**: Enables a computer to store, at least temporarily, data and programs.

• **Mass storage device**: Allows a computer to permanently retain large amounts of data. Common mass storage devices include disk drives and tape drives.

• **Input device**: **Usually** a keyboard and mouse are the input device through which data and instructions enter a computer.

• **Output device:** A display screen, printer, or other device that lets you see what the computer has accomplished.

• **Central processing unit (CPU):** The heart of the computer, this is the component that actually executes instructions. • In addition to these components, many others make it possible for the basic components to work together efficiently. • For example, every computer requires a bus that transmits data from one part of the computer to another.

COMPUTER TYPES

Computers can be generally classified by size and power as follows, though there is considerable overlap:

• **Personal computer**: A small, single-user computer based on a microprocessor.

• In addition to the microprocessor, a personal computer has a keyboard for entering data, a monitor for displaying information, and a storage device for saving data.

• **Working station**: A powerful, single-user computer. A workstation is like a personal computer, but it has a more powerful microprocessor and a higher quality monitor.

• **Minicomputer**: A multi-user computer capable of supporting from 10 to hundreds of users simultaneously.

• **Mainframe:** A powerful multi-user computer capable of supporting many hundreds or thousands of users simultaneously.

• **Supercomputer:** An extremely fast computer that can perform hundreds of millions of instructions per second.

Minicomputer: • A midsized computer. In size and power, minicomputers lie between workstations and mainframes.

• A minicomputer, a term no longer much used, is a computer of a size intermediate between a microcomputer and a mainframe.

• Typically, minicomputers have been stand-alone computers (computer systems with attached terminals and other devices) sold to small and mid-size businesses for general business applications and to large enterprises for department-level operations.

• In recent years, the minicomputer has evolved into the "mid-range server" and is part of a network. IBM's AS/400e is a good example.

• The AS/400 - formally renamed the "IBM iSeries," but still commonly known as AS/400 - is a midrange server designed for small businesses and departments in large enterprises and now redesigned so that it will work well in distributed networks with Web applications.

• The AS/400 uses the PowerPC microprocessor with its reduced instruction set computer technology. Its operating system is called the OS/400.

• With multi-terabytes of disk storage and a Java virtual memory closely tied into the operating system, IBM hopes to make the AS/400 a kind of versatile all-purpose server that can replace PC servers and Web servers in the world's businesses, competing with both Wintel and Unix servers, while giving its present enormous customer base an immediate leap into the Internet.

Workstation:

1) A type of computer used for engineering applications (CAD/CAM), desktop publishing, software development, and other types of applications that require a moderate amount of computing power and relatively high quality graphics capabilities. • Workstations generally come with a large, high- resolution graphics screen, at least 64 MB (mega bytes) of RAM, built-in network support, and a graphical user interface.

2) In networking, *workstation* refers to any computer connected to a local-area network. It could be a workstation or a personal computer.

• **Mainframe:** A very large and expensive computer capable of supporting hundreds, or even thousands, of users simultaneously. In the hierarchy that starts with a simple microprocessors (in watches, for example) at the bottom and moves to supercomputer at the top, mainframes are just below supercomputers.

• In some ways, mainframes are more powerful than supercomputers because they support more simultaneous programs.

• But supercomputers can execute a single program faster than a mainframe. The distinction between small mainframes and minicomputers is vague, depending really on how the manufacturer wants to market its machines.

• **Microcomputer:** The term *microcomputer* is generally synonymous with personal computer, or a computer that depends on a microprocessor.

• Microcomputers are designed to be used by individuals, whether in the form of PCs, workstations or notebook computers. • A microcomputer contains a CPU on a microchip (the microprocessor), a memory system (typically ROM and RAM), a bus system and I/O ports, typically housed in a motherboard.

• **Microprocessor**: A silicon chip that contains a CPU. In the world of personal computers, the terms *microprocessor* and CPU are used interchangeably.

• A **microprocessor** (sometimes abbreviated μP) is a digital electronic component with miniaturized transistors on a single semiconductor integrated circuit (IC). • One or more microprocessors typically serve as a central processing unit (CPU) in a computer system or handheld device. • Microprocessors made possible the advent of the microcomputer. • At the heart of all personal computers and most working stations sits a microprocessor. • Microprocessors also control the logic of almost all digital devices, from clock radios to fuel-injection systems for automobiles. • Three basic characteristics differentiate microprocessors:

• **Instruction set**: The set of instructions that the microprocessor can execute.

• **Bandwidth**: The number of bits processed in a single instruction.

• **Clock speed**: Given in megahertz (MHz), the clock speed determines how many instructions per second the processor can execute.

In both cases, the higher the value, the more powerful the CPU. For example, a 32 bit microprocessor that runs at 50MHz is more powerful than a 16-bit microprocessor that runs at 25MHz.

• In addition to bandwidth and clock speed, microprocessors are classified as being either RISC (reduced instruction set computer) or CISC (complex instruction set computer).

• **Supercomputer**: A supercomputer is a computer that performs at or near the currently highest operational rate for computers.

• A supercomputer is typically used for scientific and engineering applications that must handle very large databases or do a great amount of computation (or both).

• At any given time, there are usually a few well-publicized supercomputers that operate at the very latest and always incredible speeds.

• The term is also sometimes applied to far slower (but still impressively fast) computers.

• Most supercomputers are really multiple computers that perform parallel processing.

Computer Types

Computer is a fast electronic calculating machine which accepts digital input, processes it according to the internally stored instructions (Programs) and produces the result on the output device.

The computers can be classified into various categories as given below:

- Micro Computer
- Laptop Computer
- Work Station
- Super Computer
- Main Frame
- Hand Held
- Multi core

Micro Computer:

A personal computer; designed to meet the computer needs of an individual. Provides access to a wide variety of computing applications, such as word processing, photo editing, e-mail, and internet.

Laptop Computer:

A portable, compact computer that can run on power supply or a battery unit. All components are integrated as one compact unit. It is generally more expensive than a comparable desktop. It is also called a Notebook.

Work Station:

Powerful desktop computer designed for specialized tasks. Generally used for tasks that requires a lot of processing speed. Can also be an ordinary personal computer attached to a LAN (local area network).

Super Computer:

A computer that is considered to be fastest in the world. Used to execute tasks that would take lot of time for other computers. For Ex: Modeling weather systems, genome sequence, etc (Refer site: http://www.top500.org/)

Main Frame:

Large expensive computer capable of simultaneously processing data for hundreds or thousands of users. Used to store, manage, and process large amounts of data that need to be reliable, secure, and centralized.

Hand Held:

It is also called a PDA (Personal Digital Assistant). A computer that fits into a pocket, runs on batteries, and is used while holding the unit in your hand. Typically used as an appointment book, address book, calculator and notepad.

Multi Core:

Have Multiple Cores – parallel computing platforms. Many Cores or computing elements in a single chip. Typical Examples: Sony Play station, Core 2 Duo, i3, i7 etc.

GENERATION OF COMPUTERS

Development of technologies used to fabricate the processors, memories and I/O units of the computers has been divided into various generations as given below:

- First generation
- Second generation
- Third generation
- Fourth generation
- Beyond the fourth generation

First generation: 1946 to 1955:

Computers of this generation used Vacuum Tubes. The computes were built using stored program concept. Ex: ENIAC, EDSAC, IBM 701. Computers of this age typically used about ten thousand vacuum tubes. They were bulky in size had slow operating speed, short life time and limited programmingfacilities.

Second generation: 1955 to 1965:

Computers of this generation used the germanium transistors as the active switching electronic device. Ex: IBM 7000, B5000, IBM 1401. Comparatively smaller in size About ten times faster operating speed as compared to first generation vacuum tube based computers. Consumed less power, had fairly good reliability. Availability of large memory was an added advantage.

Third generation: 1965 to 1975:

The computers of this generation used the Integrated Circuits as the active electronic components. Ex: IBM system 360, PDP minicomputer etc. They were still smaller in size. They had powerful CPUs with the capacity of executing 1 million instructions per second (MIPS). Used to consume very less power consumption.

Fourth generation: 1976 to 1990:

The computers of this generation used the LSI chips like microprocessor as their active electronic element. HCL horizen III, and WIPRO"S Uniplus+ HCL"s Busybee PC etc. They used high speed microprocessor as CPU. They were more user friendly and highly reliable systems. They had large storage capacity disk memories.

Beyond Fourth Generation: 1990 onwards:

Specialized and dedicated VLSI chips are used to control specific functions of these computers. Modern Desktop PC"s, Laptops or Notebook Computers.

Computer Registers

→ Computer Instructions are normally stored in Consequitive memory locations and are executed sequentially one at a time.

→ The Control Reads an instruction from a Specific address in memory and executes it. It then Continues by Reading the next instruction in Sequence and executes it, and so on.

Note: The instruction Sequence needs a Counter to Calculate the Address of next instruction after Current instruction is Completed.

→ Register is necessary in Control Unit for Storing the instruction Code after it is Read from memory. The Computer needs processor Registers for manipulating data and a Register for holding a memory address.

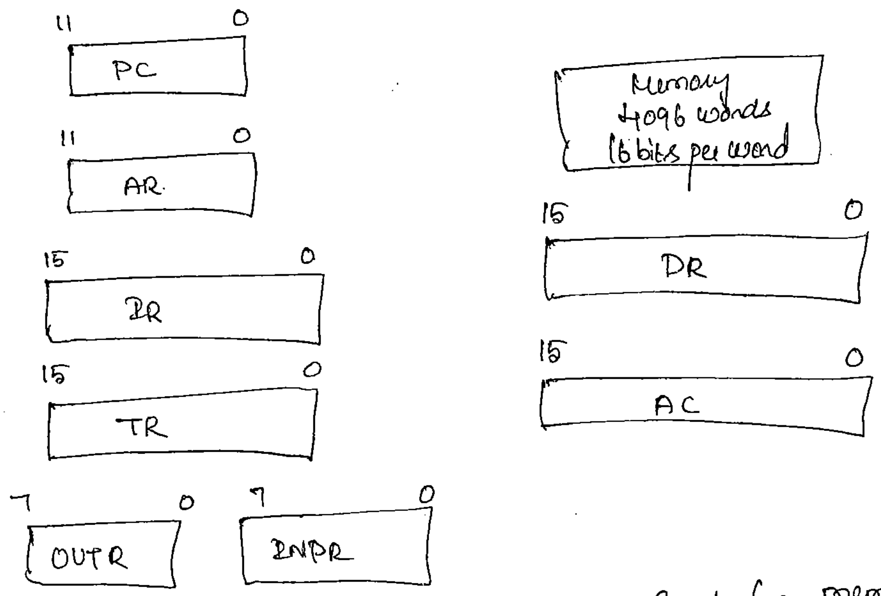
The table shows a list of Registers with description and number of bits they Contain.

Register Symbol	Number of bits	Register name	function.
DR	16	Data Register	Holds memory operands (data)
AR	12	Address Register	Holds address for memory.
AC	16	Accumulator	processor Register
IR	16	Instruction register	Holds instruction Code.
PC	12	program Counter.	Holds Address of instruction.
TR	16	Temporary Register	Holds Temporary data.
INPR	8	Input register	Holds input character
OUTR	8	output register	Holds output character

(words)

The Memory Unit has a Capacity of 4096 locations and each word has 16 bits. Twelve bits of an instruction word specify address of operand, three bits for the Operation and One bit to Specify direct or Indirect address.

Basic Computer registers and memory



The data register (DR) holds the operand read from memory. The Accumulator (AC) register is a general purpose processing register. The instruction read from memory is placed in instruction register (IR). The temporary register (TR) is used to store temporary data during the processing.

→ The Memory address register (AR) has 12 bits since this is the width of memory address. The program Counter (PC) also has 12 bits. And it holds the address of next instruction to be read from memory after current instruction is executed.

→ The 'pc' causes the computer to read sequential instructions unless a branch instruction is encountered. [The branch instruction calls for a transfer to a nonconsecutive instruction in the program]

→ To read an instruction the content of 'pc' is taken as address for memory and a memory read cycle is initiated. 'pc' is incremented by 'one' and holds the address of next instruction in sequence.

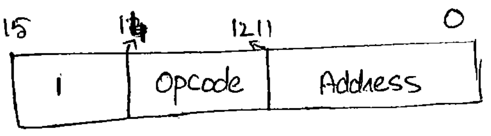
→ The two registers are used for input & output. The input register (INPR) receives '8' bit character from input device. The output register (OUTR holds a 8 bit character for output device.

Computer Instructions

The basic Computer has three instruction Code formats. Each format has 16 bits. The operation code (opcode) Contains three bits and the meaning of remaining 13 bits depends on the Operation Code encountered.

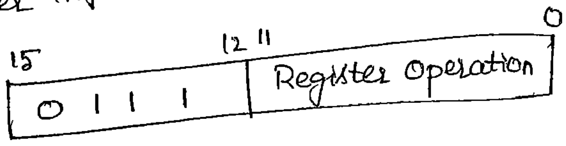
Basic Computer instruction formats

ⓐ Memory reference instruction.



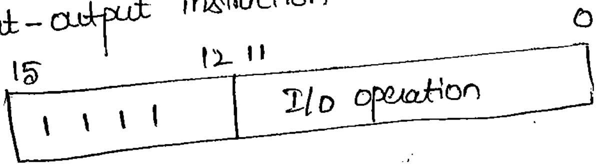
[Opcode = 000 through 110]

ⓑ Register reference instruction



[opcode = 111, I = 0]

ⓒ Input-output instruction.



[opcode = 111, I = 1]

→ The Memory reference instruction uses 12 bits to specify address and one bit to specify the addressing Mode 'I'. I is equal to '0' for direct address and '1' for indirect address.

→ The register reference instruction are recognized by operation code '111' with a '0' in the left-most bit of instruction. (15) [It performs an operation on AC registes]

Note :- An data (operand) from memory is not needed hence the other 12 bits specify the operation to be executed.

→ The Input output instruction doesnot need a reference from memory And recognized by operation code '111' with a '1' in the left-most bit. (15)

Note: The remaining bits specify the type of I/O operation.

The type of instruction is recognized by Computer Control from the instruction,
four bits in positions 12 through 15 of the
~~The~~ only three bits of the instruction are used for the operation code. It may seem that Computer is restricted to a maximum of eight distinct operations.
The instructions for a Computer are listed as. { basic Compu
25 instruction

Basic Computer Instructions

Symbol.	Hexadecimal Code I=0	I=1	Description
AND.	0XXX	8XXX	→ AND Memory word to F
ADD	1XXX	9XXX	→ ADD Memory word to F
LDA	2XXX	AXXX	→ Load memory word to A
STA	3XXX	BXXX.	→ Store. Content of AC in
BUN	4XXX	CXXX	→ Branch Unconditionally
BSA	5XXX	DXXX	→ Branch and Save return adc
ISZ	6XXX	EXXX	→ Increment and Skip if zero.

CLA →	7800 →	Clear AC
CLE →	7400 →	clear E
CMA →	7200 →	Complement AC
CME →	7100 →	Complement E
CIR →	7080 →	Circulate Right AC and E
CIL →	7040 →	Circulate left AC and E
INC →	7020 →	Increment AC
SPA →	7010 →	Skip next instruction If AC is positi
SNA →	7008 →	Skip next instruction If AC is nega
SZA. →	7004 →	skip next instruction if AC is ze
SZE →	7002 →	Skip next instruction if E is ze
HLT →	7001 →	Halt Computer.

→ Arithmetic, logic and shift process the data that users wish to employ. The information in a digital computer is stored in memory but all computations are done on processor registers. Therefore user has to move data b/w these two units.

→ Program control instructions such as branch instructions are used to change the sequence of execution. Input and output are needed for communication b/w computer and user. [Programs and data must be transferred to memory and results of computations must be transferred back to user].

→ There is one arithmetic instruction 'ADD', and two related instructions, complement AC (CMA) and increment AC (INC). With these we can add and subtract. The circulate instructions 'CIR' and 'CIL' can be used for arithmetic shifts, as well as other type of shifts.

→ The three logic operations (AND, complement AC (CMA) and clear AC (CLA). [ex: the 'AND' & complement provide a NAND operation

→ Moving information from memory to 'AC' is with load AC (LDA) instruction. Storing information from AC to memory is by STA (AC) instruction.

→ The input & output instructions cause information to be transferred b/w computer and external devices.

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.

→ The above symbols are used by programmers and users. The hexadecimal code is equivalent hexadecimal number of binary code used for instruction. [with this representation the 16 bits of instruction in binary code is reduced to four (4) bits.]
↓
hexadecimal ~~codes~~ digits.

→ In Register reference instruction use 16 bits to specify an operation. The left four bits are always 0111, which is equivalent to hexadecimal 7. The other three hexadecimal digits give binary equivalent of remaining 12 bits.

→ The input-output instructions also use all 16 bits to specify an operation. The last four bits are always 1111, equivalent to hexadecimal F.

Instruction set Completeness

A Computer should have a set of instructions so that user can construct machine language programs to evaluate any function that is known to be computable.

The set of instructions are said to be complete if computer includes a sufficient number of instructions in each of following categories.

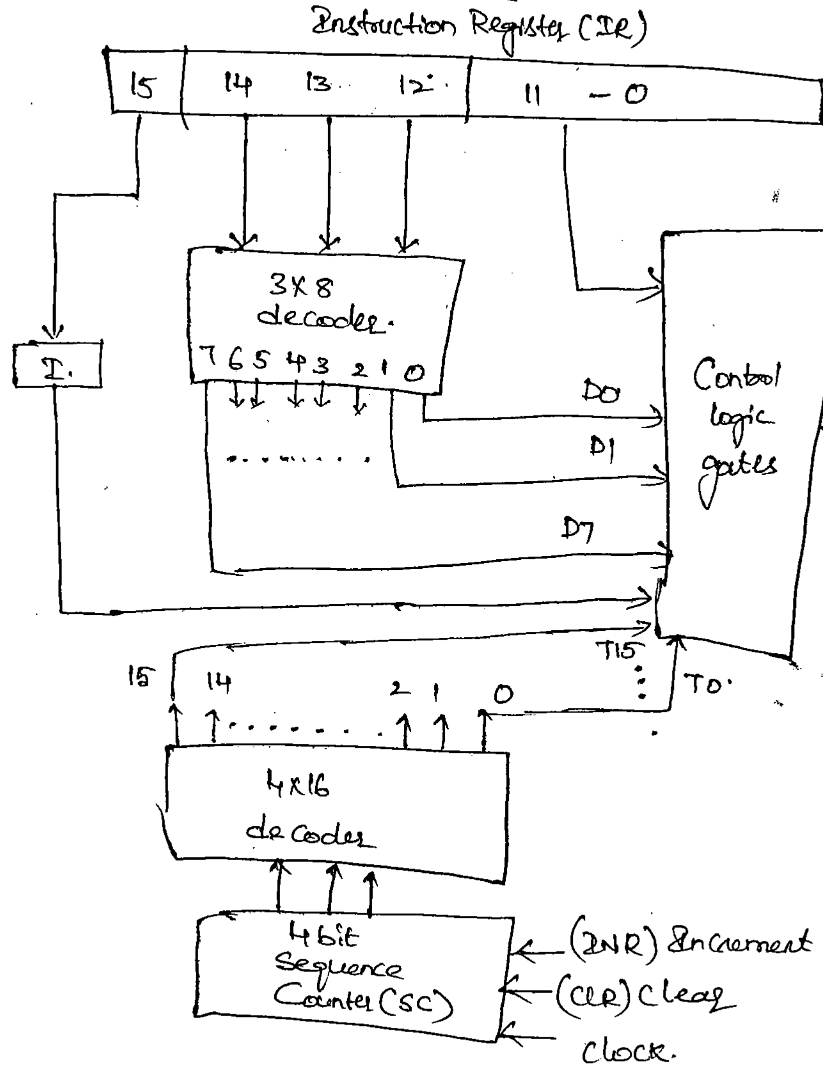
1. Arithmetic, logical and shift instructions.
2. Instructions for moving information to and from memory and processor registers.
3. program control instructions that check status cond
4. Input and output instructions.

Timing and Control

The timing for all registers in basic Computer is Controlled by a master Clock generator. The Clockpulses are Applied to all flip-flops and register. in the System, including the flipflops and registers in Control unit. The Clock pulses do not change the State of register Unless the register is enabled by a Control signal.

The Control s/gns are genrated in the Control unit and provide Control signals inputs for processor registers and microoperations for the Accumulator.

The Control unit organisation is of hardwired Control where the Control logic is implemented with gates, flipflops, decoders etc. The block diagram of Control Unit as represented here.



The block diagram of Control unit Consists of two decoders, a Sequence Counter, and a numbers of Control logic gates. An instruction read from memory is placed in instruction register (IR). The instructio register is divided into three parts: The 'I' bit, operation code and bits 0 through 11.

The operation code in bits 12 through 14 are decoded with a 3x8 decoder. The eight outputs are designated by symbols D_0 through

The Bit 15 of instruction is transferred to flipflop designate by symbol 'I'. Bits 0 through 11 are applied to Control logic gates.

The 4 bit Sequence Counter can count in binary from 0 through 15. The outputs of Counter are decoded into 16 timing s/gns T_0 through The Sequence Counter (SC) is incremented to provide the Sequence timing signals out of 4x16 decoder.

For example: The Register transfer statement.

$$\begin{tikzcd} \boxed{T_0 : AR \leftarrow PC} \arrow[r] \arrow[d] & \text{program Counter.} \\ \text{Address Register} \end{tikzcd}$$

The above instruction specifies a transfer of content of 'PC' into 'AR' if the timing s/g 'T_0' is active. During this time the content of 'PC' is placed on bus and the LD (load) input o 'AR' is enabled.

Instruction cycle

→ The program present in the memory Of Computer Consist of Sequence of instructions.

→ The program is executed by going through a Cycle for each instruction.

→ The basic Computer instruction cycle is divided into following phases.

(a) Fetch an instruction from Memory

(b) decode the instruction

(c) Read the effective address from Memory if the instruction has indirect Address.

(d) Execute the instruction.

Upon Completion of all '4' steps the Control goes back to 'step 1' to fetch, decode and execute. This process Continues unless a Halt instruction is encountered.

Fetch and decode :-

Initially the program counter 'pc' is loaded with the Address of first instruction in the program. The Sequence Counter 'sc' is Cleared to '0', providing a decoded timing s/g 'To'. After each clock pulse, 'sc' is incremented by one, so that timing s/g go through To, T1, T2 & so on.

The Microoperations for fetch and decode as shown

$T_0 : AR \leftarrow PC$

$T_1 : IR \leftarrow M[AR], PC \leftarrow PC+1$

$T_2 : D_0, \ldots D_7 \leftarrow \text{decode } IR(12\text{-}14), AR \leftarrow IR(0\text{-}11), I \leftarrow IR(15)$

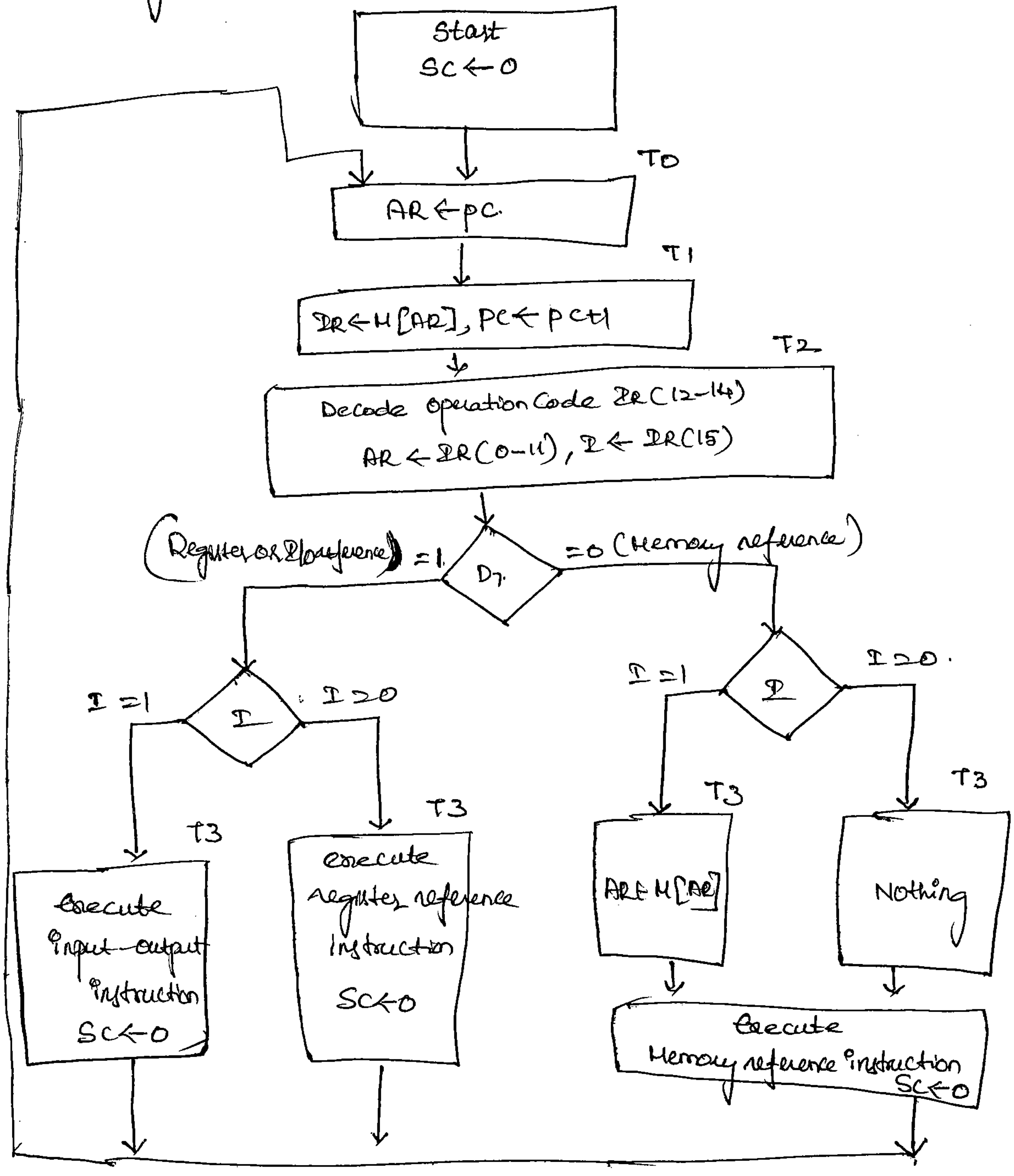
Since 'AR' is Connected to Address inputs of memory, it is necessary to transfer the address from pc to AR during timing s/g 'To'.

The instruction Read from Memory is placed in Instruction register(IR) with timing s/g 'T1', at Same time 'pc' is incremented by '1' to point to Next instruction.

At time 'T2', the operation code in 'IR' is decoded, the indirect bit is transfered to flipflop 'I' and the address part of instruction is transfered to 'AR'.

The flowchart presents an initial Configuration for instruction cycle. And shows how the Control determines the instruction type after decoding.

Instruction Cycle flowchart



Start
SC←0
T0
AR←PC
T1
IR←M[AR], PC←PC+1
T2
Decode operation Code IR(12-14)
AR←IR(0-11), I←IR(15)
D7
(Register or I/o reference) =1
=0 (Memory reference)
I
I=1
I=0
T3
Execute input-output instruction
SC←0
Execute register reference instruction
SC←0
AR←M[AR]
Nothing
Execute Memory reference instruction
SC←0

CO-Unit-3

Computer Arithmetic

Arithmetic instructions in digital Computers manipulate data to produce Results necessary for Computational problems. The four basic arithmetic operations are addition, subtraction, Multiplication and division. from the above four operations, it is possible to formulate other. arithmetic functions.

→ An arithmetic processor is a part of processor unit to Perform arithmetic operations. An arithmetic instruction Can Specify binary or decimal data, and in each Case the data may be Fixed point or floating point form.

Algorithm :→ An algorithm will Contain a number of procedural Steps which are dependent on Results of previous Steps. A Convenient method for presenting algorithm is flowchart.

The Computational Steps are Specified in flowchart inside Rectangular boxes. The decision Steps are indicated inside diamond Shaped boxes from which two or more alternate paths emerge.

In this topics we discuss about various algorithms and Show the procedure for implementing them with digital hardware.

1. The binary data in Signed Magnitude representation.
2. The binary data in Signed-2's Complement representation.

Addition and Subtraction

There are three ways of Representing negative binary numbers. Signed magnitude, Signed 1's Complement, Signed 2's Complement. Most Computers use the Signed 2's Complement representation when performing arithmetic operations with integers.

Here we develop the algorithms for addition and Subtraction for data in Signed magnitude and again in Signed 2's Complement.

Addition and Subtraction with Signed Magnitude data

The Representation of numbers in Signed magnitude is familiar because it is used in everyday arithmetic calculations.

Designate the Magnitude of 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 numbers and operation performed.

The algorithms for addition and Subtraction is derived from the table.

Addition and Subtraction of Signed Magnitude numbers.

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

Addition (Subtraction) algorithm :

When the Signs of `A' and `B' are identical (different), add the two magnitudes and attach the Sign of `A' to the result. When the signs of A and B are different (identical), Compare the magnitudes and Subtract Smaller number from larger.

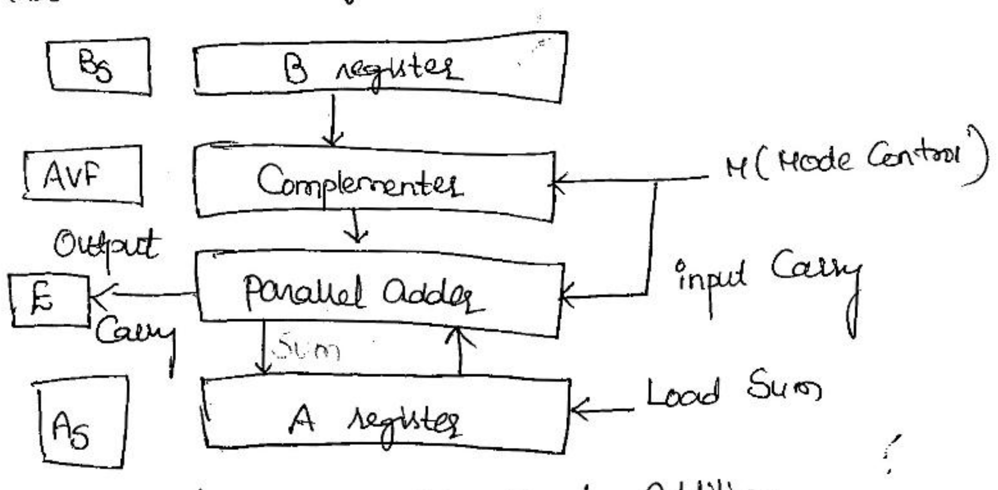
Choose the Sign of the result to be same as `A' if A>B or the Complement of Sign of `A' if A<B. If two are equal Subtract B from A and make the Sign of result positive.

The two algorithms are Similar Concept ~~of~~ for Sign Comparison. [The procedure followed for identical signs in addition is same as for different signs in Subtraction algorithm].

Hardware Implementation :-

To implement the two arithmetic operations Addition & Subtraction with hardware, it is first necessary that the two numbers be stored in Registers.

Let `A' and `B' be two registers that holds the magnitude of numbers, and `As' and `Bs' be two flipflops that holds the Corresponding signs. The Result Can be transferred into `A' and `As'. Thus `A' and `As' together form an Accumulator register.



Hardware for Signed Magnitude addition and Subtraction.

Hardware implementation of algorithms

(1) First a parallel adder is needed to perform the Microoperation A+B.

(2) Second a Comparator is needed to establish if A>B, A<B or A=B.

(3) Third two parallel Subtractors Circuits to perform the Microoperations A-B and B-A.

The procedure requires Comparator, adder and two Subtractors. However different procedure ~~to~~ that requires less equipment is found.

(1) The Subtraction can be Accomplished by means of Complement and Add. Hence this procedure requires only one adder and Complementer.

In the above diagram the Subtraction is done by adding 'A' to the 2's Complement of 'B'. The output Carry is transferred to flip-flop 'E', where it can be checked to determine the magnitudes of two numbers.

The Add overflow flip (AVF) holds the overflow bit when A and B are added.

Operation: The addition of 'A' plus 'B' is done through parallel adder. The S(Sum) output of adder is Applied to the input of 'A' register. The Complementer provides an output of 'B' or Complement of 'B' depending on the State of mode Control 'M'. The parallel adder Consists full adder Circuits.

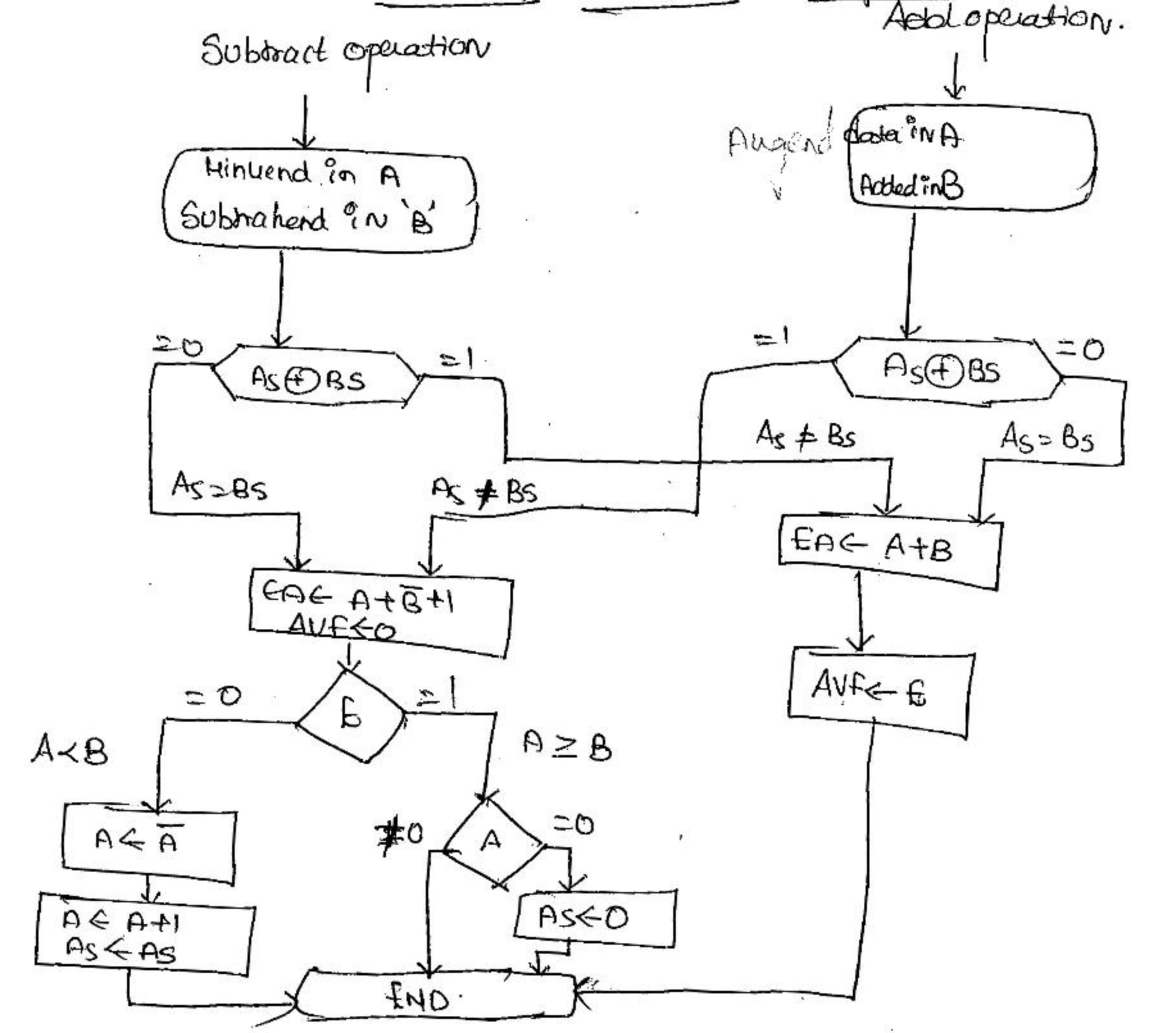
The 'M' signal is applied to the input Carry of adder. When M=0 the output of 'B' transfered to adder, the input Carry '0', and the output of adder is equal to Sum A+B.]

[When M=1, the 1's Complement of 'B' is Applied to adder, the input Carry is '1' and output $S = A+\bar{B}+1$. This is equal to Subtraction A-B

Hardware Algorithm

The flowchart of hardware algorithm is presented.

Flowchart for Add & Subtract Operations



The two Signs A_s and B_s are Compared by an Exclusive OR gate. If the output of gate is '0', the signs are identical; if it is '1' the Signs are different.

For an Add operation, identical Signs dictate that the magnitudes to be added. For a Subtract operation, different Signs dictate that the magnitudes be added. The Magnitudes are added with Microoperation $EA \leftarrow A+B$, where EA is a register that Combines 'E' and 'A'. The Carry in 'E' after addition Constitutes an Overflow.

The 'E' is transferred to add-overflow flipflop AVF.

The two magnitudes are subtracted if the signs are different for add operation or identical for subtract operation. The magnitudes are subtracted by adding 'A' to the 2's complement of 'B'. No overflow can occur if numbers are subtracted so AVF is cleared to '0'.

A '1' in E indicates that $A \geq B$ and the number in 'A' is the correct result. A '0' in 'E' indicates that $A < B$. for this case it is necessary to take 2's complement of value in 'A'. This is done with one microoperation $A \leftarrow \bar{A}+1$. The final result is found in register 'A' and its sign in As. The value of AVF indicates that an overflow indication.

Addition and Subtraction with signed 2's complement data:

* When two numbers of 'n' digits are added and the sum occupies n+1 digits, then the overflow is occured.
* Overflow can be detected by inspecting the last two carries out of addition.
* The last two carries are applied to XOR gate and if the O/p of XOR is '1', it is the indication of overflow.
* The hardware implementation is given in fig. 3.3

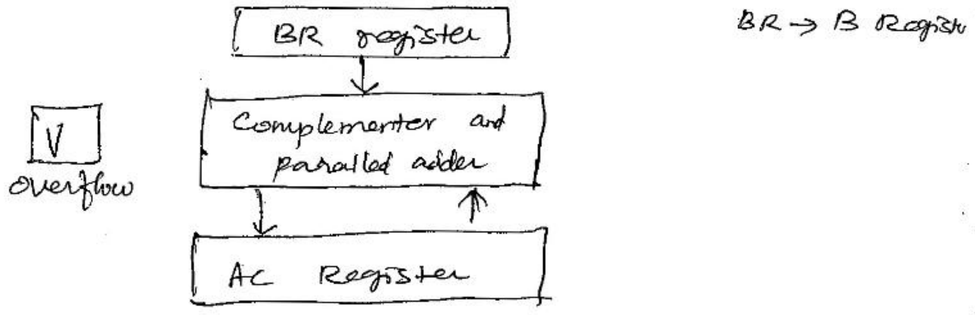


fig. 3.3 - Hardware implementation of 2's Complement addition & subtraction.

* The leftmost bit in AC and BR (Accumulator and B-Register) represent the sign bits of the number.
* The two sign bits are added or subtracted together with the other bits in the complementer and parallel adder.
* The overflow flip-flop V(AVF) is set to 1 if there is an overflow.

The algorithm for adding / subtracting two binary numbers in signed 2's complement representation is shown in flow chart of fig. 3.4.

25/8/15
2, 4, 22, 2~~4~~,
41, 45, 57, 59,
60, 403,

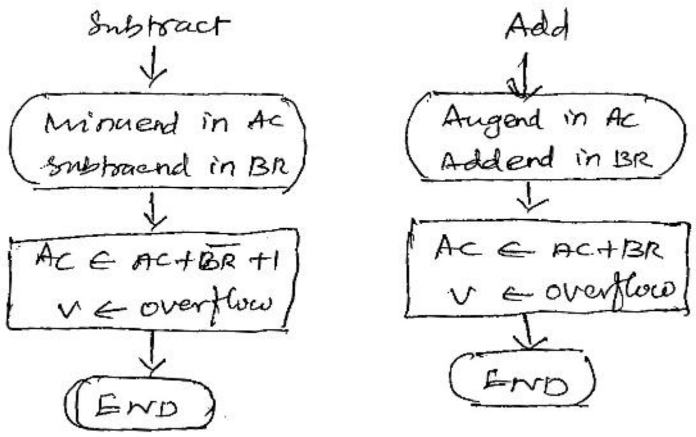


fig. 3.4 - Algorithm for addition / Subtraction using 2's complement method - flow chart.

* Sum is obtained by adding AC & BR. V is set to '1' if XOR of last two carries is set to '1'.

* The subtraction is done by adding the content of AC with 2's complement of BR.

Multiplication algorithms

→ Multiplication of two binary numbers in Signed Magnitude Representation is done with Successive Shift and Add operation.

ex:

```
23        10111  Multiplicand
19      × 10011  Multiplier.
          -----
          10111
         10111      +
        00000
       00000
      10111
      ---------
437   110110101  Product.
```

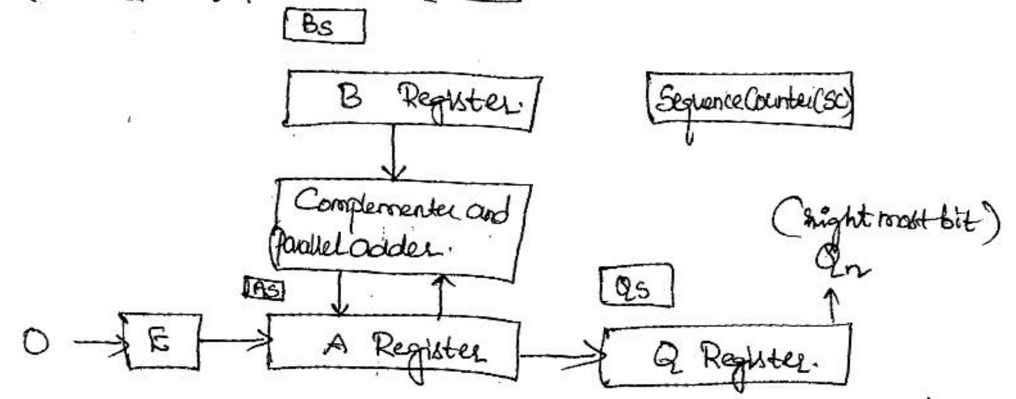
If the Multiplier bit is '1' the Multiplicand is Copied down,
If the Multiplier bit is '0' the zero's are Copied down. finally all the products are Added to get the desired product. The Sign of product is determined from the Sign of Multiplicand and multiplier.
[If they are Same Sign the product is positive, otherwise the Sign of product is negative]

Hardware implementation for Signed Magnitude data

The points to be Considered for H/w implementation:-

1. First Instead of providing Register to Store and add Simultaneously as many numbers there are in Multiplier, It is Convinient to provide adder for two binary numbers and Successively Accumulate.
2. Second instead of Shifting the Multiplicand to left, the partial product is Shifted to right.
3. Third When Corresponding bits of Multiplier is zero, there is no need to add all zeros to the partial product.

The hardware equipment Consists of :-



Steps: The Multiplier is stored in Q register and its sign in 'Qs'. The 'B' register holds the multiplicand and Sign in 'Bs'. The Sequence Counter is initialized with the number equal to number of bits in multiplier.

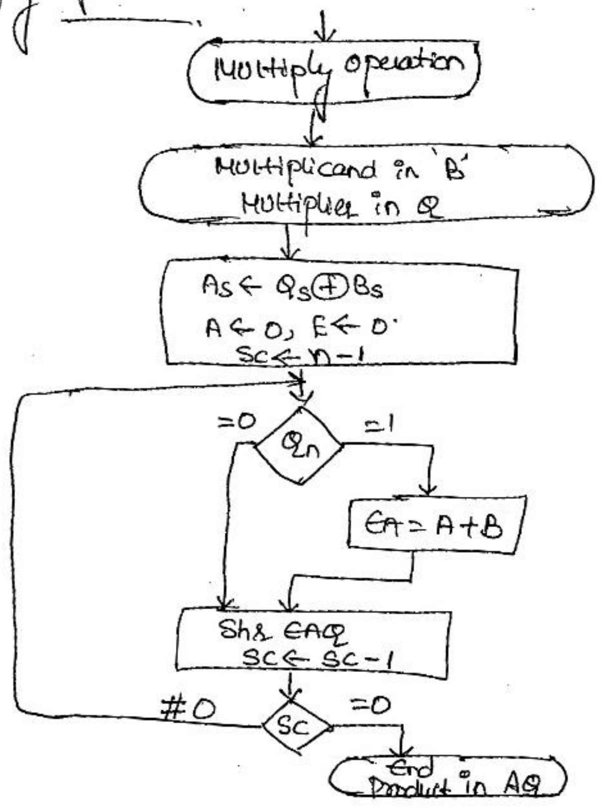
→ The Counter is decremented by '1' after forming each partial product. When the Counter reaches zero, the product is formed and process stops.

→ The Sum of A and 'B' forms the partial products which is transferred to 'EA' register. The shift will be denoted by Statement Shr EAQ + designate the right shift.

→ The least significant bit of 'A' is shifted into the most significant position of Q.

Hardware Algorithm

Flowchart for Multiply operation :-



Hardware algorithm.

Steps :-

① Initially the Multiplicand is in 'B' and the multiplier in 'Q' and there Corresponding Signs are in 'Bs' and 'Qs' respectively.

② Register 'A' and 'E' are Cleared and the Sequence Counter 'Sc' is Set to a number equal to the number of bits in multiplier.

③ After the initialization, the low order bit of Multiplier in Q_n is tested. If it is '1' the Multiplicand in 'B' is added to the present partial product in 'A'. If it is '0', nothing is done.

④ Register EAQ is Shifted once to the right to form the New partial product.

⑤ The 'Sc' is decremented by 'one' and its new Value is Checked. If it is not zero the process is repeated and if it is zero the process Stops.

⑥ The final product is Available in both 'A' And 'Q' with 'A' holding Most Significant bits and 'Q' holding least Significant bits.

example. (Multiply 23×19 = 437)

Multiplicand B = 10111	E	A	Q	Sc.
Multiplier in 'Q'	0	00000	10011	101
$Q_n = 1$; add 'B'		10111		
First partial product →	0	10111		
Shift right EAQ. →	0	01011	11001	100
$Q_n = 1$; add B →		10111		
Second partial product →	1	00010		
Shift right EAQ. →	0	10001	01100	011
$Q_n = 0$; Shift right EAQ. →	0	01000	10110	010
$Q_n = 0$; Shift right EAQ →	0	00100	01011	001
$Q_n = 1$ add B →		10111		
Fifth partial product →	0	11011		
Shift right EAQ →	0	01101	10101	000

final product in $\boxed{AQ = 0110110101}$

Booth Multiplication Algorithm (Signed 2's Complement representation)

The Booth algorithm gives a procedure for multiplying binary numbers in Signed 2's Complement representation. It operates on the fact that Strings of 0's in the multiplier require no addition but just Shifting.

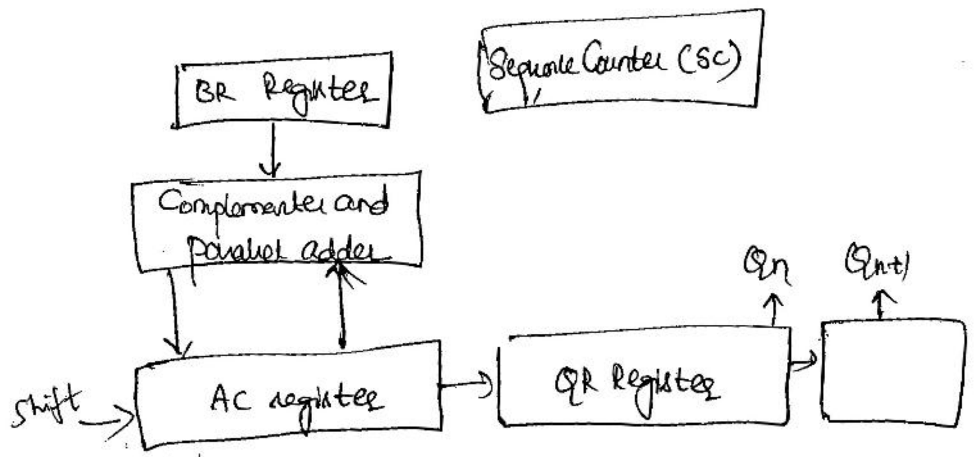
The Rules followed are:-

(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 multiplicand is added to the partial product Upon encountering first '0' 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.

The hardware implementation of Booth algorithm.

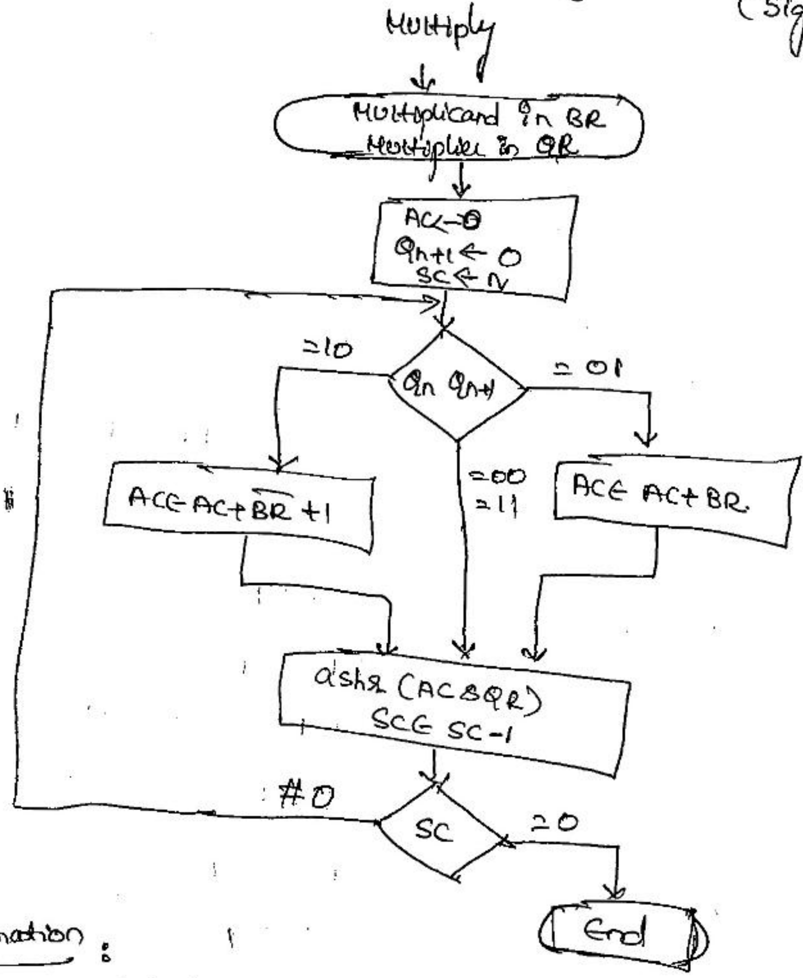


The hardware requires the register Configuration. This is Similar to the Signed Magnitude Representation hardware except the Sign bits are not Seperated from the Rest of the Registers.

To Show the difference the registers are renamed as A, B and Q as AC, BR and QR respectively.

'Q_n' designates the least significant bit of multiplier in register. QR. An extra flipflop Q_{n+1} is appended to QR to facilitate. a double inspection of multiplier.

The flowchart for Booth algorithm for Multiplication (Signed 2's Complement Numbers)



explanation :

Initially the 'AC' and Q_{n+1} is cleared to '0' and the Sequence Counter is set to a number 'n' equal to number of bits in multiplier.

The two bits of the multiplier 'Q_n' and 'Q_{n+1}' are inspected.

If two bits are equal to 10, it means the first '1' has been encountered. This requires a subtraction of multiplicand from the partial product in 'AC'

→ If two bits are equal to 01, it means that the first '0' in string '0's has been encountered. This requires addition of multiplicand to the partial product in 'AC'.

→ When the two bits are equal, the partial product does not change.

The next step is to shift right the partial product and the multiplier (including bit Q_{n+1}). This is an arithmetic shift right (ashr) operation which shifts 'Ac' and 'QR' to right, and leaves the sign bit in 'Ac' unchanged. The Sequence Counter is decremented and the loop runs 'n' times.

example of multiplication Booth algorithm $((-9)\times(-13) = +117)$

Q_n Q_{n+1}	BR = 10111 $\overline{BR}+1 = 01001$	Ac	QR	Q_{n+1}	SC.
	Initial	00000	10011	0	101
1 0	Subtract BR.	01001			
		01001			
	ashr	00100	11001	1	100.
1 1	ashr	00010	01100	[illegible]	011
0 1	Add BR	10111			
		11001			
	ashr	11100	10110	0	010
0 0	ashr	01110	01011	0	001
1 0	Subtract BR	01001			
		00111			
	ashr	00011.	10101	1	000

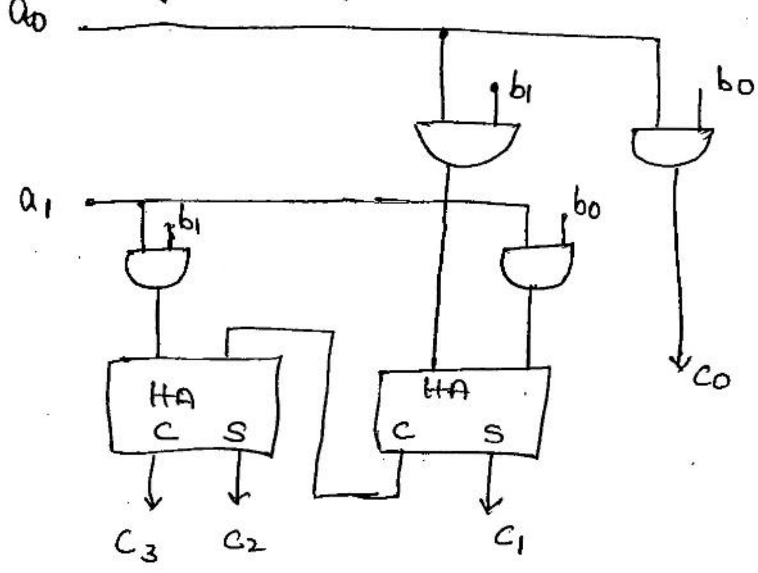
Array Multiplier

→ Checking the bits of Multiplier one at a time and forming Partial product is a Sequential operation that requires a Sequence of Add and Shift microoperation.

→ The Multiplication of two binary numbers can be done with one micro-operation by means of a Combinational Circuit that forms the product bits all at Once.

→ This is a fast way of Multiplying two numbers Since all it takes is the time for the Signals to propogate through the gate that forms Multiplication array.

ex: 2 bit by 2 bit array Multiplier.



ex:

$b_1 \quad b_0 \rightarrow$ 2 bit number

$a_1 \quad a_0 \rightarrow$ 2 bit number

$a_0b_1 \quad a_0b_0$.

$a_1b_1 \quad a_1b_0$

$C_3 \quad C_2 \quad C_1 \quad C_0$.

→ The Multiplicand bits are b_1 and b_0, multiplier bits are 'a_1' and 'a_0' and the product bits are C_3, C_2, C_1 and C_0.

→ The first partial product is obtained by multiplying a_0 by $b_1 b_0$.

→ The Multiplication of two bits a_0 & b_0 produces '1' if both are '1', otherwise it produces '0'. This is identical to 'AND' operation.

→ The Second partial product is formed by multiplying 'a_1' by $b_1 b_0$ and is Shifted one position to left.

→ The two partial products are added with two half adder. Circuit.

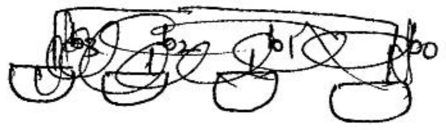
→ The Combinational Circuit with More bits Can be Constructed in Similar fashion.

→ The binary output in each level of 'AND' gate is Added in parallel with partial product of previous level to form a New partial product

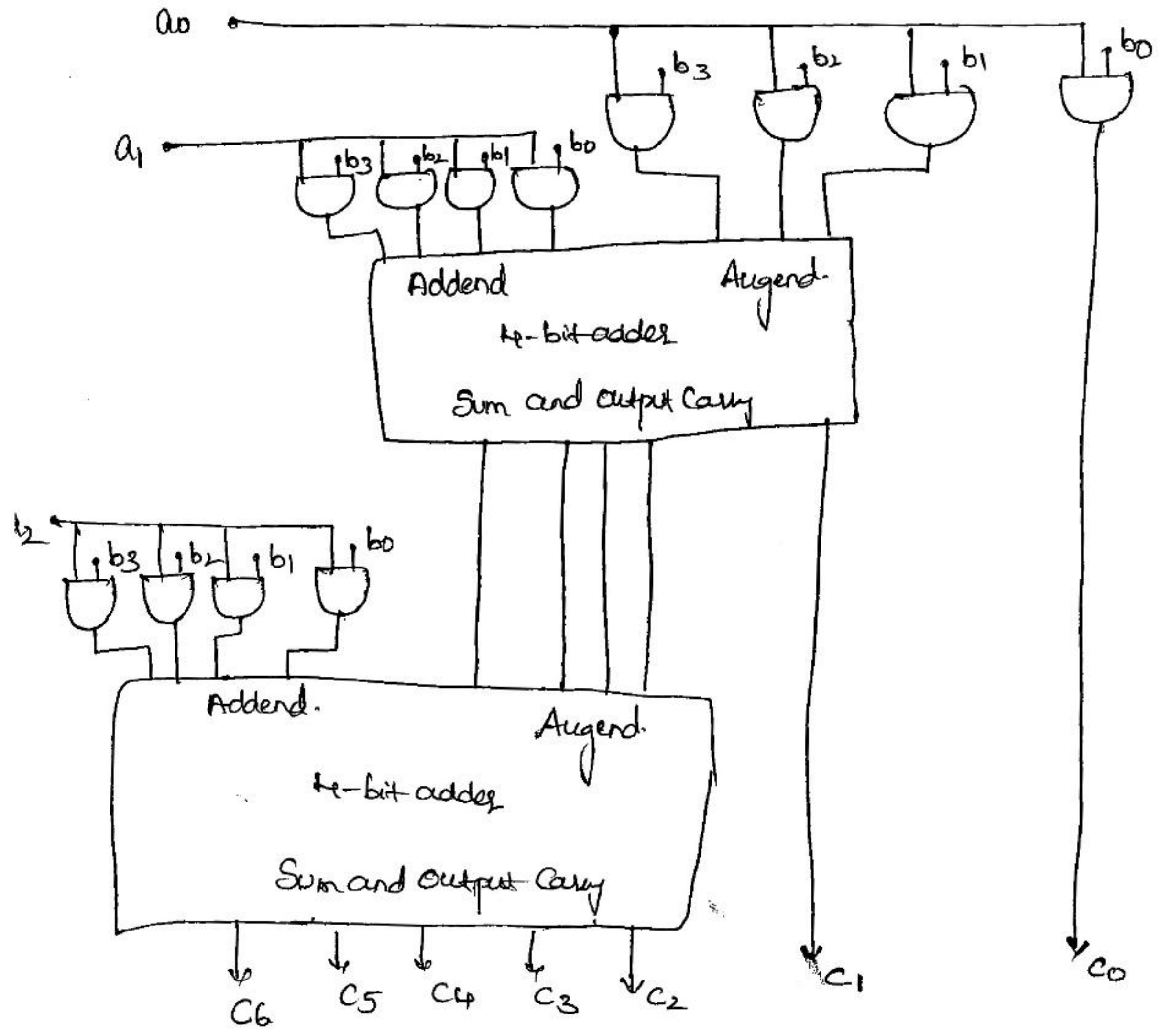
→ (ex: 2) Consider a multiplier that multiplies binary number of 4 bits with a '3' bit number.

→ The Multiplicand is represented as $b_3 b_2 b_1 b_0$ And the multiplier by $a_2 a_1 a_0$.

The logic diagram of Multiplier is Shown as

Since the $k=4$ and $j=3$, we need. 

'12' AND gates and two 4 bit adders to produce a product of. Seven bits ($j+k$) bits.



4 bit by 3 bit array Multiplier

Division Algorithm

→ The basis for the Algorithm involves repetitive shifting and addition or subtraction operation.

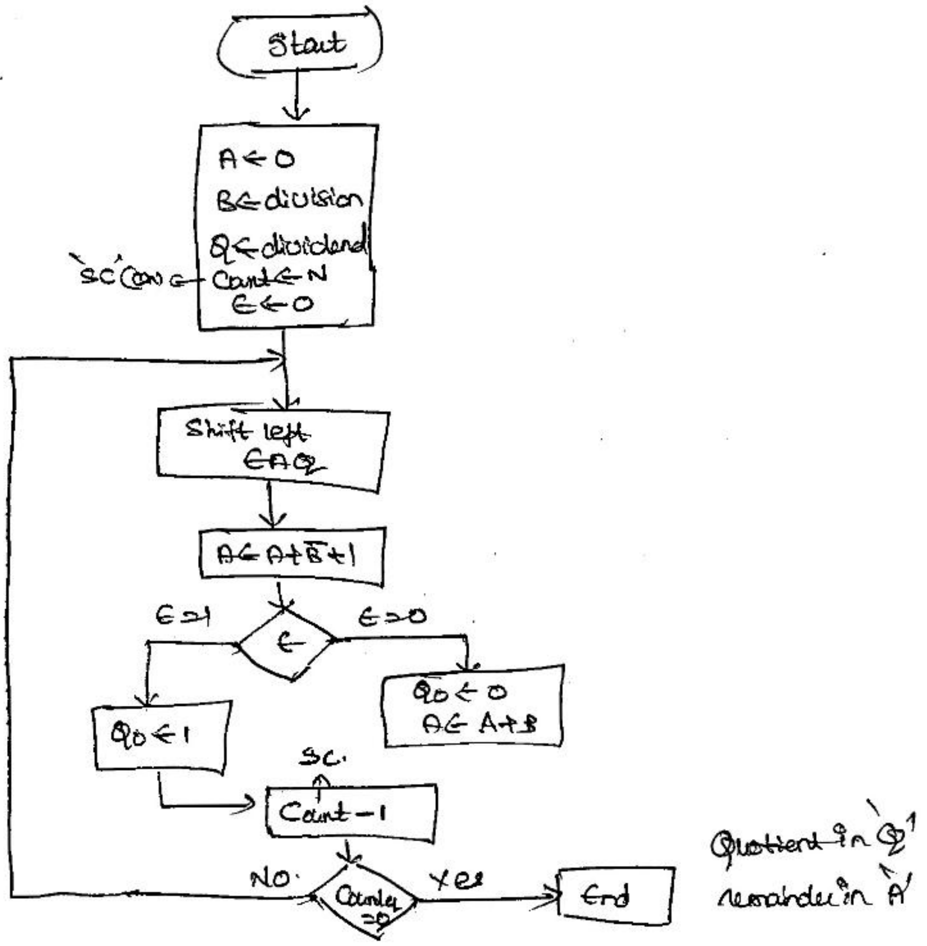
→ Binary division is simpler than decimal division because the quotient digits are either '0' or '1' and there is no need to estimate how many times the dividend fits in-to the divisor.

ex: 147/11 = 13 with remainder '4'

```
                      00001101 ← quotient
  Divisor → 1011 | 10010011 ← dividend
                    1011↓
                  ----------
 partial      → 001110
 remainders        1011
                  ----------
              → 001111
                     1011
                  ----------
                      100 ← Remainder.
```

* first the bits of dividend are examined from left to right, until the set of bits examined represent the number greater than or equal to divisor.
* Until this event occurs 0's are placed in quotient from left to right.
* when event occurs '1' is placed in quotient and divisor is subtracted from the partial dividend. The result is referred as partial remainder.
* At each cycle the additional bits from the dividend are appended to the partial remainder, until the result is greater than or equal to divisor & this process continues all the bits of dividend are compared.

The flowchart of division algorithm



The algorithm can be summarised as

1. Load the divisor into 'B' reg, dividend into 'Q' register, SC (Sequence Counter) with 'n' number of bits in divisor and 'A', 'E' register with zeros.

2. Shift EAQ left '1' bit position.

3. perform the subtraction EA ← A + $\bar{B}$ + 1

4. 'E' value is checked whether it is equal to 0, if it does Q_0 bit is set to '0' and 'B' is added back to A.

EA ← A+B

5. otherwise Q_0 is set to '1'. Sequence Counter (SC) is decremented by '1'.

6. The process continues for 'n' times and finally quotient will be in 'Q' and remainder in 'A'.

Assume the division operation.

$3 \rightarrow 00011 \rightarrow B$

$7 \rightarrow 0111 \rightarrow Q$.

$3\overline{)7}(2 \rightarrow Q$
$\underline{6}$
$1 \rightarrow R$.

00011 ← B
(3)

Initial values.

E	A	Q	
0	00000.	0111 (7) ↖ '0' inserted	
0	00000	1110	Shift left.
0	11101	1110	Subtract 'B' from A.
1	00000	1110	when E=0 Add B to A.
1	00000	1110	Clear q_0
0	00001	1100	Shift left.
0	11110	1100	Subtract B from A.
1	00001	1100	when E=0 add B to A.
1	00001	1100	Clear q_0
0	00011	1000	Shift left
1	00000	1000	Subtract B from A.
1	00000	1001	Set q_0
0	00001	0010	Shift left.
0	11110	0010	Subtract ~~Left~~ B from A.
0	00001	0010	when E=0 add B to A.
1	00001	0010	Clear q_0.

00001 ↓ remainder.

0010 ↓ Quotient

$00001 \rightarrow$ '1' $\rightarrow$ Remainder

$0010 \rightarrow$ '2' $\rightarrow$ quotient.

$A+\bar{B}+1$

$00011 \rightarrow B$

$11100 \rightarrow \bar{B}$

$11100 \quad \bar{B}+1$
$\underline{\quad\quad 1}$
11101

$A+\bar{B}+1$

$11101 \rightarrow \bar{B}+1$
$\underline{00000}$
'A' ← $11101 \rightarrow A+\bar{B}+1$

when 'B' added to 'A'

Now 'A' is 11101
$\underline{00011}$
$\boxed{1}00000$

Decimal Arithmetic Unit

The ~~User of a Computer~~

The CPU (Central processing Unit) with an Arithmetic logic Unit (ALU) can perform arithmetic microoperations with binary data. To perform arithmetic operations with decimal data, it is necessary to convert the input decimal numbers to binary, and perform calculations, finally convert the results into decimal.

Note: When application calls for large amount of calculations, it becomes convenient to do the arithmetic directly with decimal numbers. Computers capable of performing decimal arithmetic must store the decimal data in binary coded form (BCD). The decimal numbers are then applied to decimal arithmetic unit for decimal microoperations.

ex: Electronic calculators use decimal arithmetic unit because input and output are frequent. Many computers have hardware for arithmetic calculations with both binary and decimal data. It is responsibility of user to specify instructions whether they want the computer to perform calculations with binary or decimal data.

This decimal arithmetic unit can add or subtract decimal numbers, usually by forming the 9's and 10's complement of subtrahend.

The decimal arithmetic unit consists of nine binary input variables and five binary output variables, since a minimum of four bits is required to represent each coded decimal digit. Each stage must have four inputs for augent digit, four inputs of addend digit, and an input carry. The output includes four terminals for sum digit and one for output carry.

BCD Adder :-

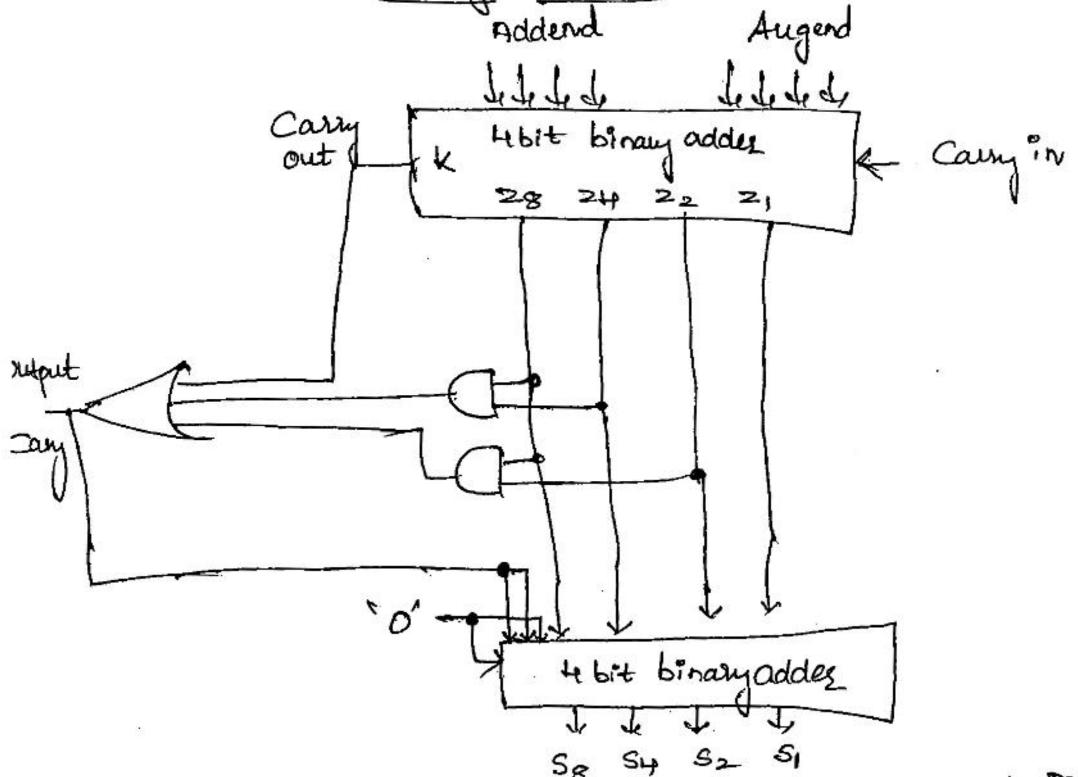
The binary numbers labeled by Symbols K, Z_8, Z_4, Z_2 and Z_1. where 'k' is the Carry and the Subscripts under the letter 'z' represents the weights 8, 4, 2 and '1' that Can be assigned to four bits in BCD Code.

The first Column in table lists the binary sums as they appear in the outputs of 4 bit binary adder. The output Sum of two decimal numbers are ~~stored~~ Listed in Second Column and represented in BCD.

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

By examining the table it is apparent that when binary sum is equal to or less than 1001, the corresponding BCD number is identical and no conversion is needed. When binary sum is greater than 1001, we obtain an nonvalid BCD number. The addition of binary 6 (0110) to the binary sum converts to correct BCD and also produces output carry as required.

Block diagram of BCD adder



The above circuit adds two BCD digits in parallel and produces the sum also in BCD. The correction logic is included in internal construction. The logic circuit detects the necessary correction. It is obvious that a correction is needed when binary sum has an output carry K = 1.

The condition for correction and output carry can be expressed by Boolean function.

$$C = K + Z_8 Z_4 + Z_8 Z_2$$

When C = 1, it is necessary to add 0110 (6) to the binary sum and provide output carry to next stage.

THE 8086 MICROPROCESSOR

1.1 FEATURES OF 8086

- The 8086 is a 16 bit processor.
- The 8086 has a 16 bit Data bus.
- The 8086 has a 20 bit Address bus.
- Direct addressing capability 1 M Byte of Memory (2^{20}).
- It provides fourteen 16-bit register.
- 24 Operand addressing modes.
- Bit, Byte, Word, and Block operations.
- 8 and 16-bit Signed and Unsigned arithmetic operations including multiply and divide.
- Four general-purpose 16-bit registers: AX, BX, CX, DX
- Two Pointer group registers: Stack Pointer (SP), Base Pointer (BP)
- Two Index group registers: Source Index (SI), Destination Index (DI)
- Four Segment registers: Code Segment (CS), Data Segment (DS), Stack Segment (SS), Extra Segment (ES)
- 6 status flags and 3 control flags.
- Memory is byte-addressable—each address stores an 8-bit value.
- Addresses can be up to 32 bits long, resulting in up to 4 GB of memory.
- Range of clock rates: 5 MHz for 8086, 8 MHz for 8086-2, 10 MHz for 8086-1
- Multibus system compatible interface
- Available in 40pin Plastic Package and Lead Cerdip.

1.2 8086 MICROPROCESSOR ARCHITECTURE

The internal functions of the 8086 processor are partitioned logically into two processing units as shown in the Fig.1.1.

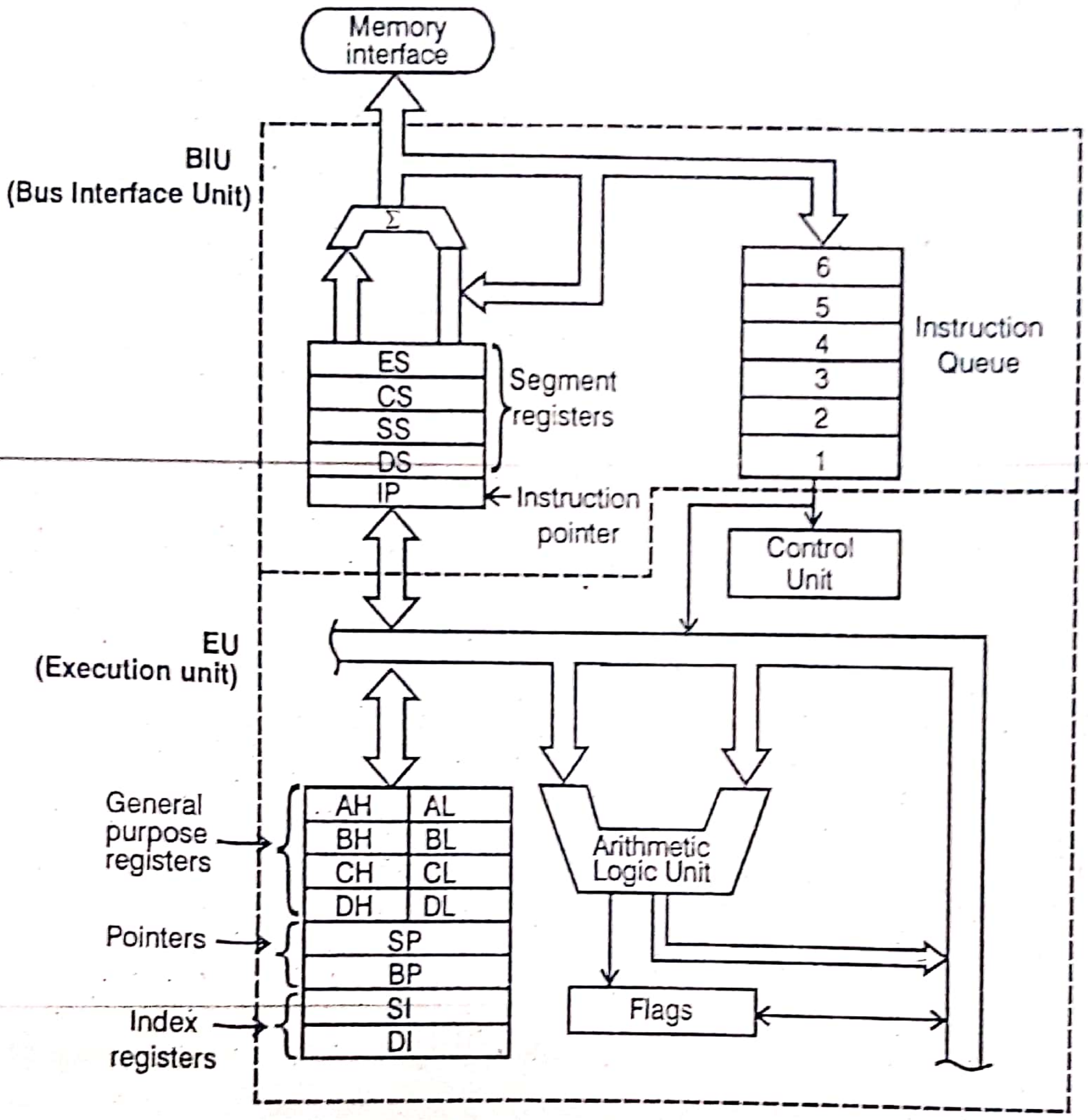


Fig. 1.1. Architecture of 8086

1. Bus Interface Unit (BIU)
2. Execution Unit (EU)

The BIU and EU function independently. The BIU interfaces the 8086 to the outside world. The BIU fetches instructions, reads data from memory and ports, and writes data to memory and I/O ports.

EU receives program instruction codes and data from the BIU, executes these instructions and stores the results either in the general registers or output them through the BIU. EU has no connections to the system buses. It receives and outputs all its data through the BIU.

The BIU contains

1. **Segment registers**
2. **Instruction pointer**
3. **Instruction queue**

The EU contains

1. ALU
2. General purpose registers
3. Index registers
4. Pointers
5. Flag register

1.2.1 General Purpose Registers

All general registers of the 8086 microprocessor can be used for arithmetic and logic operations. The 16 bit general registers are:

1. Accumulator register (AX)
2. Base register (BX)
3. Count register (CX)
4. Data register (DX)

(i) Accumulator register

It consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL contains the low-order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations and string manipulation.

(ii) Base register

It consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.

(iii) Count register

It consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. When combined, CL register contains the low-order byte of the word, and CH contains the high-order byte. Count register can be used as a counter in string manipulation and shift/rotate instructions.

(iv) Data register

It consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX. When combined, DL register contains the low-order byte of the word, and DH contains the high-order byte. Data register can be used as a port number in I/O operations. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

1.2.2 Segment Registers

Most of the registers contain data/instruction offsets within 64 KB memory segment. There are four different 64 KB segments for instructions, stack, data and extra data. The segment registers are:

1. Code segment (CS)
2. Stack segment (SS)
3. Data segment (DS)
4. Extra segment (ES)

(i) Code segment (CS)

It is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS register for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during FAR JUMP, FAR CALL and FAR RET instructions.

(ii) Stack segment (SS)

It is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by ~~the stack pointer~~ (SP) and base pointer (BP) registers are located in the stack segment. SS register can be changed directly using POP instruction.

(iii) Data segment (DS)

It is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions.

(iv) Extra segment (ES)

It is a 16-bit register containing address of 64KB segment usually with program data. By default, the processor assumes that the DI register ~~references the ES segment~~ in string manipulation instructions. ES register can be changed directly using POP and LES instructions.

It is possible to change default segments used by general and index registers by prefixing instructions with a CS, SS, DS or ES prefix.

1.2.3 Pointer Registers

(i) Stack Pointer (SP)

It is a 16-bit register pointing to program stack.

(ii) Base Pointer (BP)

It is a 16-bit register pointing to data in the stack segment. BP register is usually used for based, based indexed or register indirect addressing.

1.2.4 Index Registers

(i) Source Index (SI)

It is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data address in string manipulation instructions.

(ii) Destination Index (DI)

It is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.

1.2.5 Instruction Pointer (IP)

It is a 16-bit register. The operation is same as the program counter. The IP register is updated by the BIU to point to the address of the next instruction. Programs do not have direct access to the IP, but during execution of a program the IP can be modified or saved and restored from the stack.

1.2.6 Flag register

It is a 16-bit register containing nine 1-bit flags:

Six status or condition flags (OF, SF, ZF, AF, PF, CF)

Three control flags (TF, DF, IF)

- **Overflow Flag** (OF) - set if the result is too large positive number, or is too small negative number to fit into destination operand.
- **Sign Flag** (SF) - set if the most significant bit of the result is set.
- **Zero Flag** (ZF) - set if the result is zero.
- **Auxiliary carry Flag** (AF) - set if there was a carry from or borrow to bits 0-3 in the AL register.
- **Parity Flag** (PF) - set if parity (the number of "1" bits) in the low-order byte of the result is even.
- **Carry Flag** (CF) - set if there was a carry from or borrow to the most significant bit during last result calculation.
- **Trap or Single-step Flag** (TF) - if set then single-step interrupt will occur after the next instruction.
- **Direction Flag** (DF) - if set then string manipulation instructions will auto-decrement index registers. If cleared then the index registers will be auto-incremented.
- **Interrupt-enable Flag** (IF) - setting this bit enables maskable interrupts.

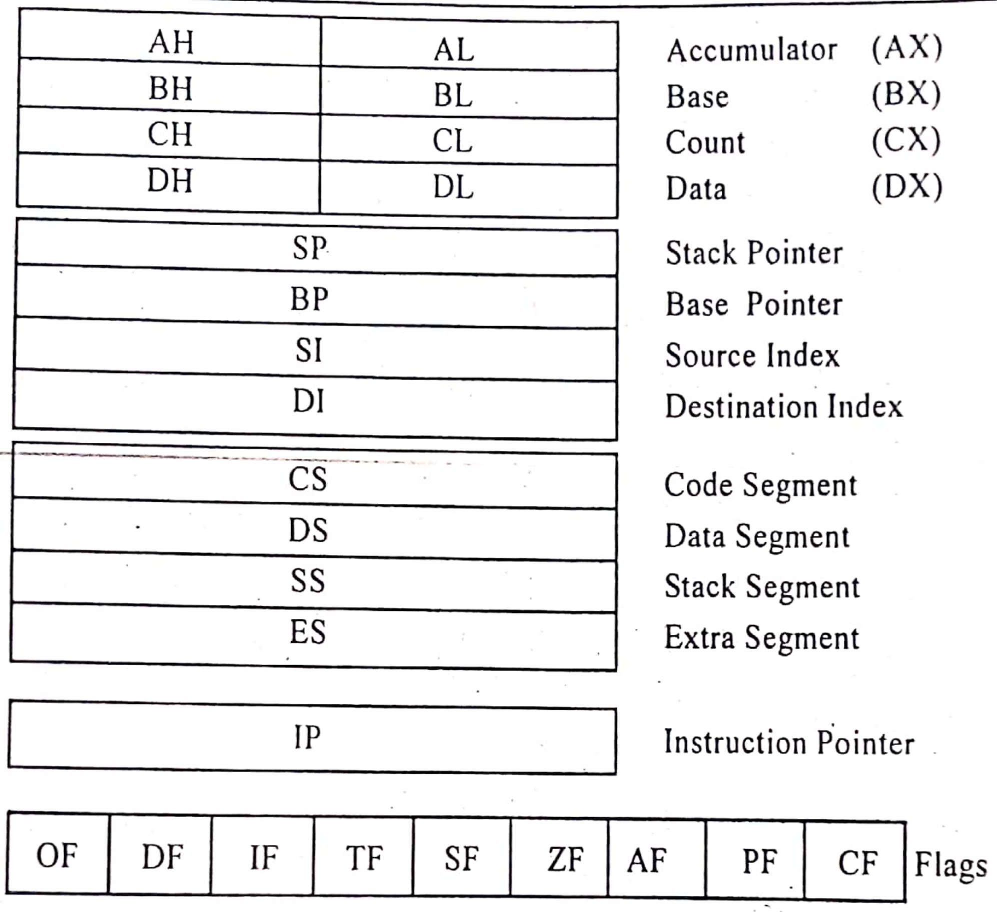


Fig 1.2. Internal Registers in 8086

1.2.7 Instruction Queue

- The instruction queue is a First-In-First-out (FIFO) group of registers where 6 bytes of instruction code is pre-fetched from memory ahead of time. It is being done to speed-up program execution by overlapping instruction fetch and execution. This mechanism is known as **PIPELINING.**

- If the queue is full, the BIU does not perform any bus cycle. If the BIU is not full and can store atleast 2 bytes and EU does not request it to access memory, the BIU may pre-fetch instructions.

- If the BIU is interrupted by the EU for memory access while pre-fetching, the BIU first completes fetching and then services the EU. In case of JMP instruction, the BIU will reset the queue and begin refilling after passing the new instruction to the EU.

1.2.8 ALU

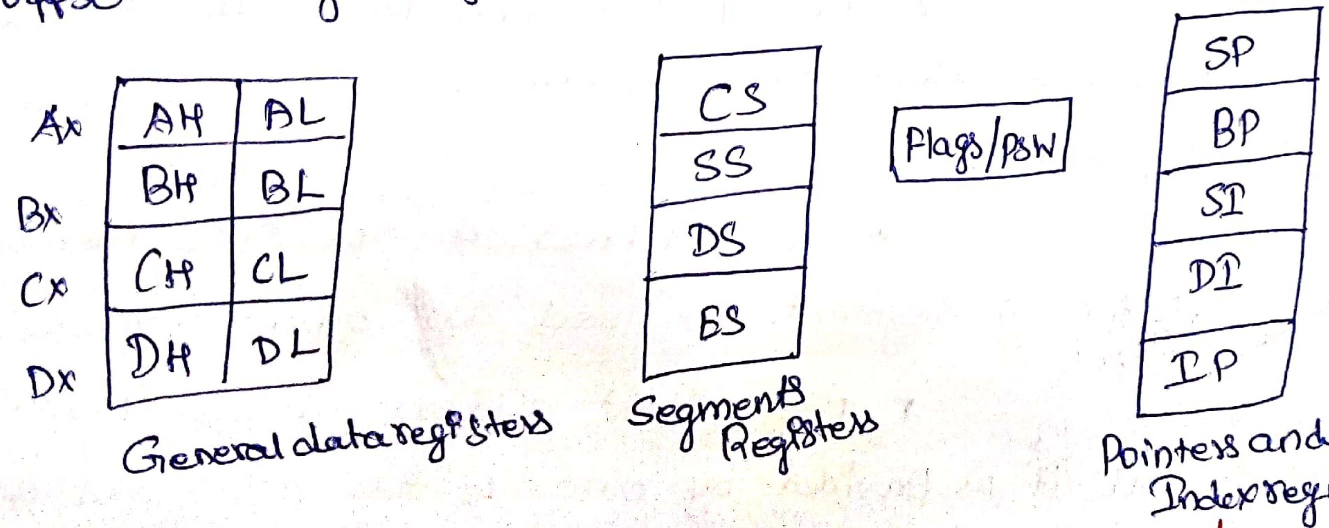
It is a 16 bit register. It can add, subtract, increment, decrement, complement, shift numbers and performs AND, OR, XOR operations.

1.2.9 Control unit

The control unit in the EU directs the internal operations like $\overline{RD}$, $\overline{WR}$, M/$\overline{IO}$

REGISTER ORGANISATION OF 8086

* 8086 has a powerful set of registers known as General Purpose and Special Purpose registers.
* All of them are 16 bit them are 16-bit registers.
* The general purpose registers can be used as either 8 bit registers (or) 16 bit registers.
* They may be either used for holding data, variables and intermediate results temporarily or for other purposes like a counter for storing offset address for some particular addressing modes etc.
* The special purpose registers are used as segment registers, pointers, index registers or as offset storage registers for particular addressing modes.



Refer the Explanation of each Registers in 8086 Architecture

MEMORY SEGMENTATION

* Two Types of memory organisations are Commonly used.

These are i) ~~linear~~ addressing
ii) Segmented addressing

* In linear addressing the entire memory space is available to the Processor in one linear array.

* In the Segmented addressing on the other hand, the available memory space is divided into "Chunks" Called Segments.

* Each segment is 64K bytes in size and addressed by one of the segment registers.

* In 8086 system the available memory Space is 1 Mbytes.

* The 16 bit contents of the segment register gives the starting base address of a Particular Segment.

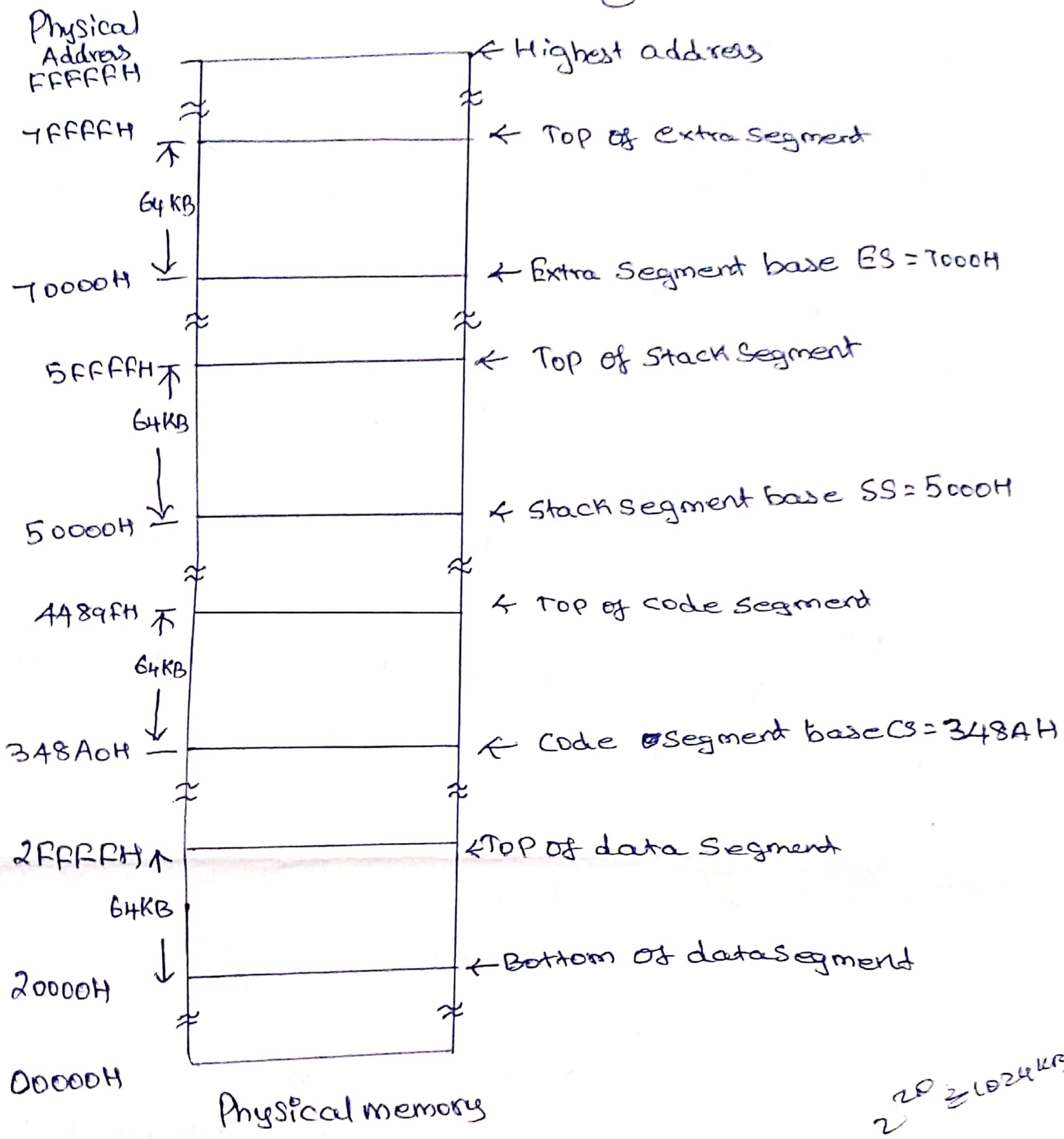
* To address a specific memory location within a Segment we need an offset address.

* The offset address is also 16 bit wide and it is Provided by one of the associated Pointer or Index register.

RULES FOR MEMORY SEGMENTATION

1. The Four segments Can overlap for Small Programs. In a minimum Systems all four segments Can Start the address 00000H

2. The segment can begin/start at any memory address which is divisible by 16.



$2^{20} = 1024KB$

MEMORY SEGMENTATION

ADVANTAGES OF MEMORY SEGMENTATION

1. It Allows the memory addressing capacity to be 1Mbyte even though the address associated with individual instruction is only 16 bit.

2. It allows Instruction code, data, stack and portion of program to be more than 64KB long by using more than one code, data, stack segment and extra segment.

*3. It facilitates use of seperate memory areas for program, data and stack.

4. It permits a program or its data to be put in different areas of memory, each time the program is executed.

2.1 8086 SIGNALS

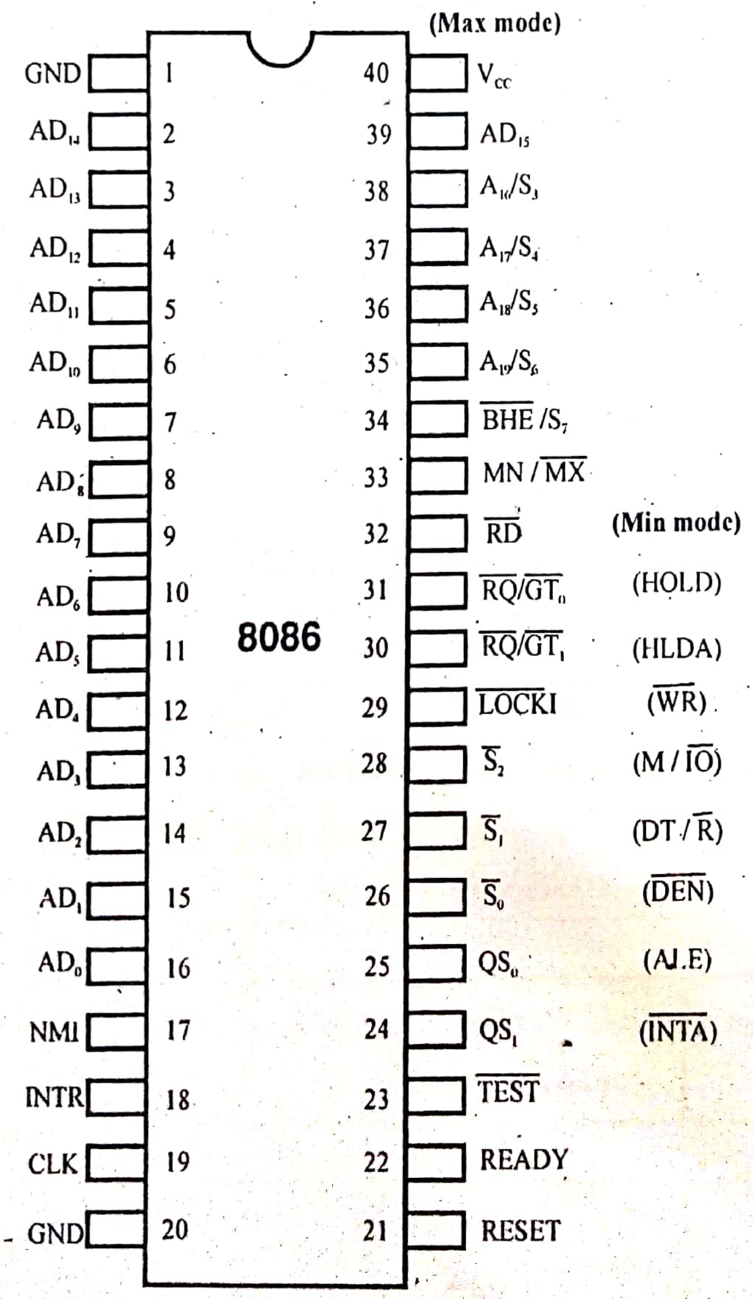


Fig.2.1. Pin Diagram of 8086

A 40 pin DIP 8086 microprocessor is shown in Fig.2.1. 8086 microprocessor can operate in two modes: Minimum mode and Maximum mode. The pins 24 to 31 have alternate functions for every mode.

Minimum mode

MN/$\overline{MX}$ pin is connected to +5V. Used in small systems including only one CPU.

Maximum mode

MN/$\overline{MX}$ pin is connected to ground. Used in large systems and systems with more than one processor.

Minimum Mode Signals:

Address/data/status		
AD_{15}-AD_0	Address/data bus	Bidirectional, 3-state
A_{19}/S_6-A_{16}/S_3	Address/status bus	output,3-state
$\overline{RD}$	Read from memory/IO	output,3-state
READY	Ready signal	input
M/$\overline{IO}$	Select memory or IO	output,3-state
$\overline{WR}$	Write to memory/IO	output,3-state
ALE	Address latch enable	output
DT/$\overline{R}$	Data transmit/receive	output
$\overline{DEN}$	Data bus enable	output
$\overline{BHE}/S_7$	Bus high enable	output
INTR	Interrupt request	input
NMI	Non-maskable interrupt	input
RESET	Reset	input
$\overline{INTA}$	Interrupt acknowledge	output
HOLD	Hold request	input
HLDA	Hold acknowledge	output
$\overline{TEST}$	Test pin tested by WAIT instruction	input
MN/$\overline{MX}$	Minimum/maximum mode, 5V	input
CLK	Clock pin for basic timing signal	input
V_{cc}	Power supply, +5 V	
GND	Ground connection, 0V	

Maximum Mode Signals:

Address/data/status		
AD_{15}-AD_0 A_{19}/S_6-A_{16}/S_3	Address/data bus Address/status bus	Bidirectional, 3-state output,3-state
$\overline{RD}$ READY $\overline{BHE}/S_7$ $\overline{S_2}, \overline{S_1}, \overline{S_0}$	Read from memory/IO Ready signal Bus high enable Status/handshake bits indicating the function of the current bus cycle	output,3-state input output output
INTR NMI RESET	Interrupt request Non-maskable interrupt Reset	input input input
$\overline{RQ}/\overline{GT_1}$, $\overline{RQ}/\overline{GT_0}$ $\overline{LOCK}$	Request/grant pins for bus access Used to lock the bus, activated by LOCK prefix on any instruction	bidirectional output
QS_1,QS_0 $\overline{TEST}$ MN/$\overline{MX}$ CLK V_{CC} GND	Queue status Test pin tested by WAIT instruction Minimum/maximum mode, 0V Clock pin for basic timing signal Power supply, +5 V Ground connection, 0V	output input input input

2.1.1 Address / Data Bus (AD_{15}–AD_0)

The multiplexed Address/ Data bus acts as address bus during the first part of machine cycle (T1) and data bus for the remaining part of the machine cycle.

2.1.2 Address/Status (A_{19}/S_6, A_{18}/S_5, A_{17}/S_4, A_{16}/S_3)

During T1 these are the four most significant address lines for memory operations. During I/O operations these lines are LOW. During memory and I/O operations, status information is available on these lines during T2, T3, T_{WAIT}, T4. The status of the interrupt enable FLAG bit (S_5) is updated at the beginning of each CLK cycle. Function of status bits S_3 and S_4 as shown below:

S_4	S_3	Function
0	0	ES, Extra segment
0	1	SS, Stack Segment
1	0	CS, Code segment
1	1	DS, Data segment

2.1.3 Bus High Enable/Status ($\overline{BHE}/S_7$)

During T1 the bus high enable signal ($\overline{BHE}$) should be used to enable data onto the most significant half of the data bus, pins $D_{15} \pm D_8$.

$\overline{BHE}$ is LOW during T1 for read, write, and interrupt acknowledge cycles when a byte is to be transferred on the high portion of the bus. The S_7 status information is available during T2, T3, and T4.

$\overline{BHE}$	A_0	Characteristics
0	0	Whole word
0	1	Upper byte from/to odd address
1	0	Lower byte from/to even address
1	1	None

2.1.4 Read ($\overline{RD}$)

This signal is used to read data from memory or I/O device which reside on the 8086 local bus.

2.1.5 Ready

If this signal is low the 8086 enters into WAIT state. The READY signal from memory/IO is synchronized by the 8284A clock generator to form READY. This signal is active HIGH.

2.1.6 Interrupt Request (INTR)

It is a level triggered maskable interrupt request. A subroutine is vectored via an interrupt vector lookup table located in system memory. It can be internally masked by software resetting the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.

2.1.7 $\overline{TEST}$

This input is examined by the "Wait" instruction. If the TEST input is LOW execution continues, otherwise the processor waits in an "Idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.

2.1.8 Non-Maskable Interrupt (NMI)

It is an edge triggered input which causes a type 2 interrupt. NMI is not maskable internally by software. A transition from LOW to HIGH initiates the interrupt at the end of the current instruction.

2.1.9 Reset

This signal is used to reset the 8086. It causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. It restarts execution when RESET returns LOW.

2.1.10 Clock (CLK)

This signal provides the basic timing for the processor and bus controller. The clock frequency may be 5 MHz or 8 MHz or 10 MHz depending on the version of 8086.

2.1.11 V_{CC}

It is a +5V power supply pin.

2.1.12 Ground (GND)

Two pins (1 and 20) are connected to ground ie, 0 V power supply.

2.1.13 Minimum/Maximum (MN/$\overline{MX}$)

This pin indicates what mode the processor is to operate in. The 8086 can be configured in either minimum mode or maximum mode using this pin.

2.1.14 Minimum Mode Signals

MEMORY / IO (M/$\overline{IO}$)

It is used to distinguish a memory access from an I/O access. M = HIGH, I/O = LOW.

WRITE($\overline{WR}$)

It indicates that the processor is performing a write memory or write I/O cycle, depending on the state of the M/$\overline{IO}$ signal.

Interrupt Acknowledge ($\overline{INTA}$)

This signal indicates recognition of an interrupt request. It is used as a read strobe for interrupt acknowledge cycles.

Address Latch Enable (ALE)

This signal is used to demultiplex the AD_0-AD_{15} into A_0-A_{15} and D_0-D_{15}. It is a HIGH pulse active during T1 of any bus cycle.

Data Transmit/Receive (DT/$\overline{R}$)

This signal desires to use a data bus transceiver (8286/8287). It is used to control the direction of data flow through the transceiver. A high signal on this pin indicates that 8086 is transmitting the data and low indicates that 8086 is receiving the data.

Data Enable($\overline{DEN}$)

This signal informs the transceivers (8286/8287) that the 8086 is ready to send or receive data.

Hold

This signal indicates that another master (DMA or processor) is requesting the host 8086 to handover the system bus.

Hold Acknowledge (HLDA)

On receiving HOLD signal 8086 outputs HLDA signal HIGH as an acknowledgement.

2.1.15 Maximum Mode Signals

Maximum mode operation differs from minimum mode in that some of the control signals must be externally generated. This requires additional circuitry. however, a chip -the 8288 bus controller- designed for this purpose is available.

Status ($\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$)

These three status signals indicate the type of machine cycle used. These status lines are encoded as shown below:

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Machine cycle
0	0	0	Interrupt acknowledge
0	0	1	I/O read
0	1	0	I/O write
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Passive

Request/Grant ($\overline{RQ}/\overline{GT_0}$, $\overline{RQ}/\overline{GT_1}$)

These pins are used by other local bus masters to force $\overline{RQ}/\overline{GT_1}$ the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with $\overline{RQ}/\overline{GT_0}$ having higher priority than $\overline{RQ}/\overline{GT_1}$.

$\overline{\text{LOCK}}$

This signal indicates that other system bus masters are not to gain control of the system bus while LOCK is active LOW. The LOCK signal is activated by the "LOCK" prefix instruction and remains active until the completion of the next instruction. This signal is active LOW.

Queue Status (QS_1, QS_0)

The queue status is valid during the CLK cycle after which the queue operation is performed. QS1 and QS0 provide status to allow external tracking of the internal 8086 instruction queue.

QS_1	QS_0	Characteristics
0	0	No operation
0	1	First byte of opcode from Queue
1	0	Empty the Queue
1	1	Subsequent byte from Queue

PHYSICAL MEMORY ORGANISATION

* In an 8086 based system, the 1M bytes memory is physically organised as an odd bank and an even bank, each of 512Kbytes, addressed in parallel by the processor.

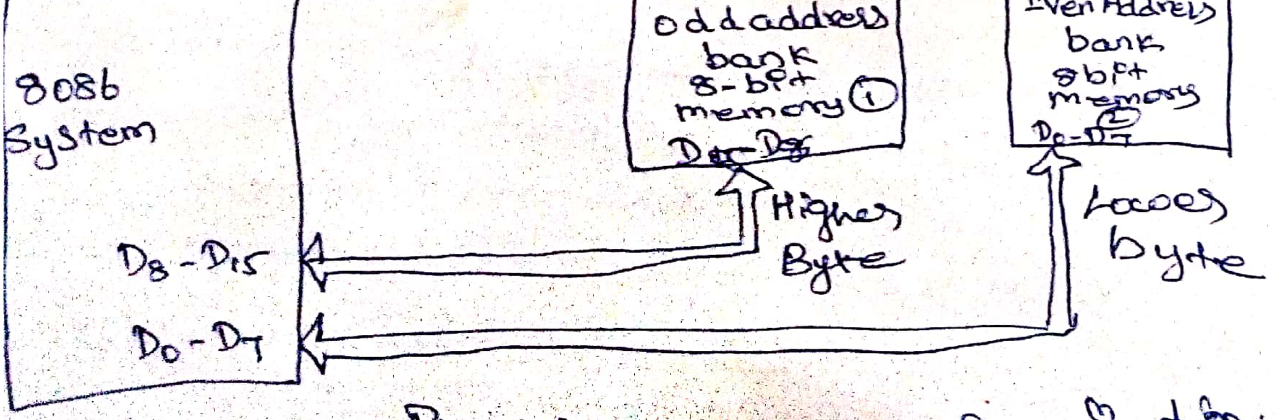
* Byte data with an even address is transferred on $D_7 - D_0$, while the byte data with an odd address is transferred on $D_{15} - D_8$ bus lines

* The processor provides two enable signals, $\overline{BHE}$ and A_0 for selection of either even or odd or both the banks.

* The instruction stream is fetched from memory as words and is addressed internally by the processor as necessary.

* In other words, if the processor fetches ~~internally~~ from memory, a word and is ~~addressed~~ there are different possibilities, like:

1. Both the bytes may be operands
2. Both the bytes may contain opcode bits.
3. one of the bytes may be opcode while the other may be data



Physical memory organisation.

* All the above Possibilities are taken care of by the Internal Decoder circuit of the microprocessor.

* The opcodes and operands are identified by the internal decoder circuit which further Derives the signals those act as input to the timing & control unit.

* The timing & control unit then derives all the signals required for execution of the instruction.

* While referring to word data, the BIU requires one or two memory cycles, depending upon whether the starting byte is located at an even or odd address.

* It is always better to locate at an even address, only one read or write cycle is required

* If the word is located at an odd address, The first read or write cycle is required to ~~be~~ accessing the lower byte while the second one is required for accessing the upper byte.

* Thus Two bus cycles are required, if a word is located at an odd address. It should be kept in mind that while initialising the structures. like Stack they should be initialised at an even address for efficient operation.

* 8086 is a 16 bit microprocessor and hence can access two bytes of data in one memory or I/o read or write operation.

* But the commercially available memory chips are only byte size, i.e. the can store only one byte in a memory location.

* To store 16bit data, two successive memory locations are used and the lowes byte of 16 bit data can be stored in first memory location. While the seconde byte is stored in next location.

* In a sixteen bit read or write operation both of these bytes will be read or written in a single machine cycle.

* A map of an 8086 memory system starts at 00000H and ends at FFFFFH. 8086 being a 16bit processor is expected to access 16 bit data to / from 8bit commercially available memory chips in parallel.

Thus bits $D_0 - D_7$ of a 16bitdata will be transferred over $D_0 - D_7$ (lower byte) of a 16 bit data bus to/ from 8bit memory (2) and bit $D_8 - D_{15}$ of the 16 bit data will be transferred over $D_8 - D_{15}$ (higher byte) of the 16 bit data bus of the MP. to /from 8 bit memory (1). Thus to acheive 16 bit data transfer using 8 bit memories, in Parallel, the map of the complete system byte memoryaddresses will obviously be divided into twomemory banks.

The lower byte of a 16bit data is stored at the first address of the map 00000H and it is to be transferred over $D_0 - D_7$ of the microprocessor bus so 00000H must be in 8 bit memory (2)

Higher byte of the 16 bit data is stored in the next address 00001H; it is to be transferred over $D_8 - D_{15}$ of the microprocessor bus so the address 00001H must be in 8bit memory (1)

* On similar lines for the next 16 bit data is stored in the memory, immediately after the previousone, the lower byte will be stored at the next address 00002H and it must be in 8 bit memory (2).

* while the higher byte will be stored at the next address 00003H that must be in 8 bit memory (1)

Thus if it is imagined that the complete memory map of 8086 is filled with 16 bit data, all the lower bytes $(D_0 - D_7)$ will be stored the 8 bit memory bank(2) and all the higher bytes $(D_8 - D_{15})$ will be stored in the 8 bit memory bank (1)

* consequentally it can be observed that all the lower bytes have to be stored at even address and all the higher byte have to be stored in odd address.

* Thus 8 bit memory bank (1) will be called an odd address bank and the 8 bit memory bank (2) will be called even address bank.

* The complete memory map of 8086 system is thus divided into even and odd address memory banks.

* The 8086 transfera 16 bit data.

* Two signals A_0 and $\overline{BHE}$ solve the problem of selection of appropriate memory banks.

* Certain locations in memory are reserved for specific CPU operations. The locations from FFFF0H to FFFFFH are reserved for operations including Jump to initialisation programme and I/O Processor initialization.

* The locations 00000H to 003FFH are reserved for interrupt vector table. The interrupt structure provides space for a total of 256 interrupt vectors.

* The vectors, i.e CS and IP for each interrupt routine requires 4 bytes for storing it in the interrupt vector table.

* Hence 256 types of Interrupt require 256×4 =03FFFH (1kbyte) locations for the Complete interrupt vector table.

Interrupt

→ is an signal → from a hardware
→ from a Program

→ Software interrupt also may called Trap

→ Interrupt service routine

→ Programmess using Interrupt to support multitask

Performance overhead

* cost of storing & restoring Process states
* Lost of flusing the interrupt pipeline
* Restoring the instruction into the pipeline when the Process is restarted.

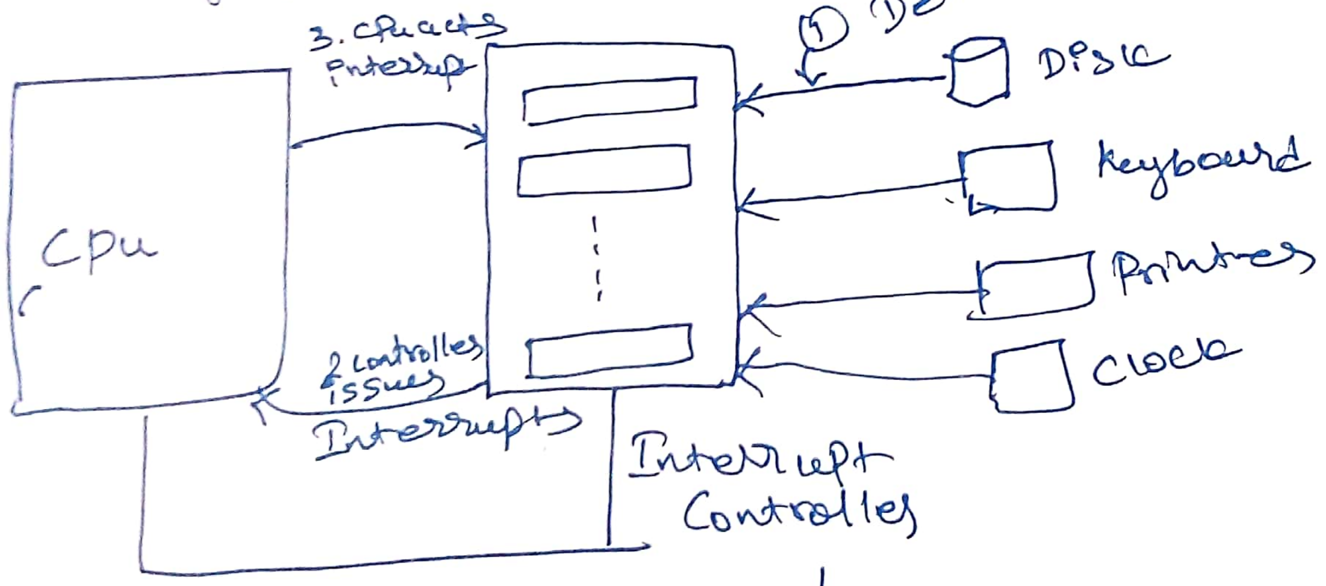
RS2
14, 16, 29,

Input/output

Interrupts

* CPU Interrupt request line triggered by I/O device

* Interrupt handler receives Interrupts.
* → Determines the best course of action
  * Find out the source of Interrupts generation
  * Analysis its status
  * Restarts it when appropriate with the next operation
  * Returns control to the interrupted devices
  * Device is finished



Interrupt Handler → Hardware

↓

Software

Contd

* maskable to ignore or delay some Interrupts
* Interrupt vector to dispatch interrupt to correct handler, based on Priority
* → Some non maskable
* Interrupt mechanism also used for exception

INSTRUCTION SET AND PROGRAMMING WITH 8086

Instruction Formats -Addressing Modes-Instruction Set, Assembler Directives-Macros, Programs Involving Logical, Branch Instructions – Sorting and Evaluating Arithmetic Expressions – String Manipulations-Simple ALPs.

Assembler Instruction Format

The general format of an assembler instruction is

Label : Mnemonic Operand, Operand ; Comments

Label:

- A label is an identifier that is assigned to the address of the first byte of instruction in which it appears.
- An instruction may or may not have label, it provide a symbolic name which is used in branch instruction to branch to the instruction.

Mnemonic:

- Mnemonic must present in all instruction. It defines the type of operation such as ADD,SUB MUL etc.

Operand:

- It may or may not present, depends on the type of instruction.
- One operand may appear. If we use two operands used, then we need to use a comma to separate it.
- If we use two operand, destination operand must appear first and source operand must appear second.

Comments:

- After semicolon we can use whatever we want to write.
- It is optional.

ADDRESSING MODES :

The different ways in which a source operand is denoted in an instruction are known as the addressing modes. There are 8 different addressing modes in 8086 programming. They are

- 1. Immediate addressing mode
- 2. Register addressing mode
- 3. Direct addressing mode
- 4. Register indirect addressing mode
- 5. Based addressing mode
- 6. Indexed addressing mode.
- 7. Based indexed addressing mode
- 8. Based, Indexed with displacement.

  Immediate addressing mode: The addressing mode in which the data operand is a part of the instruction itself is called Immediate addressing mode.

  For Ex: MOV CX, 4847 H
  ADD AX, 2456 H
  MOV AL, FFH

- **Register addressing mode :** Register addressing mode means, a register is the source of an operand for an instruction.

  For Ex : MOV AX, BX copies the contents of the 16-bit BX register into the 16-bit AX register.
  EX : ADD CX,DX

- **Direct addressing mode:** The addressing mode in which the effective address of the memory location at which the data operand is stored is given in the instruction.i.e the effective address is just a 16-bit number is written directly in the instruction.

  For Ex: MOV BX, [1354H]
  MOV BL,[0400H]

  . The square brackets around the 1354 H denotes the contents of the memory location. When executed, this instruction will copy the contents of the memory location into BX register. This addressing mode is called direct because the displacement of the operand from the segment base is specified directly in the instruction.

- **Register indirect addressing mode**: Register indirect addressing allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI and SI.

- Ex: MOV AX, [BX]. Suppose the register BX contains 4675H ,the contents of the 4675 H are moved to AX.
  ADD CX,{BX}

- **Based addressing mode**: The offset address of the operand is given by the sum of contents of the BX or BP registers and an 8-bit or 16-bit displacement.
- Ex: MOV DX, [BX+04]
  ADD CL,[BX+08]

- **Indexed Addressing mode**: The operands offset address is found by adding the contents of SI or DI register and 8-bit or 16-bit displacements.
- Ex: MOV BX,[SI+06]
  ADD AL,[DI+08]

- **Based -index addressing mode*:*** The offset address of the operand is computed by summing the base register to the contents of an Index register.

  Ex: ADD CX,[BX+SI]

  MOV AX,[BX+DI]

- **Based Indexed with displacement mode:** The operands offset is computed by adding the base register contents, an Index registers contents and 8 or 16-bit displacement**.**
  Ex : MOV AX,[BX+DI+08]
  ADD CX,[BX+SI+16]

INSTRUCTION SET OF 8086

Operand types:

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.

SREG: DS, ES, SS, and only as second operand: CS.

The 8086 microprocessor supports 6 types of Instructions. They are

1. Data transfer instructions
2. Arithmetic instructions
3. Bit manipulation instructions
4. String instructions
5. Program Execution Transfer instructions (Branch & loop Instructions)
6. Processor control instructions

1. Data Transfer instructions : These instructions are used to transfer the data from source operand to destination operand. All the store, move, load, exchange ,input and output instructions belong to to this group.

General purpose byte or word transfer instructions:

MOV **:** Copy byte or word from specified source to specified destination

PUSH **:** Push the specified word to top of the stack

POP **:** Pop the word from top of the stack to the specified location

PUSHA **:** Push all registers to the stack

POPA **:** Pop the words from stack to all registers

XCHG **:** Exchange the contents of the specified source and destination operands one of which may be a register or memory location.

XLAT : Translate a byte in AL using a table in memory

Simple input and output port transfer instructions

1. IN : Reads a byte or word from specified port to the accumulator
2. OUT : Sends out a byte or word from accumulator to a specified port

Special address transfer instructions

1. LEA : Load effective address of operand into specified register
2. LDS : Load DS register and other specified register from memory
3. LES : Load ES register and other specified register from memory.

Flag transfer registers

1. LAHF : Load AH with the low byte of the flag register
2. SAHF : Store AH register to low byte of flag register
3. PUSHF : Copy flag register to top of the stack
4. POPF : Copy word at top of the stack to flag register

2. Arithmetic instructions : These instructions are used to perform various mathematical operations like addition, subtraction, multiplication and division etc….

Addition instructions

1.ADD **:** Add specified byte to byte or word to word

2.ADC **:** Add with carry

3.INC **:** Increment specified byte or specified word by 1
4.AAA **:** ASCII adjust after addition

5.DAA **:** Decimal (BCD) adjust after addition

Subtraction instructions

1. SUB : Subtract byte from byte or word from word
2. SBB : Subtract with borrow
3. DEC : Decrement specified byte or word by 1
4. NEG : Negate or invert each bit of a specified byte or word and add 1(2's complement)
5. CMP : Compare two specified byte or two specified words
6. AAS : ASCII adjust after subtraction
7. DAS : Decimal adjust after subtraction

Multiplication instructions

1. MUL **:** Multiply unsigned byte by byte or unsigned word or word.
2. IMUL **:** Multiply signed bye by byte or signed word by word
3. AAM **:** ASCII adjust after multiplication

Division instructions

1. DIV : Divide unsigned word by byte or unsigned double word by word
2. IDIV : Divide signed word by byte or signed double word by word
3. AAD : ASCII adjust after division
4. CBW : Fill upper byte of word with copies of sign bit of lower byte
5. CWD : Fill upper word of double word with sign bit of lower word.

3. Bit Manipulation instructions : These instructions include logical , shift and rotate instructions in which a bit of the data is involved.

Logical instructions

1. NOT :Invert each bit of a byte or word.
2. AND : ANDing each bit in a byte or word with the corresponding bit in another byte or word.
3. OR : ORing each bit in a byte or word with the corresponding bit in another byte or word.
3. XOR : Exclusive OR each bit in a byte or word with the corresponding bit in another byte or word.
4. TEST :AND operands to update flags, but don't change operands.

Shift instructions

1. SHL/SAL : Shift bits of a word or byte left, put zero(S) in LSBs.
2. SHR : Shift bits of a word or byte right, put zero(S) in MSBs.
3. SAR : Shift bits of a word or byte right, copy old MSB into new MSB.

Rotate instructions

1. ROL : Rotate bits of byte or word left, MSB to LSB and to Carry Flag [CF]
2. ROR : Rotate bits of byte or word right, LSB to MSB and to Carry Flag [CF]
3. RCR :Rotate bits of byte or word right, LSB TO CF and CF to MSB
4. RCL :Rotate bits of byte or word left, MSB TO CF and CF to LSB

4. String instructions

A string is a series of bytes or a series of words in sequential memory locations. A string often consists of ASCII character codes.

1. REP : An instruction prefix. Repeat following instruction until CX=0
2. REPE/REPZ : Repeat following instruction until CX=0 or zero flag ZF=1
3. REPNE/REPNZ : Repeat following instruction until CX=0 or zero flag ZF=1
4. MOVS/MOVSB/MOVSW: Move byte or word from one string to another
5. COMS/COMPSB/COMPSW: Compare two string bytes or two string words
6. LODS/LODSB/LODSW: Load string byte in to AL or string word into AX

5.Program Execution Transfer instructions

These instructions are similar to branching or looping instructions. These instructions include conditional & unconditional jump or loop instructions.

Unconditional transfer instructions

1. CALL : Call a procedure, save return address on stack
2. RET : Return from procedure to the main program.
3. JMP : Goto specified address to get next instruction

Conditional transfer instructions

1. JA/JNBE : Jump if above / jump if not below or equal
2. JAE/JNB : Jump if above /jump if not below
3. JBE/JNA : Jump if below or equal/ Jump if not above
4. JC : jump if carry flag CF=1
5. JE/JZ : jump if equal/jump if zero flag ZF=1
6. JG/JNLE : Jump if greater/ jump if not less than or equal
7. JGE/JNL : jump if greater than or equal/ jump if not less than
8. JL/JNGE : jump if less than/ jump if not greater than or equal
9. JLE/JNG : jump if less than or equal/ jump if not greater than
10. JNC : jump if no carry (CF=0)
11. JNE/JNZ : jump if not equal/ jump if not zero(ZF=0)
12. JNO : jump if no overflow(OF=0)
13. JNP/JPO : jump if not parity/ jump if parity odd(PF=0)
14. JNS : jump if not sign(SF=0)
15. JO : jump if overflow flag(OF=1)
16. JP/JPE : jump if parity/jump if parity even(PF=1)
17. JS : jump if sign(SF=1)

6.Iteration control instructions

These instructions are used to execute a series of instructions for certain number of times.

1. LOOP :Loop through a sequence of instructions until CX=0
2. LOOPE/LOOPZ : Loop through a sequence of instructions while ZF=1 and CX = 0
3. LOOPNE/LOOPNZ : Loop through a sequence of instructions while ZF=0 and CX =0
4. JCXZ : jump to specified address if CX=0

7. Interrupt instructions

1. INT : Interrupt program execution, call service procedure
2. INTO : Interrupt program execution if OF=1
3. IRET : Return from interrupt service procedure to main program

1. BOUND : Check if effective address within specified array bounds

8.Processor control instructions

Flag set/clear instructions

1. STC : Set carry flag CF to 1
2. CLC : Clear carry flag CF to 0
3. CMC : Complement the state of the carry flag CF
4. STD : Set direction flag DF to 1 (decrement string pointers)
5. CLD : Clear direction flag DF to 0
6. STI : Set interrupt enable flag to 1(enable INTR input)
7. CLI : Clear interrupt enable Flag to 0 (disable INTR input)

10. External Hardware synchronization instructions

1. HLT : Halt (do nothing) until interrupt or reset
2. WAIT : Wait (Do nothing) until signal on the test pin is low
3. ESC : Escape to external coprocessor such as 8087 or 8089
4. LOCK : An instruction prefix. Prevents another processor from taking the bus while the adjacent instruction executes.

ASSEMBLER DIRECTIVES :

Assembler directives are the directions to the assembler which indicate how an operand or section of the program is to be processed. These are also called pseudo operations which are not executable by the microprocessor. The various directives are explained below.

1. ASSUME : The ASSUME directive is used to inform the assembler the name of the logical segment it should use for a specified segment.

Ex: ASSUME DS: DATA tells the assembler that for any program instruction which refers to the data segment ,it should use the logical segment called DATA.

2.DB -Define byte. It is used to declare a byte variable or set aside one or more storage locations of type byte in memory.

For example, CURRENT_VALUE DB 36H tells the assembler to reserve 1 byte of memory for a variable named CURRENT_ VALUE and to put the value 36 H in that memory location when the program is loaded into RAM .

3. DW -Define word. It tells the assembler to define a variable of type word or to reserve storage locations of type word in memory.

4. **DD(define double word**) :This directive is used to declare a variable of type double word or restore memory locations which can be accessed as type double word.

5.DQ (define quadword) : This directive is used to tell the assembler to declare a variable 4 words in length or to reserve 4 words of storage in memory .

6.DT (define ten bytes): It is used to inform the assembler to define a variable which is **10** bytes in length or to reserve 10 bytes of storage in memory.

7. EQU –Equate It is used to give a name to some value or symbol**.** Every time the assembler finds the given name in the program, it will replace the name with the value or symbol we have equated with that name

8.ORG -**Originate** : The ORG statement changes the starting offset address of the data.

It allows to set the location counter to a desired value at any point in the program.For example the statement ORG 3000H tells the assembler to set the location counter to 3000H.

9 .PROC- Procedure: It is used to identify the start of a procedure. Or subroutine.

10. END- End program .This directive indicates the assembler that this is the end of the program module.The assembler ignores any statements after an END directive.

11. **ENDP**- End procedure: It indicates the end of the procedure (subroutine) to the assembler.

12.ENDS-End Segment: This directive is used with the name of the segment to indicate the end of that logical segment.

Ex: CODE SEGMENT : Start of logical segment containing code

CODE ENDS : End of the segment named CODE.

13. EVEN: • Align on even memory address. The EVEN directive tells the assembler to increment the location counter to the next even address if it is not already at an even address.

• The 8086 can read a word from memory in one bus cycle if word is at even address, two bus cycles, if word is at Odd address. A NOP instruction is inserted in the location incremented over.

Differences between Procedures and Macros

What is macro

A macro is a series of instructions that have a name that the programmer can use anywhere in the program . In addition, a macro begins with the macro directive and ends with the% endmacro directive.

The syntax of Macro is as follows.

% macro macro_name

<Macro body>

% end macro

The macro_name helps to identify the macro and the number_of_params relates to the number parameters. It is also possible to call the macro using the macro name with the required parameters. If it is necessary to execute the same instruction set several times, the programmer can therefore write these instructions into a macro and use this in his program..

What is procedure

Procedures are useful in making a large program easier to read, maintain, and change. Typically, a procedure consists of three main sections. First, the procedure name, which helps identify the procedure. Second, the instructions within the body describing the task to be performed. Finally, the return statement, which denotes the return statement.

The syntax of Macro is as follows.

proc_name:

Procedural bod

RET

MACRO

VERSUS

PROCEDURE

MACRO	PROCEDURE
Sequence of instructions that is written within the macro definition to support modular programming	Set of instructions which can be called repetitively that performs a specific task
Requires more memory	Requires less memory
Does not require CALL and RET instructions	Requires CALL and RET instructions
Machine code is generated each time the macro is called	Machine code generates only once
Parameters are passed as a part of statement which calls the macro	Parameters are passed in registers and memory locations of stack
Macro executes faster than a procedure	Procedure executes slower than a macro
Eliminates the overhead time to call the procedure and to return the program	Requires more overhead time to call the procedure and to return back to the calling procedure
Used for less than 10	Used for more than 10

Dr.K.Gopi,
Department of ECE.

UNIT-5

SYLLABUS

INTERFACING DEVICES:

8255 PPI- Block Diagram, Various Modes of Operation, Interface D/A and A/D interfacing, Keyboard Interfacing, 8259 Interrupt controller,8279 Keyboard and display controller, seven segment display,8251 serial communication protocol.

(8255- Programmable Peripheral Interface)

Def: The 8255 is a general purpose programmable I/O device designed to transfer the data from I/O to Microprocessor or Microprocessor to I/O devices. It can be used with almost any microprocessor (8 bit, 16 bit or 32 bit).It consists of 24 I/O lines which can be configured as per the requirement

Features of 8255:

1. It has 24 I/O lines with which it communicates with I/O devices.
2. The 24 I/O lines of 8255 are arranges in three ports, i.e., PORT A, PORT B, and PORT C.
3. Port A contains 8-bit I/O (PA0-PA7), Port B 8 I/O lines are represented as (PB0-PB7), Port C can be split into two parts, i.e. PORT C lower (PC0-PC3) and PORT C upper (PC7-PC4). It is also 8 bit port.
4. These three ports are further divided into two groups, i.e. Group A includes PORT A and Upper PORT C. Group B includes PORT B and Lower PORT C.
5. All these ports can be programmed as Input or Output using Control word register (CWR) of 8255.
6. Peripheral devices: Printers, keyboards, displays, floppy disk controllers, CRT controllers, machine tools, D-to-A and A-to-D converters, etc are connected to the microprocessor through he 8255A Port Pins.
7. It is also called as Parallel Communication Interface.
8. It can be operated in two basic modes:
   i. Bit Set/ResetM ode
   ii. I/O Mode
9. I/O mode is further divided into 3 modes:
   i. Simple I/O mode (Mode 0)
   ii. Strobed I/O mode (Mode l)
   iii. Bidirectional Data Transfer mode (Mode 2)

Architecture of 8255 Programmable Peripheral Interface

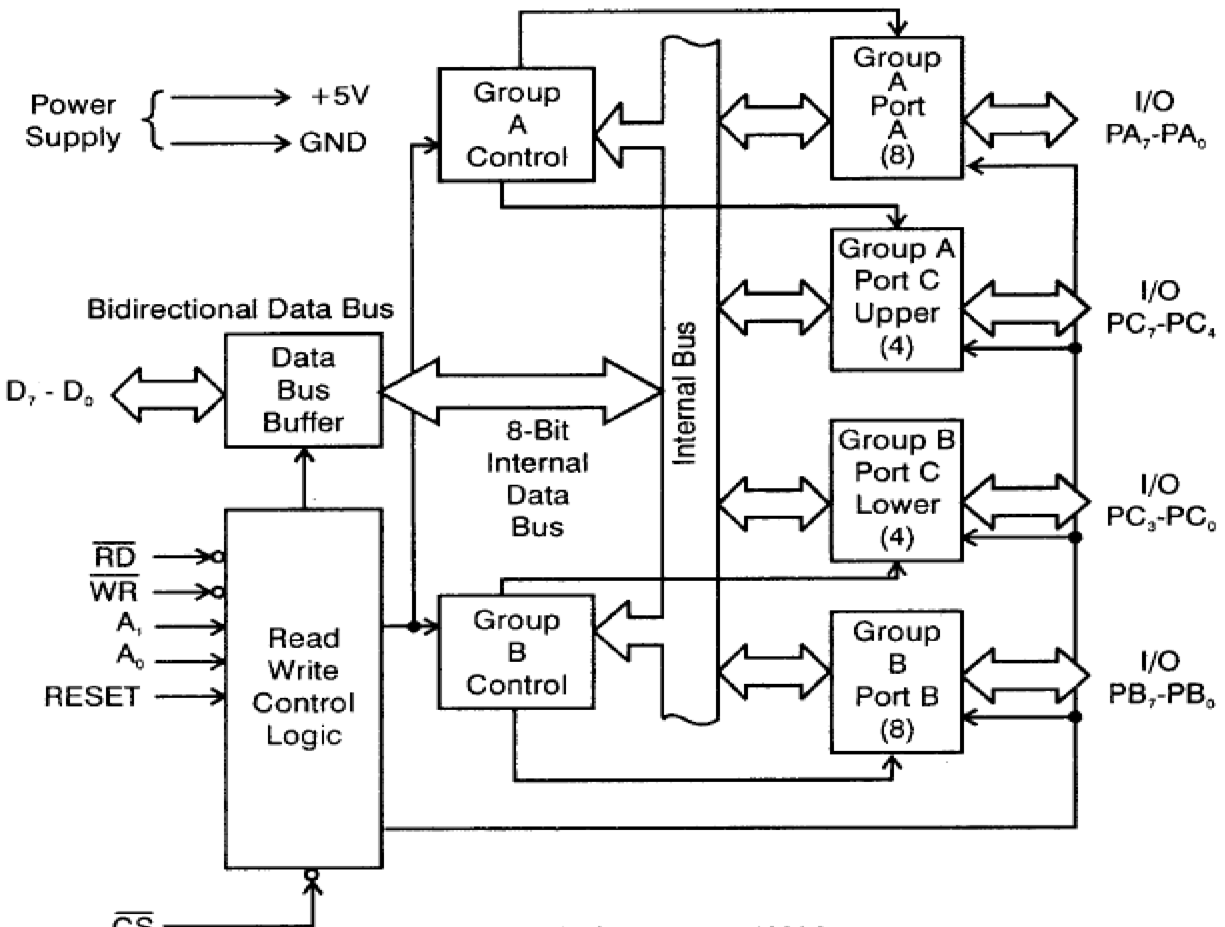


Fig. : Block Diagram of 8255A

The 8255 consists of Four sections namely

1. Data Bus Buffer
2. Read/Write Control Logic
3. Group A Control
4. Group B Control

l. Data Bus Buffer

This is a Bi-Directional Data Bus used to interface the internal data bus of 8255A to the System Data Bus of 8086. Using IN or OUT instructions, CPU can read or write the data from/to the Data Bus Buffer. It can also be used to transfer control words and status information between C PU and 8255,A..

2. Read/Write Control Logic

This block controls the Chip Selection ($\overline{CS}$), Read ($\overline{RD}$) and Write ($\overline{WR}$) operations. It consists of A_0 and A_1 signals which are generally connected to the MPU address lines A_0 and A_1 respectively. When $\overline{CS}$ (Chip Select) signal goes LOW, different values of A_0 and A_1 select one of the I/O ports or Control Register as shown in the Table 4.1 given below.

Table : 8255 Port Selection

$\overline{CS}$	A_1	A_0	Selected
0	0	0	PORT A
0	0	1	PORT B
0	1	0	PORT C
0	1	1	Control Register
1	X	X	8255A is not Selected

<u>Group A Control and Group B Control</u>

To execute peripheral data transfer, three 8 -bit ports are provided in 8255,{ i.e. ports A, B and C . For the purpose of programming 8255A these ports are grouped as follows

Group A	:	Port A and Most Significant Bits (MSB) of Port C ($PC_4 - PC_7$)
Group B	:	Port B and Least Significant Bits (LSB) of Port C ($PC_0 - PC_3$)

Port A: One 8-bit data output latch/buffer and one 8-bit input latch buffer.

Port B: One 8-bit data input/output latch/buffer.

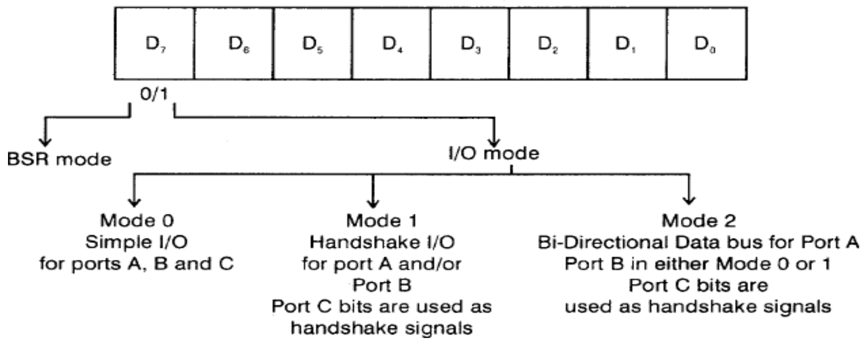
Port C: One 8-bit data output latch/buffer and one 8-bit data input buffer. This port can be divided into two 4-bit ports and it can be used for the control signal outputs and status signal inputs in conjunction with ports A and B.

<u>Operating Modes</u> :

l. Bit Set/Reset (BSR) mode

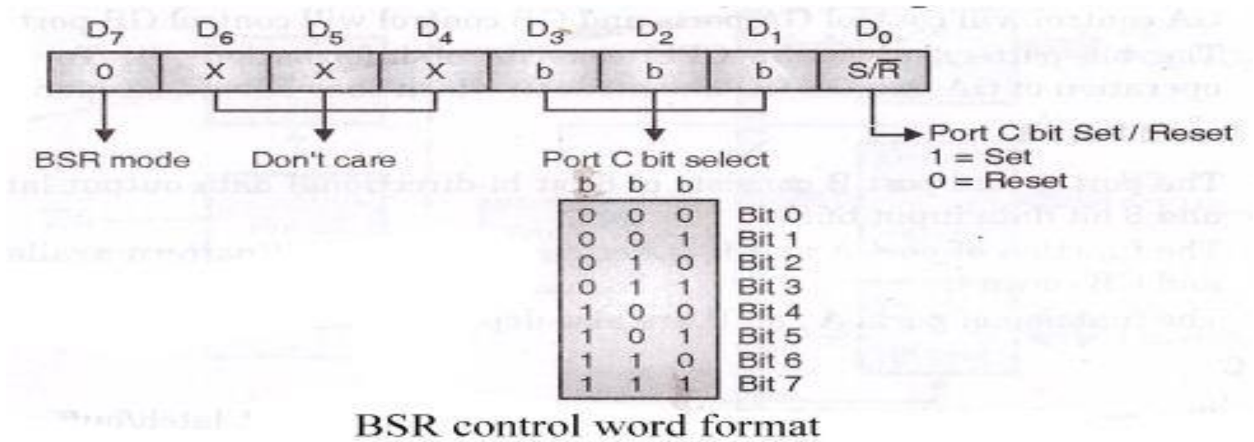
2. I/O Mode

D_7 bit of Control word decides the type of Mode. If it is "1", I/O mode is selected. If it is "0". BSR mode is selected.



Bit set/Reset Mode:

- **The Bit Set/Reset (BSR) mode is applicable to port C only.**
- **Each line of port C (PC0 - PC7) can be set/reset by suitably loading the control word register as shown in Figure 4.**
- **BSR mode and I/O mode are independent and selection of BSR mode does not affect the operation of other ports in I/O mode.**



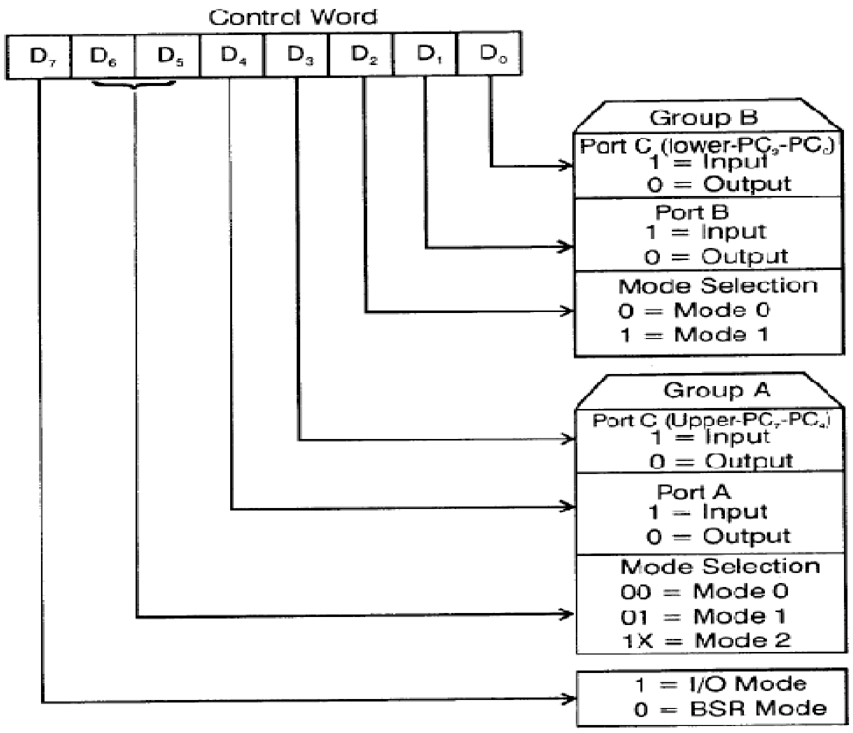
BSR control word format

- **D7 bit is always 0 for BSR mode.**
- **Bits D6, D5 and D4 are don't care bits.**
- **Bits D3, D2 and D1 are used to select the pin of Port C.**
- **Bit D0 is used to set/reset the selected pin of Port C.**
- **Selection of port C pin is determined as follows: B3 B2 B1 Bit/pin of port C set.**

I/O Mode :

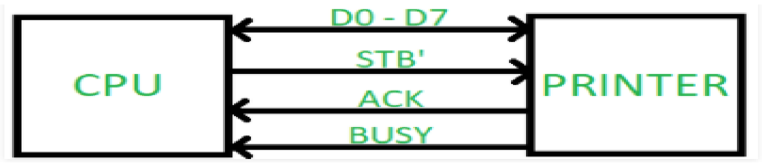
The I/O mode is divided in to three modes Mode 0, Mode I and Mode 2 given below.

1. Mode 0 - Basic I/O Mode
2. Mode I - Strobed I/O Mode
3. Mode2 - Bi-Directional Data Transfer Mode

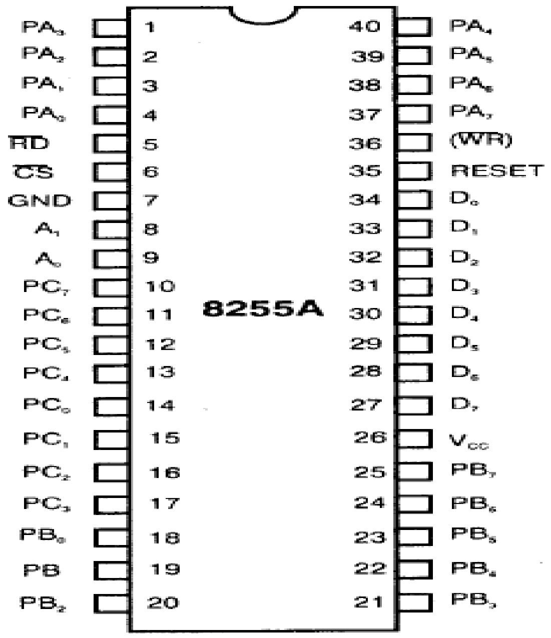


- **Mode 0 –In this mode all the three ports (port A, B, C) can work as simple input function or simple output function.**
- **Mode 1 – Handshake I/O mode or strobbed I/O mode. In this mode either port A or port B can work as simple input port or simple output port, and port C bits are used for handshake signals before actual data transmission.**

Example: A CPU wants to transfer data to a printer. In this case since speed of processor is very fast as compared to relatively slow printer, so before actual data transfer it will send handshake signals to the printer for synchronization of the speed of the CPU and the peripherals.



PIN DIAGRAM OF 8255 (Programmable Peripheral interface)



Pin Diagram of 8255A

Data Bus (D_7 _ D_0)	Bi-directional, tri state 8-bit data bus. This is used to interface 8255A to the System Data Bus.
$\overline{RD}$ (Read)	It enables the 8255A to send the data or status information to the CPU on the data bus i.e. it allows the CPU to read the data or status word from 8255A.
$\overline{WR}$ (Write)	It allows the CPU to write the data or control words in to 8255A.
A_0 and A_1	When $\overline{CS}$ = 0, these signals A_0 and A_1 are used to control the selection of ports and Control Register.
RESET	A "high" signal of this input clears the Control Register and all ports (A, B and C)
$\overline{CS}$ (Chip Select)	It enables the communication between 8255A and the CPU.
PA_0–PA_7	Port A bidirectional I/O pins
PB_0–PB_7	Port B bidirectional I/O pins
PC_0–PC_7	Port C bidirectional I/O pins

Dr.K.Gopi,
Department of ECE.

INTEL 8259 (Programmable Interrupt controller)

Introduction:

- The dictionary meaning of 'Interrupt' is to break the sequence of operation.
- While CPU (Processor) is executing a program when interrupt occurs it breaks the normal sequence of execution and diverts its execution to some other program called as Interrupt service routine (ISR) or Sub program.
- After execution of sub program the control is transferred to Once again to main program which was being executed at the time of interrupt.

The Intel 8259 is a Programmable Interrupt Controller (PIC) designed for the Intel 8085 and Intel 8086 microprocessors.

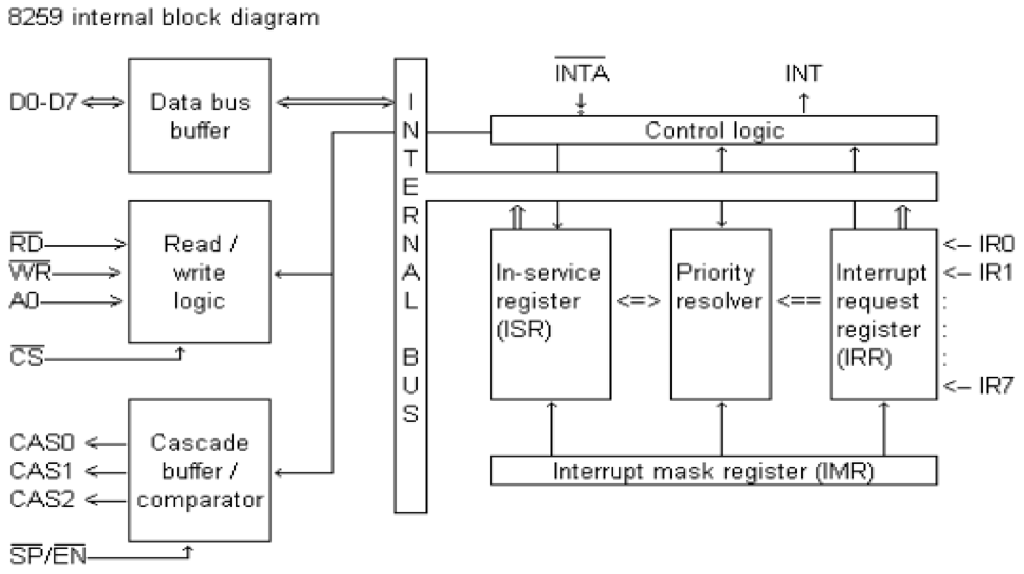
Defnition: PIC is a device which is used to increase the interrupt handling capacity of the microprocessor.

Features:

1. The Intel 8259A Programmable Interrupt Controller handles up to eight vectored priority interrupts for the CPU.
2. It is cascadable for up to 64 vectored priority interrupts without additional circuitry.
3. It is packaged in a 28-pin DIP, requires a single 5V supply.
4. The 8259 combines multiple interrupt input sources into a single interrupt output to the host microprocessor, extending the interrupt levels available in a system beyond the one or two levels found on the processor chip.
5. It accepts requests from the peripheral equipment, determines which of the incoming requests is of the highest importance (priority), checks whether the incoming request has a higher priority value than the level currently being serviced, and issues an interrupt to the CPU based on this determination .
6. It minimize the software and real-time overhead in handling multiple interrupt priorities

Block diagram of 8259 (Programmable Interrupt controller)

Internal Block Diagrma


8259 internal block diagram
D0-D7
Data bus buffer
RD
WR
A0
Read / write logic
CS
CAS0
CAS1
CAS2
Cascade buffer / comparator
SP/EN
INTERNAL BUS
INTA
INT
Control logic
In-service register (ISR)
Priority resolver
Interrupt request register (IRR)
IR0
IR1
IR7
Interrupt mask register (IMR)

Data bus buffer: It is a bidirectional data bus that interfaces 8259 bus to Microprocessor system data bus. Control word, status information pass through the data bus buffer during read or writes operation.

Read/ Write control logic: This block is responsible to accept the control words from the Microprocessor. This block also allows the status of the 8259 to be transferred on to the data bus.

- **CS (CHIP SELECT):** A LOW on this input enables the 8259A. No reading or writing of the chip will occur unless the device is selected.
- **WR (WRITE):** A LOW on this input enables the CPU to write control words (ICWs and OCWs) to the 8259A.
- **RD (READ) :** A LOW on this input enables the 8259A to send the status of the Interrupt Request Register (IRR), In Service Register (ISR), the Interrupt Mask Register (IMR), or the Interrupt level onto the Data Bus.
- **A0:** This input signal is used in conjunction with WR and RD signals to write commands into the various command registers, as well as reading the various status registers of the chip. This line can be tied directly to one of the address lines.

- **The CASCADE BUFFER/COMPARATOR:** This function block stores and compares the IDs of all 8259A"s used in the system. The associated three I/O pins (CAS0-2) are outputs when the 8259A is used as a master and are inputs when the 8259A is used as a slave. As a master, the 8259A sends the ID of the interrupting slave device onto the CAS0 – 2 lines. The slave thus selected will send its preprogrammed subroutine address onto the Data Bus during the next one or two consecutive INTA pulses.

- **Control Logic:**

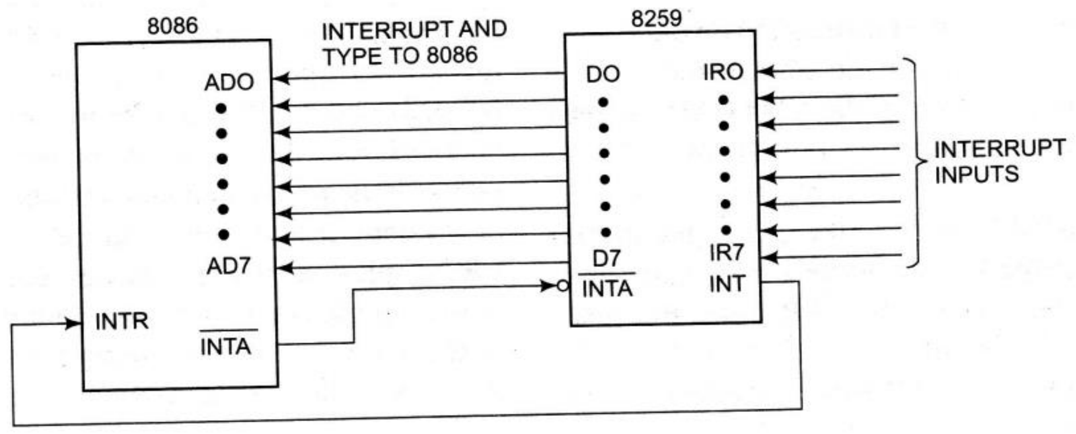
INT: This pin goes high whenever a valid interrupt request is occurred. It is used to interrupt the Microprocessor(CPU) and is connected to the interrupt input of the Microprocessor(CPU).

INTA: The processor acknowledges with the Interrupt acknowledge signal when it accepts the interrupt.

INTERRUPT REQUEST REGISTER (IRR) AND IN-SERVICE REGISTER (ISR)

- The **IRR** is used to store all the interrupt levels which are requesting service; and the **ISR** is used to store all the interrupt levels which are being serviced.
- **PRIORITY RESOLVER:** This logic block determines the priorities of the bits set in the IRR. The highest priority is selected and strobed into the corresponding bit of the ISR during INTA pulse.
- **INTERRUPT MASK REGISTER (IMR)**: The IMR stores the bits which mask the interrupt lines to be masked. The IMR operates on the IRR. Masking of a higher priority input will not affect the interrupt request lines of lower quality.

Connection of 8259A with 8086 microprocessor (Single Mode)



In the above diagram if we see the interface connections between 8086 and 8259. The 8259 can support 8 interrupts with the help of IR0 to IR7. The interrupts coming from the interfacing devices are stored in the interrupt request register and the priority resolver will decide whether fixed priority or Rotating priority is selected, and sends the interrupt signal(INT) to the microprocessor to inform that external devices wants the service. Now the 8086 will send an Acknowledgement signal which helps to serve the subroutine, once the interrupt is serviced once again the microprocessor will execute its own task.

<u>The steps are explained here as follows:</u>

<u>Interrupt sequence:</u>

1) One or more of the INTERRUPT REQUEST lines (IR_0-IR_7) are raised high, setting the, corresponding IRR bit(s).

2) The priority resolver checks three registers : The IRR for interrupt requests, the IMR for masking bits, and the ISR for the interrupt request being served. It resolves the priority and sets the INT high when it is appropriate to do so.

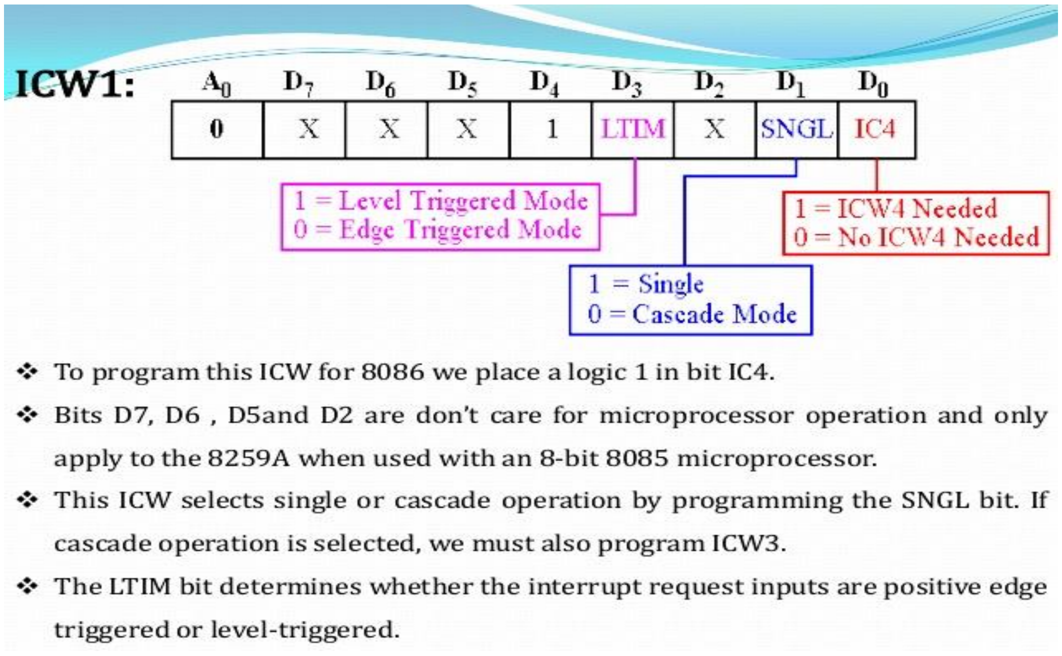
3) In response to the INTR signal, 8086 completes current instruction cycle and executes interrupt acknowledge cycle, thus giving an INTA pulse.

4) Upon receiving an INTA from the 8086, the highest priority ISR bit is set and the corresponding IRR bit is reset. Then 8259A places the opcode for CALL instruction on the data bus.

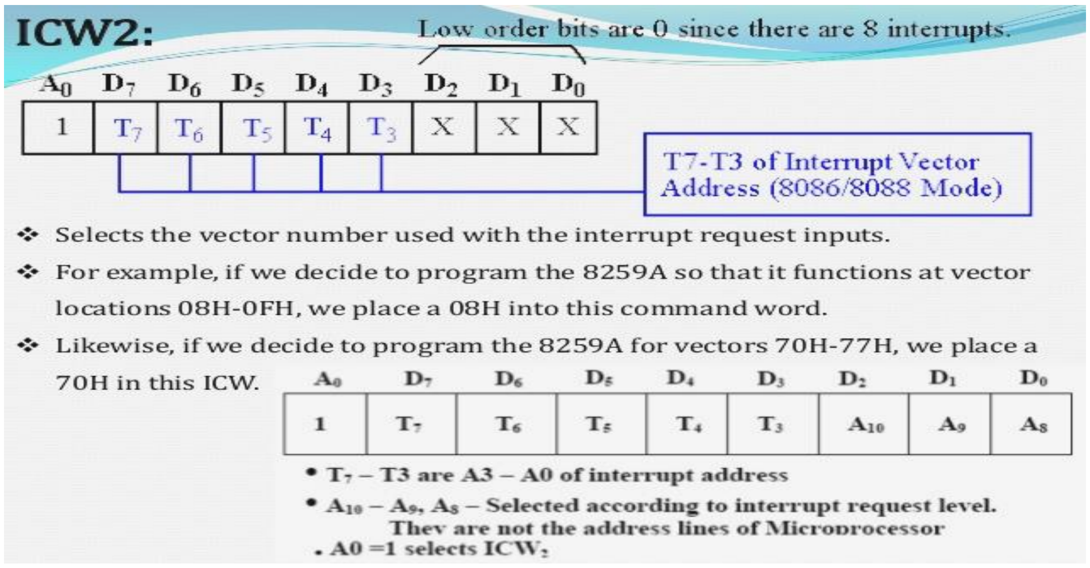
5) The interrupt acknowledge cycles allow the 8259 to release preprogrammed subroutine address onto, the data bus and the subprogram starts execution.

7) This completes the interrupt cycle. In the AEOI (Automatic End of Interrupt) mode the ISR bit is reset at the end of the second INTA pulse.

<u>Initialization Command words:</u>



- To program this ICW for 8086 we place a logic 1 in bit IC4.
- Bits D7, D6 , D5and D2 are don't care for microprocessor operation and only apply to the 8259A when used with an 8-bit 8085 microprocessor.
- This ICW selects single or cascade operation by programming the SNGL bit. If cascade operation is selected, we must also program ICW3.
- The LTIM bit determines whether the interrupt request inputs are positive edge triggered or level-triggered.



ICW2:

Low order bits are 0 since there are 8 interrupts.

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	T_7	T_6	T_5	T_4	T_3	X	X	X

T7-T3 of Interrupt Vector Address (8086/8088 Mode)

- ❖ Selects the vector number used with the interrupt request inputs.
- ❖ For example, if we decide to program the 8259A so that it functions at vector locations 08H-0FH, we place a 08H into this command word.
- ❖ Likewise, if we decide to program the 8259A for vectors 70H-77H, we place a 70H in this ICW.

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	T_7	T_6	T_5	T_4	T_3	A_{10}	A_9	A_8

- T_7 – T3 are A3 – A0 of interrupt address
- A_{10} – A_9, A_8 – Selected according to interrupt request level. They are not the address lines of Microprocessor
- A0 =1 selects ICW_2

Dr.K.Gopi,
Department of ECE.

8279-Keyboard and Display controller

Definition: The INTEL 8279 programmable keyboard/display controller is designed specially for interfacing Keyboard and Display to 8085/8086 microprocessor based systems.

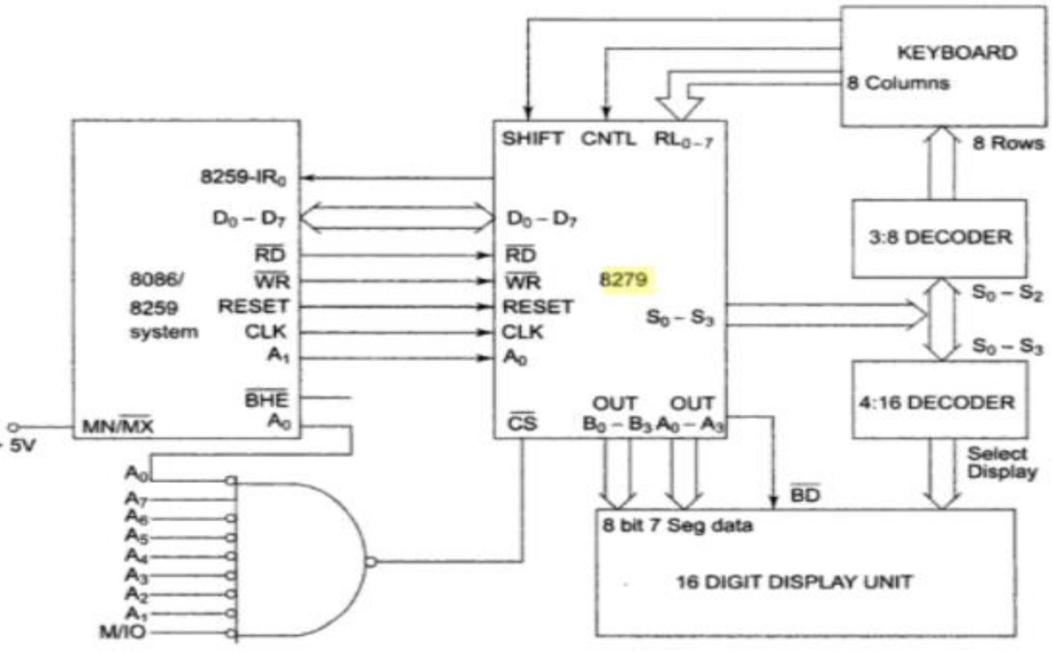
Introduction:

- 8279 programmable keyboard/display controller simultaneously drives the display of a system and interfaces a Keyboard with the CPU.
- The keyboard display interface first scans the keyboard and identifies if any key has been pressed. It then sends their relative code of the pressed key to the CPU and vice-a-versa. It also transmits the data received from the CPU, to the display device. Both the functions are performed by the controller without involving the CPU.
- The Keyboard is interfaced either in the Interrupt or the polled mode.

Features of 8279:

- The keyboard section can interface an array of a maximum of 64keys with the CPU.
- The keyboard entries are debounced and stored in an 8 byte FIFO RAM, that is further accessed by the CPU to read the keycodes.
- The 8279 normally provides a maximum of sixteen 7 segment display interface with the CPU.
- If a FIFO contains a valid key entry the CPU is interrupted or polled mode to read the entry.
- Simultaneous keyboard and display operations
- Scanned keyboard mode.
- 8-character keyboard FIFO.
- Right or left entry 16-byte display RAM.
- Used for Interaction between keyboard and different microprocessor.

8279 Interfacing with 8086 Microprocessor



Pin description of 8279:

Data Bus Lines, DB_0 - DB_7:

These are 8 bidirectional data bus lines used to transfer the data to/from the CPU.

CLK:

The clock input is used to generate internal timings required by the microprocessor.

RESET:

As the name suggests a high on this line is used to Reset the the Intel 8279. when it is enabled the command registers are cleared and 8279 enters in to initial state.

CS Chip Select:

When this pin is set to low, it allows read/write operations, else this pin should be set to high.

A_0:

This pin indicates the transfer of command information. When it is low, it indicates the transfer of data.

RD, WR:

This Read/Write pin enables the data buffer to send/receive data over the data bus.

IRQ:

This interrupt output line goes high when there is data in the FIFO RAM. The interrupt line goes low with each FIFO RAM read operation. However, if the FIFO RAM further contains any key-code entry to be read by the CPU, this pin again goes high to generate an interrupt to the CPU.

V_{ss}, V_{cc}:

These are the ground and power supply lines of the microprocessor.

SL_0 – SL_3:
These are the scan lines used to scan the keyboard matrix and display the digits. These lines can be programmed as encoded or decoded, using the mode control register.
RL_0 – RL_7
These are the Return Lines which are connected to one terminal of keys, while the other terminal of the keys is connected to the decoded scan lines.
SHIFT:
The Shift input line status is stored along with every key code in FIFO in the scanned keyboard mode.
CNTL/STB - CONTROL/STROBED I/P Mode:
In the keyboard mode, this line is used as a control input and stored in FIFO on a key closure. The line enters the data into FIFO RAM, in the strobed input mode.
BD:
It stands for blank display. It is used to blank the display during digit switching.

$OUTA_0$ – $OUTA_3$ and $OUTB_0$ – $OUTB_3$:

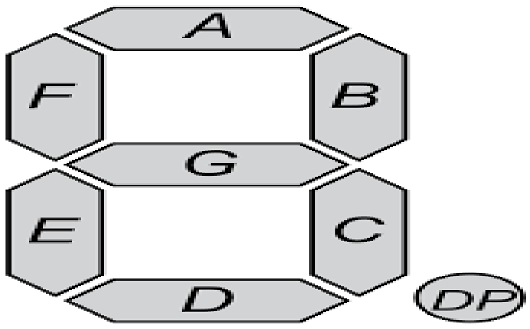
These are the output ports for 16x8 internal display refresh registers. The data from these lines is synchronized with the scan lines to scan the display.

Introduction to seven segment display

Introduction:

- Microprocessor is a device used for Multipurpose, it means it can be used for performing Arithmetic and Logical operations and also used for turning on or off the devices.
- Whenever the information is processed by the Microprocessors, the result obtained by the processor has to be displayed to the user through the output devices(LED, Monitor screen, LCD etc).
- For displaying only numbers, letters and hexadecimal letters, simple 7 segment displays are used.
- The 7 segment type is the least expensive, most commonly used and easiest to interface.
- 7 segment LED display is very popular and it can display digits from 0 to 9 and quite a few characters like A, b, C, ., H, E, e, F, n, o,t,u,y, etc.

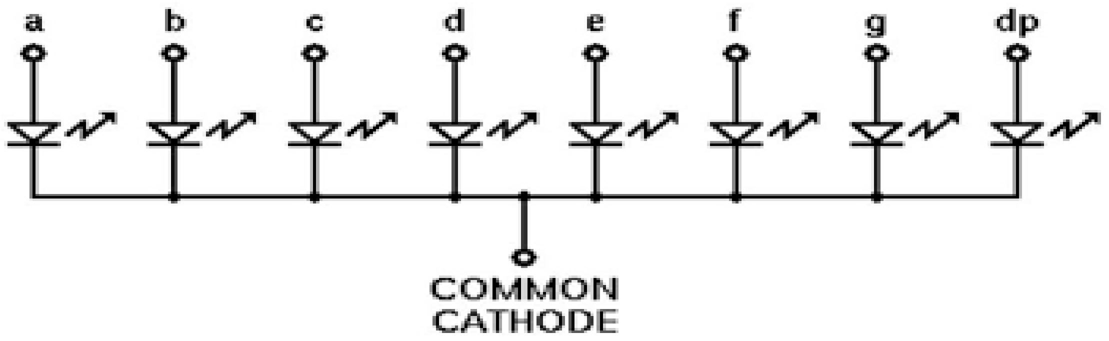
Seven segment display structure:



- Seven segment displays internally consist of 8 LEDs. In these LEDs, 7 LEDs are used to indicate the digits 0 to 9 and Letters and single LED is used for indicating decimal point.
- Generally seven segments are two types, one is common cathode and the other is common anode.
- Seven-segment displays are widely used in digital clocks, electronic meters, basic calculators, and other electronic devices that display numerical information.

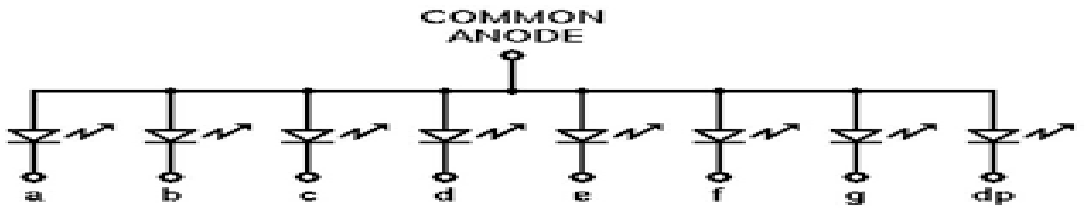
Common cathode display:

- In common cathode the cathode of all Led’s are connected to common pin and the anodes are left free.
- In the common cathode display, all the cathode connections of the LED segments are joined together to logic “0” or ground. The individual segments are illuminated by application of a “HIGH”, or logic “1”.



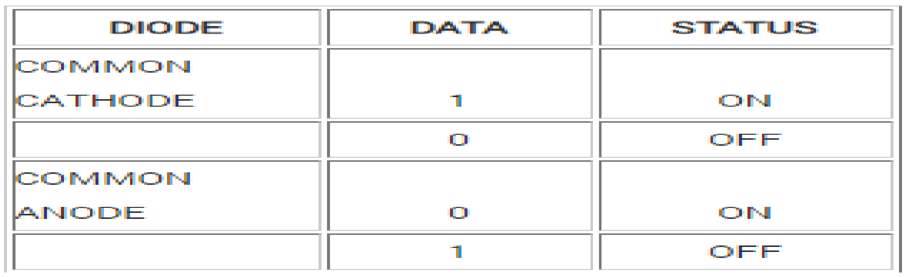
Common Anode Display:

- In common anode the anodes of Led's are connected to common pin and the cathodes are left free.
- In the common anode display, all the anode connections of the LED segments are joined together to logic "1". The individual segments are illuminated by applying a ground, logic "0" or "LOW" signal via a suitable current limiting resistor to the Cathode of the particular segment (a-g)



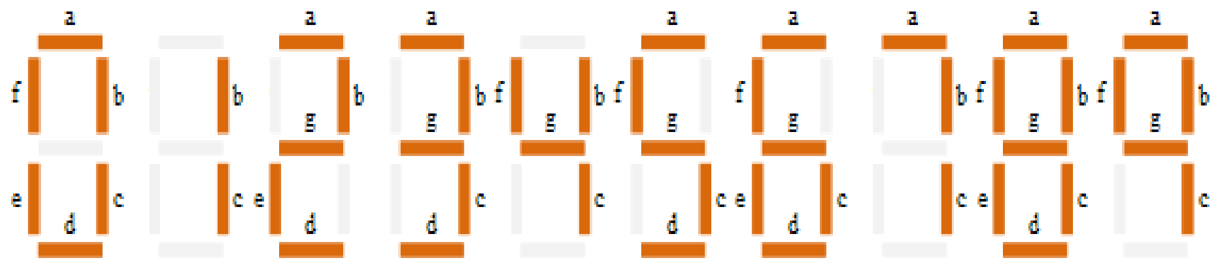
Common anode

- In Common Anode display, anodes of the all LEDs are connected to Vcc, cathodes are connected to microprocessor port pins via 8255.
- In Common Cathode display, cathodes of the all LEDs are connected to Gnd, anodes are connected to microprocessor port pins via 8255.

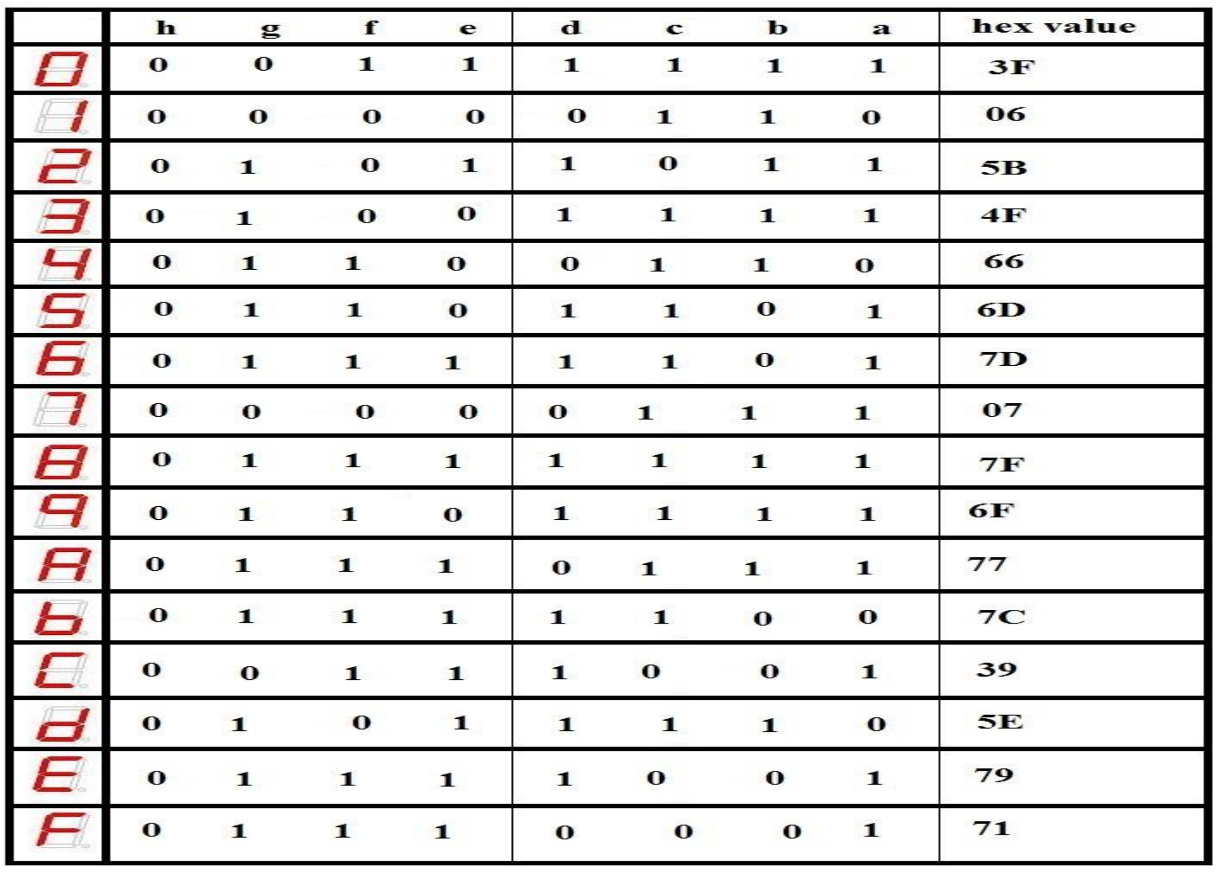


DIODE	DATA	STATUS
COMMON CATHODE	1	ON
	0	OFF
COMMON ANODE	0	ON
	1	OFF

Depending upon the decimal digit to be displayed, the particular set of LEDs to be lighten up. For instance, to display the numerical digit 0, we will need to light up six of the LED segments corresponding to a, b, c, d, e and f. Thus the various digits from 0 through 9 can be displayed using a 7-segment display as shown.

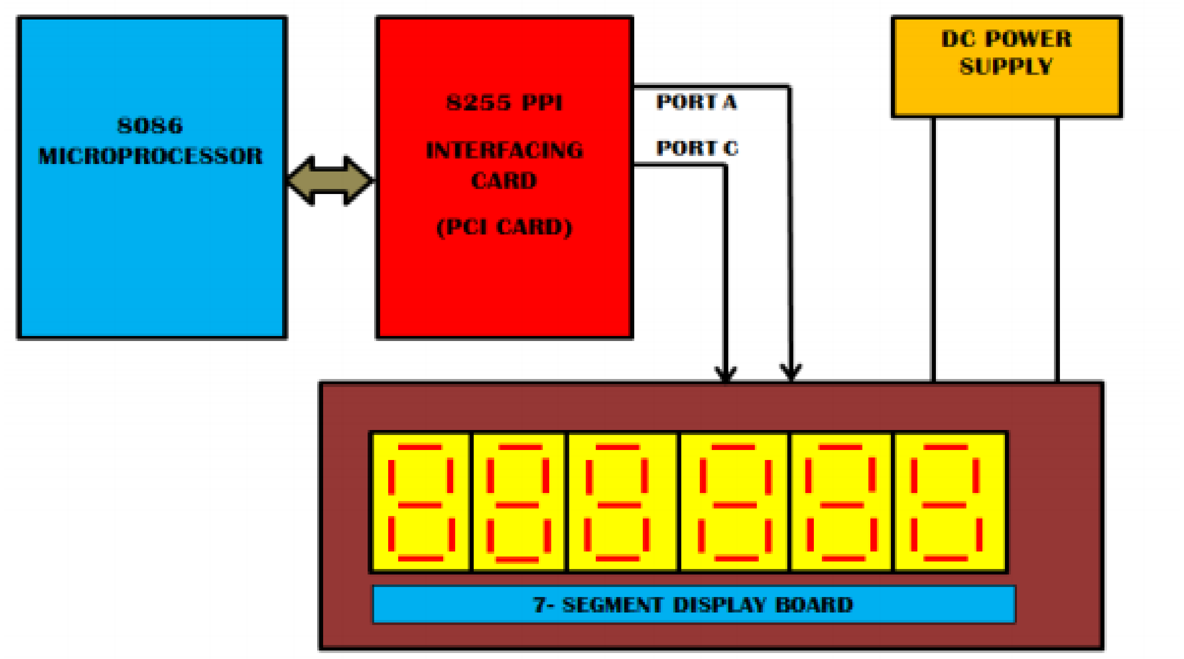


<u>Common cathode table for displaying digits starting from 0,1,2…….9.</u>



	h	g	f	e	d	c	b	a	hex value
0	0	0	1	1	1	1	1	1	3F
1	0	0	0	0	0	1	1	0	06
2	0	1	0	1	1	0	1	1	5B
3	0	1	0	0	1	1	1	1	4F
4	0	1	1	0	0	1	1	0	66
5	0	1	1	0	1	1	0	1	6D
6	0	1	1	1	1	1	0	1	7D
7	0	0	0	0	0	1	1	1	07
8	0	1	1	1	1	1	1	1	7F
9	0	1	1	0	1	1	1	1	6F
A	0	1	1	1	0	1	1	1	77
b	0	1	1	1	1	1	0	0	7C
C	0	0	1	1	1	0	0	1	39
d	0	1	0	1	1	1	1	0	5E
E	0	1	1	1	1	0	0	1	79
F	0	1	1	1	0	0	0	1	71

Block diagram of interfacing Seven Segment Display with 8086 using 8255



- Port A line of the 8255 is buffered and connected to the segments of the seven segment displays, as shown in the table1 .

SL.NO	PORT-LINE	SEGMENT
1.	PA0	A
2.	PA1	B
3.	PA2	C
4.	PA3	D
5.	PA4	E
6.	PA5	F
7.	PA6	G
8.	PA7	DECIMAL POINT

- Port lines PC2, PC1 and PC0 are connected to the inputs of a 74LS145 decoder driver, the outputs of which are connected to the common cathode of the six, Seven-segment displays. The port line combination and corresponding digits enabled are listed in table 2.

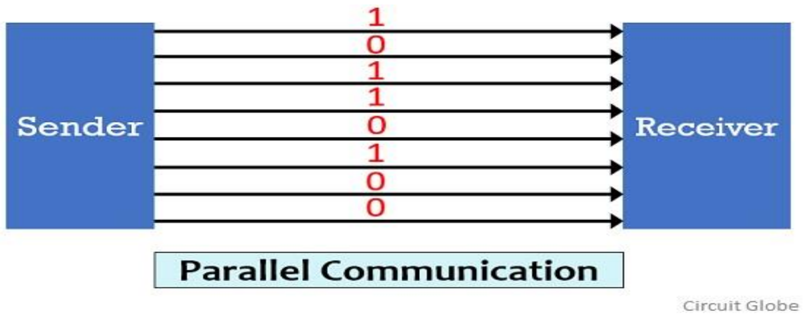
SL.NO	PC2	PC1	PC0	DIGIT
1.	0	0	0	D1
2.	0	0	1	D2
3.	0	1	0	D3
4.	0	1	1	D4
5.	1	0	0	D5
6.	1	0	1	D6

To display a word on a display unit Di (i= 0 to 5), the seven segment code of the character to be displayed on the unit is to be output through port A, while the display unit is selected by the combination of PC2 to PC0 bits of port C.

Parallel Communication

In parallel communication the data bits are simultaneously transmitted using multiple communication links between sender and receiver. Here, various links are used and each bit of data is transmitted separately over all the communication link.

- The figure below shows the transmission of 8 byte data using parallel communication technique:



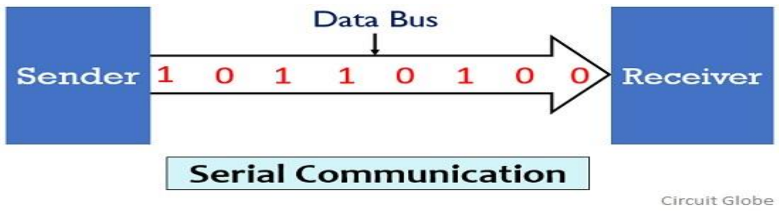
Here, as we can see that for the transmission of 8-bit of data, 8 separate communication links are utilized. This leads to a faster communication between the sender and receiver. But for connecting multiple lines between sender and receiver multiple connecting units are to be present between a pair of sender and receiver.

Disadvantage: parallel communication is not suitable for long distance transmission, because connecting multiple lines to large distances is very difficult and expensive.

Serial Communication

In serial communication the data bits are transmitted serially over a common communication link one after the other. Basically it does not allow simultaneous transmission of data because only a single channel is utilized. Thereby allowing sequential transfer rather than simultaneous transfer.

- The figure below shows the serial data transmission



It is highly suitable for long distance signal transmission as only a single wire or bus is used. So, it can be connected between two points that are separated at a large distance with respect to each other.

Disadvantage: But as only a single data bit is transmitted thus the transmission of data is a quiet time taking process.

Comparision chart of Serial Vs Parallel communication

Basis of comparision	**Serial communication**	**Parallel communication**
Number of communication link used	Single	Multiple
Number of transmitted bit/clock cycle	only one bit.	n number of links will carry n bits.
Data transmission speed	Slow	Comparatively fast
Suitable for	Long distance	Short distance
Cost	Low	High
System Up-gradation	Easy	Quite difficult

Transmission modes

Need for synchronization

Whenever an electronic device transmits digital information to another electronic device. There must certain link establish between the two devices, that is the receiving device must have some way of knowing where each data unit begins and where it ends.

- So there are two types of synchronization

1. Synchronous
2. Asynchronous

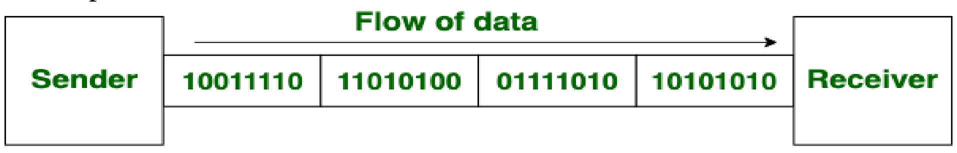
- In digital electronics, both Synchronous and Asynchronous Transmission are the type of serial data transmission in which data is transmitted between sender and receiver.

Synchronous transmission

Synchronous simply means that the communications happen in real time, with all parties engaged simultaneously. Here the data is sent in the form of blocks.

Characteristics of synchronous communication:

- There are no spaces (gaps) in between characters being sent.
- Timing is provided between devices sender and receiver.
- Special 'syn' characters goes before the data being sent, this includes the timing functions.
- It is more efficient and more reliable to transmit large amount of data.
- Examples of synchronous transmissions:
  - Chatrooms
  - Video conferencing
  - Telephonic conversations



Synchronous Transmission

Asynchronous transmission

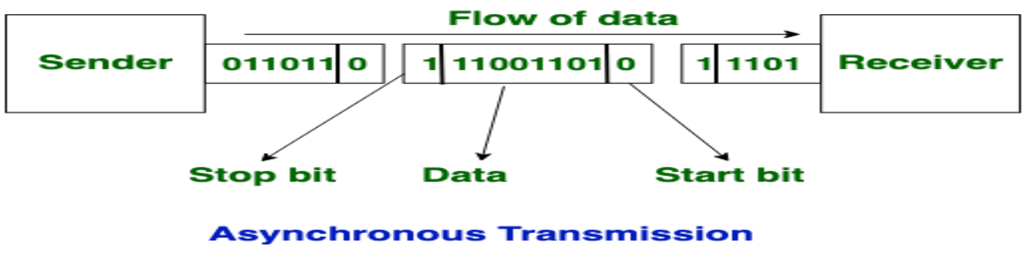
In Asynchronous Transmission, data is sent in form of byte or character. In this transmission start bits and stop bits are added with data. It does not require synchronization.For asynchronous transmission start bit is used to identify the beginning of transmission and stop bit to identify the end of transmission.

Characteristics of Asynchronous Transmission

- Each byte is added with start and stop bits, this is called framing.
- There may be gaps or spaces in between characters.

Examples of Asynchronous Transmission

- Emails
- Radios
- Televisions



Difference between synchronous and asynchronous

Points of comparison	Synchronous transmission	Asynchronous transmission
Definition	Transmits data in the form of blocks	Transmits one byte at a time
Speed of transmission	Quick	slow
cost	Expensive	Less cost
Time interval	constant	Random
Gaps between data	Does not exist	Gaps are introduced
examples	Teleconference, videoconference, chatrooms	Emails, letters etc

Dr.K.Gopi,
Department of ECE.

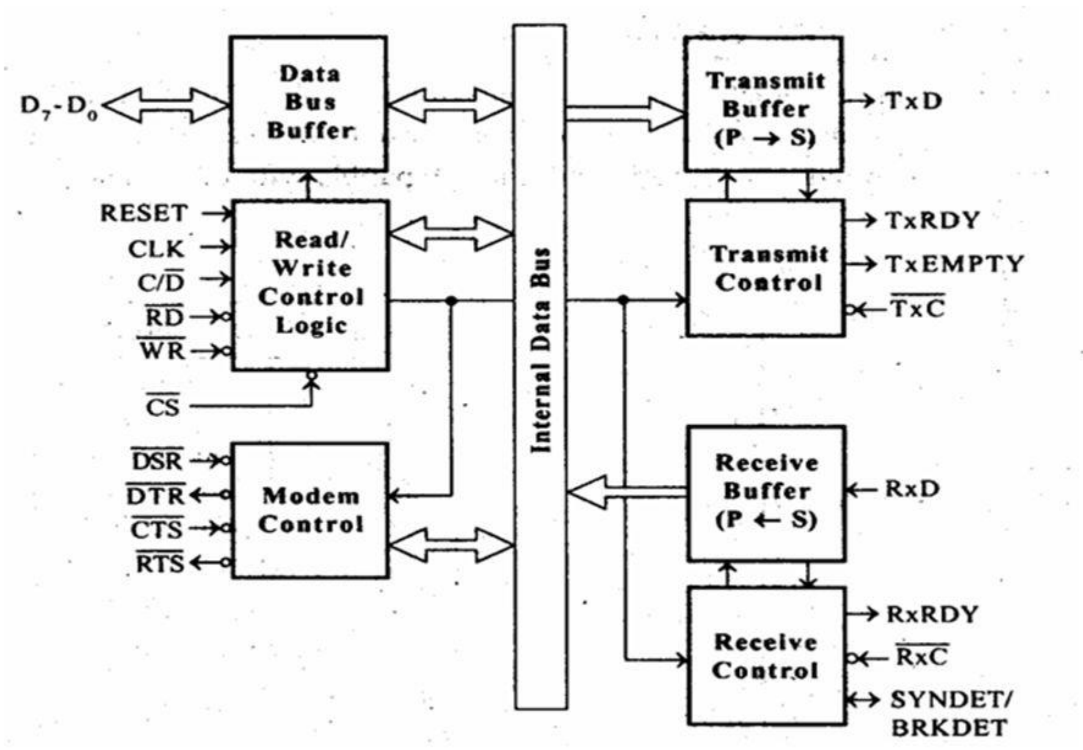
UNIVERSAL SYNCHRONOUS & ASYNCHRONUS TRANSMITTER AND RECEIVER (USART) - 8251 IC

It is sometimes called the programmable Communications Interface (PCI). It is an IC which converts the parallel data to serial data and serial data to parallel data. It supports both synchronous and Asynchronous data transmission. Synchronous operation uses a clock and data line while there is no separate clock accompanying the data for Asynchronous transmission.

Features of 8251 USART:

1. 8251 IC is defined as USART for serial communication.
2. Programmable IC designed for synchronous /asynchronous serial data communication.
3. Receives parallel data from CPU (processor) converts in to serial data after conversion.
4. It also receives serial data from outside and coverts in to parallel data to the CPU (processor) after conversion.
5. It has built-in Baud rate generator
6. It allows full duplex transmission
7. It provides error detection logic, which detects parity, overrun and framing errors.
8. It has 28 pins; DIP package is available

Architecture



The above Fig shows the block diagram of 8251A. It has five sections.

l. Read/Write Control logic

2. Data Bus Buffer

3. Transmitter Section

4. Receiver Section

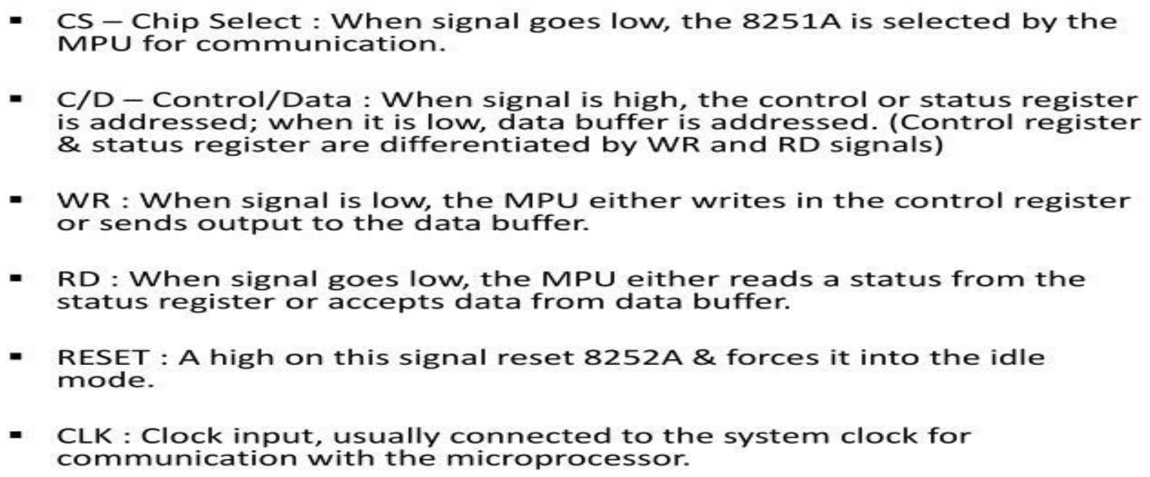
5. Modem Control

- The Read/Write control logic determines the functions of the chip according to the control word in its register and monitors the data flow. The Data Bus Buffer transfers the control word/status information between the chip and CPU.

The Transmitter section converts the parallel word received from CPU to serial bits and transmits over T xD line to peripherals. The Receiver section receives the serial bits from the peripheral, converts them in to parallel word and transfers to the CPU. The Modem Control section extends the data communication through any Modem over telephone lines

<u>Data bus buffer:</u> This data bus buffer is used to write the command, status or data from or to the 8251.

<u>Read/ Write control logic</u>: Interfaces the 8251 with the Microprocessor, determines the function of the chip according to the control word in the control register and monitors the data flow.



- CS – Chip Select : When signal goes low, the 8251A is selected by the MPU for communication.
- C/D – Control/Data : When signal is high, the control or status register is addressed; when it is low, data buffer is addressed. (Control register & status register are differentiated by WR and RD signals)
- WR : When signal is low, the MPU either writes in the control register or sends output to the data buffer.
- RD : When signal goes low, the MPU either reads a status from the status register or accepts data from data buffer.
- RESET : A high on this signal reset 8252A & forces it into the idle mode.
- CLK : Clock input, usually connected to the system clock for communication with the microprocessor.

Transmit buffer –

This block is used for parallel to serial converter that receives a parallel byte for conversion into serial signal and further transmission onto the common channel.

- **TXD:** It is an output signal, if its value is one, means transmitter will transmit the data.

Transmit control –

This block is used to control the data transmission with the help of following pins:

- **TXRDY:** It means transmitter is ready to transmit data character.
- **TXEMPTY:** An output signal which indicates that TXEMPTY pin has transmitted all the data characters and transmitter is empty now.
- **TXC:** An active-low input pin which controls the data transmission rate of transmitted data.

Receive buffer –
This block receives the serial data and converts it to parallel data.

- **RXD:** An input signal which receives the data.

Receive control –
This block controls the receiving data.

- **RXRDY:** An input signal indicates that it is ready to receive the data.
- **RXC:** An active-low input signal which controls the data transmission rate of received data.
- **SYNDET/BD:**

❖ During asynchronous mode, the signal SYNDET/BRKDET will indicate the break in the data transmission.

❖ During synchronous mode, the signal SYNDET/BRKDET will indicate the reception of synchronous character.

Modem control:

- The MODEM control unit allows to interface a MODEM to 8251A and to establish data communication through MODEM over telephone lines.
- A device converts analog signals to digital signals and vice-versa and helps the computers to communicate over telephone lines or cable wires. The following are active-low pins of Modem.

DSR: Data Set Ready signal, This signal when low indicates Modem is ready for communication

DTR: Data terminal Ready signal, this signal when low indicates terminal equipment is ready for communication.

CTS: clear to send, this signal when high indicates that DTE has data and ready to transmit the data to Modem

RTS: Request to send, this signal when low indicates that the DCE is free and can receive the data now for DTE.

DTE (DATA TERMINALEQUIPMENT): It includes any unit that functions either as source or destination for binary digital data.

DCE (Data communication equipment): it includes any circuit that transmits and receive the data in the form or analog or digital signal.

OPERATING MODE OF 8251

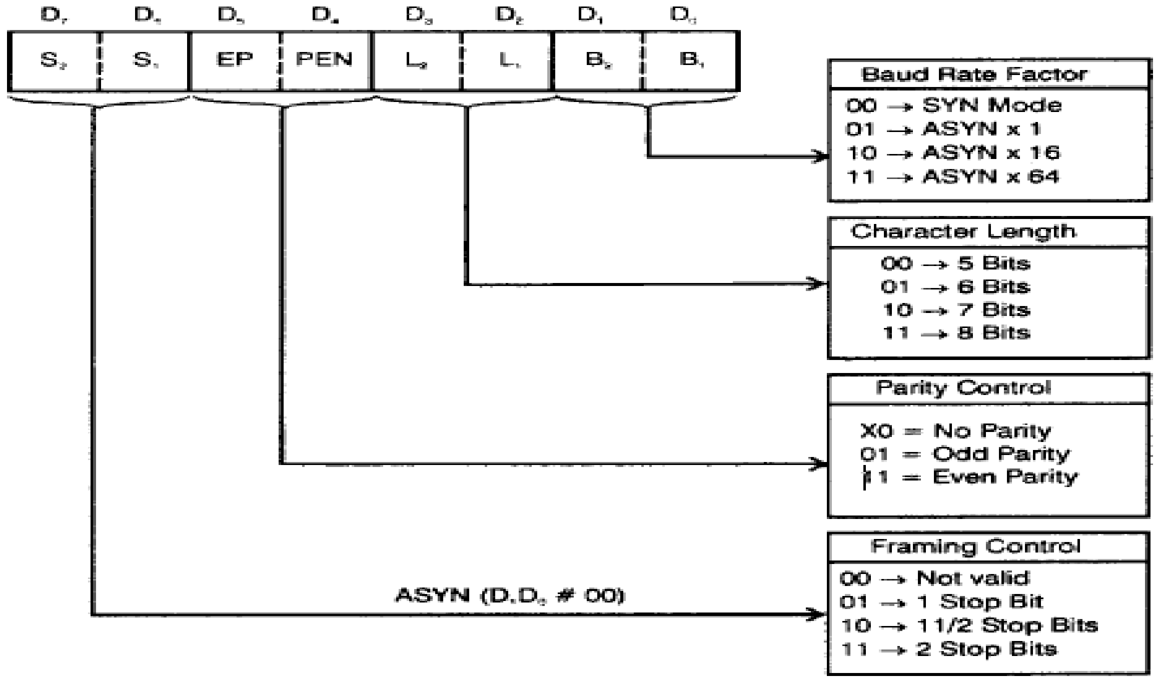
To communicate with 8251A.t he CPU has to inform the details about mode, baud rate, stop bits, parity bit etc., to USART. This is done by a set of control words. The CPU must check the status (ready) of the peripheral by reading the status register. The control words are divided in to two formats.

Mode word

The Mode word specifies the general characteristics of operation such as baud, parity, number of stop bits.

Mode Word

Fig. shows the mode word format.



- The format can be considered as four 2-bit fields. The first 2-bit field (D_1-D_0) determines whether the USART is to operate in the synchronous (00) or asynchronous mode.
- In the asychronous mode, this field determines the division factor for clock to decide the baud rate. For example, if D_1 and D_0 are both ones, the RxC and TxC will be divided to generate the baudrate(Bits per second)

- The second 2-bit field (D_3-D_2) determines number of data bits in one character. With this 2-bit field we can set character length from 5-bits to 8 bits.

- The third 2-bit field, (D_5-D_4), controls the parity generation. The parity bit is added to the data bits only if parity is enabled.

The last field, (D_7-D_6), has two meanings depending on whether operation is to be in the synchronous or asynchronous mode. For asynchronous mode, (i.e. $D_1D_0 \neq 00$), it controls the number of STOP bits to be transmitted with the character. In synchronous mode, (i.e. D_1D_0) = **00) this field controls the synchronizing process. It decides whether to transmit single synchronizing character or two synchronizing characters**