UNIT-I

- **Number Systems & Code conversion, Boolean Algebra & Logic Gates, Truth Tables, Universal Gates, Simplification of Boolean functions, SOP and POS methods – Simplification of Boolean functions using K-maps,Signed and Unsigned Binary Numbers.**

Number System Conversion Table

Binary Numbers	Octal Numbers	Decimal Numbers
1000	10	8
1001	11	9
1010	12	10
1011	13	11

There are many methods or techniques which can be used to convert numbers from one base to another. We'll demonstrate here the following −

- Decimal to Other Base System
- Other Base System to Decimal
- Other Base System to Non-Decimal
- Shortcut method − Binary to Octal
- Shortcut method − Octal to Binary
- Shortcut method − Binary to Hexadecimal
- Shortcut method − Hexadecimal to Binary

Decimal to Other Base System

Steps

- **Step 1** − Divide the decimal number to be converted by the value of the new base.
- **Step 2** − Get the remainder from Step 1 as the rightmost digit (least significant digit) of new base number.
- **Step 3** − Divide the quotient of the previous divide by the new base.
- **Step 4** − Record the remainder from Step 3 as the next digit (to the left) of the new base number.

Repeat Steps 3 and 4, getting remainders from right to left, until the quotient becomes zero in Step 3.

The last remainder thus obtained will be the Most Significant Digit (MSD) of the new base number.

Example −

Decimal Number: 29_{10}

Calculating Binary Equivalent −

Step	Operation	Result	Remainder
Step 1	29 / 2	14	1
Step 2	14 / 2	7	0
Step 3	7 / 2	3	1
Step 4	3 / 2	1	1
Step 5	1 / 2	0	1

Other Base System to Decimal System

Steps

- **Step 1** − Determine the column (positional) value of each digit (this depends on the position of the digit and the base of the number system).
- **Step 2** − Multiply the obtained column values (in Step 1) by the digits in the corresponding columns.
- **Step 3** − Sum the products calculated in Step 2. The total is the equivalent value in decimal.

Example

Binary Number − 11101_2

Calculating Decimal Equivalent −

Step	Binary Number	Decimal Number
Step 1	11101_2	$((1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10}$
Step 2	11101_2	$(16 + 8 + 4 + 0 + 1)_{10}$
Step 3	11101_2	29_{10}

Binary Number − 11101_2 = Decimal Number − 29_{10}

Other Base System to Non-Decimal System

Steps

- **Step 1** − Convert the original number to a decimal number (base 10).
- **Step 2** − Convert the decimal number so obtained to the new base number.

Example

Octal Number − 25_8

Calculating Binary Equivalent −

Step 1 − Convert to Decimal

Step	Octal Number	Decimal Number
Step 1	25_8	$((2 \times 8^1) + (5 \times 8^0))_{10}$
Step 2	25_8	$(16 + 5)_{10}$
Step 3	25_8	21_{10}

Octal Number − 25_8 = Decimal Number − 21_{10}

Step 2 – Convert Decimal to Binary

Step	Operation	Result	Remainder
Step 1	21 / 2	10	1
Step 2	10 / 2	5	0
Step 3	5 / 2	2	1
Step 4	2 / 2	1	0
Step 5	1 / 2	0	1

Decimal Number – 21_{10} = Binary Number – 10101_2

Octal Number – 25_8 = Binary Number – 10101_2

Shortcut method - Binary to Octal

Steps

- **Step 1** − Divide the binary digits into groups of three (starting from the right).
- **Step 2** − Convert each group of three binary digits to one octal digit.

Example

Binary Number − 10101_2

Calculating Octal Equivalent −

Step	Binary Number	Octal Number
Step 1	10101_2	010 101
Step 2	10101_2	2_8 5_8
Step 3	10101_2	25_8

Binary Number − 10101_2 = Octal Number − 25_8

Shortcut method - Octal to Binary

Steps

- **Step 1** − Convert each octal digit to a 3 digit binary number (the octal digits may be treated as decimal for this conversion).
- **Step 2** − Combine all the resulting binary groups (of 3 digits each) into a single binary number.

Example

Octal Number − 25_8

Calculating Binary Equivalent −

Step	Octal Number	Binary Number
Step 1	25_8	$2_{10}\ 5_{10}$
Step 2	25_8	$010_2\ 101_2$
Step 3	25_8	010101_2

Octal Number − 25_8 = Binary Number − 10101_2

Shortcut method - Binary to Hexadecimal

Steps

- **Step 1** − Divide the binary digits into groups of four (starting from the right).
- **Step 2** − Convert each group of four binary digits to one hexadecimal symbol.

Example

Binary Number − 10101_2

Calculating hexadecimal Equivalent −

Step	Binary Number	Hexadecimal Number
Step 1	10101_2	0001 0101
Step 2	10101_2	1_{10} 5_{10}
Step 3	10101_2	15_{16}

Binary Number − 10101_2 = Hexadecimal Number − 15_{16}

Shortcut method - Hexadecimal to Binary

Steps

- **Step 1** − Convert each hexadecimal digit to a 4 digit binary number (the hexadecimal digits may be treated as decimal for this conversion).
- **Step 2** − Combine all the resulting binary groups (of 4 digits each) into a single binary number.

Example

Hexadecimal Number − 15_{16}

Calculating Binary Equivalent −

Step	Hexadecimal Number	Binary Number
Step 1	15_{16}	$1_{10}\ 5_{10}$
Step 2	15_{16}	$0001_2\ 0101_2$
Step 3	15_{16}	00010101_2

Hexadecimal Number − 15_{16} = Binary Number − 10101_2

LOGIC GATES & BOOLEAN ALGEBRA

Buffer
$F = A$

A	F
0	0
1	1

AND
$F = AB$

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

OR
$F = A+B$

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

XOR
$F = A \oplus B$

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

Inverter
$F = \overline{A}$

A	F
0	1
1	0

NAND
$F = \overline{AB}$

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

NOR
$F = \overline{A+B}$

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

XNOR
$F = \overline{A \oplus B}$

A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

Boolean Algebra

Boolean algebra, a logic algebra, allows the rules used in the algebra of numbers to be applied to logic. It formalizes the rules of logic. Boolean algebra is used to simplify Boolean expressions which represent combinational logic circuits. It reduces the original expression to an equivalent expression that has fewer terms which means that less logic gates are needed to implement the combinational logic circuit.

Boolean Expression Calculator

Use the calculator to find the reduced boolean expression or to check your

Boolean expression ~(A * B) * (~A + B) * (~B + B)

Your answer ~A + ~B * B

The boolean expression is reduced to ~A

Your answer is equivalent

Truth Table

A	B	output
0	0	1
0	1	1
1	0	0
1	1	0

Notes:

- Use ~ * + to represent NOT AND OR respectively. Do not omit the * operator for an AND operation.
  (~AB)+(B~C)+(AB) will return an error
  (~A*B)+(B*~C)+(A*B) is OK
- Boolean operations follows a precedence order of NOT AND OR. Expressions inside brackets () are always evaluated first, overriding the precedence order.
- Please enter variables only, constants like 0,1 are not allowed.
- Variables E, I, N, O, Q, S are not allowed

Boolean Expression Simplification

The following example shows how to use algebraic techniques to simplify a boolean expression

~(A * B) * (~A + B) * (~B + B)	
~(A * B) * (~A + B) * 1	6 - Complement law
~(A * B) * (~A + B)	5 - Identity law
(~A + ~B) * (~A + B)	8 - DeMorgan's law
~A + ~B * B	4 - Distributive law
~A + 0	6 - Complement law
~A	5 - Identity law

Each line (or step) gives a new expression and the rule or rules used to derive it from the previous one. There can be several ways to arrive at the final result. You can use our calculator to check the intermediate steps of your answer. Equivalent means your answer and the original boolean expression have the same truth table.

Laws of Boolean Algebra

Boolean Algebra Laws are used to simplify boolean expressions.

Basic Boolean Laws

1. Idempotent Law
   - A * A = A
   - A + A = A
2. Associative Law
   - (A * B) * C = A * (B * C)
   - (A + B) + C = A + (B + C)
3. Commutative Law
   - A * B = B * A
   - A + B = B + A
4. Distributive Law
   - A * (B + C) = A * B + A * C
   - A + (B * C) = (A + B) * (A + C)
5. Identity Law
   - A * 0 = 0 A * 1 = A
   - A + 1 = 1 A + 0 = A
6. Complement Law
   - A * ~A = 0
   - A + ~A = 1
7. Involution Law
   - ~(~A) = A
8. DeMorgan's Law
   - ~(A * B) = ~A + ~B
   - ~(A + B) = ~A * ~B

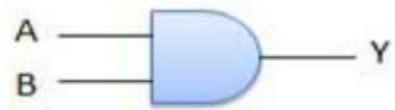
Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on a **certain logic**. Based on this, logic gates are named as AND gate, OR gate, NOT gate etc.

AND Gate

A circuit which performs an AND operation is shown in figure. It has n input (n >= 2) and one output.

Y = A AND B AND C ..... N
Y = A.B.C ..... N
Y = ABC ..... N

Logic diagram



Truth Table

Inputs		Output
A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate

A circuit which performs an OR operation is shown in figure. It has n input (n >= 2) and one output.

Y = A OR B OR C ..... N
Y = A + B + C ..... N

Logic diagram

Truth Table

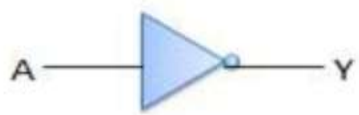
Inputs		Output
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gate

NOT gate is also known as **Inverter**. It has one input A and one output Y.

Y = NOT A
Y = $\overline{A}$

Logic diagram



Truth Table

Inputs	Output
A	B
0	1
1	0

NAND Gate

A NOT-AND operation is known as NAND operation. It has n input (n >= 2) and one output.

Y = A NOT AND B NOT AND C ..... N
Y = A NAND B NAND C ..... N

Logic diagram

Truth Table

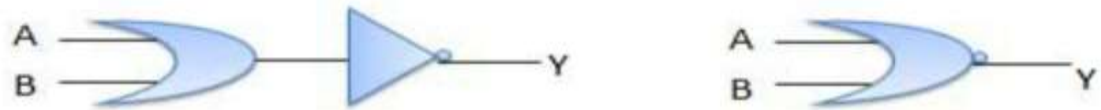
Inputs		Output
A	B	$\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

A NOT-OR operation is known as NOR operation. It has n input (n >= 2) and one output.

$$Y = \text{A NOT OR B NOT OR C ..... N}$$
$$Y = \text{A NOR B NOR C ..... N}$$

Logic diagram



Truth Table

Inputs		Output
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

XOR Gate

XOR or Ex-OR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-OR gate is abbreviated as EX-OR gate or sometime as X-OR gate. It has n input (n >= 2) and one output.

$$Y = \text{A XOR B XOR C ..... N}$$
$$Y = A \oplus B \oplus C \text{ ..... N}$$
$$Y = \overline{A}B + A\overline{B}$$

Logic diagram

Truth Table

Inputs		Output
A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

XNOR Gate

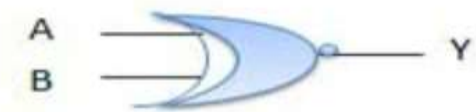
XNOR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-NOR gate is abbreviated as EX-NOR gate or sometime as X-NOR gate. It has n input (n >= 2) and one output.

$$Y = A \text{ XOR } B \text{ XOR } C \ldots\ldots N$$

$$Y = A \odot B \odot C \ldots\ldots N$$

$$Y = \overline{A}B + A\overline{B}$$

Logic diagram

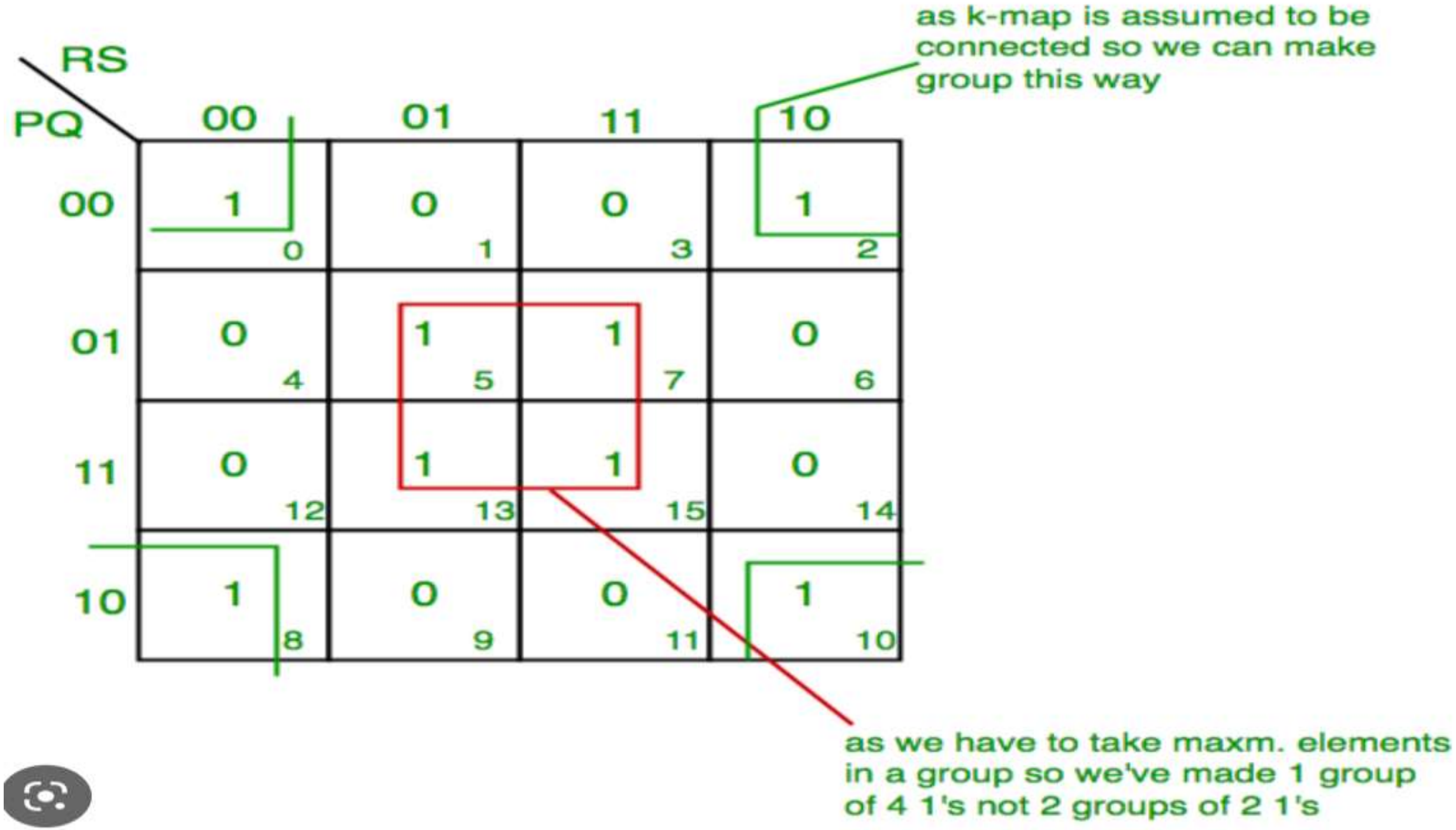


TRUTH TABLE

a proposition p	not p (negation) $\sim p$	a proposition p	a proposition q	p and q (conjunction) $p \,\&\, q$	p or q, inclusive (inclusive disjunction) p or q	p or q, exclusive (exclusive disjunction) p or q	if p then q (implication) $p \rightarrow q$	p if and only if q (biconditional) $p \leftrightarrow q$
T	F	T	T	T	T	F	T	T
F	T	T	F	F	T	T	F	F
		F	T	F	T	T	T	F
		F	F	F	F	F	T	T

T = true

F = false


RS
PQ
00
01
11
10
00
01
11
10
1 0 0 1
0 4 5 7 6 0 1 1 0
0 1 1 0
1 0 0 1
0 1 3 2
12 13 15 14
8 9 11 10
as k-map is assumed to be connected so we can make group this way
as we have to take maxm. elements in a group so we've made 1 group of 4 1's not 2 groups of 2 1's

UNIT-II

Combinational Logic Circuits: Adders &Subtractors, Multiplexers, Demultiplexers, Encoders, Decoders, Programmable Logic Devices.

Adders and Subtractors in Digital Logic

Subtraction of two binary numbers can be accomplished by adding 2's complement of the subtrahend to the minuend and disregarding the final carry if any. If the MSB bit in the result of addition is a '0'. then the result of addition is the correct answer. If the MSB bit is a '1'. , this implies that the answer has a negative sign. The true magnitude, in this case, is given by 2's complement of the result of the addition.

Block Diagram of Combinational Logic Circuit:

Points to Remember on Combinational Logic Circuit:

1. Output depends upon the combination of inputs.
2. Output is a pure function of present inputs only i.e., Previous State inputs won't have any effect on the output. Also, It doesn't use memory.
3. In other words,

```
OUTPUT=f(INPUT)
```

1. Inputs are called Excitation from circuits and outputs are called Responses of combinational logic circuits.

Classification of Combinational Logic Circuits:

1. Arithmetic:

- Adders
- Subtractors
- Multipliers
- Comparators

2. Data Handling:

- Multiplexers
- DeMultiplexers
- Encoders and Decoders

3. Code Converters:

- BCD to Excess-3 code and vice versa
- BCD to Gray code and vice versa
- Seven Segment

Multiplexer

A multiplexer is a combinational circuit that has 2^n input lines and a single output line. Simply, the multiplexer is a multi-input and single-output combinational circuit. The binary information is received from the input lines and directed to the output line. On the basis of the values of the selection lines, one of these data inputs will be connected to the output.

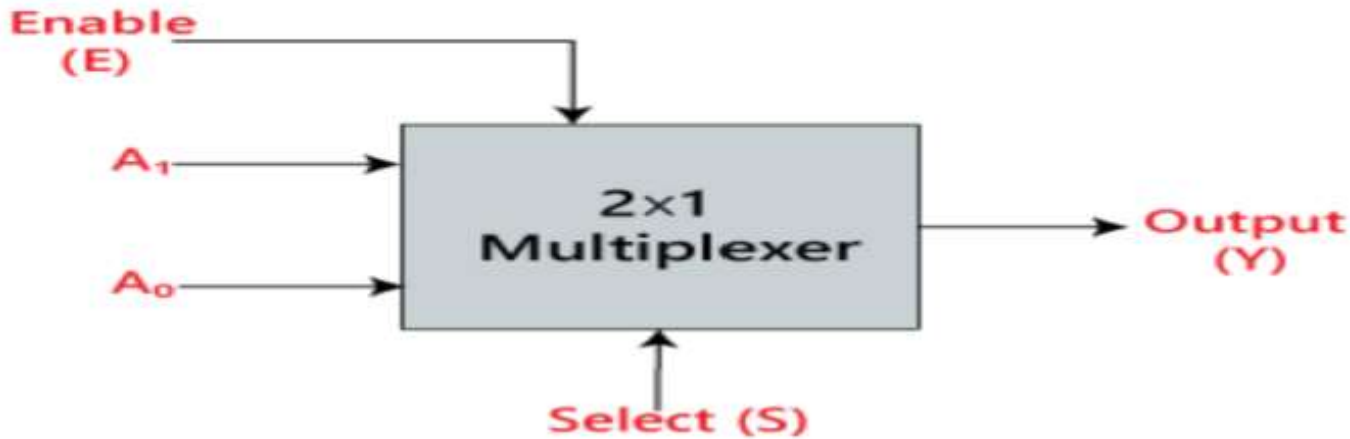
Unlike encoder and decoder, there are n selection lines and 2^n input lines. So, there is a total of 2^N possible combinations of inputs. A multiplexer is also treated as **Mux**.

There are various types of the multiplexer which are as follows:

2×1 Multiplexer:

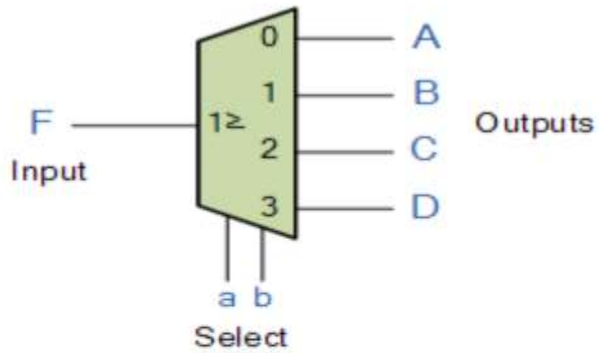
In 2×1 multiplexer, there are only two inputs, i.e., A_0 and A_1, 1 selection line, i.e., S_0 and single outputs, i.e., Y. On the basis of the combination of inputs which are present at the selection line S^0, one of these 2 inputs will be connected to the output. The block diagram and the truth table of the 2×1 multiplexer are given below.

Block Diagram:



Truth Table:

INPUTS	Output
S_0	Y
0	A_0
1	A_1



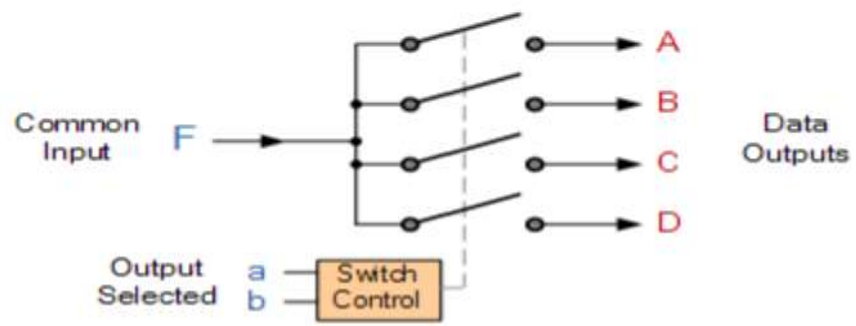
The Demultiplexer

The demultiplexer is a combinational logic circuit designed to switch one common input line to one of several seperate output line

The data distributor, known more commonly as the demultiplexer or "Demux" for short, is the exact opposite of the Multiplexer we saw in the previous tutorial.

The *demultiplexer* takes one single input data line and then switches it to any one of a number of individual output lines one at a time. The **demultiplexer** converts a serial data signal at the input to a parallel data at its output lines as shown below.

1-to-4 Channel De-multiplexer



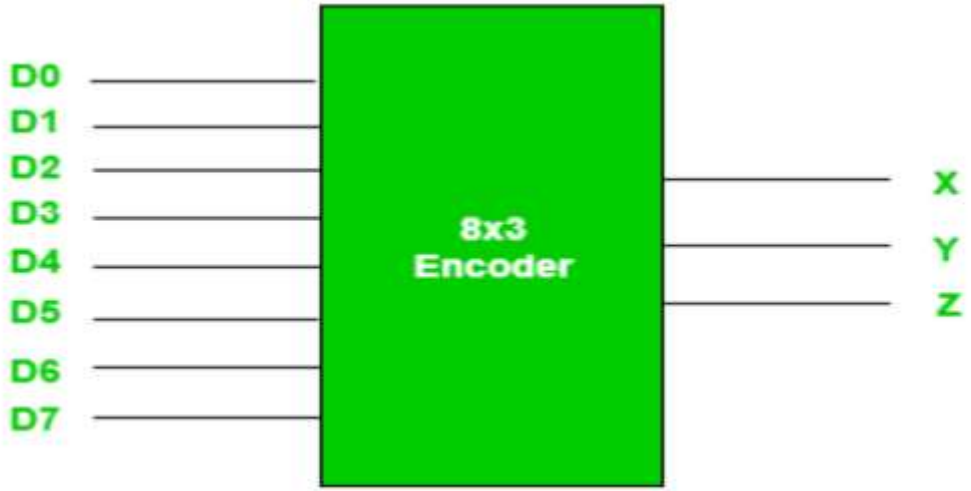
Encoders and Decoders in Digital Logic

Binary code of N digits can be used to store 2^N distinct elements of coded information. This is what encoders and decoders are used for. **Encoders** convert 2^N lines of input into a code of N bits and **Decoders** decode the N bits into 2^N lines.

1. Encoders –

An encoder is a combinational circuit that converts binary information in the form of a 2^N input lines into N output lines, which represent N bit code for the input. For simple encoders, it is assumed that only one input line is active at a time.

As an example, let's consider **Octal to Binary** encoder. As shown in the following figure, an octal-to-binary encoder takes 8 input lines and generates 3 output lines.



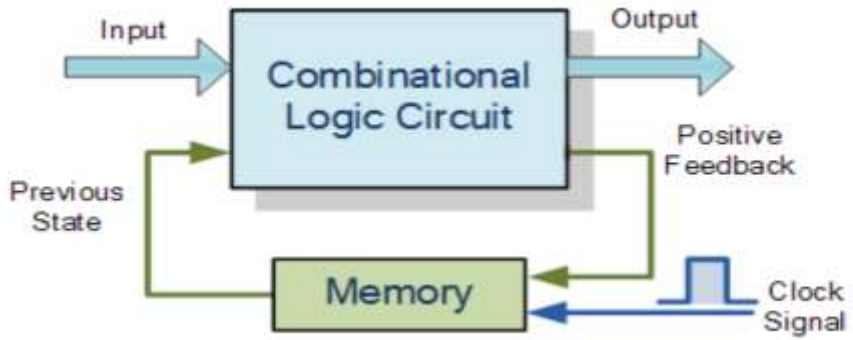
Programmable logic device

A **programmable logic device** (**PLD**) is an electronic component used to build reconfigurable digital circuits. Unlike digital logic constructed using discrete logic gates with fixed functions, a PLD has an undefined function at the time of manufacture. Before the PLD can be used in a circuit it must be programmed to implement the desired function.[1] Compared to fixed logic devices, programmable logic devices simplify the design of complex logic and may offer superior performance.[2] Unlike for microprocessors, programming a PLD changes the connections made between the gates in the device.

PLDs can broadly be categorised into, in increasing order of complexity, Simple Programmable Logic Devices (SPLDs), comprising programmable array logic, programmable logic array and generic array logic; Complex Programmable Logic Devices (CPLDs) and Field-Programmable Gate Arrays (FPGAs).

UNIT-III

Sequential Logic Circuits: RS, Clocked RS, D, JK, Master Slave JK, T Flip-Flops, Shift Registers, Types of Shift Registers, Counters, Ripple Counter, Synchronous Counters, Asynchronous Counters, Up-Down Counter.



Sequential Logic Circuits

Sequential Logic Circuits use flip-flops as memory elements and in which their output is dependent on the input state

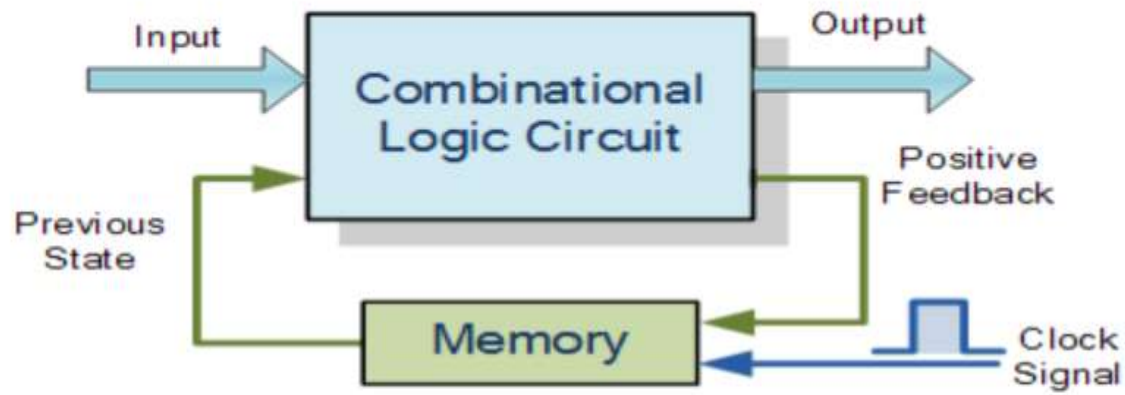
Unlike Combinational Logic circuits that change state depending upon the actual signals being applied to their inputs at that time, **Sequential Logic** circuits have some form of inherent "Memory" built in.

This means that sequential logic circuits are able to take into account their previous input state as well as those actually present, a sort of "before" and "after" effect is involved with sequential circuits.

In other words, the output state of a "sequential logic circuit" is a function of the following three states, the "present input", the "past input" and/or the "past output". *Sequential Logic circuits* remember these conditions and stay fixed in their current state until the next clock signal changes one of the states, giving sequential logic circuits "Memory".

Sequential logic circuits are generally termed as *two state* or Bistable devices which can have their output or outputs set in one of two basic states, a logic level "1" or a logic level "0" and will remain "latched" (hence the name latch) indefinitely in this current state or condition until some other input trigger pulse or signal is applied which will cause the bistable to change its state once again.

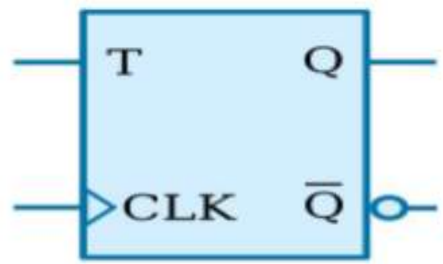
Sequential Logic Representation



The word "Sequential" means that things happen in a "sequence", one after another and in **Sequential Logic** circuits, the actual clock signal determines when things will happen next. Simple sequential logic circuits can be constructed from standard **Bistable** circuits such as: *Flip-flops*, *Latches* and *Counters* and which themselves can be made by simply

T Flip-Flop

The toggle, or T, flip-flop is a two-input flip-flop. The inputs are the toggle (T) input and a clock (CLK) input. If the toggle input is HIGH, the T flip-flop changes state (toggles) when the clock signal is applied. If the toggle input is LOW, the T flip-flop holds the previous state.

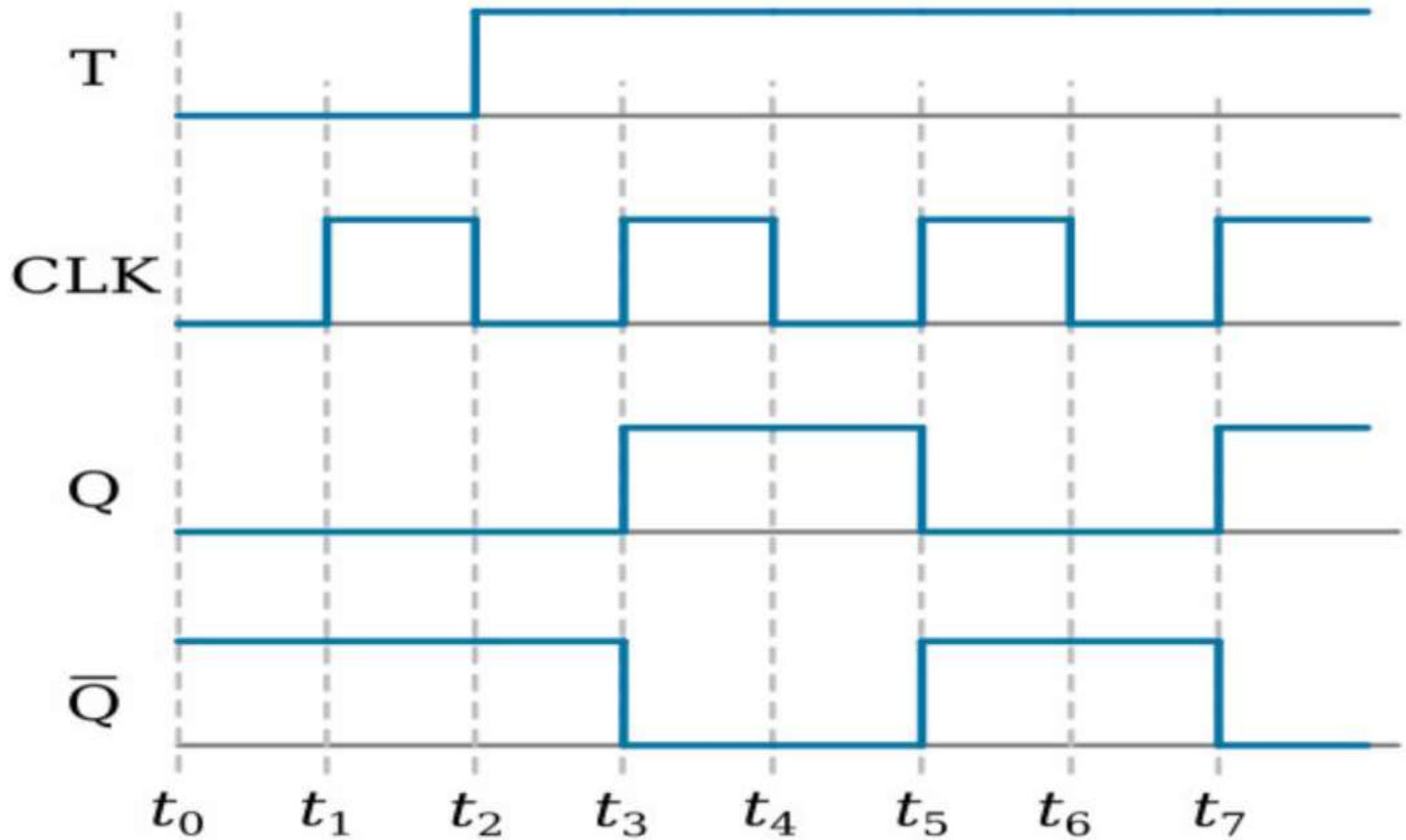


T flip-flop symbol.

The standard symbol for a T flip-flop is illustrated in the figure above. The clock input may be preceded by an inverter. An inverter indicates a flip-flop will toggle on a HIGH-to-LOW transition of the clock pulse. The absence of an inverter indicates the flip-flop will toggle on a LOW-to-HIGH transition of the pulse.

Truth table			
CLK	T	Q_{next}	Comment
Rising edge	0	Q	Hold state
Falling edge	0	Q	Hold state
Rising edge	1	$\overline{Q}$	Toggle
Falling edge	1	Q	No change

Q_{next} - "after the clock transition" output
Q - the current output


T
CLK
Q
$\overline{Q}$
t_0
t_1
t_2
t_3
t_4
t_5
t_6
t_7

Now, follow the explanation of the circuit using the truth table and the timing diagram shown in the figure above. The timing diagram shows the inputs and the resulting outputs. We will assume an initial condition (t_0) of Q being LOW and $\overline{Q}$ being HIGH. At t_1, when the clock changes from a LOW to a HIGH, the outputs remain the same as the T input is LOW. The T input goes HIGH at t_2. At t_3, the clock changes from a LOW to a HIGH and the device changes state; Q goes HIGH and $\overline{Q}$ goes LOW. The outputs remain the same at t_4 since the device is switched only by a LOW-to-HIGH transition. At t_5, when the clock goes HIGH, Q goes LOW and $\overline{Q}$ goes HIGH; they remain that way until t_7.

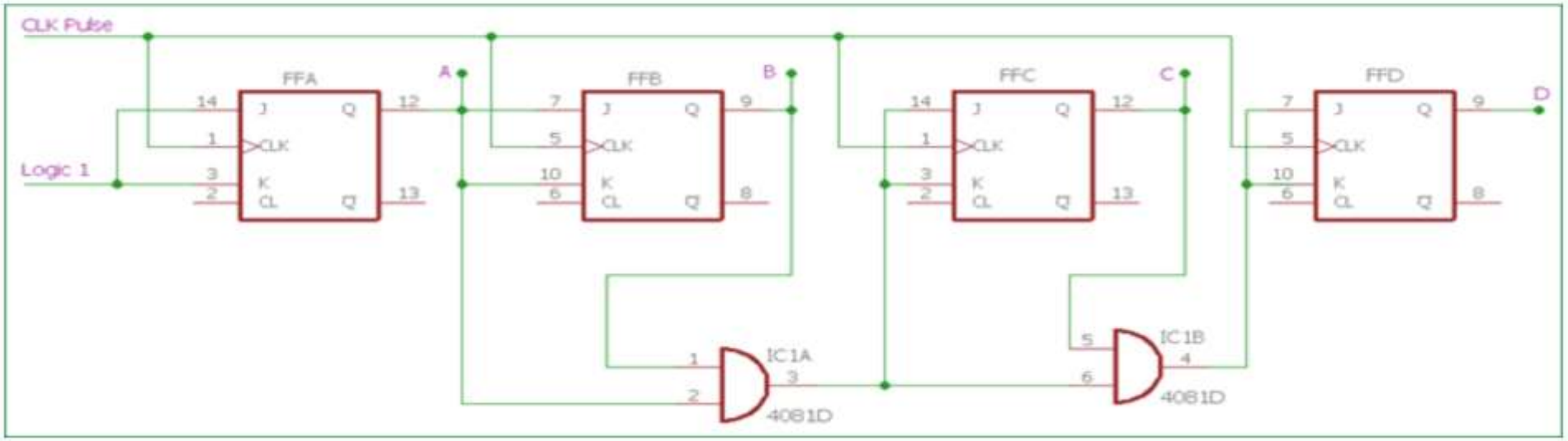
Between t_3 and t_7, two complete cycles of CLK occur. During the same time period, only one cycle is observed for Q or $\overline{Q}$. Since the output frequency is one-half the clock (input) frequency, this device can be used to divide the input frequency by 2.

The most commonly used T flip-flops are J-K flip-flops wired to perform a toggle function. This use will be demonstrated later in this section.

Synchronous Counter

Synchrounous generally refers to something which is cordinated with others based on time. Synchronous signals occur at same clock rate and all the clocks follow the same reference clock.
In previous tutorial of Asynchronous Counter, we have seen that the output of that counter is directly connected to the input of next subsequent counter and making a chain system, and due to this chain system propagation delay appears during counting stage and create counting delays. **In synchronous counter**, the clock input across all the flip-flops use the same source and create the same clock signal at the same time. So, a counter which is using the same clock signal from the same source at the same time is called **Synchronous counter**.

Synchronous Up Counter



In the above image, the basic Synchronous counter design is shown which is **Synchronous up counter**. A **4-bit Synchronous up counter** start to count from 0 (0000 in binary) and increment or count upwards to 15 (1111 in binary) and then start new counting cycle by getting reset. Its operating frequency is much higher than the same range Asynchronous counter. Also, there is **no propagation delay** in the synchronous counter just because all flip-flops or counter stage is in parallel clock source and the clock triggers all counters at the same time.

The external clock is directly provided to all J-K Flip-flops at the same time in a parallel way. If we see the circuit, **the first flip-flop**, FFA which is the least significant bit in this 4-bit synchronous counter, is connected to a Logic 1 external input via J and K pin. Due to this connection, **HIGH** logic across the Logic 1 signal, change the state of first flip-flop on every clock pulse.

Next stage, **the second flip-flop FFB**, input pin of J and K is connected across the output of the first Flip-flop. For the case of FFC and FFD, two separate AND gate provide the necessary logic across them. Those AND gates create logic using the input and output from the previous stage flip-flops.
We can create the same counting sequence used in the Asynchronous counter by making a situation where each flip-flops change its state depending on whether or not all preceding flip-flops output is HIGH in logic. But in this scenario, there will be no ripple effect just because all flip-flops are clocked at the same time.

Asynchronous Counters

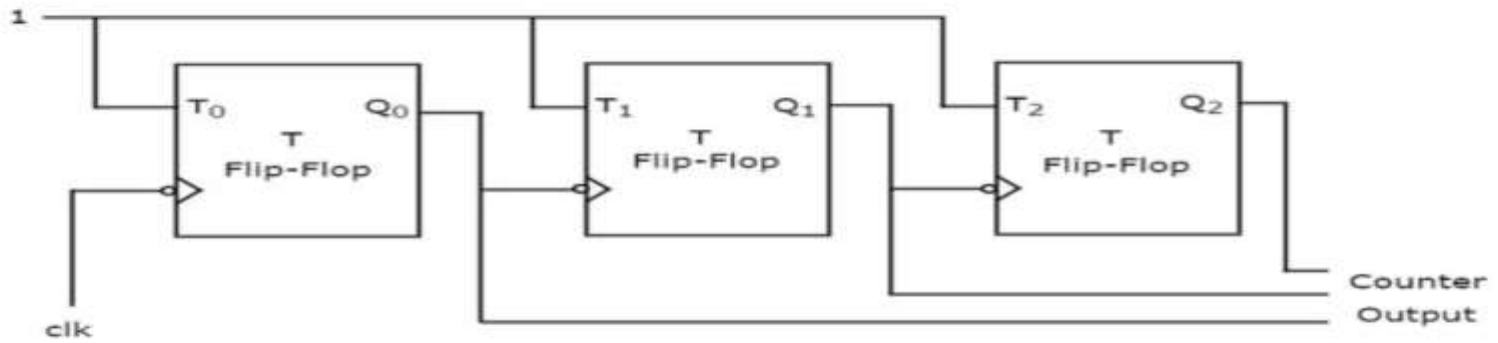
If the flip-flops do not receive the same clock signal, then that counter is called as **Asynchronous counter**.
The output of system clock is applied as clock signal only to first flip-flop.
The remaining flip-flops receive the clock signal from output of its previous stage flip-flop.
Hence, the outputs of all flip-flops do not change affectaffect at the same time.

Now, let us discuss the following two counters one by one.

- Asynchronous Binary up counter
- Asynchronous Binary down counter

Asynchronous Binary Up Counter

An 'N' bit Asynchronous binary up counter consists of 'N' T flip-flops. It counts from 0 to $2^N - 1$. The **block diagram** of 3-bit Asynchronous binary up counter is shown in the following figure.



The 3-bit Asynchronous binary up counter contains three T flip-flops and the T-input of all the flip-flops are connected to '1'. All these flip-flops are negative edge triggered but the outputs change asynchronously. The clock signal is directly applied to the first T flip-flop. So, the output of first T flip-flop **toggles** for every negative edge of clock signal.

The output of first T flip-flop is applied as clock signal for second T flip-flop. So, the output of second T flip-flop toggles for every negative edge of output of first T flip-flop. Similarly, the output of third T flip-flop toggles for every negative edge of output of second T flip-flop, since the output of second T flip-flop acts as the clock signal for third T flip-flop.

Assume the initial status of T flip-flops from rightmost to leftmost is $Q_2Q_1Q_0 = 000$. Here, Q_2 & Q_0 are MSB & LSB respectively. We can understand the **working** of 3-bit asynchronous binary counter from the following table.

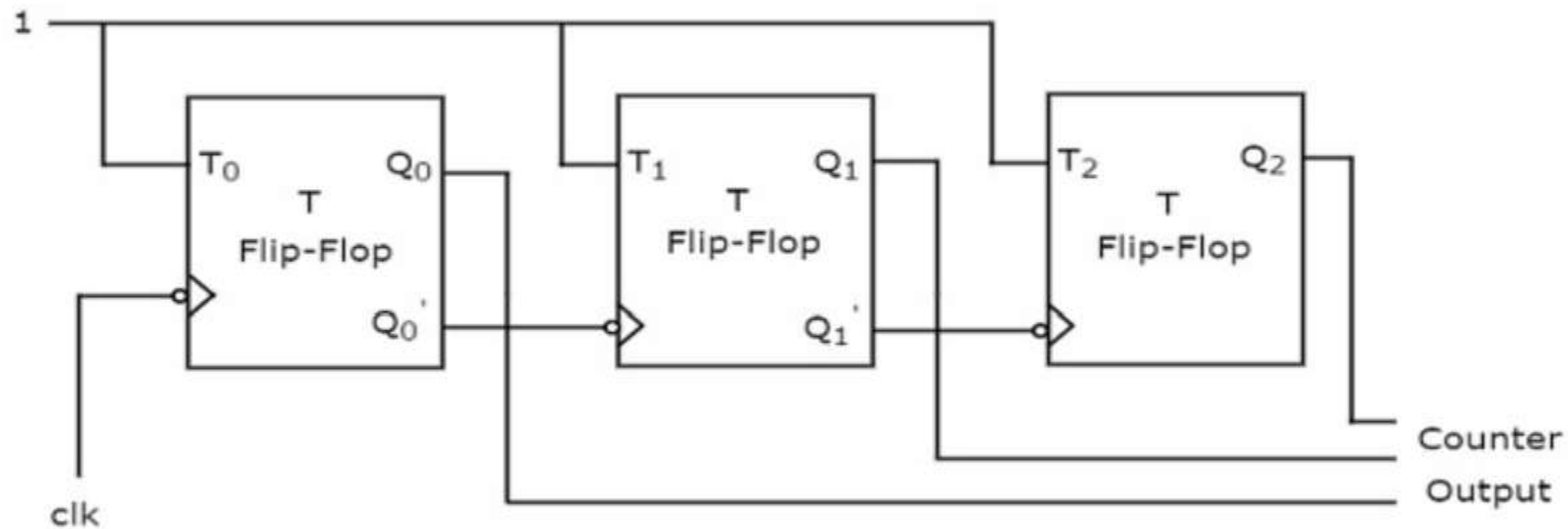
No of negative edge of Clock	Q_0 LSB	Q_1	Q_2 MSB
0	0	0	0
1	1	0	0
2	0	1	0
3	1	1	0
4	0	0	1
5	1	0	1
6	0	1	1
7	1	1	1

Here Q_0 toggled for every negative edge of clock signal. Q_1 toggled for every Q_0 that goes from 1 to 0, otherwise remained in the previous state. Similarly, Q_2 toggled for every Q_1 that goes from 1 to 0, otherwise remained in the previous state.

The initial status of the T flip-flops in the absence of clock signal is $Q_2Q_1Q_0 = 000$. This is incremented by one for every negative edge of clock signal and reached to maximum value at 7^{th} negative edge of clock signal. This pattern repeats when further negative edges of clock signal are applied.

Asynchronous Binary Down Counter

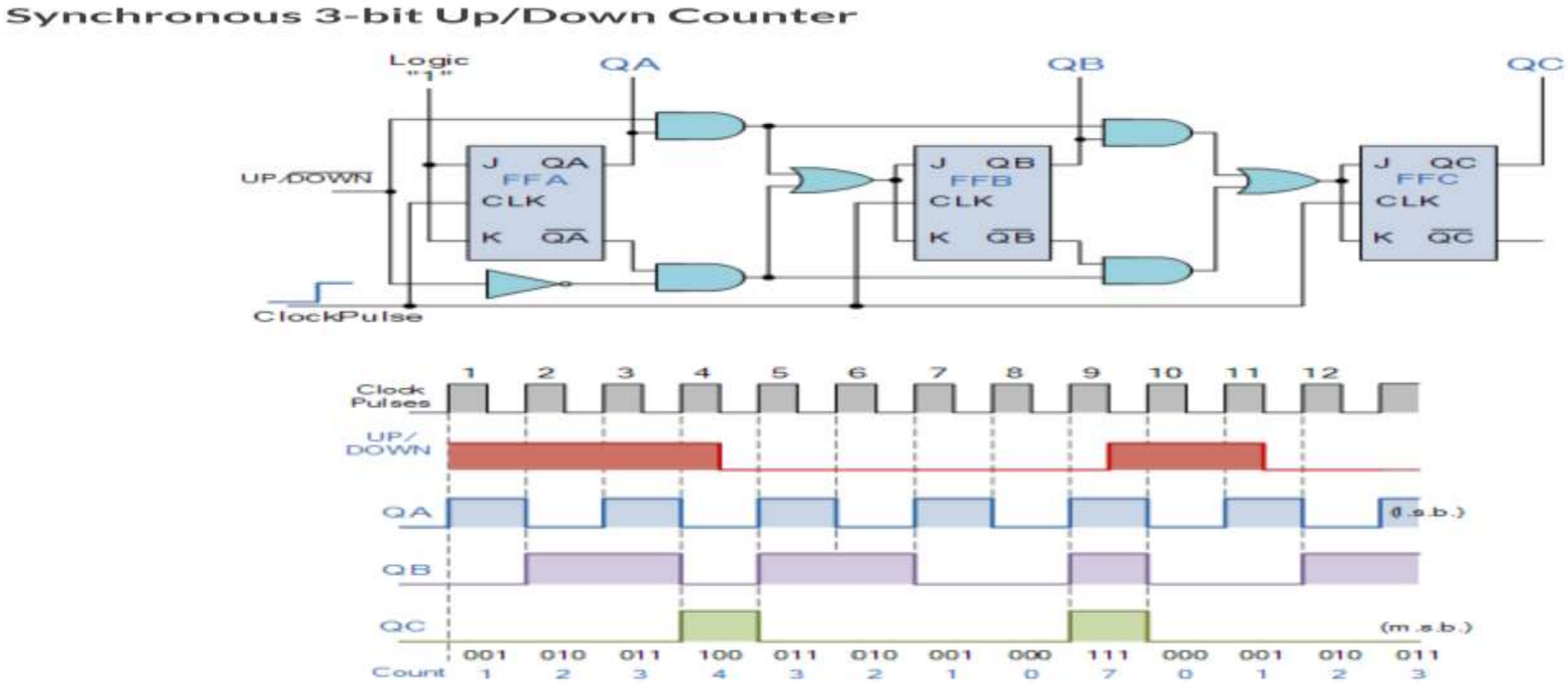
An 'N' bit Asynchronous binary down counter consists of 'N' T flip-flops. It counts from $2^N - 1$ to 0. The **block diagram** of 3-bit Asynchronous binary down counter is shown in the following figure.



The block diagram of 3-bit Asynchronous binary down counter is similar to the block diagram of 3-bit Asynchronous binary up counter. But, the only difference is that instead of connecting the normal outputs of one stage flip-flop as clock signal for next stage flip-flop, connect the **complemented outputs** of one stage flip-flop as clock signal for next stage flip-flop. Complemented output goes from 1 to 0 is same as the normal output goes from 0 to 1.

Bidirectional Counter

Both Synchronous and Asynchronous counters are capable of counting “Up” or counting “Down”, but their is another more “Universal” type of counter that can count in both directions either Up or Down depending on the state of their input control pin and these are known as **Bidirectional Counters**. *Bidirectional counters*, also known as Up/Down counters, are capable of counting in either direction through any given count sequence and they can be reversed at any point within their count sequence by using an additional control input as shown below.



UNIT-IV

8085 microprocessor Review (brief details only), 8086 microprocessor, Functional Diagram, register organization 8086, Flag register of 8086 and its functions, Addressing modes of 8086, Pin diagram of 8086 , Minimum mode & Maximum mode operation of 8086, Interrupts in 8086.

8085 is pronounced as "eighty-eighty-five" microprocessor. It is an 8-bit microprocessor designed by Intel in 1977 using NMOS technology.

It has the following configuration −

- 8-bit data bus
- 16-bit address bus, which can address upto 64KB
- A 16-bit program counter
- A 16-bit stack pointer
- Six 8-bit registers arranged in pairs: BC, DE, HL
- Requires +5V supply to operate at 3.2 MHZ single phase clock

It is used in washing machines, microwave ovens, mobile phones, etc.

8085 Microprocessor – Functional Units

8085 consists of the following functional units −

Accumulator

It is an 8-bit register used to perform arithmetic, logical, I/O & LOAD/STORE operations. It is connected to internal data bus & ALU.

Arithmetic and logic unit

As the name suggests, it performs arithmetic and logical operations like Addition, Subtraction, AND, OR, etc. on 8-bit data.

General purpose register

There are 6 general purpose registers in 8085 processor, i.e. B, C, D, E, H & L. Each register can hold 8-bit data.

These registers can work in pair to hold 16-bit data and their pairing combination is like B-C, D-E & H-L.

Program counter

It is a 16-bit register used to store the memory address location of the next instruction to be executed. Microprocessor increments the program whenever an instruction is being executed, so that the program counter points to the memory address of the next instruction that is going to be executed.

Stack pointer

It is also a 16-bit register works like stack, which is always incremented/decremented by 2 during push & pop operations.

Temporary register

It is an 8-bit register, which holds the temporary data of arithmetic and logical operations.

Flag register

It is an 8-bit register having five 1-bit flip-flops, which holds either 0 or 1 depending upon the result stored in the accumulator.

These are the set of 5 flip-flops −

- Sign (S)
- Zero (Z)
- Auxiliary Carry (AC)
- Parity (P)
- Carry (C)

Its bit position is shown in the following table −

D7	D6	D5	D4	D3	D2	D1	D0
S	Z		AC		P		CY

Instruction register and decoder

It is an 8-bit register. When an instruction is fetched from memory then it is stored in the Instruction register. Instruction decoder decodes the information present in the Instruction register.

Timing and control unit

It provides timing and control signal to the microprocessor to perform operations. Following are the timing and control signals, which control external and internal circuits −

- Control Signals: READY, RD', WR', ALE
- Status Signals: S0, S1, IO/M'
- DMA Signals: HOLD, HLDA
- RESET Signals: RESET IN, RESET OUT

Interrupt control

As the name suggests it controls the interrupts during a process. When a microprocessor is executing a main program and whenever an interrupt occurs, the microprocessor shifts the control from the main program to process the incoming request. After the request is completed, the control goes back to the main program.

There are 5 interrupt signals in 8085 microprocessor: INTR, RST 7.5, RST 6.5, RST 5.5, TRAP.

8086 microprocessor

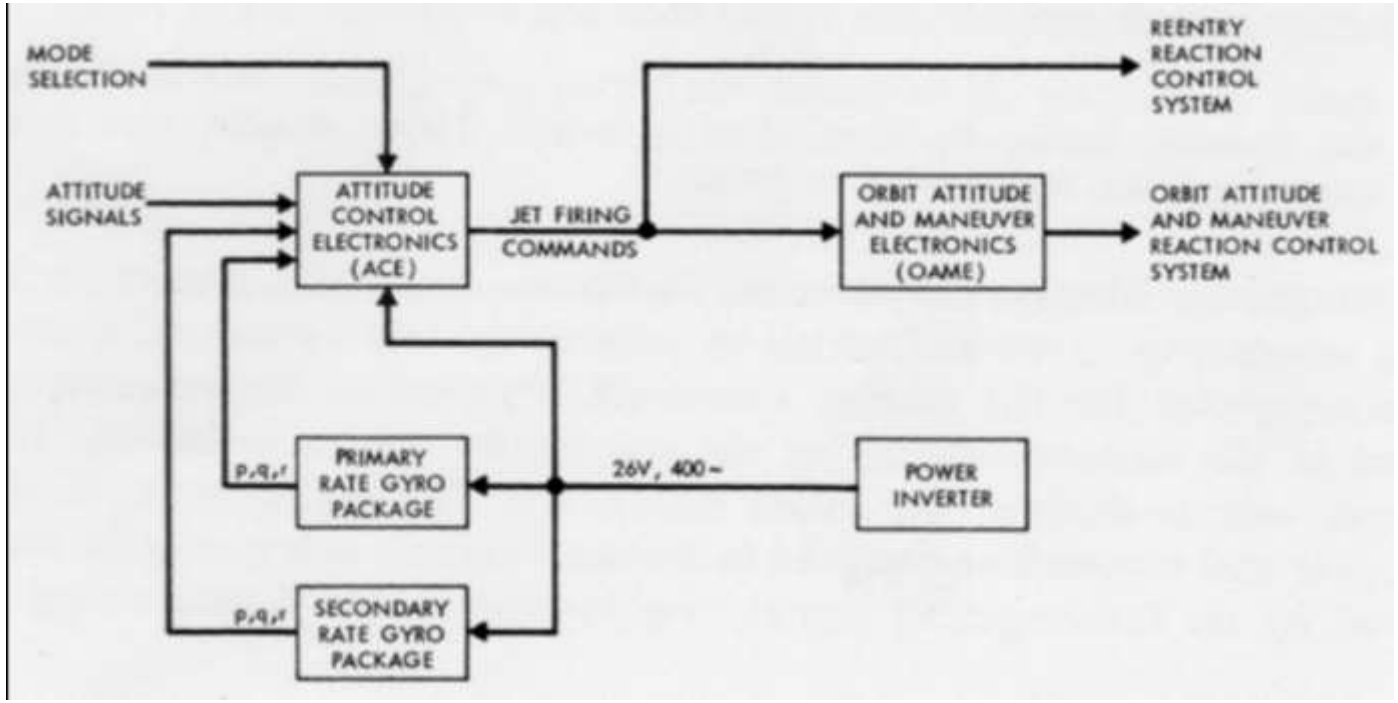
Intel 8086

- Intel 8086 microprocessor is the enhanced version of Intel 8085 microprocessor. It was designed by Intel in 1976.
- The 8086 microprocessor is a16-bit, N-channel, HMOS microprocessor. Where the HMOS is used for "**High-speed Metal Oxide Semiconductor**".
- Intel 8086 is built on a single semiconductor chip and packaged in a 40-pin IC package. The type of package is DIP (Dual Inline Package).
- Intel 8086 uses 20 address lines and 16 data- lines. It can directly address up to 2^{20} = 1 Mbyte of memory.
- It consists of a powerful instruction set, which provides operation like division and multiplication very quickly.
- 8086 is designed to operate in two modes, i.e., Minimum and Maximum mode.

Difference between 8085 and 8086 Microprocessor

8085 Microprocessor	8086 Microprocessor
It is an 8-bit microprocessor.	It is a 16-bit microprocessor.
It has a 16-bit address line.	It has a 20-bit address line.
It has a 8-bit data bus.	It has a 16-bit data bus.
The memory capacity is 64 KB.	The memory capacity is 1 MB.
The Clock speed of this microprocessor is 3 MHz.	The Clock speed of this microprocessor varies between 5, 8 and 10 MHz for different versions.
It has five flags.	It has nine flags.
8085 microprocessor does not support memory segmentation.	8086 microprocessor supports memory segmentation.
It does not support pipelining.	It supports pipelining.
It is accumulator based processor.	It is general purpose register based processor.
It has no minimum or maximum mode.	It has minimum and maximum modes.
In 8085, only one processor is used.	In 8086, more than one processor is used. An additional external processor can also be employed.
It contains less number of transistors compare to 8086 microprocessor. It contains about 6500 transistor.	It contains more number of transistors compare to 8085 microprocessor. It contains about 29000 in size.
The cost of 8085 is low.	The cost of 8086 is high.

Functional Diagram



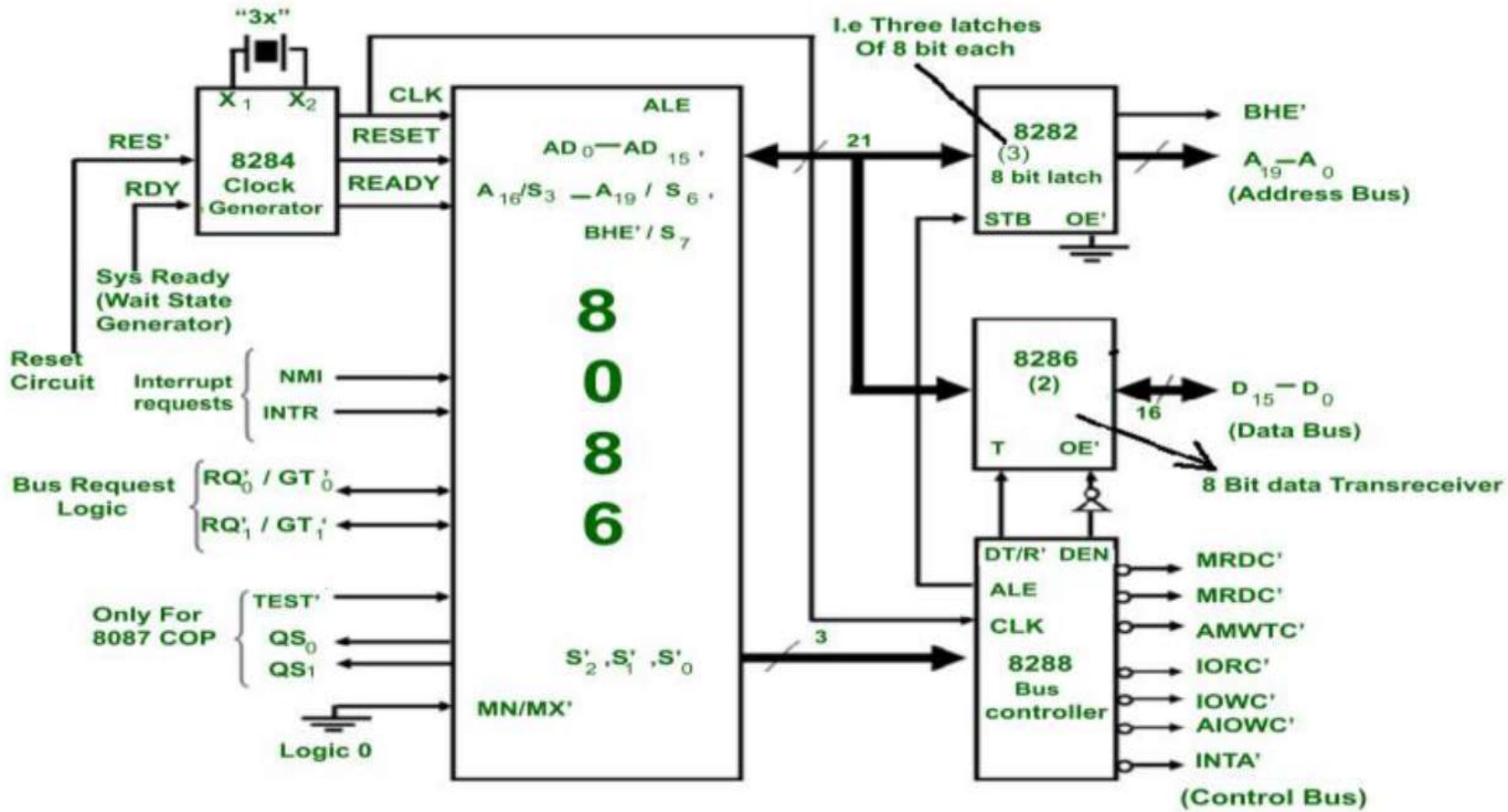
Maximum mode configuration of 8086 microprocessor (Max mode)

8086 microprocessor characteristics:

- It contains 20 bit address bus.
- It contains 16-bit data bus, therefore 8086 is called as **16-bit microprocessor.**
- It is 2-stage pipelined processor. It can prefetch 6 bytes from memory and store into queue to increase the speed of the execution.
- It's control bus carries signals for executing operations such as read ,write etc.
- It has Memory Banks. 2 banks of 512KB each. These banks are called as lower Bank (even) and higher Bank (odd).
- In 8086 the entire memory is divided into four memory segments which are code ,stack, data and extra segment.
- 8086 has 16 bit IO address.
- It has 256 interrupts.

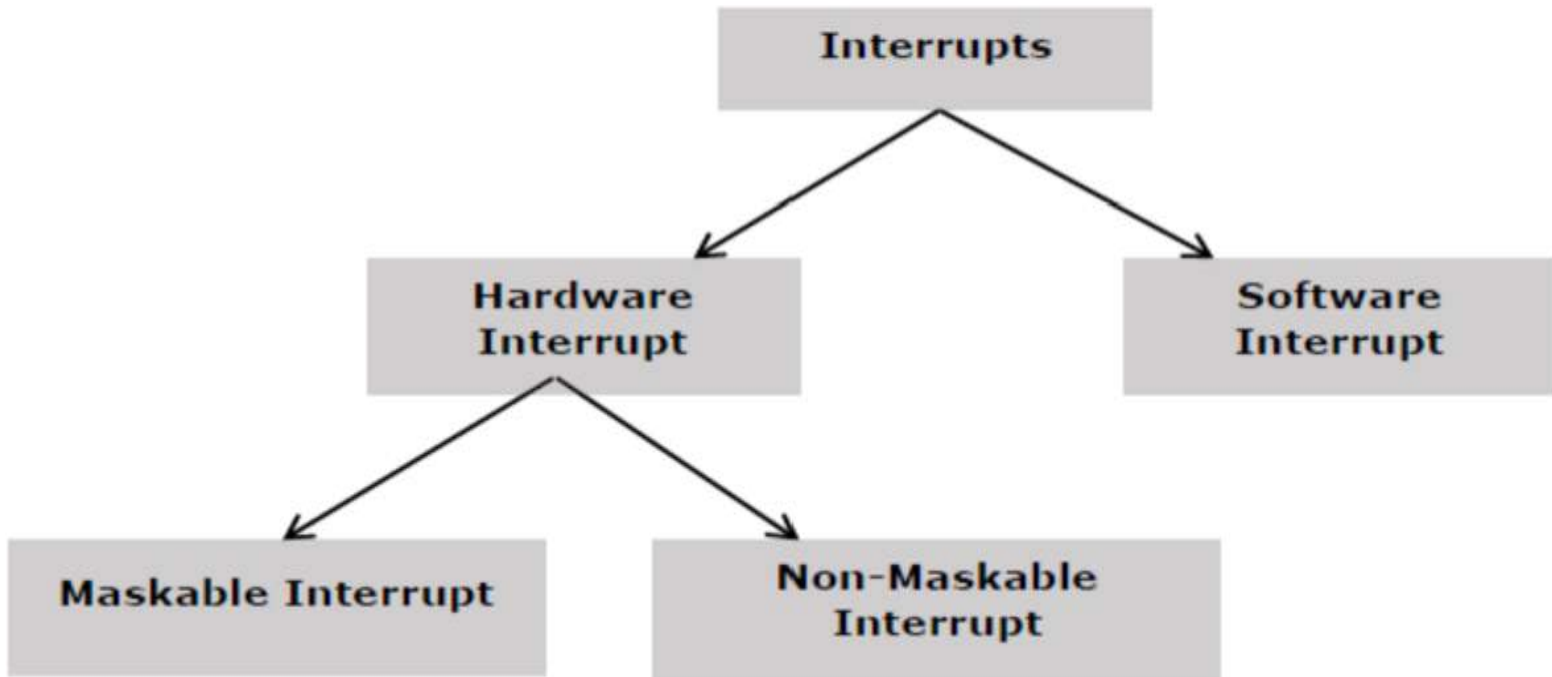
8086 has two operating Modes:

1. Minimum mode
2. Maximum mode


"3x"
X1
X2
8284
Clock
Generator
RES'
RDY
Sys Ready
(Wait State
Generator)
Reset
Circuit
CLK
RESET
READY
Interrupt
requests
NMI
INTR
Bus Request
Logic
RQ'0 / GT'0
RQ'1 / GT'1
Only For
8087 COP
TEST'
QS0
QS1
Logic 0
ALE
AD0—AD15 '
A16/S3 —A19 / S6 '
BHE' / S7
8086
S'2 ,S'1 ,S'0
MN/MX'
21
3
I.e Three latches
Of 8 bit each
8282
(3)
8 bit latch
STB
OE'
BHE'
A19—A0
(Address Bus)
8286
(2)
T
OE'
16
D15—D0
(Data Bus)
8 Bit data Transreceiver
DT/R'
DEN
ALE
CLK
8288
Bus
controller
MRDC'
MRDC'
AMWTC'
IORC'
IOWC'
AIOWC'
INTA'
(Control Bus)

Interrupt is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor. The microprocessor responds to that interrupt with an **ISR** (Interrupt Service Routine), which is a short program to instruct the microprocessor on how to handle the interrupt.

The following image shows the types of interrupts we have in a 8086 microprocessor −



Hardware Interrupts

Hardware interrupt is caused by any peripheral device by sending a signal through a specified pin to the microprocessor.

The 8086 has two hardware interrupt pins, i.e. NMI and INTR. NMI is a non-maskable interrupt and INTR is a maskable interrupt having lower priority. One more interrupt pin associated is INTA called interrupt acknowledge.

NMI

It is a single non-maskable interrupt pin (NMI) having higher priority than the maskable interrupt request pin (INTR)and it is of type 2 interrupt.

When this interrupt is activated, these actions take place −

- Completes the current instruction that is in progress.
- Pushes the Flag register values on to the stack.
- Pushes the CS (code segment) value and IP (instruction pointer) value of the return address on to the stack.
- IP is loaded from the contents of the word location 00008H.
- CS is loaded from the contents of the next word location 0000AH.
- Interrupt flag and trap flag are reset to 0.

INTR

The INTR is a maskable interrupt because the microprocessor will be interrupted only if interrupts are enabled using set interrupt flag instruction. It should not be enabled using clear interrupt Flag instruction.

The INTR interrupt is activated by an I/O port. If the interrupt is enabled and NMI is disabled, then the microprocessor first completes the current execution and sends ‘0’ on INTA pin twice. The first ‘0’ means INTA informs the external device to get ready and during the second ‘0’ the microprocessor receives the 8 bit, say X, from the programmable interrupt controller.

These actions are taken by the microprocessor −

- First completes the current instruction.
- Activates INTA output and receives the interrupt type, say X.
- Flag register value, CS value of the return address and IP value of the return address are pushed on to the stack.
- IP value is loaded from the contents of word location X × 4
- CS is loaded from the contents of the next word location.
- Interrupt flag and trap flag is reset to 0

UNIT-V

Instruction set of 8086, Assembler directives, Procedures and Macros, Simple programs involving arithmetic, logical, branch instructions, Ascending, Descending and Block move programs, String Manipulation Instructions. Overview of 8051 microcontroller, Architecture, I/O ports and Memory organization, addressing modes and instruction set of 8051(Brief details only), Simple Programs.

Instruction Set of 8086

Instructions are classified on the basis of functions they perform.
They are categorized into the following main types:

Data Transfer instruction

All the instructions which perform data movement come under this category.
The source data may be a register, memory location, port etc. the destination may be a register, memory location or port. The following instructions come under this category:

Instruction	Description
MOV	Moves data from register to register, register to memory, memory to register, memory to accumulator, accumulator to memory, etc.
LDS	Loads a word from the specified memory locations into specified register. It also loads a word from the next two memory locations into DS register.
LES	Loads a word from the specified memory locations into the specified register. It also loads a word from next two memory locations into ES register.
LEA	Loads offset address into the specified register.
LAHF	Loads low order 8-bits of the flag register into AH register.
SAHF	Stores the content of AH register into low order bits of the flags register.
XLAT/XLATB	Reads a byte from the lookup table.
XCHG	Exchanges the contents of the 16-bit or 8-bit specified register with the contents of AX register, specified register or memory locations.
PUSH	Pushes (sends, writes or moves) the content of a specified register or memory location(s) onto the top of the stack.
POP	Pops (reads) two bytes from the top of the stack and keeps them in a specified register, or memory location(s).
POPF	Pops (reads) two bytes from the top of the stack and keeps them in the flag register.
IN	Transfers data from a port to the accumulator or AX, DX or AL register.
OUT	Transfers data from accumulator or AL or AX register to an I/O port identified by the second byte of the instruction.

Logical Instructions

Instruction of this group perform logical AND, OR, XOR, NOT and TEST operations. **The following instructions come under this category:**

Instruction	Description
AND	Performs bit by bit logical AND operation of two operands and places the result in the specified destination.
OR	Performs bit by bit logical OR operation of two operands and places the result in the specified destination.
XOR	Performs bit by bit logical XOR operation of two operands and places the result in the specified destination.
NOT	Takes one's complement of the content of a specified register or memory location(s).
TEST	Perform logical AND operation of a specified operand with another specified operand.

Rotate Instructions

The following instructions come under this category:

Instruction	Description
RCL	Rotate all bits of the operand left by specified number of bits through carry flag.
RCR	Rotate all bits of the operand right by specified number of bits through carry flag.
ROL	Rotate all bits of the operand left by specified number of bits.
ROR	Rotate all bits of the operand right by specified number of bits.

Macro and Procedure

1. Macro :

Macro is a set of instruction and the programmer can use it anywhere in the program by using its name. It is mainly used to achieve modular programming. So same set of instructions can be used multiple times when ever required by the help of macro. Wherever macro's identifier is used, it is replaced by the actual defined instructions during compilation thereby no calling and return occurs.

Syntax of macro :

```
%macro macro_name number_of_parameters
<macro body>
%endmacro
```

2. Procedure :

Procedures are also like macro, but they are used for large set of instruction when macro is useful for small set of instructions. It contains a set of instructions which performs a specific task. It contains three main parts i.e Procedure name to identify the procedure, procedure body which contains set of instructions, and RET statement which denotes return statement. Unlike macros, procedures follow call-return method thereby achieving true modularity.

S.No.	MACRO	PROCEDURE
01.	Macro definition contains a set of instruction to support modular programming.	Procedure contains a set of instructions which can be called repetitively which can perform a specific task.
02.	It is used for small set of instructions mostly less than ten instructions.	It is used for large set of instructions mostly more than ten instructions.
03.	In case of macro memory requirement is high.	In case of procedure memory requirement is less.
04.	CALL and RET instruction/statements are not required in macro.	CALL and RET instruction/statements are required in procedure.
05.	Assembler directive MACRO is used to define macro and assembler directive ENDM is used to indicate the body is over.	Assembler directive PROC is used to define procedure and assembler directive ENDP is used to indicate the body is over.
06.	Execution time of macro is less as it executes faster than procedure.	Execution time of procedures is high as it executes slower than macro.
07.	Here machine code is created multiple times as each time machine code is generated when macro is called.	Here machine code is created only once, it is generated only once when the procedure is defined.
08.	In a macro parameter is passed as part of statement that calls macro.	In a procedure parameters are passed in registers and memory locations of stack.
09.	Overhead time does not take place as there is no calling and returning.	Overhead time takes place during calling procedure and returning control to calling program.

Branch instructions

The branch instructions are used to change the sequence of instruction execution.

Use branch instructions to change the sequence of instruction execution.

Since all branch instructions are on word boundaries, the processor performing the branch ignores bits 30 and 31 of the generated branch target address. All branch instructions can be used in unprivileged state.

A branch instruction computes the target address in one of four ways:

- Target address is the sum of a constant and the address of the branch instruction itself.
- Target address is the absolute address given as an operand to the instruction.
- Target address is the address found in the Link Register.
- Target address is the address found in the Count Register.

Using the first two of these methods, the target address can be computed sufficiently ahead of the branch instructions to prefetch instructions along the target path.

Using the third and fourth methods, prefetching instructions along the branch path is also possible provided the Link Register or the Count Register is loaded sufficiently ahead of the branch instruction.

The branch instructions include Branch Unconditional and Branch Conditional. In the various target forms, branch instructions generally either branch unconditionally only, branch unconditionally and provide a return address, branch conditionally only, or branch conditionally and provide a return address. If a branch instruction has the Link bit set to 1, then the Link Register is altered to store the return address for use by an invoked subroutine. The return address is the address of the instruction immediately following the branch instruction.

The assembler supports various extended mnemonics for branch instructions that incorporate the BO field only or the BO field and a partial BI field into the mnemonics.

String manipulation instructions in 8086 microprocessor

String is a group of bytes/words and their memory is always allocated in a sequential order. String is either referred as byte string or word string. Here we will see some instructions which are used to manipulate the string related operations.

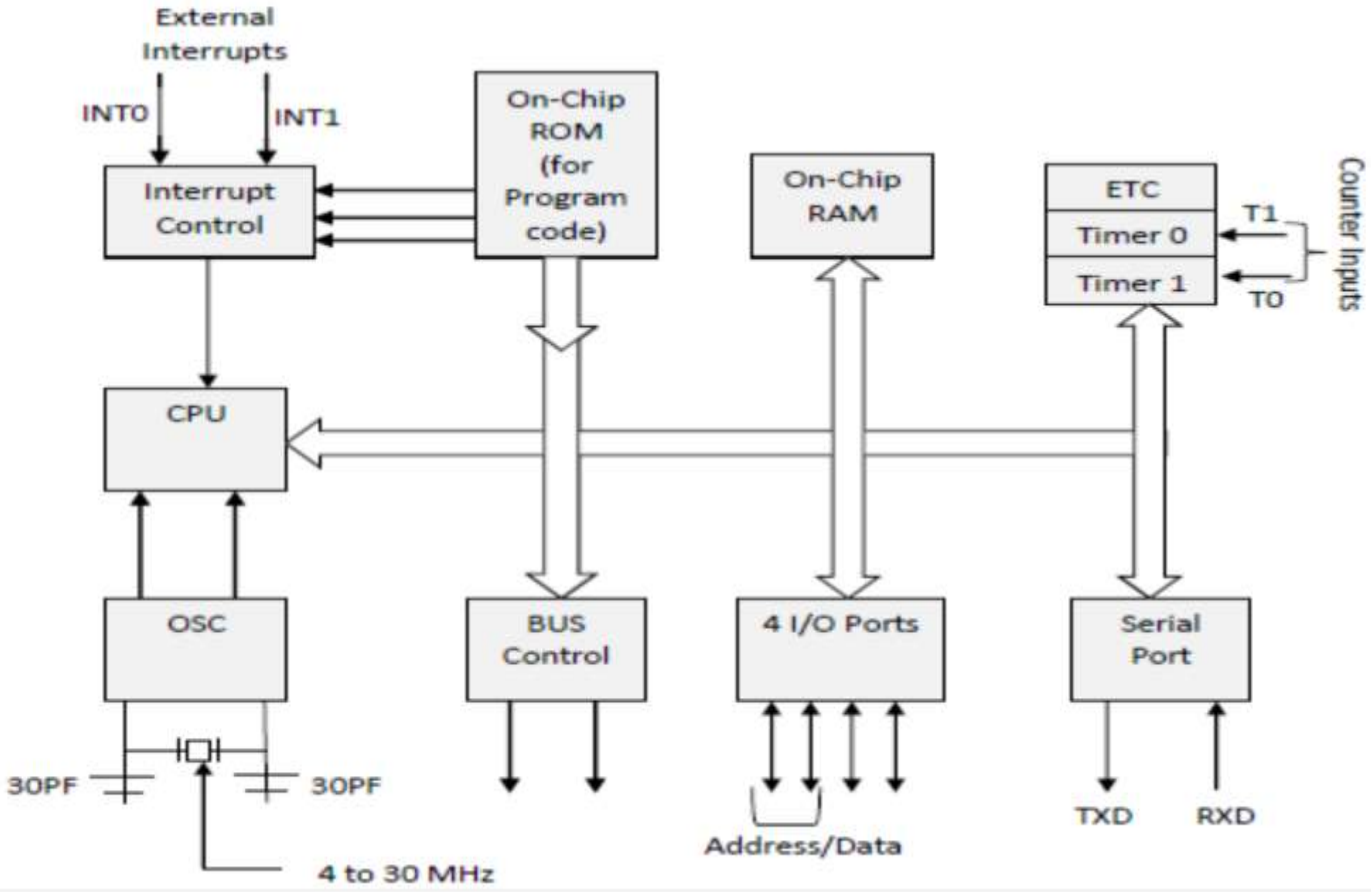
The String manipulation instructions are as follows.

Opcode	Operand	Description
REP	Instruction	Used to repeat the given instruction till CX ≠ 0.
REPE/REPZ	Instruction	Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.
REPNE/REPNZ	Instruction	Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.
MOVS/MOVSB/MOVSW	----	Used to move the byte/word from one string to another.
COMS/COMPSB/COMPSW	----	Used to compare two string bytes/words.
INS/INSB/INSW	----	Used as an input string/byte/word from the I/O port to the provided memory location.
OUTS/OUTSB/OUTSW	----	Used as an output string/byte/word from the provided memory location to the I/O port.
SCAS/SCASB/SCASW	----	Used to scan a string and compare its byte with a byte in AL or string word with a word in AX.
LODS/LODSB/LODSW	----	Used to store the string byte into AL or string word into AX.

Microcontrollers - 8051 Architecture

Let us now discuss the architecture of 8051 Microcontroller.

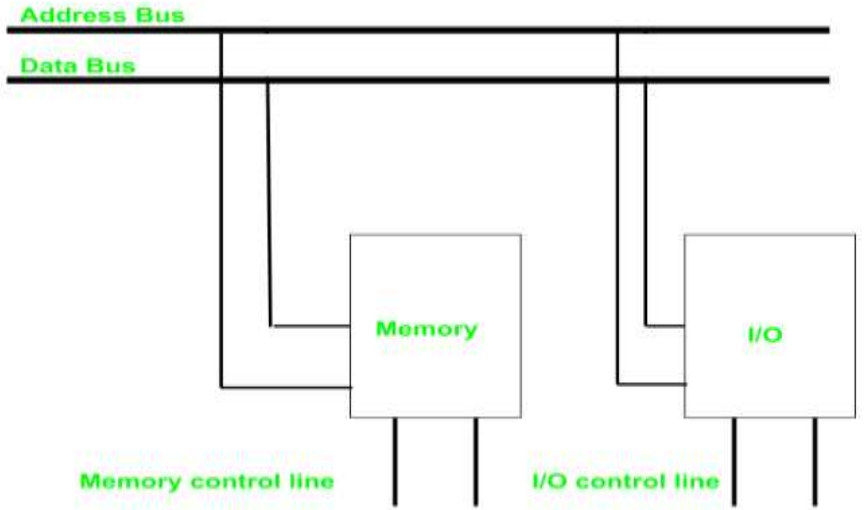
In the following diagram, the system bus connects all the support devices to the CPU. The system bus consists of an 8-bit data bus, a 16-bit address bus and bus control signals. All other devices like program memory, ports, data memory, serial interface, interrupt control, timers, and the CPU are all interfaced together through the system bus.



Memory mapped I/O and Isolated I/O

Isolated I/O –

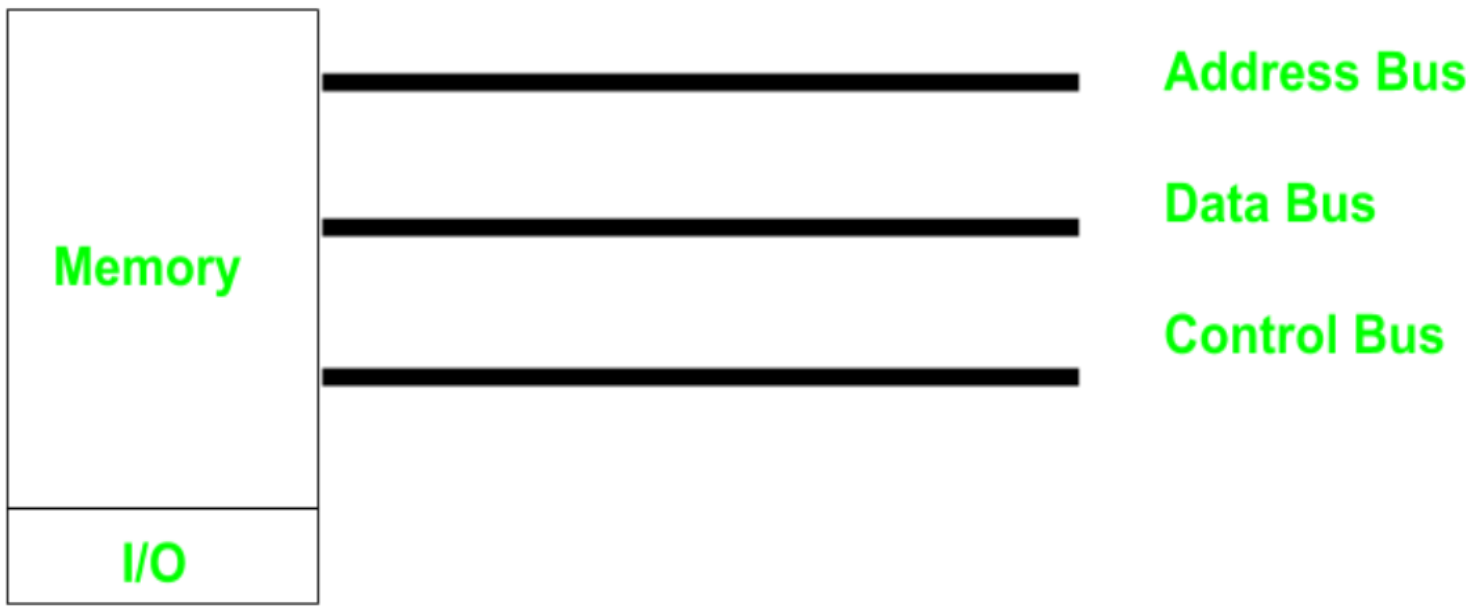
Then we have Isolated I/O in which we Have common bus(data and address) for I/O and memory but separate read and write control lines for I/O. So when CPU decode instruction then if data is for I/O then it places the address on the address line and set I/O read or write control line on due to which data transfer occurs between CPU and I/O. As the address space of memory and I/O is isolated and the name is so. The address for I/O here is called ports. Here we have different read-write instruction for



both I/O and memory.

Memory Mapped I/O –

In this case every bus in common due to which the same set of instructions work for memory and I/O. Hence we manipulate I/O same as memory and both have same address space, due to which addressing capability of memory become less because some part is occupied by the I/O.



There are 5 different ways to execute this instruction and hence we say, we have got 5 addressing modes for 8051.
They are :

1) **Immediate addressing mode**

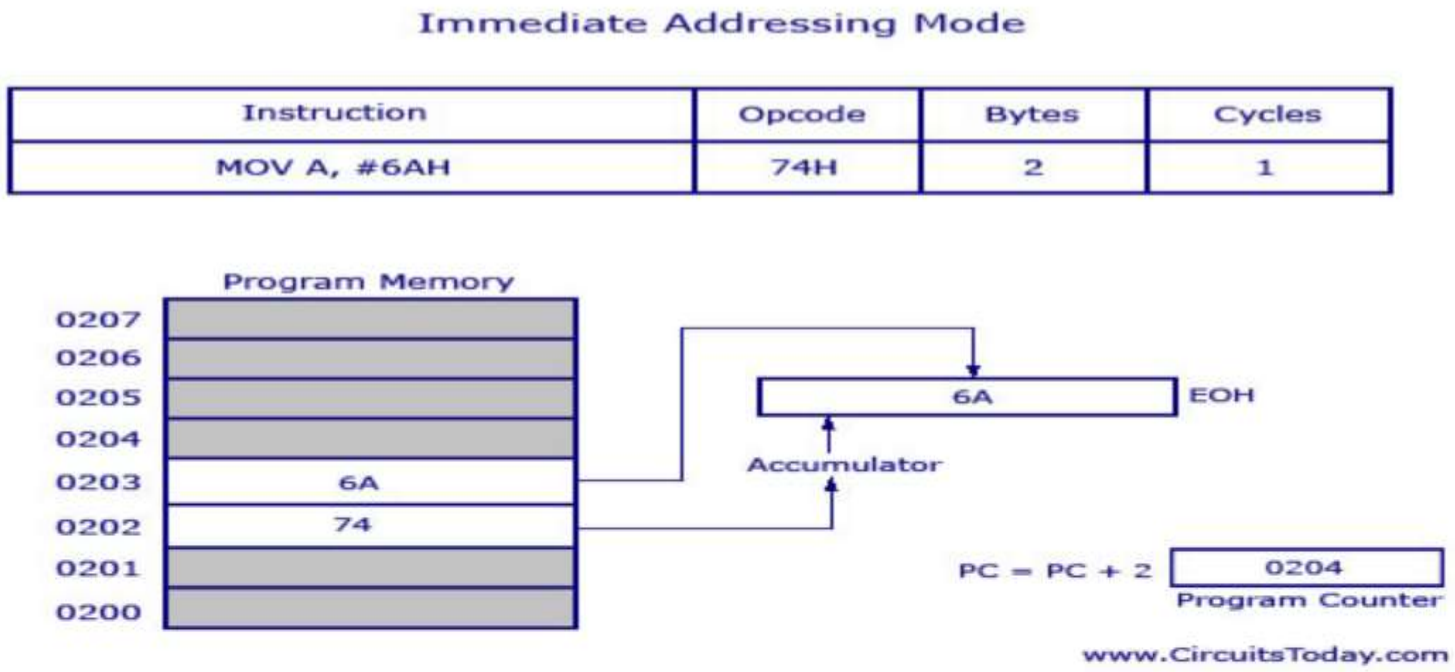
2) Direct addressing mode

3) Register direct addressing mode

4) Register indirect addressing mode

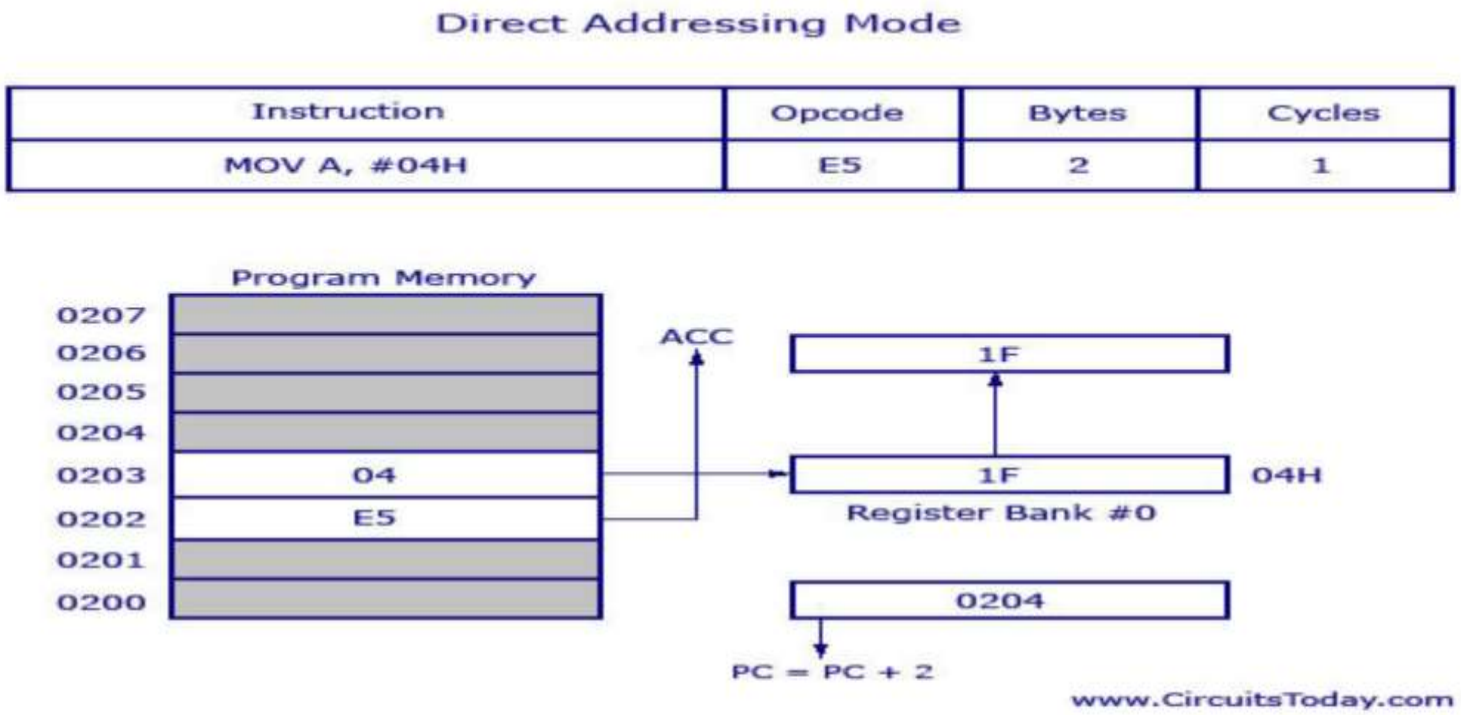
5) Indexed addressing mode.

Immediate Addressing Mode



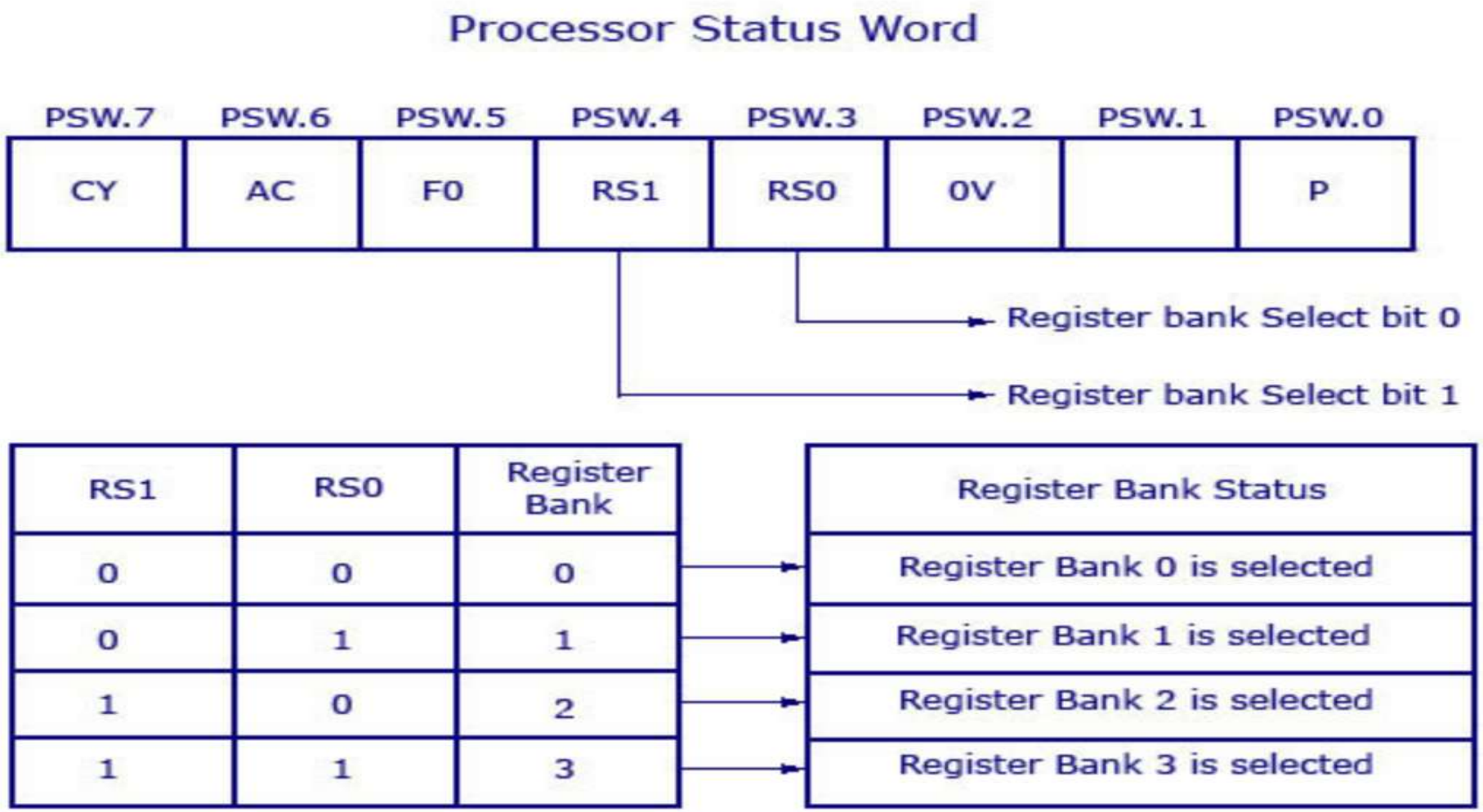
The picture above describes the above instruction and its execution. The opcode for MOV A, # data is 74H. The opcode is saved in program memory at 0202 address. The data 6AH is saved in program memory 0203. (See, any part of the program memory can be used, this is just an example) When the opcode 74H is read, the next step taken would be to transfer whatever data at the next program memory address (here at 0203) to accumulator A (E0H is the address of accumulator). This instruction is of two bytes and is executed in one cycle. So after the execution of this instruction, program counter will add 2 and move to o204 of program memory.

Direct Addressing Mode



As shown in picture above this is a 2 byte instruction which requires 1 cycle to complete. Program counter will increment by 2 and stand in 0204. The opcode for instruction ***MOV A, address*** is E5H. When the instruction at 0202 is executed (E5H), accumulator is made active and ready to receive data. Then program control goes to next address that is 0203 and look up the address of the location (04H) where the source data (to be transferred to accumulator) is located. At 04H the control finds the data 1F and transfers it to accumulator and hence the execution is completed.

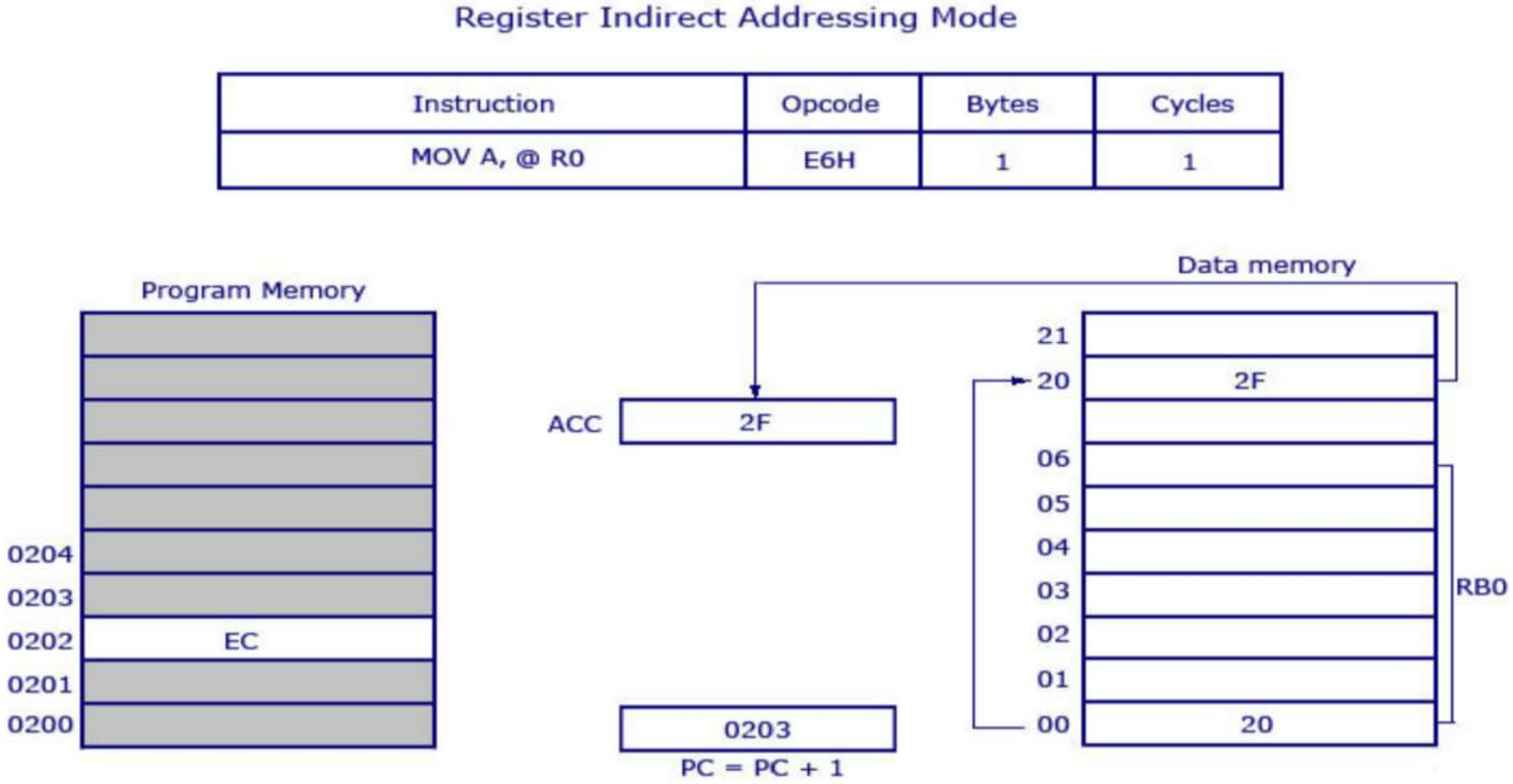
Register Direct Addressing Mode



RS1	RS0	Register Bank
0	0	0
0	1	1
1	0	2
1	1	3

Register Bank Status
Register Bank 0 is selected
Register Bank 1 is selected
Register Bank 2 is selected
Register Bank 3 is selected

Register Indirect Addressing Mode



Indexed Addressing Mode