Introduction to Computer Systems Architecture

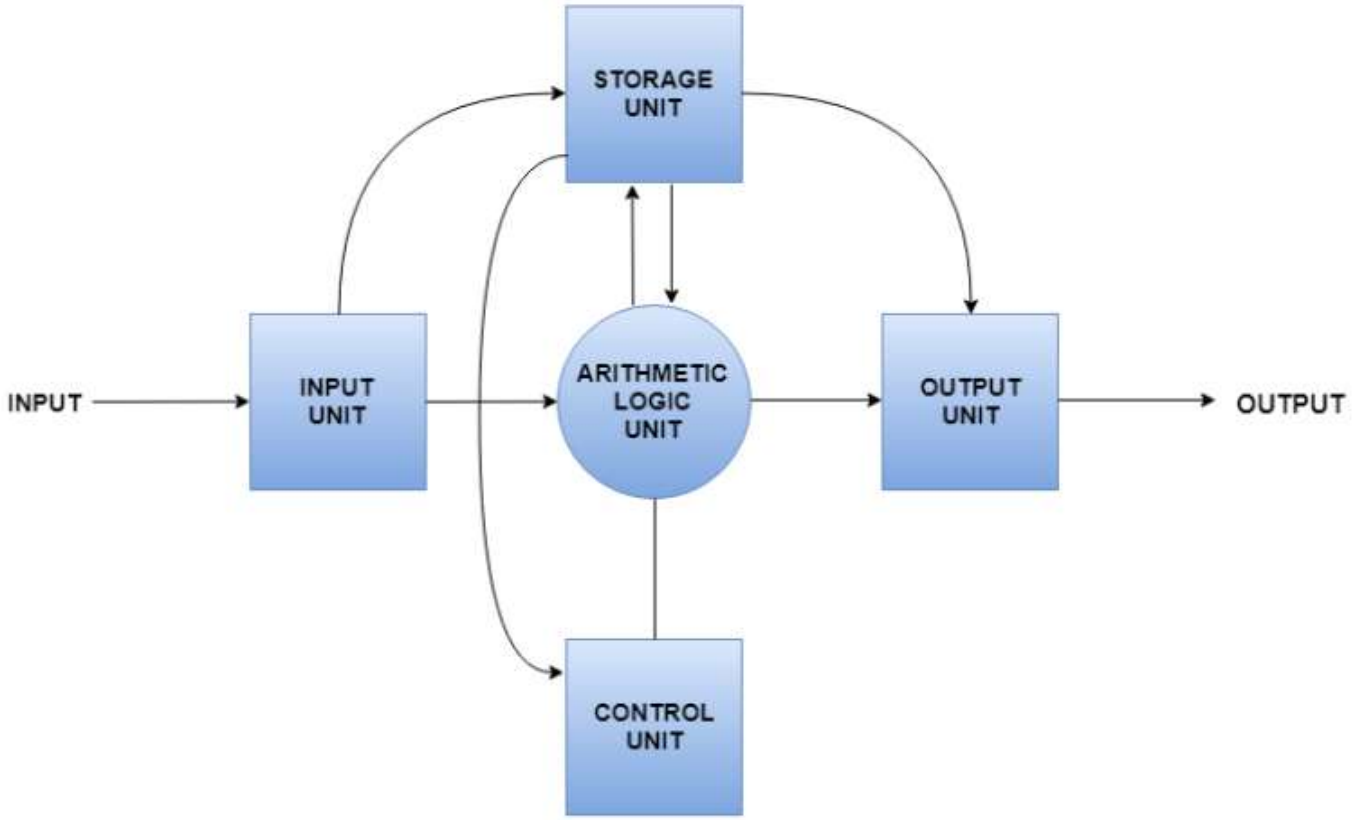
Computer Systems Architecture

- A **computer system** is basically a machine that simplifies complicated tasks.
- **Computer Architecture** is concern with the structure and behaviour of the computer as seen by the user. It includes the information, the instruction set and techniques for addressing memory. The architectural design of a computer system is concerned with the specifications of the functional modules, such as processor, memory and structuring them together into a computer system.

"Computer Architecture is a study of computer systems"

Components in the Computer System Architecture

Input Unit, Output Unit, Storage Unit, Arithmetic Logic Unit and Control Unit etc.



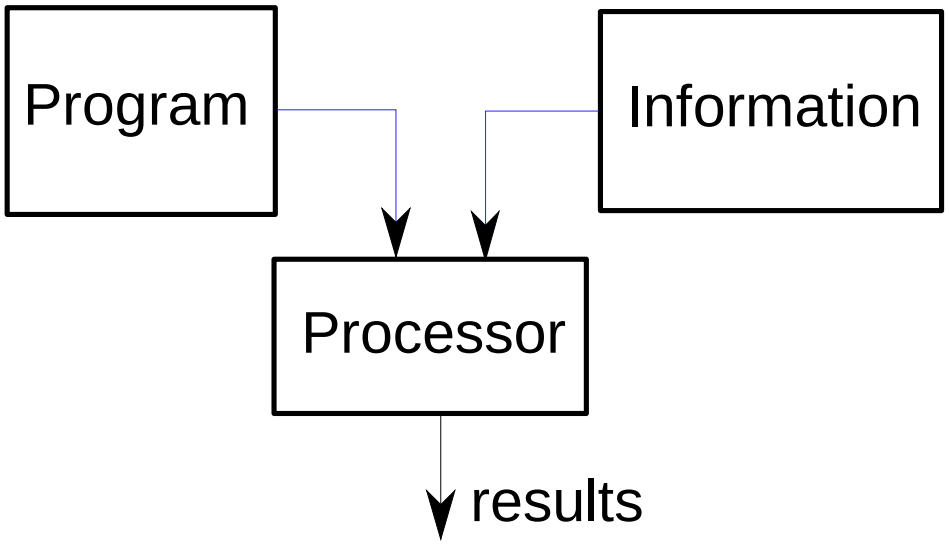
Broad categories

- A computer system is subdivided into two functional entities:
- **Hardware** of the computer system consists of all the electronic components and electromechanical devices that comprise the physical entity of the device.
- **Software** *consists of the instructions* and data that the computer manipulates to perform various data-processing tasks.

Broad categories

- **Hardware** of the computer system is usually divided into three major parts.
  - Input devices
  - CPU (ALU,CU and Memory)
  - Output devices
- **Software** consists of a collection of **programs** whose purpose is to make more efficient use of the computer.
  - **Program:** A set of instructions that directs a computer's hardware to perform a task.
  - **Instructions** are basic commands used to communicate with the processor. A computer can execute billions of instructions per second.

How Computer System Works ?



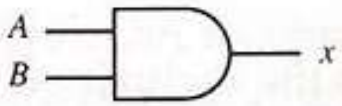
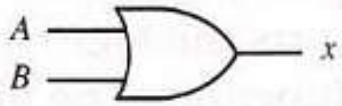
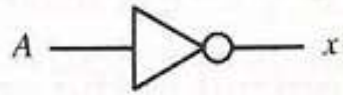
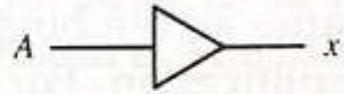
- **Program** – List of instructions given to the computer
- **Information**– **data**, images, files, videos
- **Processor**– Process the information according to the instructions in the program

Binary System & Logic Gates

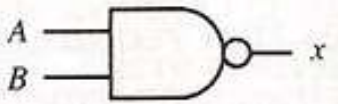
- Computer is a digital device, which works on two levels of signal. These two levels of signal as **High** and **Low**.
- Digital computers use the binary number system, which has two digits: 0 and 1.

  0 means Low

  1 means High
- Manipulation of binary information is done by logic circuits called **Gates**.
- Each gate has a distinct graphic symbol and its operation can be described by means of an algebraic expression.
- Input-Output relationship of the binary variables for each gate can be represented in tabular form by a **truth table**.

Name	*Graphic symbol*	*Algebraic function*	*Truth table*
AND		$x = A \cdot B$ or $x = AB$	A B | x 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$x = A + B$	A B | x 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$x = A'$	A | x 0 | 1 1 | 0
Buffer		$x = A$	A | x 0 | 0 1 | 1

NAND



$x = (AB)'$

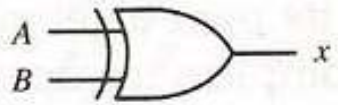
A	B	x
0	0	1
0	1	1
1	0	1
1	1	0

NOR

$x = (A + B)'$

A	B	x
0	0	1
0	1	0
1	0	0
1	1	0

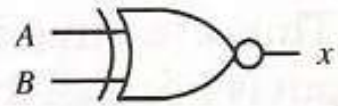
Exclusive-OR (XOR)



$x = A \oplus B$
or
$x = A'B + AB'$

A	B	x
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive-NOR or equivalence



$x = (A \oplus B)'$
or
$x = A'B' + AB$

A	B	x
0	0	1
0	1	0
1	0	0
1	1	1

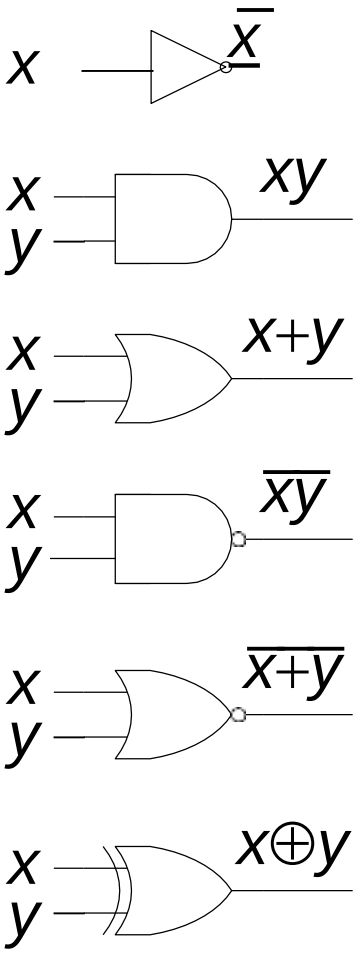
Book

- Morris Mano, Computer System Architecture, 3rd Edition, Pearson

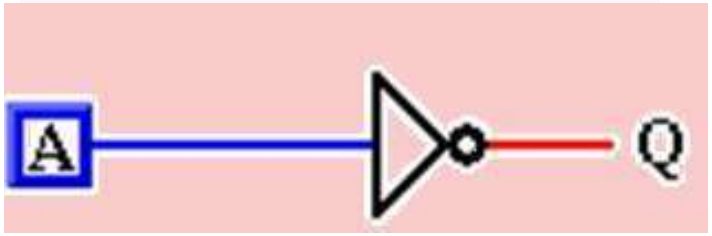
Logic Gates

Basic logic gates

- Not
- And
- Or
- Nand
- Nor
- Xor



Inversion (NOT)

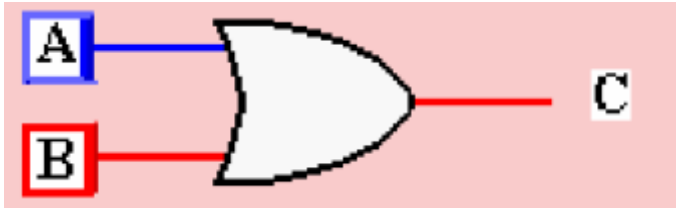


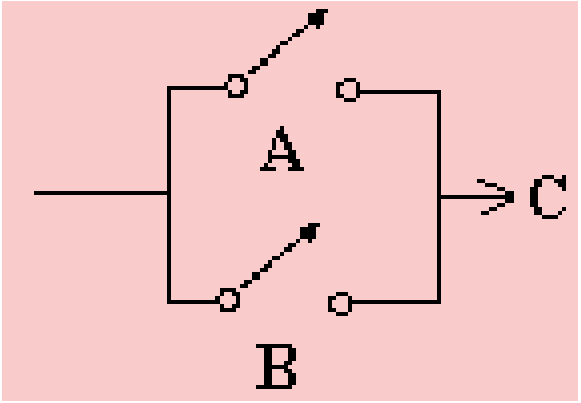
Logic: $Q = \overline{A}$

A	Q
0	1
1	0

OR Gate

- Current flows if either switch is closed
  - Logic notation A + B = C

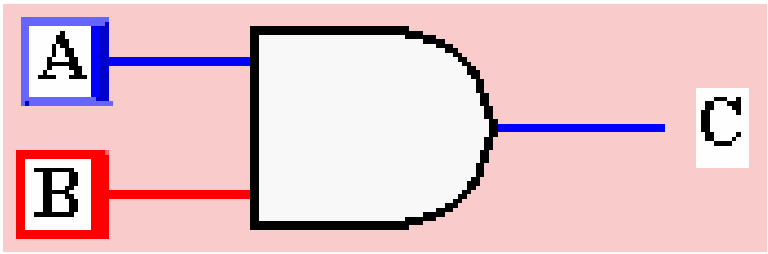


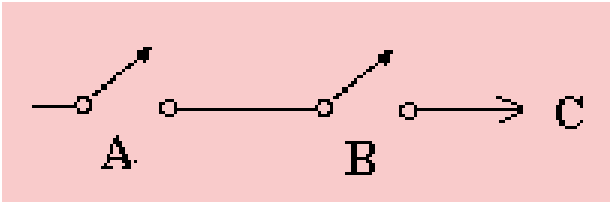


A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

AND Gate

- In order for current to flow, both switches must be closed
  - Logic notation A•B = C
    (Sometimes AB = C)

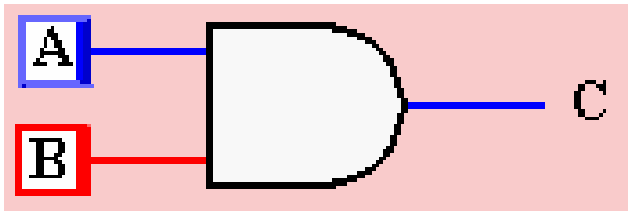




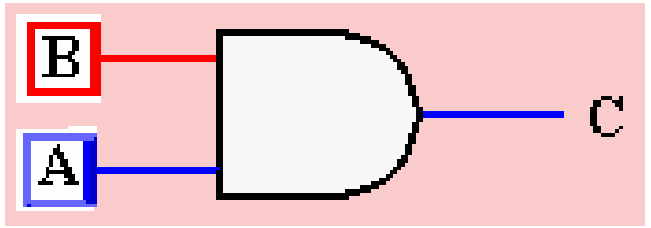
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

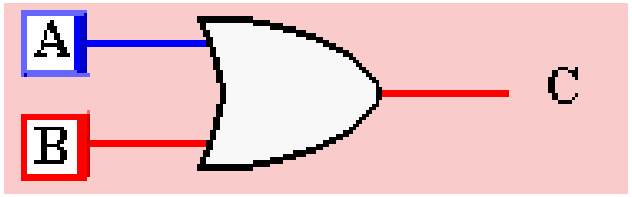
Properties of AND and OR

- Commutation
  - A + B = B + A
  - A • B = B • A

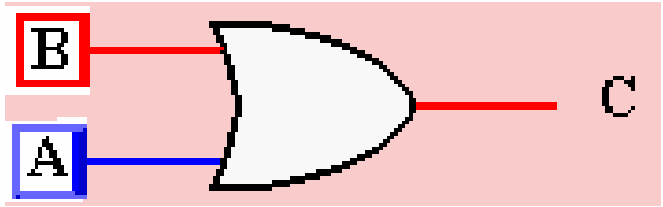


Same as

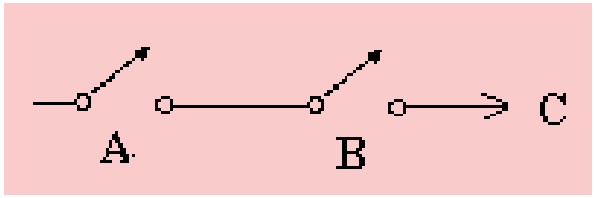




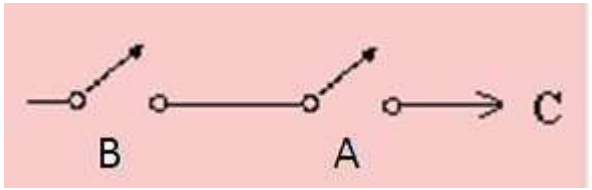
Same as



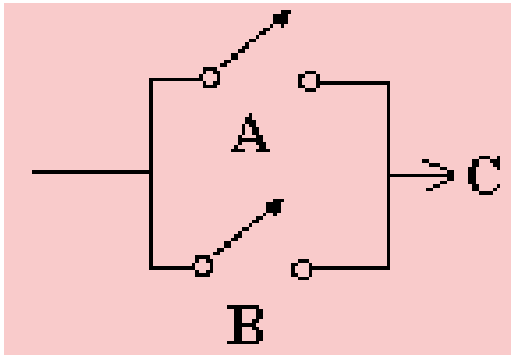
Commutation Circuit



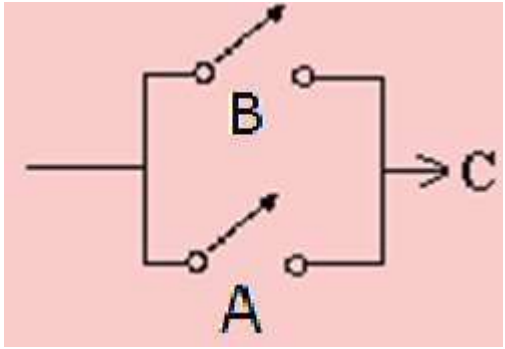
A • B



B • A



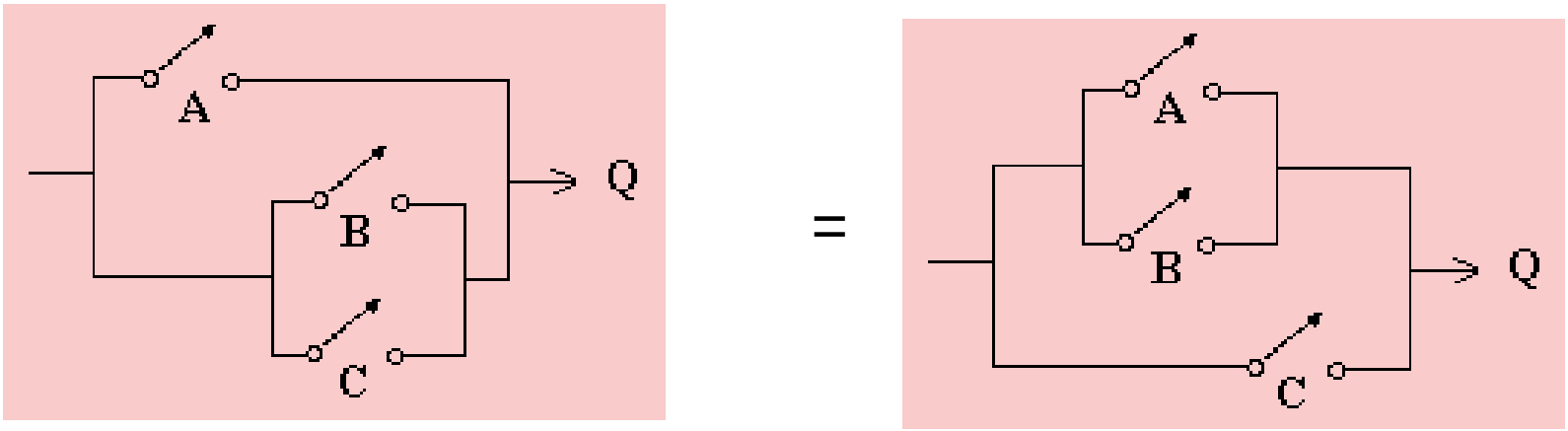
A + B



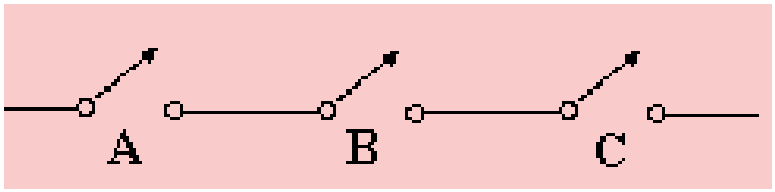
B + A

Properties of AND and OR

- Associative Property
  - ❖ A + (B + C) = (A + B) + C

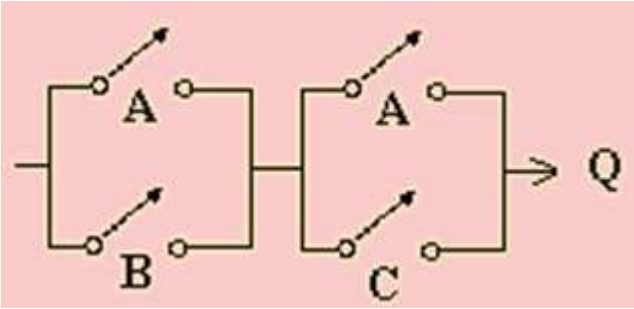


  - ❖ A • (B • C) = (A • B) • C



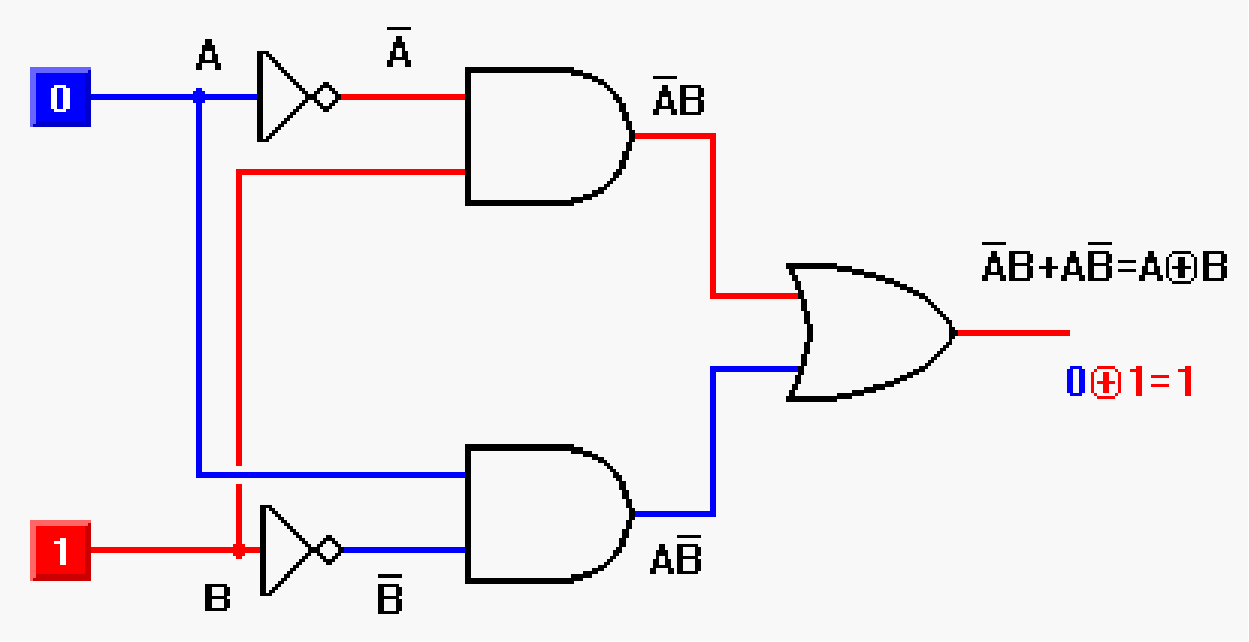
Distributive Property

$(A + B) \bullet (A + C)$



A	B	C	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Circuit for XOR



$$A \oplus B = \overline{A} \cdot B + A \cdot \overline{B}$$

Circuits

- **Combinational Circuit:**

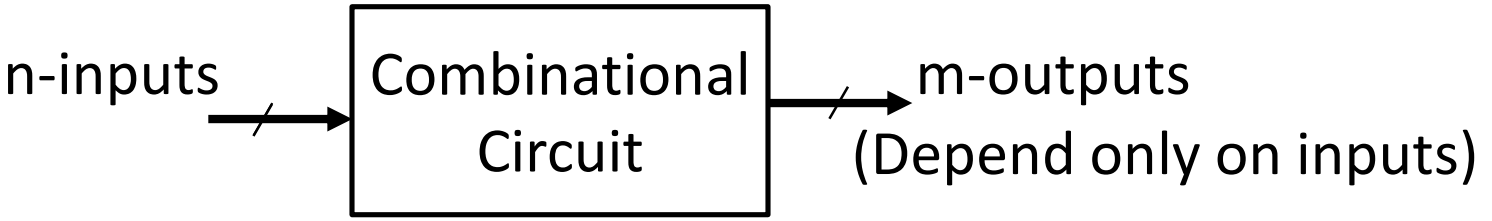
A combinational circuit consists of logic gates whose outputs, at any time, are determined by combining the values of the inputs.

- **Sequential Circuit:**

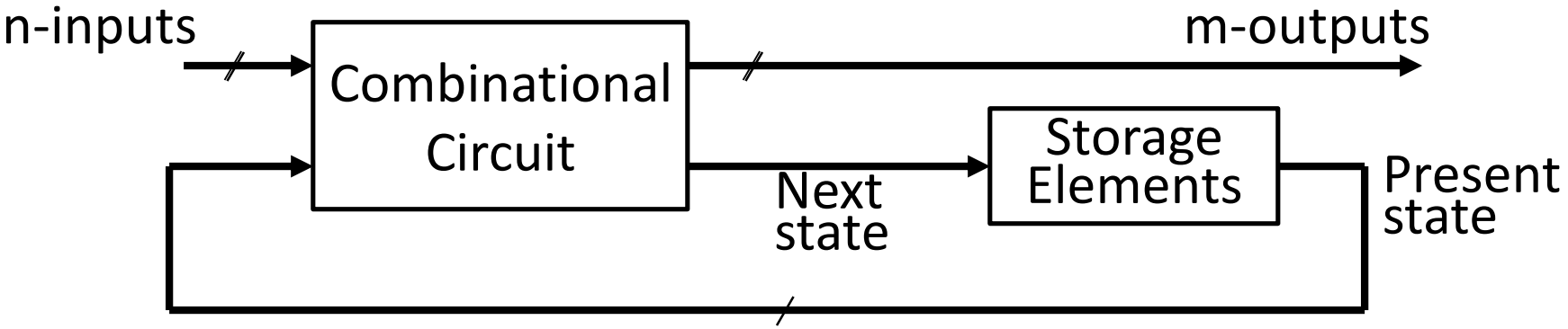
Sequential circuit is the type of circuit where output not only relies on the current input but also depends on the previous output.

- The combinational circuit does not use any memory.
- Sequential circuit has memory.

Combinational vs. Sequential Circuits

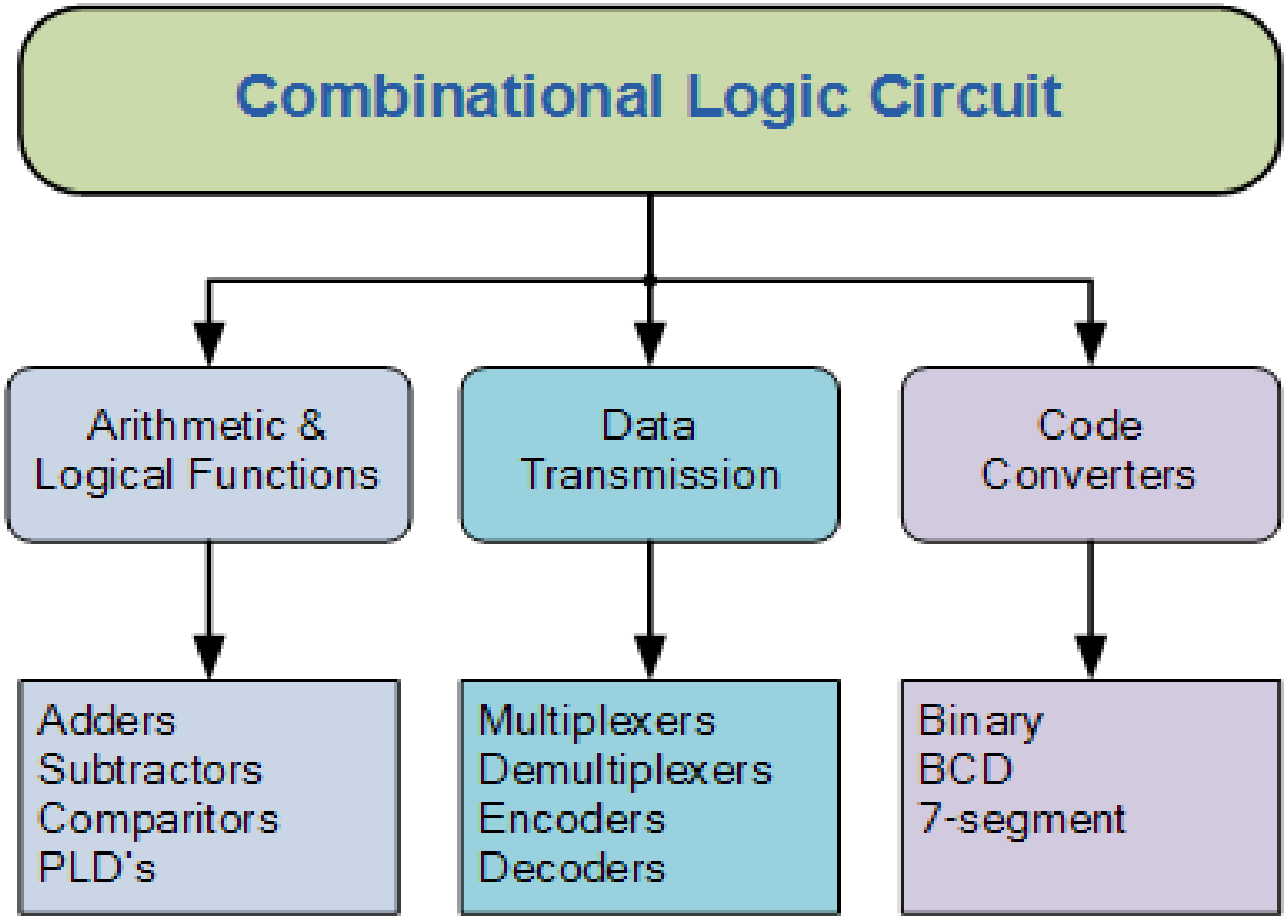


Combinational Circuit



Sequential Circuit

Classification of Combinational Logic



Classification of Sequential Logic

There are two main types of sequential circuits:

- Synchronous
- Asynchronous

- Example of sequential circuits:
  - Flip-Flops
  - Counters
  - Shift registers

Sequential circuits

Sequential circuits

- Sequential Circuit Models
- Latches
- Flip-Flops

Sequential circuits

- The main characteristic of **combinational logic** circuits is that their output values depend on their **present input values**.
- **Sequential logic circuits** differ from combinational logic circuits because they contain memory elements so that their output values depend on both **present** and **past input values.**

Sequential circuits

Sequential circuits can be Synchronous or Asynchronous.

Synchronous sequential circuits change their states and output values at **fixed points of time**, i.e. **clock signals**.

Asynchronous sequential circuits change their states and output values whenever a change in input values occurs.

- Asynchronous sequential circuit: circuit output can change at any time (clockless).

Latch vs flip-flop

- A Latch or Flip-flop is use to store one bit of information.

Latches: Latches are “transparent” (= any change on the inputs is seen at the outputs immediately).

- This causes synchronization problems.
- A latch has two stages *set* and *reset*.
- *Set* stage sets the output to 1.
- *Reset* stage set the output to 0 (Clear to 0).

Flip-flops : A flip-flop is a memory device that has clock signals control the state of the device.

- Clocked sequential circuit (Synchronous).

Types of Flip flops

- SR Flip-Flop
- D Flip-Flop
- JK Flip-Flop
- T Flip-Flop

SR (Set-Reset) Flip-flop

- SR Flip-Flop has three inputs, S (for set), R (for reset), and C (for Clock).
- It has an output Q.

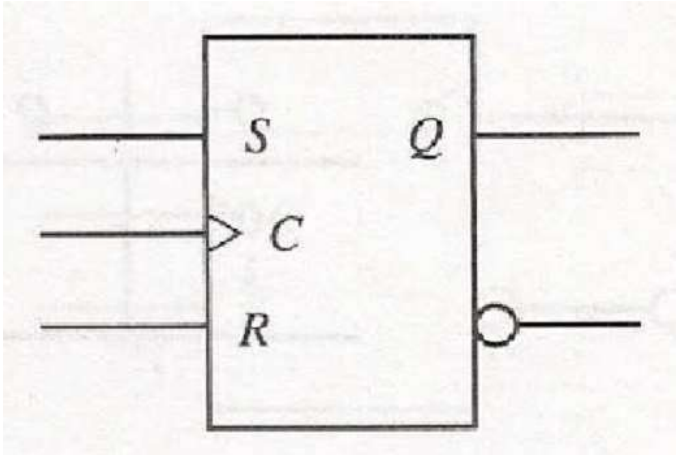


Fig: Graphic symbol

S	R	$Q(t+1)$	
0	0	$Q(t)$	No change
0	1	0	Clear to 0
1	0	1	Set to 1
1	1	?	Indeterminate

Fig: Characteristic Table

- Rarely used due to indeterminate condition.

D (Data) Flip-flop

- D Flip-Flop is a slight modification of SR flip-flop.
- D Flip-Flop is created by inserting an inverter between S and R.

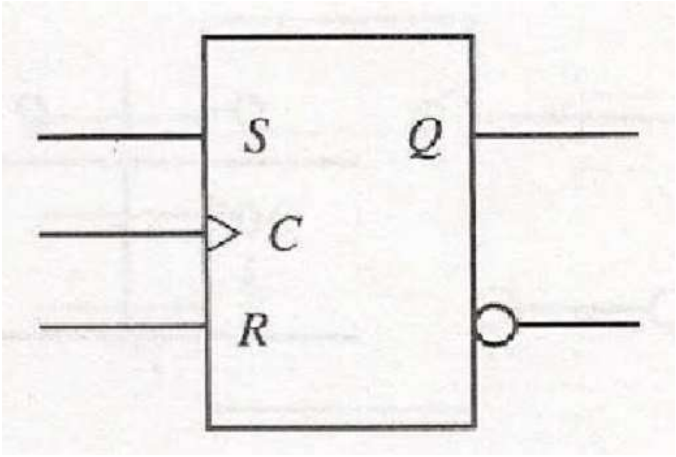


Fig: SR flip-flop

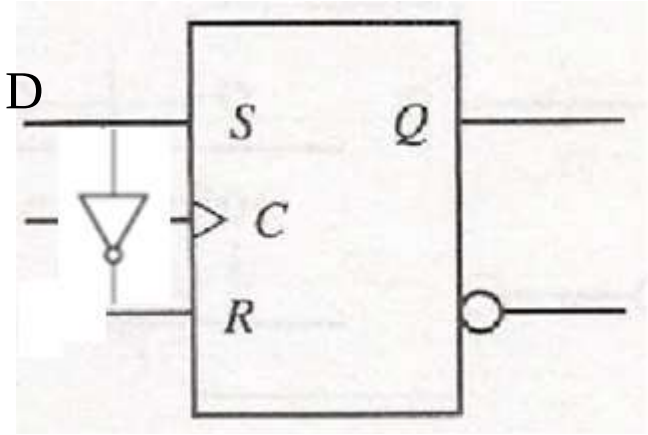


Fig: D flip-flop

D (Data) Flip-flop

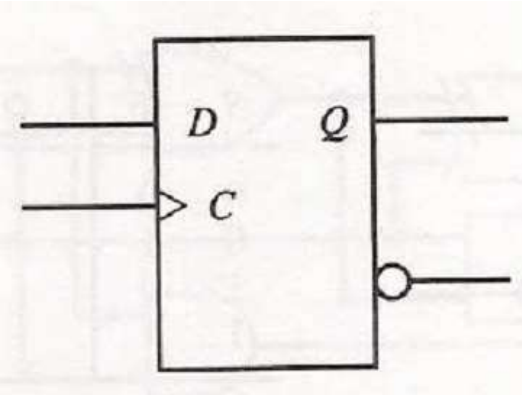


Fig: Graphic symbol

D	Q (t + 1)	
0	0	Clear to 0
1	1	Set to 1

Fig: Characteristic Table

- The output of the D flip-flop is determined by input D.
- The output can be expressed by

$$Q(t+1)=D$$

- No input condition exists for unchanged and indeterminate.

JK (Jack-Kilby) Flip-flop

- JK Flip-Flop is a refinement of SR flip-flop.
- When inputs J and K are both equal to 1, it complement the state.

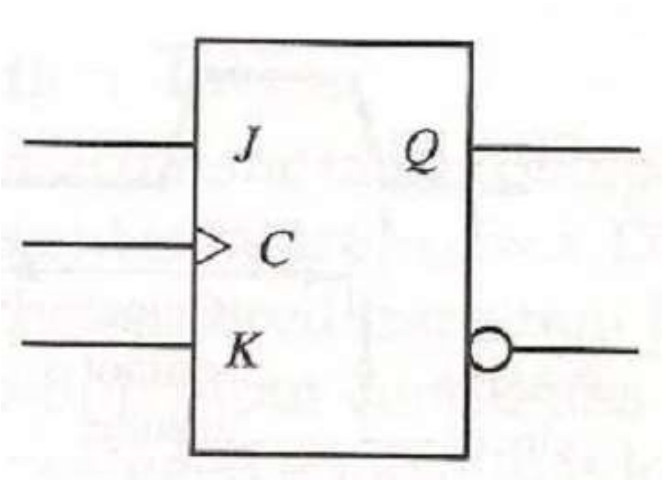


Fig: Graphic symbol

J	K	$Q(t+1)$	
0	0	$Q(t)$	No change
0	1	0	Clear to 0
1	0	1	Set to 1
1	1	$Q'(t)$	Complement

Fig: Characteristic Table

T(Toggle) Flip-flop

- T Flip-Flop is a slight modification of JK flip-flop.
- Input JK are connected to provide a single input by T.
- Only has two input conditions T=0 (J=K=0) and T=1 (J=K=1).
- The output can be expressed by $Q(t+1)=Q(t) \oplus T$.

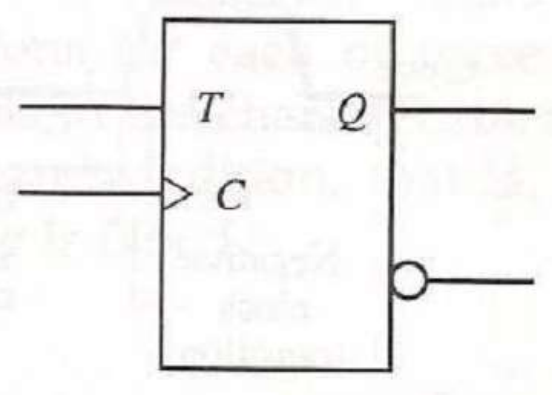


Fig: Graphic symbol

T	$Q(t+1)$	
0	$Q(t)$	No change
1	$Q'(t)$	Complement

Fig: Characteristic Table

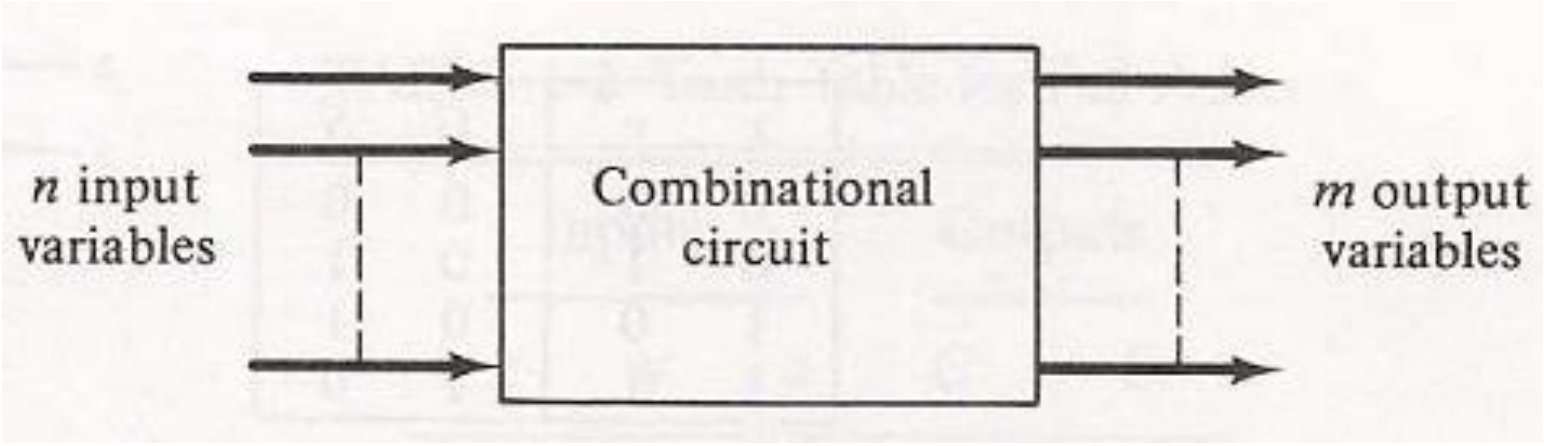
Combinational Circuits

Circuits

- **Combinational Circuit:**

output of this circuit mainly depends on the input terminals at any time.

- *n* binary input variables come and *m* binary output variables go out.
- This circuit doesn't include any memory.



Adder

An adder is a digital logic circuit that is used for the **addition of numbers**.

Adders are basically classified into two types:

- Half-Adder
- Full-Adder.

Half-Adder

Half Adder performs the arithmetic addition of two bits.

- The half adder accepts two binary digits on its inputs and produce two binary digits outputs, sum (S) and carry (C) bit.
- The carry (C) output is **0** unless both the inputs are **1**.
- **S** represents the least significant bit of the sum.

Half-Adder

Logical Expression of Half-Adder:

Sum (S) = $\overline{A} \cdot B + A \cdot \overline{B} = A \oplus B$

Carry (C) = $A \cdot B$ = A AND B

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Table: Truth Table

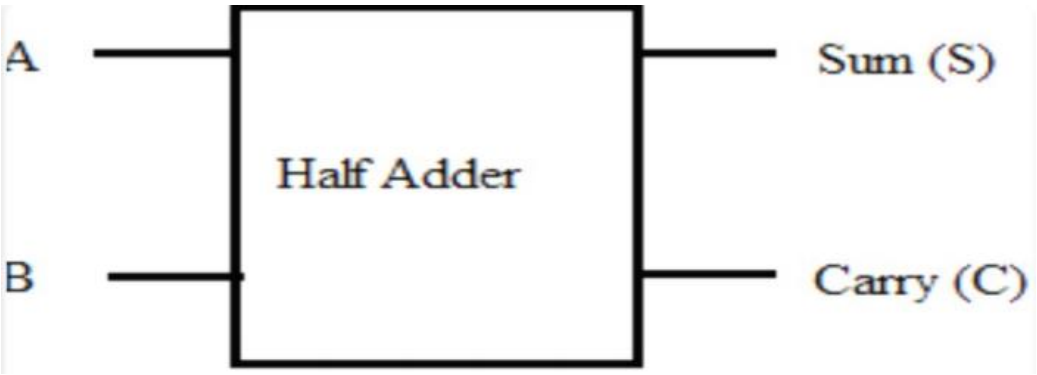


Fig: Block diagram of Half-Adder

Half-Adder

Logical Expression of Half-Adder:

Sum (S) = $\overline{A} \cdot B + A \cdot \overline{B} = A \oplus B$

Carry (C) = $A \cdot B$ = A AND B

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Table: Truth Table

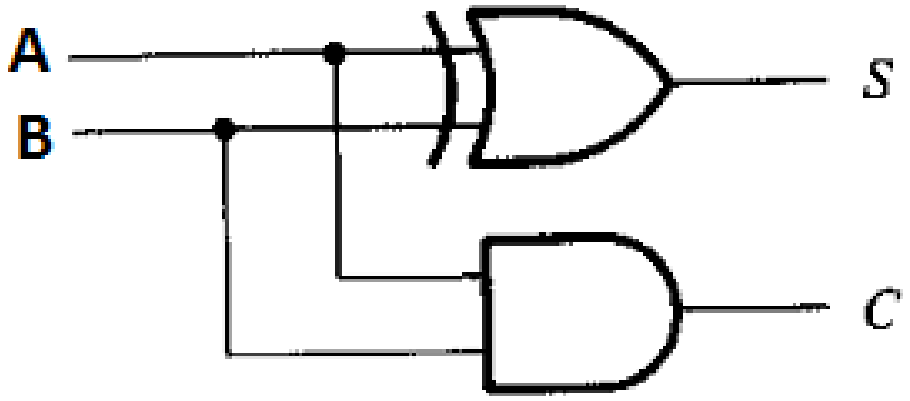


Fig: Logic diagram of Half-Adder

Full-Adder

Full Adder performs the arithmetic sum of three bits.

- The Full-Adder accepts three inputs and produce two outputs.
- The ***S*** output is equal to 1 when only one input is equal to 1 or when all three inputs are equal to 1.
- The ***C*** output has carry of 1 if two or three inputs are equal to 1.

Full-Adder

Logical Expression of Full-Adder:

Sum (S) = $\overline{A} \cdot \overline{B} \cdot C_{in} + \overline{A} \cdot B \cdot \overline{C}_{in} + A \cdot \overline{B} \cdot \overline{C}_{in} + A \cdot B \cdot C_{in} = A \oplus B \oplus C_{in}$

Carry (C_{out}) = $A \cdot B + A \cdot C_{in} + B \cdot C_{in}$

A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table: Truth Table

Fig: Block diagram of Full-Adder

Full-Adder

Logical Expression of Full-Adder:

Sum (S) = $\overline{A}\cdot\overline{B}\cdot C_{in} + \overline{A}\cdot B\cdot \overline{C}_{in} + A\cdot\overline{B}\cdot \overline{C}_{in} + A\cdot B\cdot C_{in} = A\oplus B\oplus C_{in}$

Carry(C_{out})$=A\cdot B+A\cdot C_{in}+B\cdot C_{in}=A\cdot B+(\overline{A}\cdot B+A\cdot\overline{B})\cdot C_{in}$
$=A\cdot B+(A\oplus B)\cdot C_{in}$

A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table: Truth Table

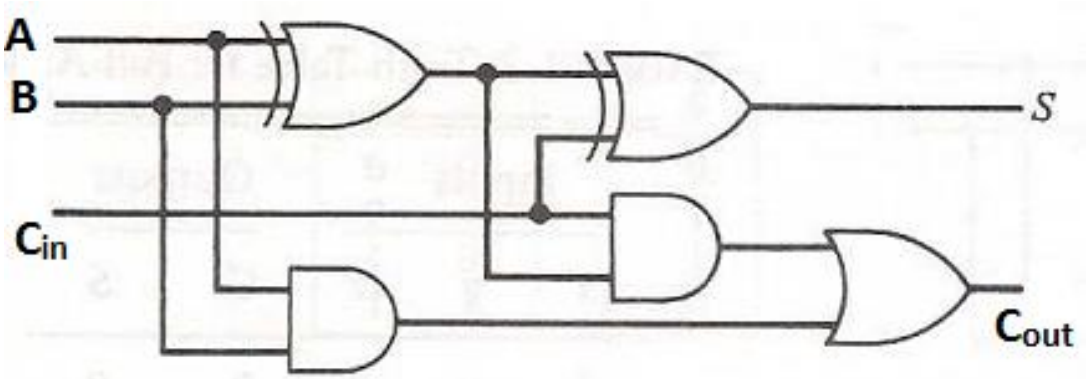


Fig: Logic diagram of Full-Adder

Subtractor

Subtractor: each subtrahend bit of the number is subtracted from its corresponding significant minuend bit to form a **difference bit**.

- If the minuend bit is smaller than the subtrahend bit, a 1 is borrowed from the next significant position.

- Half-Subtractor
- Full-Subtractor.

Half-Subtractor

A half-subtractor is a circuit that subtracts two bits and produces their difference.

- The half-subtractor needs two outputs.
  - One output generates the difference (***D***).
  - The second output for borrow (***B***).
- To perform ***x-y***, we have to check the relative magnitudes of ***x*** and ***y***. If ***x>= y***, we have three possibilities:

  0-0=0, 1-0=1 and 1-1=0.
- If ***x<y***, we have ***0-1*** and it is necessary to borrow a ***1*** from the next higher stage.

Half- Subtractor

Logical Expression of Half-Subtractor:

Difference (D) = $\overline{x} \cdot y + x \cdot \overline{y} = A \oplus B$

Borrow (B) = $\overline{x} \cdot y$ = $\overline{x}$ AND y

x	*y*	*B*	*D*
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

Table: Truth Table

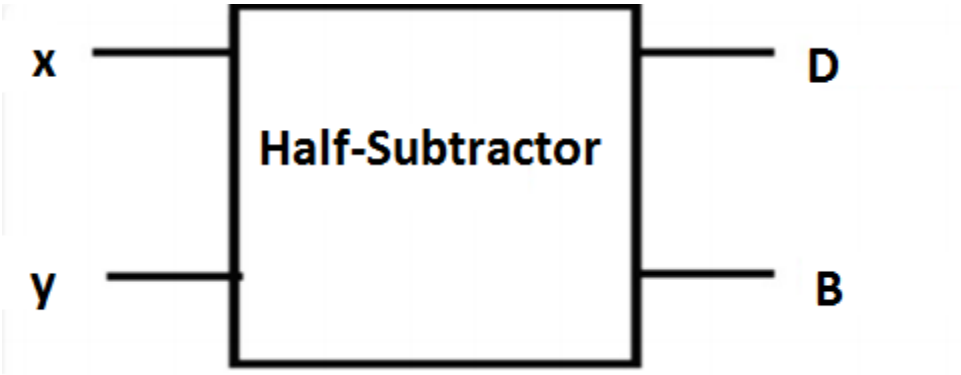


Fig: Block diagram of Half-Subtractor

Half- Subtractor

Logical Expression of Half-Subtractor:

Difference (D) = $\overline{x} \cdot y + x \cdot \overline{y} = A \oplus B$

Borrow (B) = $\overline{x} \cdot y$ = $\overline{x}$ AND y

x	*y*	*B*	*D*
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

Table: Truth Table

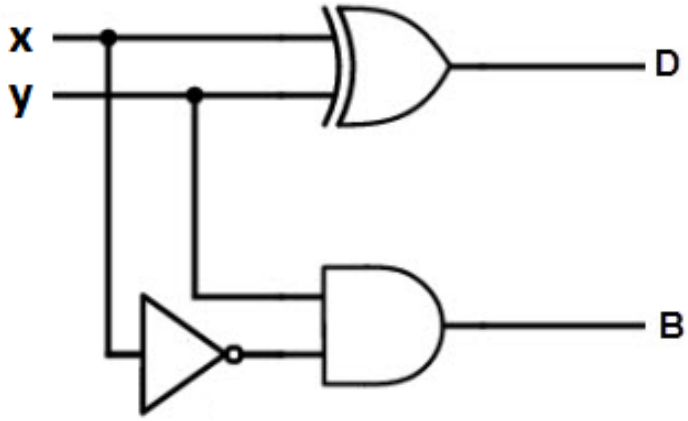


Fig: Logic diagram of Half-Subtractor

Full-Subtractor

A Full-subtractor performs a subtraction between two bits and taking into account the borrow bit.

- This circuit has three inputs and two outputs
- The three inputs x, y and B_{in}, where B_{in} is previous borrow.
- The two outputs, D and B_{out} represent the difference and output borrow, respectively.

Full-Subtractor

Logical Expression of Full-Subtractor:

Difference (D) = $\overline{x} \cdot \overline{y} \cdot B_{in} + \overline{x} \cdot y \cdot \overline{B_{in}} + x \cdot \overline{y} \cdot \overline{B_{in}} + x \cdot y \cdot B_{in} = x \oplus y \oplus B_{in}$

Borrow (B_{out}) = $\overline{x} \cdot y + \overline{x} \cdot B_{in} + y \cdot B_{in}$

x	y	B_{in}	B_{out}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Table: Truth Table

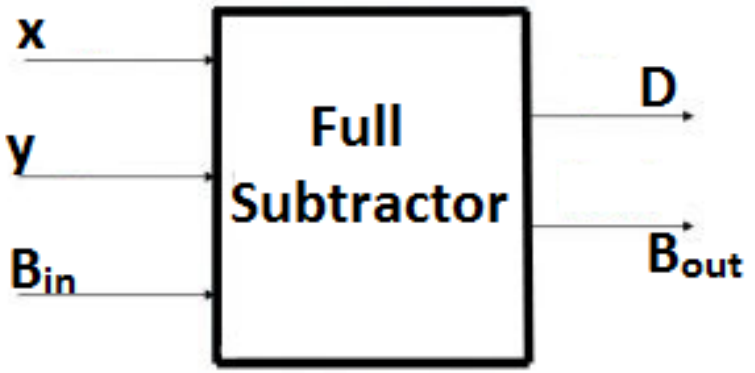


Fig: Block diagram of Full-Subtractor

Full-Subtractor

Logical Expression of Full-Subtractor:

Difference (D) = $\overline{x} \cdot \overline{y} \cdot B_{in} + \overline{x} \cdot y \cdot \overline{B_{in}} + x \cdot \overline{y} \cdot \overline{B_{in}} + x \cdot y \cdot B_{in} = x \oplus y \oplus B_{in}$

Borrow (B_{out}) = $\overline{x} \cdot y + (\overline{x \oplus y}) \cdot B_{in}$

x	y	B_{in}	B_{out}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Table: Truth Table

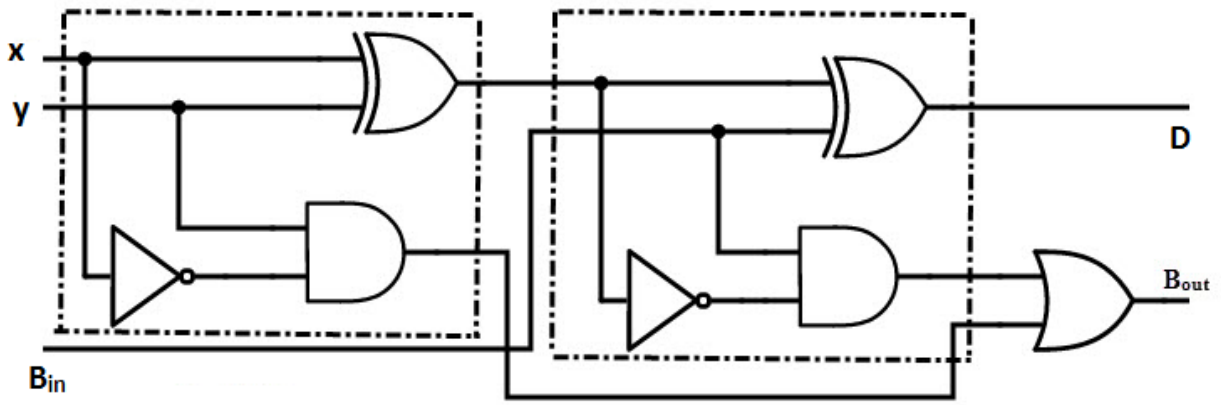


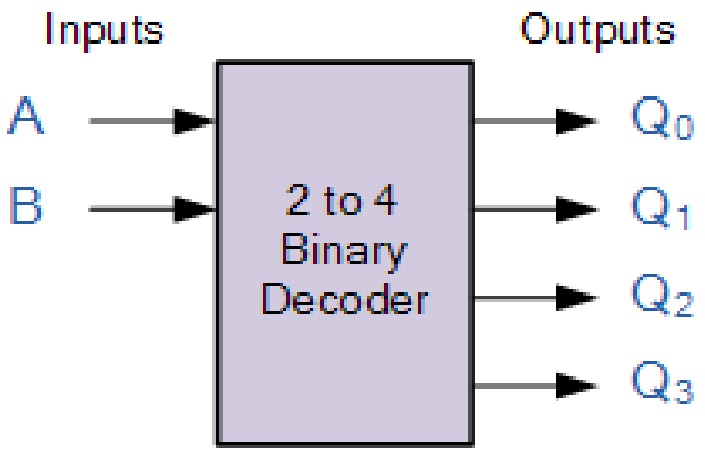
Fig: Logic diagram of Full-Subtractor

Combinational Circuits (Decoder and Encoder)

Decoder

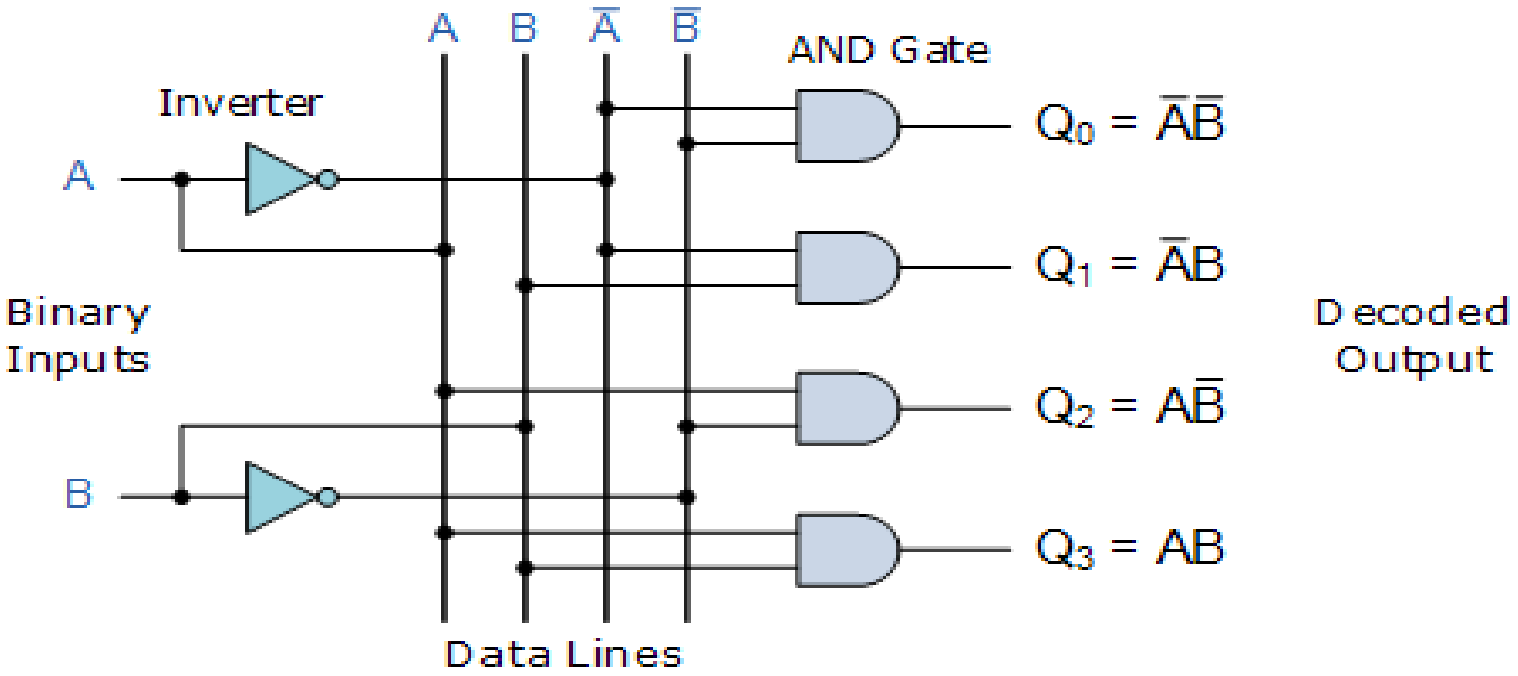
- The name **Decoder** means to translate or decode coded information from one format into another.
- A decoder is a **combinational circuit** that converts binary information from **n** input lines to a maximum of $\mathbf{2^n}$ unique output lines.
- n inputs
- 2^n outputs
- If a binary decoder receives n inputs it activates one and only one of its 2^n outputs based on that input with all other outputs deactivated.
- A decoder has n inputs and m outputs is referred to as an **n × m** decoder

Decoder

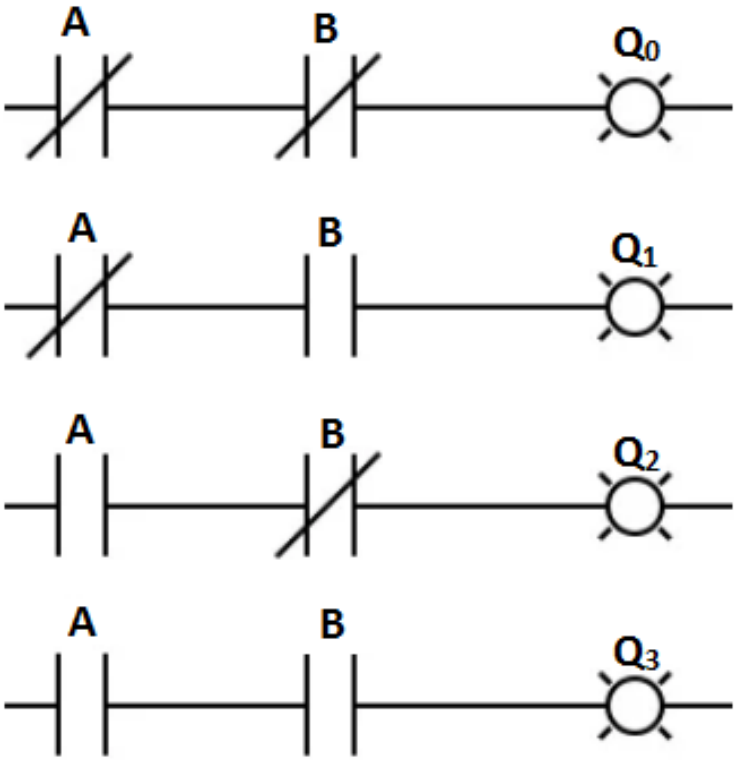


Truth Table

A	B	Q_0	Q_1	Q_2	Q_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



Decoder

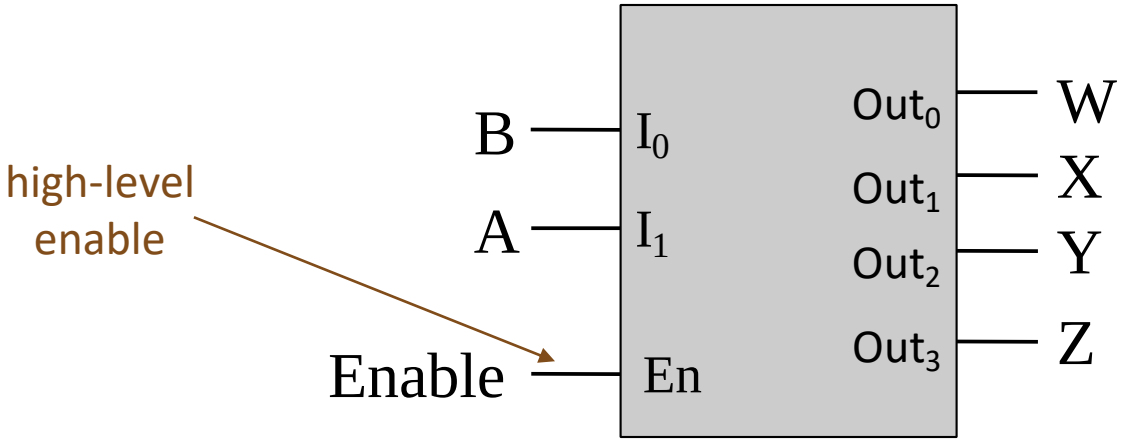


Truth Table

A	B	Q_0	Q_1	Q_2	Q_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

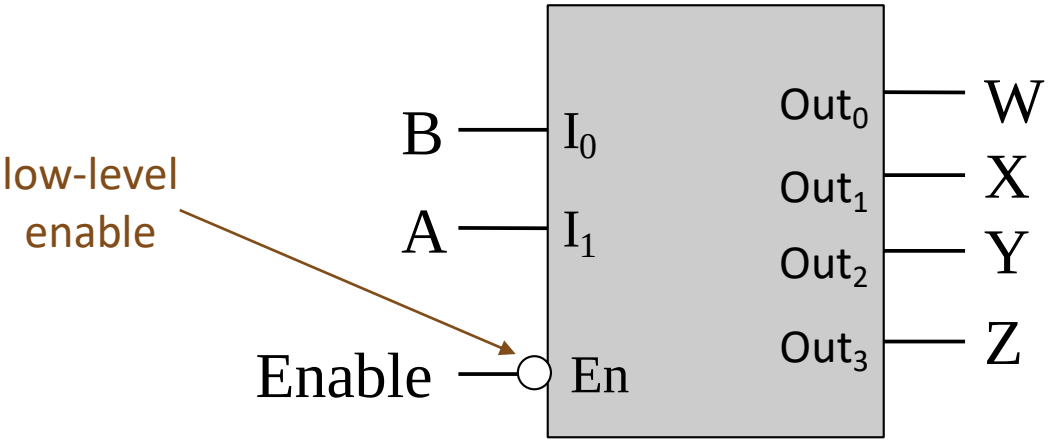
Decoder with Enable input

- Decoders include one or more enable inputs to control the operation of the circuit.
- The decoder is enabled when E is equal to 1 and disabled when E is equal to 0.



	En	A	B	W	X	Y	Z
enabled	1	0	0	1	0	0	0
	1	0	1	0	1	0	0
	1	1	0	0	0	1	0
	1	1	1	0	0	0	1
disabled	0	x	x	0	0	0	0

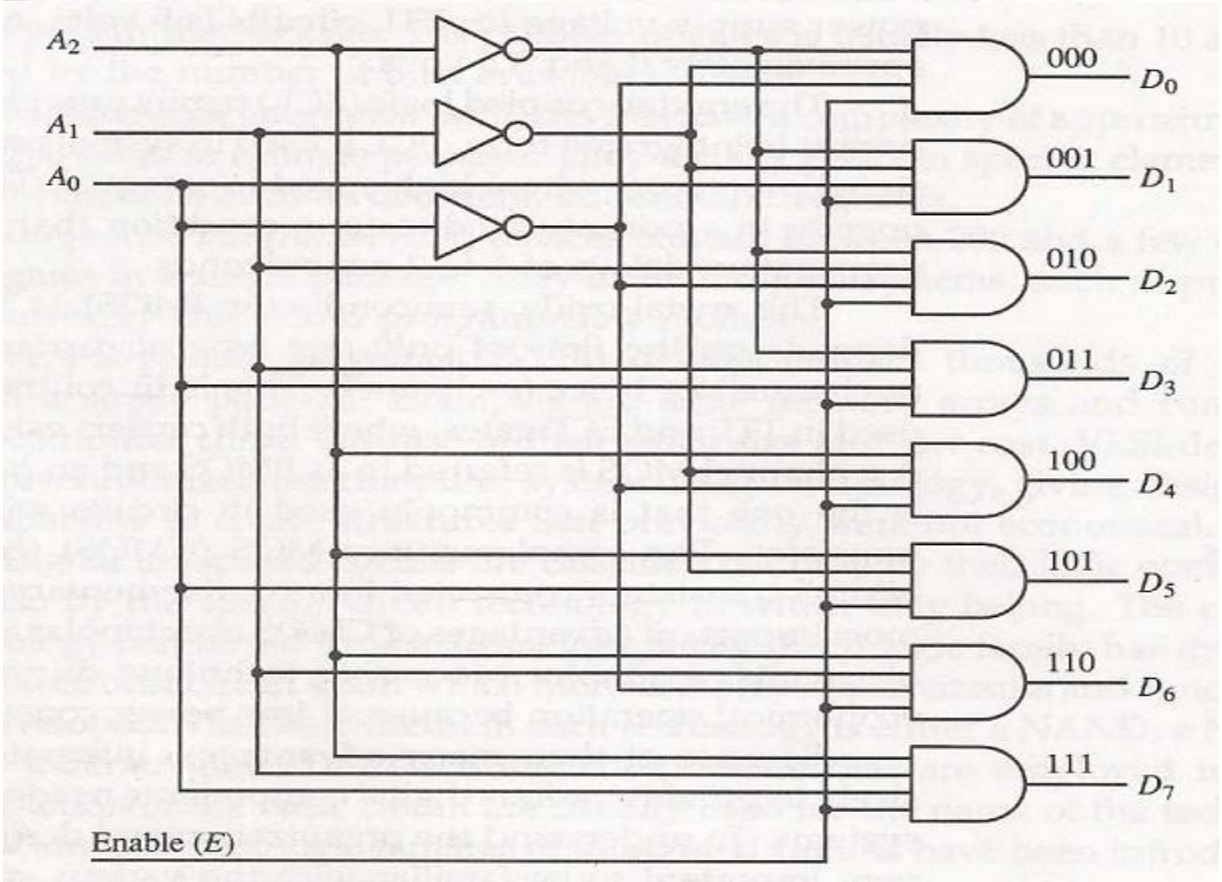
Decoder with Enable input



	En	A	B	W	X	Y	Z
enabled	0	0	0	1	0	0	0
	0	0	1	0	1	0	0
	0	1	0	0	0	1	0
	0	1	1	0	0	0	1
disabled	1	x	x	0	0	0	0

3-to-8 line Decoder

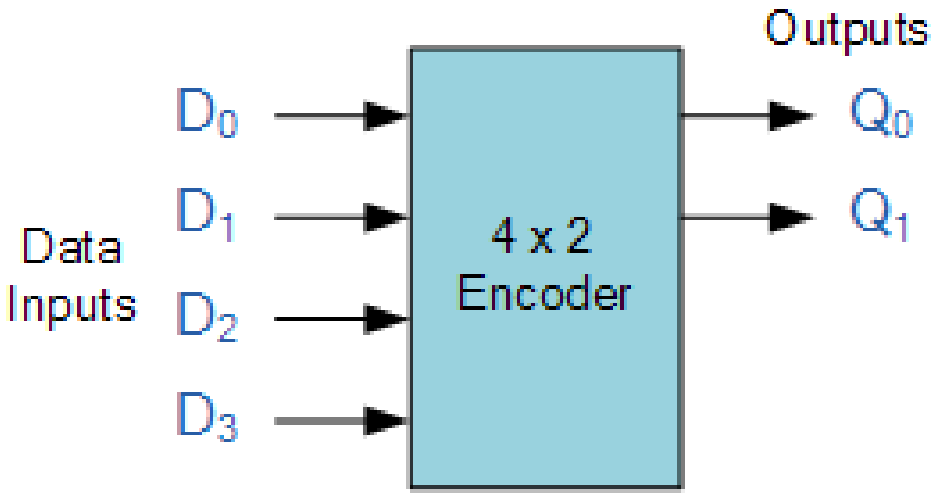
Enable	Inputs			Outputs							
E	A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	×	×	×	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0



Encoder

- Encoder is a combinational circuit that perform the inverse operation of the decoder.
- An encoder has **2^n input** lines and **n output** lines.
- Encoders assume that only one input line is active at a time.

4 × 2 Encoder



Inputs				Outputs	
D_3	D_2	D_1	D_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
0	0	0	0	x	x

8 × 3 Encoder

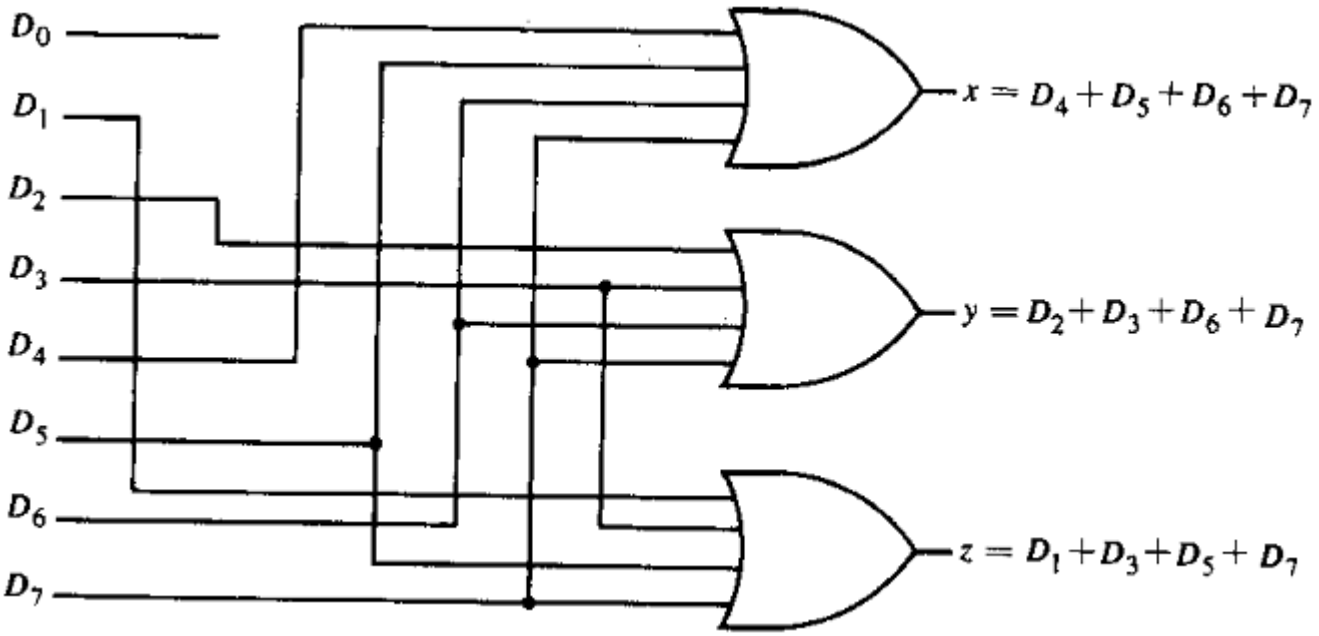
Truth Table of Octal-to-Binary Encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$$z = D_1 + D_3 + D_5 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

$$x = D_4 + D_5 + D_6 + D_7$$



Encoder Design Issues

There are two ambiguities associated with the design of a simple encoder:

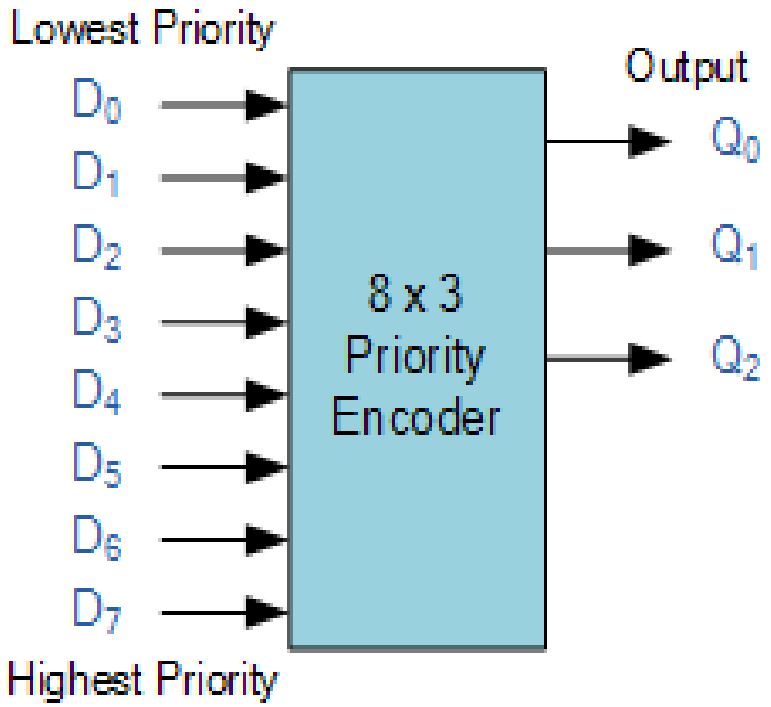
- Only one input can be active at any given time. If two inputs are active simultaneously, the output produces an **undefined** combination (for example, if D3 and D6 are 1 simultaneously, the output of the encoder will be 111.
- An output with all 0's can be generated when all the inputs are 0's,or when D_0 is equal to 1.

Priority Encoder

- Multiple asserted inputs are allowed; one has priority over all others.
- Separate indication of no asserted inputs.

- A priority encoder is a special type of encoder that includes the **priority function**.
- If two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

Priority Encoder



Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

X = don't care

Valid-output indicator

A valid-output indicator, designed by **V**, is set to 1 only when one or more of the inputs are equal to 1. If all inputs are 0, V is equal to 0 and the other outputs of the circuit are not used.

Truth Table of a Priority Encoder

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Combinational Circuits (Multiplexer and De-Multiplexer)

Multiplexer

- A **Multiplexer** is a combinational circuit that selects binary information from one of **many input** lines and directs it to a **single output** line.
- 2^n input lines
- 1 output line
- n selection lines: combinations determine which input is selected.
- Multiplexers are also known as **parallel to serial convertor**, **data selector** and **many to one** circuit.
- often abbreviated as MUX.

4-to-1-line Multiplexer

- Input lines = 4
- Selection lines = 2

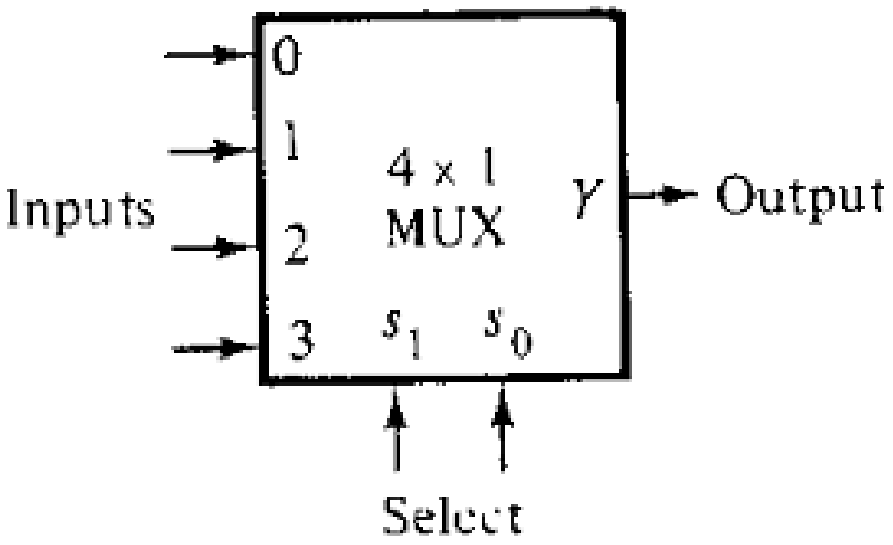


Fig: Block diagram

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Table: Truth table

4-to-1-line Multiplexer

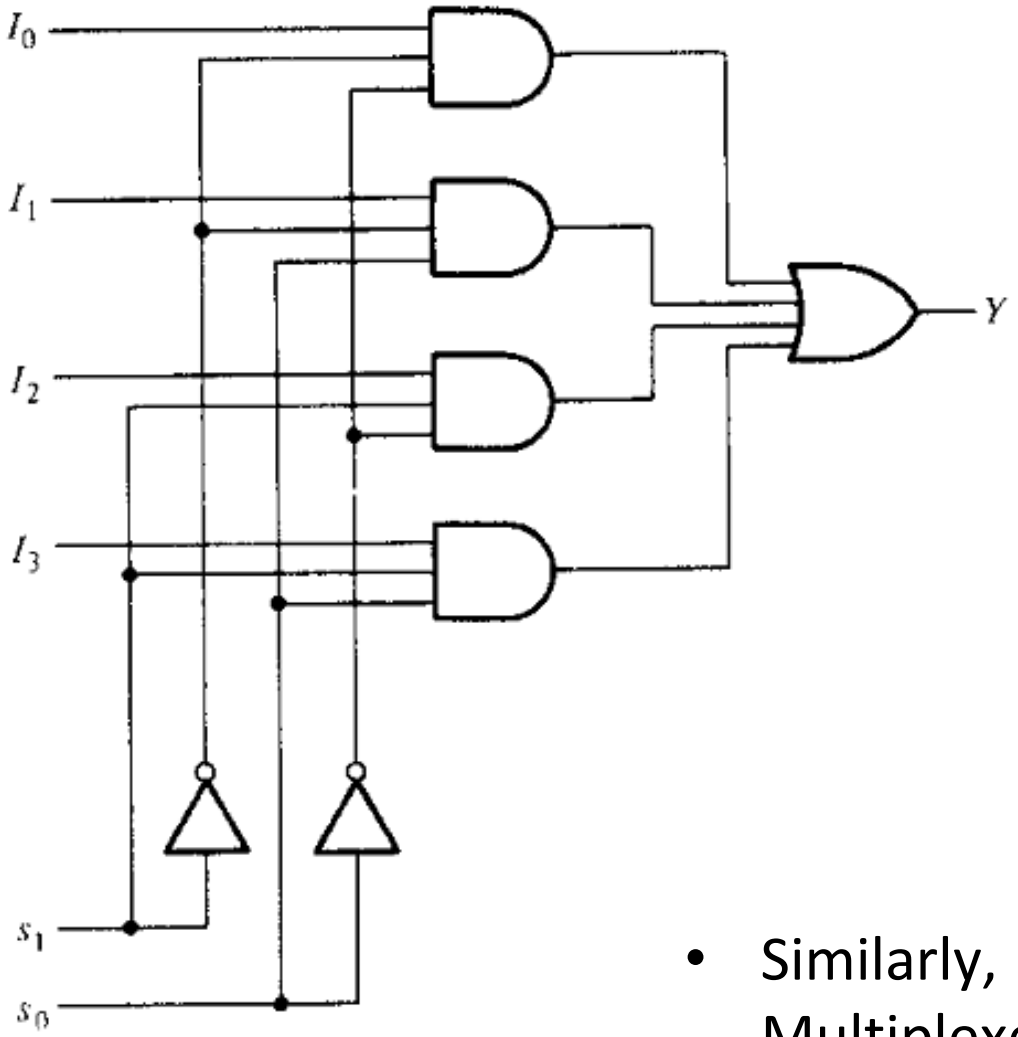


Fig: Logical diagram

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Table: Truth table

- Similarly, you can implement 8x1 Multiplexer and 16x1 multiplexer by following the same procedure.

Multiplexer with enable input

- As in decoders, multiplexer circuits may have an enable input to control the operation.
- When the enable input is in the inactive state, the outputs are disabled.
- When it is in the active state, the circuit functions as a normal multiplexer.

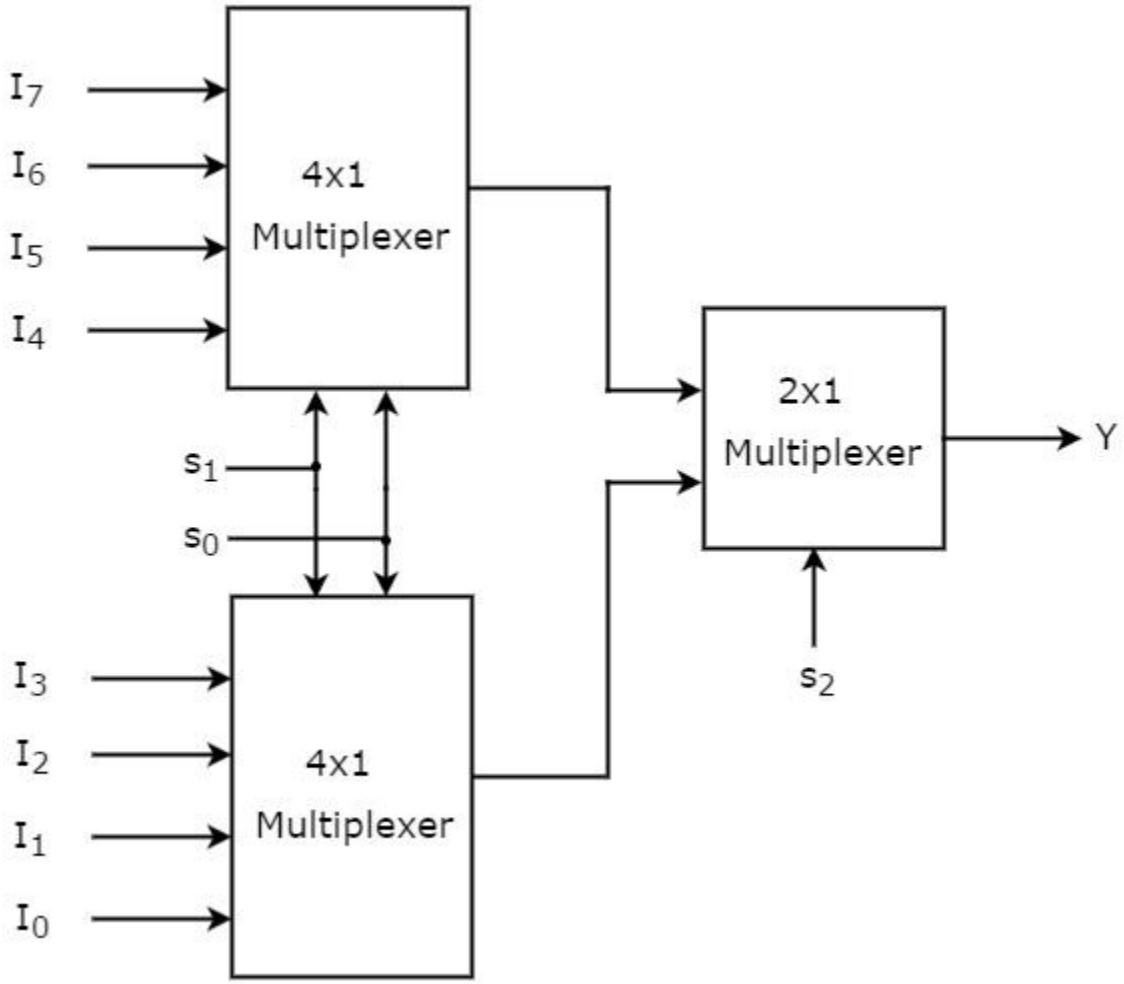
Implement of higher-order Multiplexers using lower-order Multiplexers.

- Implement 8x1 Multiplexer using 4x1 Multiplexers and 2x1 Multiplexer.

- We know that 4x1 Multiplexer has 4 data inputs, 2 selection lines and one output. Whereas, 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output.
- So, we require two **4x1 Multiplexers.**
- Each 4x1 Multiplexer produces one output, we require a **2x1 Multiplexer** considering the outputs of 4x1 Multiplexers as inputs and to produce the final output.

Implement of higher-order Multiplexers using lower-order Multiplexers.

- Implement 8x1 Multiplexer using 4x1 Multiplexers and 2x1 Multiplexer.



Selection Inputs			Output
S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

Applications of Multiplexer

- Signal routing
- Data communications (Time Division Multiplexing)
- Data bus control

Advantage of Multiplexer

- Only one serial data line is required instead of multiple parallel data lines.

Demultiplexer

- A demultiplexer is a circuit that receives information on a **single line** and transmits this information on one of the **2^n** possible output lines.
- Reverse of the Multiplexer
- 1 input line
- 2^n output lines
- n selection lines: combinations determine which output line is selected.
- Demultiplexers are also known as **serial to parallel convertor**, **Data Distributor** and **one to many** circuit.
- often abbreviated as Demux.

1-to-4-line Demultiplexer

- Output lines = 4
- Selection lines = 2

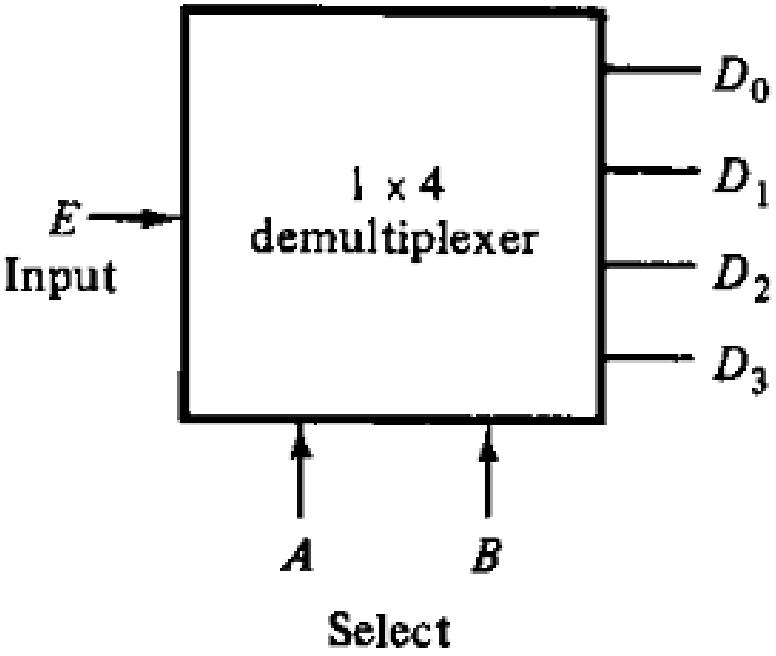


Fig: Block diagram

A	B	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Table: Truth table

1-to-4-line Demultiplexer

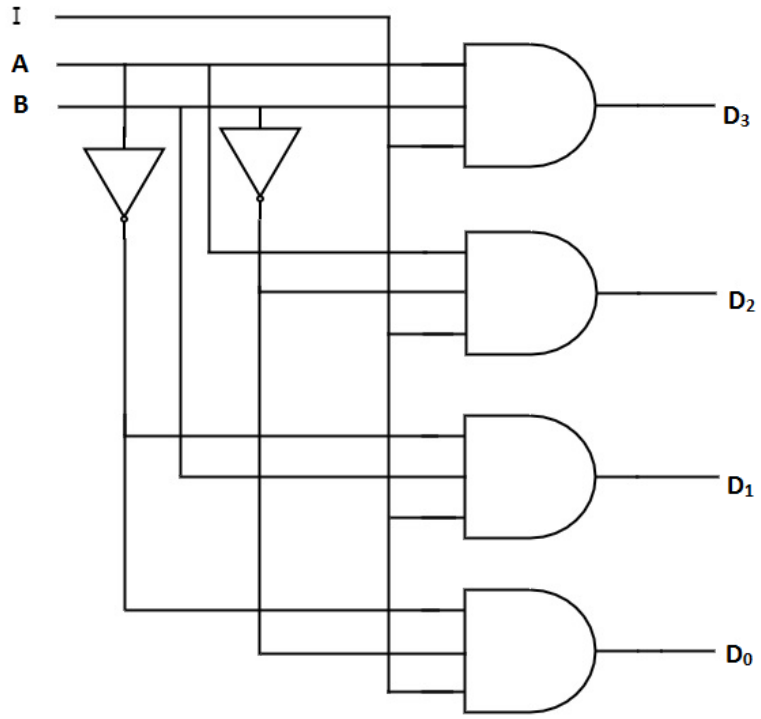


Fig: Logical diagram

A	B	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Table: Truth table

- Similarly, you can implement 1x8 Demultiplexer and 1x16 Demultiplexer by following the same procedure.

Demultiplexer with enable output

- Demultiplexer circuits may have an enable output to control the operation.
- When the enable output is in the inactive state, the outputs are disabled.
- When it is in the active state, the circuit functions as a normal Demultiplexer.

Difference between of Multiplexer and Demultiplexer

MULTIPLEXER	DEMULTIPLEXER
Multiplexer processes the digital information from various sources into a single source.	Demultiplexer receives digital information from a single source and converts it into several sources
It is known as **Data Selector**	It is known as **Data Distributor**
It follows combinational logic type	It also follows combinational logic type
It has n data input	It has single data input
It has a single data output	It has n data outputs
It works on **many to one** operational principle	It works on **one to many** operational principle
In time division Multiplexing, multiplexer is used at the **transmitter** end	In time division Multiplexing, demultiplexer is used at the **receiver** end

Sequential Logic Circuit (Shift Register)

Register

- Flip-flop: A flip-flop can store one-bit of information.
- Register: Group of flip-flops are used to store the binary data. To store multiple bits of information, we need multiple flip-flops.
- **n**-bit register has a group of **n** flip-flops and is capable of storing of information of **n** bits.

  Storage.

Register

- Load: Transfer of new information into a register.
- Clear: Clear the register to all 0's .
- Clock pulse: Applied to all flip-flops.
- Load all four inputs in parallel.

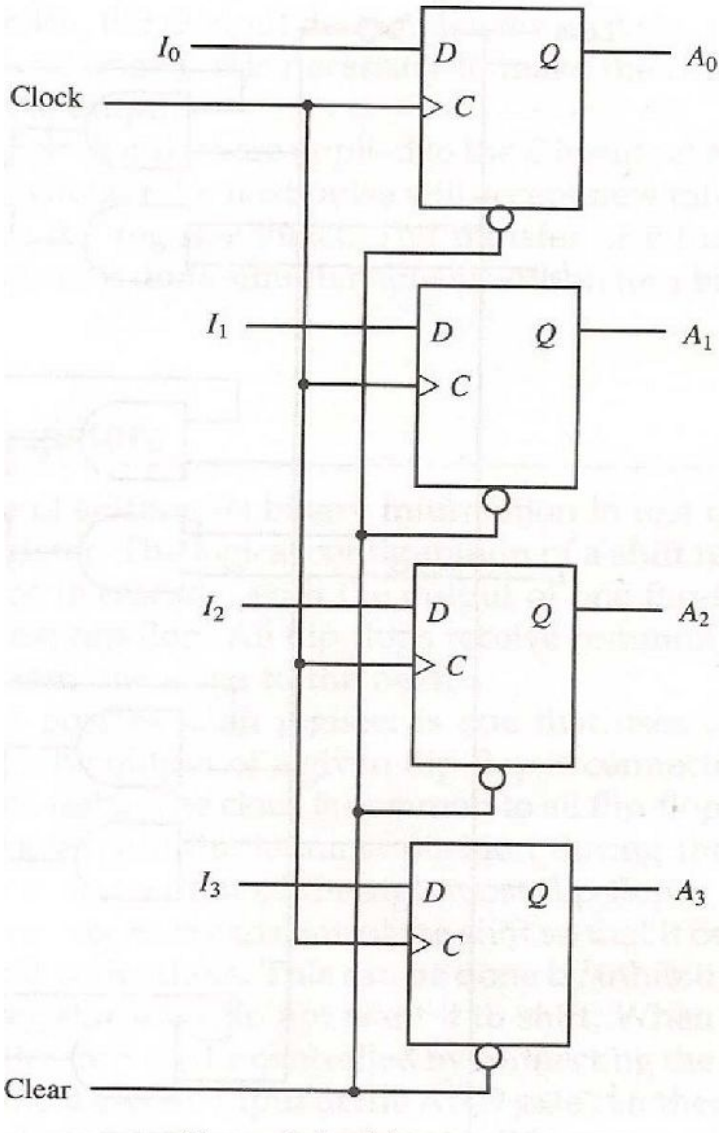


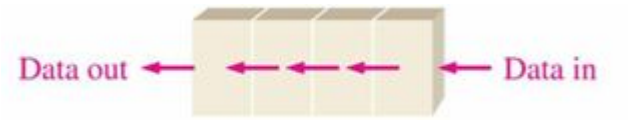
Fig: 4-bit Register

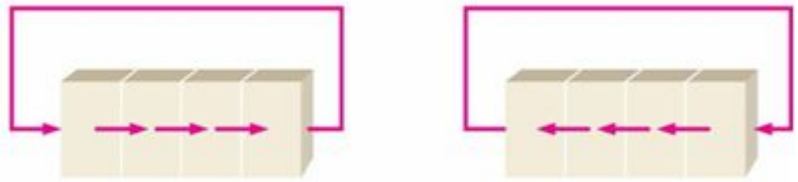
Shift Register

- Shift Register: The Shift Register is another type of **sequential logic circuit** that can be used for the **storage** or the **transfer** of binary data.
- Shift Register: A register capable of shifting its binary information either to the right or to the left.
  - Storage or the transfer
- Shift Register loads the data present on its inputs and then moves or “shifts” it to its output once every clock cycle.
- The logical configuration of a shift register consists of a chain of flip-flops connected in cascade, with the output of one flip-flop connected to the input of the next flip-flop.

Shift Register

- Direction or Date shifting
  - Left shift
  - Right shift
  - Rotate (right or left)
  - Bidirectional





Shift Operation

- A single shift is multiplication by 2
- Consider the operation 6 x 2 = 12

```
  0110
x 0010
  0000
 0110
 01100
```

Shift Operation

In binary number we can move or shift the binary point:

- Right by 1 position to multiply by 2
- Left by 1 position to divide by 2

Move or shift the binary point:

Left by 1 position to multiply by 2

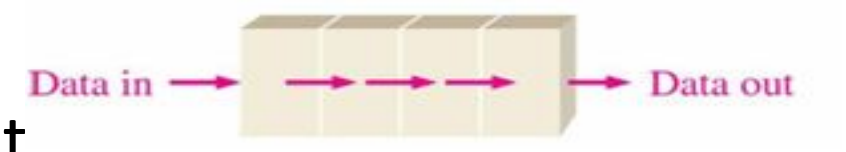
1110 x 2 → 11100.

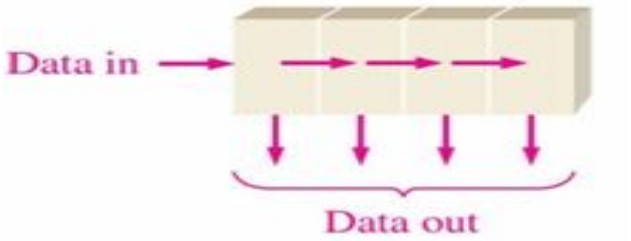
Right by 1 position to divide by 2

11100 ÷ 2 → 1110.

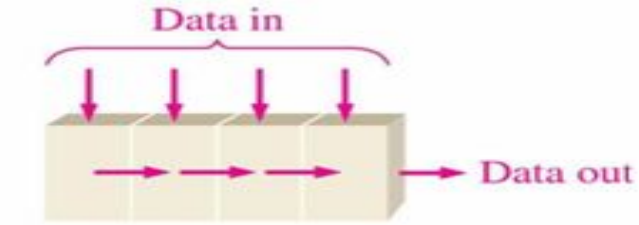
Types of Shift Register

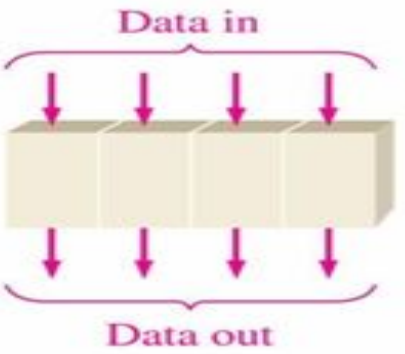
- Shift registers operate in one of four different modes with the basic movement of data.
  - Serial-in, Serial-out
  - Serial-in, Parallel-out





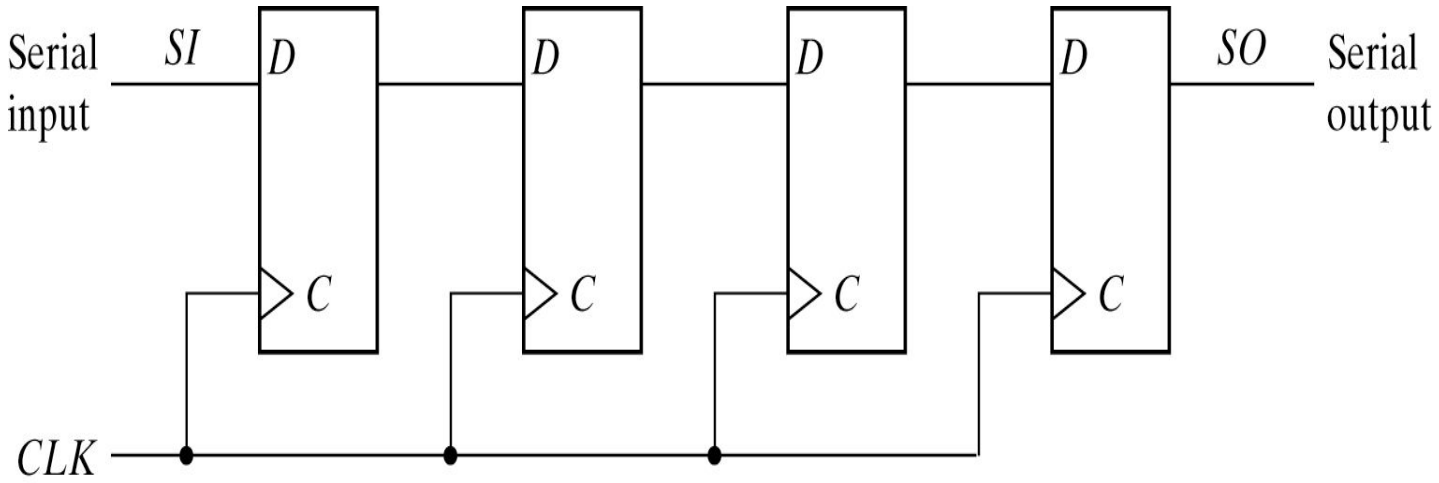
  - Parallel-in, Serial-out
  - Parallel-in, Parallel-out





Serial-In Serial-Out

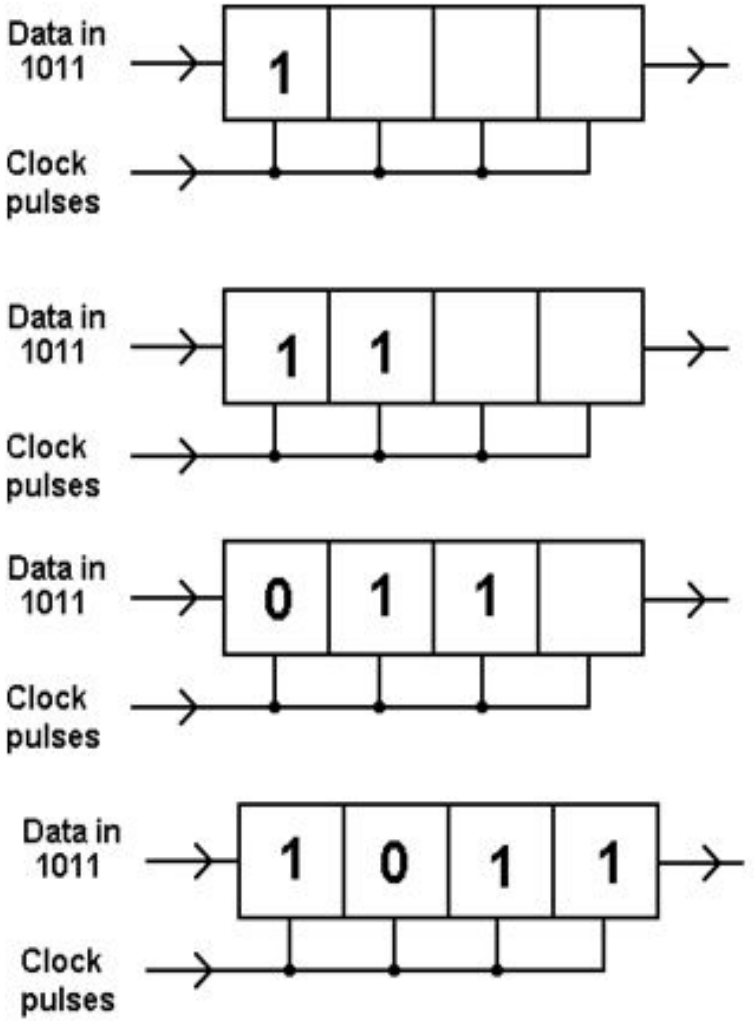
- Data bits come in one at a time and leave one at a time.
- One Flip-Flop is used to handle each bit.
- The output of a Flip-Flop is connected to the D input of the Flip-Flop at its right.



4-Bit Shift Register

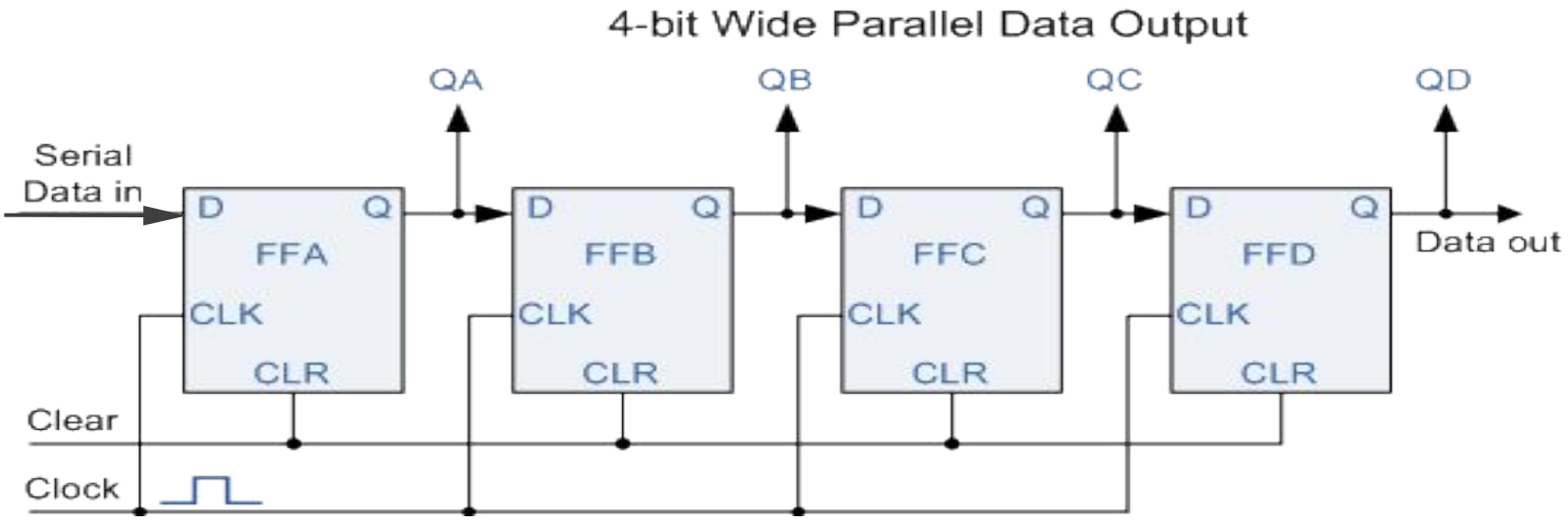
Example

- The 4-bit data word “1011” is to be shifted into a 4-bit shift register.
- One shift per clock pulse.
- Data is shown entering at left and shifting right.



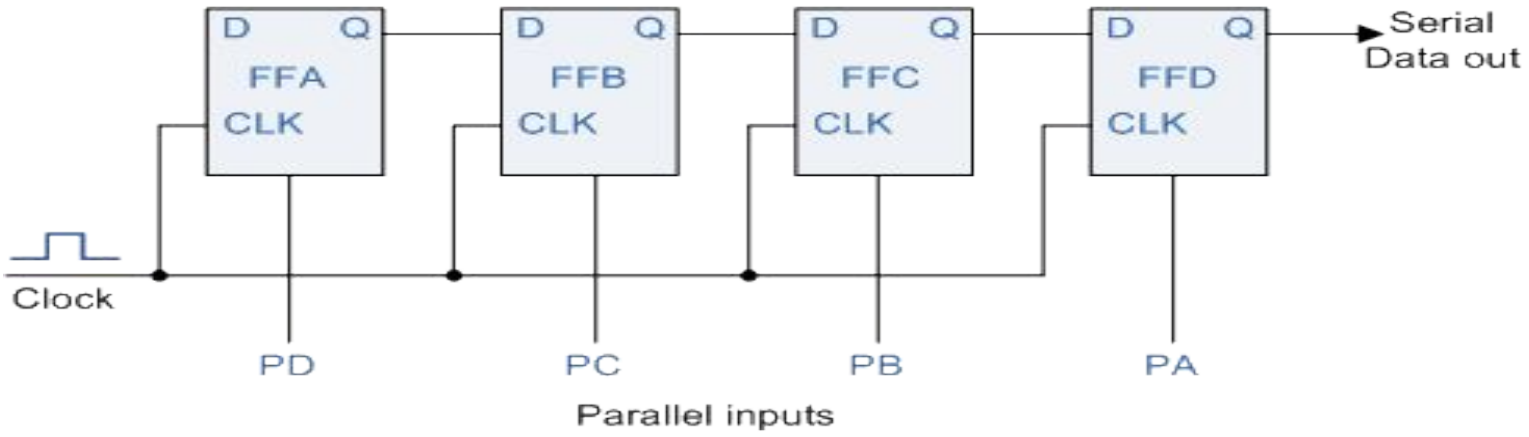
Serial-In Parallel -Out

- Serial-in to Parallel-out (SIPO) - the register is loaded with serial data, one bit at a time, with the stored data being available at the output in parallel form.
- Shift the data contents of each stage one place.
- Data value can now be read directly from the outputs.



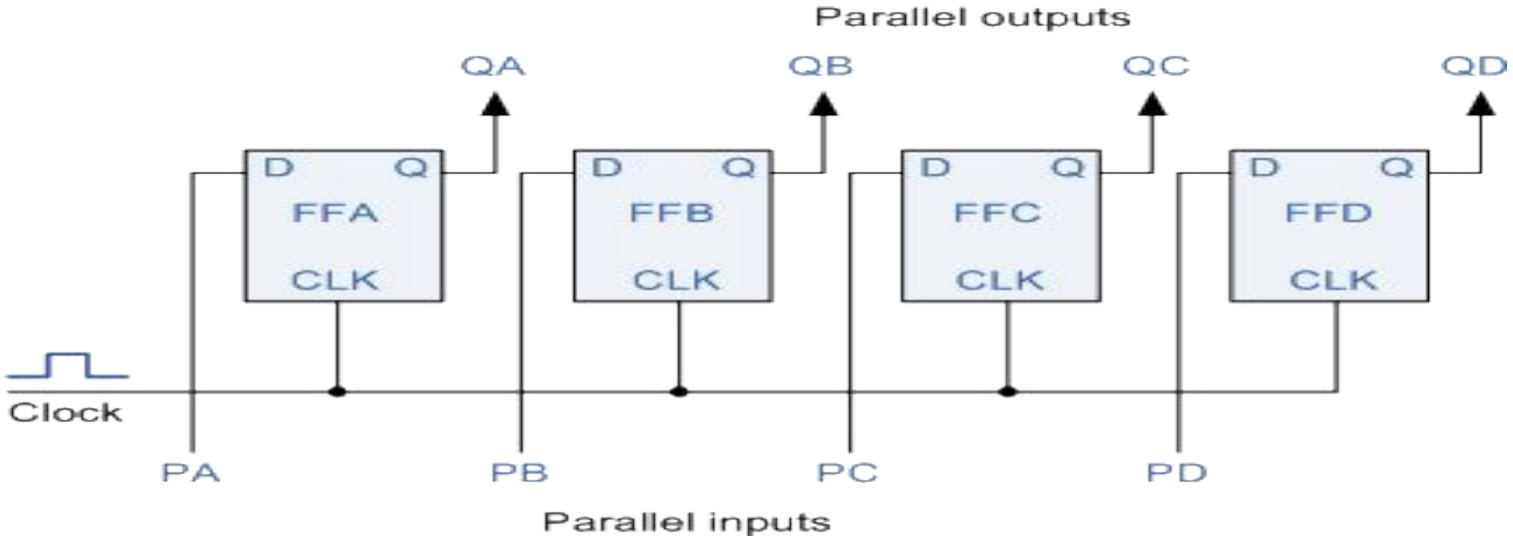
Parallel-In Serial-Out

- Parallel-in to Serial-out (PISO)- the parallel data is loaded into the register simultaneously and is shifted out of the register serially one bit at a clock pulse.
- Act in the opposite way to the serial-in parallel-out
- Data is loaded in a parallel format in which all the data bits enter their inputs simultaneously.
- The data is then read out sequentially in the normal shift-right mode.



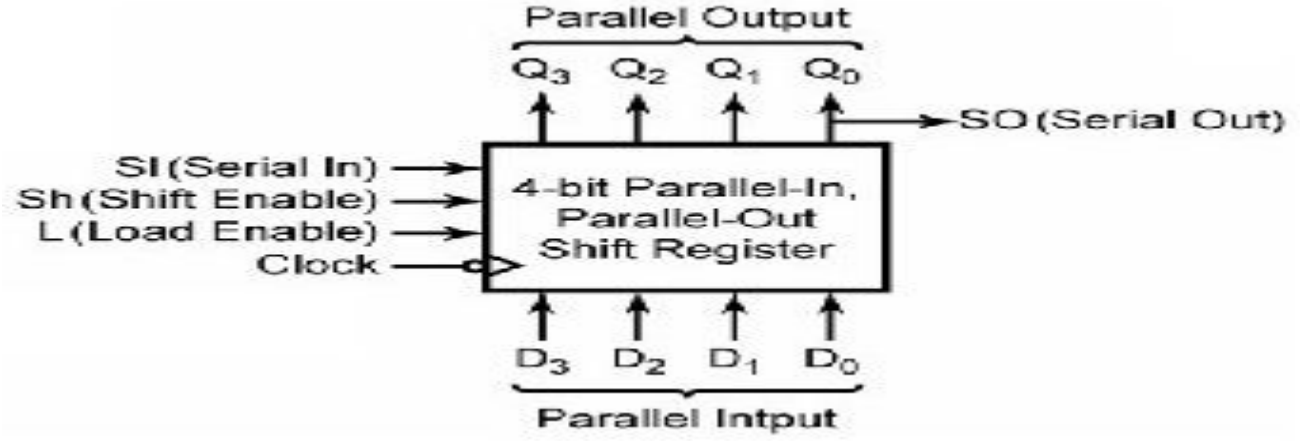
Parallel -In Parallel -Out

- Parallel-in to Parallel-out (PIPO)- the parallel data is loaded simultaneously into the register, and transferred together to their respective outputs by the same clock pulse.
- One clock pulse loads and unloads the register.
- No interconnections between the individual flip-flops since no serial shifting of the data is required.



Universal Shift Register

- Universal shift register can do any combination of parallel and serial input/output operations.
- Can be used in either serial-to-serial, serial-to-parallel, parallel-to-serial, or as a parallel-to-parallel data shifting.
- Requires additional inputs to specify desired function.
- Universal shift registers are frequently used in arithmetic operations to shift data to the left or right for multiplication or division.



Sequential Logic Circuit (Counters)

Counters

- Counter is a sequential circuit.
- Counter is a digital circuit used for counting purpose, they can count specific event happening in the circuit.
- Counters calculate or note down the number that how many times an event occurred.
- Counting means incrementing or decrementing the values of an operator, with respect to its previous state value.
- Counters are well known as **Timers**.
- Counter is the widest application of flip-flops.

Types of counters

Depending on the type of **clock** inputs, counters are of two types:

- Asynchronous counters
- Synchronous counters

Depending on the way in which the **counting progresses**, the synchronous or asynchronous counters are classified as follows –

- Up counters
- Down counters
- Up/Down counters or Bidirectional Counters

Up-Counter:

- An up-counter counts events in increasing order.
- The binary count is incremented by 1 with every input clock pulse.

Down-Counter:

- A down-counter counts events in the decreasing order.
- A binary counter with a reverse count. In a down-counter, the binary count is decremented by 1 with every input clock pulse.

Up-Down-Counter:

- An up-down counter is a combination of an up-counter and a down-counter. It can count in both directions, increasing as well as decreasing.

Asynchronous Counters

- Clock input of the flip-flops are not all driven by the same clock signal, therefore called Asynchronous Counters.
- The counters in which the change in transition doesn’t depend upon the clock signal input is known as “Asynchronous counters”.
- Asynchronous Counters use flip-flops which are serially connected together so that the input clock pulse appears to ripple through the counter.
- Asynchronous counters don’t use universal clock, only first flip flop is driven by main clock.

Asynchronous Counters

Count Sequence for a Binary Ripple Counter

Count Sequence				Conditions for Complementing Flip-Flops	
A_4	A_3	A_2	A_1		
0	0	0	0	Complement A_1	
0	0	0	1	Complement A_1	A_1 will go from 1 to 0 and complement A_2
0	0	1	0	Complement A_1	
0	0	1	1	Complement A_1	A_1 will go from 1 to 0 and complement A_2; A_2 will go from 1 to 0 and complement A_3
0	1	0	0	Complement A_1	
0	1	0	1	Complement A_1	A_1 will go from 1 to 0 and complement A_2
0	1	1	0	Complement A_1	
0	1	1	1	Complement A_1	A_1 will go from 1 to 0 and complement A_2; A_2 will go from 1 to 0 and complement A_3; A_3 will go from 1 to 0 and complement A_4
1	0	0	0	and so on . . .	

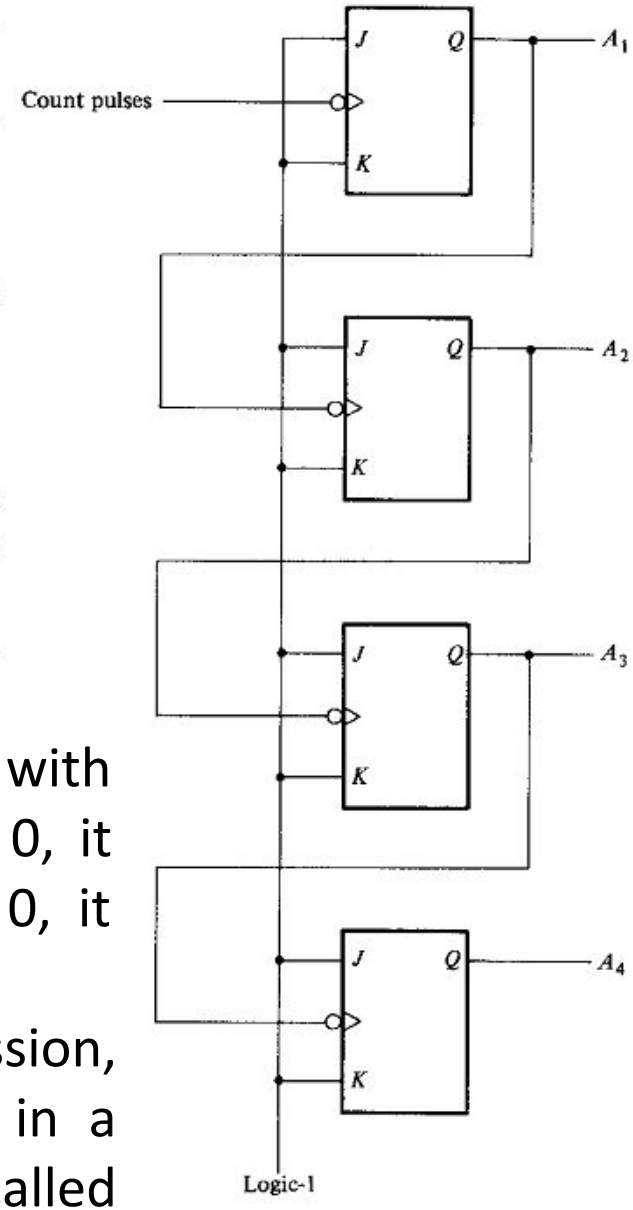


Fig: 4-bit Ripple counter

- All J and K inputs are equal to 1.
- The lowest-order bit A_1 must be complemented with each count pulse. Every time A_1 goes from 1 to 0, it complements A_2. Every time A_2 goes from 1 to 0, it complements A_3, and so on.
- The flip-flops change one at a time in rapid succession, and the signal propagates through the counter in a ripple fashion So asynchronous counters are also called **Ripple** counters.

Asynchronous Counters

Count Sequence for a Binary Ripple Counter

Count Sequence					
A_4	A_3	A_2	A_1	Conditions for Complementing Flip-Flops	
0	0	0	0	Complement A_1	
0	0	0	1	Complement A_1	A_1 will go from 1 to 0 and complement A_2
0	0	1	0	Complement A_1	
0	0	1	1	Complement A_1	A_1 will go from 1 to 0 and complement A_2; A_2 will go from 1 to 0 and complement A_3
0	1	0	0	Complement A_1	
0	1	0	1	Complement A_1	A_1 will go from 1 to 0 and complement A_2
0	1	1	0	Complement A_1	
0	1	1	1	Complement A_1	A_1 will go from 1 to 0 and complement A_2; A_2 will go from 1 to 0 and complement A_3; A_3 will go from 1 to 0 and complement A_4
1	0	0	0	and so on . . .	

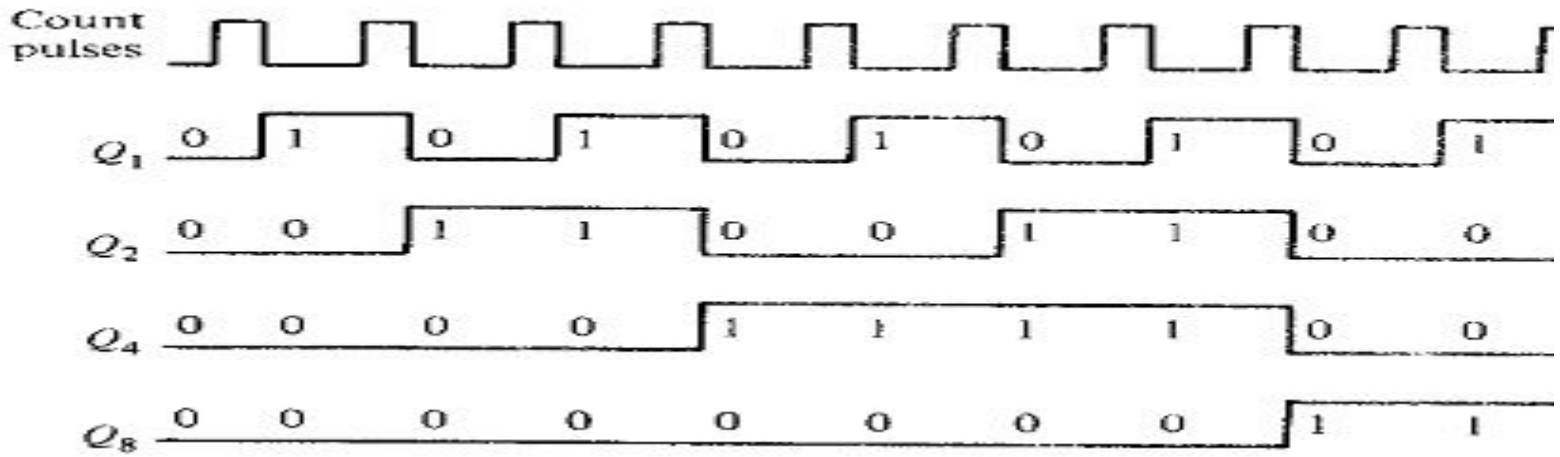


Fig: Timing Diagram

Synchronous Counters

- All flip flops in the synchronous counters are triggered by same clock signal.
- Synchronous Counters are so called because the clock input of all the individual flip-flops within the counter are all clocked together at the same time by the same clock signal.
- **Synchronous Counter**, the external clock signal is connected to the clock input of EVERY individual flip-flop within the counter so that all of the flip-flops are clocked together simultaneously (in parallel) at the same time giving a fixed time relationship.

Synchronous Counters

- The flip-flop in the lowest-order position is complemented with every pulse.
- A flip-flop in any other position is complemented with a pulse provided all the bits in the lowest-order position are equal to 1, because the lowest-order bits (when all 1's) will change to 0's on the next count pulse.
- Synchronous binary counters have a regular pattern and can easily be constructed with complementing flip-flops and gates.
- Synchronous Counters are faster and more reliable as they use the same clock signal for all flip-flops.

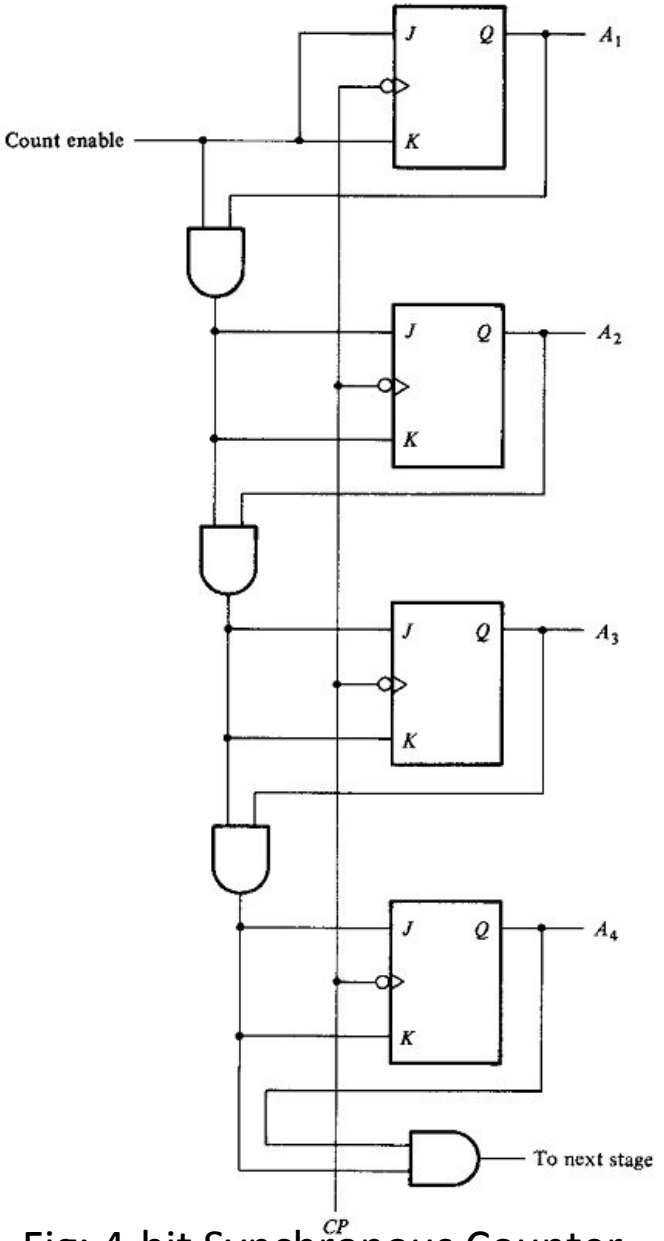


Fig: 4-bit Synchronous Counter

Asynchronous Vs Synchronous Counters

SYNCHRONOUS COUNTERS	ASYNCHRONOUS COUNTERS
All flip-flops are given the same clock simultaneously	The flip-flops are not given the same clock
There is no connection between the output of a flip-flop and the clock input of the next flip-flop.	The output of a flip-flop is given as the clock input to the next flip-flop
These are faster than that of ripple counters.	These are slow in operation.
Large number of logic gates are required to design	Less number of logic gates required.
It is known as a parallel counter	It is known as a serial counter
Synchronous circuits are easy to design.	Complex to design.
Standard logic packages available for synchronous.	For asynchronous counters, Standard logic packages are not available.

Application of counters

- Digital clock
- Time measurement
- Frequency counters
- Analog to digital convertors.
- Frequency divider circuits
- Digital triangular wave generator.
- In time measurement. That means calculating time in timers such as electronic devices like **ovens** and **washing** machines.