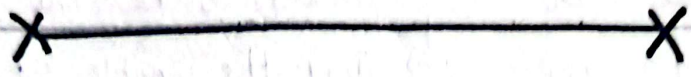
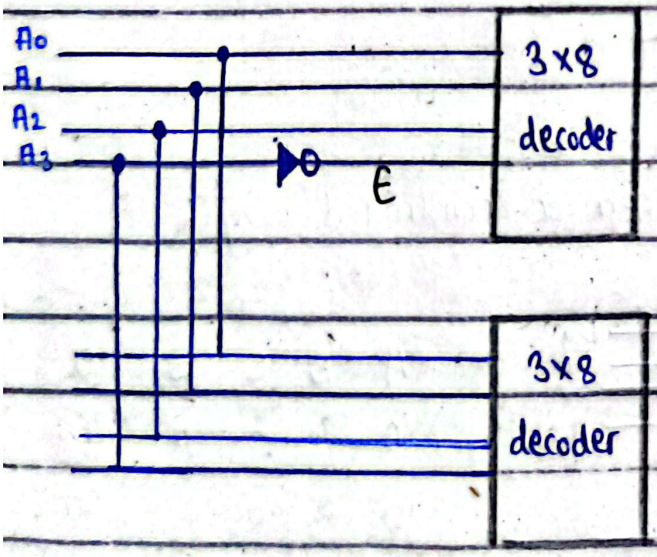
1) Implement a logical function using a decoder :-
2) Implement a 4x16 decoder using 3x8 decoder(s) :-
3) Implement a 4x16 decoder using 2x4 decoder(s) :-
4) Differentiate between active high and high low :-
5) Implement any SOP form equation using decoders and OR gates :-



One of the applications of a decoder is that a 4x16 decoder can be used to construct a BCD to decimal decoder.

⇒ Without enable inputs, we cannot construct higher order decoders, using the lower order decoders.

⇒ 4x16 Decoder using 3x8 Decoders



The two 3x8 decoders are used, with an extra bit, as the enable bit.

⇒ When A3 is equal to 1 (active high) then the second decoder will get enabled.

→ Providing a decoder with the enable bit as 0 disables the decoder.

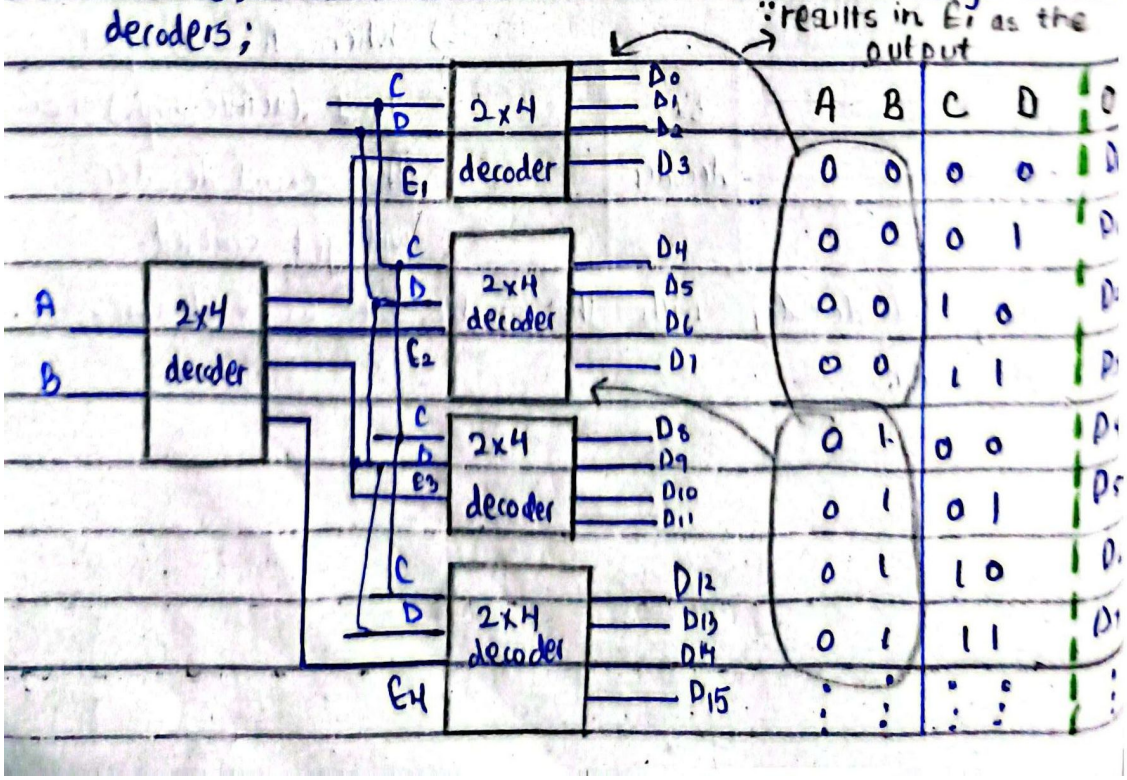
(enable)

E	X	Y	Z	O
0	0	0	0	D_0
0	0	0	1	D_1
0	0	1	0	D_2
0	0	1	1	D_3
0	1	0	0	D_4
0	1	0	1	D_5
0	1	1	0	D_6
0	1	1	1	D_7
1	0	0	0	D_8
1	0	0	1	D_9
1	0	1	0	D_{10}
1	0	1	1	D_{11}
1	1	0	0	D_{12}
1	1	0	1	D_{13}
1	1	1	0	D_{14}
1	1	1	1	D_{15}

⇒ This constitutes the truth table of the 4x16 decoder.
↳ It can be constructed using 3x8 decoders, since, we observe that the input combination across the two individual 3x8 decoders is the same.
↳ Only the enable bit is switched to active high/low, and provided as an input, which constitute the 4 inputs to 16 combinations.

⇒ 16 output lines.

⇒ Similarly, a 4x16 decoder can be constructed using 2x4 decoders;



∴ results in E_1 as the output

A	B	C	D	O
0	0	0	0	D_0
0	0	0	1	D_1
0	0	1	0	D_2
0	0	1	1	D_3
0	1	0	0	D_4
0	1	0	1	D_5
0	1	1	0	D_6
0	1	1	1	D_7
⋮	⋮	⋮	⋮	⋮

⇒ Beauty lies in the truth table.

⇒ A 4x16 decoder is constructed using 21 · 2x4 decoders whose collective outputs constitute the 16 outputs.

↳ An extra 2x4 decoder is required, whose outputs act as enable bits to the rest of the four 2x4 decoders.

⇒ For every combination of A and B, the combinations of C and D are the same, and therefore, the inputs C and D, are made common ; ⇒ 4 inputs to 16 outputs (4x16 decoder)

■ $X + \bar{X}Y = (X + \bar{X})(X + Y)$

$X + \bar{X}Y = X + Y$

e.g. $B = D3 + \underbrace{\bar{D3}\,\bar{D2}D1}_{X}$

$\therefore B = D3 + \bar{D3}X \; ; \; X + \bar{X}Y = X + Y$

$\Rightarrow B = D3 + X = D3 + \bar{D2}D1$

Multiplexer

Implementation of Boolean function using MUX

- Suppose only one multiplexer and one inverter are allowed to be used to implement any Boolean function of n variables. What is the minimum size of the multiplexer required?

⇒ The inverter implies a NOT gate.

In order to implement a boolean function using a MUX :-

i- A boolean function with n variables can be implemented with $(n-1)$ select lines.

∴ A 3 variable boolean function can be implemented with a 4x1 MUX (which has 2 selection lines).

ii- During the implementation, starting from the MSB, the first $(n-1)$ variables are connected to the selection lines and the last variable is connected to the data lines.

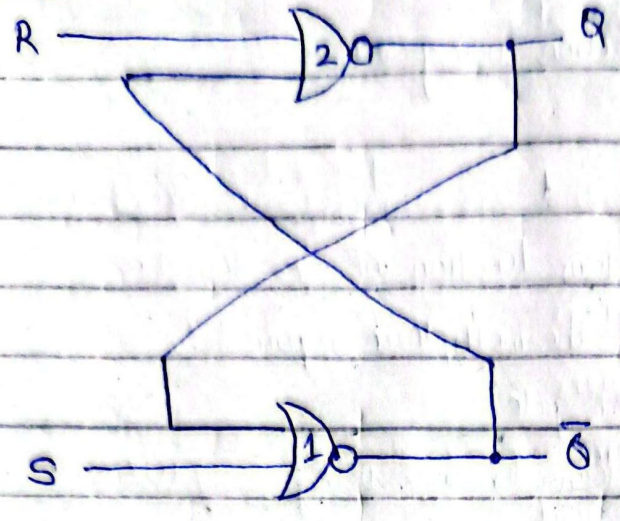
↳ Compare the last variable with the output value, and check their relationship. If the last variable is, e.g., C, then; $\{\bar{C}, C, 0, 1\}$ possible values for the data lines.

∴ The minimum number of input lines (data inputs) that the multiplexer needs to have : $\boxed{2^{n-1} \text{ to } 1}$

Latches are level triggered memory elements.

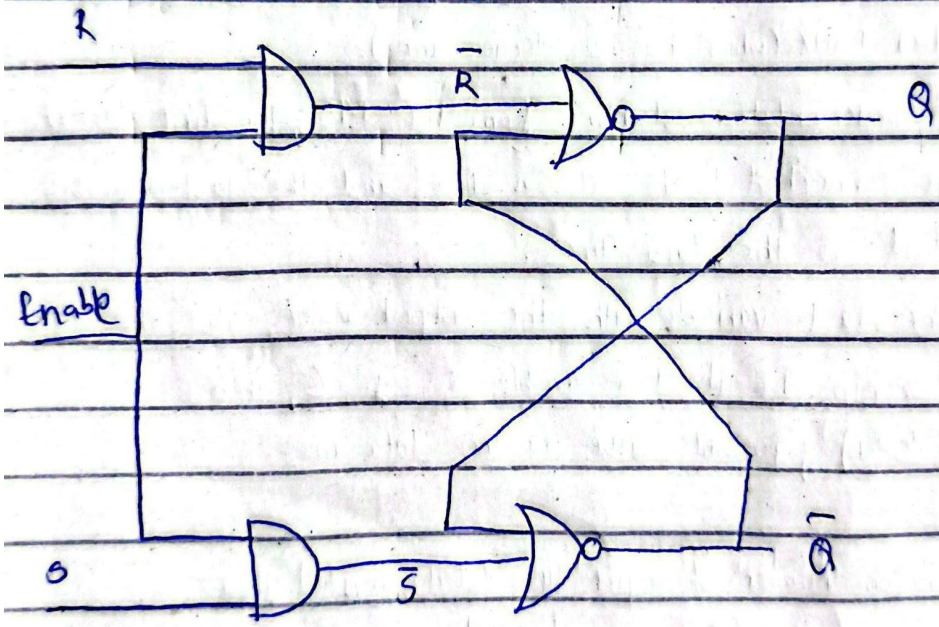
Flipflops are edge triggered memory elements.

SR Latches :



A normal (transparent) SR latch which is made using NAND gate is active low.

For a Gated SR Latch :



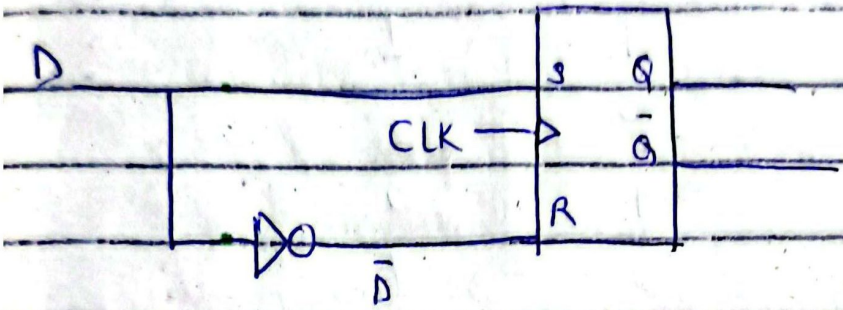
TRUTH TABLE :-

E	$\bar{S}$	$\bar{R}$	Q(Next State)	$\bar{Q}$ (Next state)
0	X	X	Present state	Present state
1	0	0	Present state	Present state
1	0	1	0 (RESET)	1
1	1	0	1 (SET)	0
1	1	1	1	1

⇒ A latch is an asynchronous memory element.

- A flipflop is preferred over gated SR latches in a synchronous sequential circuits.

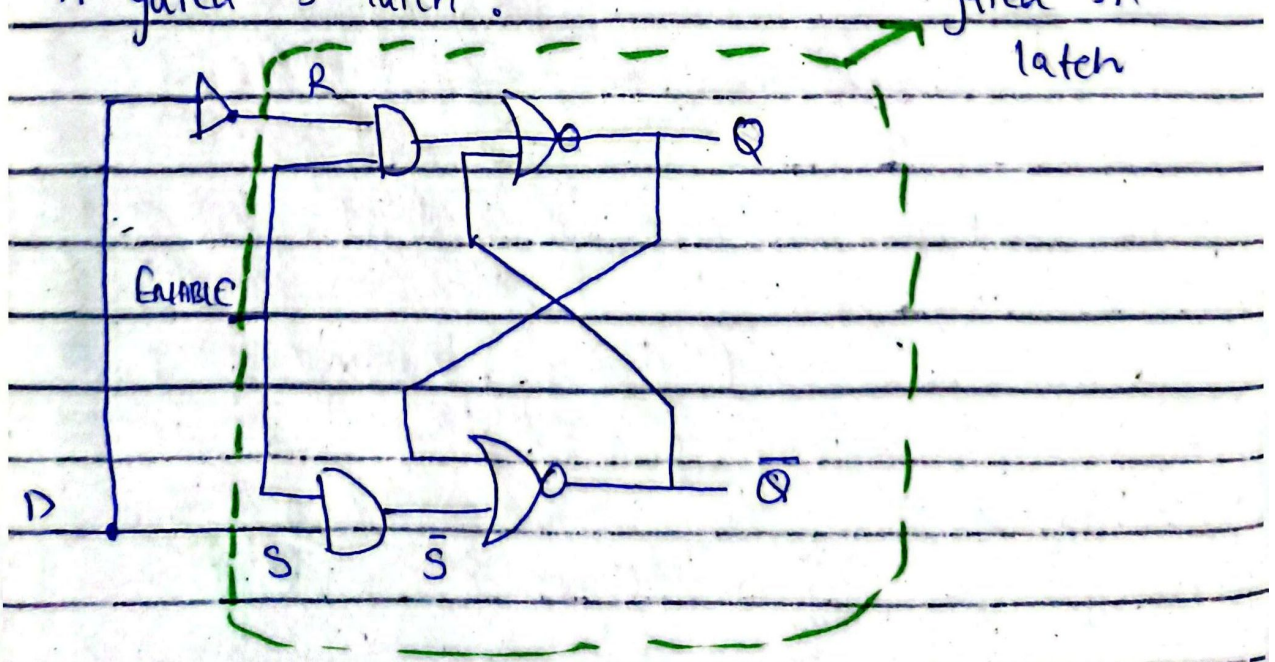
For a D-flipflop:



→

D	Q	$\bar{Q}$
1	1 (SET)	0
0	0 (RESET)	

A gated D latch:



For a T-flip flop :

Truth table :-

CLK	$\bar{T}$	Q_{n+1}
↑	0	Q_n
↑	1	$\bar{Q}_n$

S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

Q_n circulates back to the input logic } making the flip flop sequential

J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

From the truth table, we can write Q_{n+1} as a function of J, K, Q_n :

JK \ Q_n	0	1
00		1
01		
11	1	
10	1	1

$$Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$$

characteristic equation

An **excitation table** lists all the possible current states (Q_n) and next states (Q_{n+1}) of a flip-flop and indicates the corresponding input conditions that will produce the required state transitions.

- The JK flip-flop allows toggling of the output state when both inputs (J and K) are 1.

The excitation table for a J-K flip-flop outlines the input combinations required to achieve desired state transitions between the current state (Q_n) and the next state (Q_{n+1}).

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Excitation tables are invaluable in the design of sequential circuits, such as state machines, counters, and memory devices ~~counters~~. They help engineers determine the appropriate inputs for achieving specific state transitions, enabling precise control over the behavior of digital systems.

TRUTH TABLE : it shows the output of a flip-flop for each combination of inputs and current state.

EXCITATION TABLE : it shows the required inputs to move from a current state to a desired next state.

Flip-flop Conversions

QUESTION

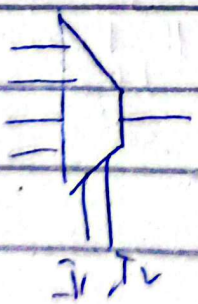
Q. Construct a circuit, using two 4x1 MUX, such that there are three (3) inputs and two (2) outputs, and the circuit operates such that it displays the number of ones in the input, in binary, as the output.
(i.e. input == 101, output == 10).

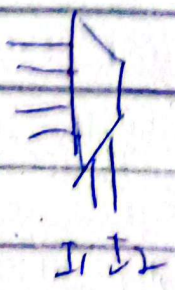
Implementation of Boolean function using MUX :-

i- A Boolean function with n variables can be implemented with a MUX with (n-1) selection lines.

i.e. A 3 variable Boolean function can be implemented with a 4x1 MUX (which has 2 selection lines).

ii- During the implementation, starting from the MSB, the first (n-1) variables are connected to the selection lines and the last variable is connected to the data lines.





I_1	I_2	I_3	O_1	O_2
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

COUNTERS

↳ A register that goes through a prescribed sequence of states upon the application of input pulses is called a counter.

⇒ A counter that follows the binary number sequence is called a binary counter. An n-bit binary counter consists of n flip-flops and can count in binary from 0 through $2^n - 1$.

Counters are available in two (2) categories :

- Ripple counters
- Synchronous counters

REGISTERS

⇒ A register is a group of flip-flops, each one of which shares a common clock and is capable of storing one bit of information.

If an input is supplied, Clear_b, which goes to the active-low R (reset) input of all four flip-flops, when this input goes to 0, all flip-flops are reset asynchronously, that is independently of the clock.

> Active-low = action happens when signal is 0

Synchronous digital systems have a master clock generator that supplies a continuous train of clock pulses.

⇒ Inserting gates into the clock path is ill-advised because it means that logic is performed with ~~to~~ clock pulses. The insertion of logic gates in the path of the clock signal produces uneven propagation delays between the master clock and the inputs to flip-flops.

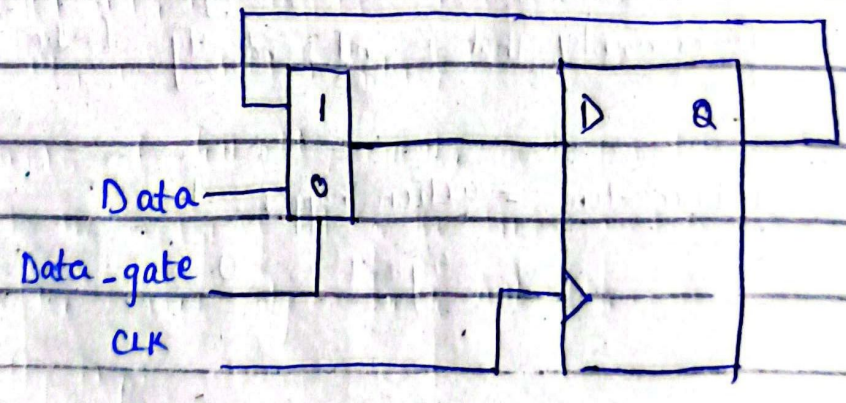
A four-bit data-storage register with a load control input that is directed through gates and into the D inputs of the flipflops is created.

⇒ The additional gates implement a two-channel MUX whose output drives the input to the register with either the data bus or the output of the register.

⇒ When the load input is 1, the data at the four external inputs are transferred into the register with the next positive of the clock.

When the load input is 0, the outputs of the flip-flops are connected to their respective inputs ; output unchanged (HOLD).

Q. Draw the logic diagram of a circuit that suspends the clock action of a D flip-flop without gating its clock. Describe the behavior of the circuit.



If Data_gate is 0, Data is transferred to Q with each active edge of the clock. If Data-gate is 1, the value of Q is recirculated through the MUX-flipflop path, with the effect that the clock appears to be suspended.

→ sequential circuit (uses flip-flop)

SERIAL ADDITION

The two binary numbers to be added serially, are stored in two shift registers.

The operation of the serial adder is as follows:

Initially, register A holds the augend, register B holds the addend, and the carry flip-flop is cleared to 0. The outputs (SO) of A and B provide a pair of significant bits for the full adder at x and y. Output Q of the flip-flop provides the Cin input to the full adder. The shift control enables both registers and the carry flip-flop, so at the next clock pulse, both registers are shifted once to the right, the sum bit from S enters the leftmost flip-flop of A, and the output carry is transferred into flip-flop Q.

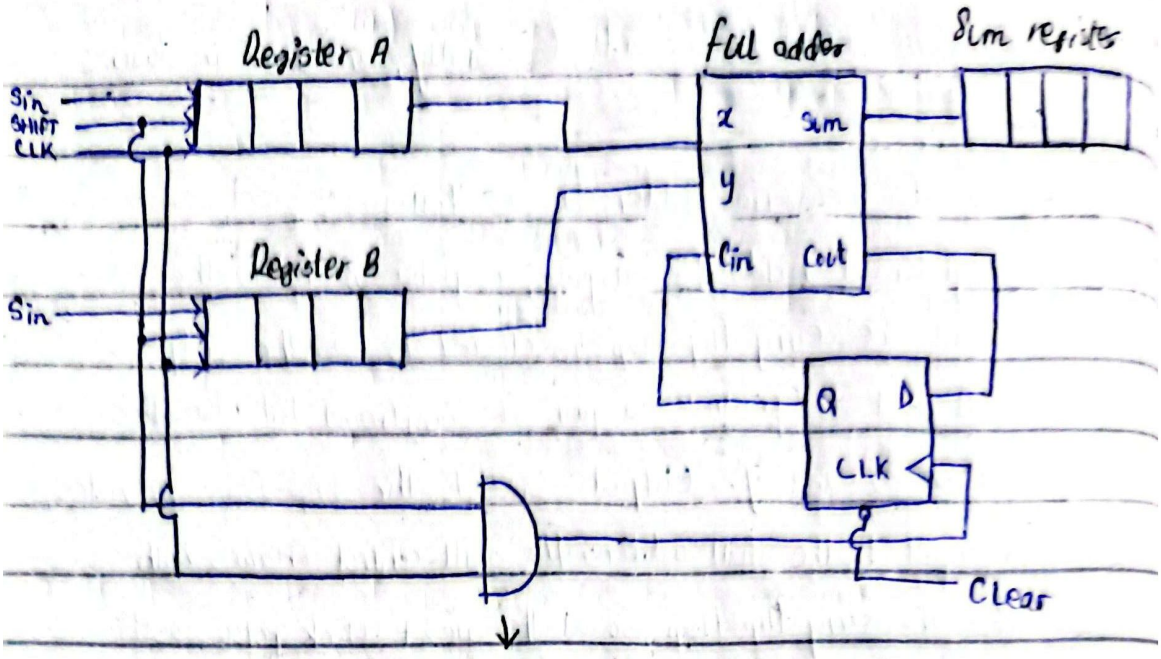
⇒ The shift control enables the registers for a number of clock pulses equal to the number of bits in the registers.

↳ eg: If two four-bit inputs are applied to the serial input of the two registers, then after four (4) clock pulses, the two numbers will be shifted into the shift register.

⇒ At the fifth (5) clock pulse, the sum output, S will be shifted into the sum register, and, the carry out, Cout, will be stored in the D-flip-flop.

- In the case of addition of two 4-bit numbers;
  - ⇒ 4 CLK cycles - loading/shifting the data in shift registers
  - 4 CLK cycles - performing the addition

∴ for adding two N bit numbers, using the serial addition circuit, with serial load, the required clock cycles = 2N.



Only allows the addition essentially on the posedge of clock, and when the shift control signal is enabled.

- Comparing the serial adder with the parallel adder:
  - The parallel adder uses registers with a parallel load, whereas the serial adder uses shift registers.
  - The number of full adder circuits in the parallel adder is equal to the number of bits in the binary numbers, whereas the serial adder requires only a single full adder circuit and a carry flip-flop.

The serial adder can be redesigned by means of sequential circuit procedure. (using JK flip-flops)

⇒ Present state (Q) = C_{in} | INPUTS: x, y, Q (Q → C_{in})

Next state (Q^+) = C_{out}

⇒ The flip-flop Q is used for storing the carry.

↓

can be any flip-flop

	Present State	Inputs		Next state	Sum	FLIP-FLOP INPUTS	
	Q	x	y	Q^+	S	J_Q	K_Q
m_0	0	0	0	0	0	0	X
m_1	0	0	1	0	1	0	X
m_2	0	1	0	0	1	0	X
m_3	0	1	1	1	0	1	X
m_4	1	0	0	0	1	X	1
m_5	1	0	1	1	0	X	0
m_6	1	1	0	1	0	X	0
m_7	1	1	1	1	1	X	0

xy \ Q	0	1
00	m_0	m_1
01	m_2	m_3 1
11	m_6 X	m_7 X
10	m_4 X	m_5 X

$\therefore J_Q = Qy$

We CANNOT form a group using only don't-cares (X).
⇒ They are included in a grap only if they help to enlarge a grap of 1's.

xy \ Q	0	1
00	m_0 X	m_1 X
01	m_2 X	m_3 X
11	m_6	m_7
10	m_4 1	m_5

$\therefore K_Q = \bar{Q}\bar{y} = \overline{(Q+y)}$

$$\text{Sum} = x \oplus y \oplus Q$$

UNIVERSAL SHIFT REGISTER

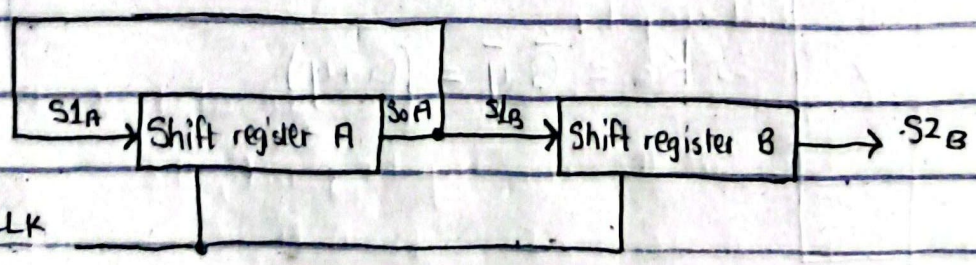
↳ There are four (4) primary operations, and depending upon the inputs of the selection lines of a 4×1 MUX;

S_1	S_0	OUTPUT
0	0	NO CHANGE - 0
0	1	SHIFT RIGHT - 1
1	0	SHIFT LEFT - 2
1	1	PARALLEL LOAD - 3

(i) If the number of operations are to be increased, in a universal shift register, then the size of the multiplexer needs to be increased.

Q. Consider two 4-bit shift registers A and B. The initial binary data of shift register A and B is 1101 and 0101, respectively.

What is the content of shift register B after 3 clock pulses?



⇒ The input of register A = output of register A

⇒ The input of register B = output of register A

clock pulses

Register A

	CLK	Input	A_3	A_2	A_1	A_0
	0	X	1	1	0	1
1	↑	1	1	1	1	0
2	↑	0	0	1	1	1
3	↑	1	1	0	1	1

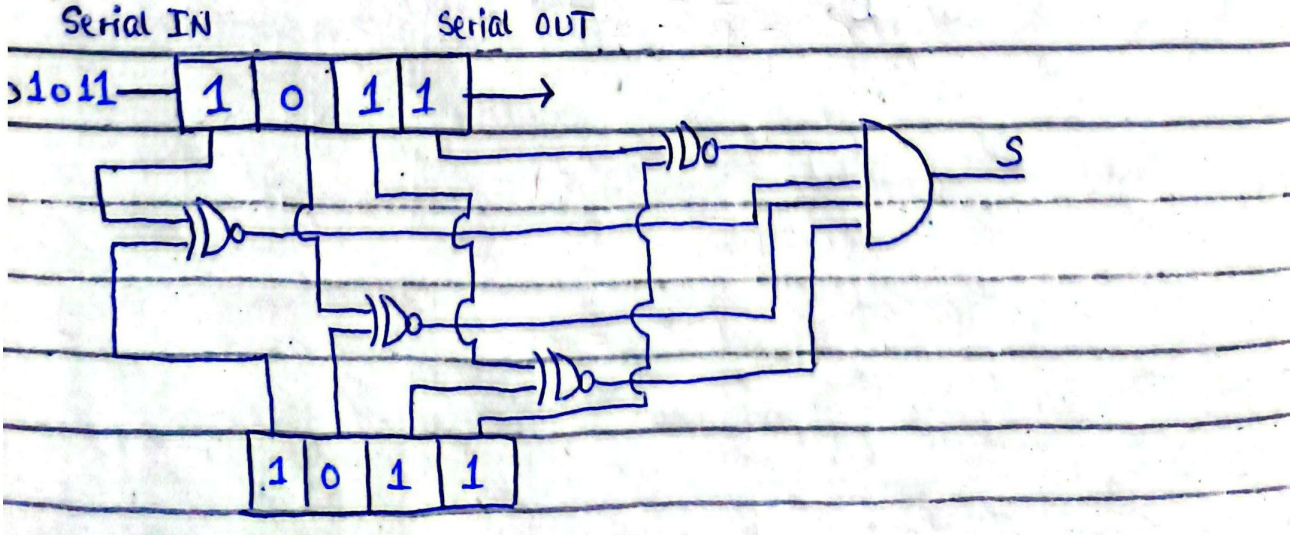
Register B

	CLK	Input	B_3	B_2	B_1	B_0
	0	X	0	1	0	1
1	↑	1	1	0	1	0
2	↑	0	0	1	0	1
3	↑	1	1	0	1	0

⇒ At the first posedge of CLK, the input to the register B is the initial A_0 (LSB of A).

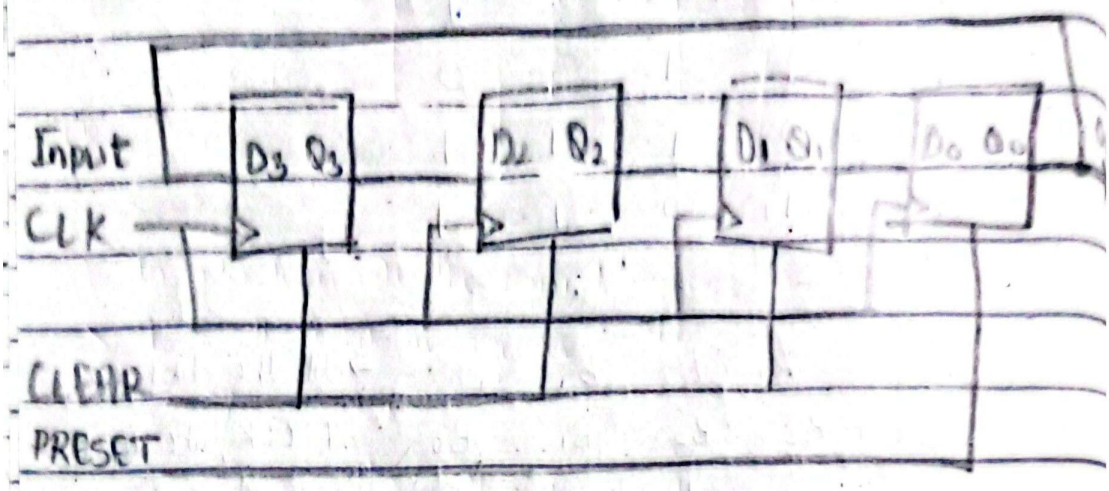
APPLICATIONS OF SHIFT REGISTER

1- Sequence Detection : Data is shifted and loaded into a register of size that is equal to the number of bits in the sequence to detect ;



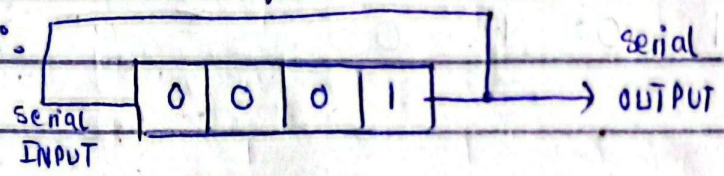
can be used as a frequency divider.

ii- Ring counter



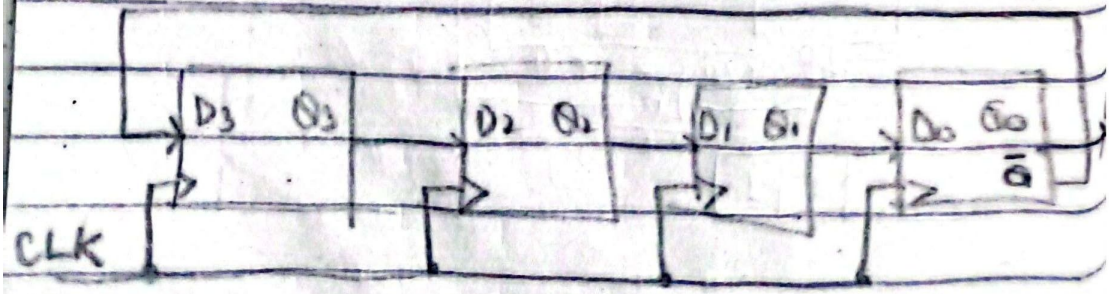
The preset defines a value for the LSB, all the while the CLEAR input signal is asserted, the flip-flops are set to 0.

∴



X	0	0	0	⇔1
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

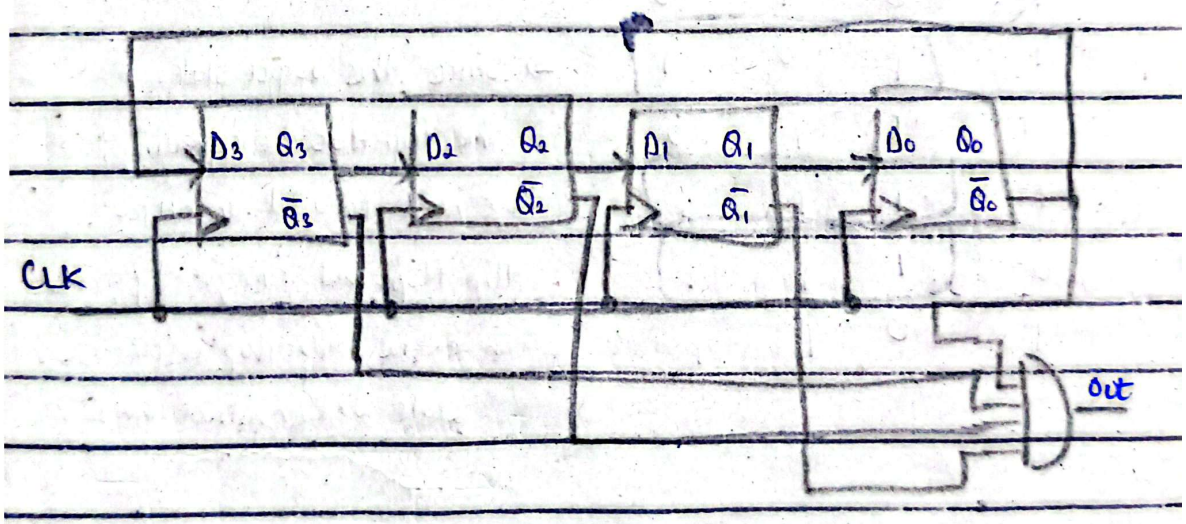
iii- Johnson Counter



CLK	Q_3	Q_2	Q_1	Q_0
X	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3.	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
8	0	0	0	0

After eight (8) clock cycles, the output sequence will get repeated.
⇒ It has 8 different states (MOD - 8 counter).
∴ for an N-bit Johnson counter, there are 2N states.

The circuit can be modified in order to count whether the 8 counts have been completed, and after how many clock pulses;



BINARY SEQUENCE GENERATOR

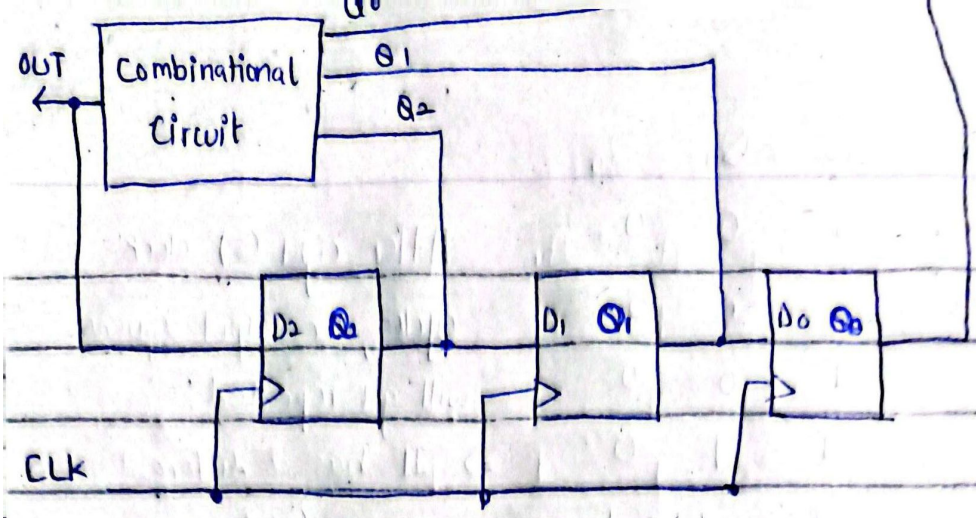
The steps for designing the sequence generator :

i- Find the required number of flip-flops

⇒ L - Length of the sequence

$L \leq 2^N - 1$

} To generate the sequence of length L, the minimum required number of flip-flops are N.



e.g. In order to generate the required sequence of : 1 0 1 1 1 0 0

⇒ L = # of bits in sequence = 7

∴ $7 \leq 2^N - 1$

$8 \leq 2^N$; $N = 3$ flip-flops

- There cannot be a repeating sequence in the table ;

F	Q_2	Q_1	Q_0
0	1	0	1
1	0	1	0
1	1	0	1
1	1	1	0
1	1	1	1
0	1	1	1
1	0	1	1

- The Q_2 column is the input sequence ;

⇒ Since the output states of the flip-flops are repeating, we need to add an extra flip-flop, and consequently an extra flip-flop output state and reconstruct the

∴

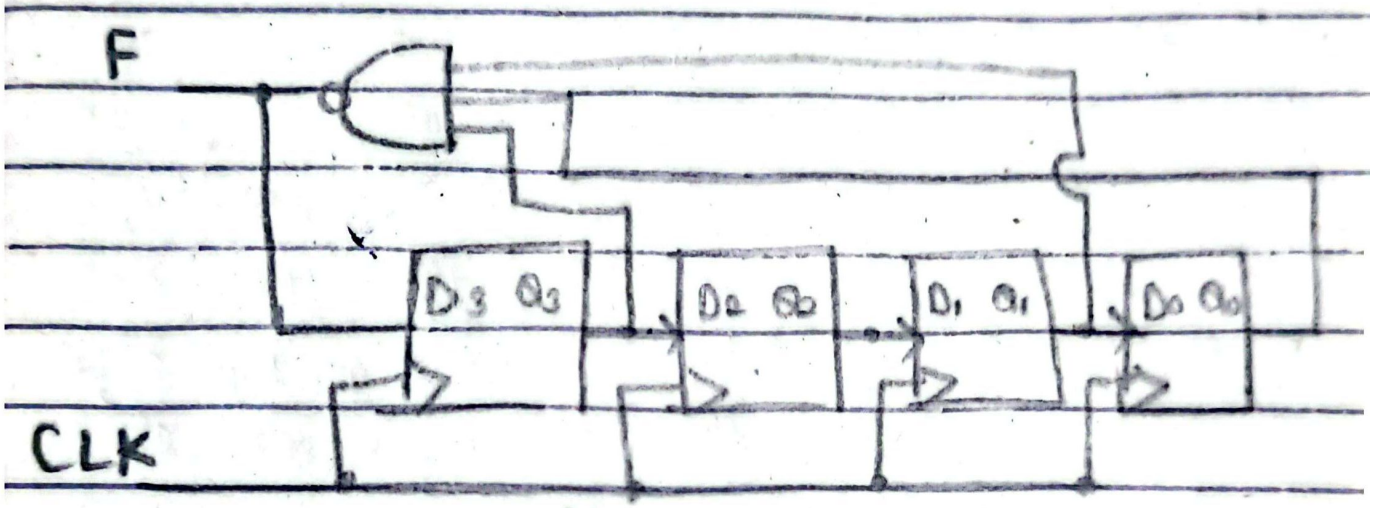
F	Q_3	Q_2	Q_1	Q_0
0	1	0	1	1
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0
0	1	1	1	1
1	0	1	1	1

→ F is essentially the output of the combinational circuit, with Q_3, Q_2, Q_1, and Q_0 as the inputs :

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	x	x	x	x
01	x	1	1	x
11	x	1	0	1
10	x	x	0	1

$\therefore F = \bar{Q_3} + \bar{Q_1} + \bar{Q_0}$

$= (\overline{Q_3 Q_1 Q_0})$



Diagram