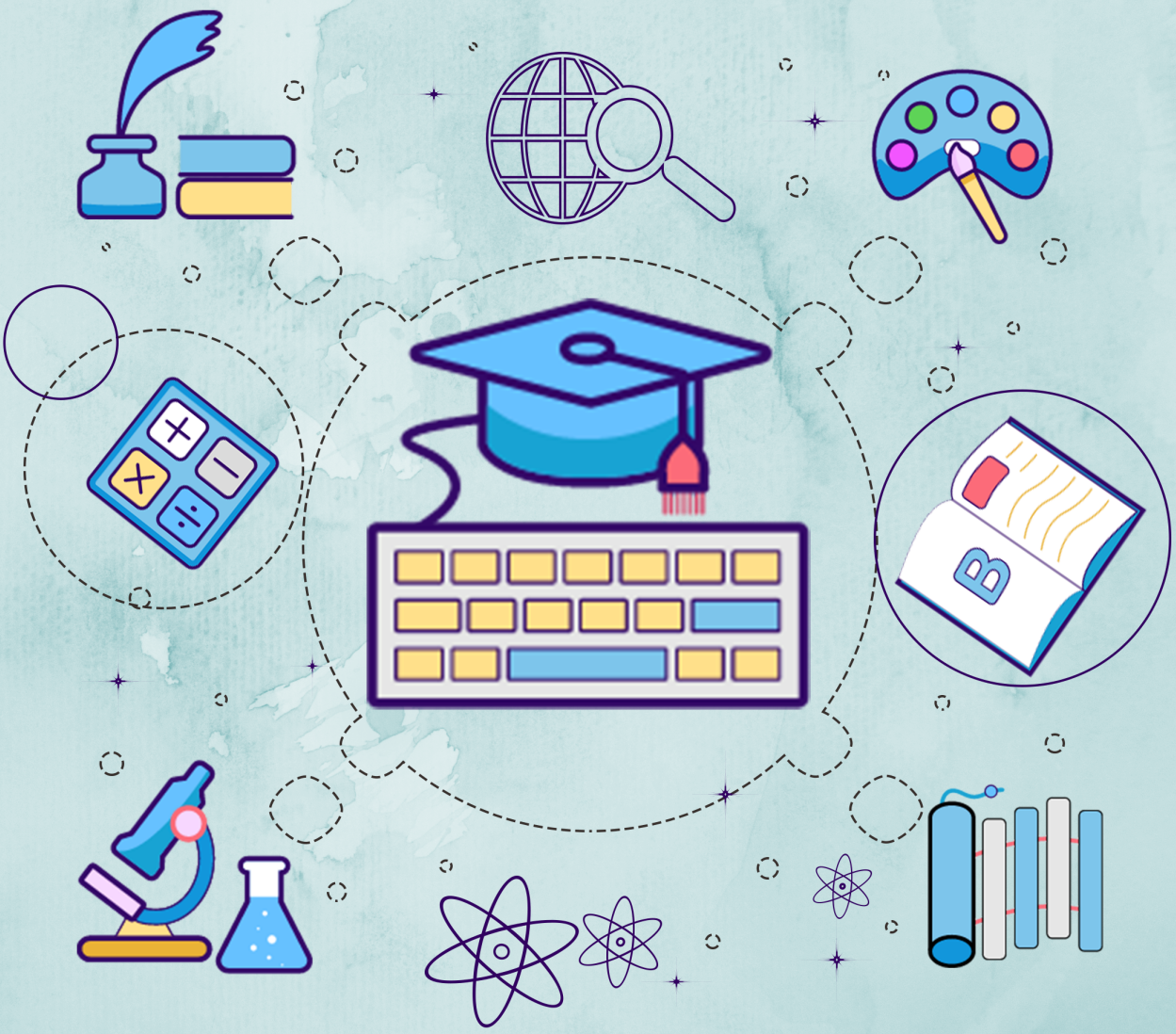
APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

KeralaNotes



SYLLABUS | STUDY MATERIALS | TEXTBOOK

PDF | SOLVED QUESTION PAPERS

www.keralanotes.com

KeralaNotes

KTU STUDY MATERIALS

LOGIC SYSTEM DESIGN

CST 203

Module 5

Related Link :

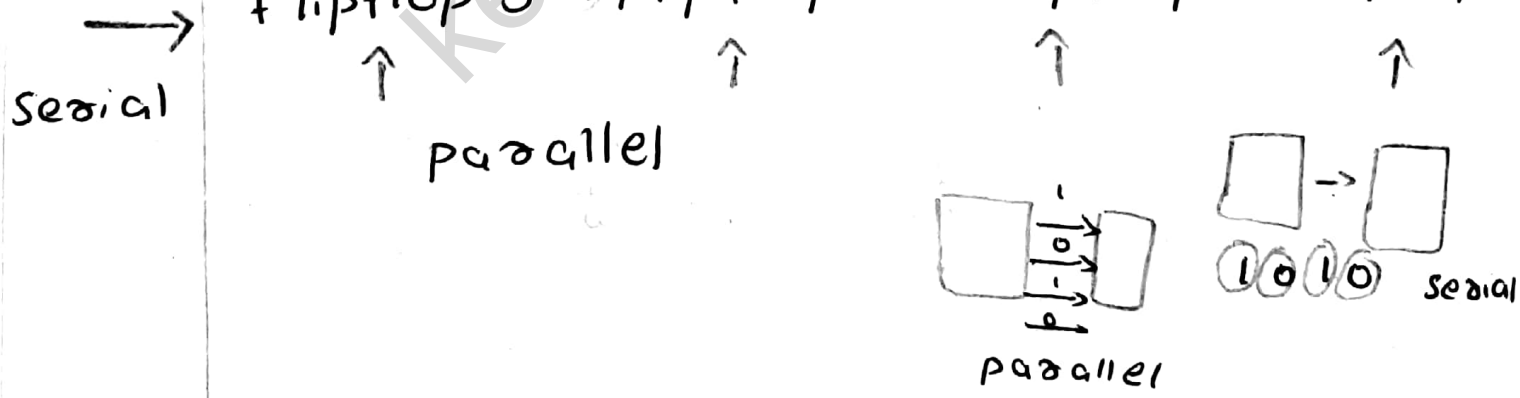
- KTU S3 STUDY MATERIALS
- KTU S3 NOTES
- KTU S3 SYLLABUS
- KTU S3 TEXTBOOK PDF
- KTU S3 PREVIOUS YEAR SOLVED QUESTION PAPER

Introduction to Registers

- Flipflop is 1 bit memory cell
- To increase the storage we have to use group of flipflops. This group of flipflop is known as Registers
- The 'n' bit register consist of 'n' number of flipflop and is capable of storing 'n-bit' word

classification of Registers

- Data can be entered in serial or ~~partial~~ parallel form



Depending upon input and output

(i) SISO [Serial input serial output]

(ii) SIPO [Serial input parallel output]

(iii) PISO [Parallel input serial output]

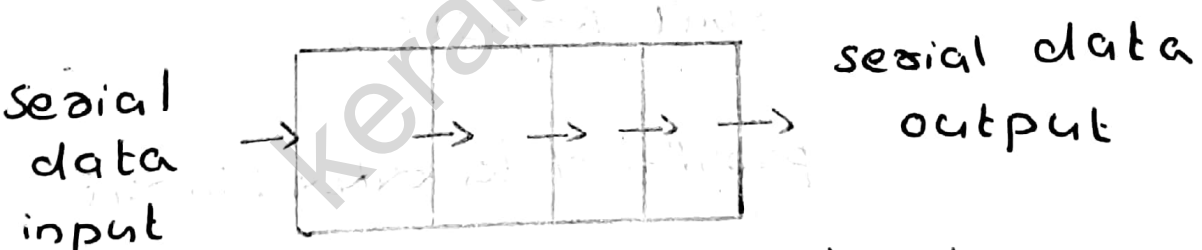
(iv) PIPO [Parallel input parallel output]

Depending upon Application

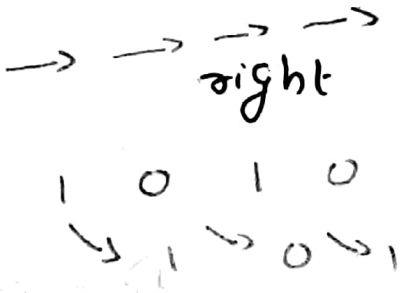
(i) Shift Registers : • SISO, • SIPO, • PISO

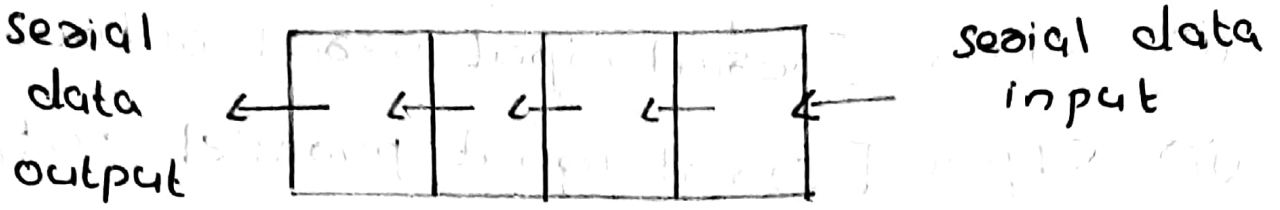
(ii) Storage Registers : PIPO

Data transfer in Registers

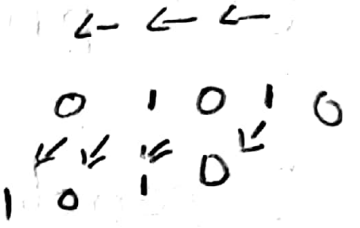


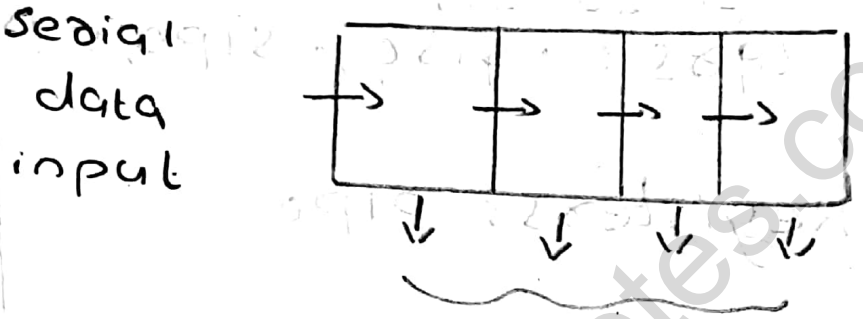
(a) serial-in, serial-out, shift-right, shift Register



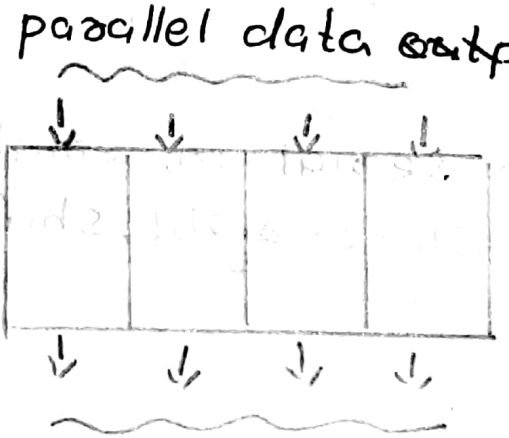


(b) Serial-in, serial-out shift left, shift Register

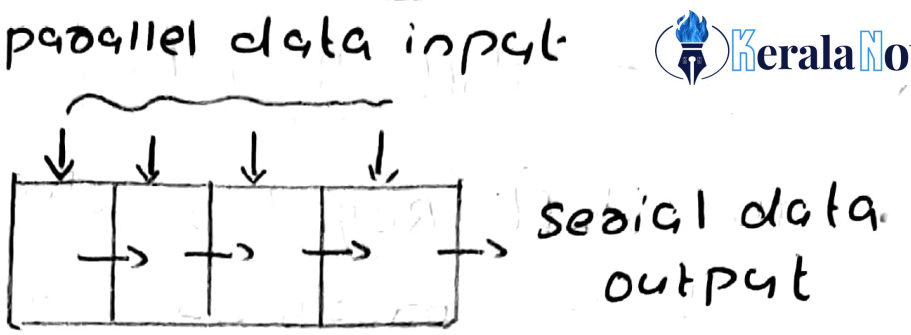




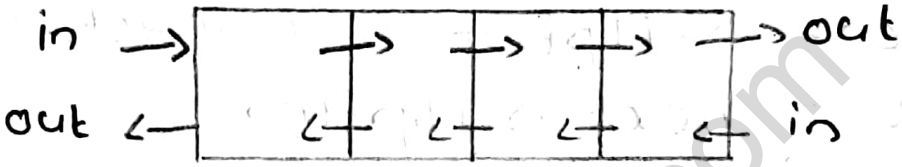
(c) Serial-in, parallel out, shift Register



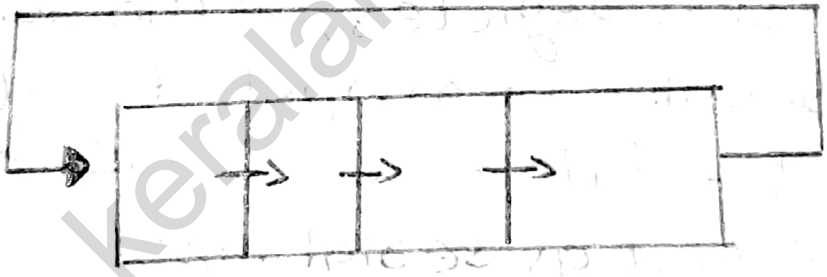
(d) parallel-in, parallel-out, shift Register



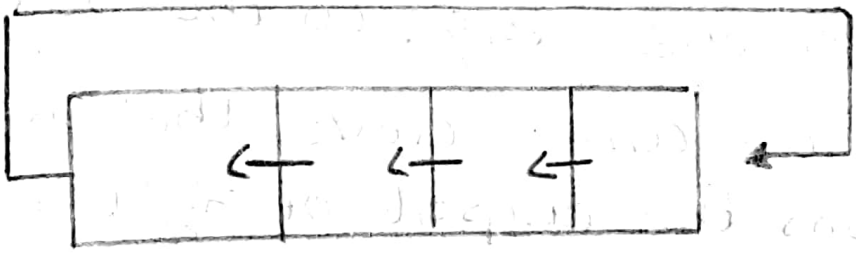
(e) parallel-in, serial out shift register



(f) serial-in, serial-out, shift-left, shift right, (bidirectional) shift register



(g) Rotate-right shift Register



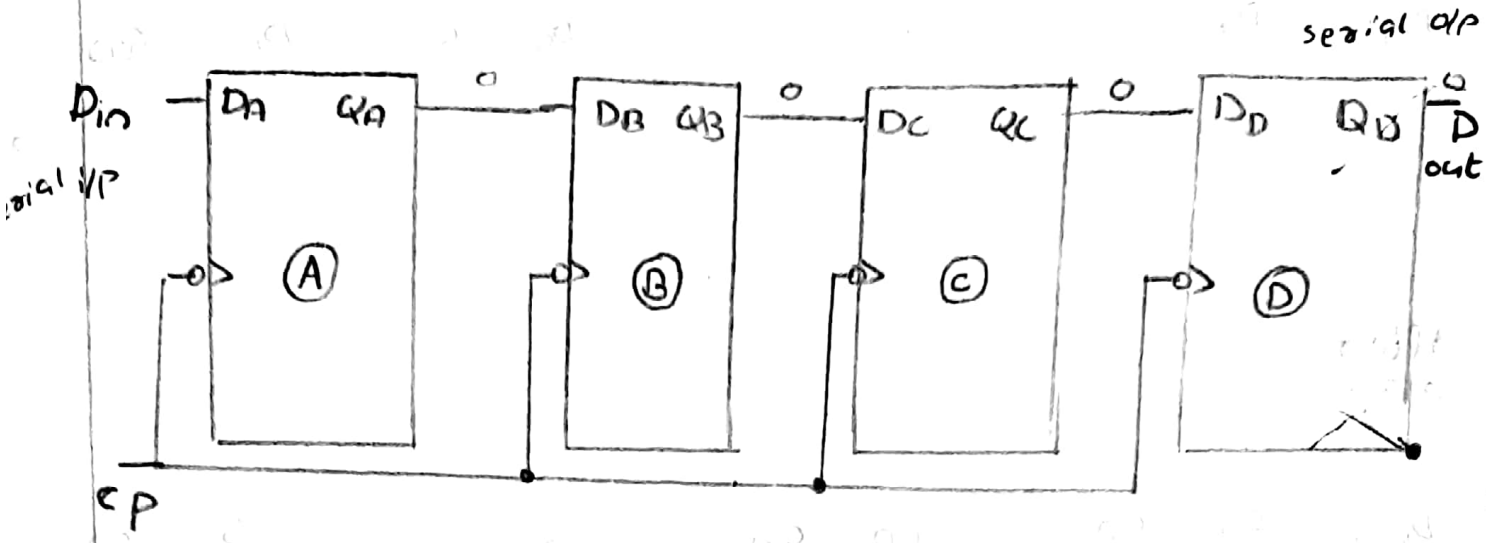
(h) Rotate-left shift Register

Imp

SISO shift Register

- The shift Register which allow serial input and produces serial output is known as serial in - serial out SISO shift Register
- This block diagram consist of D flipflop which are cascaded. That means, output the one D-flipflop is connected as the input of next D flipflop
- All these flipflop are synchronous with each other since, the same clock signal is applied to each one
- In this shift register, we can sent the bits serially from input of left most D flip flop. Hence the input is also called as serial input
- for every positive edge triggering of clock signal, the data shift from one stage to the next
- so we can recieve the bits serially from the output of right most D-flipflop Hence this output is also called serial output

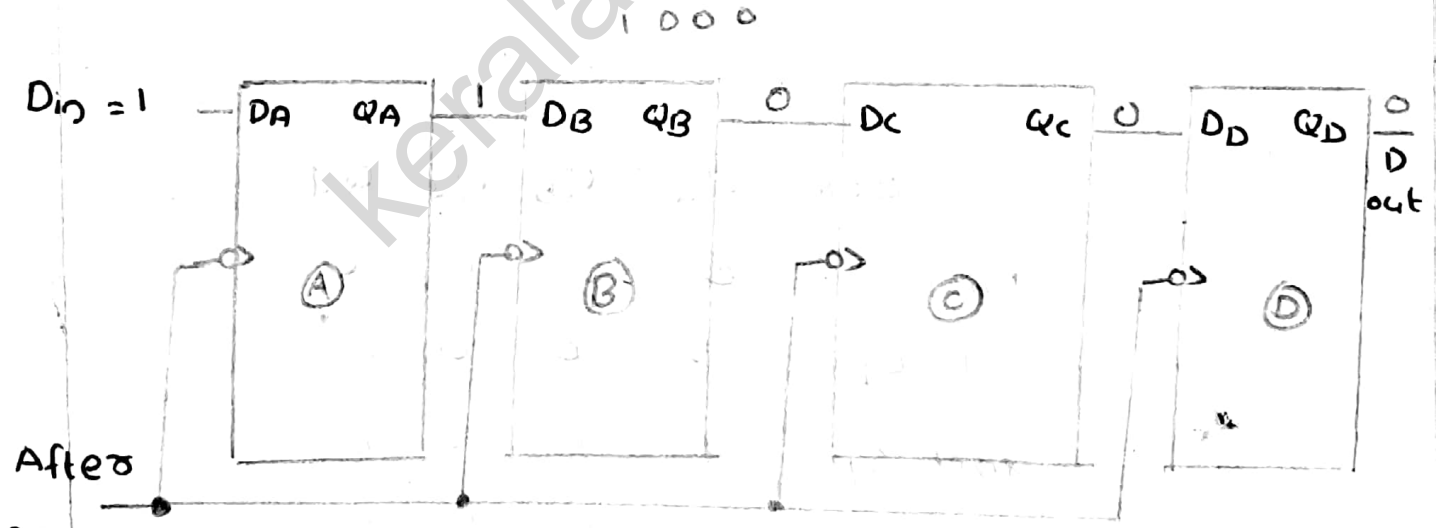
- Figure shows serial-in-serial-out

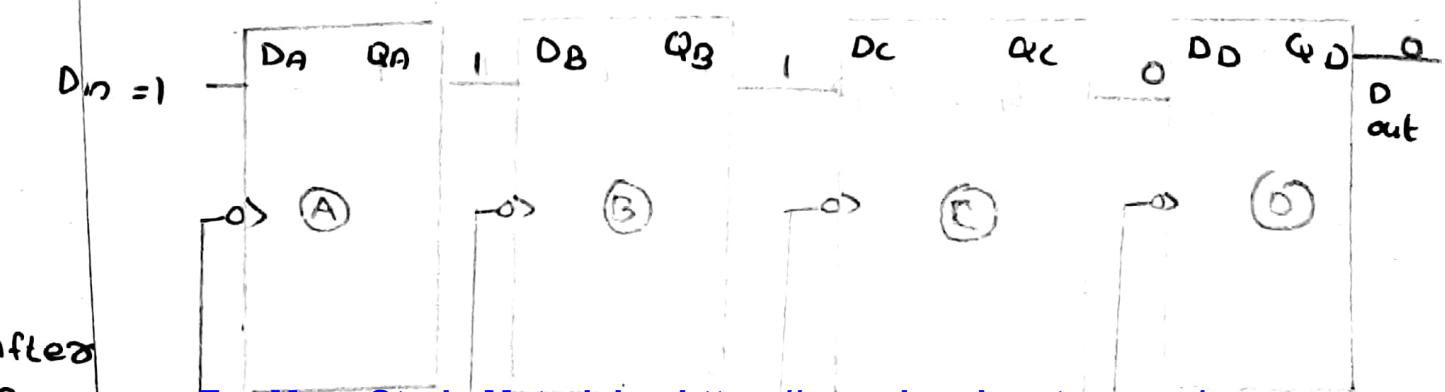


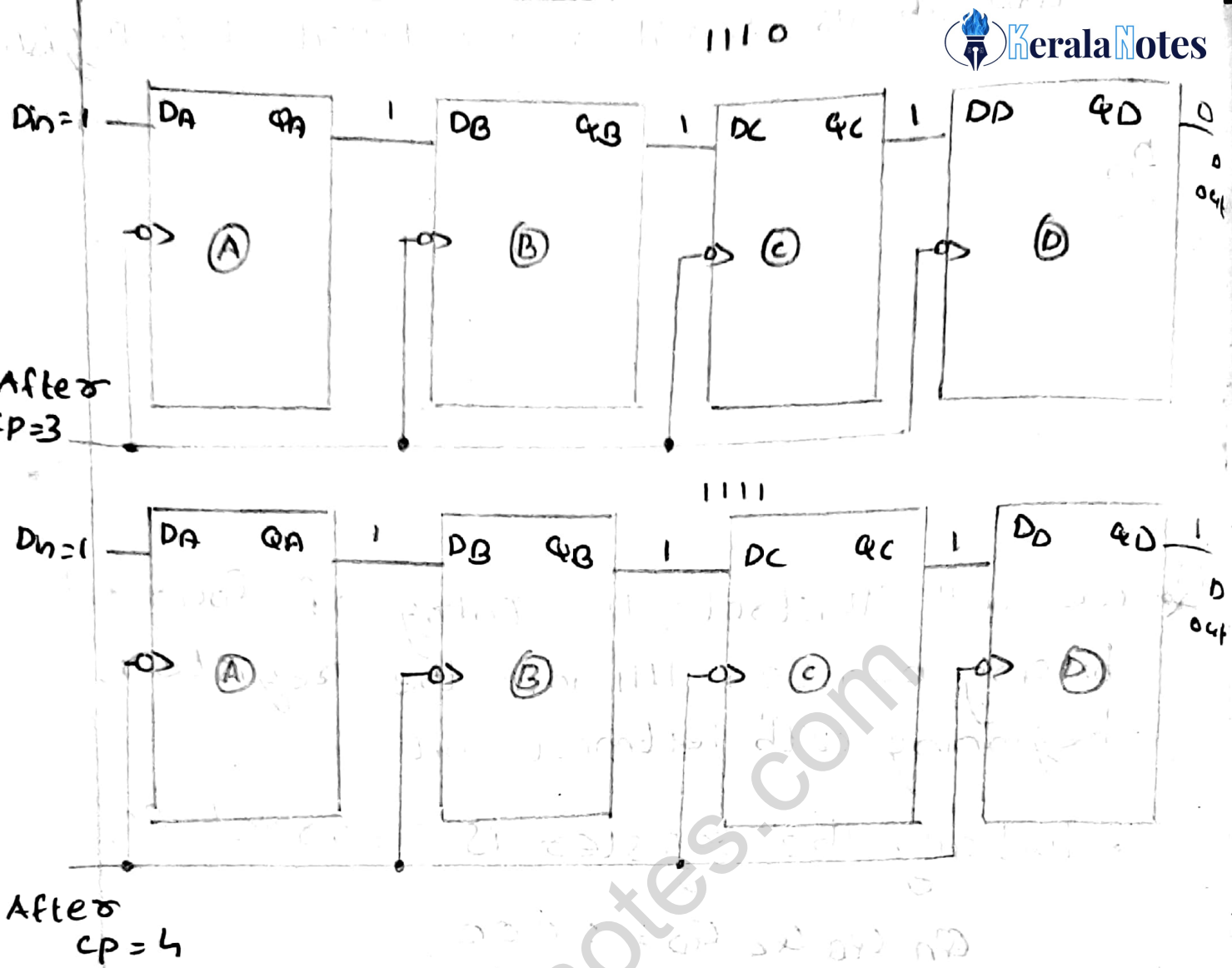
* we will illustrate the Entry of four bit binary number 1111 into the Register, beginning with leftmost bit

- Initially the register is cleared so,

$Q_A\ Q_B\ Q_C\ Q_D = 0\ 0\ 0\ 0$

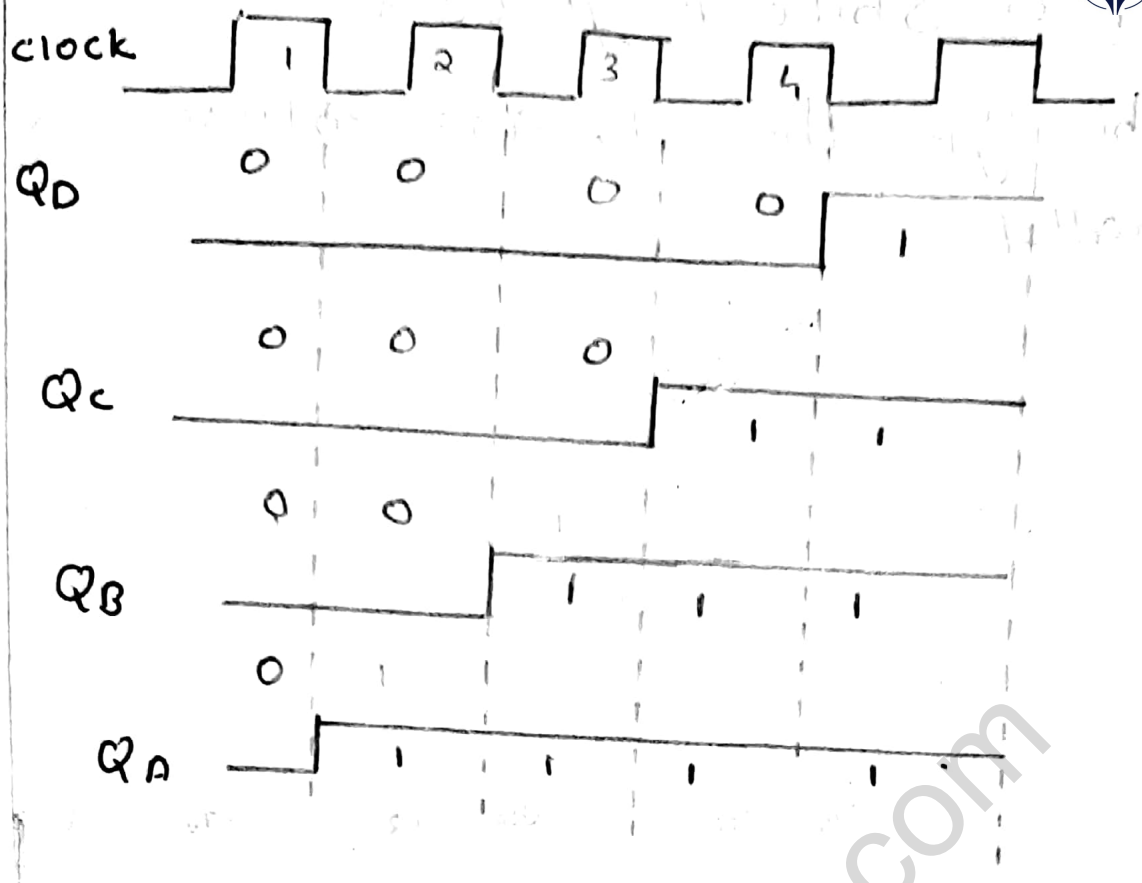






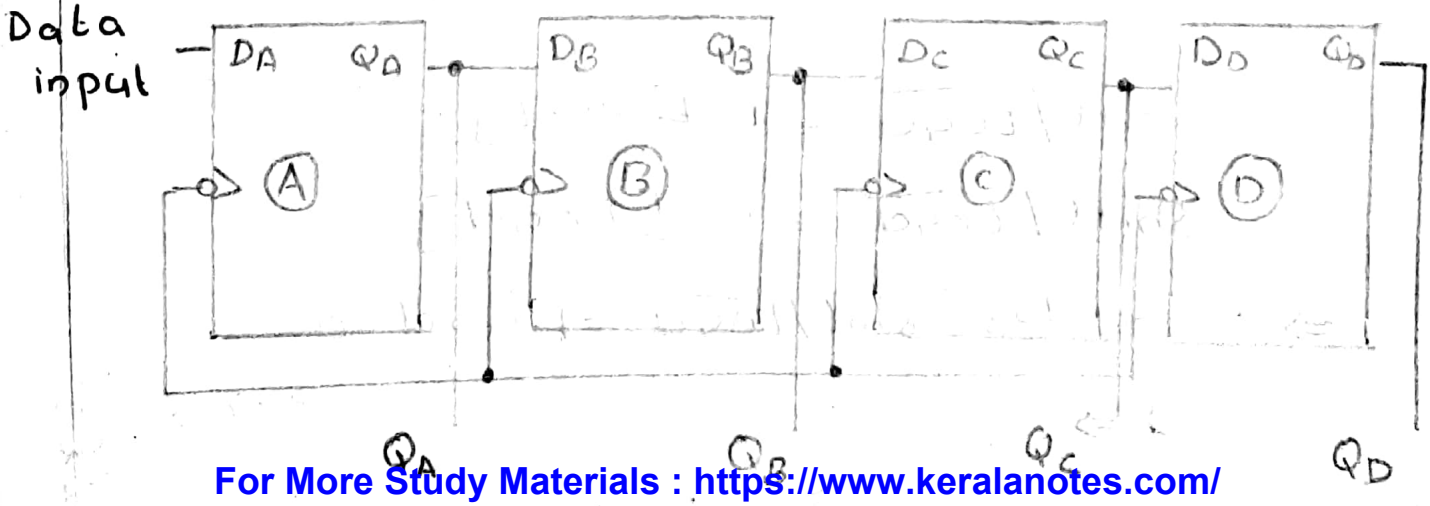
waveform of SISO

clock	Qa	Qb	Qc	Qd
initial	0	0	0	0
1st cp	1	0	0	0
2nd cp	1	1	0	0
3rd cp	1	1	1	0
4th cp	1	1	1	1



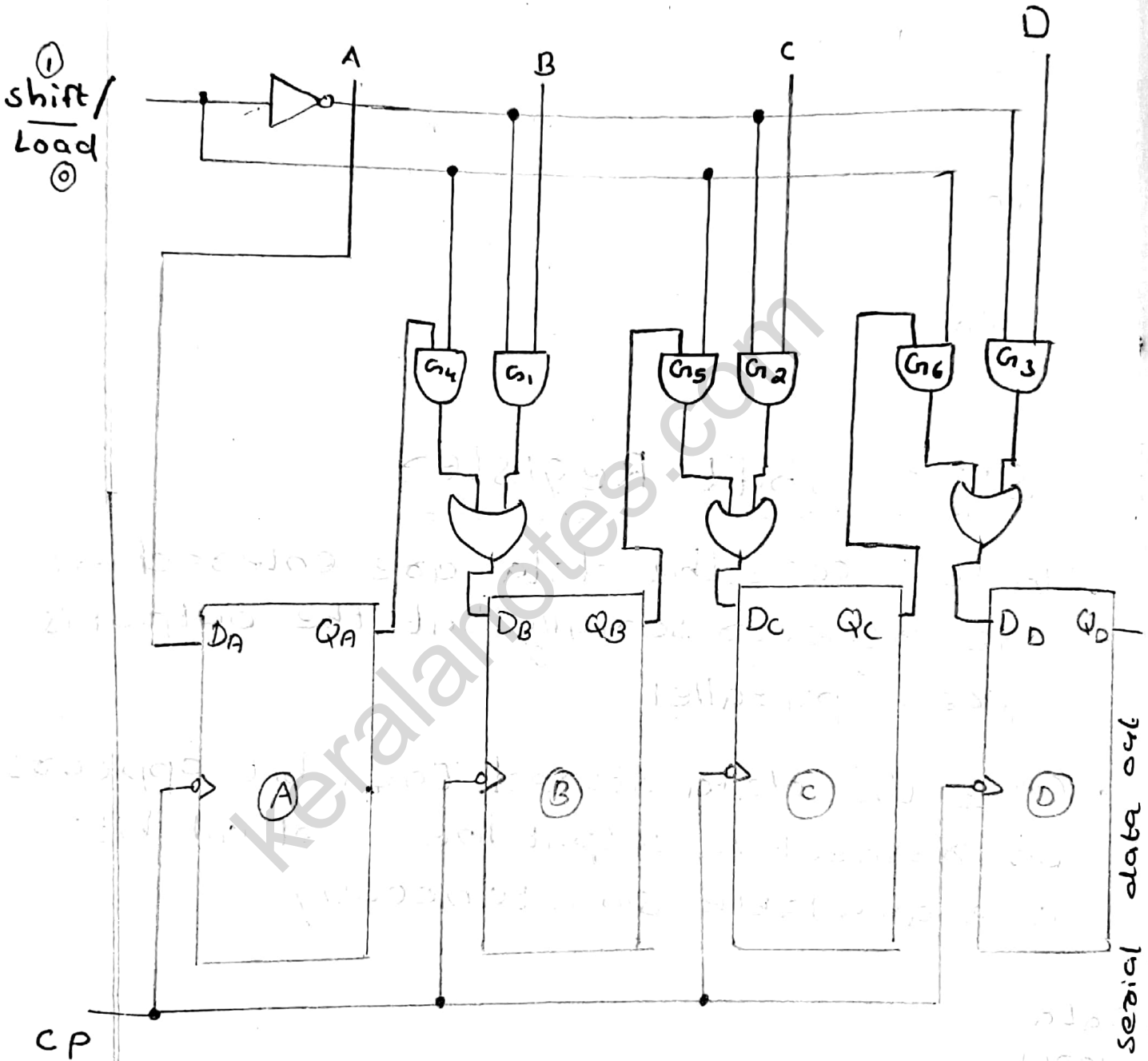
SIPO Shift Register

- In this case the data are entered into the register serially but the output is taken parallel
- Once the data stored, Each bit appears on respective output line and all bits are available simultaneously



PISO Shift Register

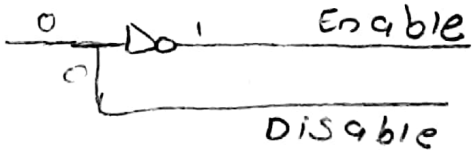
- In this type, the bits are entered in parallel



Shift / $\overline{Load}$ = 1 [Shift]

Shift / $\overline{Load}$ = 0 [Load]

⇒ • Consider Shift/$\overline{Load}$ = 1 and 0

0 ⇒ 

⇒ enable: G1 G2 G3

G_1 G_2 G_3 [Enable]

- It will store in ~~shift~~ flipflop

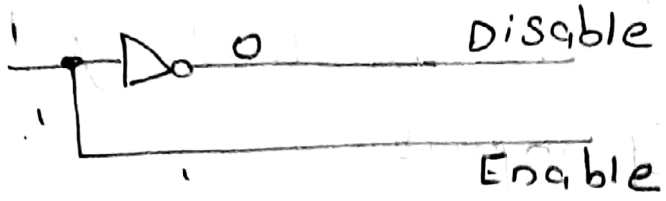
data store

G_1 G_2 G_3 ↑

G_4 G_5 G_6

Data shift

⇒ Shift / $\overline{Load}$ = 1



G_4, G_5 G_6 [Enable]

G_1 G_2 G_3 [Disable]

- there will be data shifting

→ There are 4 input line for entering data in parallel

→ Shift / $\overline{Load}$ is the control input which allow shift or loading data operation of the register

→ when shift / $\overline{Load}$ is low, the gates G_1, G_2, G_3 are enabled, allowing each input data bit to be applied to D input of respective flipflops

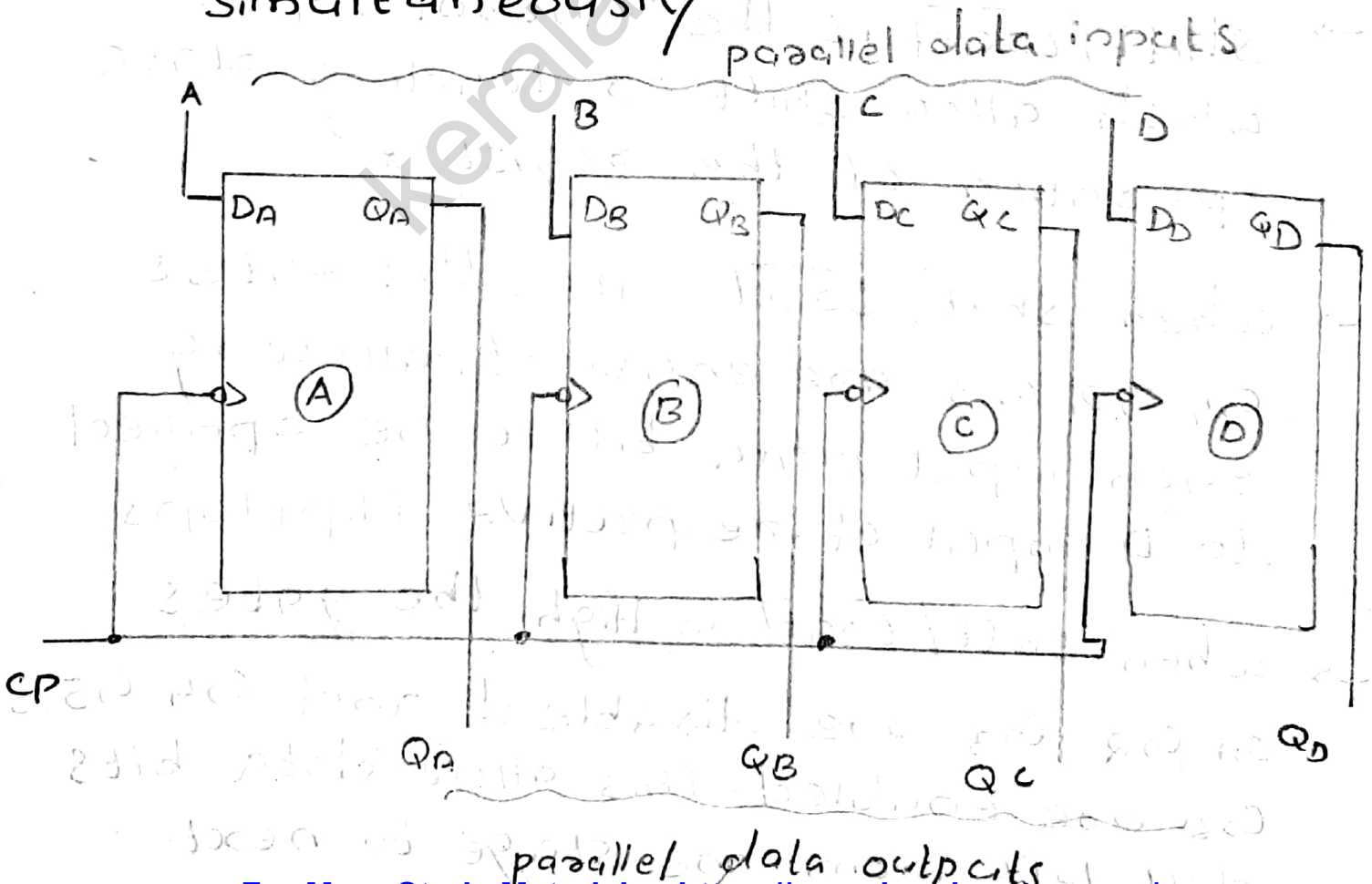
→ when shift / $\overline{Load}$ is High the gates G_1 G_2, G_3 are disabled and G_4 G_5 & G_6 are enabled. This allow data bits shift left from one stage to next.

→ The OR Gates at D input flipflop allow either the parallel data entry operation or shift operation, Depending on which AND Gate are enabled by the level shift / $\overline{load}$ input

PIPO Register

{No shift register because it is a storage register where others are shift register}

- In PIPO, there is a simultaneous entry of all data bits and the bits appears on parallel output simultaneously



imp

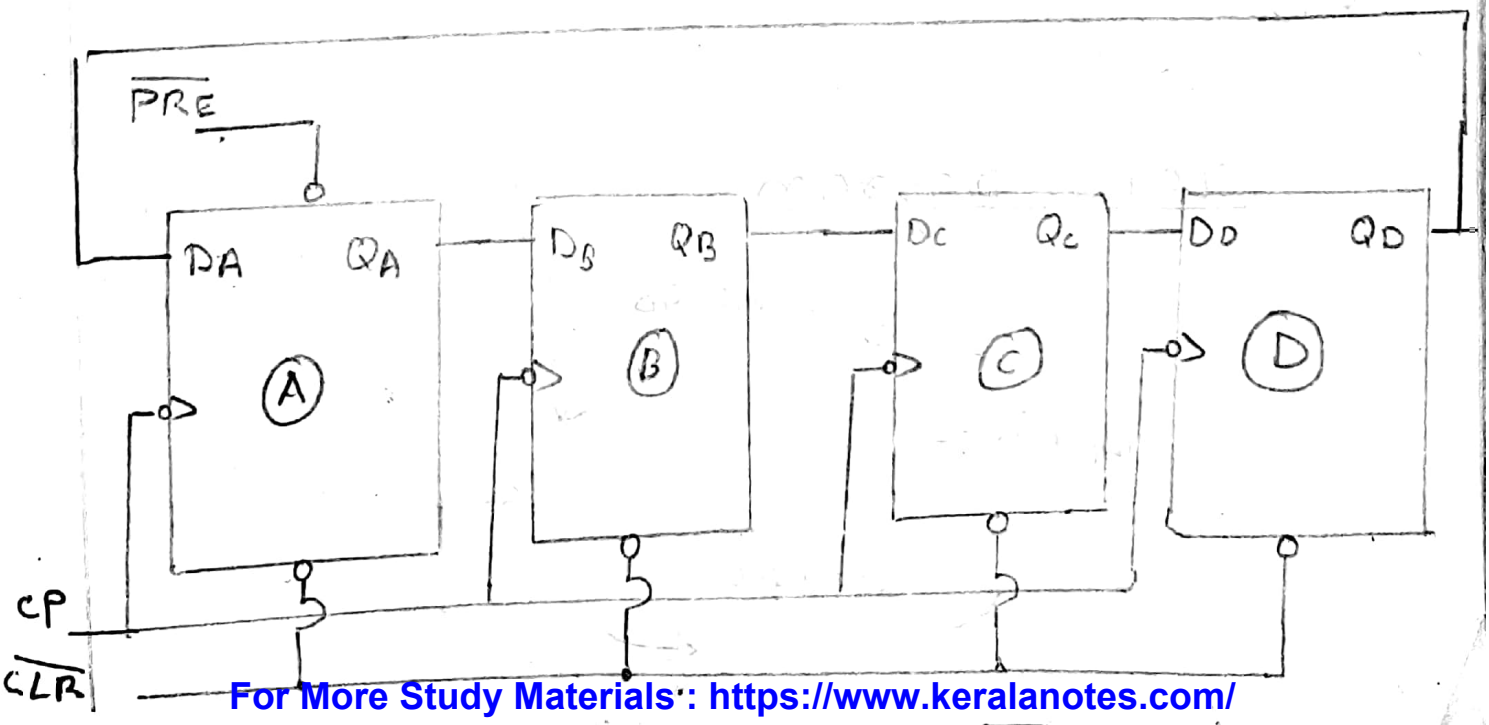
Ring Counter

→ Ring counter is a sequential logic circuit that is constructed using shift register

→ Ring counter is cascaded connection of flipflops, in which o/p of last flipflop is connected to i/p of first flipflop.

→ The o/p of first flipflop is connected to i/p of 2nd flipflop and so on.

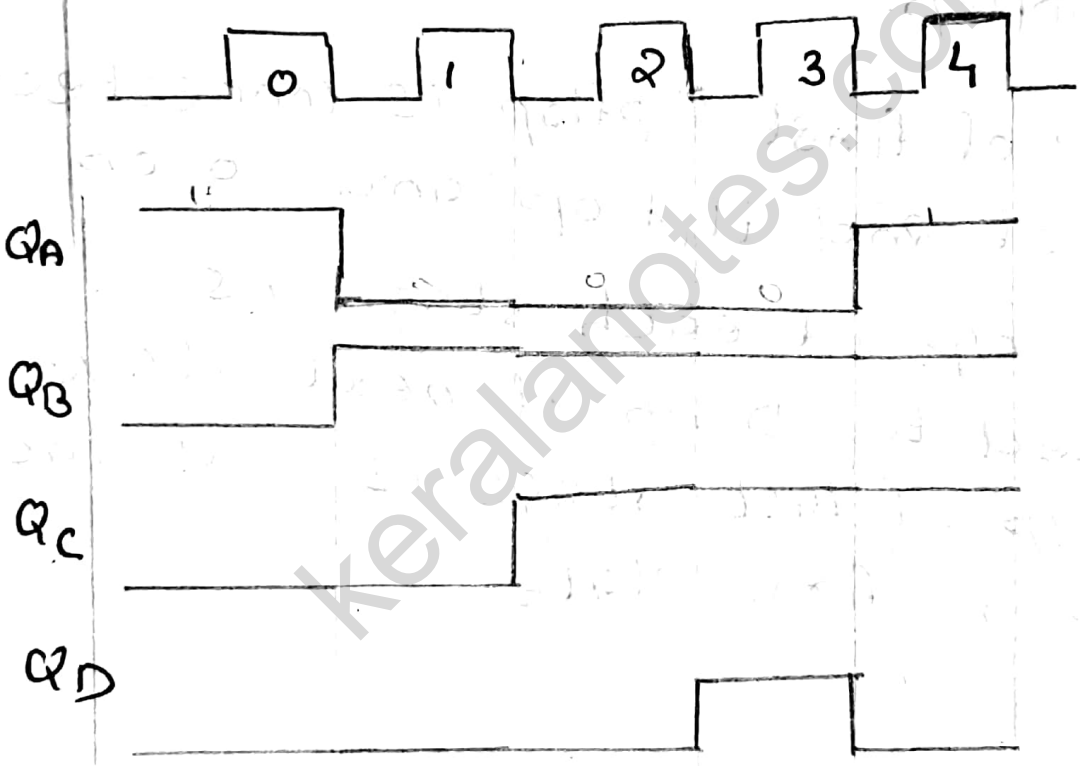
→ i.e. Q o/p of each stage is connected to D i/p of next stage and o/p of last stage is fed back to i/p of first state



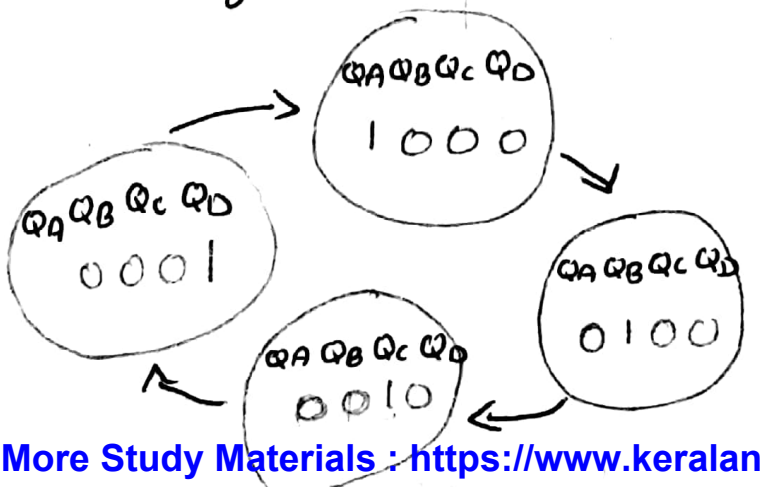
Truth table

Clock Pulse	Q_A	Q_B	Q_C	Q_D
0	1	0	0	0
1	0	1	0	0
			0	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0

Timing Diagram



State Diagram



→ $\overline{CLR}$ mark followed by $\overline{PRE}$ M... the o/p of first stage to '1' and remains o/ps are zero

→ $Q_A = 1, Q_B = Q_C = Q_D = 0$.

1 0 0 0

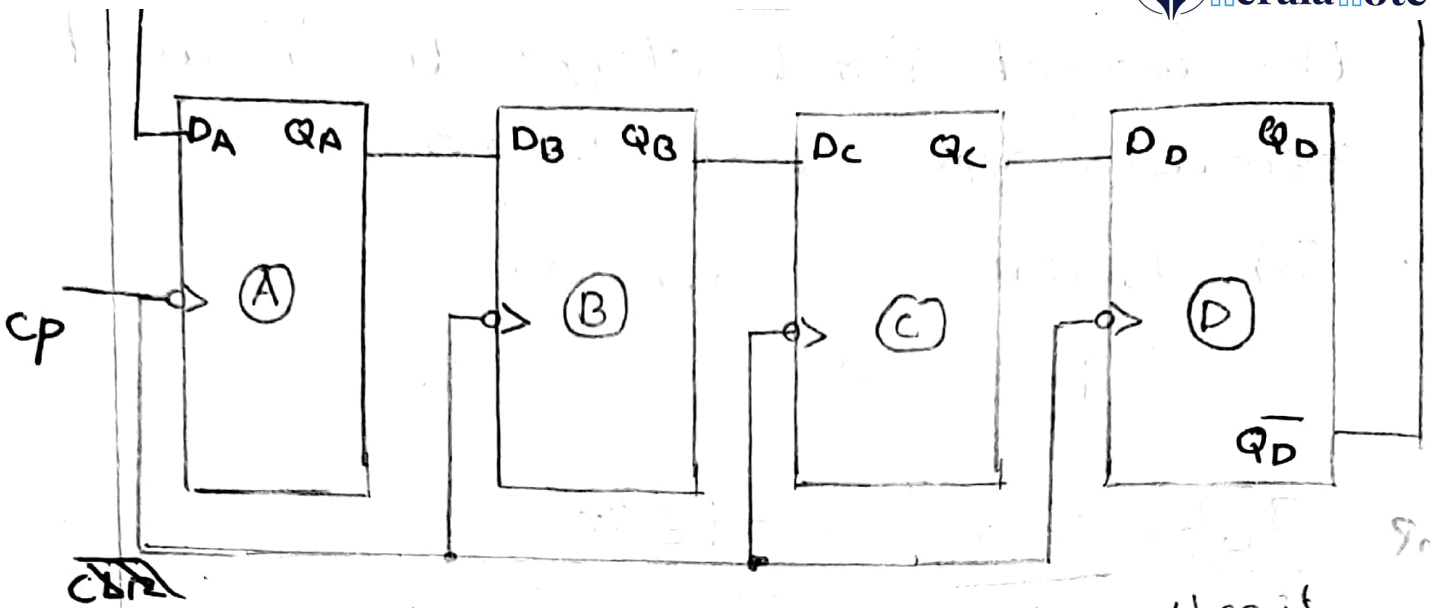
Imp

Johnson counter

→ Johnson counter / Twisted Ring counter is a sequential logic circuit that is constructed using shift register

→ In a Johnson counter Q o/p of each stage of flipflop is connected to the D i/p of next stage

The single exception is the complement o/p of last flipflop is connected back to D-i/p of first flipflop



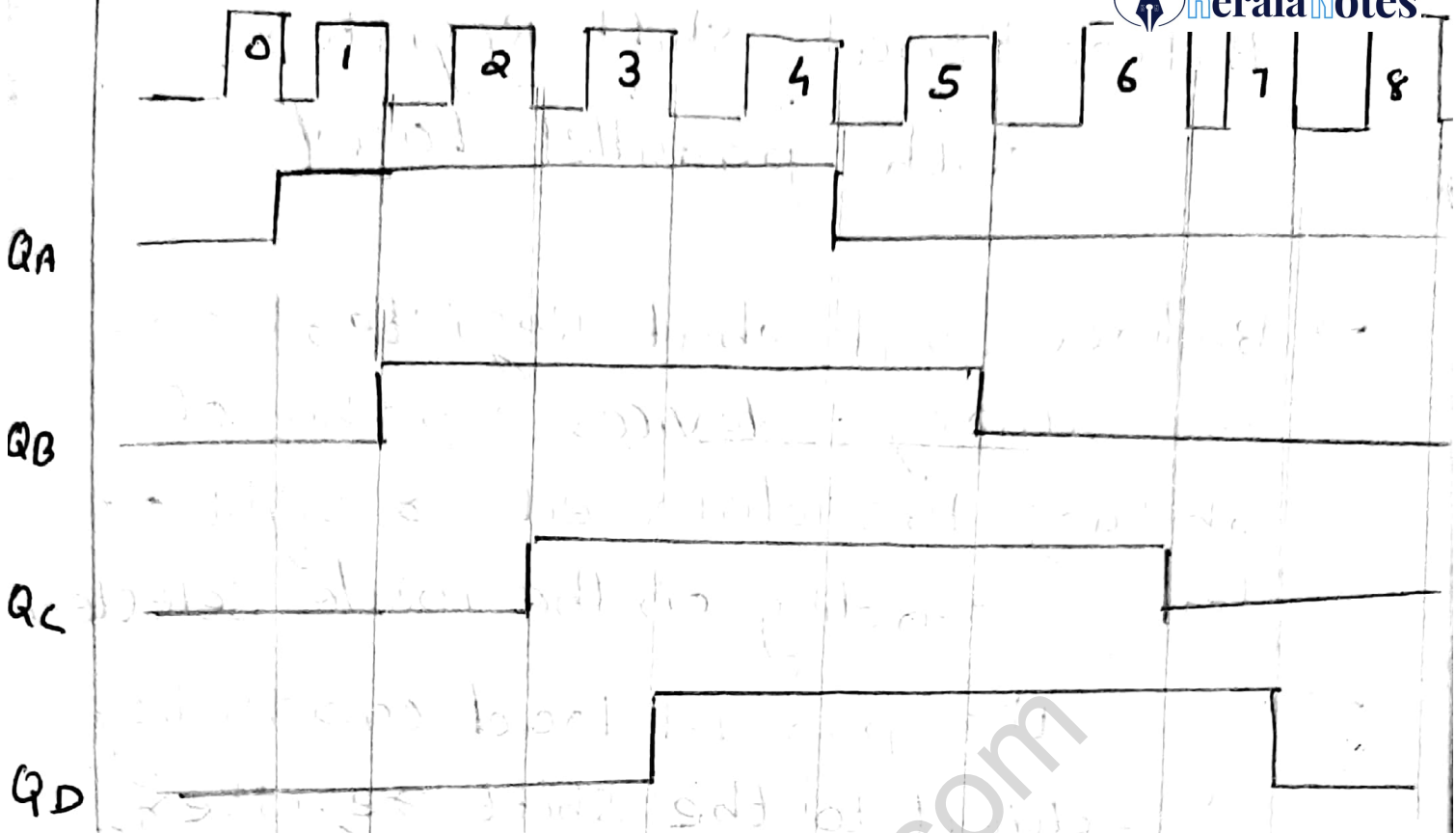
Truth table

if pse is there then it clock pulse = 1

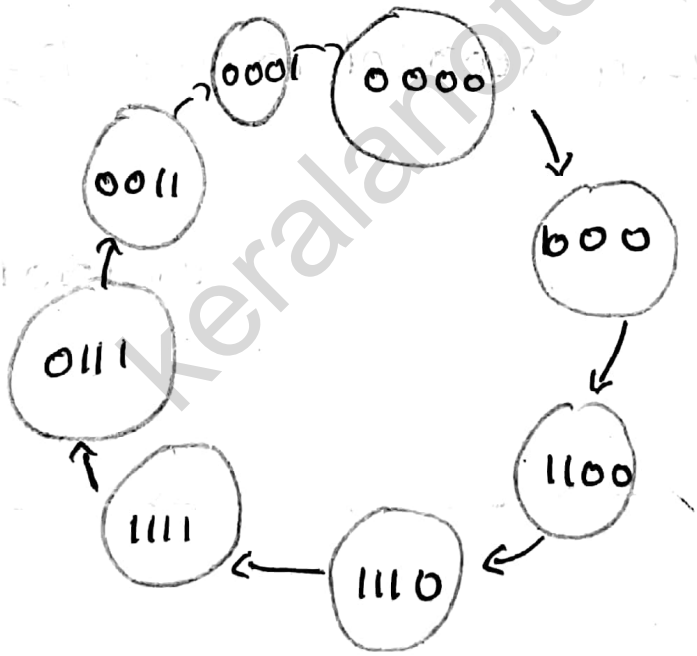
clock pulse	Q_A	Q_B	Q_C	Q_D
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
8	0	0	0	0

complement to Q_A

Timing Diagram



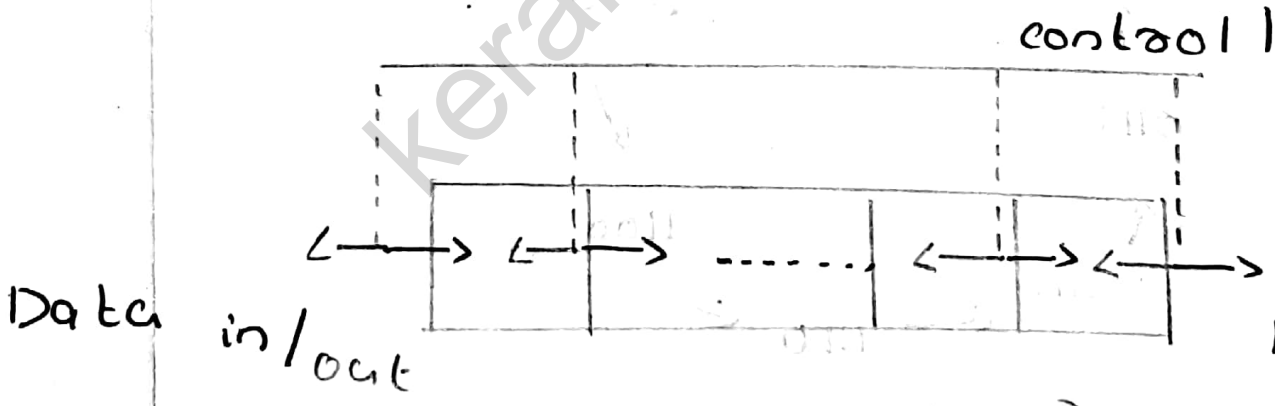
State Diagram



Bidirectional shift Registers with parallel load

→ Bidirectional shift Registers are the storage devices capable of shifting the data either right or left, depending on the mode selected.

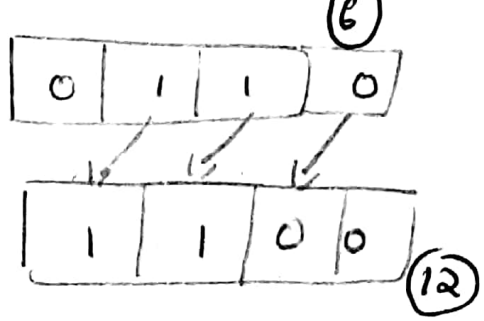
→ Here the parallel load capability is added to the shift register, the data entered in parallel can be taken out serial fashion by shifting data stored in the Register



code of 6 (0110)

Shift left

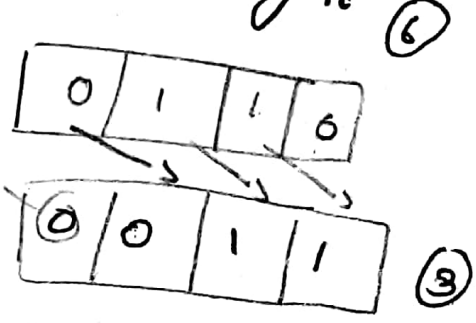
multiplied by 2



divide by 2

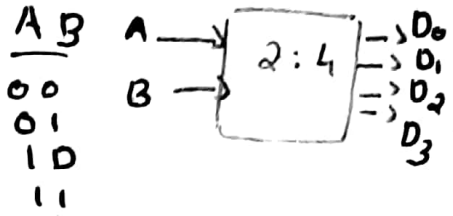
Shift right

zero will be in Vacancy space



multiplied by 2 Divid

Bidirectional Shift Register with parallel load



→ AS shown in figure D input of each flipflop has 3 sources:

(i) output of left adjacent flipflop

(ii) output of right adjacent flipflop

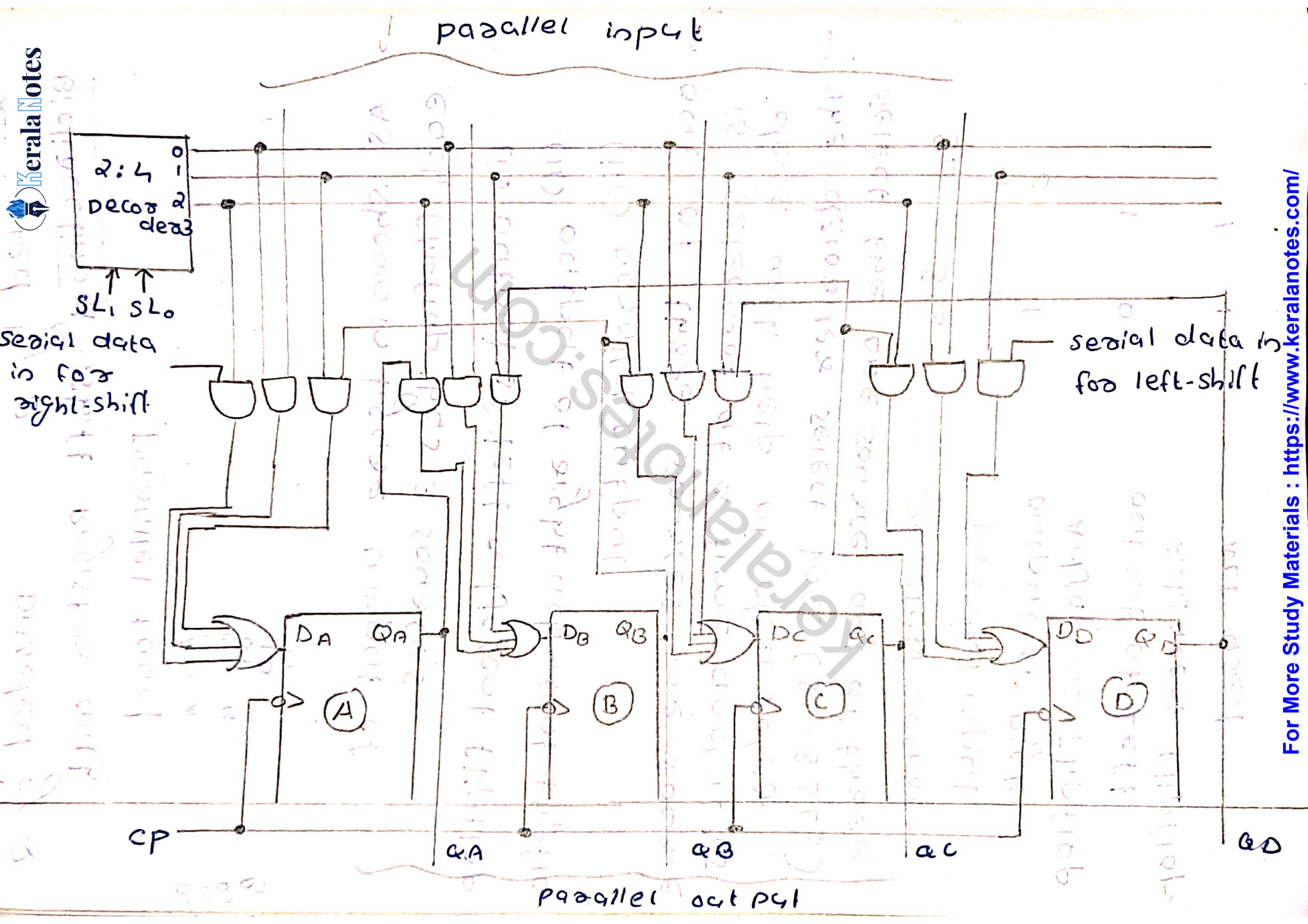
(iii) parallel output

→ out of three sources, one source is selected at a time and it is done with help of decoder

→ The decoder select lines (SL_1 & SL_2) select the one source out of three

SL_1	SL_0	selected source
0	0	parallel input
0	1	output of Right Adjacent Flipflop
1	0	output of Left Adjacent flipflop
1	1	Not used

P.T.O

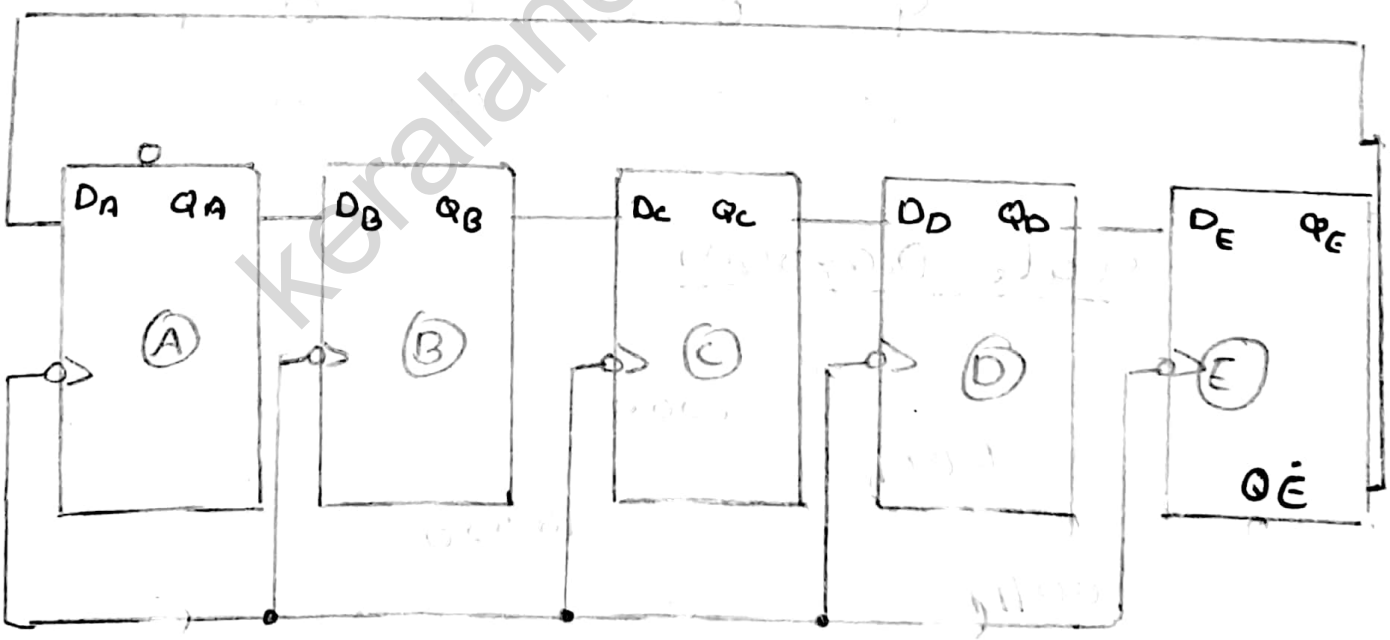


→ when selected line are 0, [illegible] from the parallel input is loaded into 4 bit Register

→ when select lines are 01, Data from within the register is shifted 1 bit left when

→ when select lines are 10, Data within the register is shifted 1 bit Right

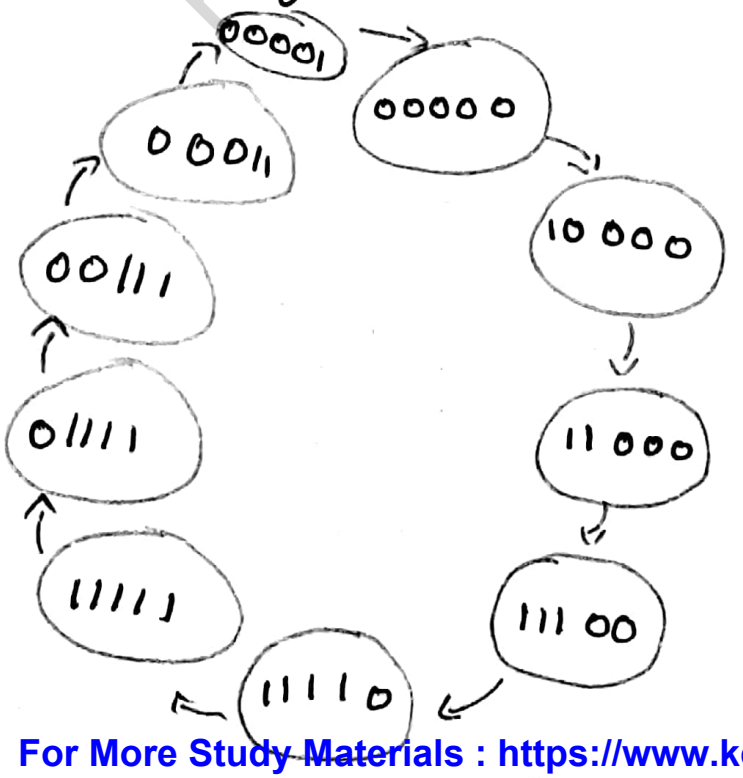
Q1. Design 5 Bit Johnson Counter



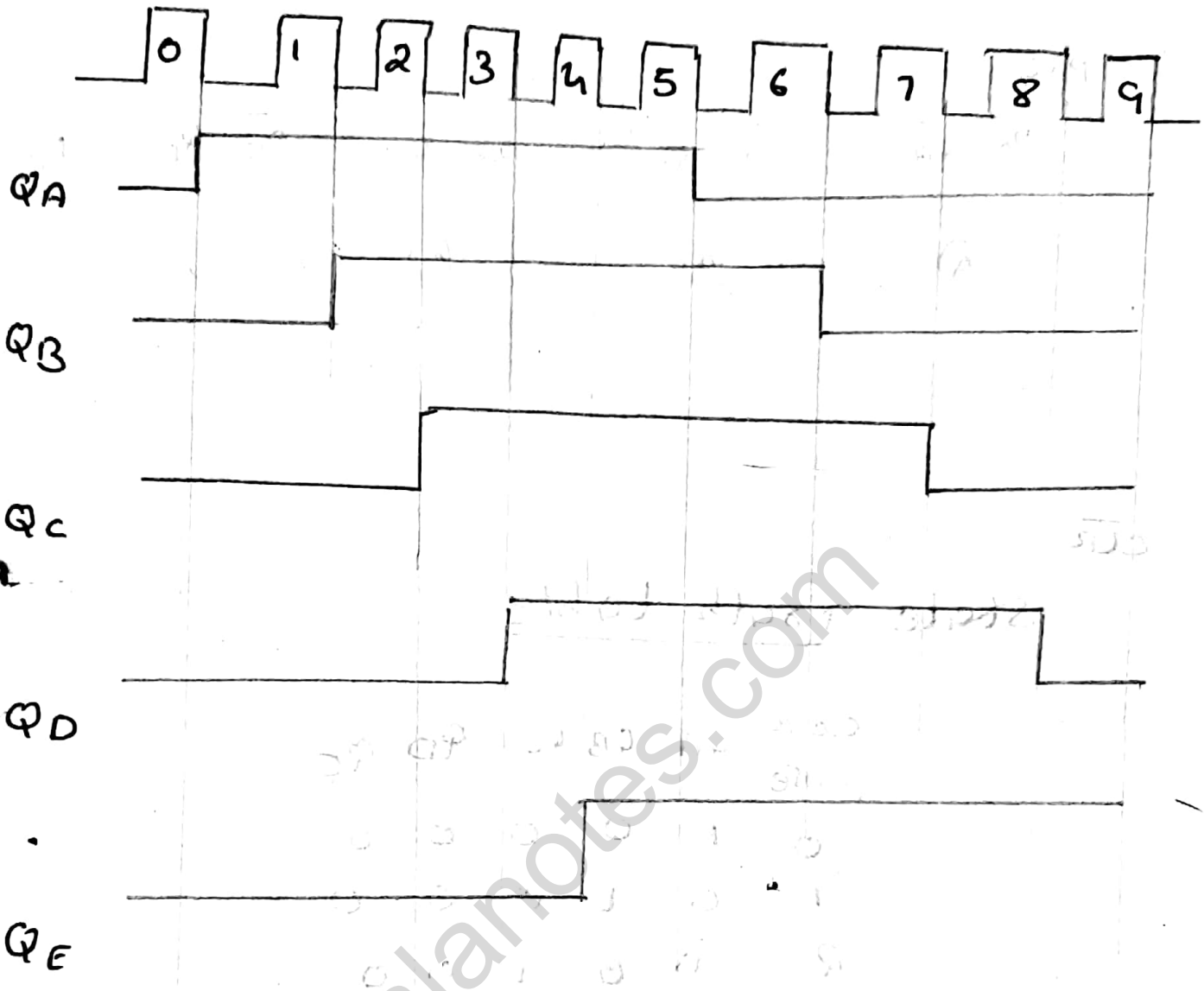
Truth table

clock pulse	Q_A	Q_B	Q_C	Q_D	Q_E
0	0	0	0	0	0
1	1	0	0	0	0
2	1	1	0	0	0
3	1	1	1	0	0
4	1	1	1	1	0
5	1	1	1	1	1
6	0	1	1	1	1
7	0	0	1	1	1
8	0	0	0	1	1
9	0	0	0	0	1
10	0	0	0	0	0

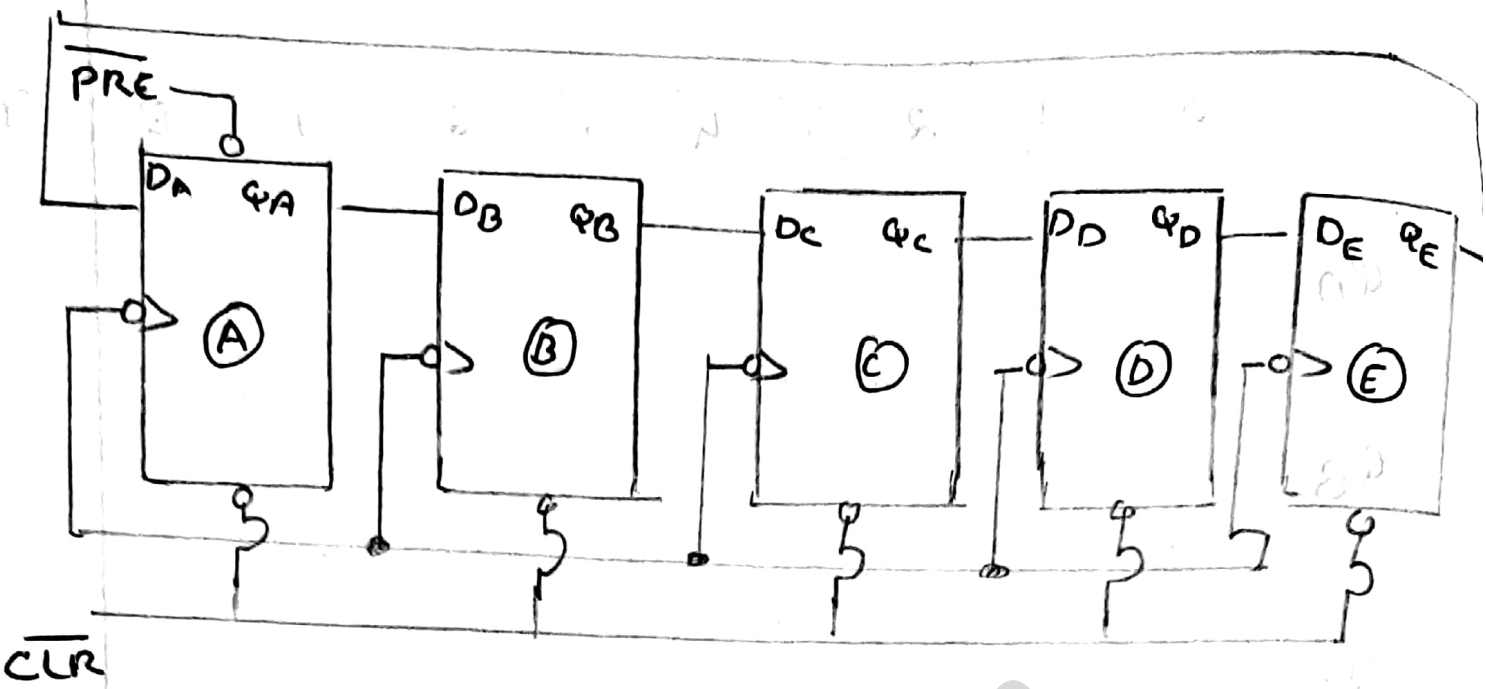
State Diagram



Timing Diagram



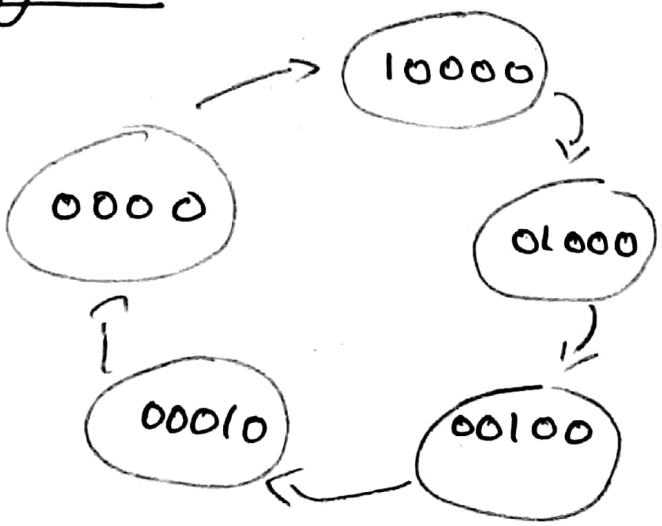
Q. Design 5 Bit Ring counter



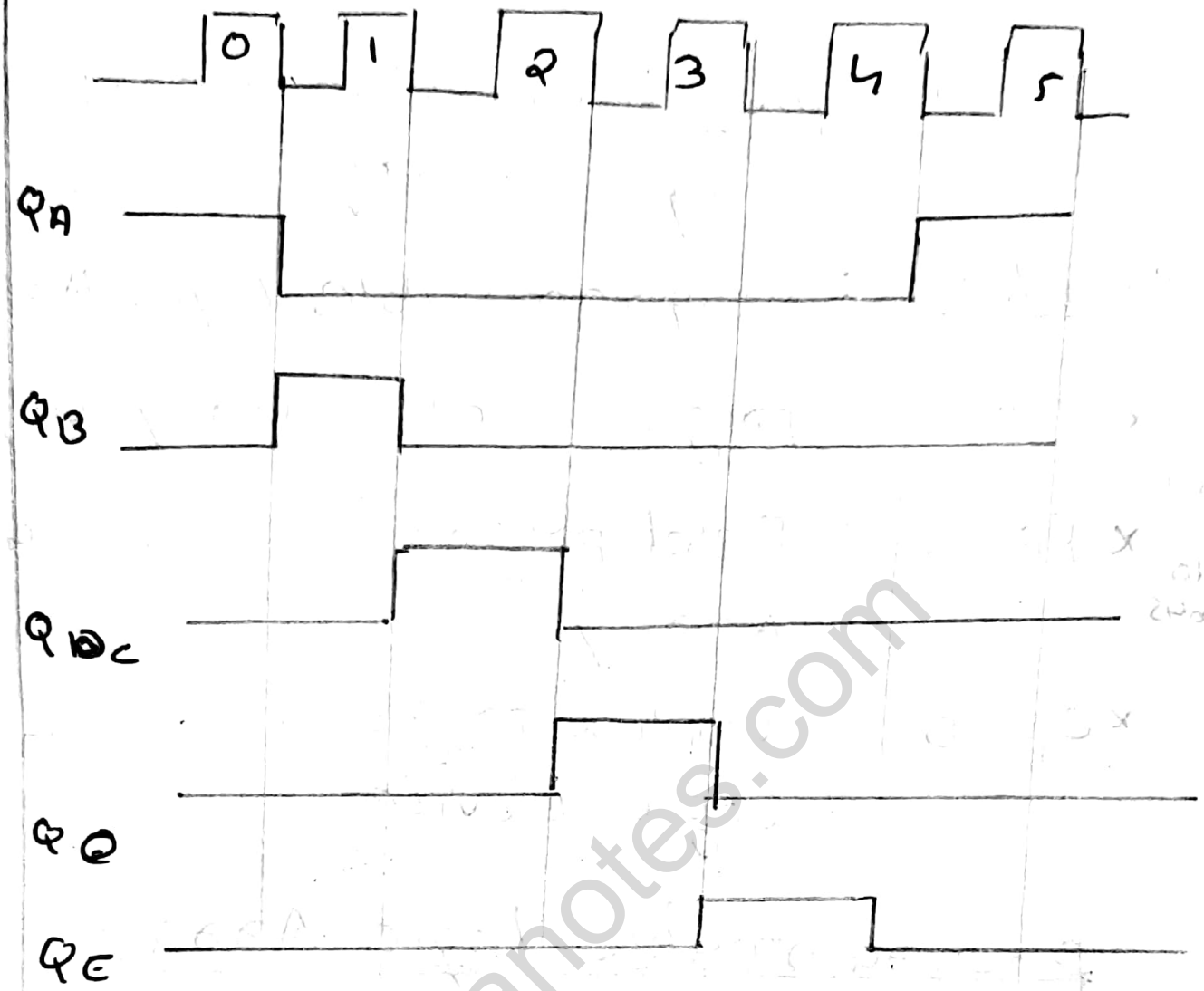
State truth table

clock pulse	Q_A	Q_B	Q_C	Q_D	Q_E
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	0
4	0	0	0	0	1
5	1	0	0	0	0

State Diagram



timing diagram



Programmable Logic Devices (PLD)

- Programmable Logic Devices are used to implement logic functions
- The main Advantage of programmable logic Devices approach is that PLD can be easily configurable by individual user for specific application

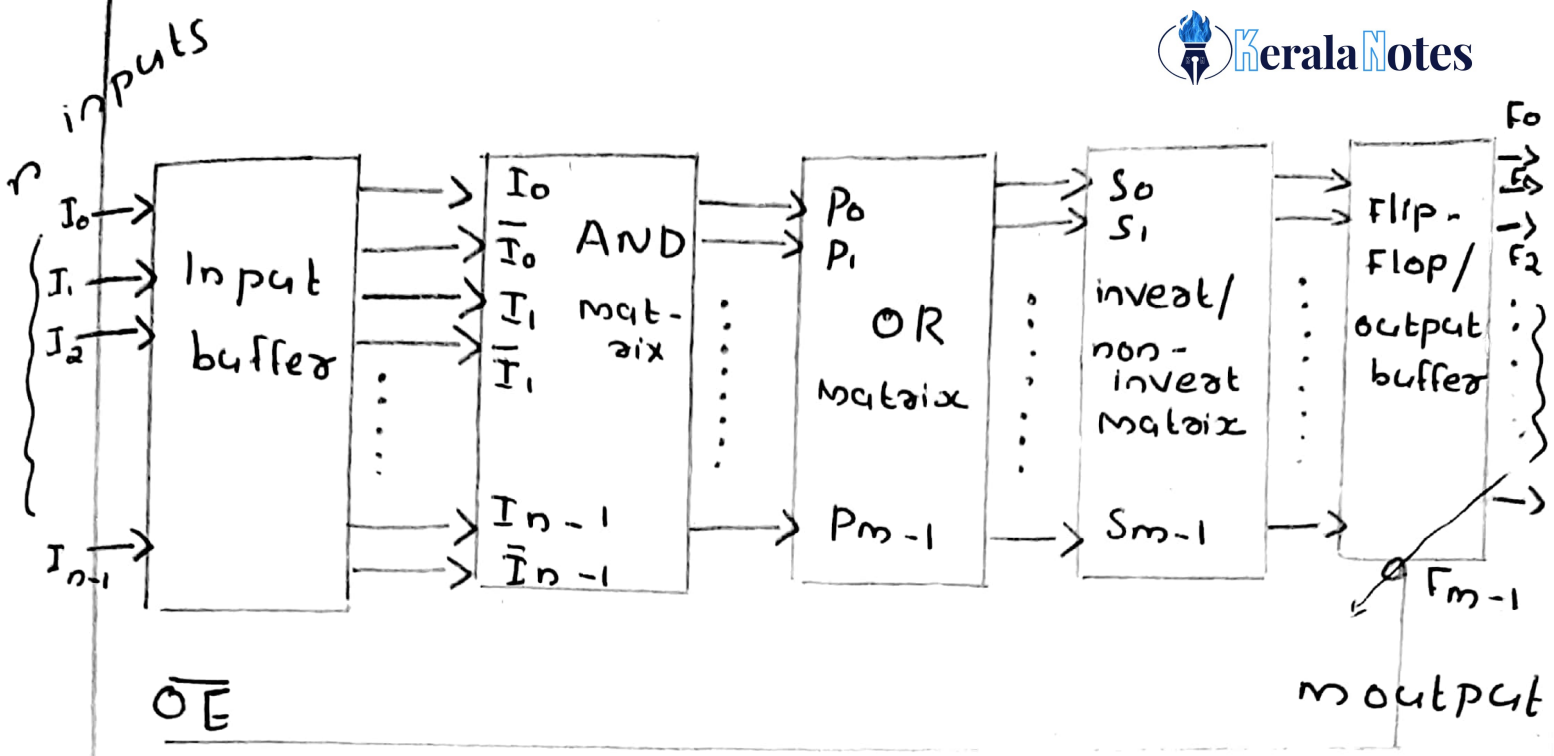
PLD's are classified as:

- PROM : programmable read only memory
- IMP → PLA : programmable logic Array
- PAL : programmable Array logic
- Not in syllabus FPGA : field programmable Gate Array
- CPLD : Complex programmable logic devices

Programmable logic Array

→ PLA is a programmable logic device that has both programmable AND array & programmable OR array. Hence, it is the most flexible PLD

→ It consist of n inputs, output buffer, wit m outputs, m product terms, m sum terms, input and output buffers



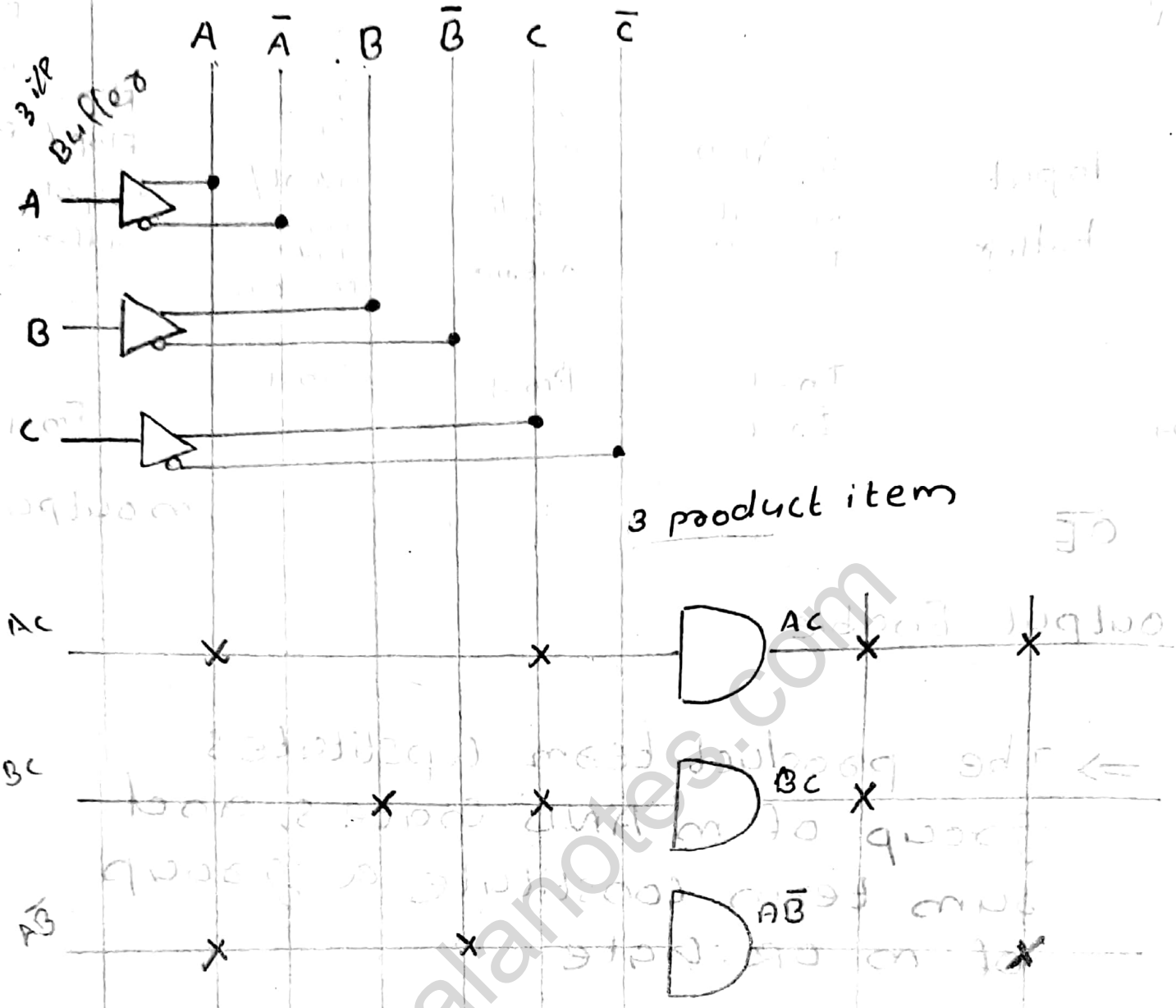
Output Enable.

⇒ The product team constitutes group of m AND Gates, and sum team constitute a group of m OR Gate

⇒ fuses are inserted between all n inputs and their output complement values of each of the AND Gate

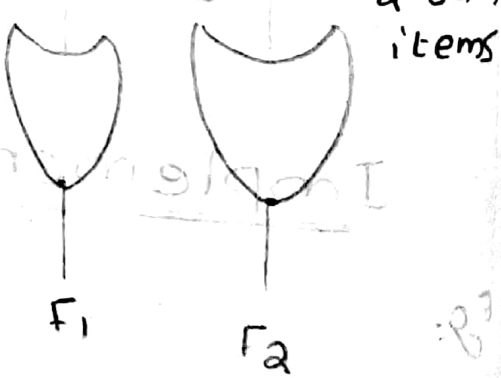
Implementation Examples of PLA

Eg:

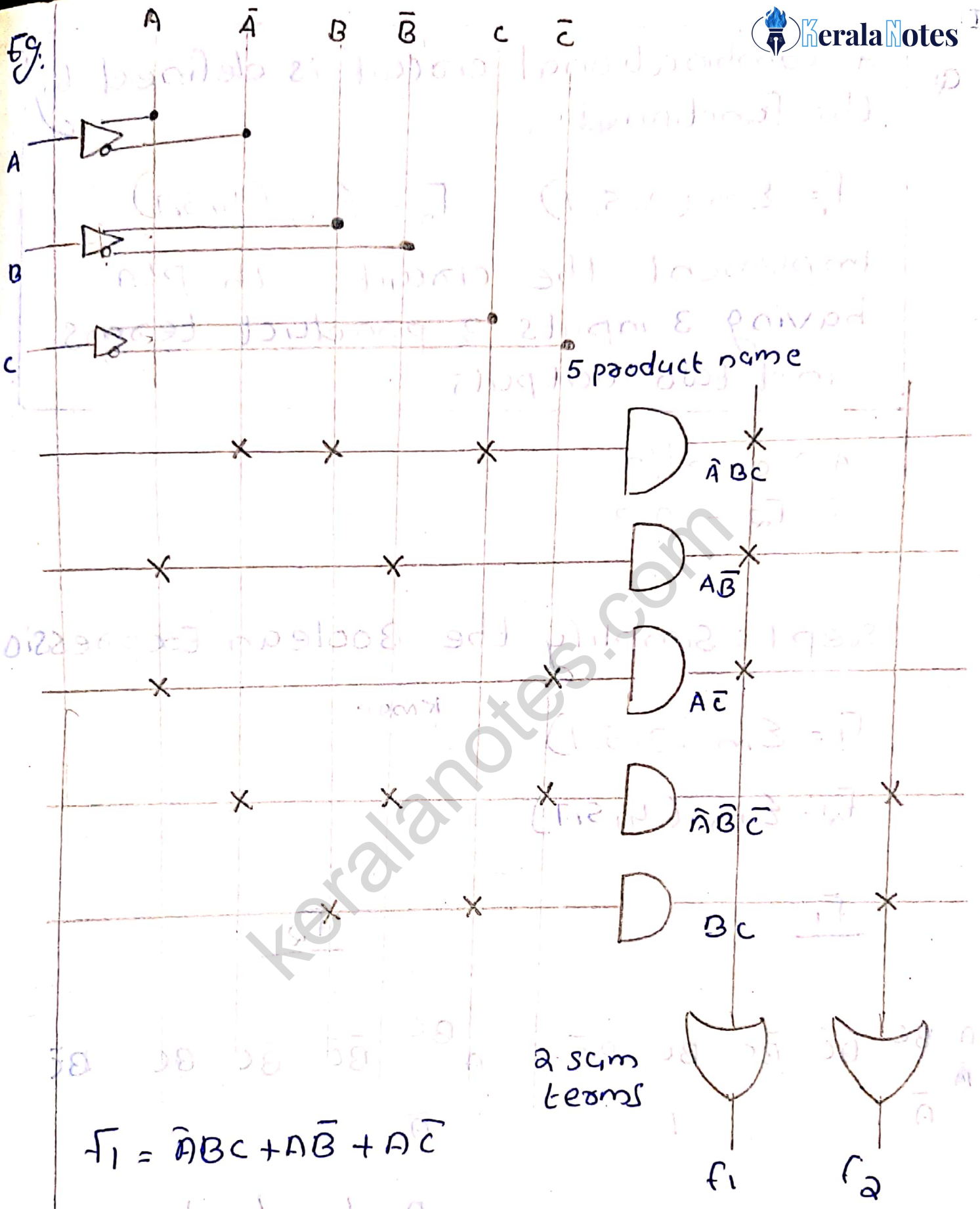


$F_1 = AC + BC$

$F_2 = AC + A\bar{B}$



Eg:



$$f_1 = \bar{A}BC + AB̄ + AC̄$$

$$F_2 = \bar{A}B̄C̄ + BC$$

Problems

IMP Q

A combinational circuit is defined by the functions:

$F_1 = \Sigma m(3,5,7)$ $F_2 = \Sigma m(4,5,7)$

Implement the circuit with PLA having 3 inputs 3 product terms and two outputs

A B C → i/p

F_1 F_2 → o/p

Step 1 : Simplify the Boolean Expression

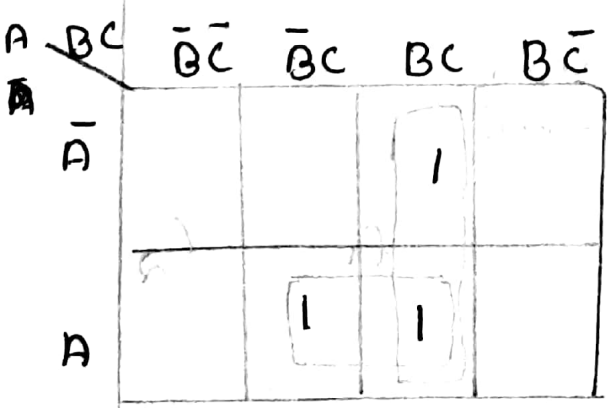
$F_1 = \Sigma m(3,5,7)$

$F_2 = \Sigma m(4,5,7)$

KMAP

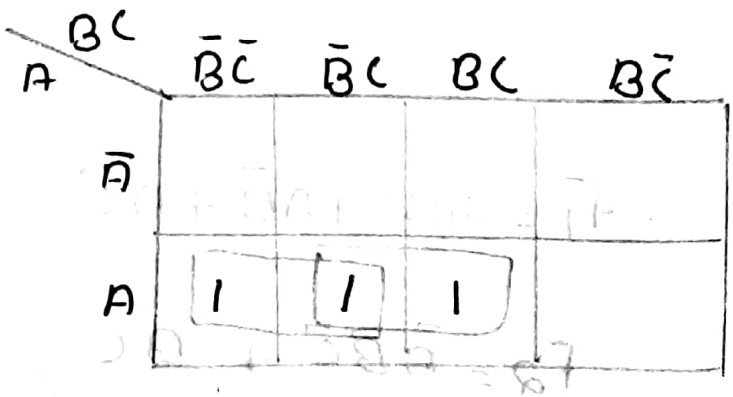
0	1	3	2
4	5	7	6

F_1



$F_1 = AC + BC$

F_2



$F_2 = A\bar{B} + AC$

Step 2: PLA program Table

$F_2 = A\bar{B} + AC$

We need only 3

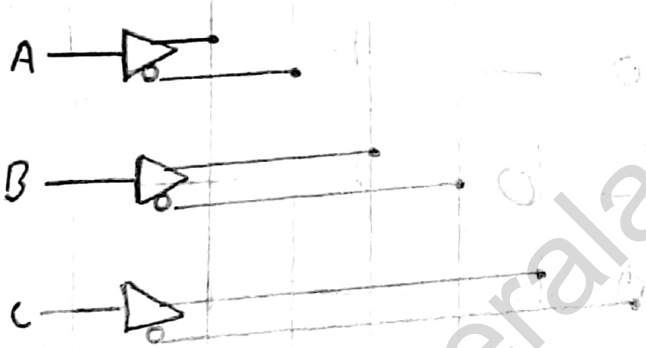
Product term	Inputs			outputs	
	A	B	C	F_1	F_2
1. AC	1	—	1	1	1
2. BC	—	1	1	1	—
3. $A\bar{B}$	1	0 since $\bar{B}$	—	—	1

2 question given

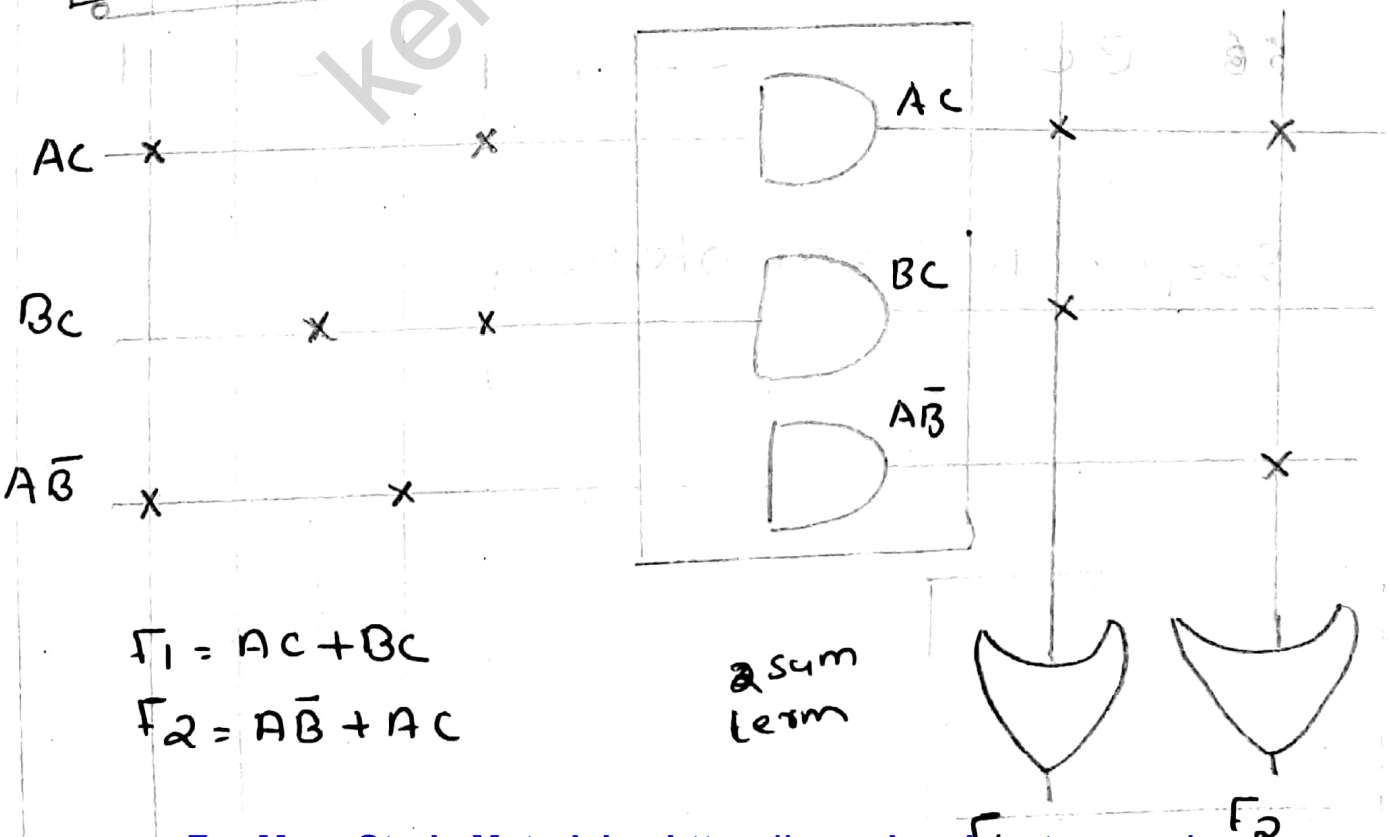
Step 3: Implementation

i/p Buffer
AND = 3
OR = 2

A $\bar{A}$ B $\bar{B}$ C $\bar{C}$



3 product term



$F_1 = AC + BC$

$F_2 = A\bar{B} + AC$

Q2 Draw a PLA circuit the implement the logic functions

$F_1 = \bar{A}BC + A\bar{B} + A\bar{C}$

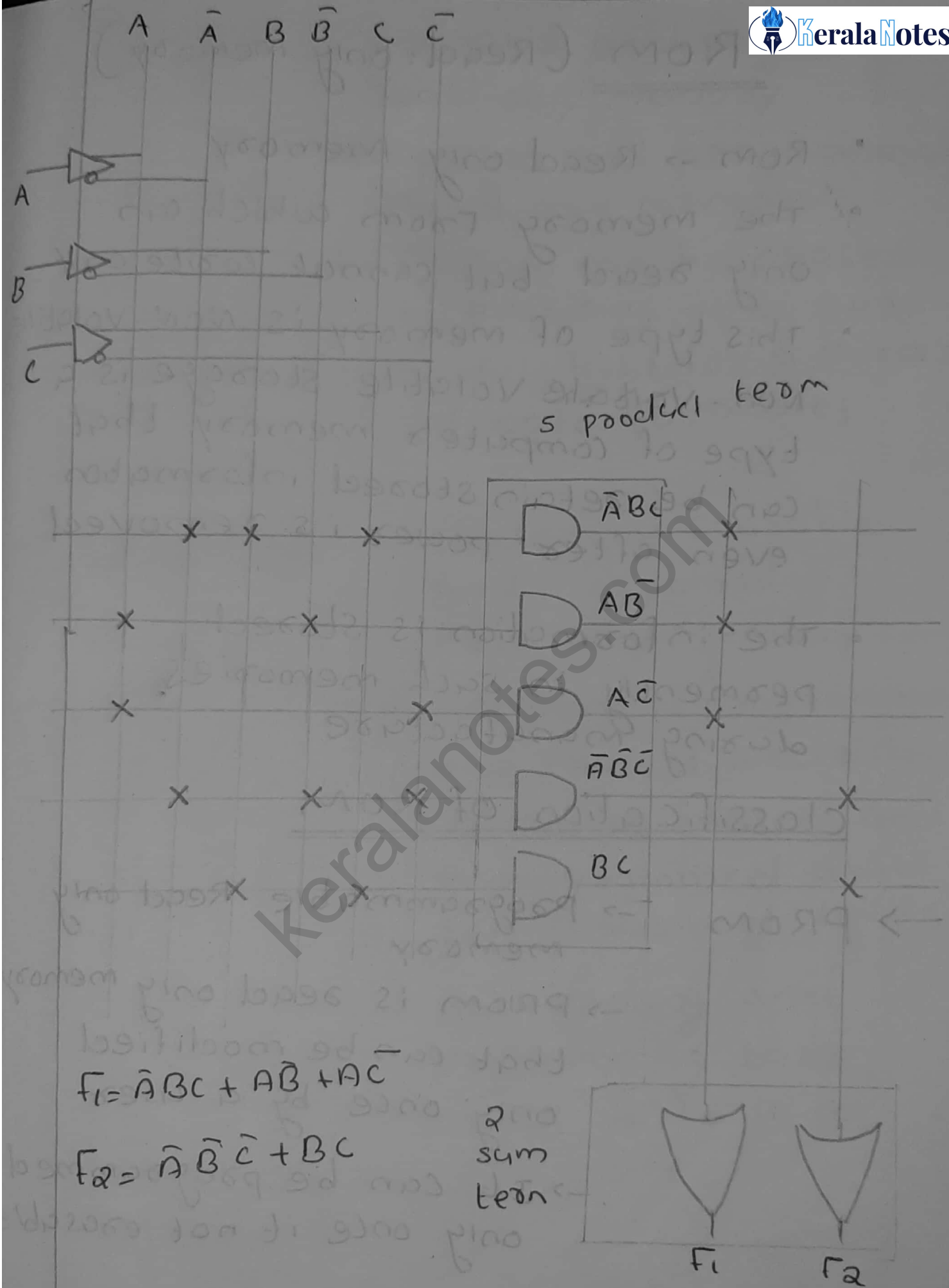
$F_2 = \bar{A}\bar{B}\bar{C} + BC$

ans: Step 1: Simplify the boolean Expression

Step 2: PLA program table

product term	Inputs			outputs	
	A	B	C	F_1	F_2
1. $\bar{A}BC$	0	1	1	1	—
2. $A\bar{B}$	1	0	—	1	—
3. $A\bar{C}$	1	—	0	1	—
4 $\bar{A}\bar{B}\bar{C}$	0	0	0	—	1
5 6. BC	—	1	1	—	1

Step 3: implementation



$F_1 = \bar{A}BC + A\bar{B} + A\bar{C}$

$F_2 = \bar{A}\bar{B}\bar{C} + BC$

ROM (Read only memory)

- ROM -> Read only memory
- The memory from which can only read but cannot write on it
- This type of memory is NON-Volatile
  Non-~~Volatile~~ Volatile storage is a type of computer memory that can be retain stored information even after power is removed
- The information is stored permently in such memories during manufacture

Classification of ROM

→ **PROM** -> Programmable Read only memory

-> PROM is read only memory that can be modified only once by a user

-> It can be programmed only once it not erasable

→ EPROM → Erasable and programmable Read only Memory

→ EPROM and programmable Read only memory

→ EPROM can be erased by exposing it to ultra-violet light for a duration of upto 40 minutes

→ EEPROM → Electrically Erasable and programming Read only memory

→ EEPROM is programmed and erased electrically

→ It can be erased and reprogrammed about ten thousand times

→ Both erasing and programming take about 4 to 10 ms

PROM [programmable read only memory]

disadvantage: erasing not possible

→ Read only memory Rom is a memory device, which store the binary information permanently

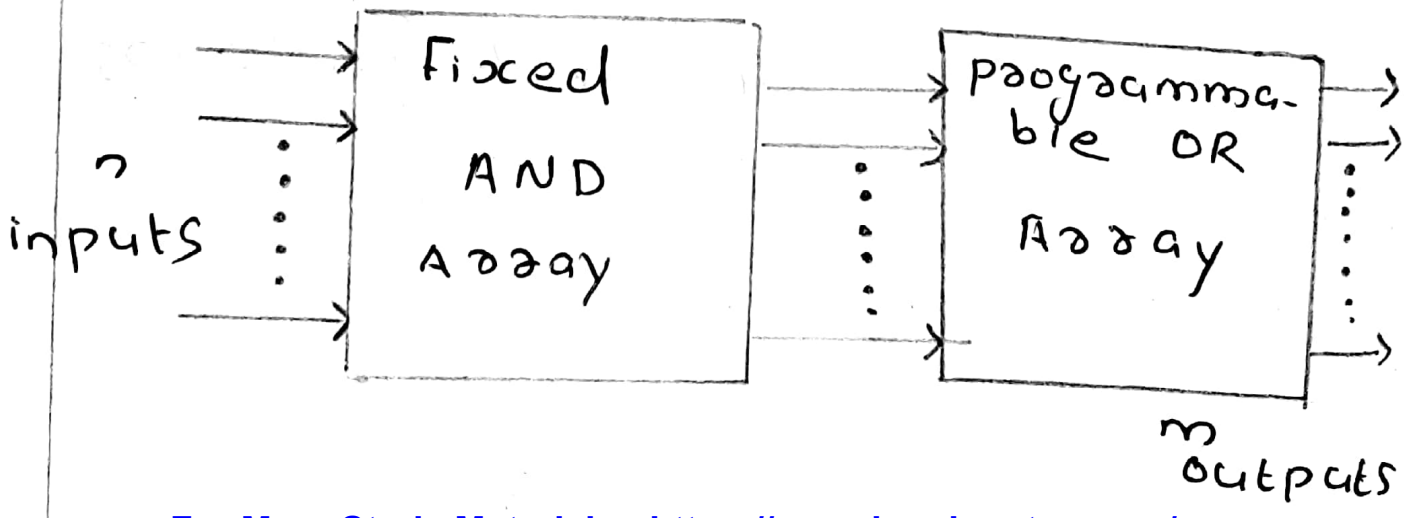
→ That means, we can't change that stored information by any means later.

→ If the Rom has programmable feature then it is called as programmable Rom PROM

→ The user have flexibility to program the binary information electrically once by using PROM programmer

→ PROM has fixed AND & OR Array

Block Diagram



Example

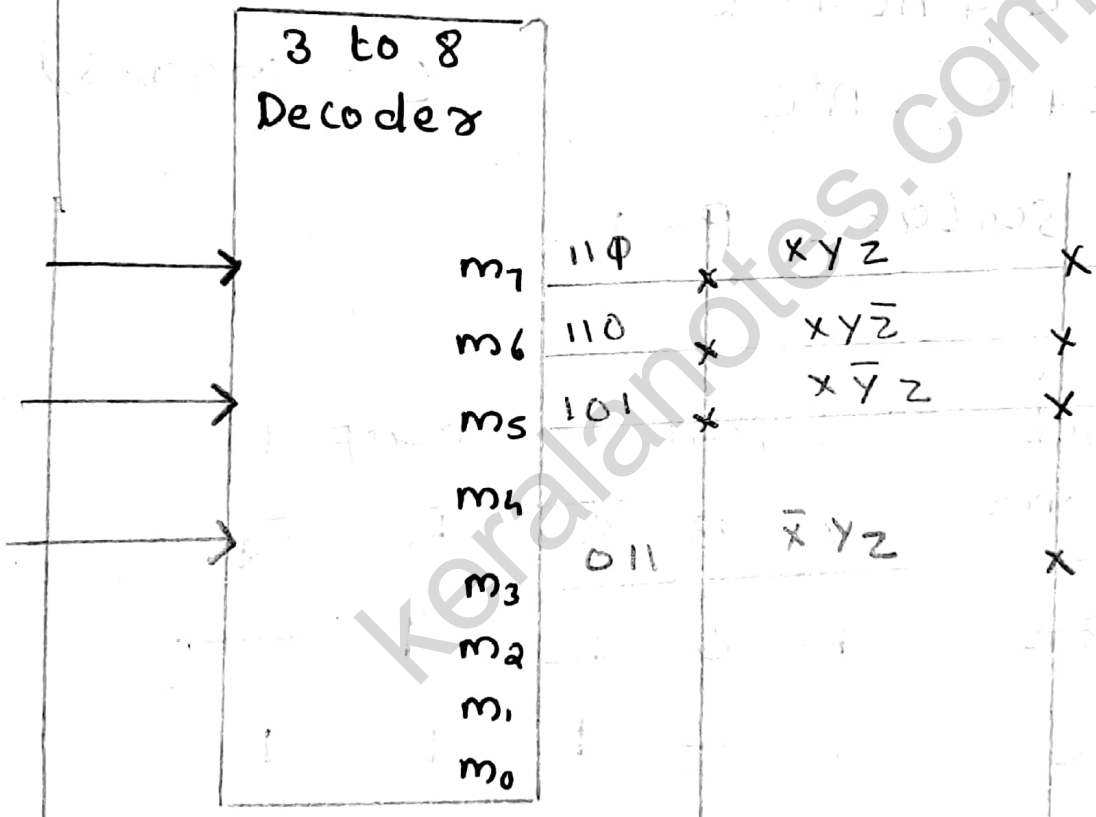
1. Implement the Boolean function

- $A(x, y, z) = \Sigma m(5, 6, 7)$

  $B(x, y, z) = \Sigma m(3, 5, 6, 7)$

Ans: It is a 3 variable (x, y, z)

∴ we need a 3 to 8 decoder



- $A(x, y, z) = \Sigma m(5, 6, 7)$

- $B(x, y, z) = \Sigma m(3, 5, 6, 7)$

PLA previous question papers

1. Display the working of PLA with a block diagram and a simple example (Sep 2020). (10 mark)

2. What is meant by PLA? Show the implementation of

$F_1 = AB'C + AC + BC$ (Dec 2019)

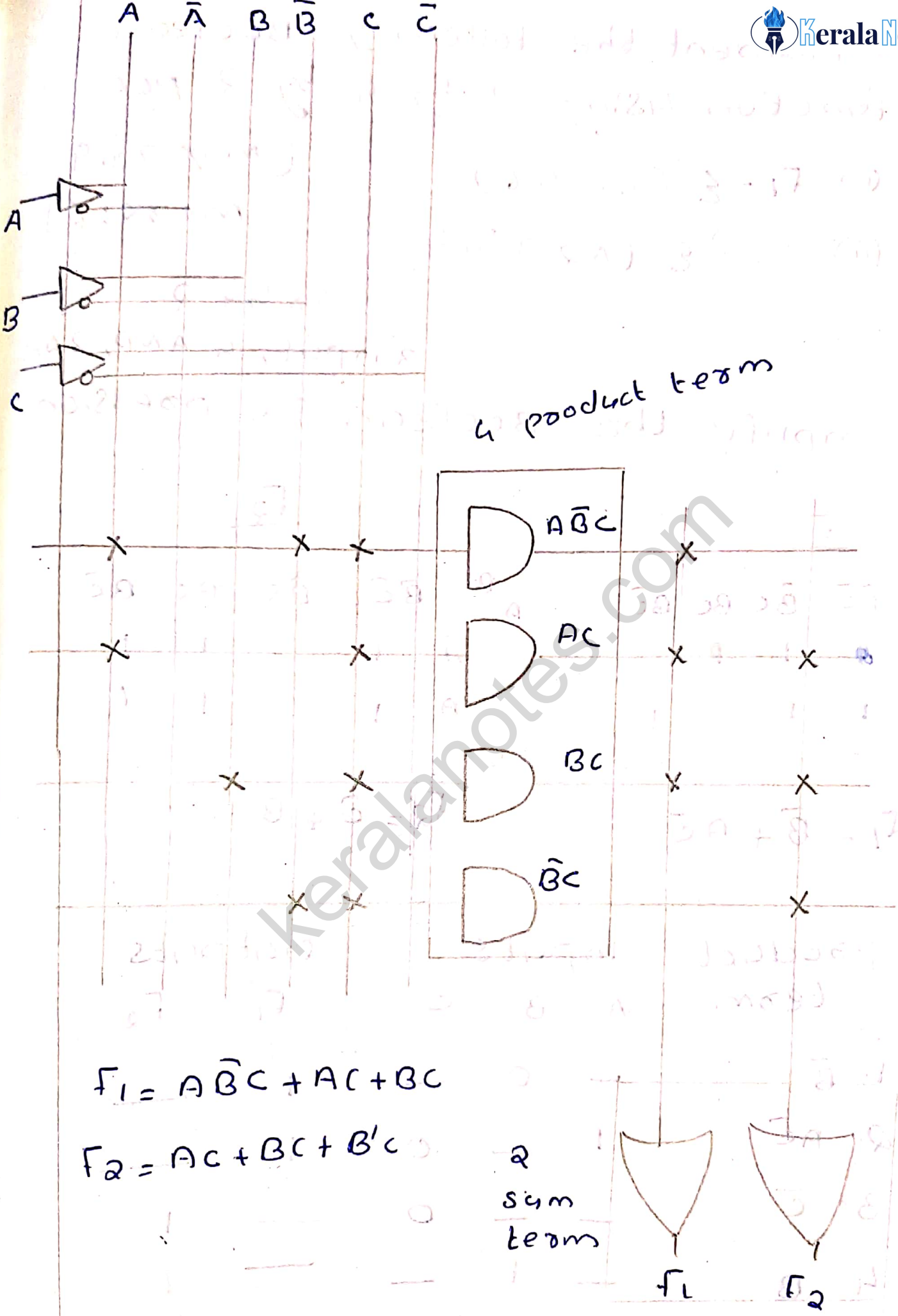
$F_2 = AC + BC + B'C$ (10 marks)

using suitable PLA

ans:

Product term	Inputs			Output	
	A	B	C	F_1	F_2
1. $AB'C$	1	0	1	1	—
2. AC	1	—	1	1	1
3. BC	—	1	1	1	1
4. $B'C$	—	0	1	—	1

Implementation



$F_1 = A\bar{B}C + AC + BC$

$F_2 = AC + BC + B'C$

3. Implement the following function using 3-by-4-by 2 PLA [may 2019 mark: 6]

(i) $F_1 = \Sigma(1, 4, 5, 6)$

(ii) $F_2 = \Sigma(0, 2, 3, 4, 6, 7)$

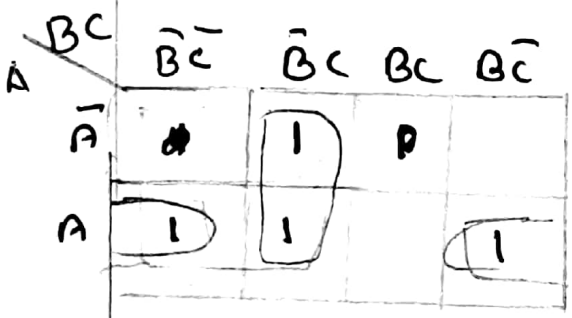
ans:

3 x 4 x 2

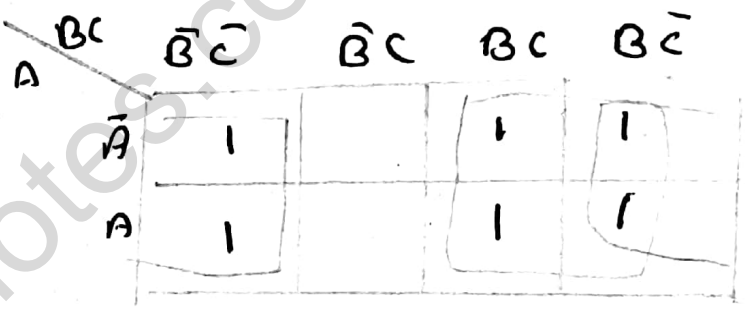
3 input, 4 AND, 2 OR

Simplify the Boolean Expression

F_1



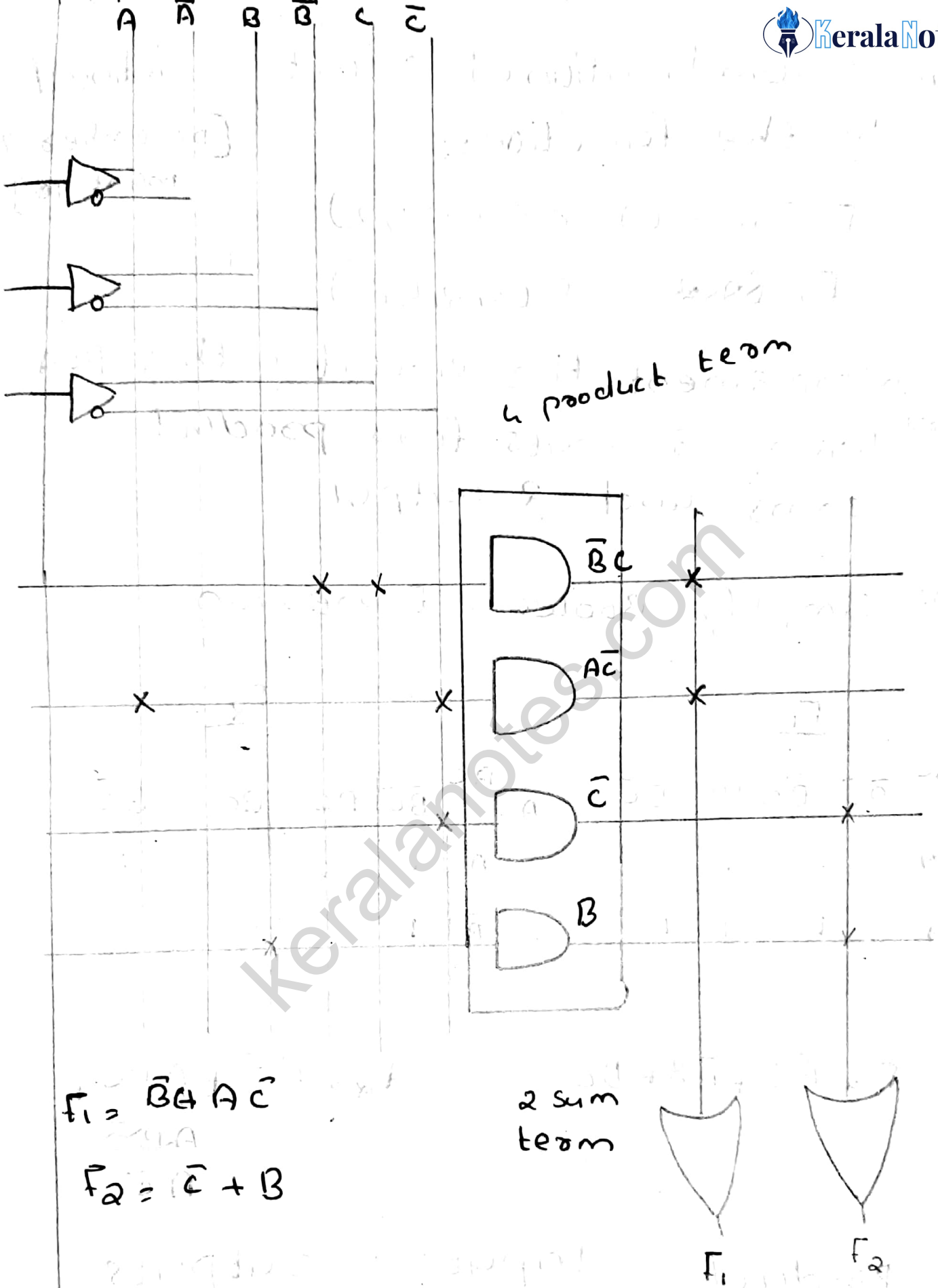
F_2



$F_1 = \bar{B} + A\bar{C}$

$F_2 = \bar{C} + B$

product term	inputs			outputs	
	A	B	C	F_1	F_2
1. $\bar{B}$	—	0	—	1	—
2. $A\bar{C}$	1	—	0	1	—
3. $\bar{C}$	—	—	0	—	1
4 B	—	1	—	—	1



$F_1 = \bar{B} + A\bar{C}$

$F_2 = \bar{C} + B$

4. A combinational circuit is defined by the functions: [december 2014 mark: 7]

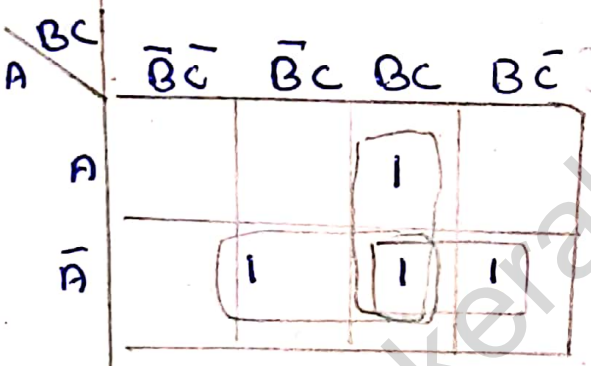
$f_1(A, B, C) = \Sigma(3, 5, 6, 7)$

$f_2(A, B, C) = \Sigma(0, 2, 4, 7)$

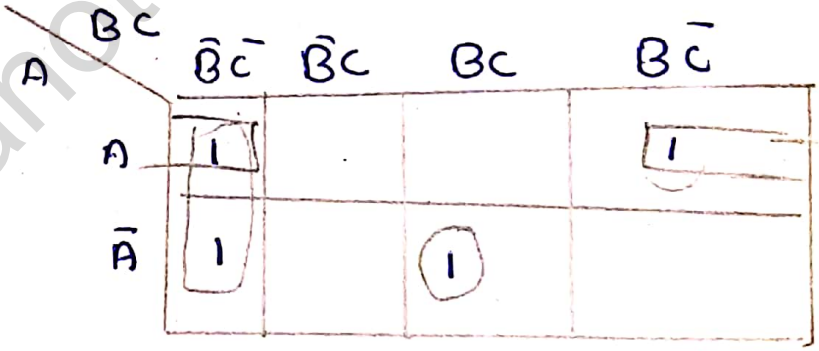
question wrong. Implement the circuit with a PLA having 3 inputs four product terms and 2 output

ans: Simplify Boolean Expression

f_1



f_2



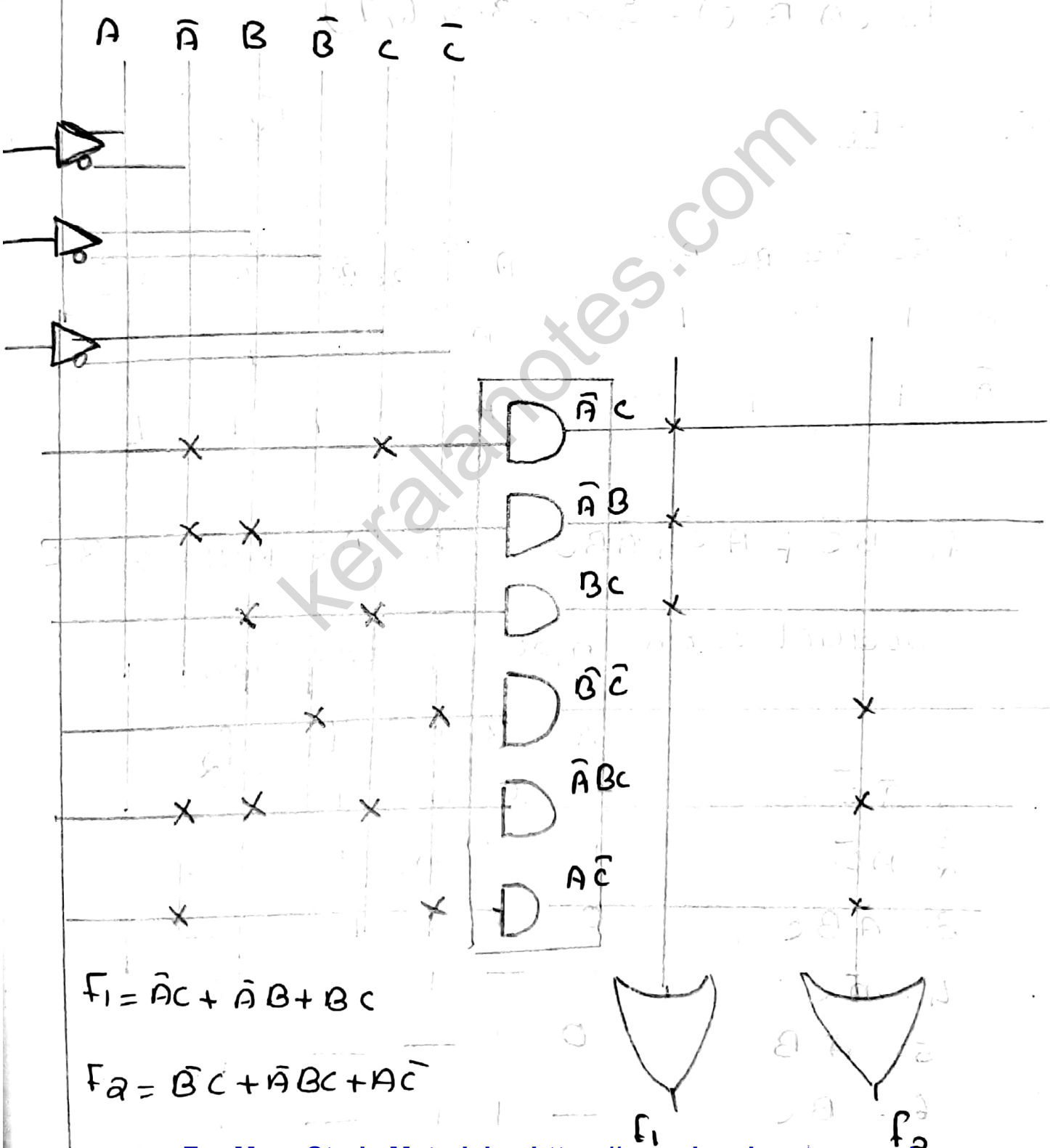
$f_1 = \bar{A}C + \bar{A}B + BC$

$f_2 = B\bar{C} + \bar{A}BC + A\bar{B}\bar{C}$

product term	Inputs			outputs	
	A	B	C	f_1	f_2
1. $\bar{A}C$	0	—	1	1	—

2. $\bar{A}B$
3. Bc
4. $B\bar{c}$
5. $\bar{A}Bc$
6. $A\bar{c}$

0	1	—	1	—
—	1	1	1	1
—	0	0	—	1
0	1	1	—	1
1	—	0	—	1



$f_1 = \bar{A}c + \bar{A}B + Bc$

$f_2 = B\bar{c} + \bar{A}Bc + A\bar{c}$

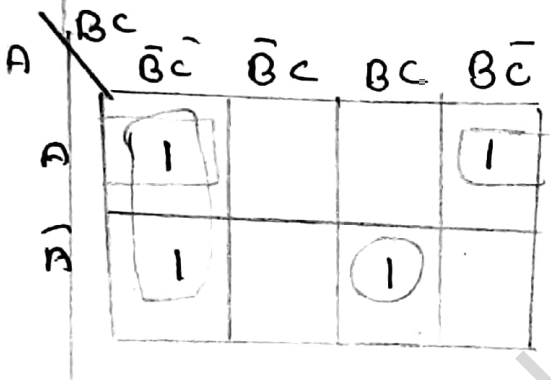
5. Find the minimum size of PLA required to implement the following function? Here implement the following function using PLA [December 2011 10 marks]

$F_1(A, B, C) = \sum m(0, 2, 4, 7)$

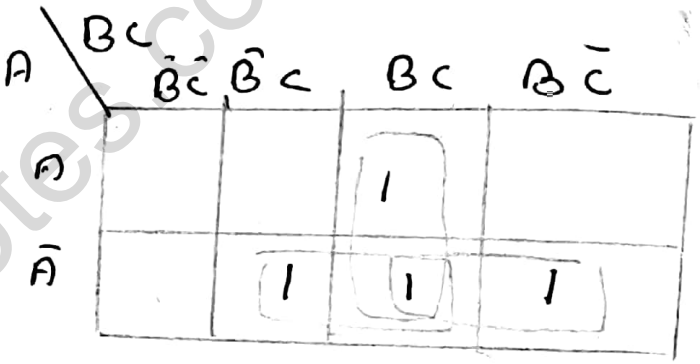
$F_2(A, B, C) = \sum m(3, 5, 6, 7)$

Ans:

F_1

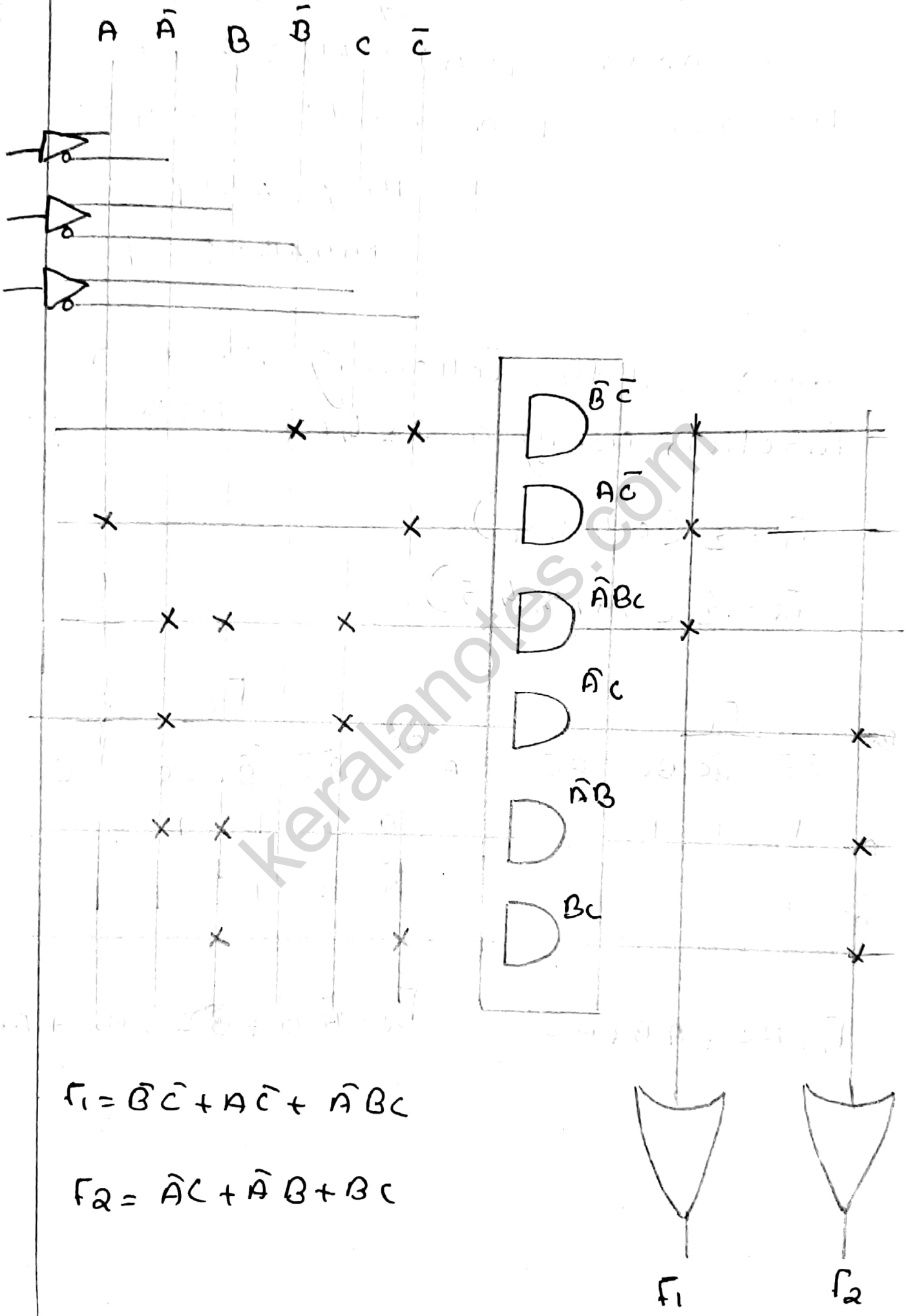


F_2



$F_1 = \bar{B}\bar{C} + A\bar{C} + \bar{A}BC$ $F_2 = \bar{A}C + \bar{A}B + BC$

Product term	Input			output	
	A	B	C	f_1	f_2
1. $\bar{B}\bar{C}$	—	0	0	1	—
2. $A\bar{C}$	1	—	0	1	—
3. $\bar{A}BC$	0	1	1	1	—
4. $\bar{A}C$	0	—	1	—	1
5. $\bar{A}B$	0	1	—	—	1
6. BC	—	1	1	—	1



$F_1 = \bar{B}\bar{C} + A\bar{C} + \bar{A}BC$

$F_2 = \bar{A}C + \bar{A}B + BC$

6. Describe the working of programmable logic array (PLA) with a block diagram and a simple example

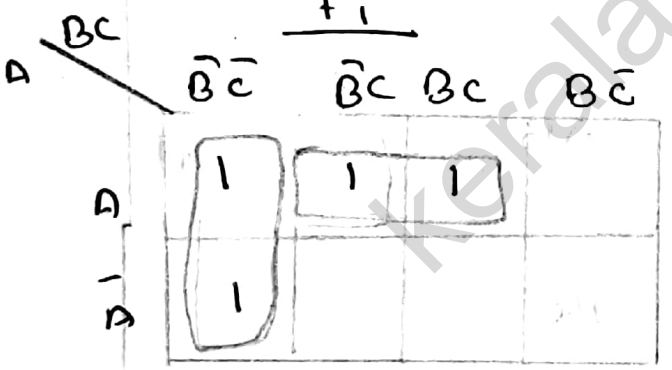
[July 2017
Mark: 6]

7. Implement the following Boolean function using a 3x4x2 PLA

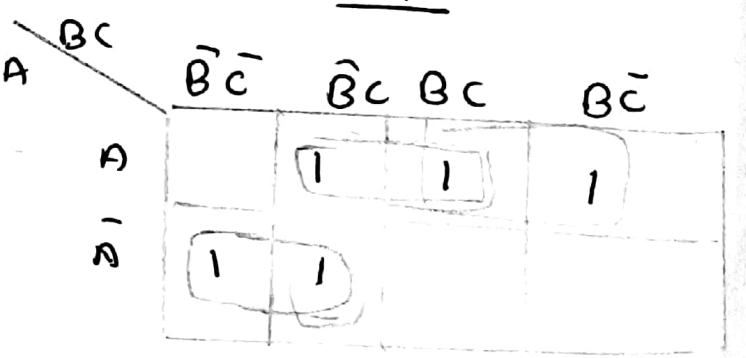
$F_1 = \Sigma(0,1,3,4)$

$F_2 = \Sigma(1,2,3,4,5)$

F_1



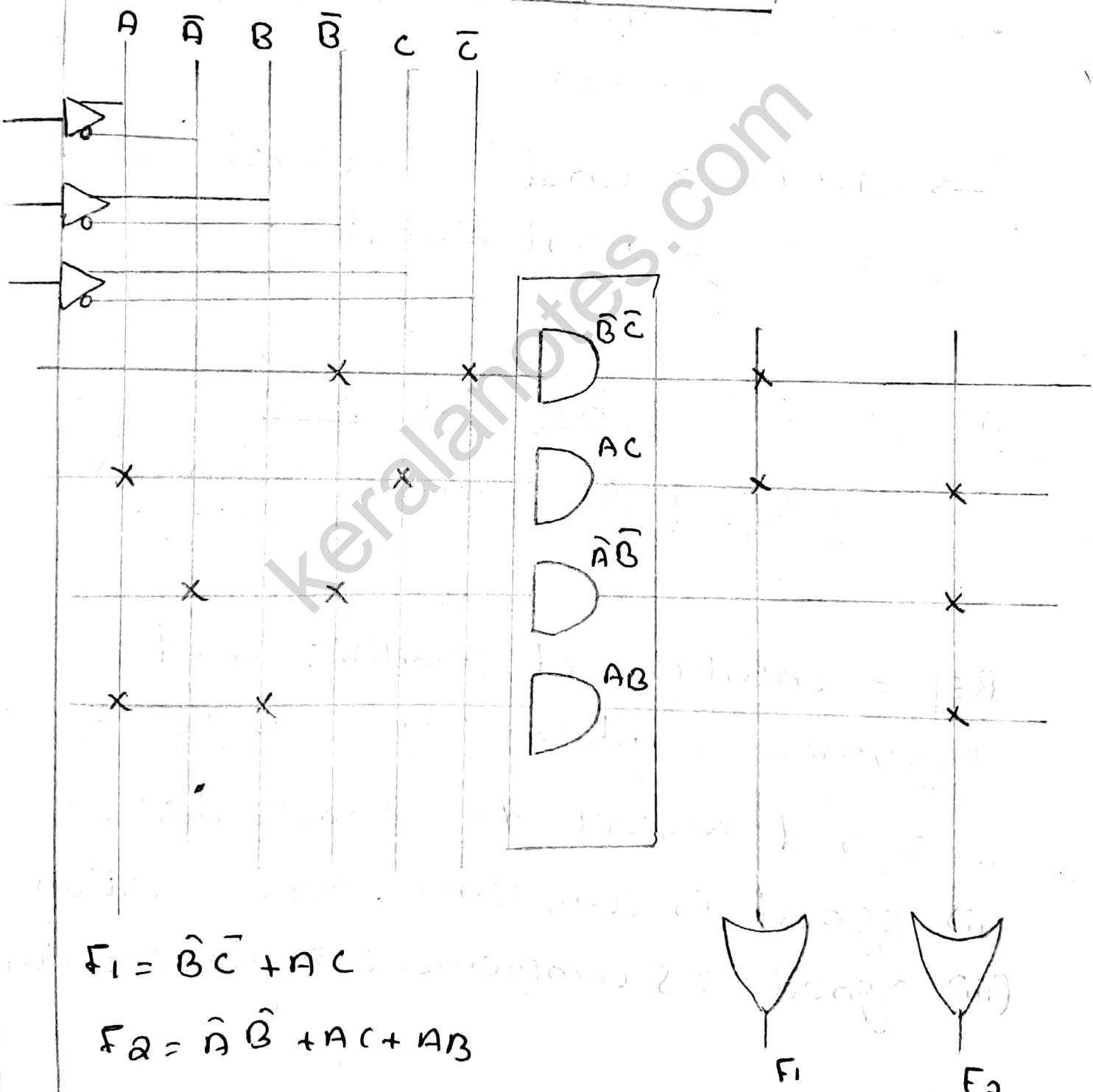
F_2



$F_1 = \bar{B}\bar{C} + AC$

$F_2 = \bar{A}\bar{B} + AC + AB$

Product term	Input			output	
	A	B	C	F_1	F_2
1. $\bar{B}\bar{C}$	—	0	0	1	—
2. AC	1	—	1	1	1
3. $\bar{A}\bar{B}$	0	0	—	—	1
4. AB	1	1	—	—	1



$F_1 = \bar{B}\bar{C} + AC$

$F_2 = \bar{A}\bar{B} + AC + AB$

Arithmetic Algorithms

→ Addition and substraction of Binary Numbers in signed magnitude and 2's complement representation

→ Addition and substraction of BCD numbers

→ Addition and substraction of floating point Numbers

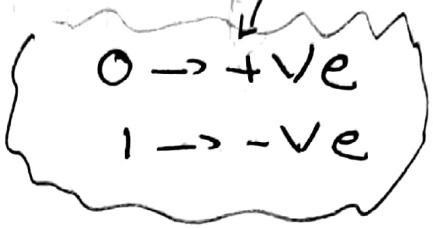
Addition & substraction of Signed magnitude Number

Representation of positive and negative number Representation

(i) Signed magnitude representation
(ii) signed 1's complement representation
(iii) Signed 2's complement representation

Eg: Represent +9 & -9 in 7 bit binary number

- +9 => 1001 -> 7 bit -> 0 0010 01



+9 => 0001001

-9: 3 ways

(i) signed magnitude: 1 0010 01

(ii) signed - 1's complement: 1 1101 10

1 replace others ie 1->0, 0->1

1 fixed [it represent -ve symbol]

(iii) signed - 2's complement:

1's comp + 1 => 2's comp

1110110 + 1 => 1110111

1110111

Eg: Represent +18 & -18 in 7 bit binary number

2	18	0
2	9	1
2	4	0
2	2	0
	1	

18 binary -> 10010

+18 => 0 10010

0 → +ve

-18 =>

(i) signed magnitude => 1 10010

(ii) signed 1's complement => 11011

(1) Replacing others
0 → 1
1 → 0

(ii) signed 2's complement =>

1's + 1

```
  1
10110 1 +
      1
-------
11011 1 0
```

=> 11011 0

Addition & substraction of signed magnitude form

Operation	Add magnitudes	Subtract magnitudes		
		when A > B	when A < B	when A = B
(+A) + (+B)	+ (A + B)			
(+A) + (-B)		+ (A - B)	- (B - A)	+ (A - B)
(-A) + (+B)		- (A - B)	+ (B - A)	+ (A - B)
(-A) + (-B)	- (A + B)			
(+A) - (+B)		+ (A - B)	- (B - A)	+ (A - B)
(+A) - (-B)	+ (A + B)			
~~(-A) - (-B)~~				
(-A) - (+B)	- (A + B)			
(-A) - (-B)		- (A - B)	+ (B - A)	+ (A - B)

Subtraction

1. A > B A → + symbol of A + (A - B)
   B < A B → - - (B - A) ↳ symbol of B
   A = B ± 0

2. (-A) + (+B)
   A > B - (A - B)
   B < A + (B - A)
   A = B + (A - B) = ± 0

3. (+A) - (+B)
   + (A - B)
   - (B - A)
   + (A - B)

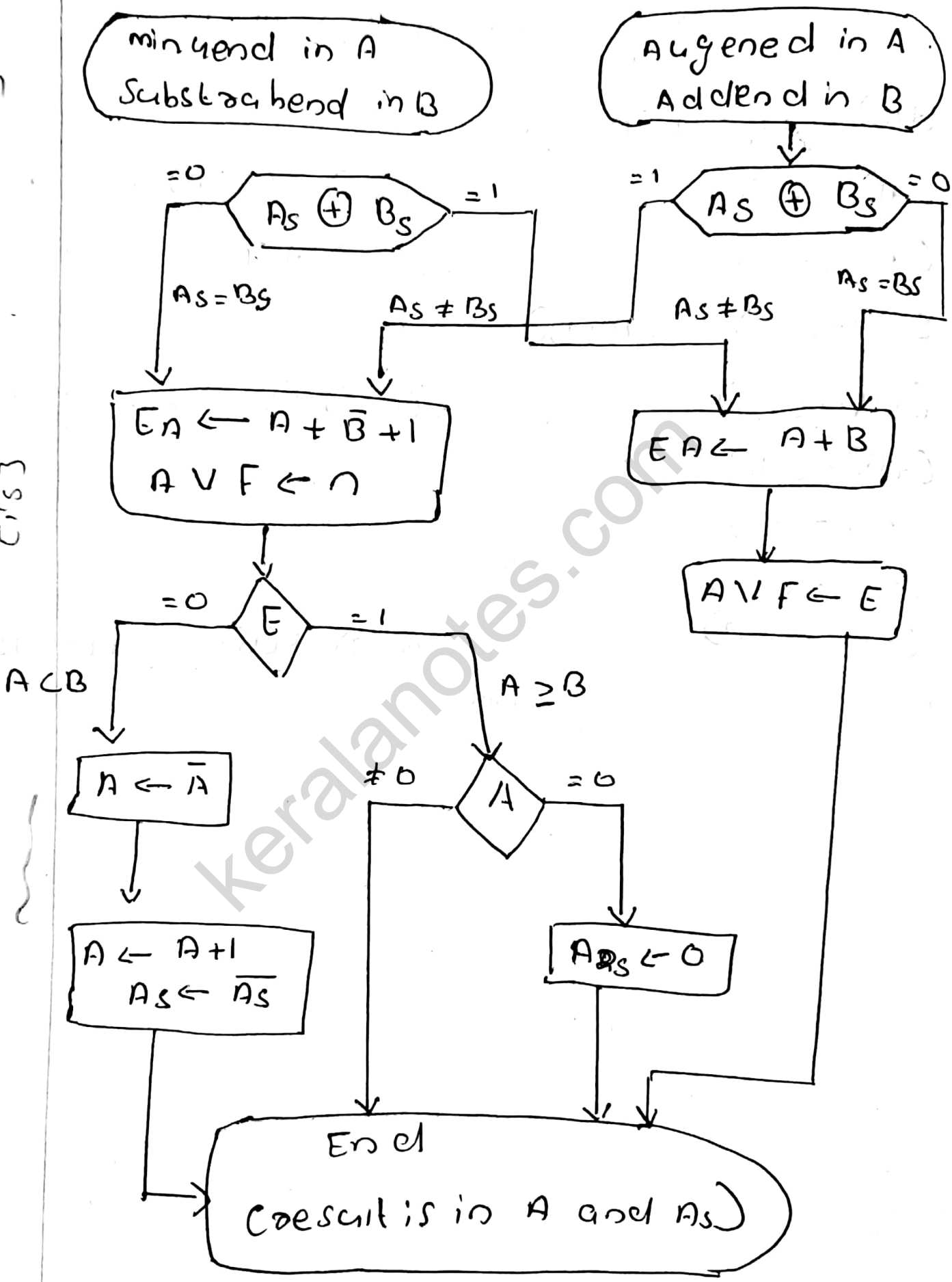
4. (-A) - (-B)
   - (A - B)
   - (B - A)
   + (A - B) = ± 0

Addition

1. (+A) + (+B) = + (A + B)
2. (-A) + (-B) = - (A + B)
3. (+A) - (-B)
   + (A + B)
4. (-A) - (-B)

Sub → 2's compliment method

Add



2's complement use A → Ā → A+1 [2's Comp]
(1's)

Description

→ As Sign of A

→ Bs Sign of B

→ As & A Accumulator

→ AVF overflow bit for A+B

→ E Output carry for parallel adder

XOR

0	0	0
0	1	1
1	0	1
1	1	0

If two sign (+ or -) are equal output => 0

If two sign are different output = 1

Floating point Representation

$$\pm \text{ mantissa} \times \text{Base}^{\pm \text{Exponent}}$$

Eg: 0.0000000000005

$= 5 \times 10^{-10}$

Eg: 5000000000000

$= 5 \times 10^{10}$

Normalisation Rules

1. The integer part should be zero
2. $0.d_1, d_2 \ldots d_n \times B^{\pm E}$

   Then $d_1 > 0$, and all $\sum_{i=2}^{n} d_i \geq 0$

   Eg: 1.23×10^3 × integer is not zero

   0.0123×10^5 ×

   0.123×10^4 ✓

st

Floating point Algorithm Arithmetic

Steps to ADD / Substract

KeralaNotes

Floating point Numbers

1. compare magnitude of two exponent and make suitable alignment to the number with smaller magnitude of exponent
2. perform addition or substraction
3. perform normalization by shifting resultant mantissa and adjusting the resultans exponent

Example:

1. Add 1.1100×2^4 and 1.1000×2^2

1. Alignment

$1.1000 \times 2^2 \rightarrow 2^4$

$1.1000 \times 2^2 \rightarrow 0.0110 \times 2^4$

2. Addition

$$\begin{array}{l} {\scriptstyle 1\ 1} \\ 1.1100 \times 2^4 \\ {\scriptstyle 1}\ 0.0110 \times 2^4 \\ \hline 10.0010 \times 2^4 \end{array} + \quad = 10.0010 \times 2^4$$

3. Normalisation

$= 10.0010 \times 2^4$

$= 0.10010 \times 2^5$ deleting in question 4 bit these 6 bit

0.10010×2^6

we need only 4 bit

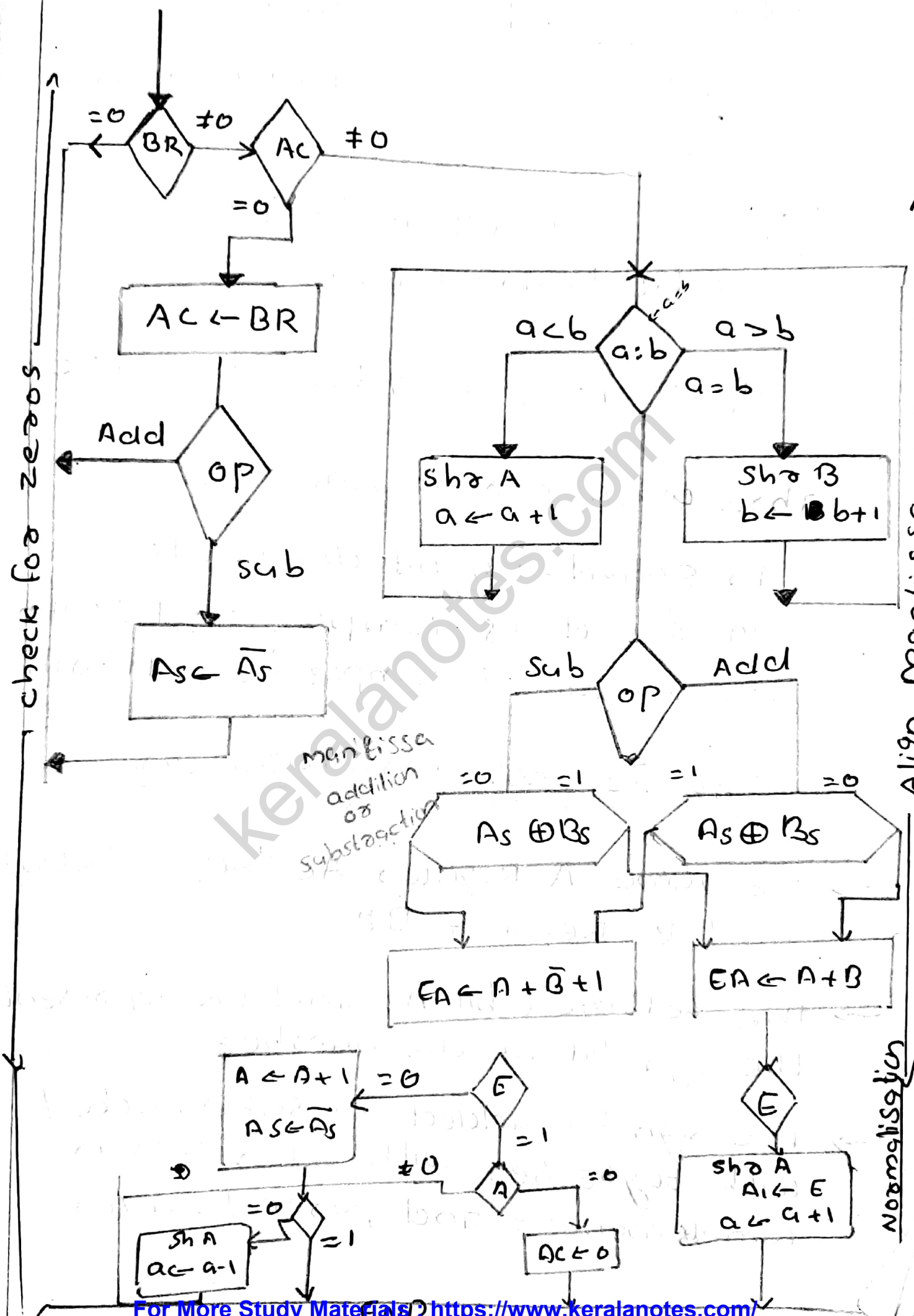
$$0.1000 \times 2^6$$

Algorithm for floating point Addition & substraction

During addition or substraction the two floating-point operands are AC and BR. The sum or difference is formed in the AC. The algorithm can be divided into four consecutive parts:

1. check for zeros
2. Align the mantissas
3. Add or substract the mantissas
4. Normalize the result

Add or Substract



Addition and substraction with signed 2's complent data

Eg: Repaesent +9 and -9 in 7 bit-binary Number

+9 => 1001 => 7 bit - 0001001 (+ve)

Thre ways to aepresent -9:

In signed-magnitude: 1001001
in signed-1's complement: 1110110
in signed-2's complement: 1110111

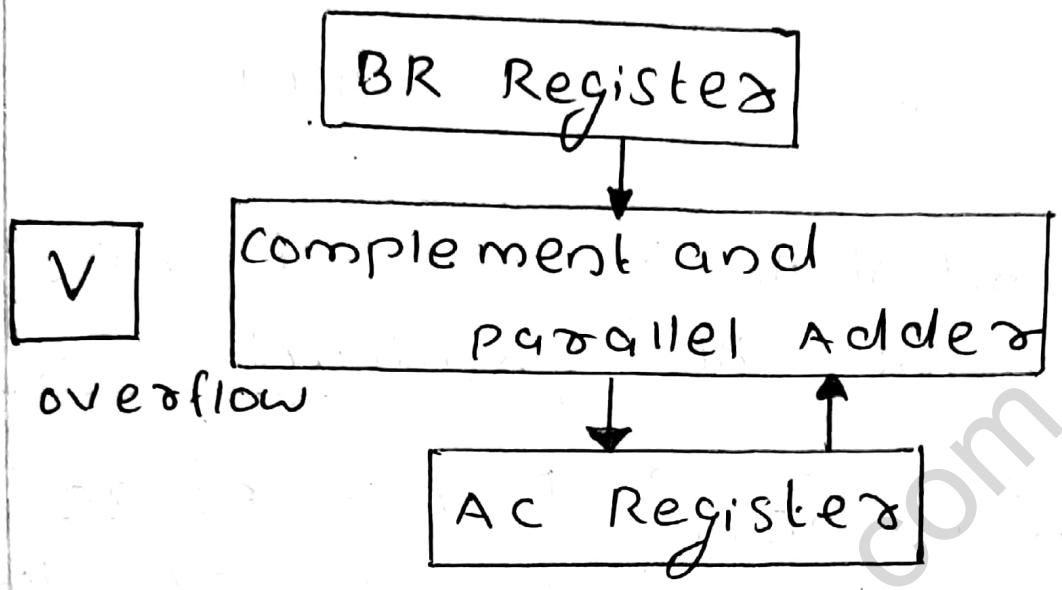
The Registea configuation

→ we name A Regisleo Ac (Accumulatoa) and B Regisleo BR

→ the leftmosl bit Ac and BR aepresents the sign bit of the numbea

→ The sign bit added oa substaacted with togbetheo with otheo bits in complementoa and paoallel addeo

→ The overflow flipflop V is set to 1 if there is overflow. The output carry is in this case is discarded



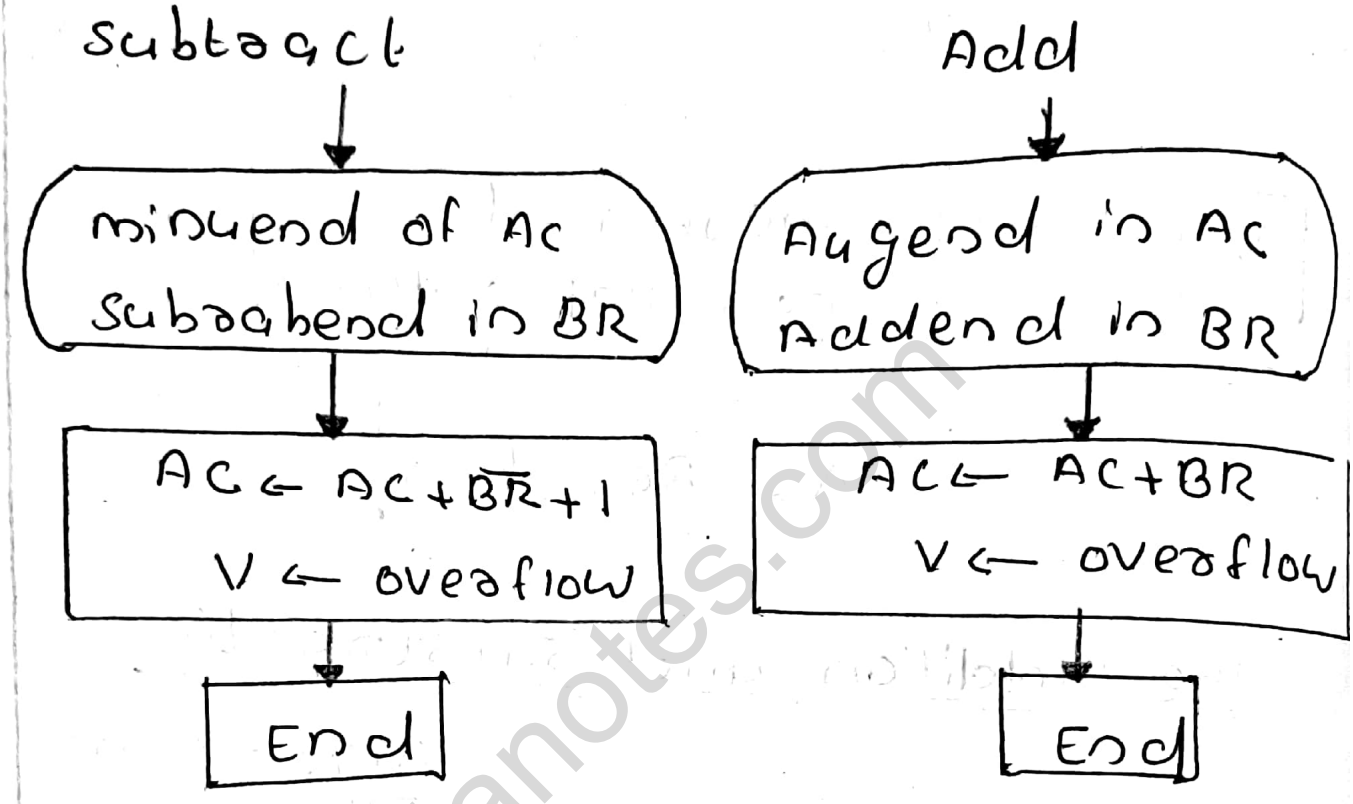
The Addition and substraction Algorithm

→ The sum is obtained by adding the content of AC and BR

→ The overflow bit V is set to 1 if the exclusive OR of the last two carries is 1, and it is cleared to 0 otherwise

→ The substraction operation is accomblished by adding the content of AC with 2's complement of BR

→ An overflow must be checked in this operation because two numbers added could have the same sign



BCD Adder

Binary coded decimal

→ Digital system handle decimal number in form of BCD

→ BCD Adder is a circuit adds two BCD digits and produce a sum digit also in BCD

Truth table

S_3	S_2	S_1	S_0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

GT
7
13
BCD ↓
0001 0011

consider
output 0-9
as 0
10-16 as 1

S_3S_2 \ S_1S_0	$\bar{S}_1\bar{S}_0$ (00)	$\bar{S}_1S_0$ (01)	S_1S_0 (11)	$S_1\bar{S}_0$ (10)
$\bar{S}_3\bar{S}_2$ (00)	0	0	0	0
$\bar{S}_3S_2$ (01)	0	0	0	0
S_3S_2 (11)	1	1	1	1
$S_3\bar{S}_2$ (10)	0	0	1	1

$Y = S_3S_2 + S_3S_1$

$Y = S_3S_2 + S_3S_1$

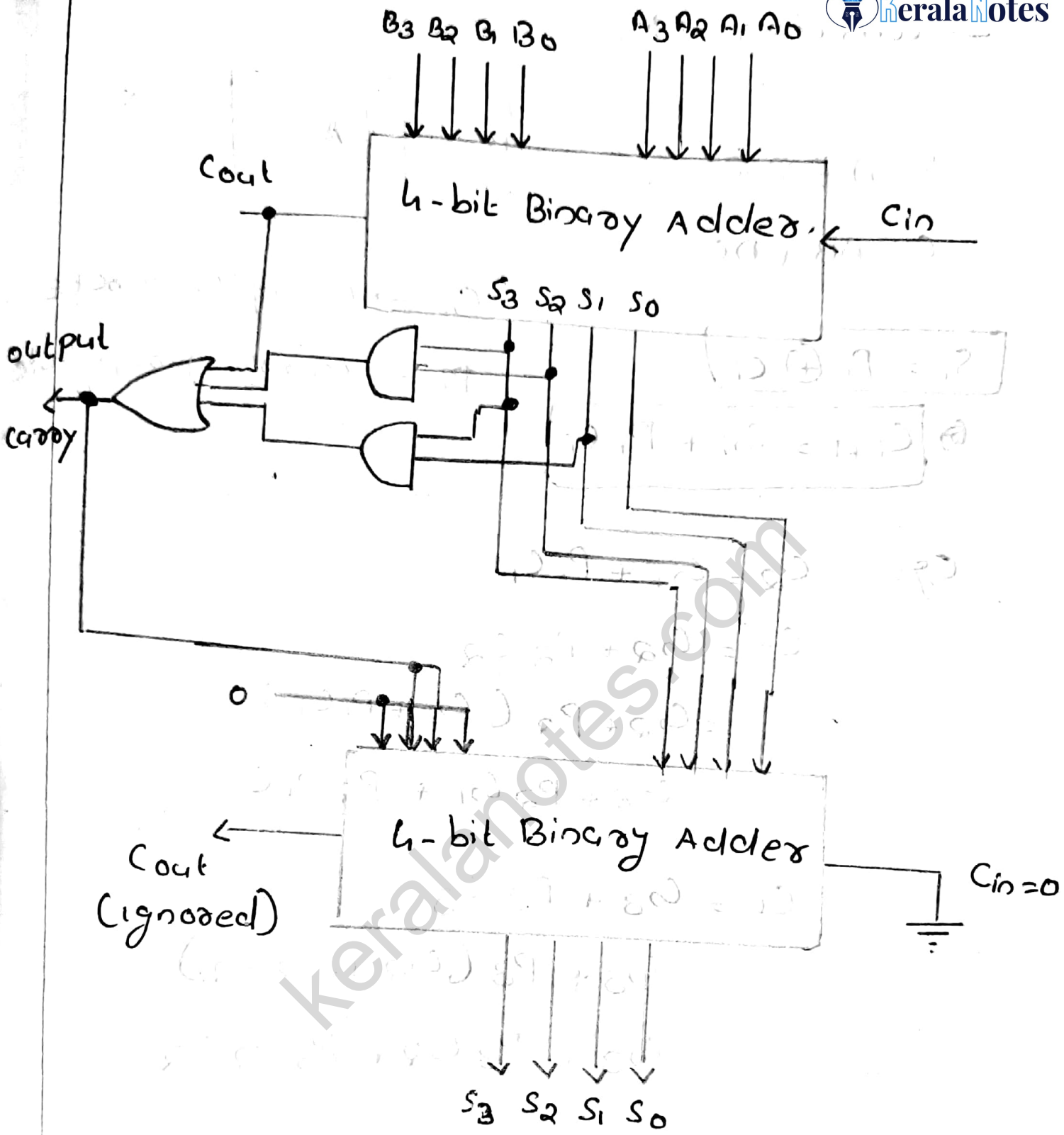
To implement BCD Adder:

1. 4-Bit Binary Adder for initial Addition
2. Logic circuit to detect sum greater than 9
3. one more 4 bit adder to add 0110 if the sum is greater than 9 or carry is 1

Sum > 9
Sum < 9, carry = 1
It will be invalid

To make it valid add 0110 (6)

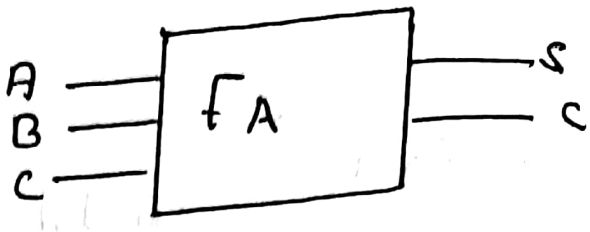
Block diagram of BCD Adder:



CArry Look Ahead Adder

→ one method of speeding up the process by eliminating inter stage carry delay called carry look ahead adder/ fast adder

→ Consider C_{i+1}



$S = A \oplus B \oplus C$

$C = AB + AC + BC$

G_i → carry generate

P_i → carry propoga

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i + P_i C_i$$

Eg: $C_2 = G_1 + P_1 C_1$

$C_3 = G_2 + P_2 C_2$

$= G_2 + P_2 (G_1 + P_1 C_1)$

$= G_2 + P_2 G_1 + P_2 P_1 C_1$

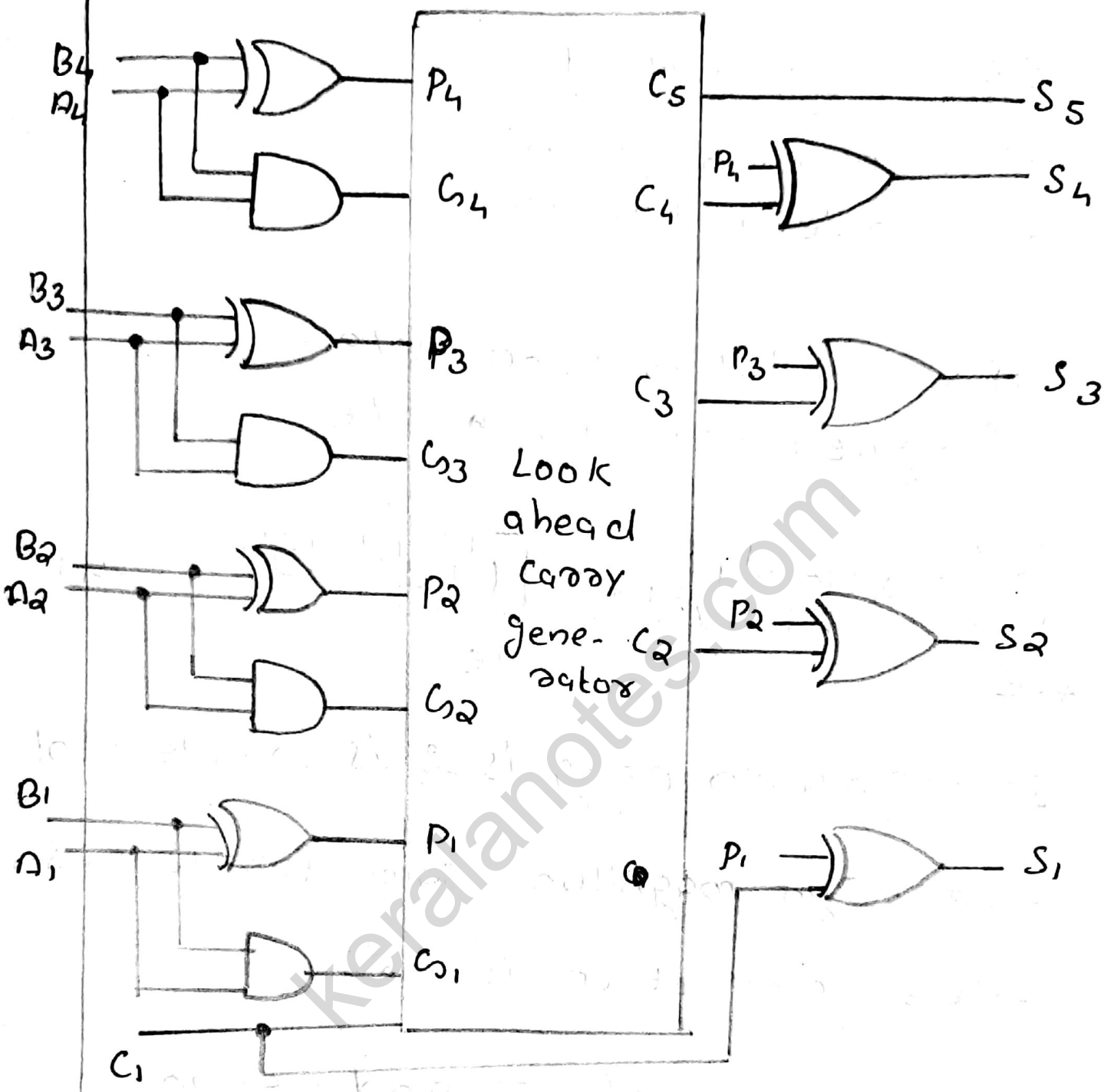
$C_4 = G_3 + P_3 C_3$

$= G_3 + P_3 (G_2 + P_2 C_2)$

$= G_3 + P_3 G_2 + P_3 P_2 C_2$

logic Diagram

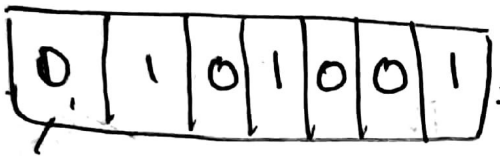
4 bit parallel adder with look ahead carry generator



Representation of Signed Numbers

Sign magnitude form
is complement form
2's complement form

if signed bit is zero, +ve
if signed bit is 1, -ve



= +41

+ve

= -41

-ve

Representation using is & 2's complement

+ve = Sign magnitude same for 1's & 2's

-ve = convert to is & 2's

1. Representation of +51 and -51 in 1's and 2's complement method

ans: 51 = 110011

signed magnitude

+ve = 0 110011 = +51

-ve = 10 110011 = -51

1's complement

+51 = 0 00 11 00 +51 1's complement

-51 = 1 00 11 00 -51 1's complement

2's complement

+51 = 0 00 11 00 -151 = 1 00 11 01

+1

0 00 11 01

a.

2. Each of the following number is signed binary number. Determine decimal value of Following cases

(i) signed magnitude form

(ii) 1's complement form

(iii) 2's complement form

(a) 0 1101

decimal = +13

1101

+ve so is s ds same

signed = +13

1's comp = +13

2's comp = +13

(b) 0 10111

decimal = +23

signed = +23

1's = +23

(c) 10111 - -ve

Decimal = 7 $\quad 7 = 0111$

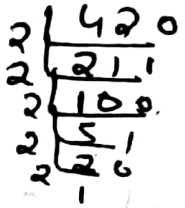
signed = -7

1's compl = 1 0111

= 1 1000 = -8

2's comp = 1 1001 = -9

(d) 1101010 -ve



Decimal = 42

signed = -42 $\quad 42 = 101010$

1's comp = 1 101010

= 1 010101 = -21

2's comp = 1 010101 +

1

010110

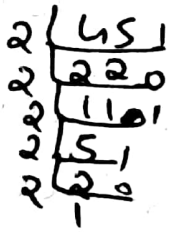
= 1 010110 = -22

3. Express -45 in 8 bit 2's complement form

~~45 = 101101 = 00101101~~

-45 = 1 1010011 = 1 0

8 bit



+45 = 00101101

-45 = | 1 | 0101101 |

1's = 1 1 0 1 0 0 1 0

2's = 1 1 0 1 0 0 1 1

4. Expoess -73.75 in 12 bit 2's complement Foam

+ 73.75 =

0 + 0 0 1 0 0 1 . 1 1 0 0

+ve

0.75 x 2 = 1.5
0.5 x 2 1.0
0.0 x 2 = 0.0
0.0 x 2 = 0.0

2	73 1
2	36 0
2	18 0
2	9 1
2	4 0
2	2 0
	1

-73.75

= 1 1 0 0 1 0 0 1 . 1 1 0 0

1's = 1 0 1 1 0 1 1 0 . 0 0 1 1

2's = 1 0 1 1 0 1 1 0 . 0 0 1 1 + = 1 0 1 1 0 1 1 0 . 0 1 0 0

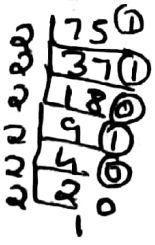
1 0 1 1 0 1 1 0 . 0 1 0 0

2's Complement Arithmetic

minuend

1. Add 2's complement of the subtrahend to
2. If carry, ignore carry
3. IF MSB = 0 result +ve and in binary form
4. IF MSB = 1 result -ve and in 2's comp form. Take its 2's comp to find magnitude in binary

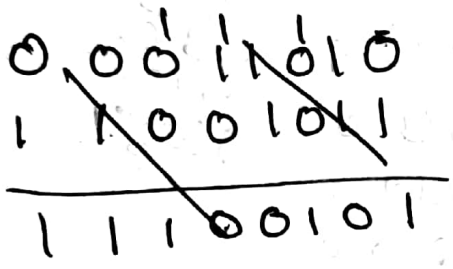
1. Add -75 + 26 using 8 bit 2's Arithmetic

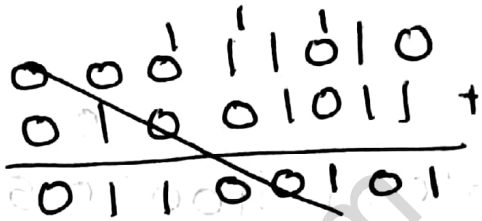


+75 = 01001011

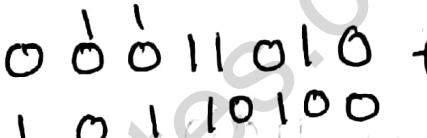
-75 = 1 1001011 $\xrightarrow{2's}$ 1 0110100

+26 = 00011010





+26
-75
49



(No carry)

No carry msB=1 so take 2's comp

⇒ 00110001 = 49

magnitude = -49

2. Add -45.75 to +87.5 using 12 bit 2's complement Arithmetic

$+45.75 = 101101.1100$

00101101.1100

0.7

$0.5 \times 2 = 1.0$

$0.0 \times 2 = 0.0$

$0.0 \times 2 = 0.0$

-45.75 = 1's comp

$= 11010010.0011$

2's = 11010010.0100 (+1)

$87.5 = 1010111.1000$

$0.5 \times 2 = 1.00$

$0.00 \times 2 = 0.00$

$0.0 \times 2 = 0.00$

$0.0 \times 2 = 0.0$

01010111.1000 +
11010010.0100
(1) 00101001.1100

neglect carry : $+41.75$

Floating point Number

$\pm M \times B^{E}$ — $\pm$ Exponent

$\pm$ mantissa x Base

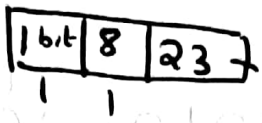
(i) 9×10^{8} — Mantissa = 9, Base = 10, Exponent = 8

(ii) 110×2^{7} — 110, 2, 7

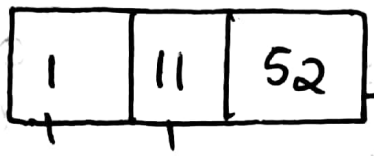
(iii) $4364.784 = 4364784 \times 10^{-3}$ — M = 4364784, Base = 10, exp = −3.

Floating point Representation

(i) Single precision format (32 bit)



(ii) Double precision format (64 bit)



Steps

1. convert decimal to Binary
2. Normalize the number
3. Apply single and double precesion equations

Single precision $(1.N)2^{E-127}$

Double precision $(1.N)2^{E-1023}$

1. Represent $(1259.125)_{10}$ in single and double precision format

① $(1001110101011.001)_2$

0.125 × 2 = 0.25
0.25 × 2 = 0.5
0.5 × 2 = 1.0

② Normalize

~~0.1001110101011~~ should be 24 bit

$1.00111010110010 \times 10^{10}_2$

③ Single precision $(1.N)2^{E-127}$

E-127 = 10

E = 137

convert 137 → Binary = $(10001001)_2$

=

1	10001001	00111010110010000000000
Sign	exponent	Mantissa
1	8	23

Double precision $(1.N)2^{E-1023}$

E-1023 = 10

E = 1033 → Binary = 10000001001

	10000001001	00111010110010000……00
sign	Exponent	Mantissa
1	11	

Fixed point Representation

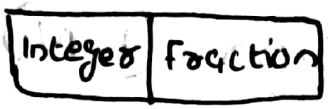
- Have fixed number of bit for integer part and for fractional part

Eg: 1111.FFFF

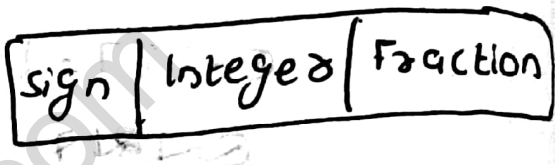
Minimum: ~~0.00~~ 0000.0001

Maximum: 9999.9999

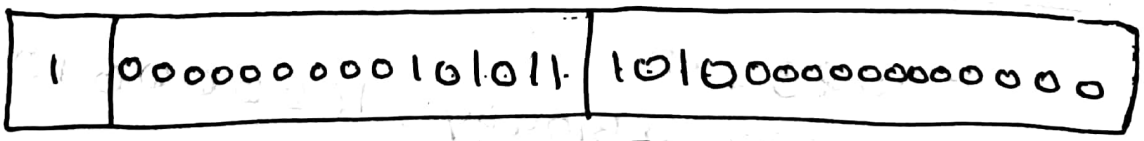
Unsigned Fixed point



signed Fixed po



Q Assume number is using 32-bit format which reserve 1 bit for the sign, 15 bit for integer part. and 16 bits for the fractional part then -43.625 is represented as follows



sign bit — Integer part 15 bit — Fractional part 16 bit

43=101011

0.625x2=(1)25

0.25x2=(0).5

0.5x2=(1)0

0.0x2=(0).0