



BMS

INSTITUTE OF TECHNOLOGY AND MANAGEMENT

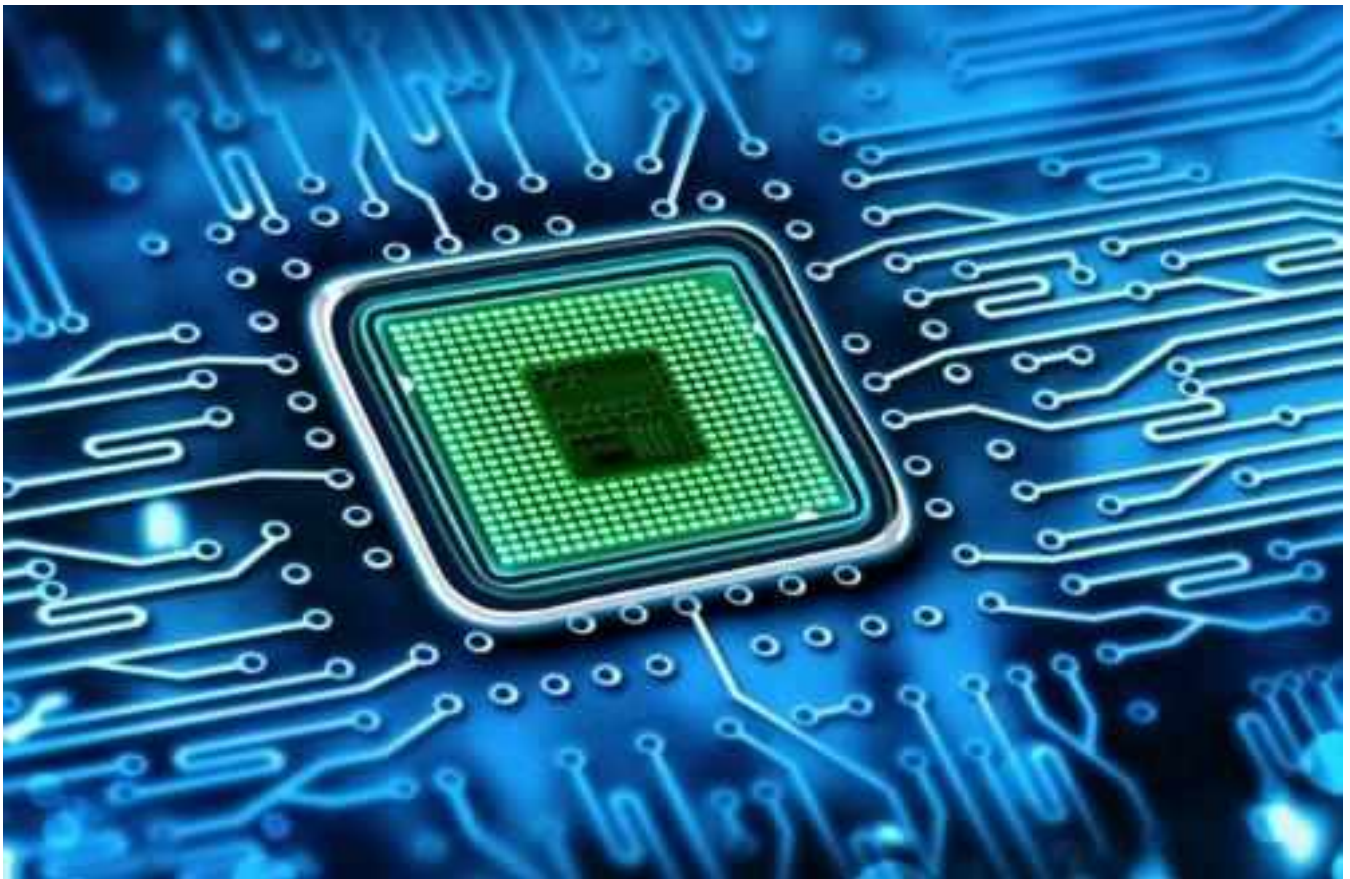
Avalahalli, Doddaballapur Main Road, Bengaluru – 560064

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Analog and Digital Electronics

18CS33

III Semester



Ravi Kumar B N

Assistant Professor Dept. of CSE

(1)

⇒ Karnaugh Maps (K-Map)

It is the visual display of the fundamental products needed for a sum of product solution.

⇒ Sum of Products (SOP) - OR of AND terms containing complemented or uncomplemented variables.

⇒ Fundamental products for Two Inputs

A	B	FP
---	---	----

0	0	$\bar{A}\bar{B}$
---	---	------------------

0	1	$\bar{A}B$
---	---	------------

1	0	$A\bar{B}$
---	---	------------

1	1	AB
---	---	------

→ The Fundamental products are also called minterms.

→ The products $\bar{A}\bar{B}$, $\bar{A}B$, $A\bar{B}$, AB are represented by m_0 , m_1 , m_2 , m_3 respectively.

→ The suffix i of m_i comes from decimal equivalent of binary values.

→ Fundamental Product for three inputs

A	B	C	FP
---	---	---	----

0	0	0	$\bar{A}\bar{B}\bar{C}$
---	---	---	-------------------------

0	0	1	$\bar{A}\bar{B}C$
---	---	---	-------------------

0	1	0	$\bar{A}B\bar{C}$
---	---	---	-------------------

0	1	1	$\bar{A}BC$
---	---	---	-------------

1	0	0	$A\bar{B}\bar{C}$
---	---	---	-------------------

1	0	1	$A\bar{B}C$
---	---	---	-------------

1	1	0	$AB\bar{C}$
---	---	---	-------------

1	1	1	ABC
---	---	---	-------

→ Sum of products equation

Consider a truth table.

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1 $\rightarrow m_3$
1	0	0	0
1	0	1	1 $\rightarrow m_5$
1	1	0	1 $\rightarrow m_6$
1	1	1	1 $\rightarrow m_7$

Locate each output 'one' in the truth table & write the fundamental product. To get the sum of product eqⁿ, we have to OR the fundamental product.

SOP eqⁿ:

$$Y = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

Alternate expⁿ is

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

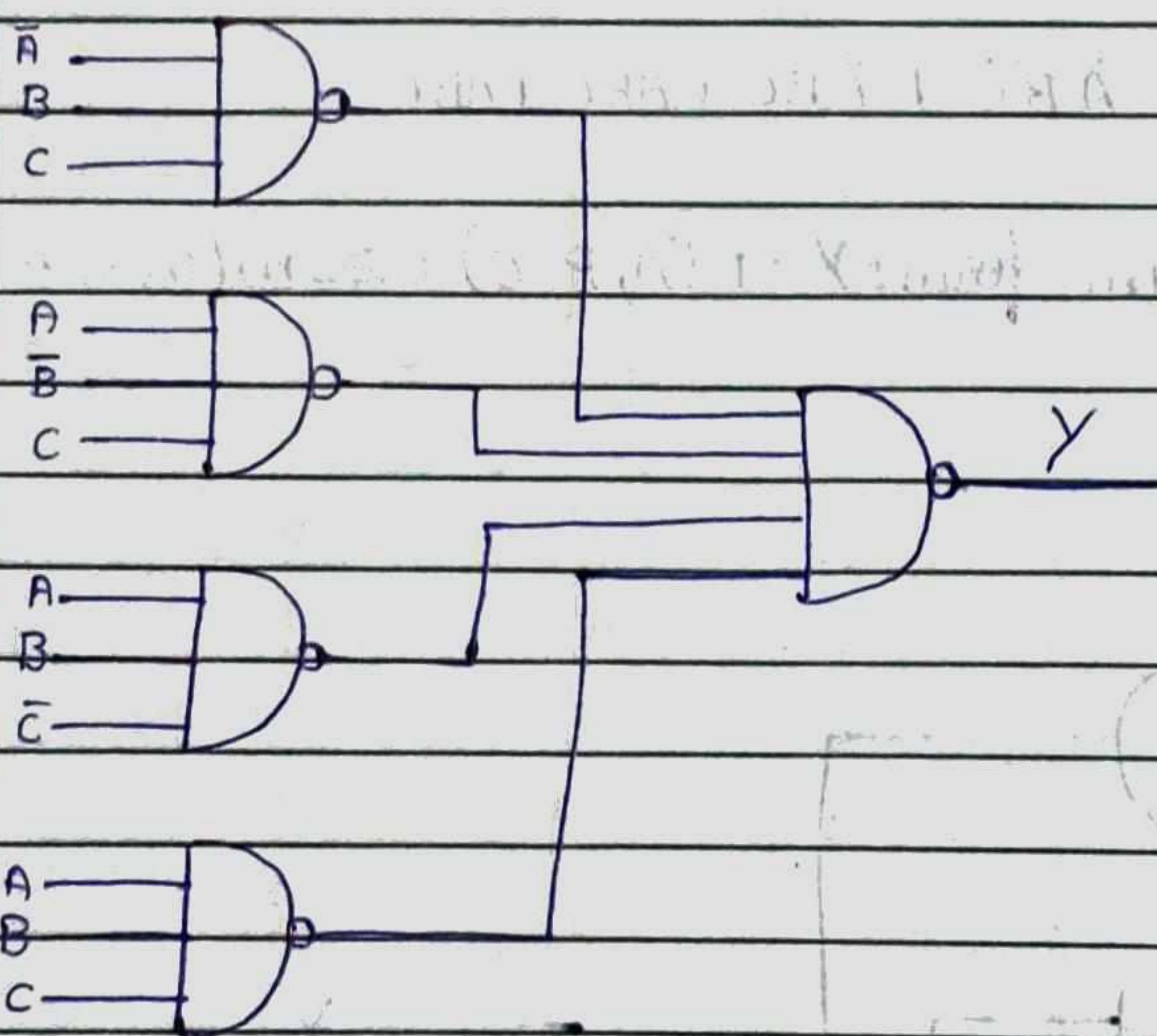
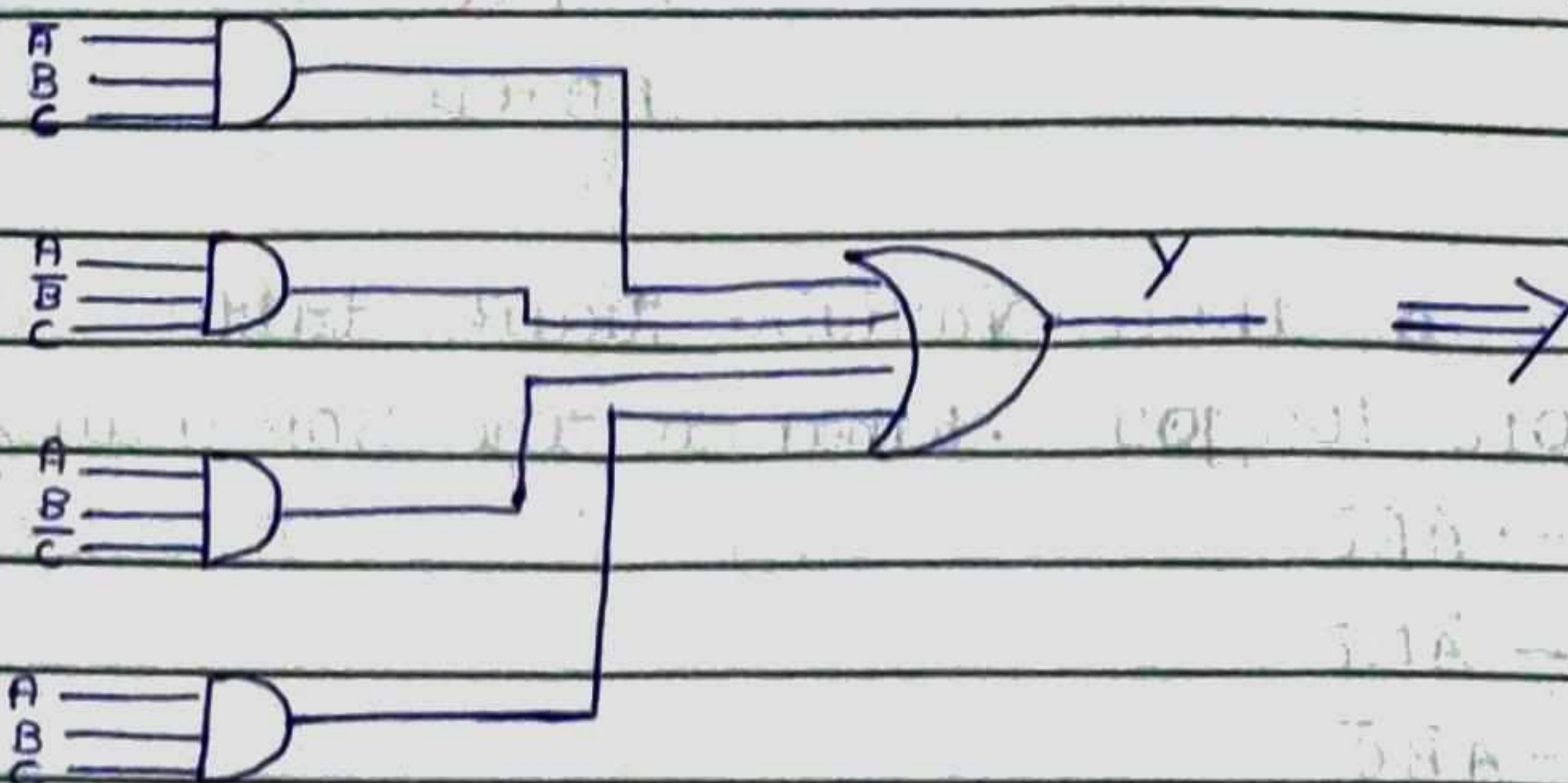
It is a canonical sum form where ' \sum ' symbolizes summation or logical OR operation that is performed on corresponding minterms.

Y is the function of 3 boolean variables A, B & C

③

→ Logic circuit.

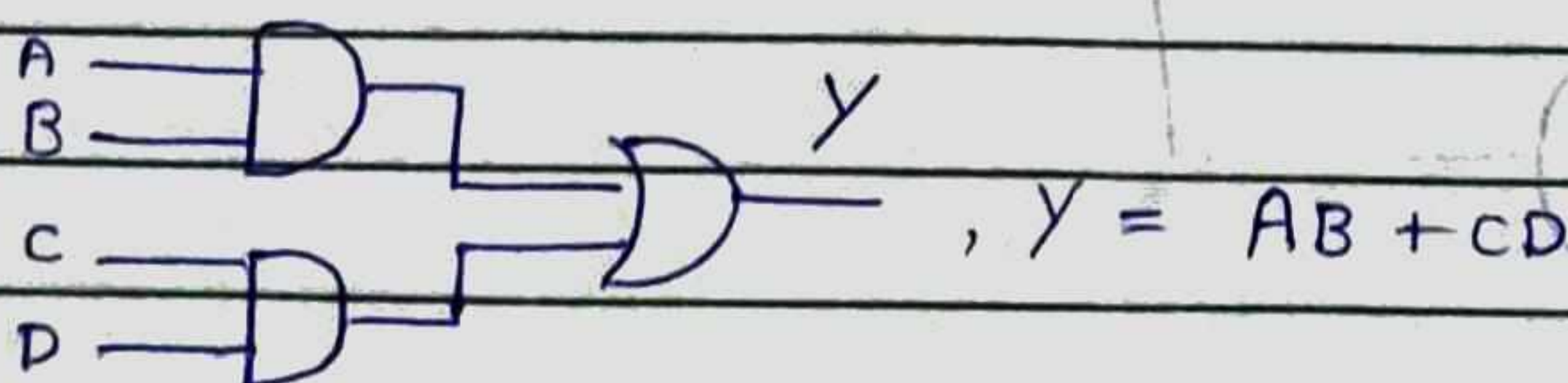
After sum of products eqⁿ, we can derive the corresponding logic circuit by drawing AND-OR network or NAND-NAND network.



Note

AND-OR and NAND-NAND gives the same o/p.

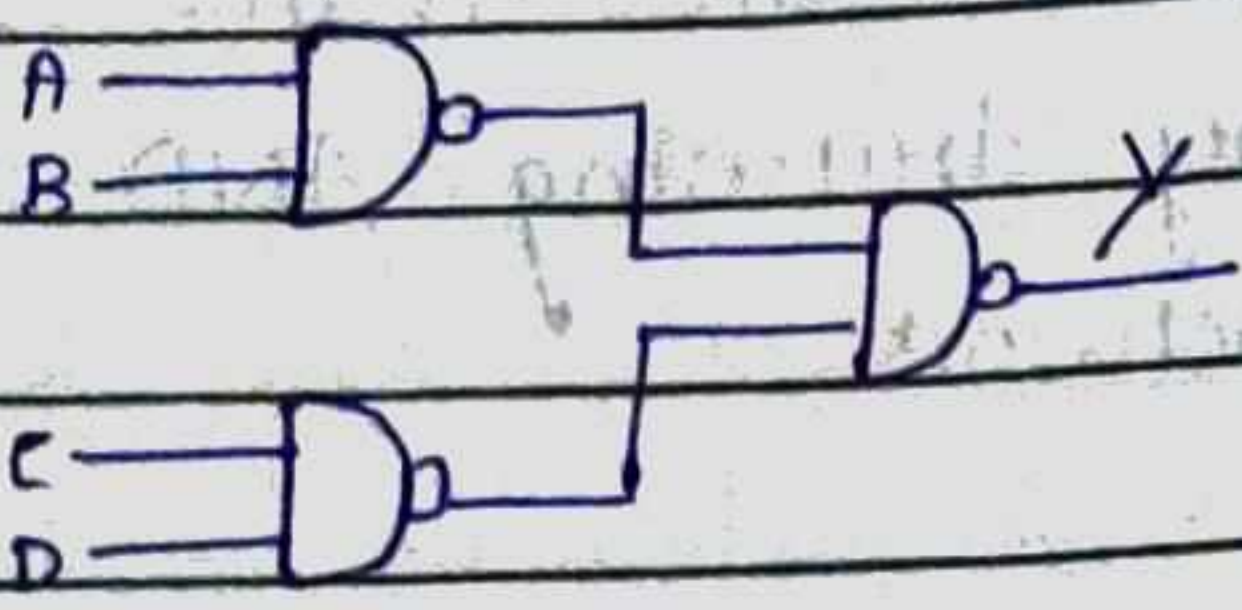
AND-OR



$$Y = AB + CD$$

(4)

NAND - NAND



$$Y = \overline{\overline{AB} \cdot \overline{CD}}$$

$$= \overline{\overline{AB}} + \overline{\overline{CD}}$$

$$= AB + CD$$

⇒ Suppose a three-variable truth table.
000, 010, 100, 110. What is the SOP circuit?

S:- $000 \rightarrow \overline{A}\overline{B}\overline{C}$

$010 \rightarrow \overline{A}B\overline{C}$

$100 \rightarrow A\overline{B}\overline{C}$

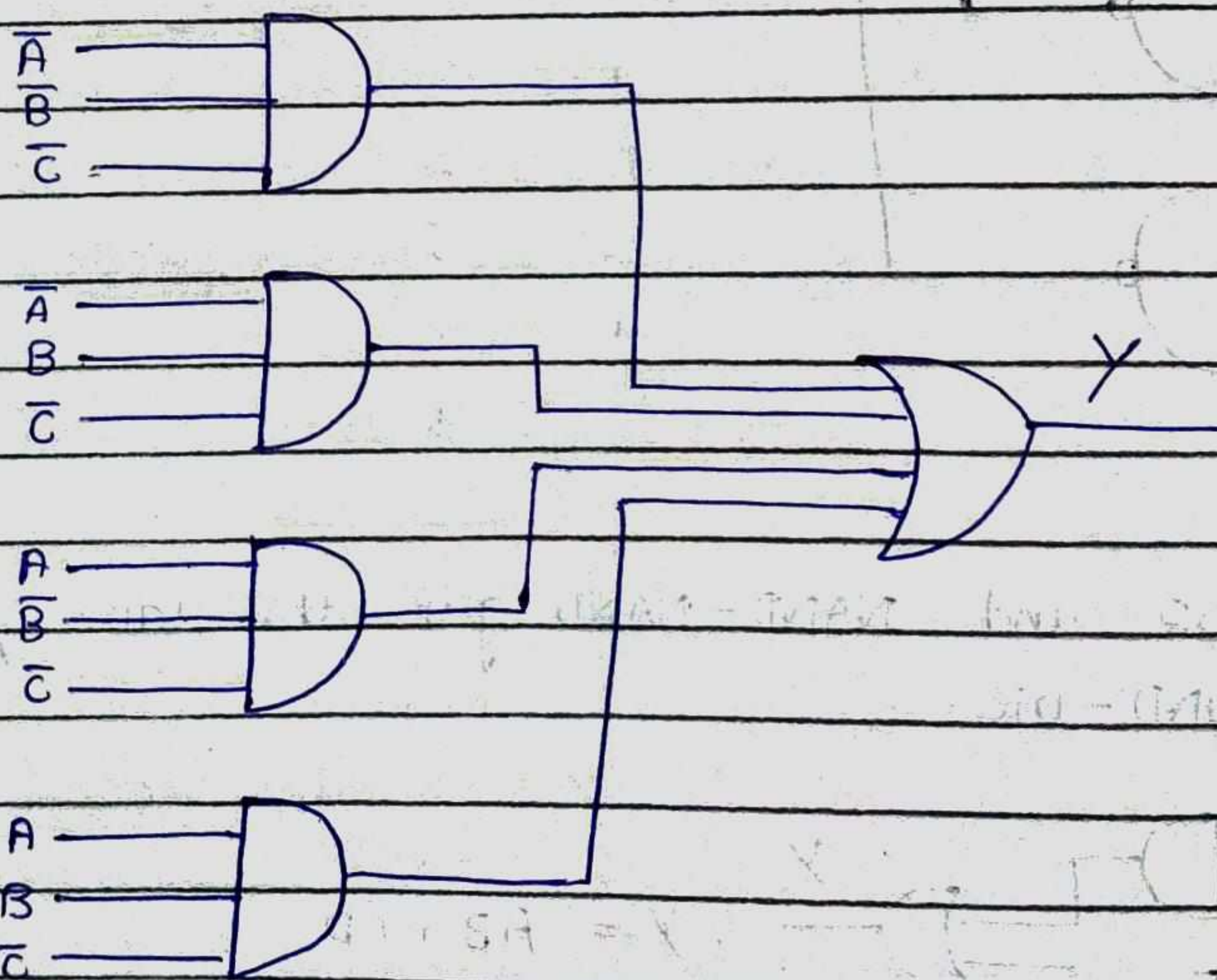
$110 \rightarrow AB\overline{C}$

SOP eqⁿ: $Y = \overline{A}\overline{B}\overline{C} + \overline{A}B\overline{C} + A\overline{B}\overline{C} + AB\overline{C}$

Canonical Sum form: $Y = F(A, B, C) = \sum m(0, 2, 4, 6)$

Logic circuit.

AND-OR



→ K-Map

Truth Table to K-Map.

→ Two Variable K-Map.

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

A \ B	\bar{B}	B
\bar{A}	0 ₀	0 ₁
A	1 ₂	1 ₃

→ Three Variable K-Map.

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

6

AB \ C		
	\bar{C}	C
$\bar{A}\bar{B}$	0 ₀	0 ₁
$\bar{A}B$	1 ₂	0 ₃
AB	1 ₆	1 ₇
$A\bar{B}$	0 ₄	0 ₅

→ Four Variable K-map

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	1 → 1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1 → 6
0	1	1	1	1 → 7
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1 → 14
1	1	1	1	0

AB \ CD				
	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0 ₀	1 ₁	0 ₃	0 ₂
$\bar{A}B$	0 ₄	0 ₅	1 ₇	1 ₆
AB	0 ₁₂	0 ₁₃	0 ₁₅	1 ₁₄
$A\bar{B}$	0 ₈	0 ₉	0 ₁₁	0 ₁₀

⇒ K-maps.

Pairs, Quads & Octets

AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	1	0	0
$\bar{A}B$	0	0	1	1
AB	0	0	0	0
$A\bar{B}$	0	0	0	1

$$Y = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}C\bar{D}$$

Overlapping

AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	1	0
$\bar{A}B$	1	1	1	1
AB	1	1	1	1
$A\bar{B}$	0	0	0	0

$$X = B + \bar{A}CD$$

Rolling the map.

AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	1	1
$\bar{A}B$	0	0	0	0
AB	1	1	0	1
$A\bar{B}$	0	0	1	1

$$Y = \bar{B}C + A\bar{B}\bar{C} + AB\bar{D}$$

8

Note:

AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	1	0	0	1
AB	1	0	0	1
$A\bar{B}$	0	0	0	0

$$Y = B\bar{C}\bar{D} + BCD$$

[not simplified, apply rolling the map]

AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	1	0	0	1
AB	1	0	0	1
$A\bar{B}$	0	0	0	0

$$Y = B\bar{D}$$

→

AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	1	0	1
$\bar{A}B$	1	1	0	1
AB	1	1	0	0
$A\bar{B}$	1	1	0	1

$$Y = \bar{C} + \bar{A}\bar{D} + \bar{B}\bar{D}$$

(9)

→ Eliminating the redundant groups

AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	1	0
$\bar{A}B$	1	1	1	0
AB	0	1	1	1
$A\bar{B}$	0	1	0	0

X

All the 1's of the Quad are used by the pairs because of this the quad is redundant & can be eliminated

AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	1	0
$\bar{A}B$	1	1	1	0
AB	0	1	1	1
$A\bar{B}$	0	1	0	0

✓

$$Y = \bar{A}\bar{B}\bar{C} + \bar{A}CD + ABC + A\bar{C}B$$

⇒ Summary of K-Map method for simplifying Boolean equations.

- Enter a '1' on the K-Map for each fundamental product that produces a one output in the truth table. Enter zeroes else where.
- Encircle the octets, quads & pairs. Remember to allow & overlap to get the largest groups possible.
- If any isolated 1's remain, encircle each.
- Eliminate any redundant group.
- Write the Boolean eqⁿ by ORing the products corresponding to the encircled groups.

(10)

1 \Rightarrow

AB \ CD				
	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	0	0	1	0
AB	1	1	1	1
$A\bar{B}$	0	1	1	1

$$Y = AD + AC + AB + BCD$$

$$2. Y = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}BCD + AB\bar{C}D + ABCD$$

$$\text{Canonical form: } Y = F(A, B, C, D) = \sum m(0, 7, 13, 15)$$

AB \ CD				
	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	0	0	0
$\bar{A}B$	0	0	1	0
AB	0	1	1	0
$A\bar{B}$	0	0	0	0

$$Y = ABD + BCD + \bar{A}\bar{B}\bar{C}\bar{D}$$

$$3. Y = F(A, B, C, D) = \sum m(1, 5, 6, 7, 9, 15)$$

AB \ CD				
	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0 ₀	1 ₁	0 ₃	0 ₂
$\bar{A}B$	0 ₄	1 ₅	1 ₇	1 ₆
AB	0 ₁₂	0 ₁₃	1 ₁₅	0 ₁₄
$A\bar{B}$	0 ₈	1 ₉	0 ₁₁	0 ₁₀

$$Y = \bar{A}BD + \bar{A}BC + BCD + \bar{B}\bar{C}\bar{D}$$

4.

AB \ CD	BD	CD	CD	C \bar{D}
AB	1	0	0	1
$\bar{A}B$	1	0	1	1
AB	1	0	1	1
$\bar{A}\bar{B}$	1	0	1	1

$$Y = \bar{D} + BC + AC$$

5.

AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	C \bar{D}
$\bar{A}\bar{B}$	1	0	0	0
$\bar{A}B$	0	1	1	0
AB	0	0	0	0
$\bar{A}\bar{B}$	0	0	0	1

$$Y = \bar{A}BD + \bar{A}\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D}$$

Proof: $Y = \bar{A}\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D}$
 $= \bar{B}\bar{D}(\bar{A}\bar{C} + AC)$

6.

AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	C \bar{D}
$\bar{A}\bar{B}$	1	0	0	1
$\bar{A}B$	0	0	0	0
AB	0	0	0	0
$\bar{A}\bar{B}$	1	0	0	1

$$\begin{aligned}
 Y &= \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} \\
 (\text{single}) &= \bar{A}\bar{B}\bar{D}(\bar{C} + C) + \bar{A}\bar{B}\bar{D}(\bar{C} + C) \\
 &= \bar{A}\bar{B}\bar{D} + \bar{A}\bar{B}\bar{D} \\
 &= \bar{B}\bar{D}(\bar{A} + A) \\
 &= \bar{B}\bar{D}
 \end{aligned}$$

$$Y = \bar{B}\bar{D} \quad (\text{Quod})$$

Don't care condition.

In some digital systems certain input conditions never occur during normal operation. Therefore the corresponding o/p never appears. It is indicated by 'X' in the truth table. We can use this as either 0 or 1, whichever produces a simpler logic circuit.

AB \ CD				
	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	0	0	0	0
AB	X	X	X	X
$A\bar{B}$	0	1	X	X

$$Y = AD$$

AB \ CD				
	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	1	1	0
$\bar{A}B$	0	1	0	X
AB	X	X	X	X
$A\bar{B}$	X	1	1	X

$$Y = A + \bar{C}D + \bar{B}D$$

Octet is redundant, $\therefore Y = \bar{B}D + \bar{C}D$

AB \ CD	CD			
	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	1	X	0
$\bar{A}B$	1	X	0	X
AB	X	0	1	1
$A\bar{B}$	X	X	X	X

$$Y = AC + \bar{B}D + B\bar{D}$$

AB \ CD	CD			
	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	0	0	1
$\bar{A}B$	X	1	1	0
AB	X	X	X	1
$A\bar{B}$	X	1	1	X

$$Y = A + BD + \bar{B}\bar{D}$$

AB \ CD	CD			
	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	X	0	X
$\bar{A}B$	X	0	X	0
AB	X	1	1	0
$A\bar{B}$	0	X	X	0

$$Y = AD + \bar{A}\bar{B}\bar{C}$$

AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	X	X	1
$\bar{A}B$	1	X	X	1
$A\bar{B}$	X	0	0	X
AB	1	0	0	X

$$Y = \bar{D}$$

AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	X	0	1	X
$\bar{A}B$	0	0	0	1
$A\bar{B}$	1	X	X	0
AB	X	0	X	1

$$Y = \bar{B}C + \bar{A}C\bar{D} + A\bar{C}\bar{D}$$

AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	X	X	X
$\bar{A}B$	0	0	0	0
$A\bar{B}$	X	X	X	1
AB	0	1	1	0

$$Y = \bar{A}\bar{B} + AB + AD$$

⇒ Product of Sums (Pos) Method.

The fundamental sum produces an o/p 0 for the corresponding i/p condition.

Converting a truth table to an equation

Fundamental sum

A	B	C	Y	Minterm (M)
0	0	0	$0 \rightarrow A+B+C$	M_0
0	0	1	1	
0	1	0	1	
0	1	1	$0 \rightarrow A+\bar{B}+\bar{C}$	M_3
1	0	0	1	
1	0	1	1	
1	1	0	$0 \rightarrow \bar{A}+\bar{B}+C$	M_6
1	1	1	1	

POS eqⁿ: $Y = (A+B+C)(A+\bar{B}+\bar{C})(\bar{A}+\bar{B}+C)$

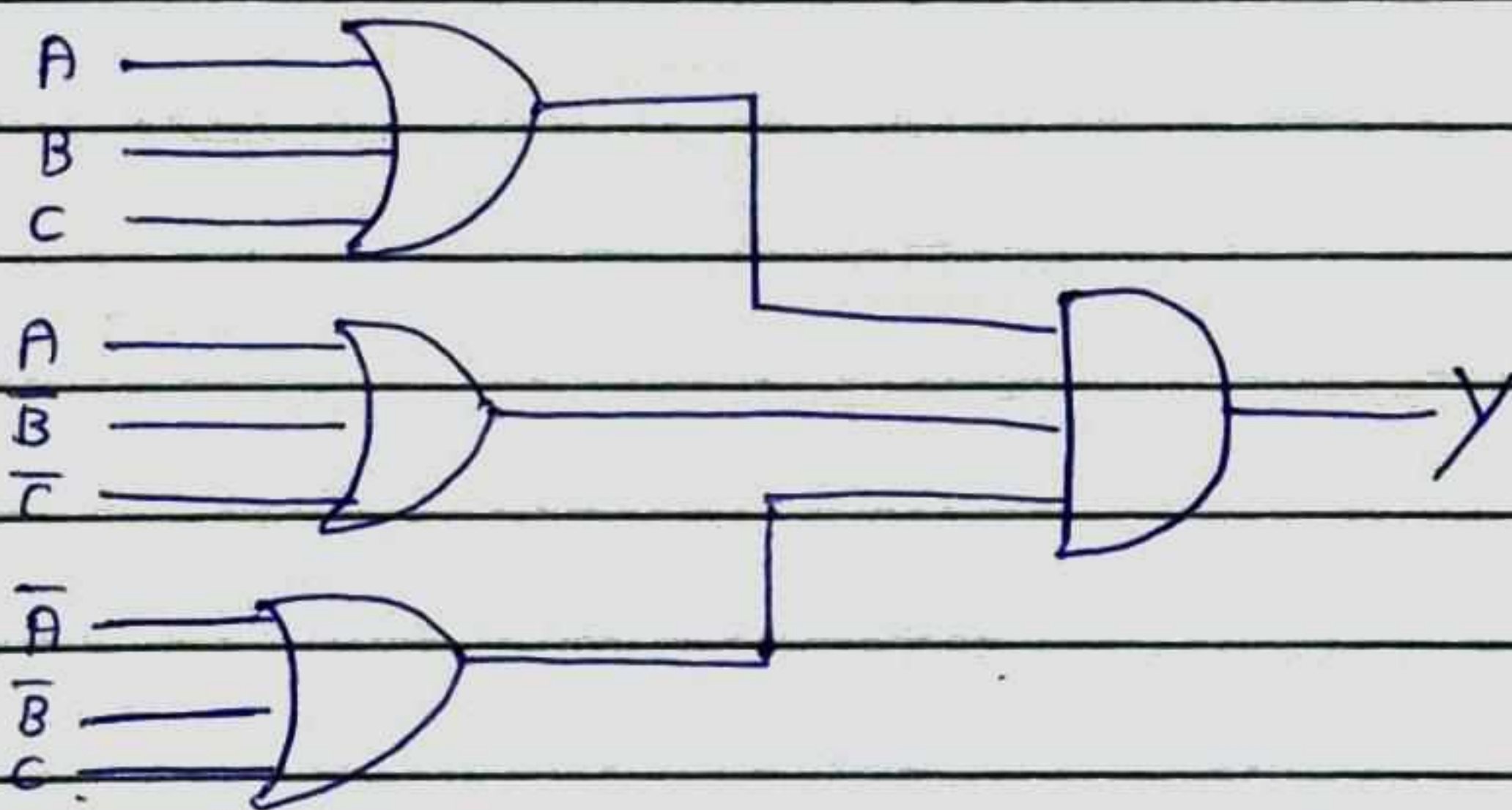
Canonical product form: $Y = F(A,B,C) = \pi M(0,3,6)$

→ Logic circuit

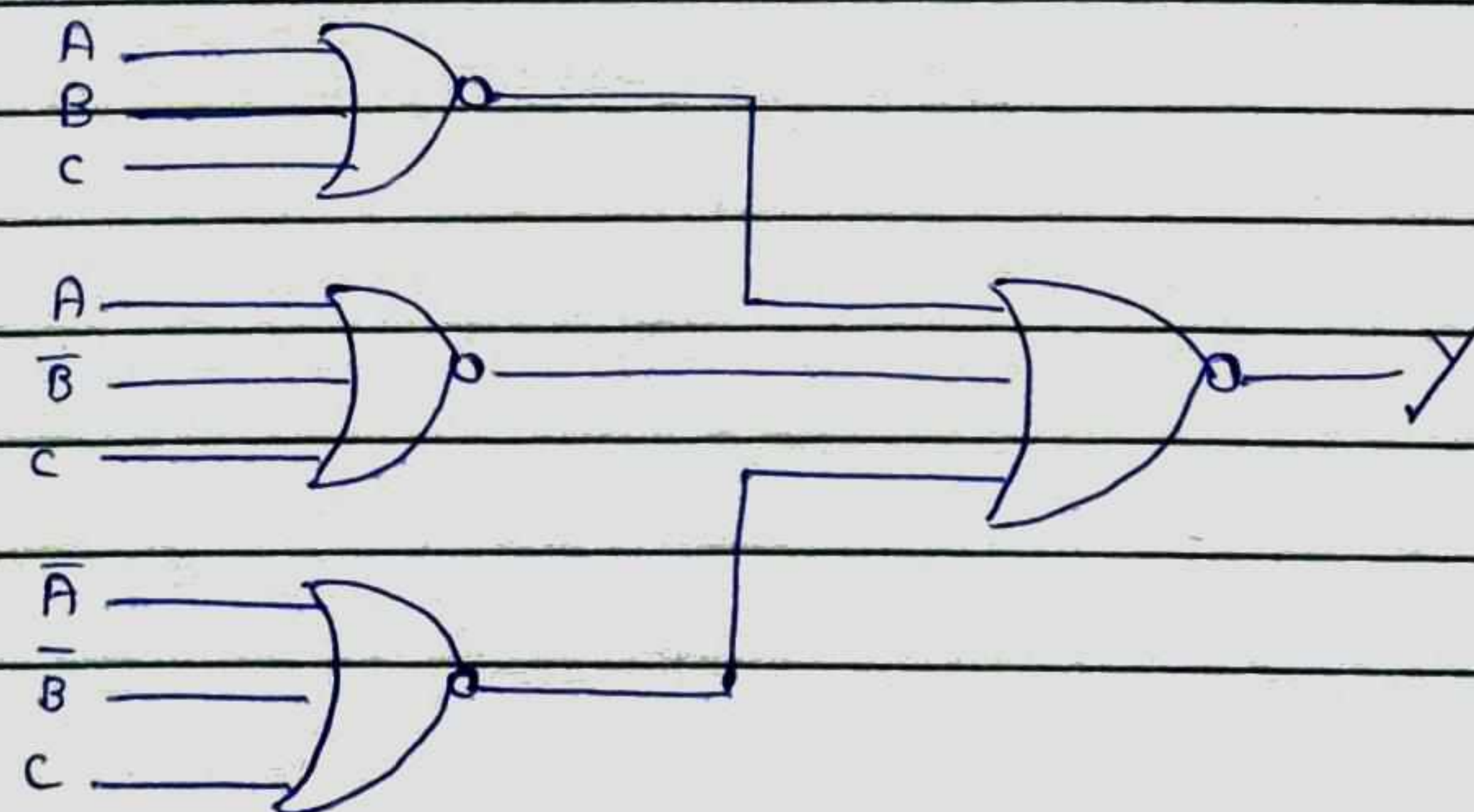
We can write the logic circuit by drawing OR-AND network or NOR-NOR network.

$$Y = (A+B+C)(A+\bar{B}+\bar{C})(\bar{A}+\bar{B}+C)$$

OR-AND:



NOR-NOR:



→ Note:

Conversion b/w SOP and POS.

$$1) Y = F(A, B, C) = \pi M(0, 3, 6) = \sum m(1, 2, 4, 5, 7).$$

$$2) Y = F(A, B, C) = \sum m(3, 5, 6, 7) = \pi M(0, 1, 2, 4)$$

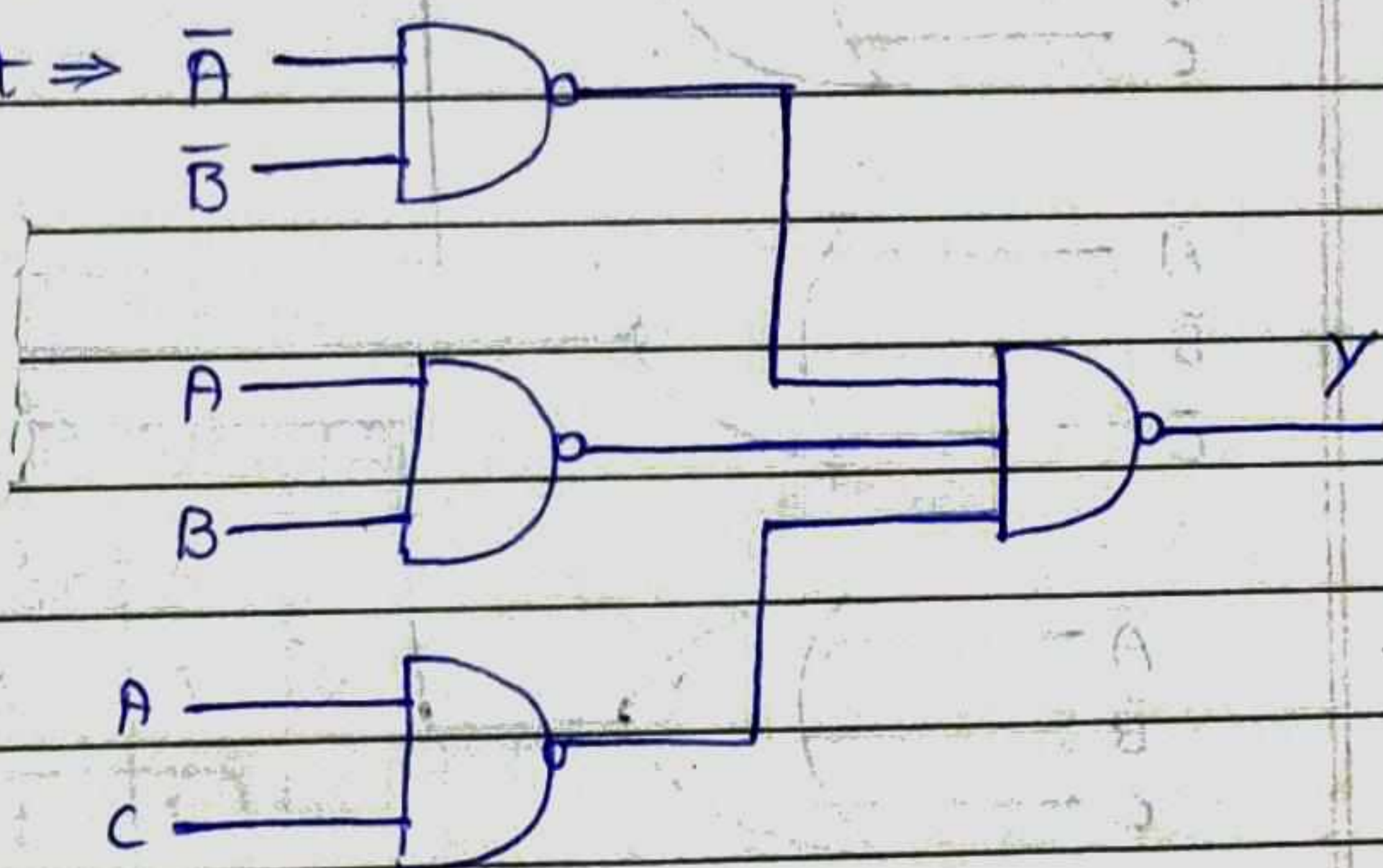
⇒ Consider a given SOP equation.

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

K-Map

AB \ CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	1	1	1
$\bar{A}B$	0	0	0	0
$A\bar{B}$	1	1	1	1
AB	0	0	1	1

NAND-NAND CKT ⇒



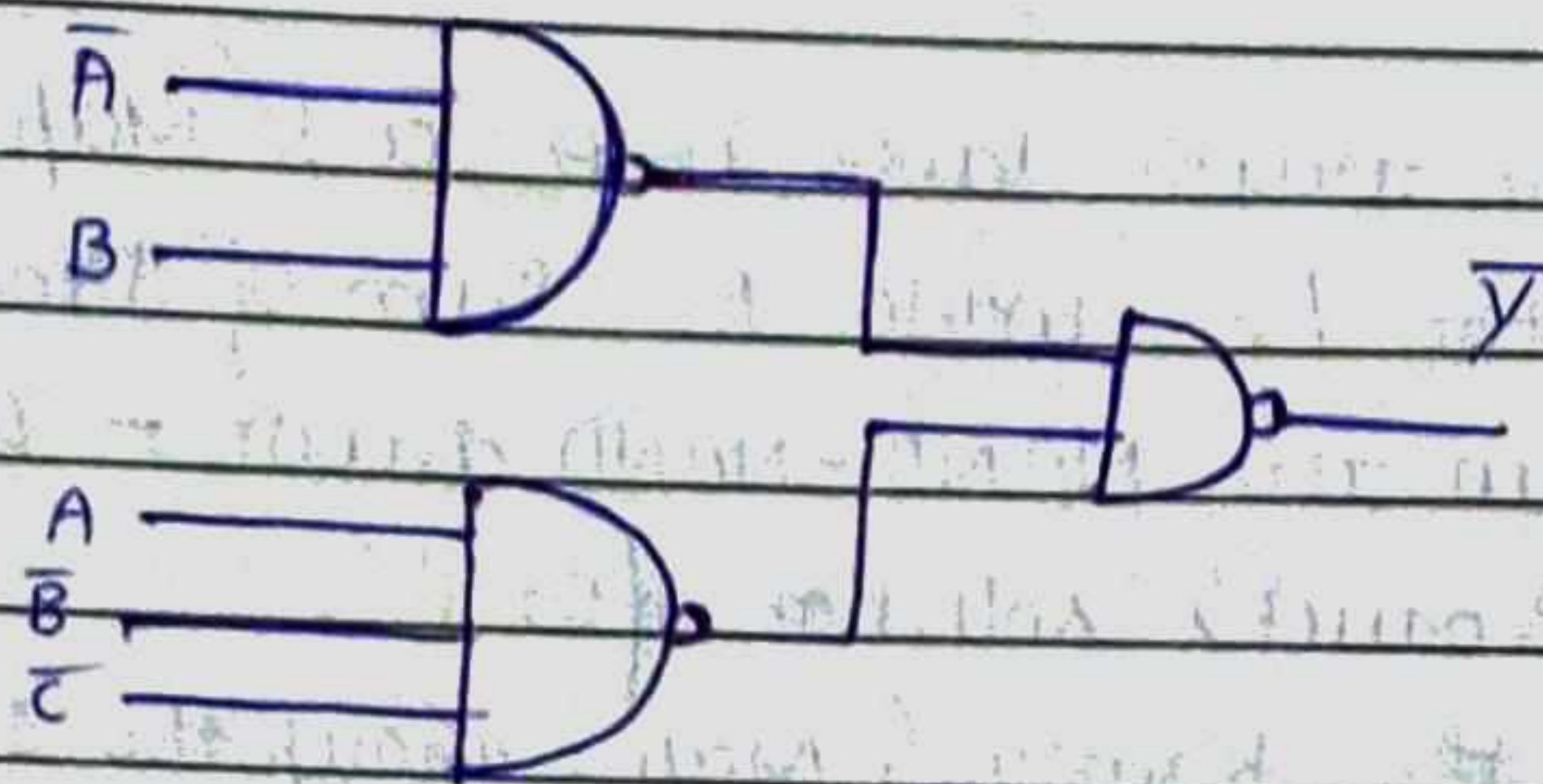
To get complementary (POS) circuit from NAND-NAND ckt.

→ complement each 0 and 1 on k-map.

AB CD

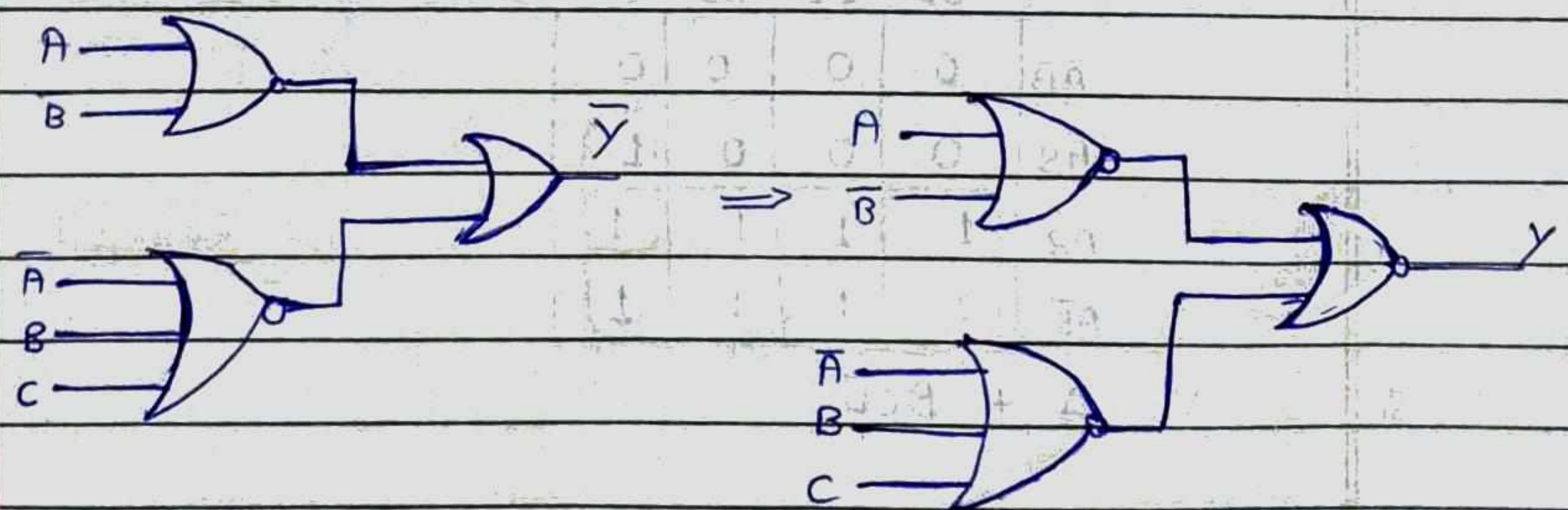
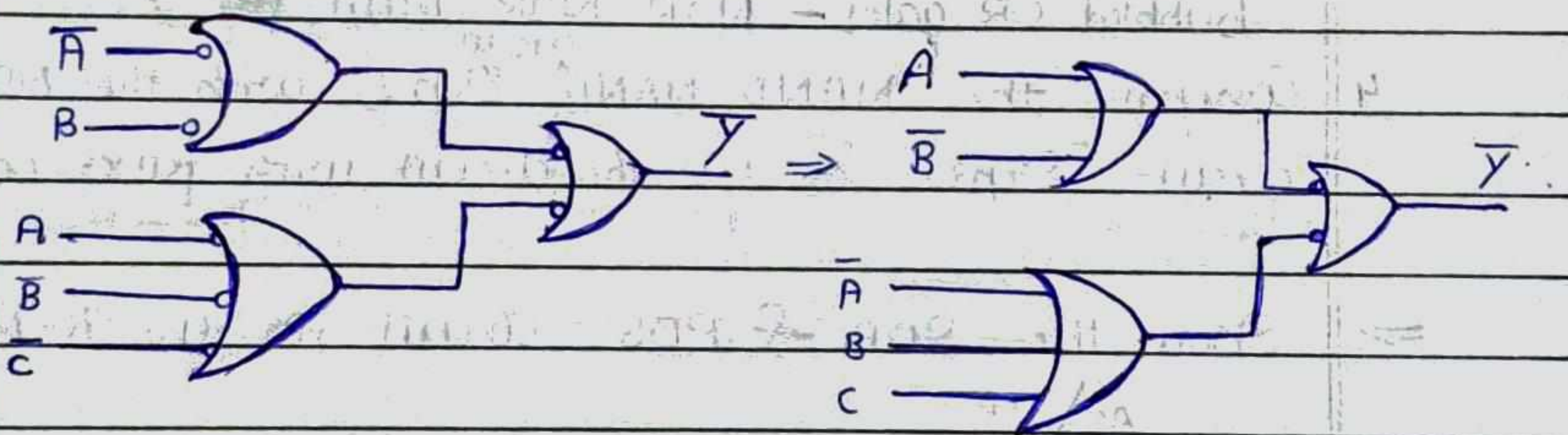
	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	1	1	1	1
AB	0	0	0	0
$A\bar{B}$	1	1	0	0

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



Deriving POS circuit.

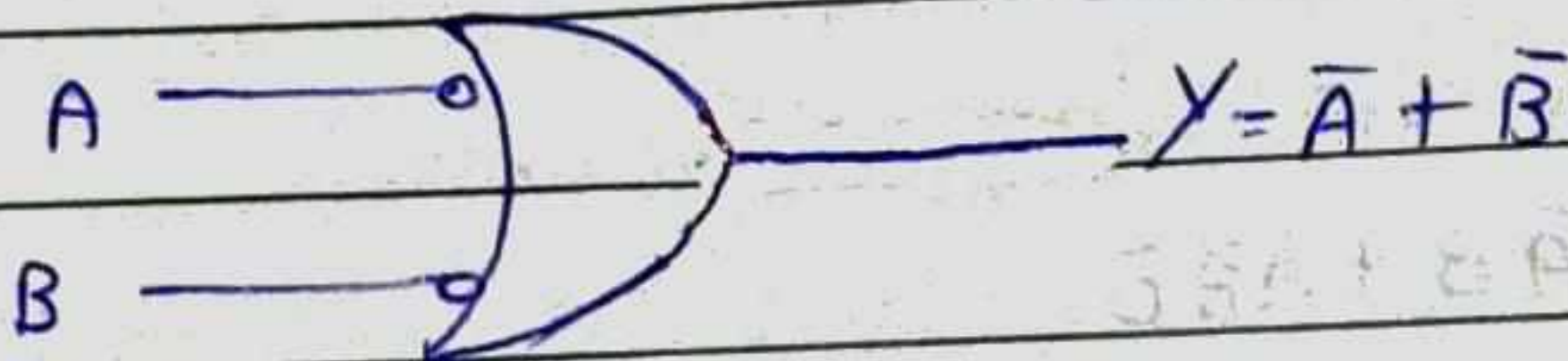
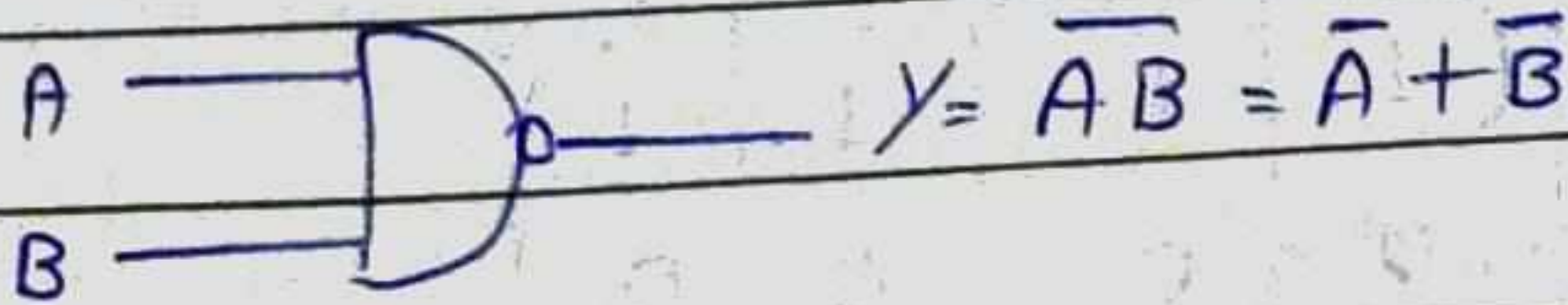
According to De Morgan's law NAND gates can be replaced by bubbled OR.



Note:

NAND Gate & Bubbled OR are same.

Ex:-



1. Convert the truth table into a K-Map. After grouping the 1's, write the Sum of Products eqⁿ & draw the NAND-NAND circuit - This is the sum of Product's solution for Y.
2. Complement the Karnaugh Map, group the 1's, write the SOP eqⁿ & draw the NAND-NAND circuit for \overline{Y} - this is the complementary NAND-NAND circuit.
3. Convert the complementary NAND-NAND circuit to NOR-NOR circuit by changing all NAND gates to bubbled OR gates - NOR-NOR circuit for Y.
4. Compare the NAND-NAND^{circuit} (Step 1) with the NOR-NOR circuit (Step 3) & use the circuit with fewer gates.

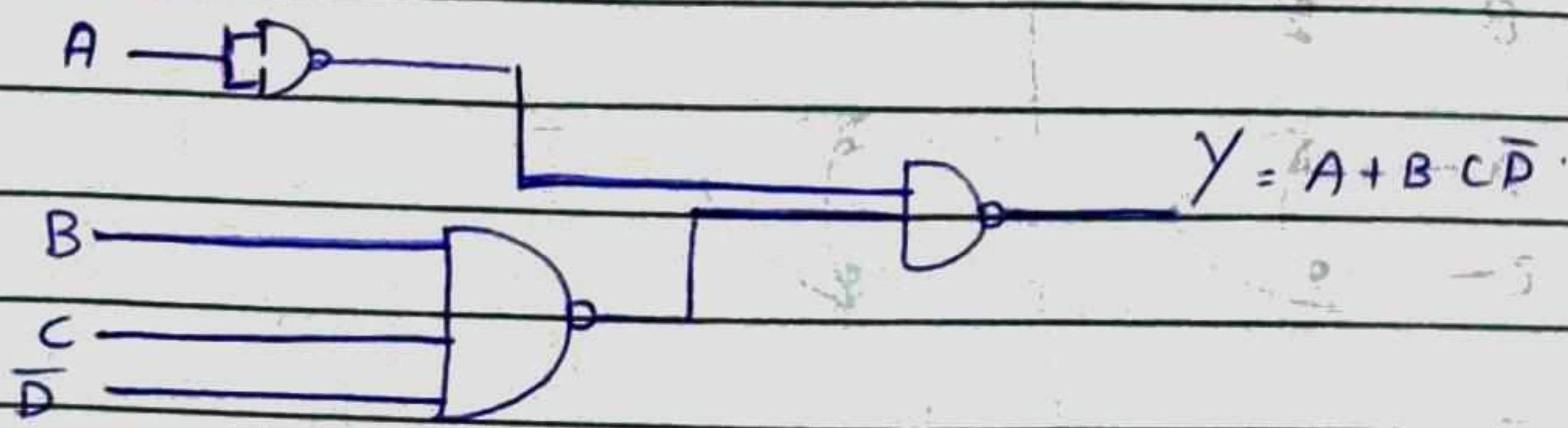
⇒ Show the SOP & POS circuit for the K-Map

AB \ CD				
	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\overline{B}$	0	0	0	0
$\overline{A}B$	0	0	0	1
AB	1	1	1	1
$A\overline{B}$	1	1	1	1

Sr

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

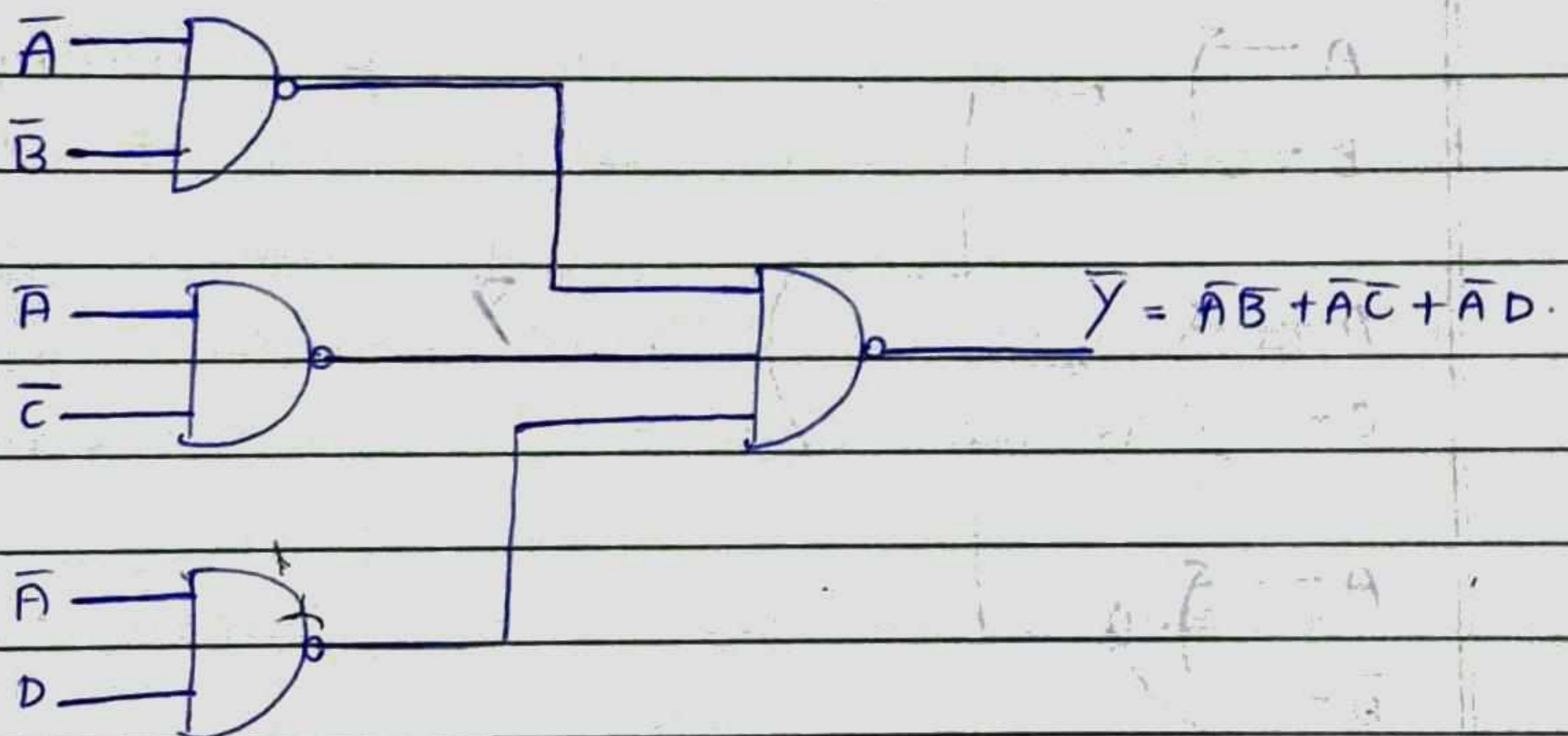
NAND-NAND ckt.



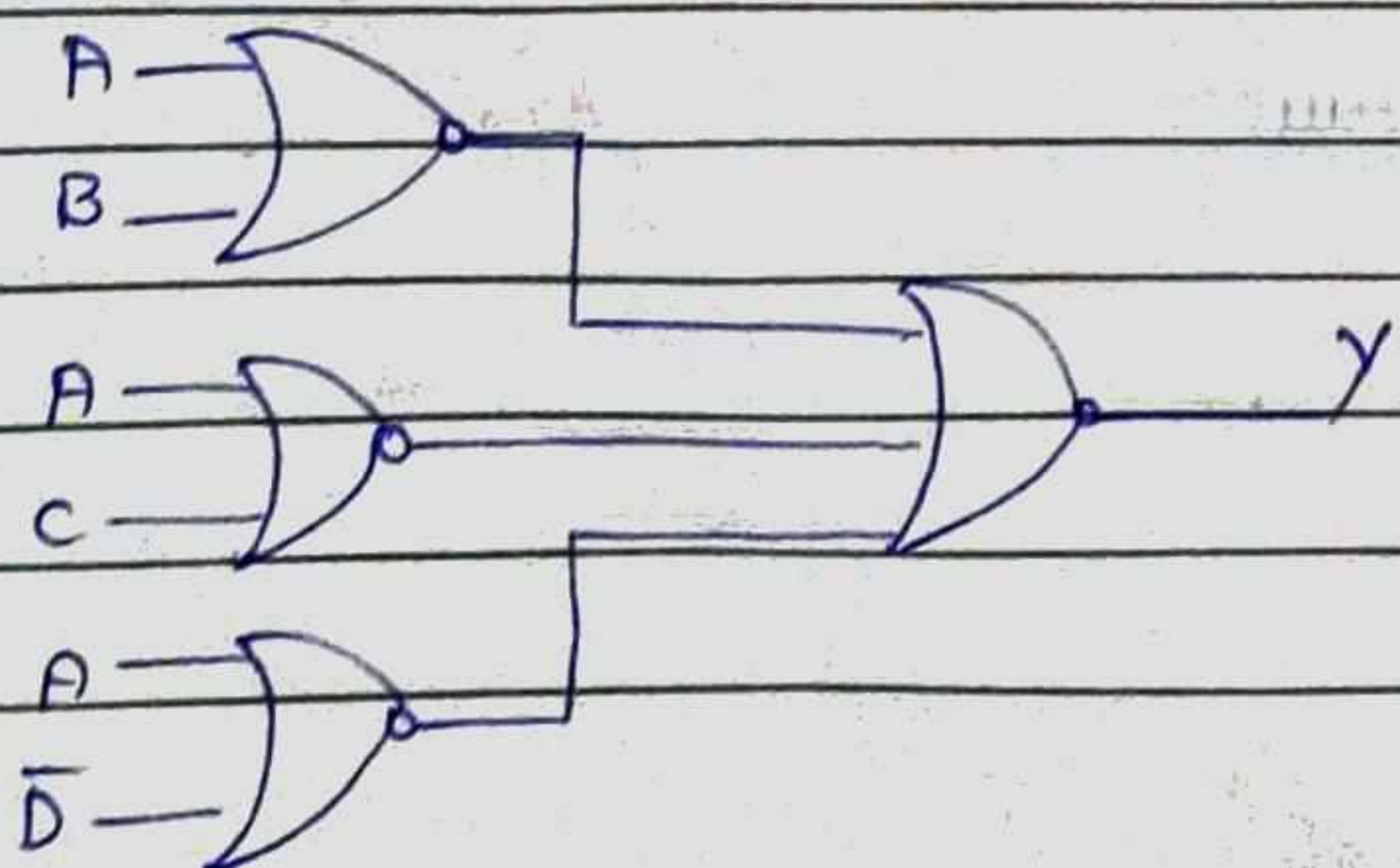
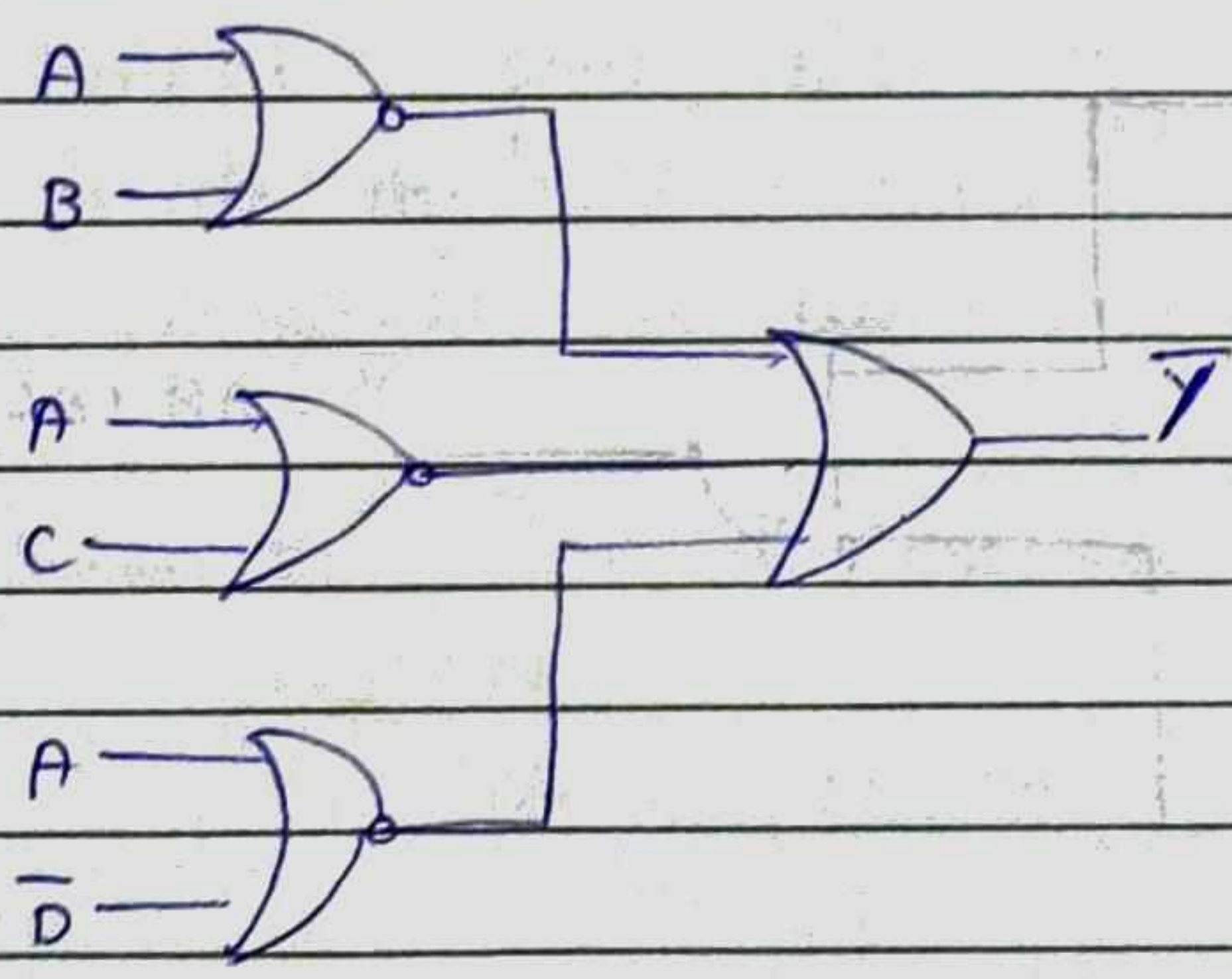
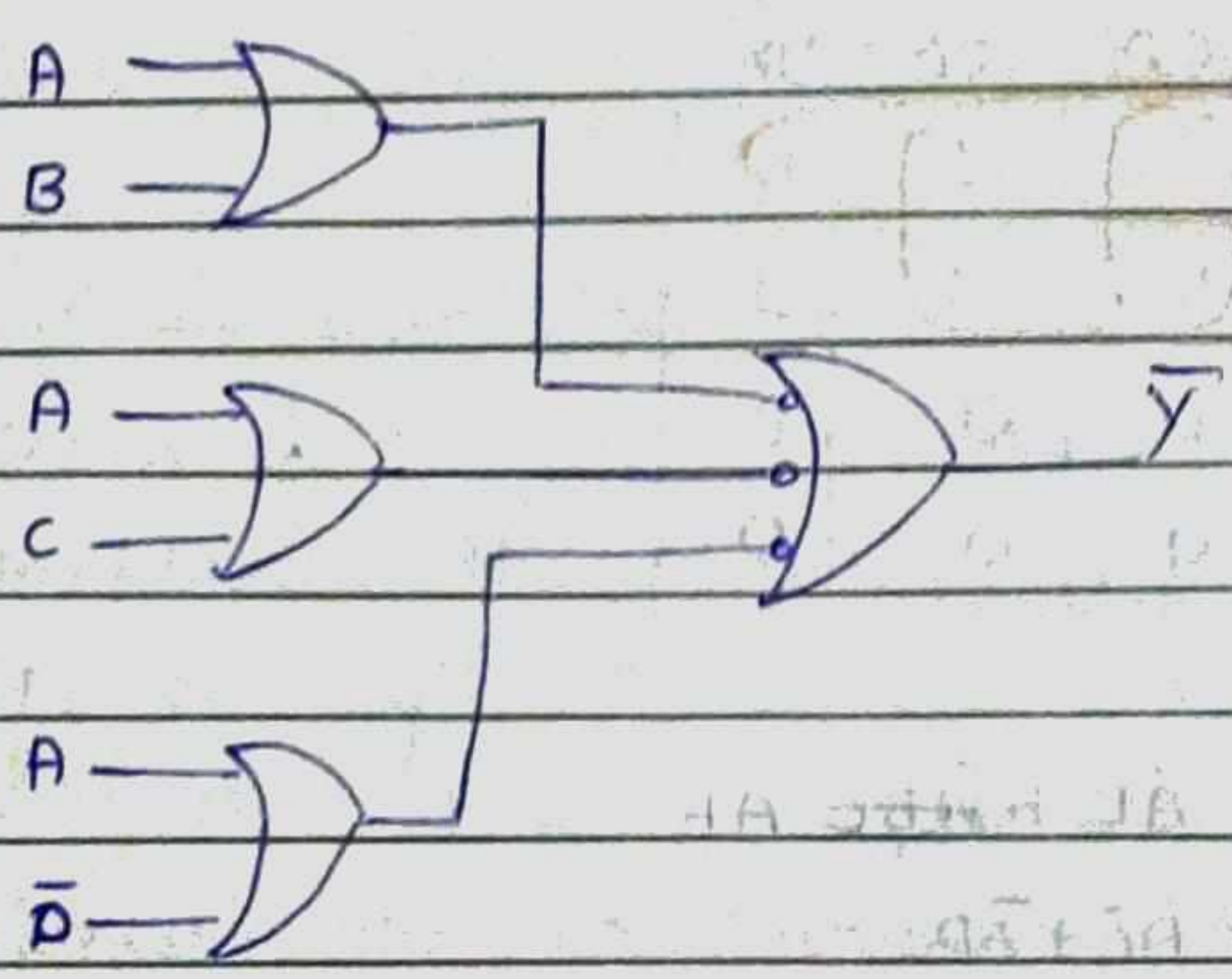
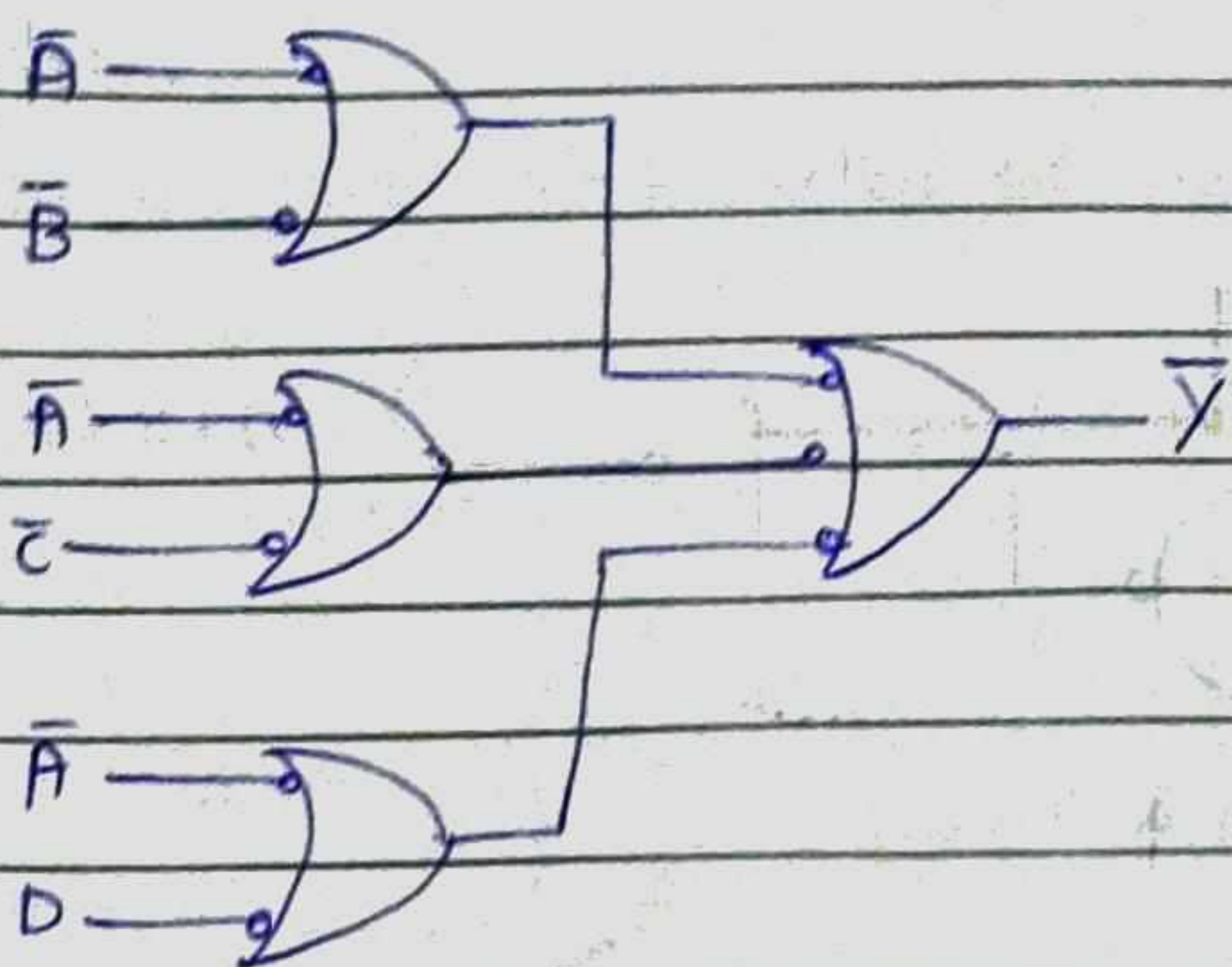
	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	1	1	1
$\bar{A}B$	1	1	1	0
$A\bar{B}$	0	0	0	0
AB	0	0	0	0

$$\bar{Y} = \bar{A}\bar{C} + \bar{A}D + \bar{A}\bar{B}$$

$$\bar{Y} = \bar{A}\bar{B} + \bar{A}\bar{C} + \bar{A}D$$



Deriving POS circuit.

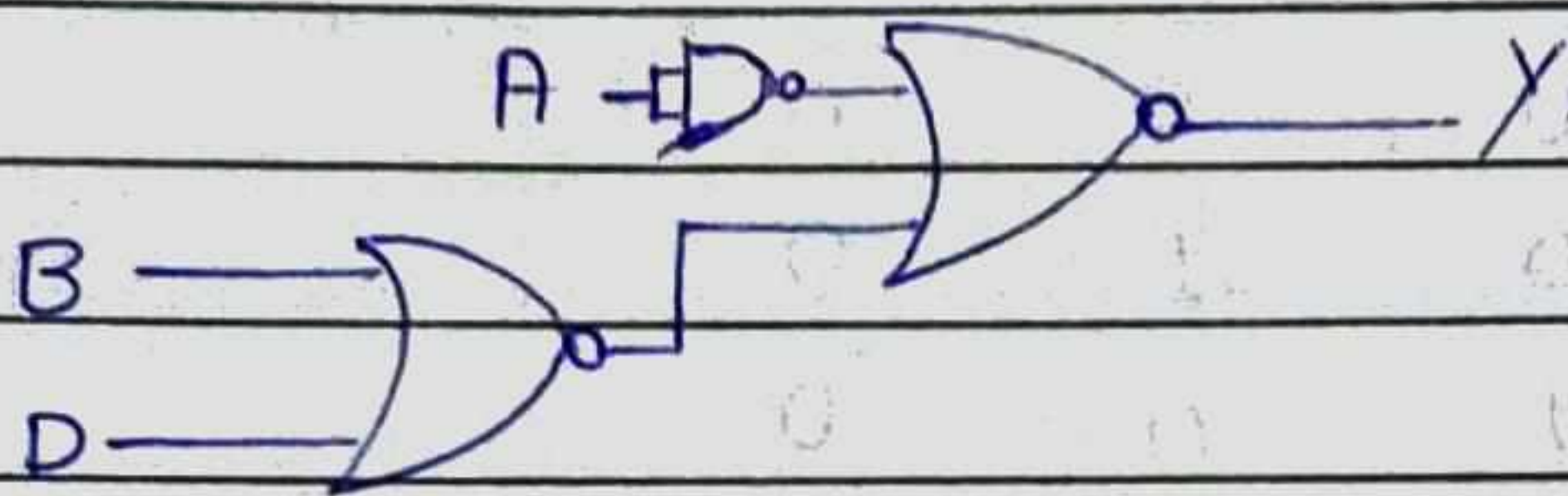


⇒ Give the simplest POS form of K-Map & also draw the logic circuit.

AB \ CD					
		$\bar{A}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	1	0	
$\bar{A}B$	0	0	1	1	
AB	X	X	X	1	
$A\bar{B}$	X	X	X	0	

$$Y = C(B + D)$$

NOR-NOR circuit.



⇒ Drawbacks of K-Map:

1. It depends on the user's ability to identify patterns that gives largest size.
2. This method becomes difficult to adapt for simplification of 5 or more variables.

To overcome from the above limitations we use Quine McClusky method.

This method involves preparation of two tables

1. Table that determines prime implicants.
2. Table that selects essential prime implicants to get minimal expression.

Prime implicants

These are expⁿ's with least no. of literals that represents all the terms given in a truth table.

Prime implicants are examined to get essential prime implicants for a particular expression that avoids any type of duplication.

Ex:

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
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

Determining prime implicants -

Stage 1 A B C D	Stage 2	Stage 3
0000 (0)✓	000- (0,1)✓	00-- (0,1,2,3)
0001 (1)✓	00-0 (0,2)✓	00-- (0,2,3)
0010 (2)✓	00-1 (1,3)✓	--01- (2,3,10,11)
0011 (3)✓	001- (2,3)✓	--01- (2,10,3,11)
1010 (10)✓	-010 (2,10)✓	
1100 (12)✓	-011 (3,11)✓	1-1- (10,11,14,15)
1011 (11)✓	101- (10,11)✓	1-1- (10,14,11,15)
1101 (13)✓	1-10 (10,14)✓	11-- (12,13,14,15)
1110 (14)✓	110- (12,13)✓	11-- (12,14,13,15)
	11-0 (12,14)✓	
1111 (15)✓	1-11 (11,15)✓	
	11-1 (13,15)✓	
	111- (14,15)✓	

Prime Implicants (members which are not ticked)

$$\bar{A}\bar{B}, \bar{B}C, AC, AB$$

Essential prime implicants

	0	1	2	3	10	11	12	13	14	15
$\bar{A}\bar{B}$ (0,1,2,3)	✓	✓	✓	✓						
$\bar{B}C$ (2,3,10,11)			✓	✓	✓	✓				
AC (10,11,14,15)					✓	✓			✓	✓
AB (12,13,14,15)							✓	✓	✓	✓

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

or

$$Y = \bar{A}\bar{B} + \bar{B}C + AB$$

(24)

Note:

AB \ CD				
	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1 0	1 1	1 3	1 2
$\bar{A}B$	4	5	7	6
AB	1 12	1 13	1 15	1 14
$A\bar{B}$	8	9	11	10

$$X = \bar{A}\bar{B}, AC, AB$$

$$Y = \bar{A}\bar{B}, \bar{B}C, AB$$

Question

$$1 \rightarrow F(A, B, C, D) = \sum m(1, 5, 6, 9, 10, 11)$$

$$2 \rightarrow \sum m(2, 3, 10, 11, 12, 13, 14, 15) + d(0, 1)$$

Answer.

1.	A	B	C	D	Y	
	0	0	0	0	0	0
	0	0	0	1	1	1
	0	0	1	0	0	2
	0	0	1	1	0	3
	0	1	0	0	0	4
	0	1	0	1	1	5
	0	1	1	0	1	6
	0	1	1	1	0	7
	1	0	0	0	0	8
	1	0	0	1	1	9
	1	0	1	0	1	10
	1	0	1	1	1	11
	1	1	0	0	0	12
	1	1	0	1	0	13
	1	1	1	0	0	14
	1	1	1	1	0	15

Verification

AB	CD	$\bar{C}\bar{D}$	$\bar{C}D$	$C\bar{D}$	CD
$\bar{A}\bar{B}$	0	1	0	0	0
$\bar{A}B$	1	0	1	0	0
$A\bar{B}$	0	0	0	1	0
AB	1	0	0	0	1

Stage 1

0001 (1) ✓

0101 (5) ✓

0110 (6) ✓

1001 (9) ✓

1010 (10) ✓

1011 (11) ✓

Stage 2

0_01 (1, 5)

_001 (1, 9)

10_1 (9, 11)

101_ (10, 11)

Prime Implicants

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

Essential prime implicants

	1	5	6	9	10	11
$\bar{A}B\bar{C}\bar{D}$ (6)			⊖			
$\bar{A}\bar{C}\bar{D}$ (1, 5)	⊙	⊙				
$\bar{B}\bar{C}\bar{D}$ (1, 9)	✓			✓		
$A\bar{B}D$ (9, 11)				✓		✓
$A\bar{B}C$ (10, 11)					⊙	⊙

$$Y = \bar{A}B\bar{C}\bar{D} + \bar{A}\bar{C}\bar{D} + \bar{B}\bar{C}\bar{D} + A\bar{B}C$$

or

$$Y = \bar{A}B\bar{C}\bar{D} + \bar{A}\bar{C}\bar{D} + A\bar{B}D + A\bar{B}C$$

$$2. \leq m(2, 3, 10, 11, 12, 13, 14, 15) + d(0, 1)$$

A	B	C	D	Y	
0	0	0	0	X	0
0	0	0	1	X	1
0	0	1	0	1	2
0	0	1	1	1	3
0	1	0	0	0	4
0	1	0	1	0	5
0	1	1	0	0	6
0	1	1	1	0	7
1	0	0	0	0	8
1	0	0	1	0	9
1	0	1	0	1	10
1	0	1	1	1	11
1	1	0	0	1	12
1	1	0	1	1	13
1	1	1	0	1	14
1	1	1	1	1	15

Stage 1.

0000 (0) ✓

0001 (1) ✓

0010 (2) ✓

0011 (3) ✓

1010 (10) ✓

1100 (12)

1011 (11) ✓

1101 (13) ✓

1110 (14) ✓

1111 (15) ✓

Stage 2

000- (0,1) ✓

00-0 (0,2) ✓

00-1 (1,3) ✓

001- (2,3) ✓

-010 (2,10) ✓

-011 (3,11) ✓

101- (10,11) ✓

1-10 (10,14) ✓

110- (12,13) ✓

11-0 (12,14) ✓

1-11 (11,15) ✓

11-1 (13,15) ✓

111- (14,15) ✓

Stage 3

00-- (0,1,2,3)

00-- (0,2,1,3)

-01- (2,3,10,11)

-01- (2,10,3,11)

1-1- (10,11,14,15)

1-1- (10,14,11,15)

11-- (12,13,14,15)

11-- (12,14,13,15)

The Prime implicants are.

$$\bar{A}\bar{B}, \bar{B}C, AC, AB$$

Essential Prime Implicants are

	2	3	10	11	12	13	14	15
$\bar{A}\bar{B} (0, 1, 2, 3)$	✓	✓						
$\bar{B}C (2, 3, 10, 11)$	✓	✓	✓	✓				
$AC (10, 11, 14, 15)$			✓	✓			✓	✓
$AB (12, 13, 14, 15)$					✓	✓	✓	✓

$$\therefore Y = AB + \bar{B}C$$

⇒ Entered variable Map.

Here

One of the input variable is placed inside Karnaugh Map.

It reduces the K-Map size by 1 degree.

i.e., a three variable problem that requires $2^3 = 8$

locations in K-Map will require $2^{(3-1)} = 4$ locations in entered variable map.

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

i) C as an entered variable.

A	B	Y
0	0	0
0	1	\bar{C}
1	0	0
1	1	1

A \ B	\bar{B}	B
\bar{A}	0	\bar{C}
A	0	1

$$Y = B\bar{C} + AB$$

ii) B as an entered variable.

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

A \ C	\bar{C}	C
\bar{A}	B	0
A	B	B

$$Y = B\bar{C} + AB$$

iii) A as an entered variable.

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

B \ C	\bar{C}	C
\bar{B}	0	0
B	1	A

$$Y = AB + B\bar{C}$$

The simplified eqⁿ for the given truth table without using entered variable map (EVM)

A \ BC	$\bar{B}\bar{C}$	$\bar{B}C$	$B\bar{C}$	BC
\bar{A}	0	0	0	1
A	0	0	1	1

$$Y = AB + B\bar{C}$$

AB \ C	\bar{C}	C
$\bar{A}\bar{B}$	0 ₀	0 ₁
$\bar{A}B$	1 ₂	0 ₃
AB	1 ₆	1 ₇
$A\bar{B}$	0 ₄	0 ₅

$$Y = B\bar{C} + AB$$

Note

eg

A \ B	\bar{B}	B
\bar{A}	0	\bar{C}
A	C	1

A	B	Y
0	0	0
0	1	\bar{C}
1	0	C
1	1	1

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

It doesn't need a separate coverage of 1 because we can write $1 = C + \bar{C}$, C is included in one group while \bar{C} in another group.

Note

In case of EVM

$$\begin{matrix} 0 & \text{or} & X \\ X & & 0 \end{matrix} \Rightarrow 0$$

$$\begin{matrix} 1 & \text{or} & X \\ X & & 1 \end{matrix} \Rightarrow 1$$

$$\begin{matrix} X \\ X \end{matrix} \Rightarrow X$$

⇒ Petrick's method

It is a technique for determining all minimum sum of products solution for a prime implicant chart.

As the no. of variables increases, the no. of prime implicants & the complexity of prime implicants chart may increase significantly. In such cases large ^{am} no. of trial & error may be required to find the min solution.

Petrick's method is the most systematic way of finding all min solⁿ.

Steps in Petrick's method.

1. Reduce the prime implicant chart by eliminating the essential prime implicant rows & the corresponding columns.
2. Label the rows of the reduced prime implicant chart P_1, P_2, P_3 etc.
3. Form a logical function P which is true when all columns are covered. P consist of a product of sum terms, each sum term having the form $(P_{i0} + P_{i1} + \dots)$ where P_{i0}, P_{i1} represents the rows which covers the column i .
4. Reduce P to a min sum of products by multiplying out & applying $X + XY = X$.
5. Each term in the result represents a solⁿ.
i.e., a set of rows which covers all of the minterms in the table. To determine the minimum solutions, find those terms which contain a min no. of variable. Each of these terms represents a solⁿ with a min no. of prime implicants.

$$\Rightarrow F = \sum m(0, 1, 2, 5, 6, 7)$$

$$0 \Rightarrow 000$$

$$2 \Rightarrow 010$$

$$6 \Rightarrow 110$$

$$1 \Rightarrow 001$$

$$5 \Rightarrow 101$$

$$7 \Rightarrow 111$$

Prime implicant table.

Stage 1	Stage 2	Prime implicants are
000 (0) ✓	00 - (0, 1)	$\bar{A}\bar{B}, \bar{A}\bar{C}, \bar{B}C, B\bar{C}, AC, AB$
001 (1) ✓	0 - 0 (0, 2)	
010 (2) ✓	- 01 (1, 5)	
101 (5) ✓	- 10 (2, 6)	
110 (6) ✓	1 - 1 (5, 7)	
111 (7) ✓	11 - (6, 7)	

Prime implicant chart

		0	1	2	5	6	7
P ₁	$\bar{A}\bar{B} (0, 1)$	X	X				
P ₂	$\bar{A}\bar{C} (0, 2)$	X		X			
P ₃	$\bar{B}C (1, 5)$		X		X		
P ₄	$B\bar{C} (2, 6)$			X		X	
P ₅	$AC (5, 7)$				X		X
P ₆	$AB (6, 7)$					X	X

$$\therefore \bar{A}\bar{B} + B\bar{C} + AC$$

or

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

Petrick's method to find minimum solutions.

1) 0 P₁ or P₂ must be true (1).

$$0 \Rightarrow (P_1 + P_2) \quad 1 \Rightarrow (P_1 + P_3) \quad 2 \Rightarrow (P_2 + P_4)$$

$$5 \Rightarrow (P_3 + P_5) \quad 6 \Rightarrow (P_4 + P_6) \quad 7 \Rightarrow (P_5 + P_6)$$

$$ii) P = (P_1 + P_2)(P_1 + P_3)(P_2 + P_4)(P_4 + P_6)(P_3 + P_5)(P_5 + P_6)$$

$$P = (P_1 + P_2 P_3)(P_4 + P_2 P_6)(P_5 + P_3 P_6)$$

$$P = (P_1 P_4 + P_1 P_2 P_6 + P_2 P_3 P_4 + P_2 P_3 P_6)(P_5 + P_3 P_6)$$

$$= P_1 P_4 P_5 + P_1 P_3 P_4 P_6 + P_1 P_2 P_5 P_6 + P_1 P_2 P_3 P_6 + P_2 P_3 P_4 P_5 +$$

$$P_2 P_3 P_4 P_6 + P_2 P_3 P_5 P_6 + P_2 P_3 P_6$$

$$= P_2 P_3 P_6 (1 + P_1 + P_4 + P_5) + P_1 P_4 P_5 + P_1 P_3 P_4 P_6 + P_1 P_2 P_5 P_6 + P_2 P_3 P_4 P_5$$

$$= P_2 P_3 P_6 + P_1 P_4 P_5 + P_1 P_3 P_4 P_6 + P_1 P_2 P_5 P_6 + P_2 P_3 P_4 P_5$$

$$= P_2 P_3 P_6, P_1 P_4 P_5$$

The min solution for the given problem is

$$P_2 P_3 P_6, P_1 P_4 P_5$$

$$i.e., \bar{A}\bar{C} + \bar{B}C + AB, \bar{A}\bar{B} + B\bar{C} + AC$$

\Rightarrow	0	1	2	3	10	11	12	13	14	15
$A'B' (0, 1, 2, 3)$	✓	✓	✓	✓						
$P_1 B'C (2, 3, 10, 11)$			✓	✓	✓	✓				
$P_2 AC (10, 11, 14, 15)$					✓	✓			✓	✓
$AB (12, 13, 14, 15)$							✓	✓	✓	✓

Essential prime implicants: $A'B'$, AB

$$10 \Rightarrow (P_1 + P_2)$$

$$11 \Rightarrow (P_1 + P_2)$$

$$P = (P_1 + P_2)(P_1 + P_2)$$

$$P = P_1 + P_2$$

$$P_1 \text{ or } P_2$$

$$A'B', AB, B'C \Rightarrow A'B' + AB + B'C$$

or

$$A'B', AB, AC \Rightarrow A'B' + AB + AC$$

$$b'c'd' + a'bd$$

(34)

\Rightarrow		0	1	2	5	6	7	8	9	10	14
	$b'c' (0, 1, 8, 9)$	✓	✓					✓	✓		
P_1	$b'd' (0, 2, 8, 10)$	✓		✓				✓		✓	
	$cd' (2, 6, 10, 14)$			✓		✓				✓	✓
P_2	$a'c'd (1, 5)$		✓		✓						
P_3	$a'bd (5, 7)$				✓		✓				
P_4	$a'bc (6, 7)$					✓	✓				

$$5 \Rightarrow (P_2 + P_3)$$

$$7 \Rightarrow (P_3 + P_4)$$

$$P = (P_2 + P_3)(P_3 + P_4)$$

$$(x + y)(x + z)$$

$$= (P_3 + P_2P_4)$$

$$x + yz$$

$$a'bd$$

$$= P_3$$

(min no. of terms)

Essential prime implicants are

$$b'c', cd', a'bd$$

\therefore The simplified eqⁿ is $b'c' + cd' + a'bd$.

①

⇒ Multiplexer (data selector / MUX):

Applⁿ: Telecommunication
It has a group of data inputs & a group of control inputs. The control inputs are used to select one of the data inputs & connect it to the output terminal.

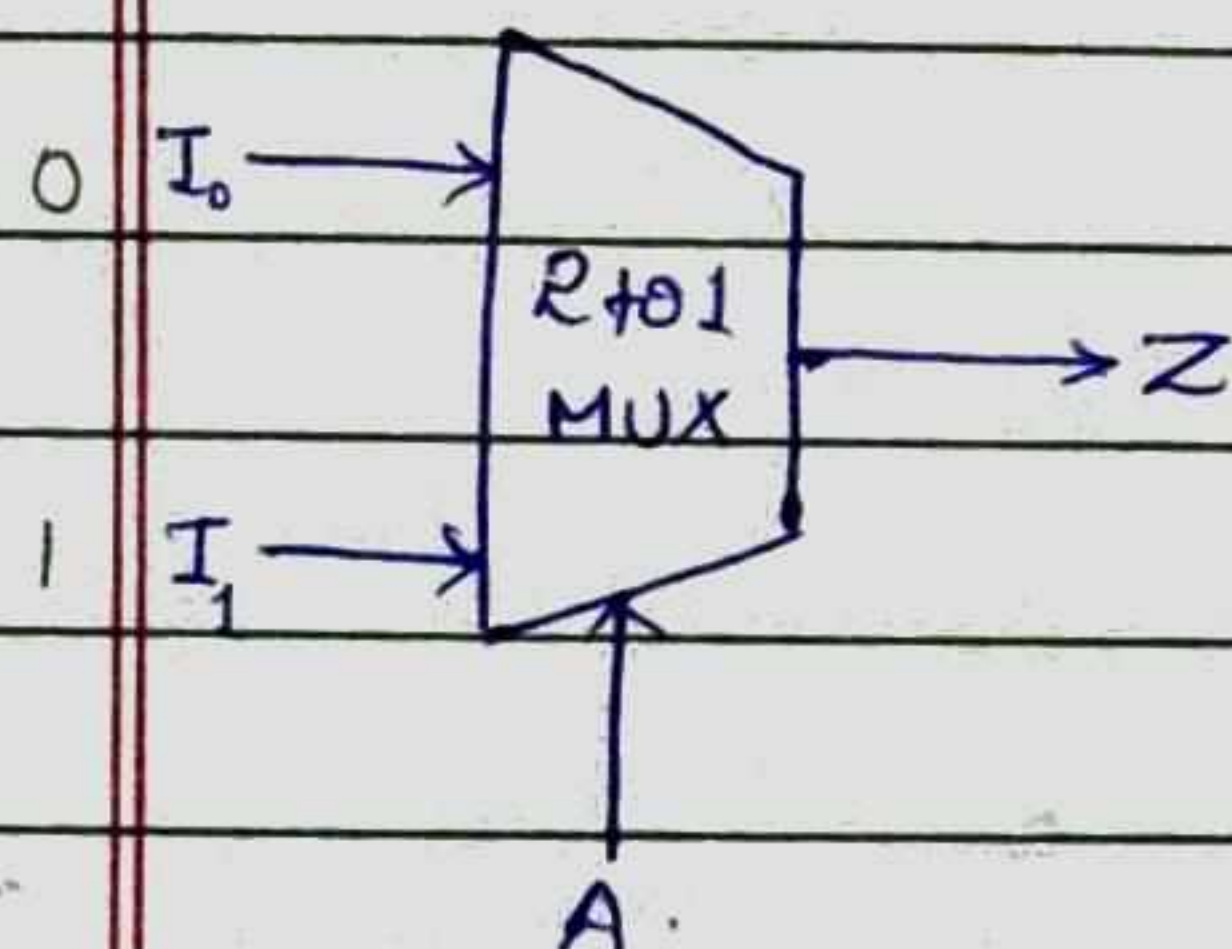
→ 2 to 1 MUX

2 → logical or data inputs

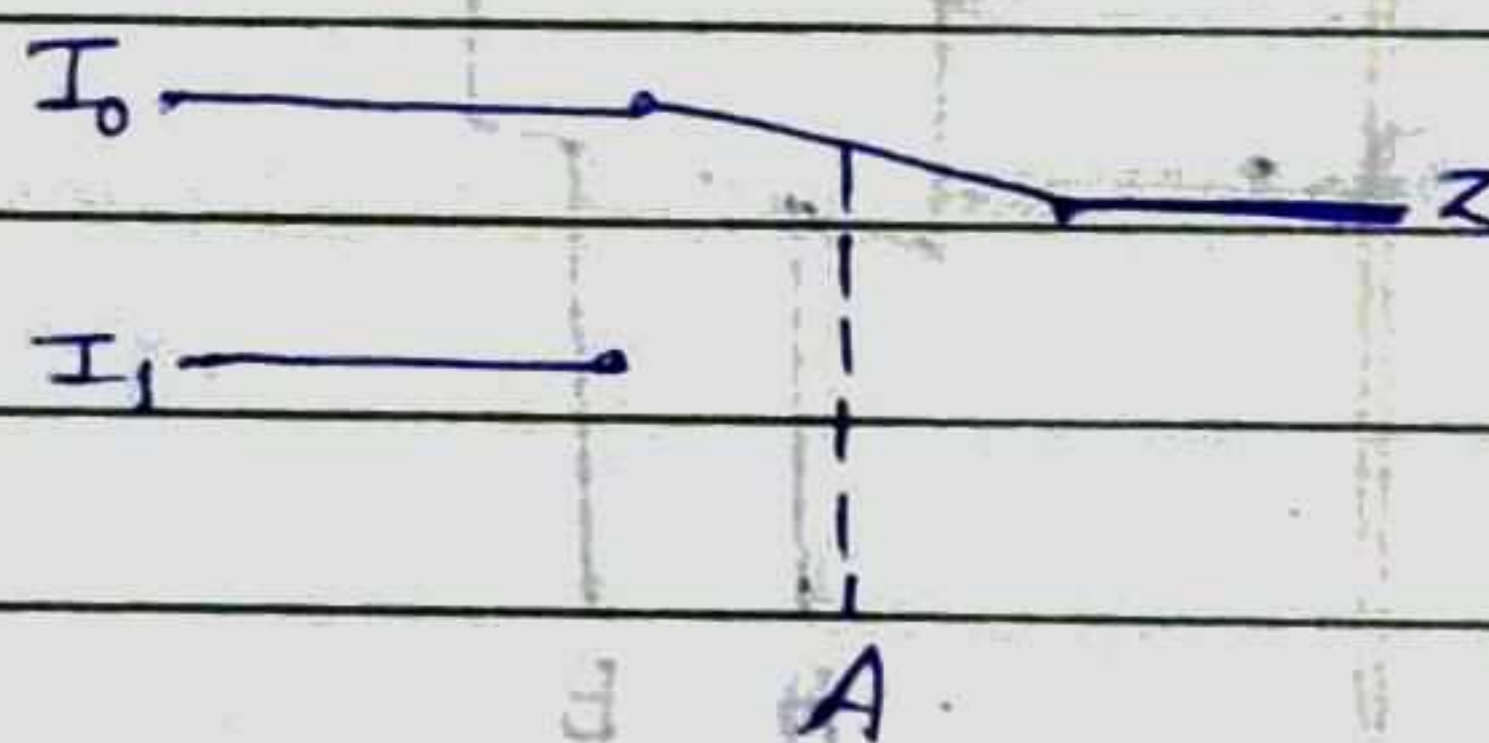
1 → output

$$2^n = 2, n = 1 \text{ control input}$$

Symbol.



Switch analog

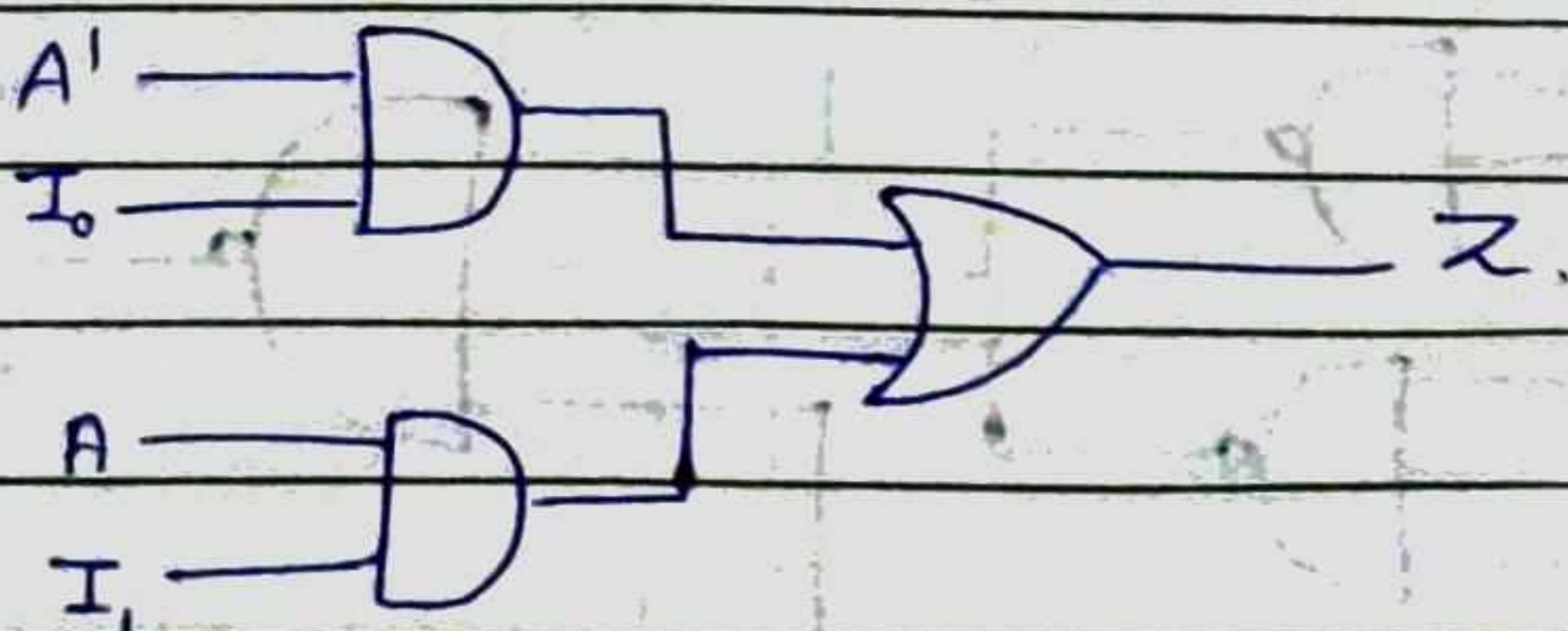


Logic equation

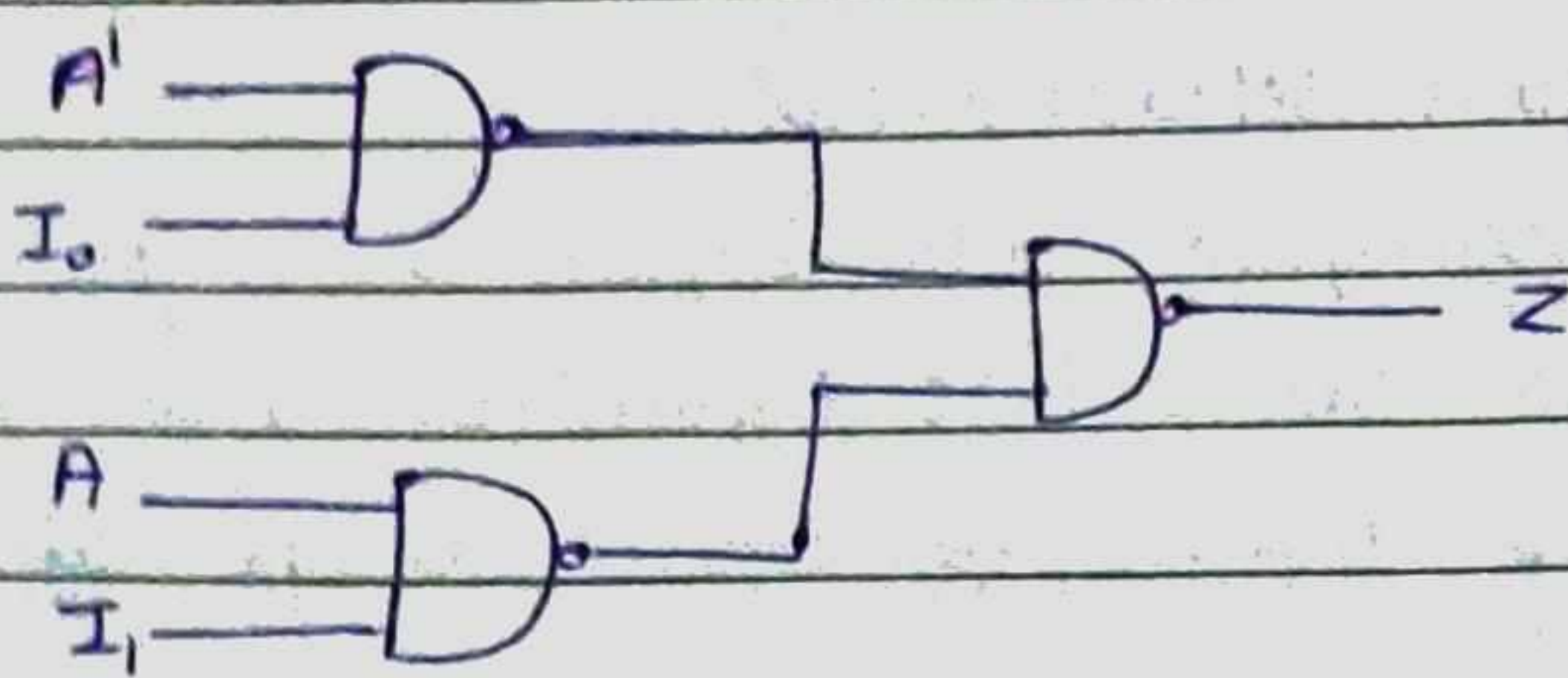
$$Z = A' I_0 + A I_1$$

Logic diagram (Implementation)

AND-OR:



NAND-NAND

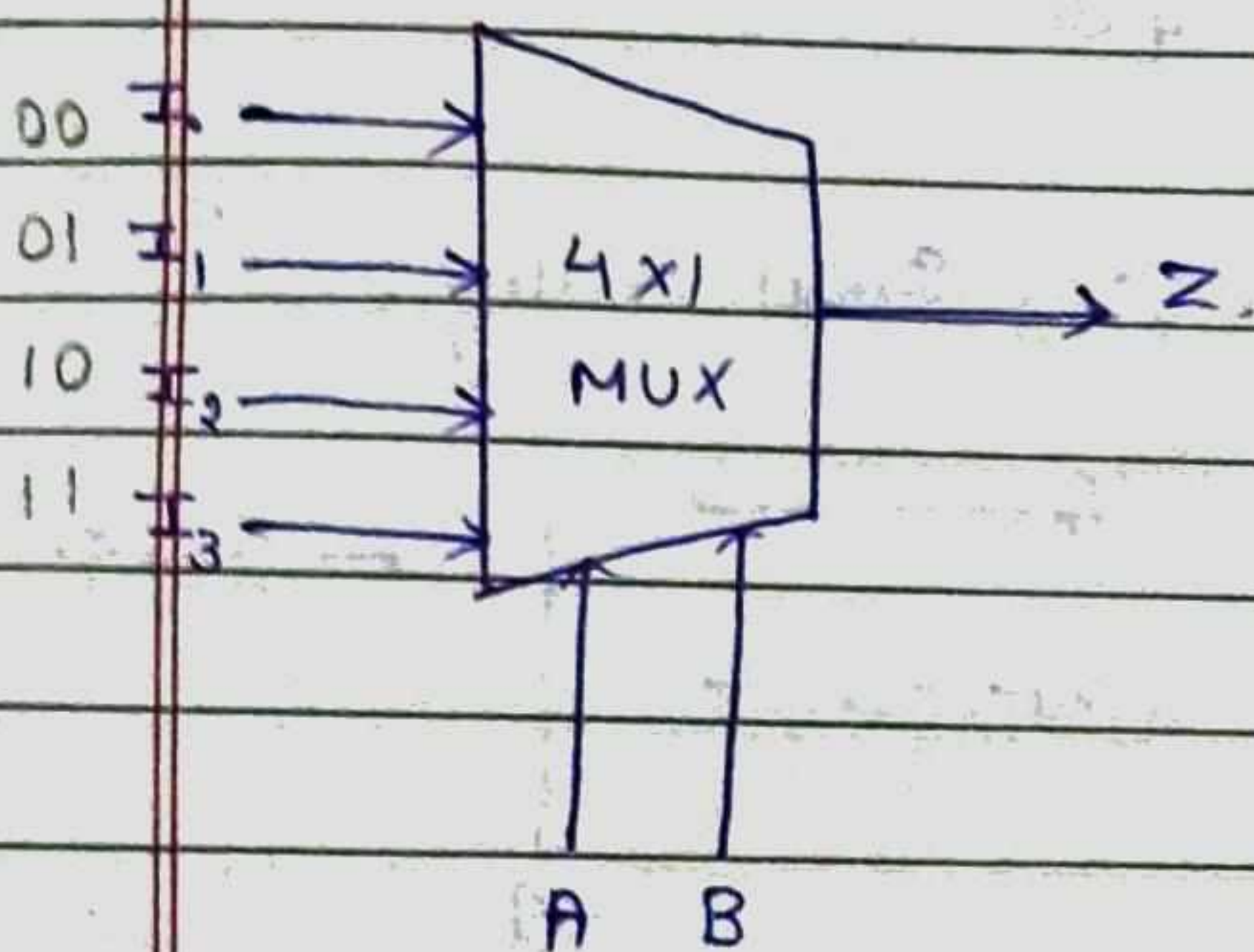


→ 4 to 1 MUX

4 → logical or data input

1 → output

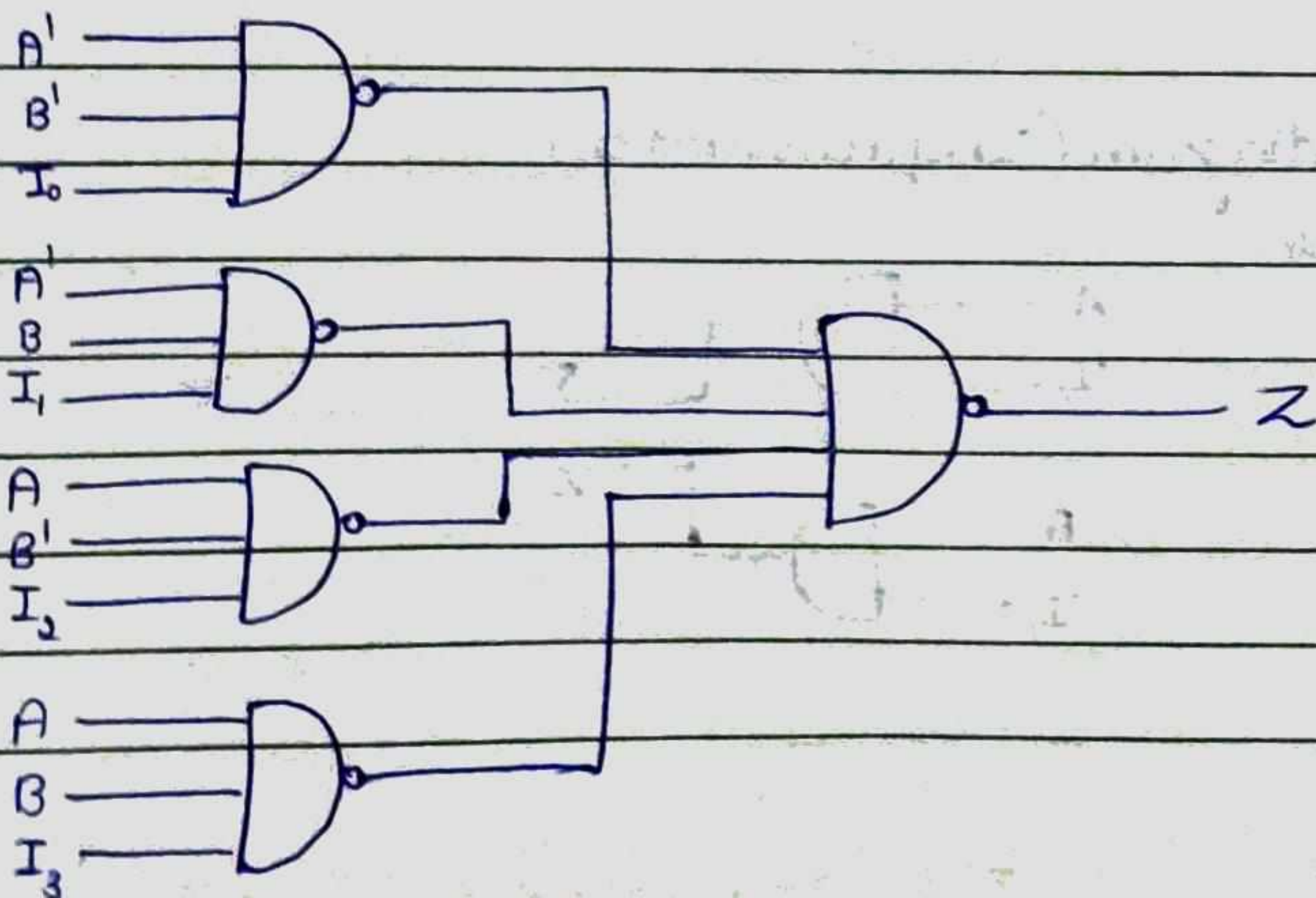
$2^n = 4$, $n = 2$ control input



Logic eqⁿ

$$Z = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3$$

NAND-NAND

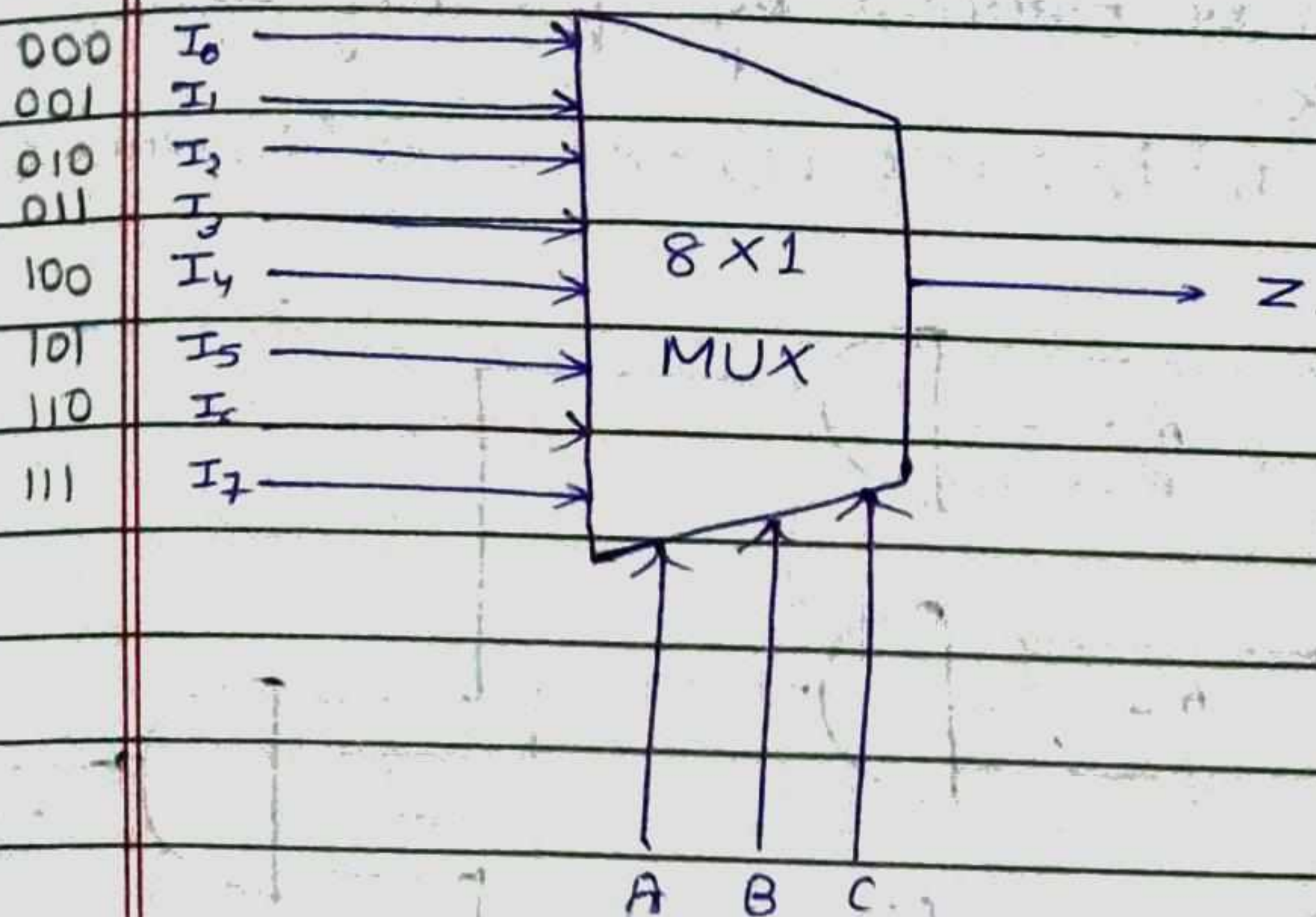


→ 8 to 1 MUX.

8 → logical or data input.

1 → output.

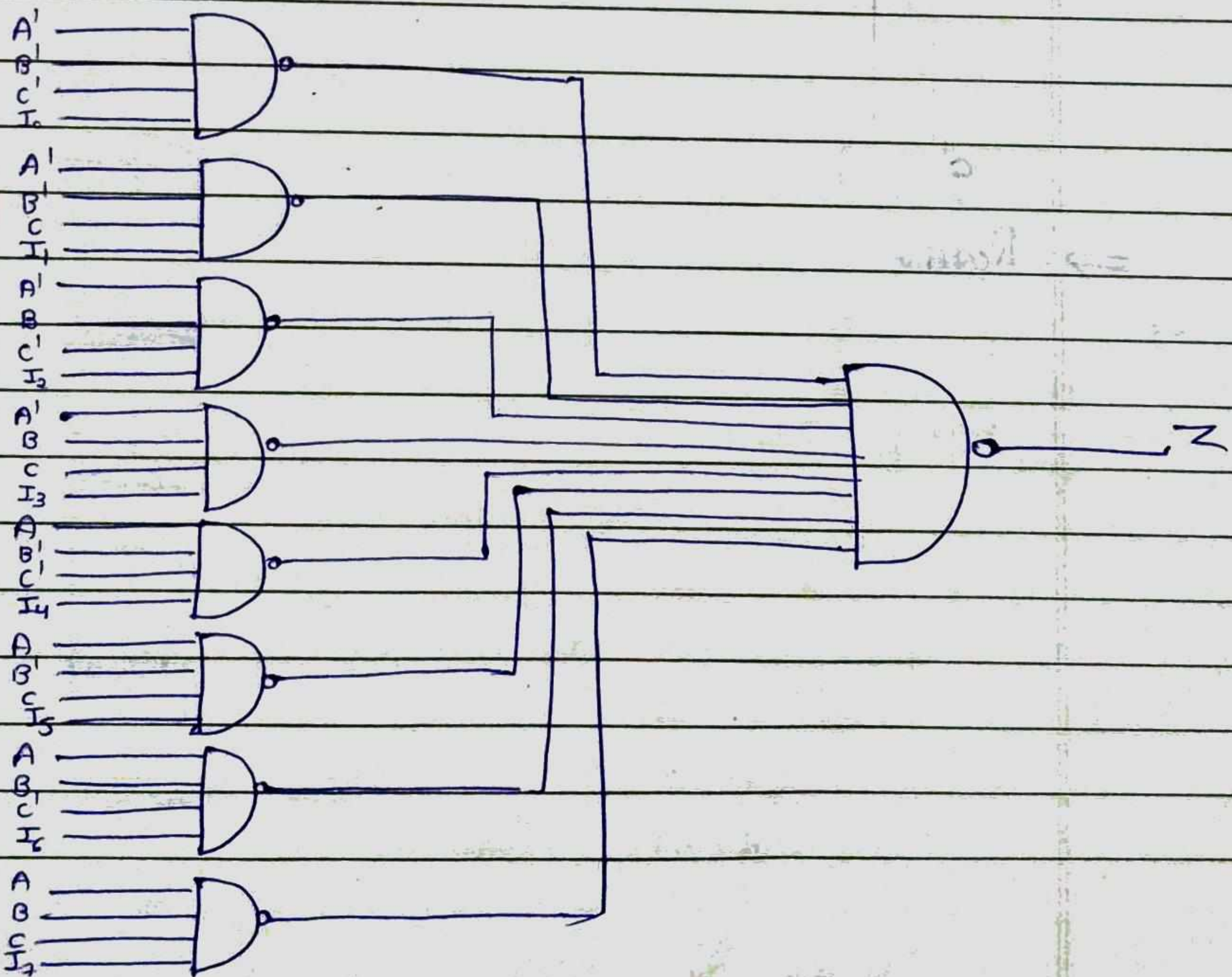
$2^n = 8, n=3$, control input.



Logic equation

$$Z = A'B'C'I_0 + A'B'CI_1 + A'BC'I_2 + A'BCI_3 + AB'C'I_4 + AB'CI_5 + ABC'I_6 + ABCI_7.$$

NAND - NAND.

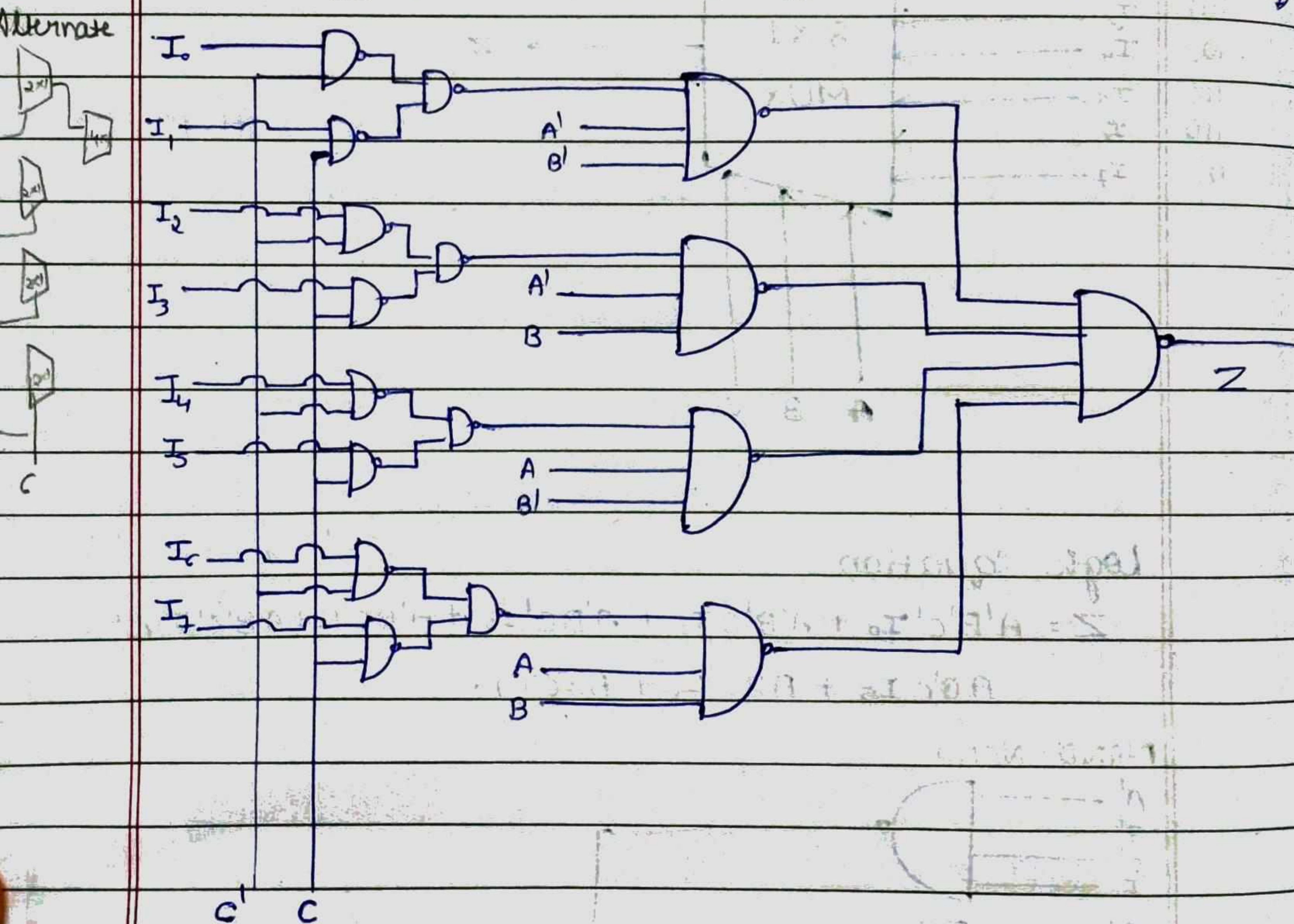


(4)

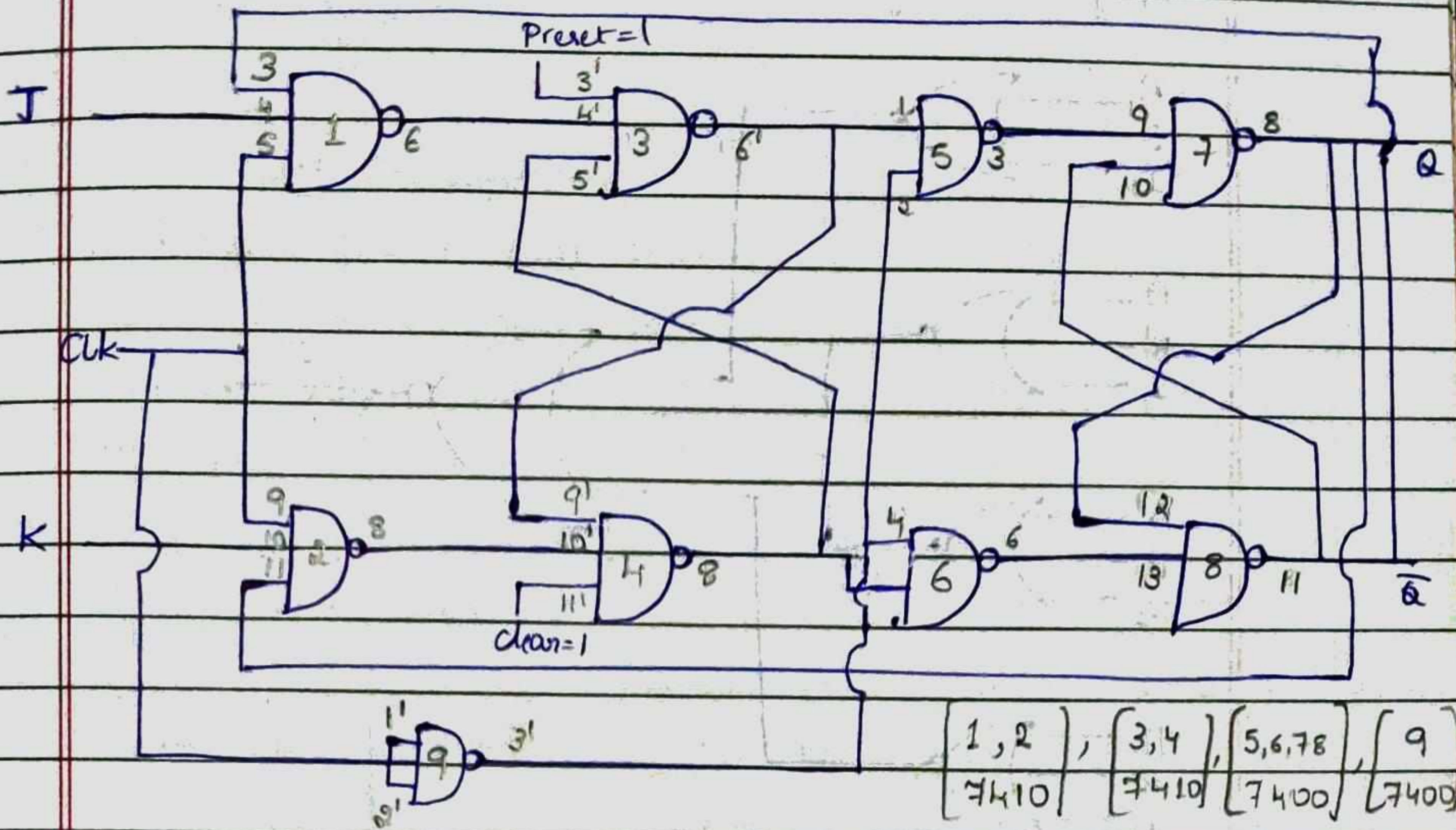
It is a 2 level implementation for 8x1 multiplexer

WV1 → Implementation with more than two levels by of gates can be obtained by factoring the equation of Z .

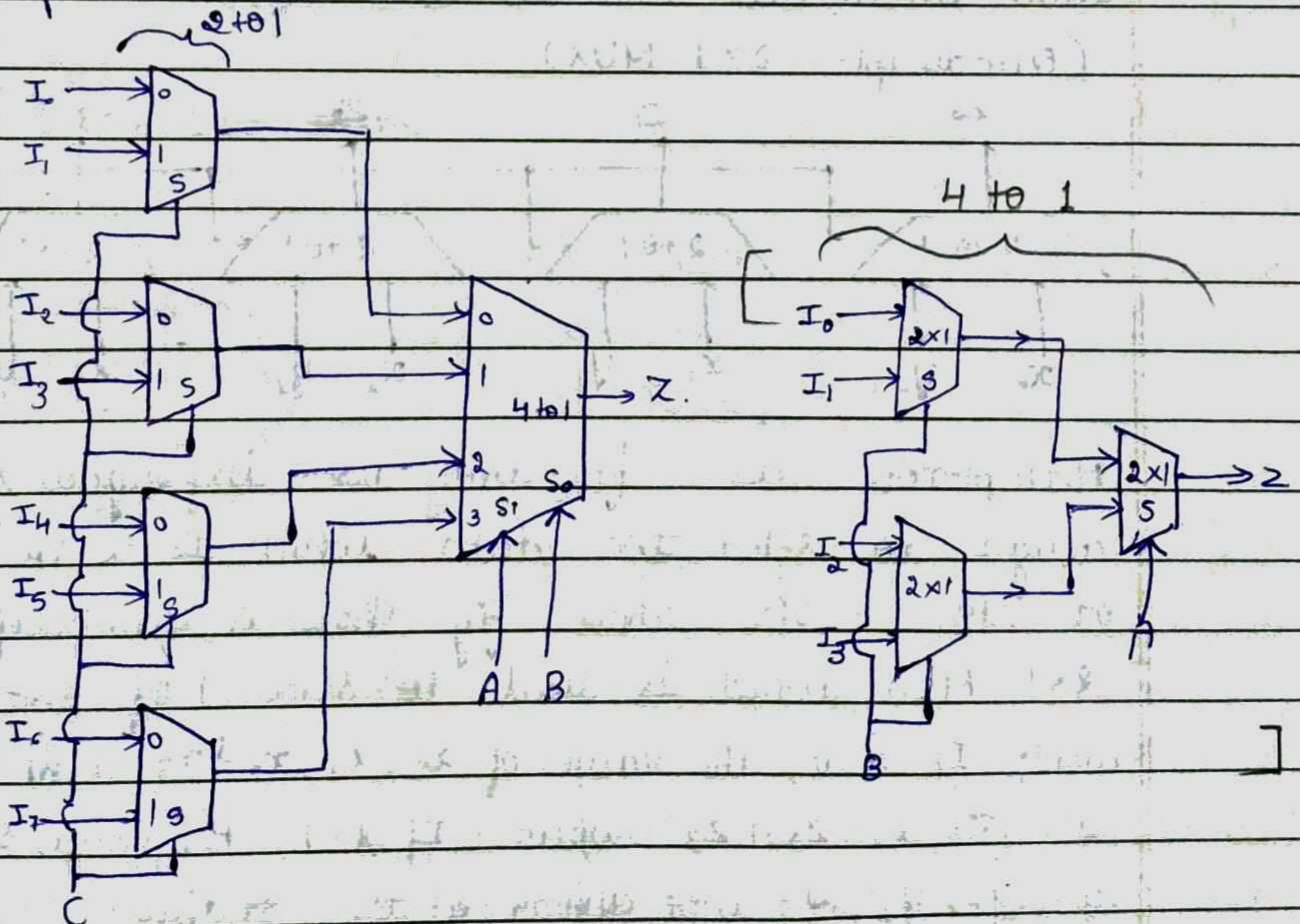
$$Z = AB'(C'I_0 + CI_1) + A'B(C'I_2 + CI_3) + AB'(C'I_4 + CI_5) + AB(C'I_6 + CI_7)$$



⇒ Realise a JK Master/Slave Flip Flop using NAND Gates and verify its truth table.



⇒ Component MUX's of multi level implementation of an 8 to 1 MUX.

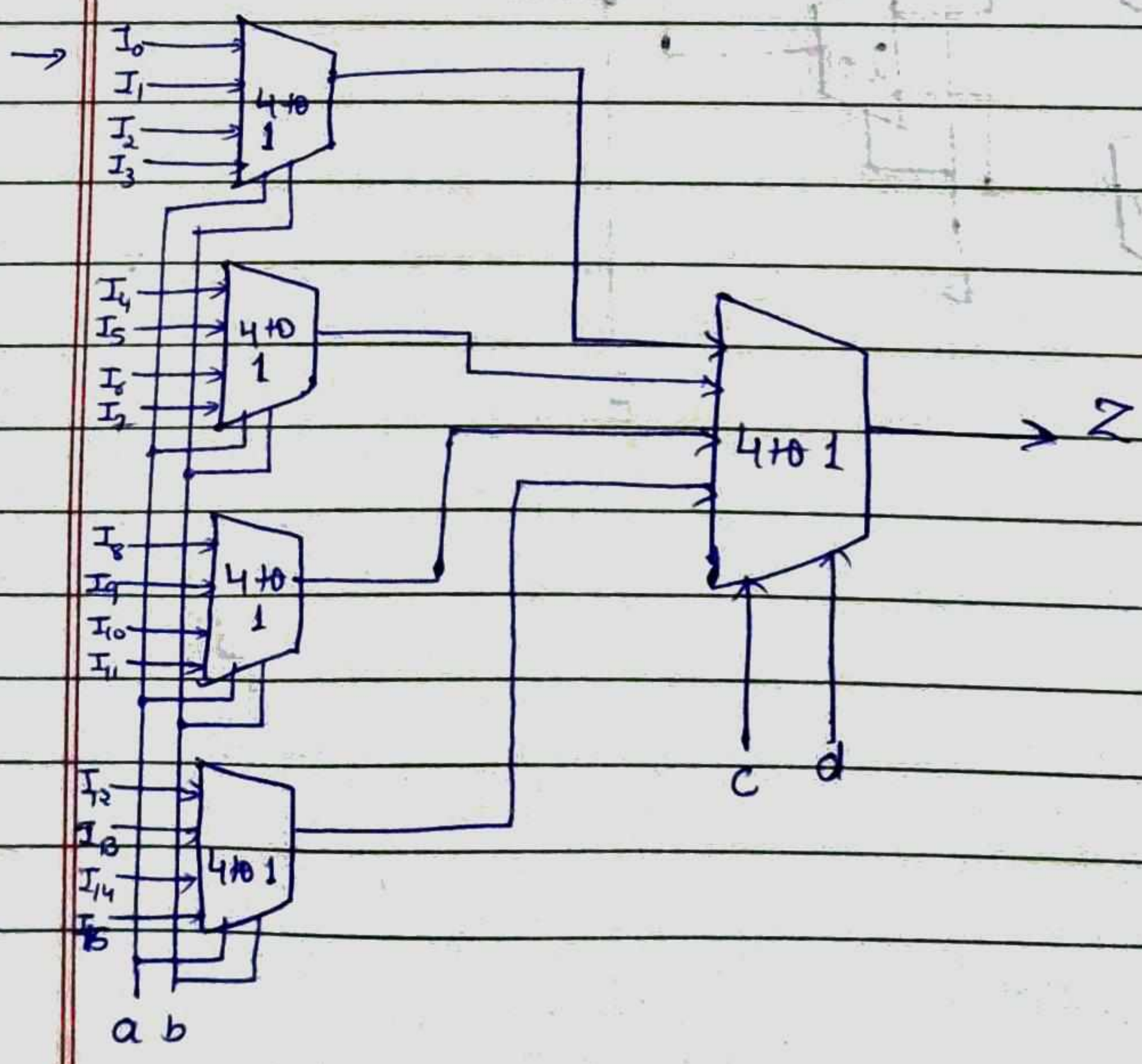
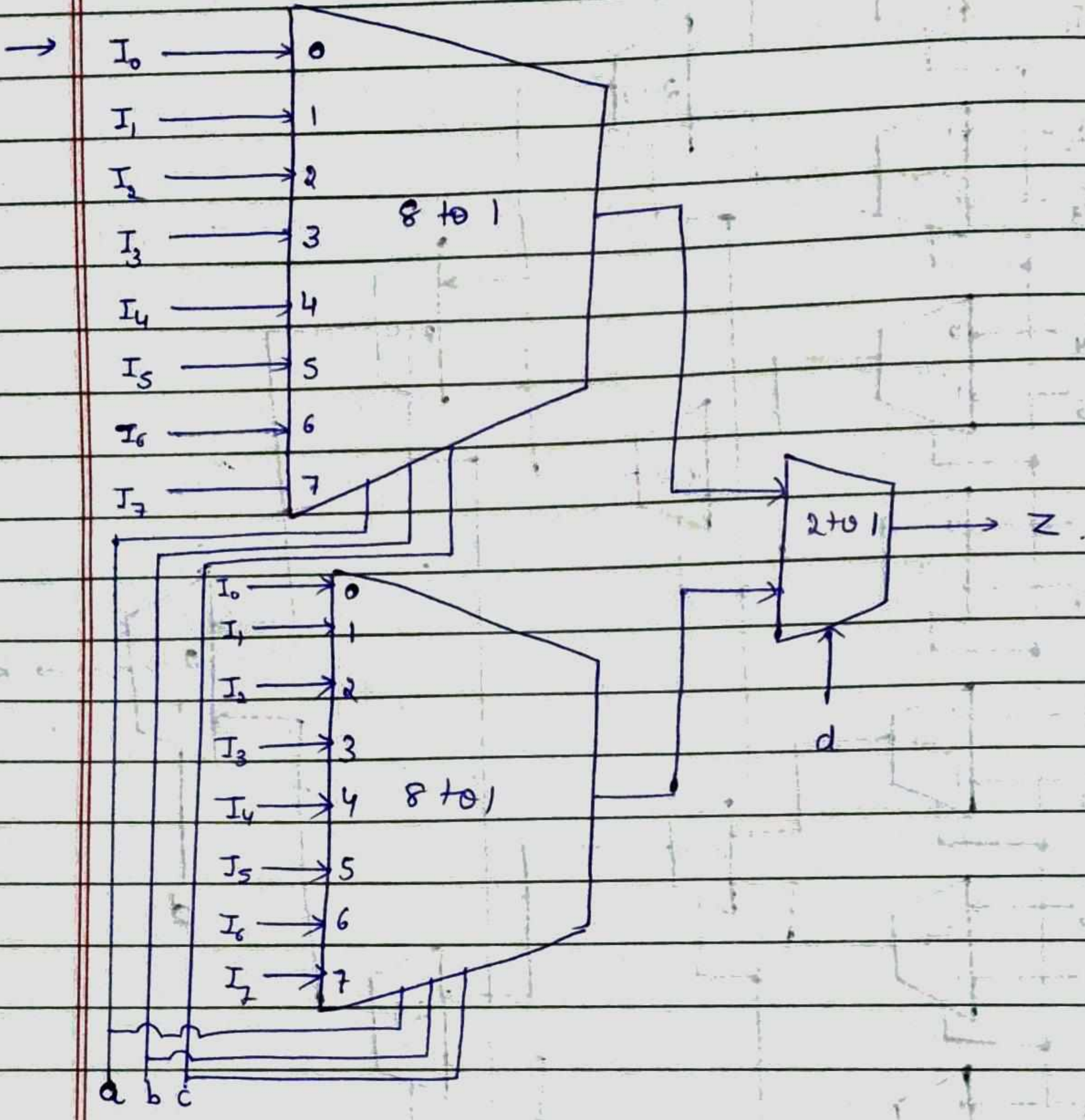


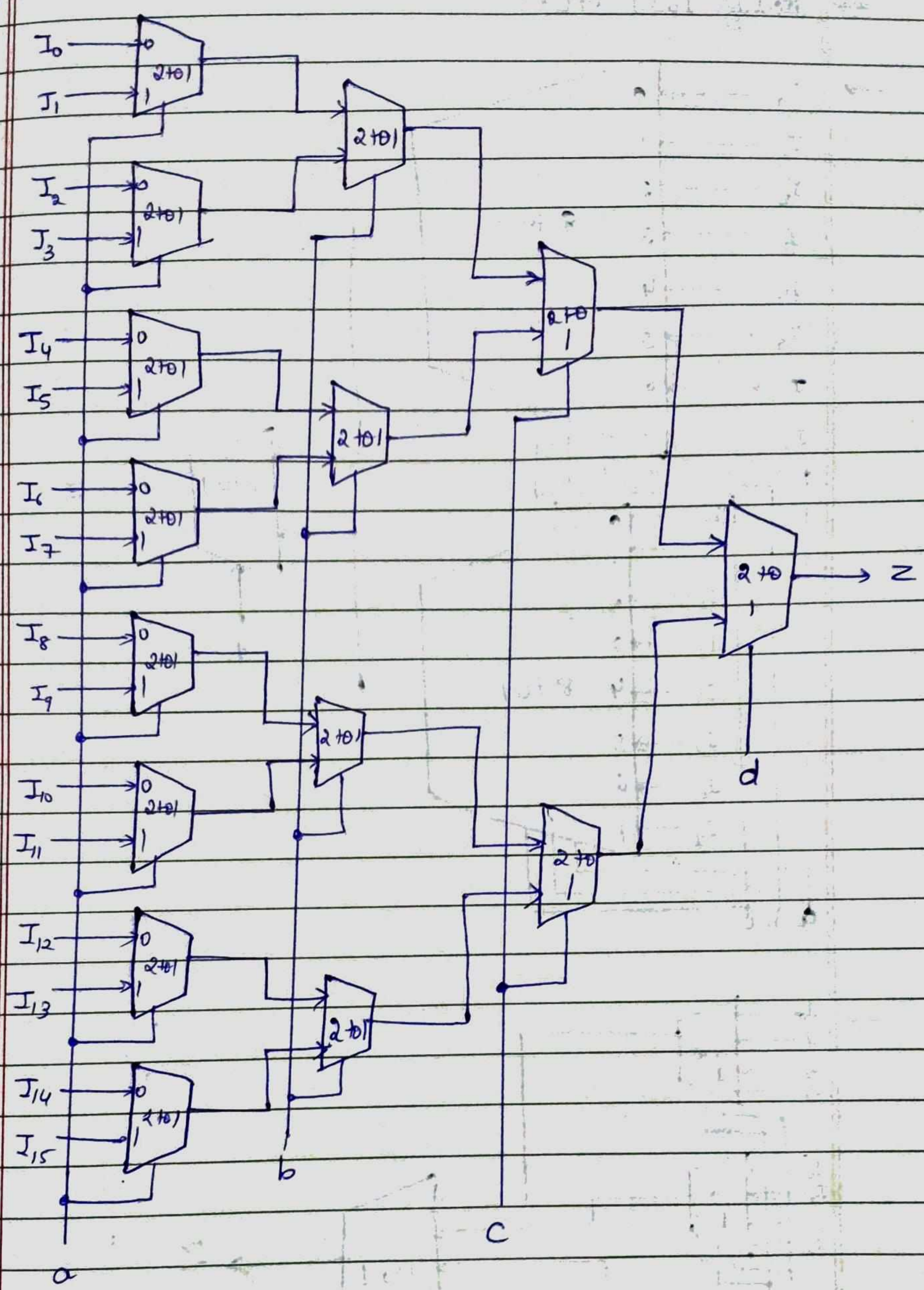
Realise 4x1 MUX using 2x1 MUX

4 inputs - I_0, I_1, I_2, I_3 , 1 output - Z ,

2 selection inputs - A, B .

⇒ Realize 16X1 MUX.

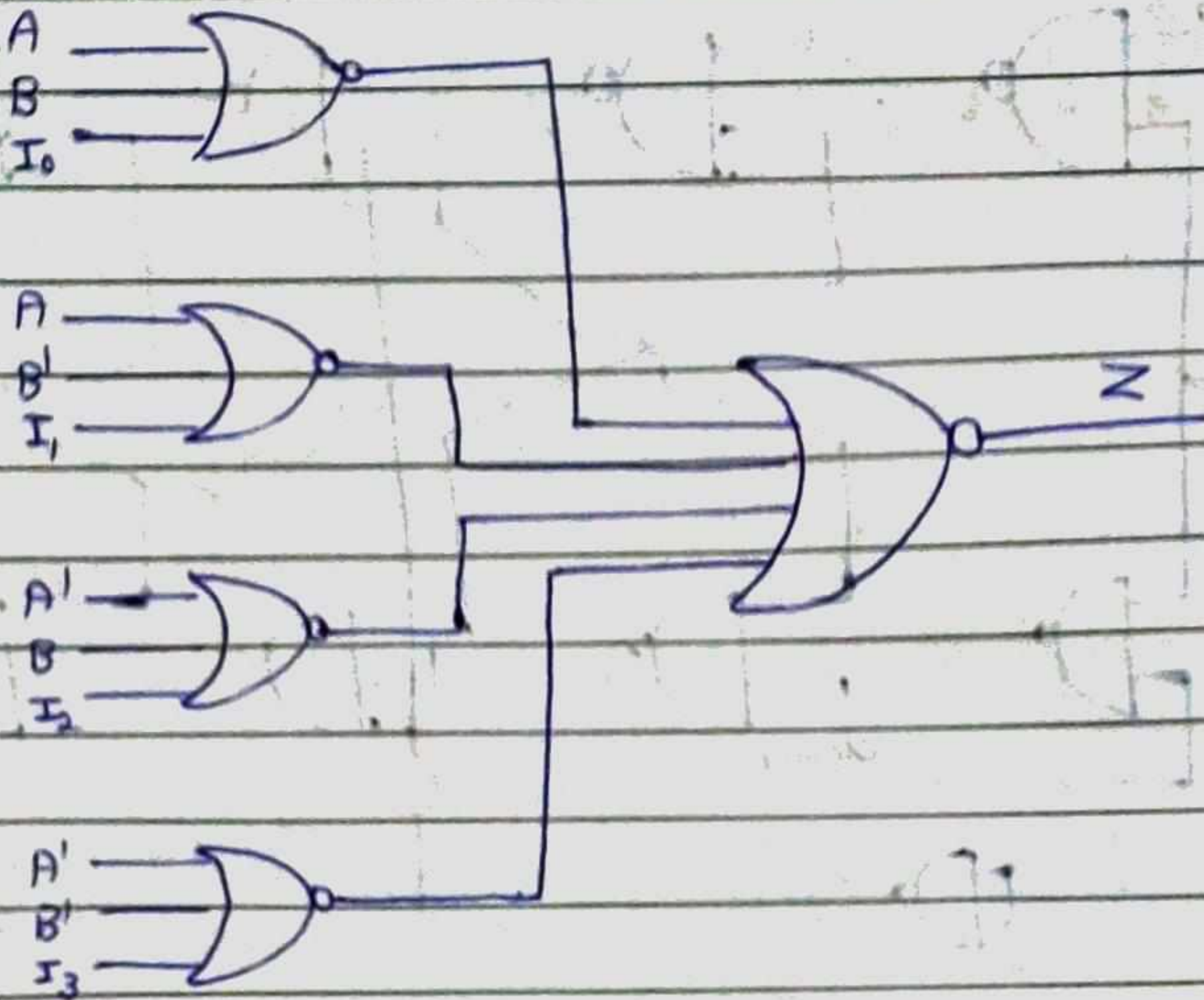




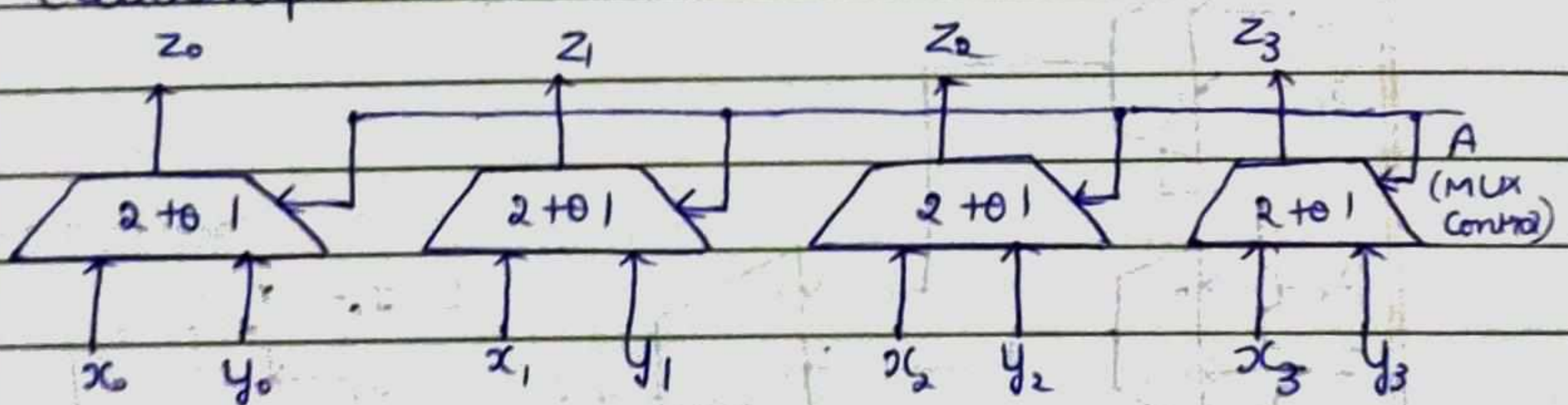
⇒ Implement 4×1 MUX using NOR-NOR circuit.

$$Z = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3$$

Apply duality, $Z = (A+B+I_0)(A+B'+I_1)(A'+B+I_2)(A'+B'+I_3)$



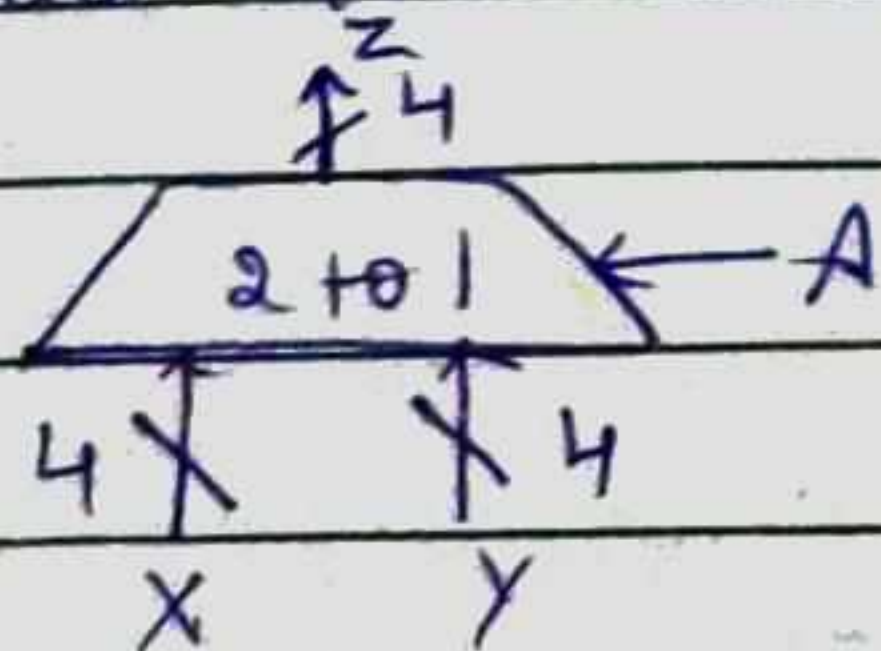
⇒ Quad multiplexer used to select data
(Quadruple 2×1 MUX).



Multiplexers are frequently used in digital system design to select the data which is to be processed or stored. The above fig shows a quadruple 2×1 MUX which is used to select 1 of two 4 bit words. If $A=0$, the values of x_0, x_1, x_2 & x_3 will appear at z_0, z_1, z_2 & z_3 outputs. If $A=1$, the values of y_0, y_1, y_2, y_3 will appear at the outputs.

Several logic signals that perform a common funcⁿ may be grouped together to form a bus.

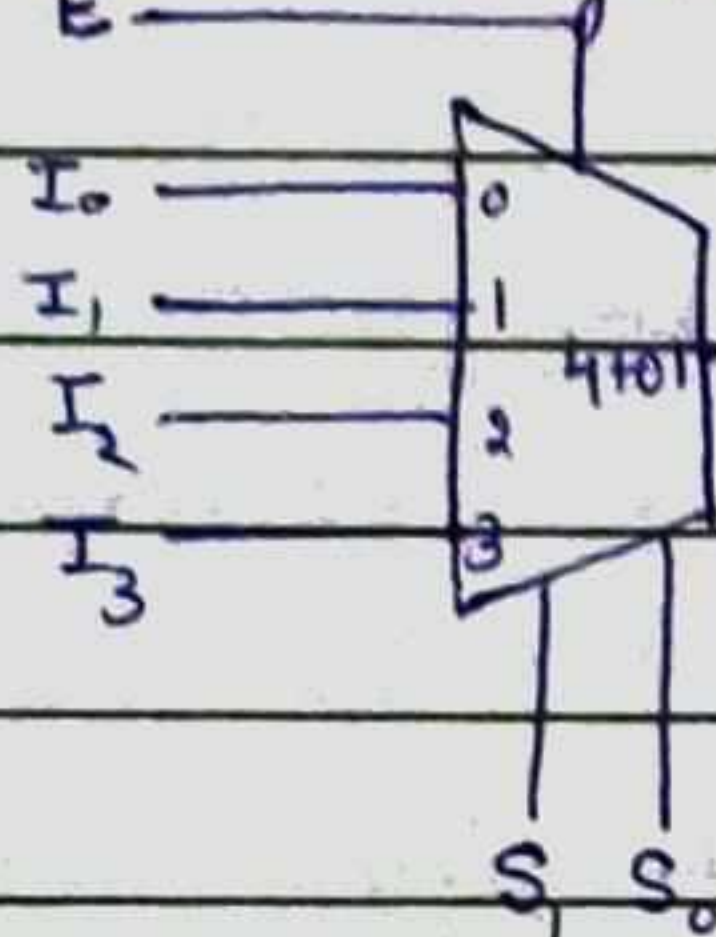
Quad MUX with bus input & outputs



Types of Multiplexers

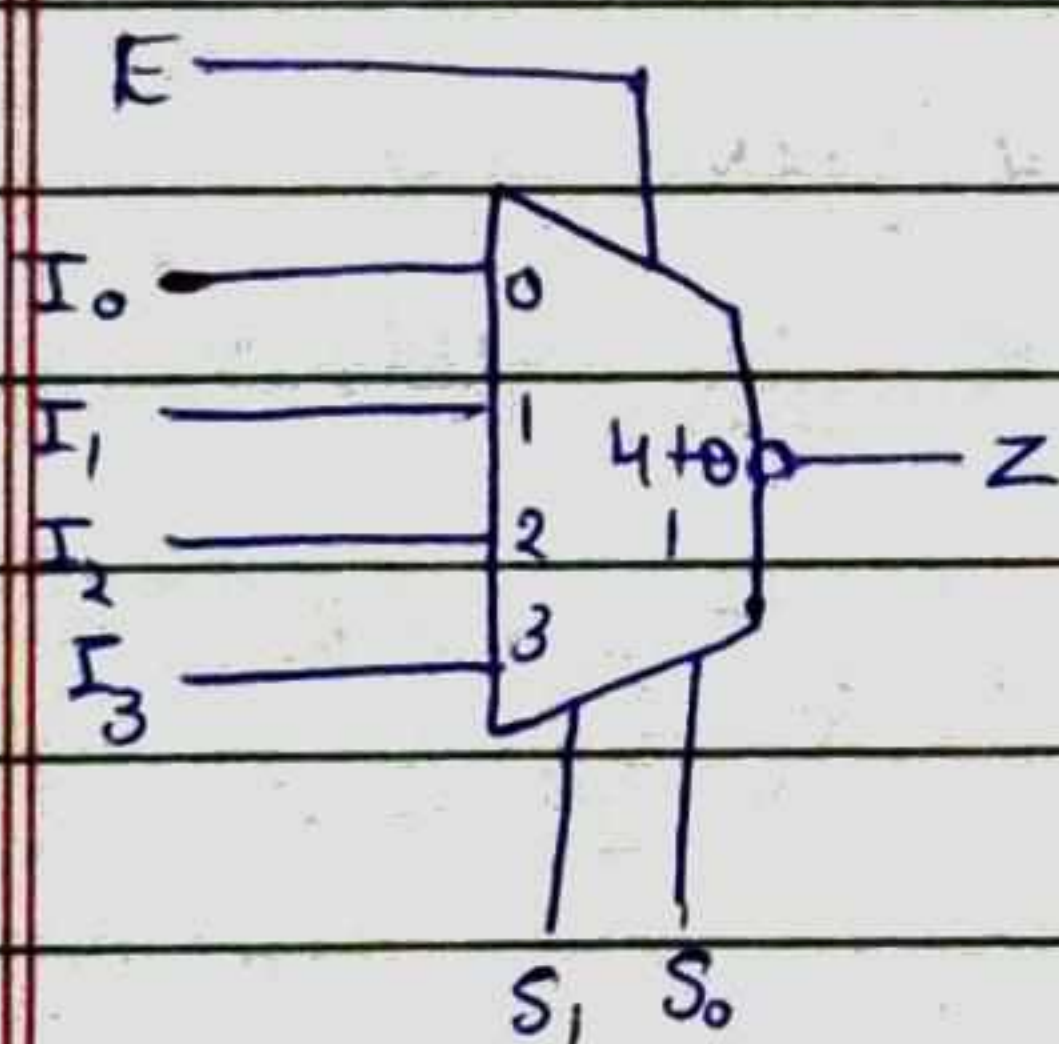
- 1) Active high output MUX.
- 2) Active low output MUX
- 3) Active high enable MUX
- 4) Active low enable MUX.

1. Active high output



Some multiplexers involve the inputs, if the multiplexers without the inversions have active high output.

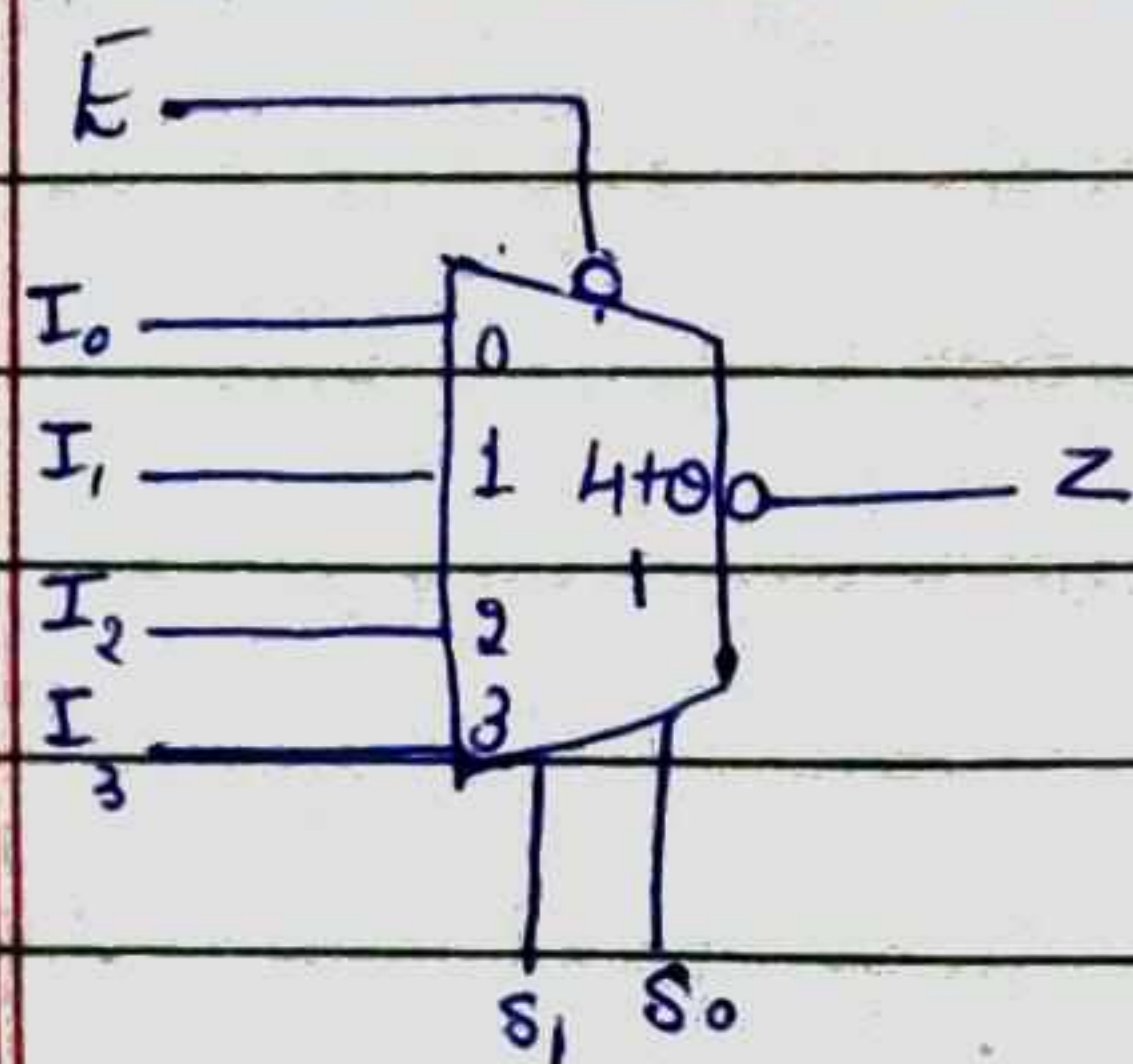
2. Active low output



MUX with inversions have active low inputs.

$$\text{If } I_0 = 0, Z = 1$$

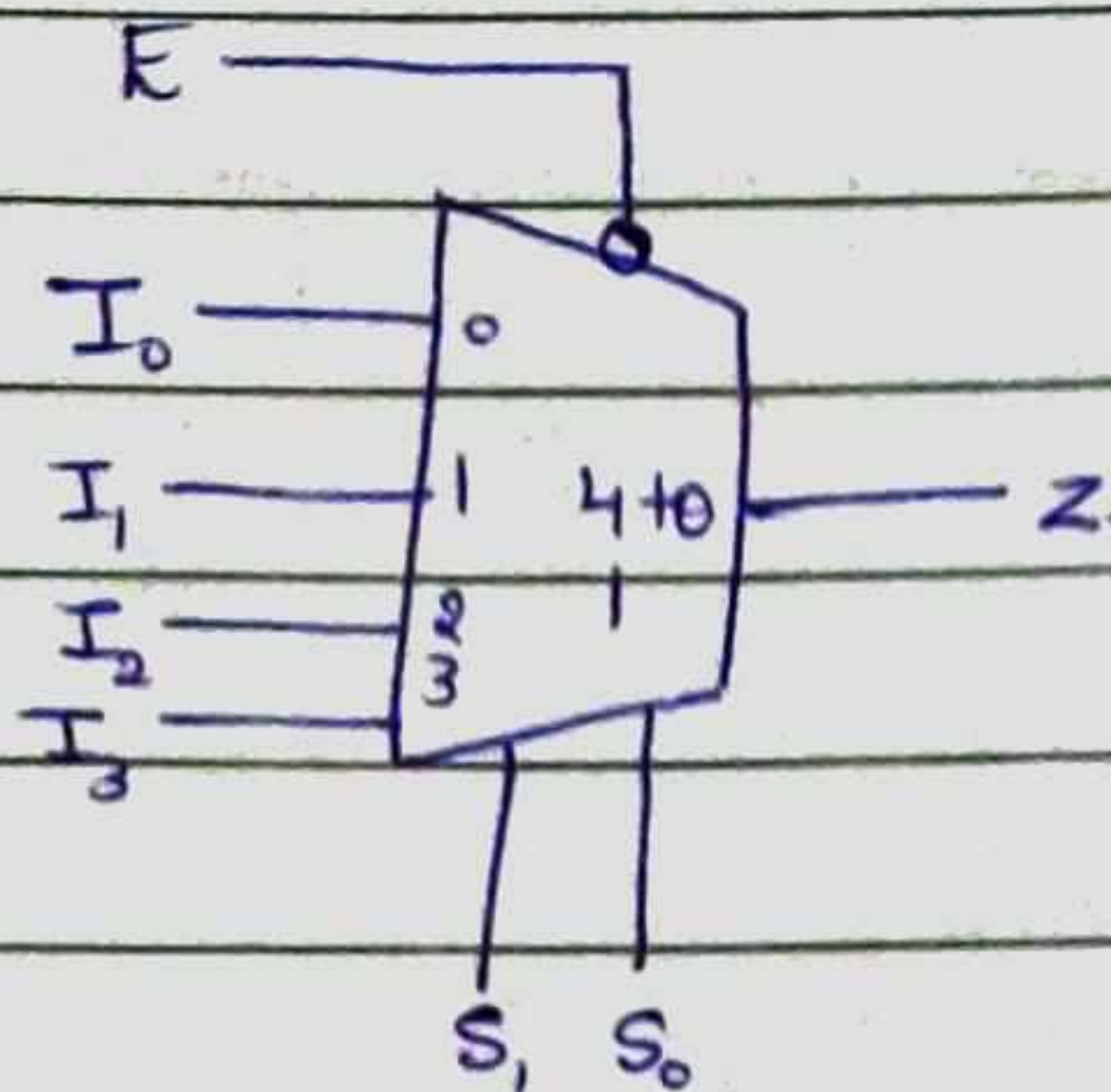
3. Active low enable



If inverter is inserted b/w \bar{E} & the AND gate, \bar{E} must be 0 for the MUX to function.

10

4 Active high enable



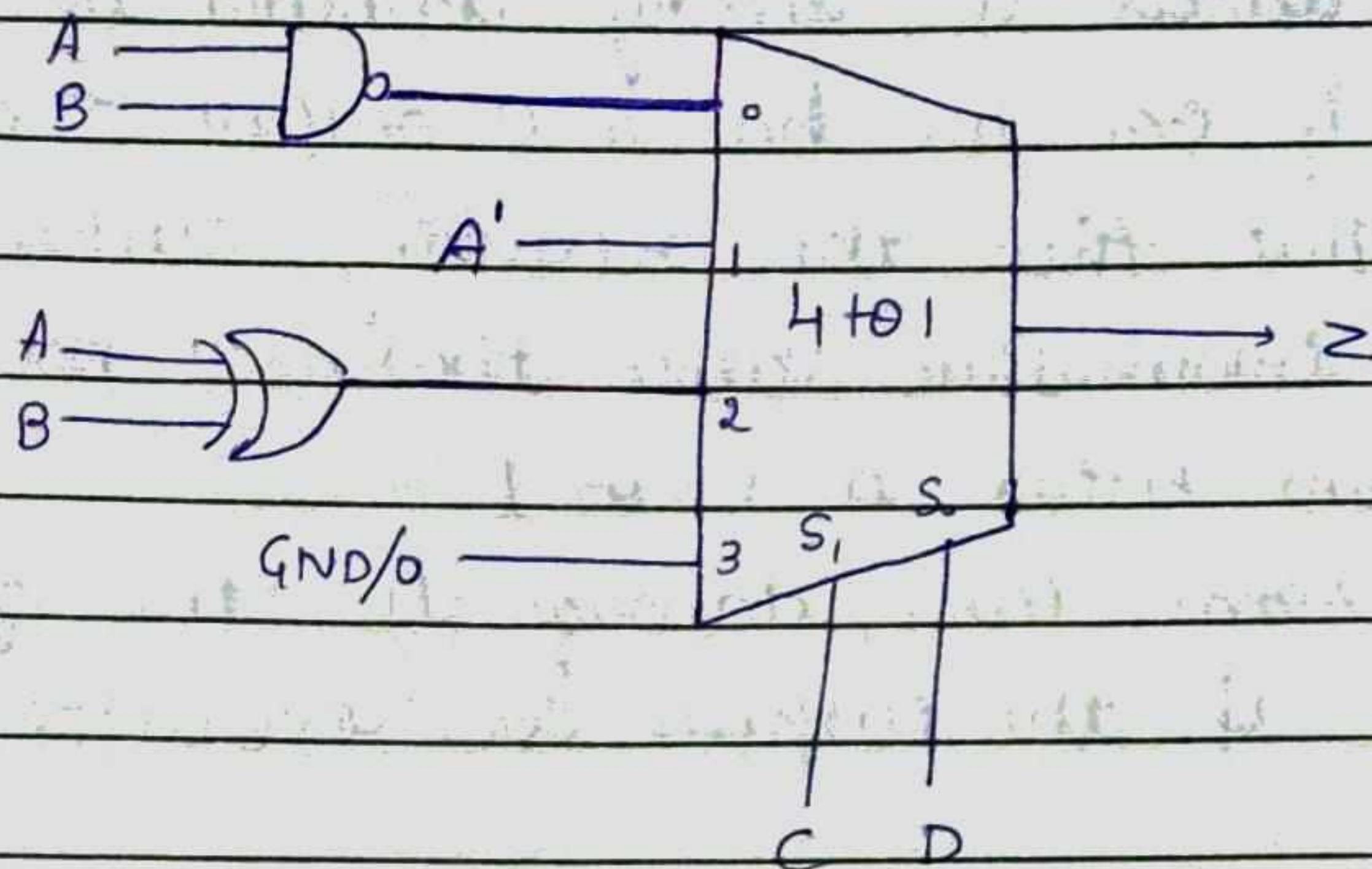
ckt

⇒ Implement 4-variable function using 4 to 1 MUX

$$Z = C'D'(A' + B') + C'D(A') + CD'(AB' + A'B) + CD(0)$$

$\underbrace{\hspace{2cm}}_{I_0} \quad \underbrace{\hspace{2cm}}_{I_1} \quad \underbrace{\hspace{2cm}}_{I_2} \quad \underbrace{\hspace{2cm}}_{I_3}$

S4

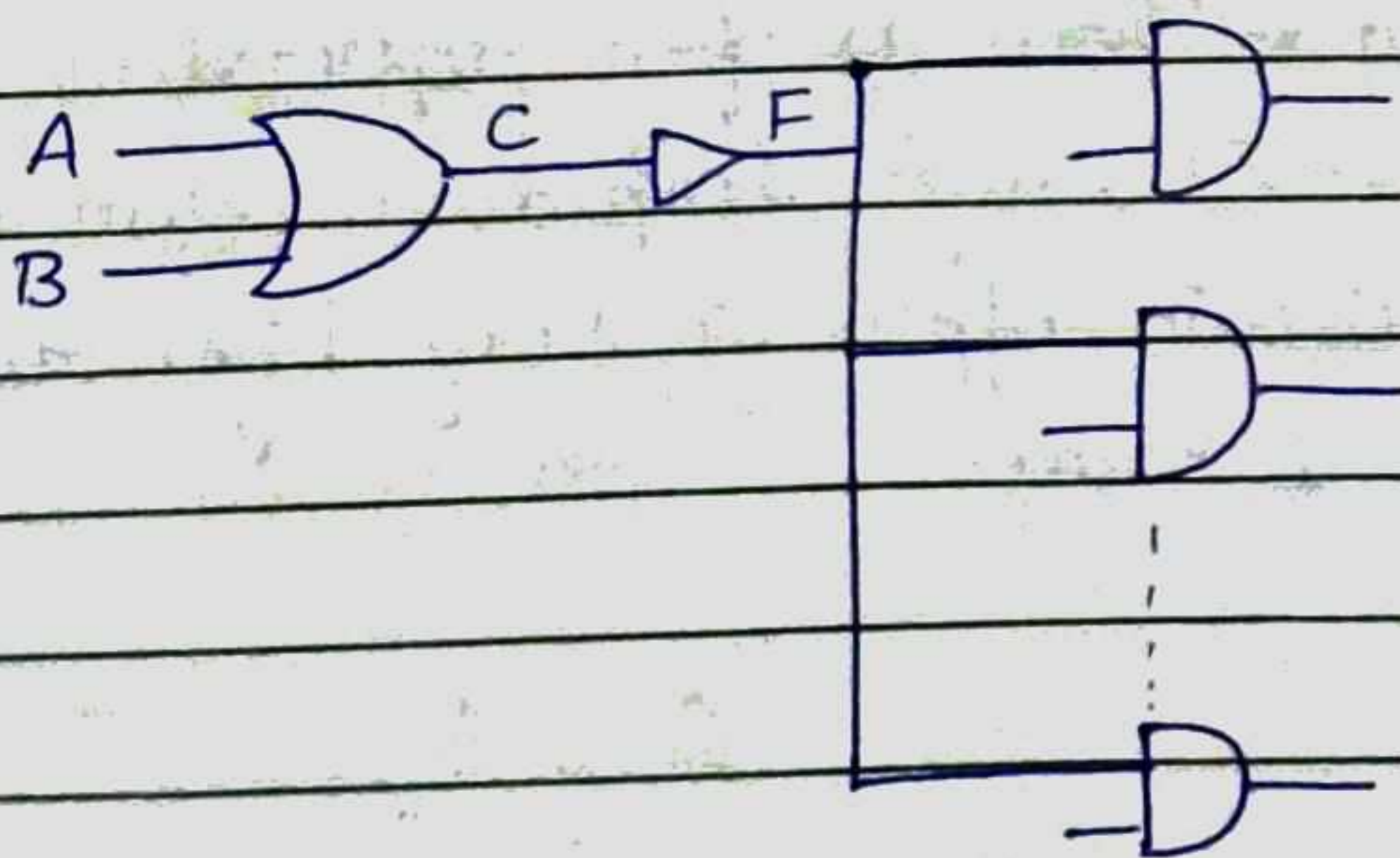


⇒ Buffers

A gate output can only be connected to a limited no. of ^{other device} inputs without degrading the performance of a digital system.

A simple buffer may be used to increase the driving capability of a gate output.

Since there is no bubble at the buffer output it can also be known/called as non-inverting buffer & the logic values of the buffer i/p & the o/p are the same.

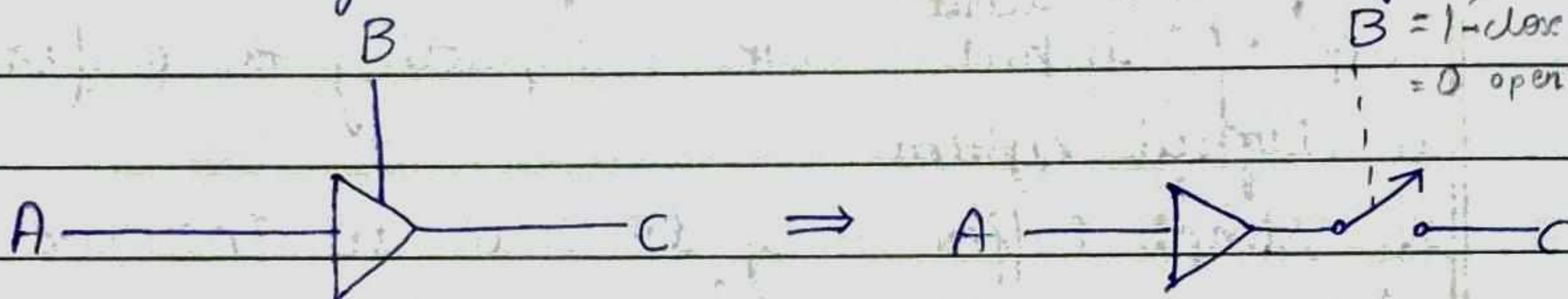


→ Why buffers are needed?

1. A logic circuit will not operate correctly if the outputs of 2 or more gates or other logic devices are directly connected to each other. For ex: If one gate has a 0 output & another has a 1 output then the resulting output may be some intermediate value that does not clearly represent either a 0 or 1.
2. In some cases, damage to the gates may result if the outputs are connected together.

VI → Three state Buffer / Tri state buffer

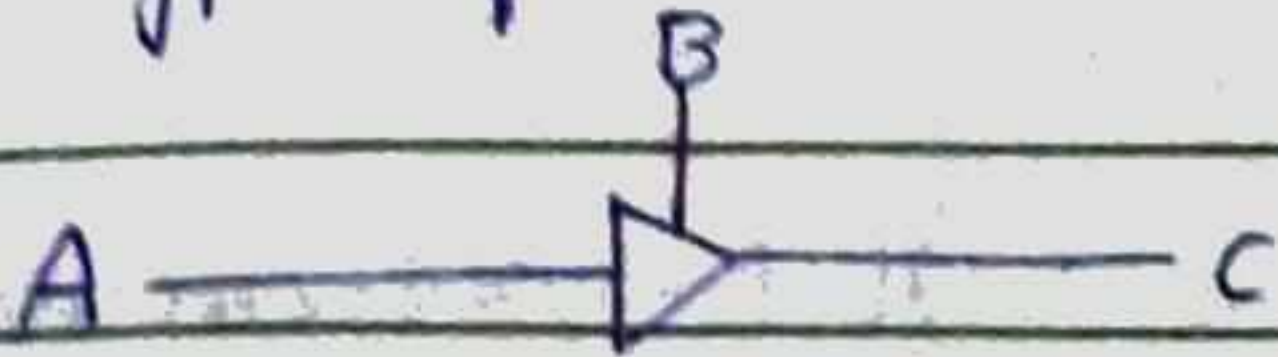
It permits the outputs of two or more gates or other logic devices to be connected together. works (Hi-Z)



Eg: Three state buffer & its logical equivalent.

- When the enable i/p B is 1, the o/p $C = A$.
- When the enable i/p B is 0, the o/p C acts like an open circuit - This is often referred as high impedance (Hi-Z), high impedance state of the o/p because the circuit offers a very high resistance to the flow of current.

→ Types of Tri State Buffer: i) Enable Input B is not inverted.

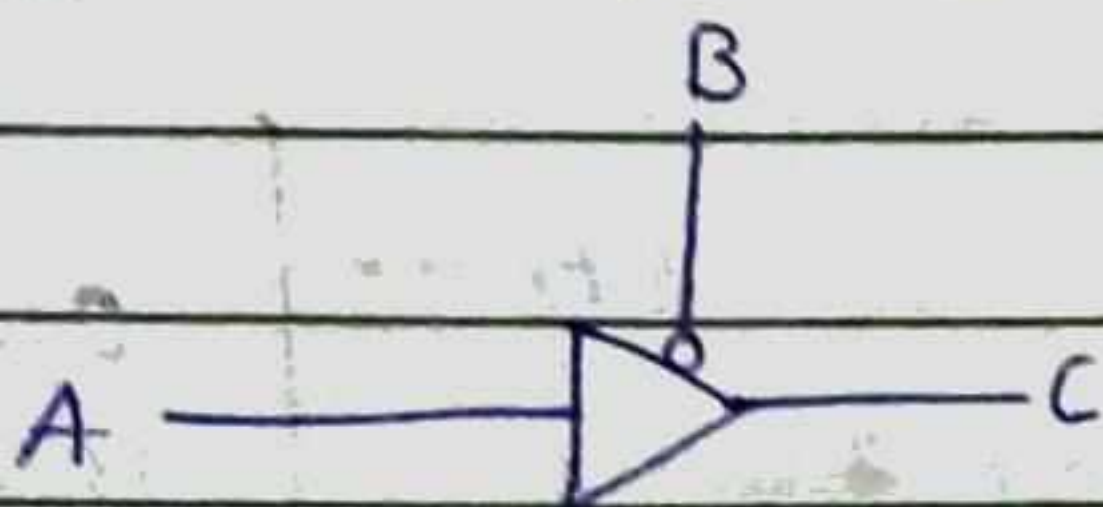


if $B=1$, $C=A$

if $B=0$, Z (open circuit)

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

ii) Enable input B is inverted

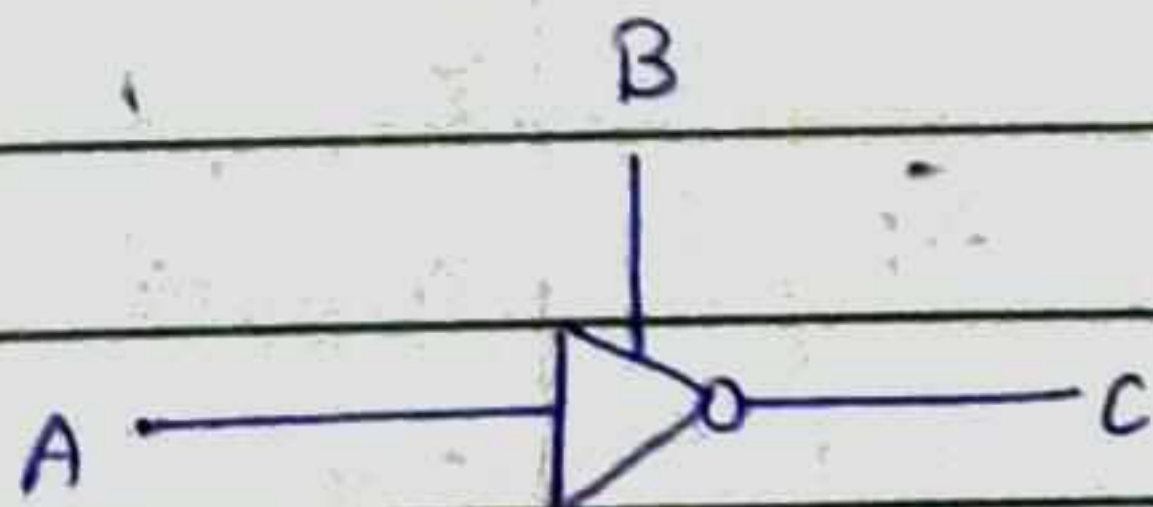


if $B=1$, Z (open circuit)

if $B=0$, $C=A$

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

iii) Enable input B is not inverted & output is inverted



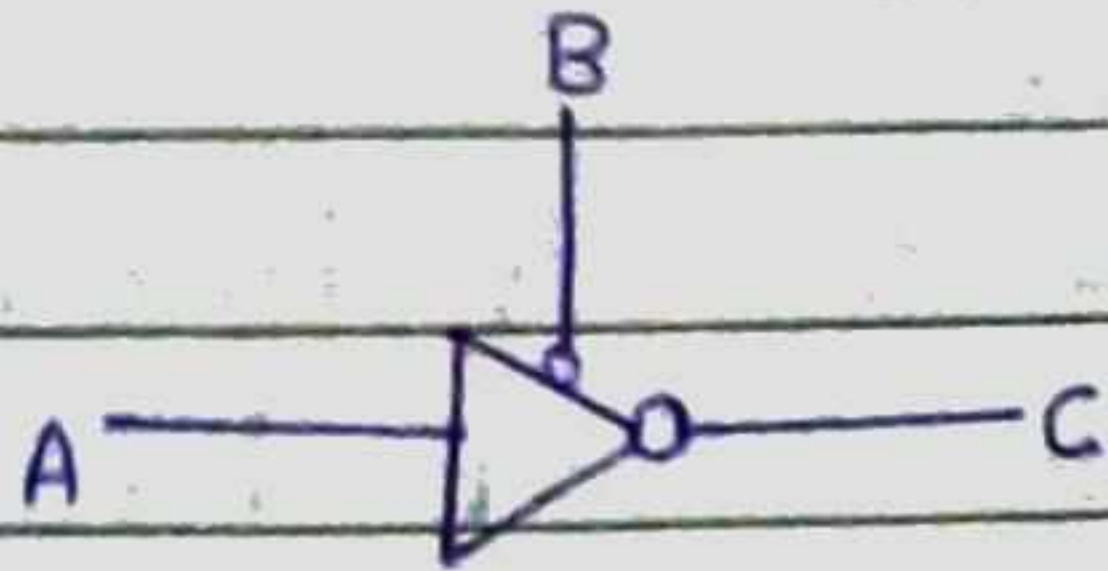
if $B=1$, $C=A'$

if $B=0$, Z (open circuit)

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

(14)

i) Enable input B is inverted & output is inverted

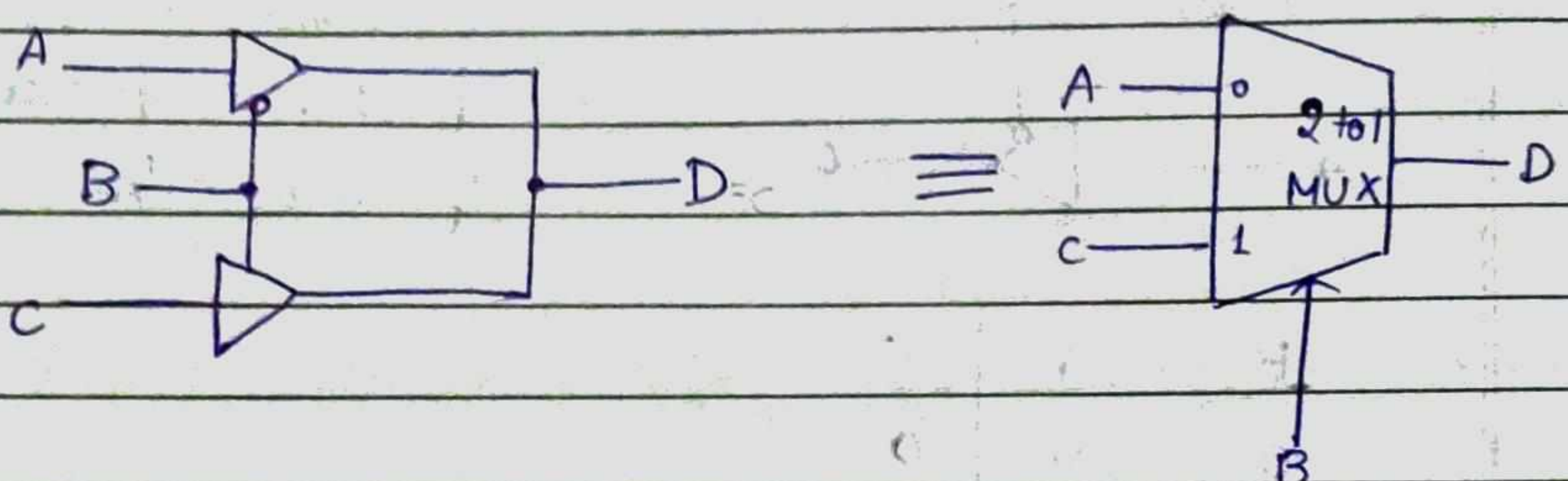


If $B=1$, Z (open circuit)

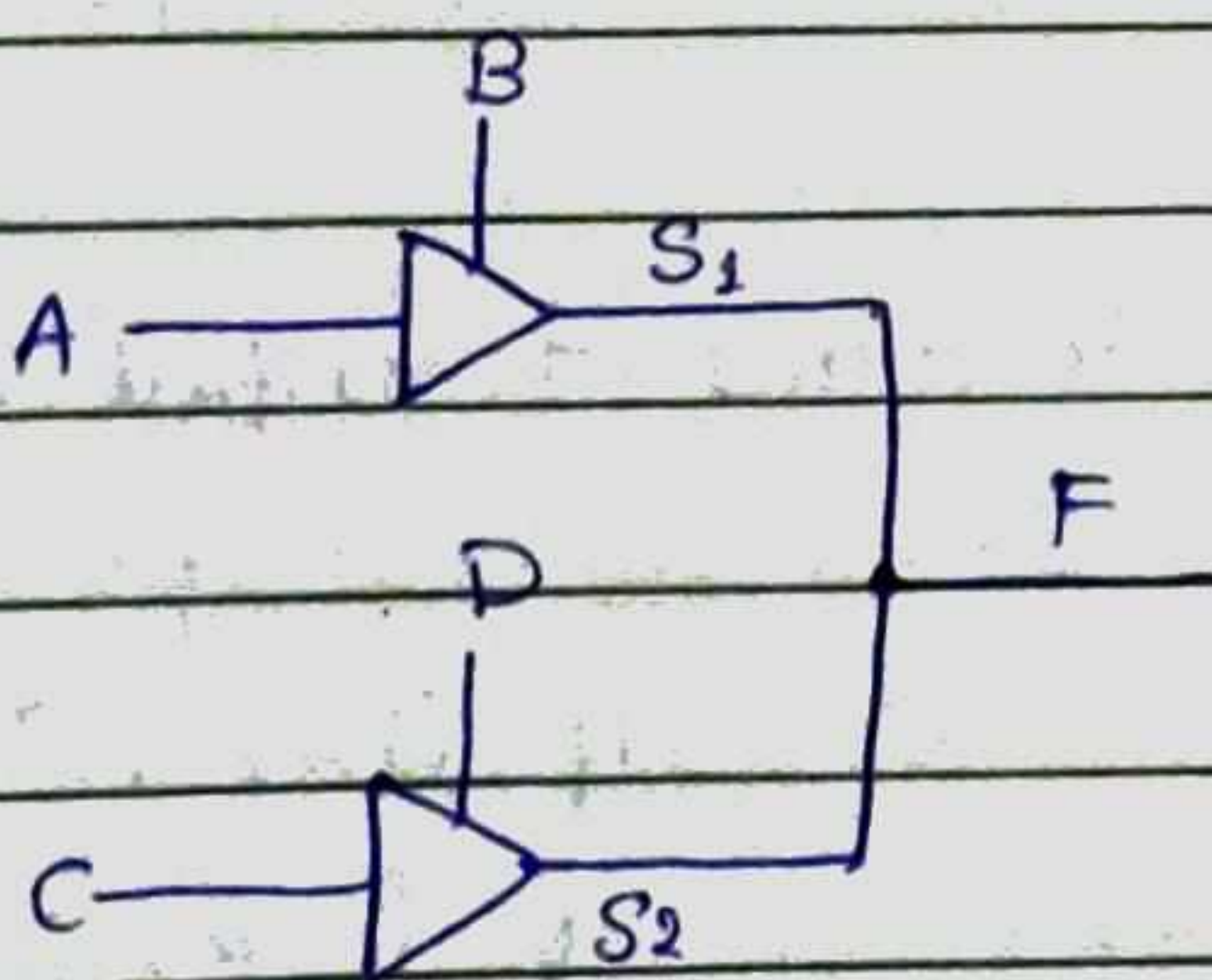
If $B=0$, $C=A'$

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

→ Data Selection (MUX) using three State Buffer

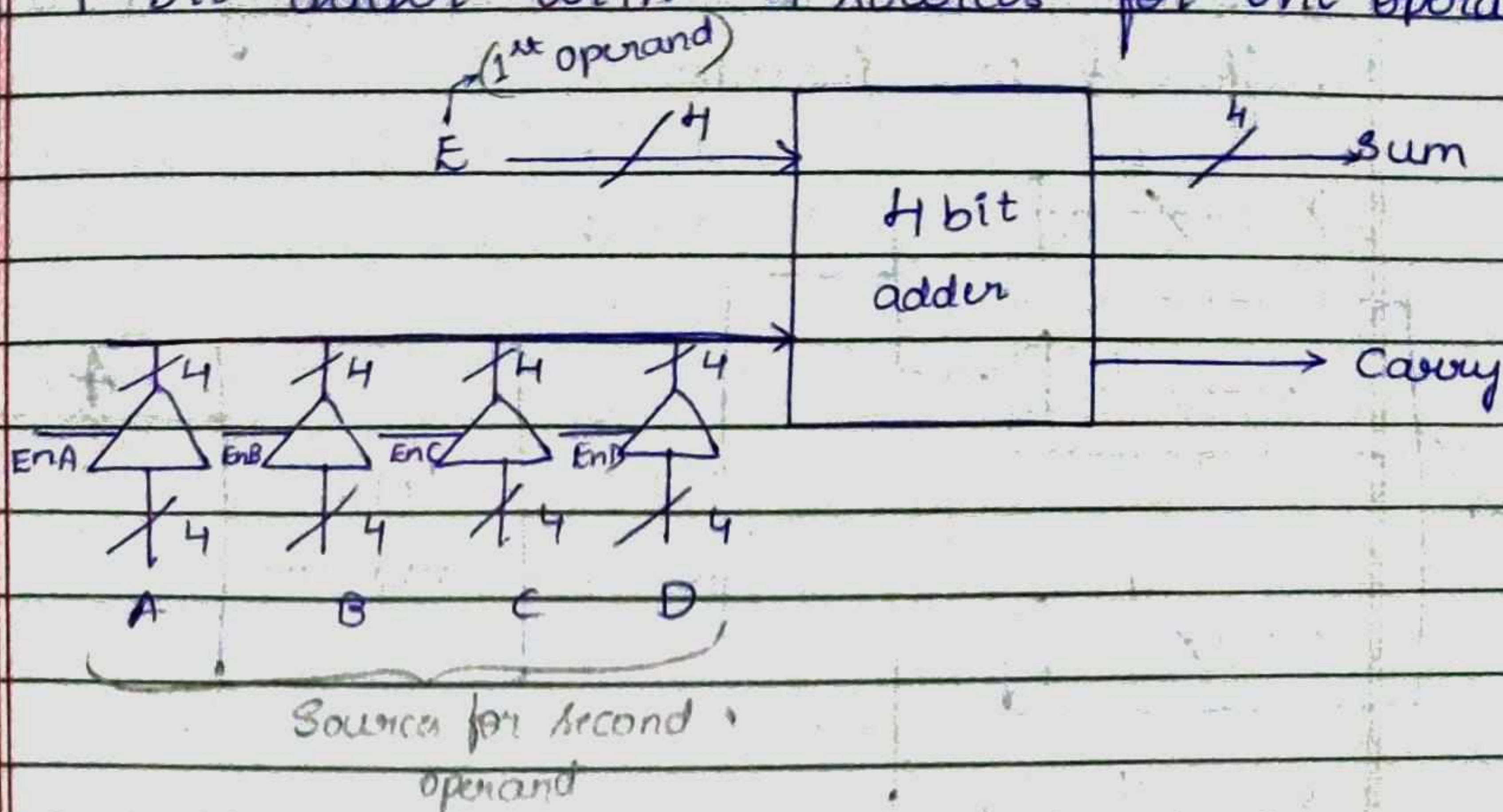


Circuit with Two Three State Buffer

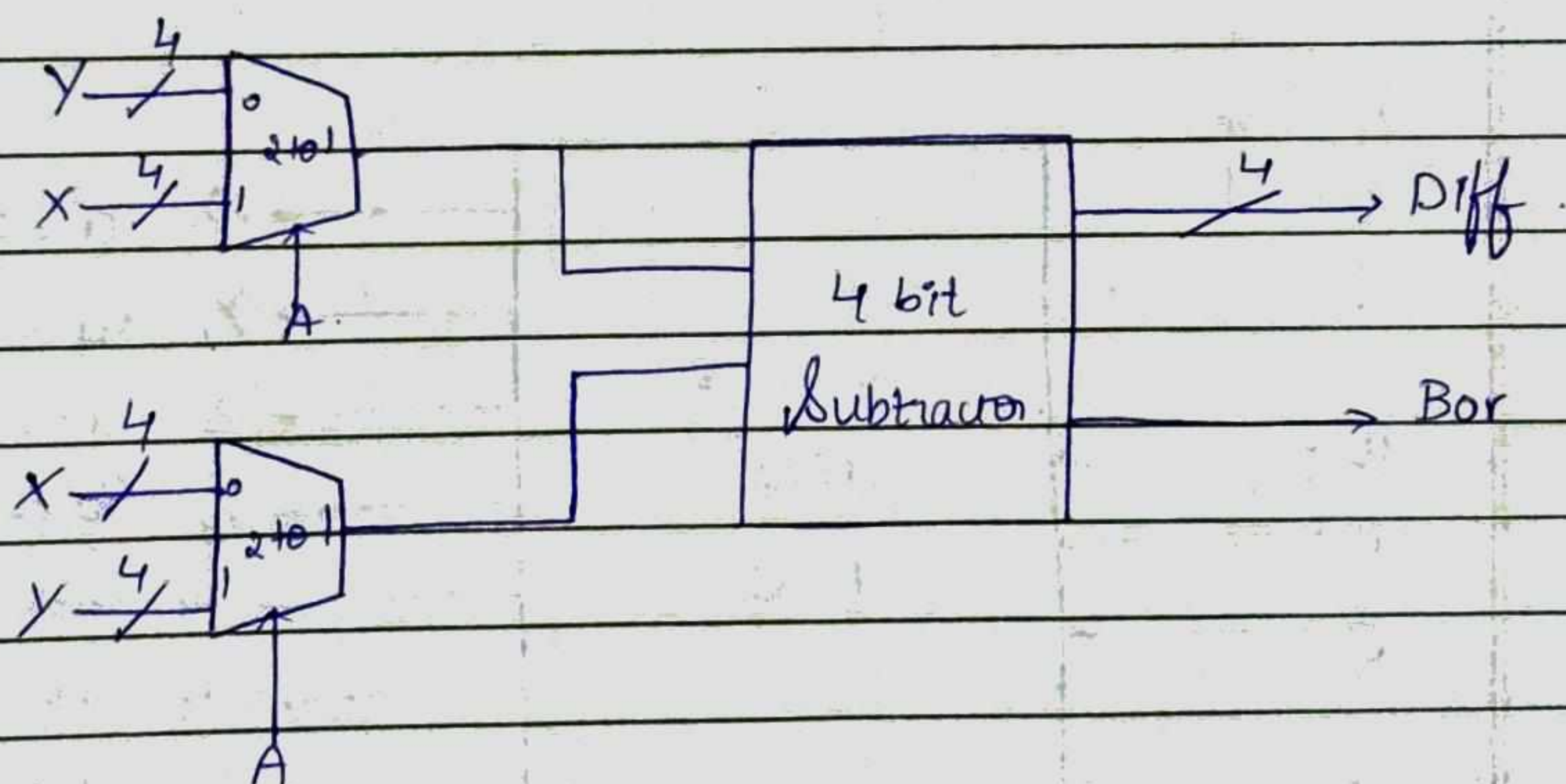


S_1	S_2	X	0	1	Z
X	X	X	X	X	X
0	0	X	0	X	0
1	1	X	X	1	1
Z	Z	X	0	1	Z

VI \Rightarrow 4-bit adder with 4 sources for one operand



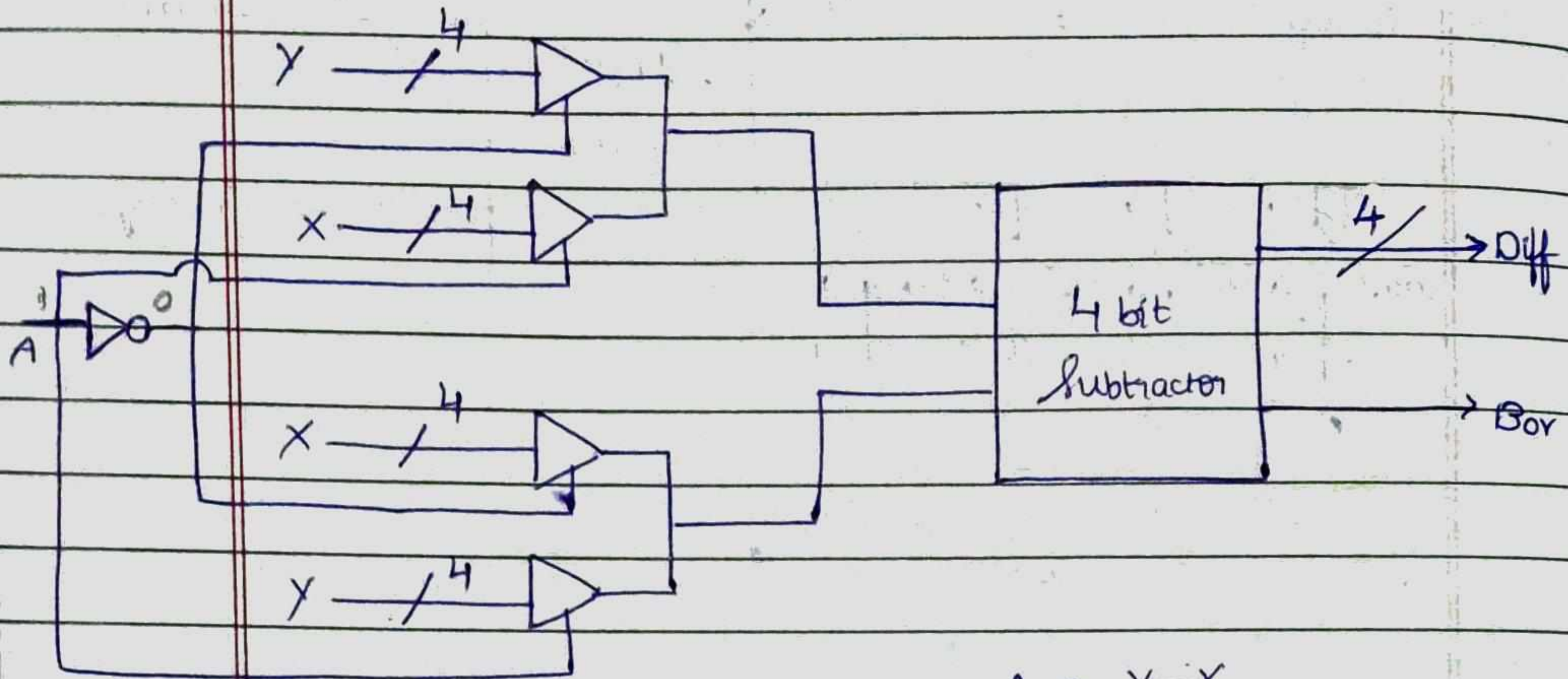
\Rightarrow Design a circuit which will either subtract X from Y or Y from X depending on the value of A . If A is equal to 1, the output should be $X - Y$ & if A is equal to 0, then the o/p should be $Y - X$. Use a 4-bit subtractor & 2 4 bit multiplexers.



$$A=0, Y-X$$

$$A=1, X-Y$$

⇒ Implement the above circuit using 4 tri-state buffers & one inverter.



$A=0, Y-X$

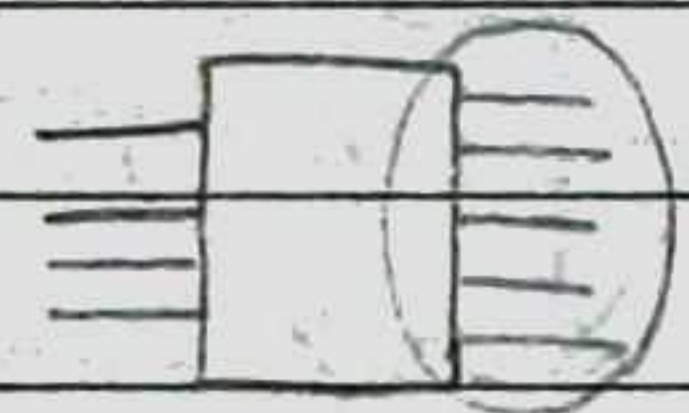
$A=1, X-Y$

⇒ Decoders

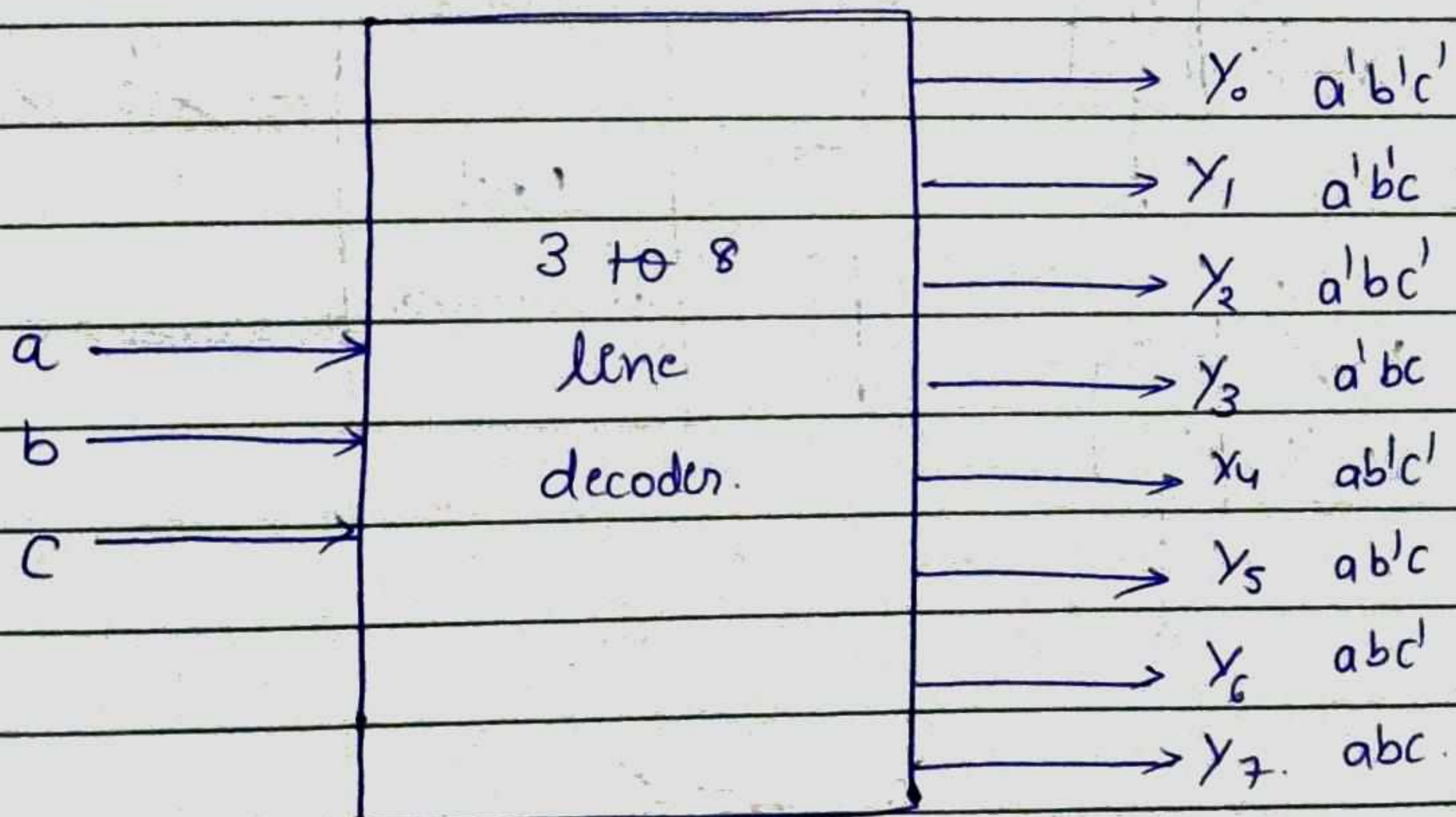
→ Binary to Decimal Decoder

3 to 8 line decoder.

3 input lines & 8 output lines.



one out of multiple outputs will be selected



(17)

Page No.

Date

a	b	c	γ_0	γ_1	γ_2	γ_3	γ_4	γ_5	γ_6	γ_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

- This decoder generates all of the minterms of the three input variables.
- Exactly one of the output lines will be high or 1 for each combination of the value of i/p variables.
- When a, b, c is equal to 0, then y_0 will be highlighted, $y_0 = a'b'c'$.
- In general n to 2^n line decoder generates all 2^n minterms or maxterms of the i/p variables.
- The o/p's can be defined by the eqⁿ

$$y_i = m_i = M_i', \text{ for } i = 0 \text{ to } 2^n - 1$$

for non-inverted output - no bubble connected to the output.

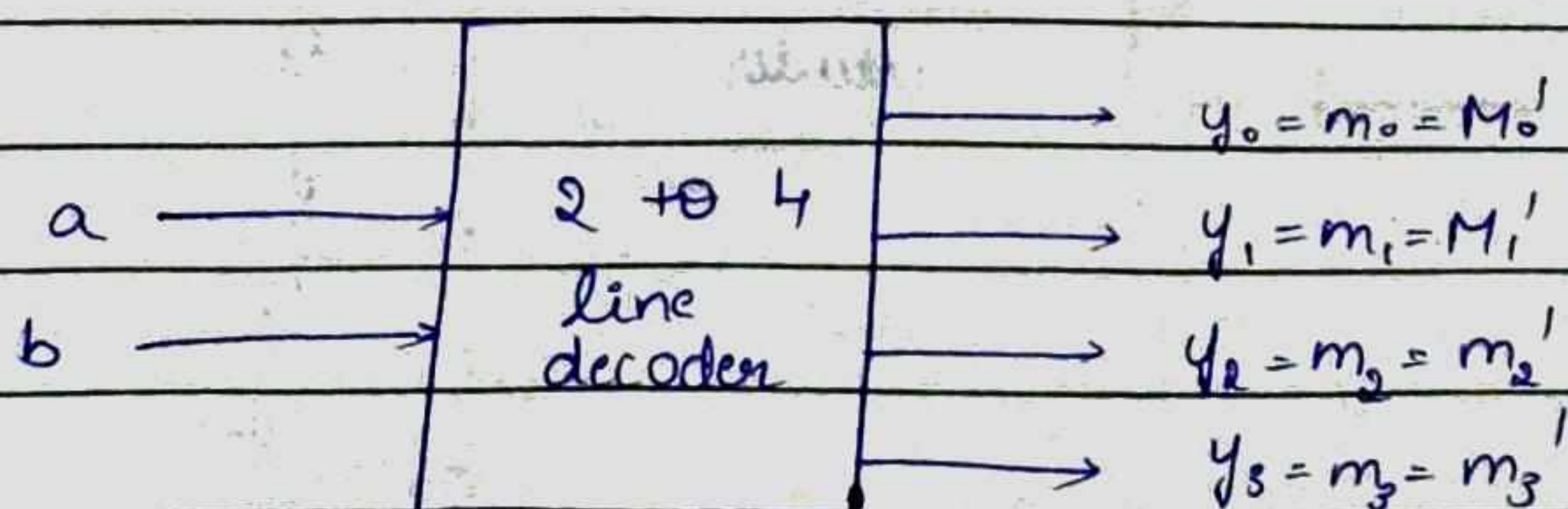
$$y_i = m_i' = M_i, \text{ for } i = 0 \text{ to } 2^n - 1$$

for inverted outputs - bubbles connected to the output.

2 to 4 line decoder

2 input lines & 4 output lines.

a	b	y_0	y_1	y_2	y_3	
0	0	1	0	0	0	$y_0 = m_0 = M_0'$
0	1	0	1	0	0	$y_1 = m_1 = M_1'$
1	0	0	0	1	0	$y_2 = m_2 = m_2'$
1	1	0	0	0	1	$y_3 = m_3 = m_3'$

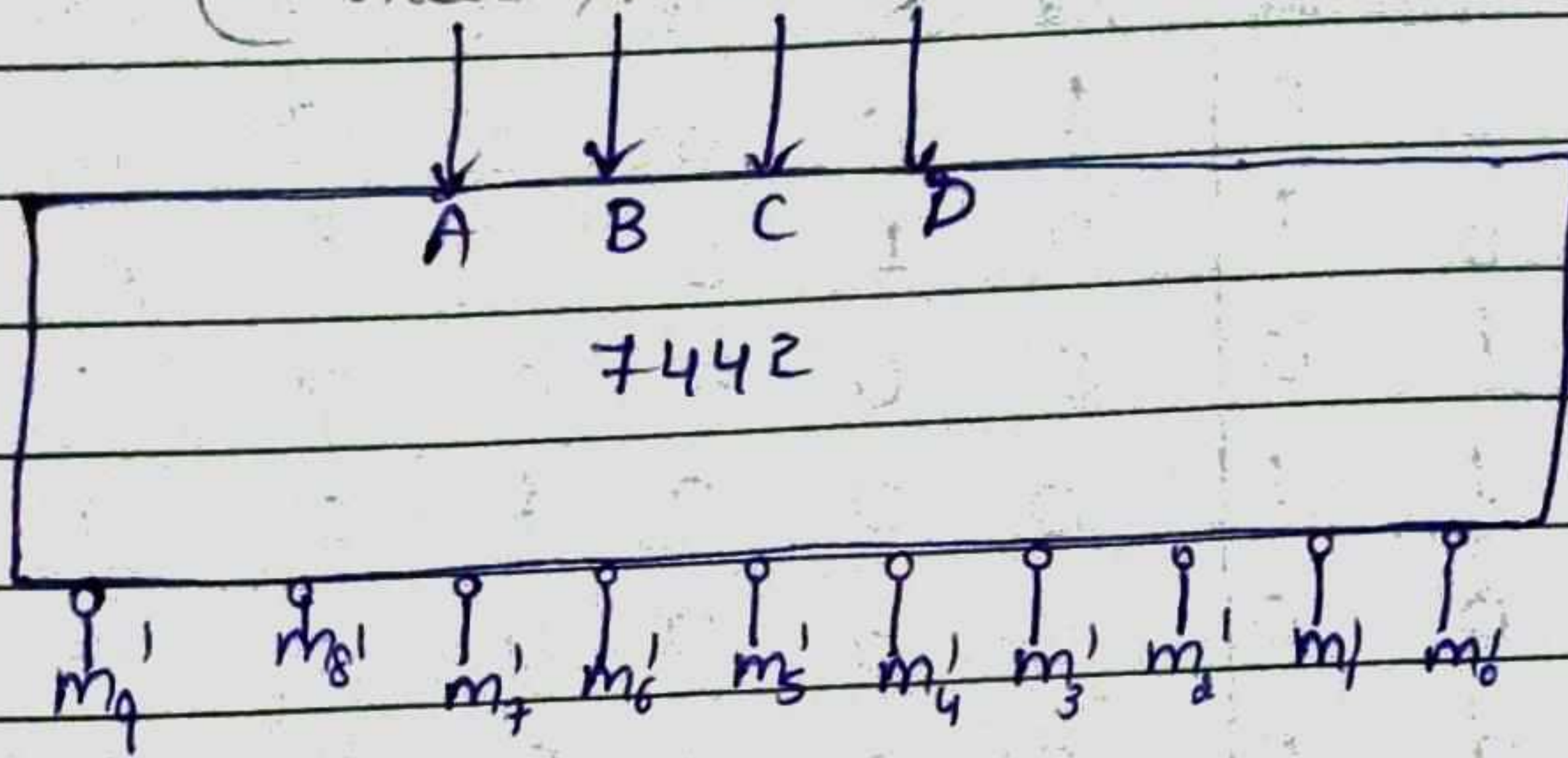


⇒ 4 to 10 decoder (BCD to decimal decoder)

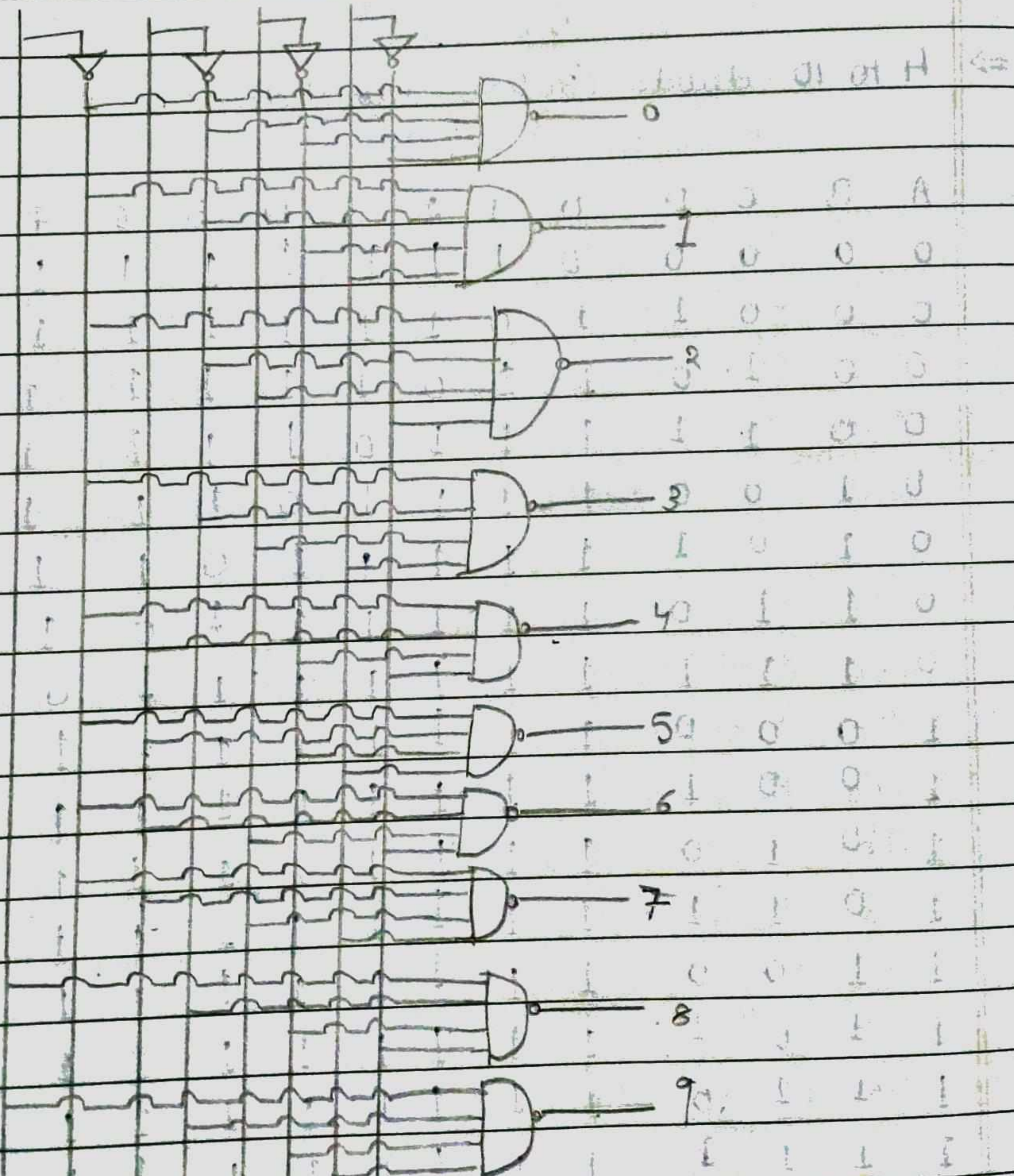
[illegible]

(20)

(Inverted O/P decoder)

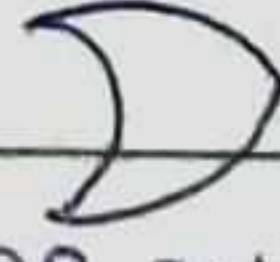
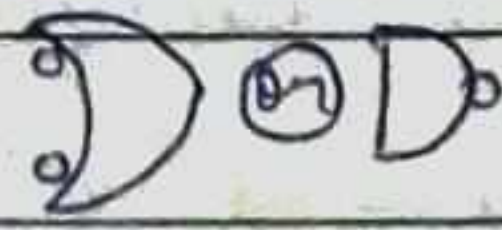
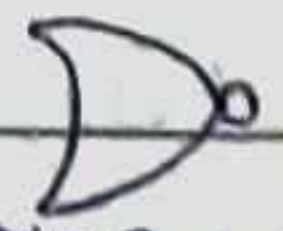



A \bar{A} B \bar{B} C \bar{C} D \bar{D}



21

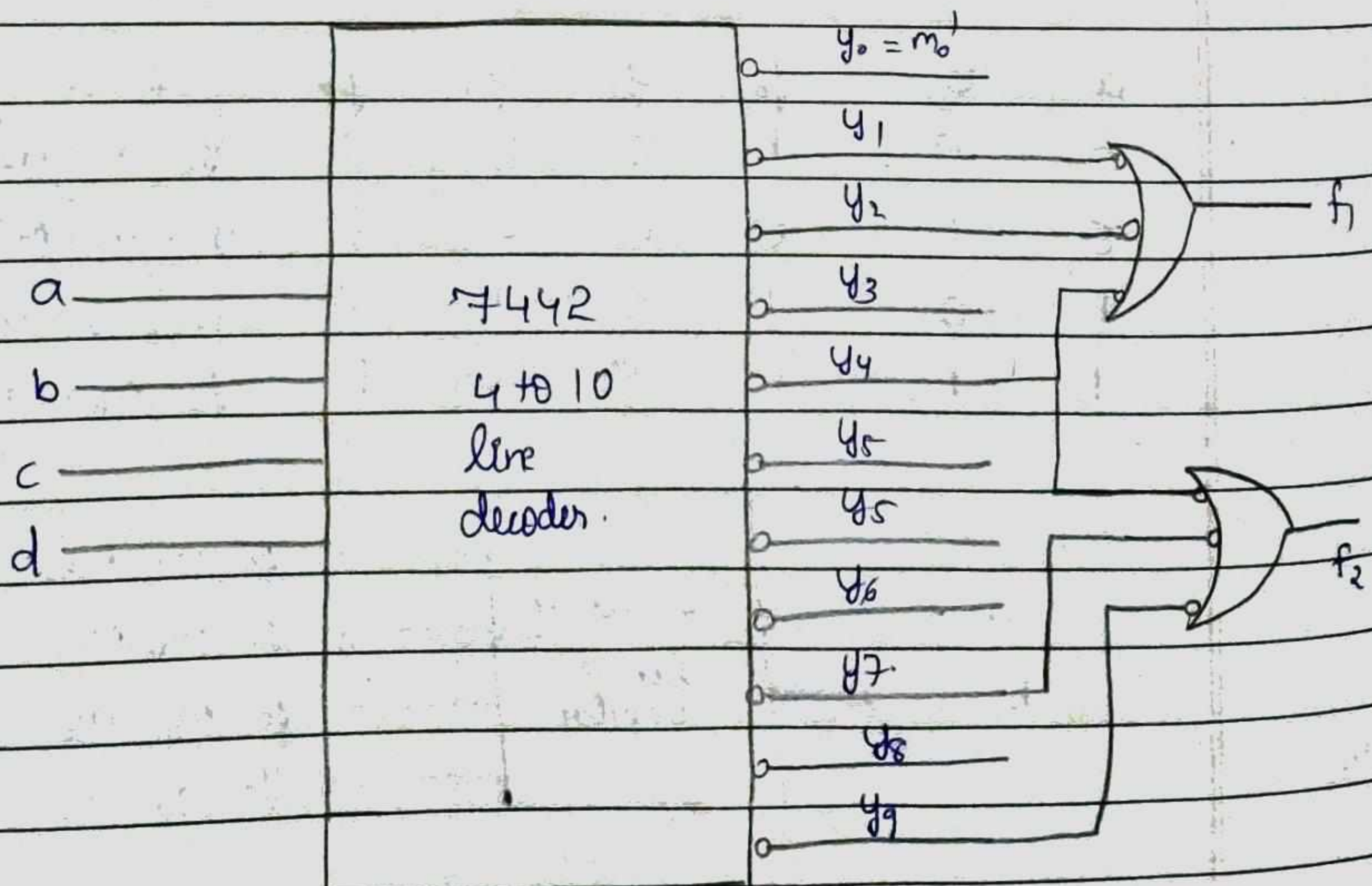
⇒ Realisation of decoder

	Non inverted output	Inverted output
minterm	 OR gate	 Bubbled OR (NAND)
maxterm	 NOR gate	 AND gate

⇒ Realise $f_1(a,b,c,d) = m_1 + m_2 + m_4$
(1, 2, 4)

$f_2(a,b,c,d) = m_4 + m_7 + m_9$
(4, 7, 9)

Using 7442 IC (BCD to decimal decoder)



⇒

Note:

7445 is also a BCD to decimal decoder with identical pin description as that of IC 7442. i.e., 7442 & 7445 are same.

⇒

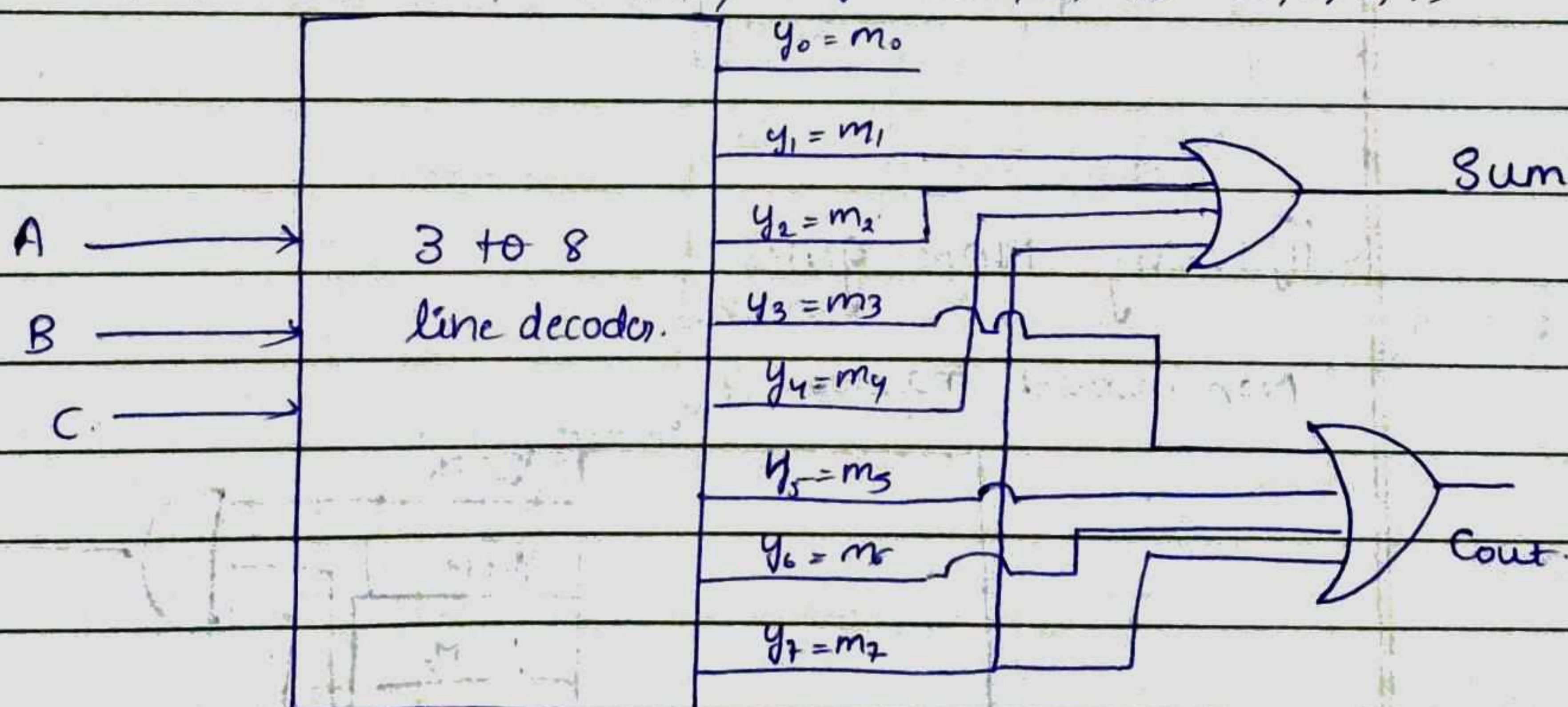
Realise a full adder using 3 to 8 line decoder.

S:- Full adder

Truth Table

A	B	C _{in}	Sum	Carry (C _{out})
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$\text{Sum} = f_1(A, B, C_{in}) = \sum m(1, 2, 4, 7), \text{Carry} = f_2(A, B, C_{in}) = \sum m(3, 5, 6, 7)$$



→ Realise the Boolean expression $f_1(x_2, x_1, x_0) = \pi M(0, 1, 3, 5)$
 $f_2(x_2, x_1, x_0) = \pi M(1, 3, 6, 7)$

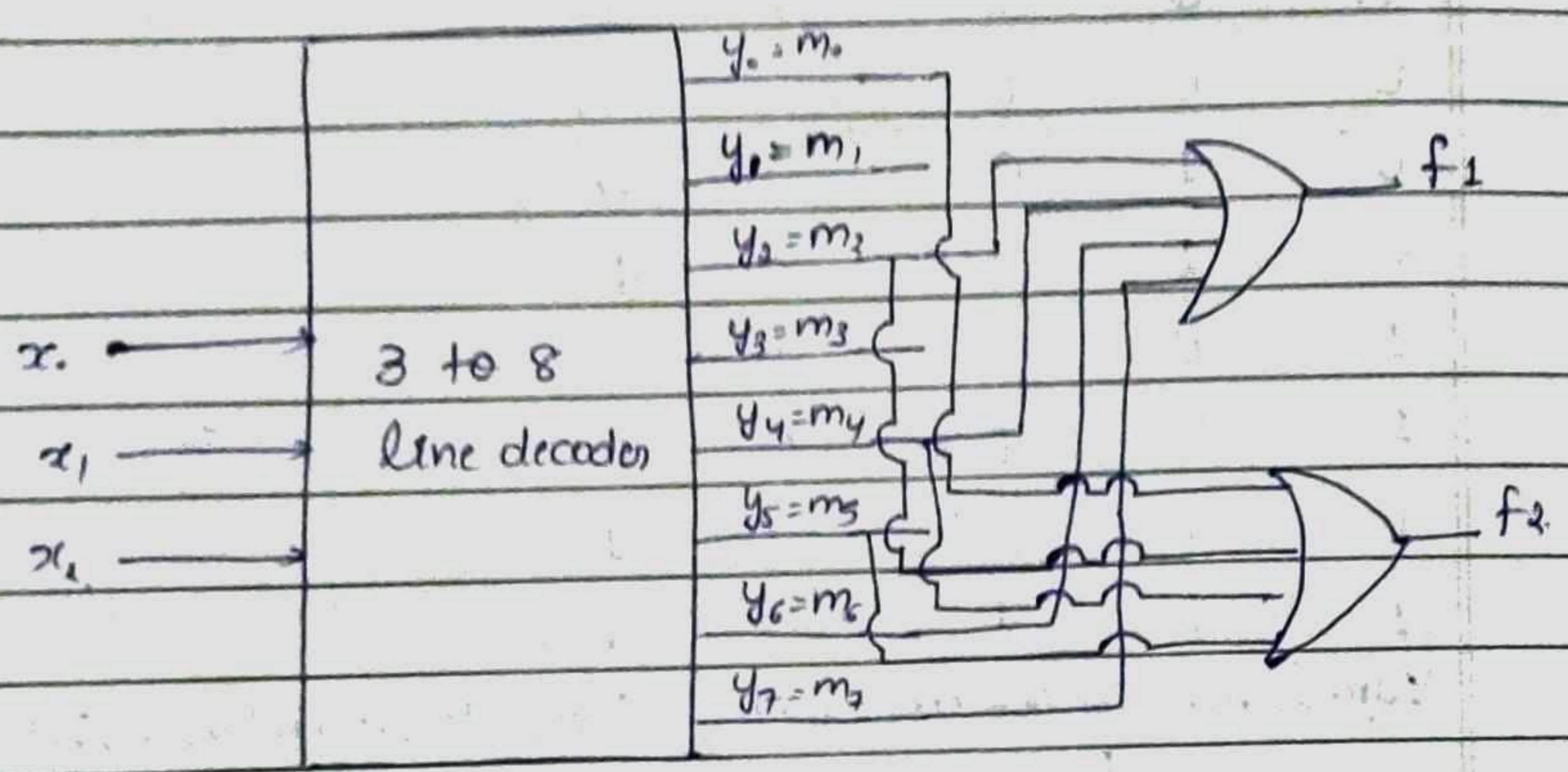
Using OR gates.

3. Non-inverted outputs of minterms = OR gate.

Convert maxterms into minterms.

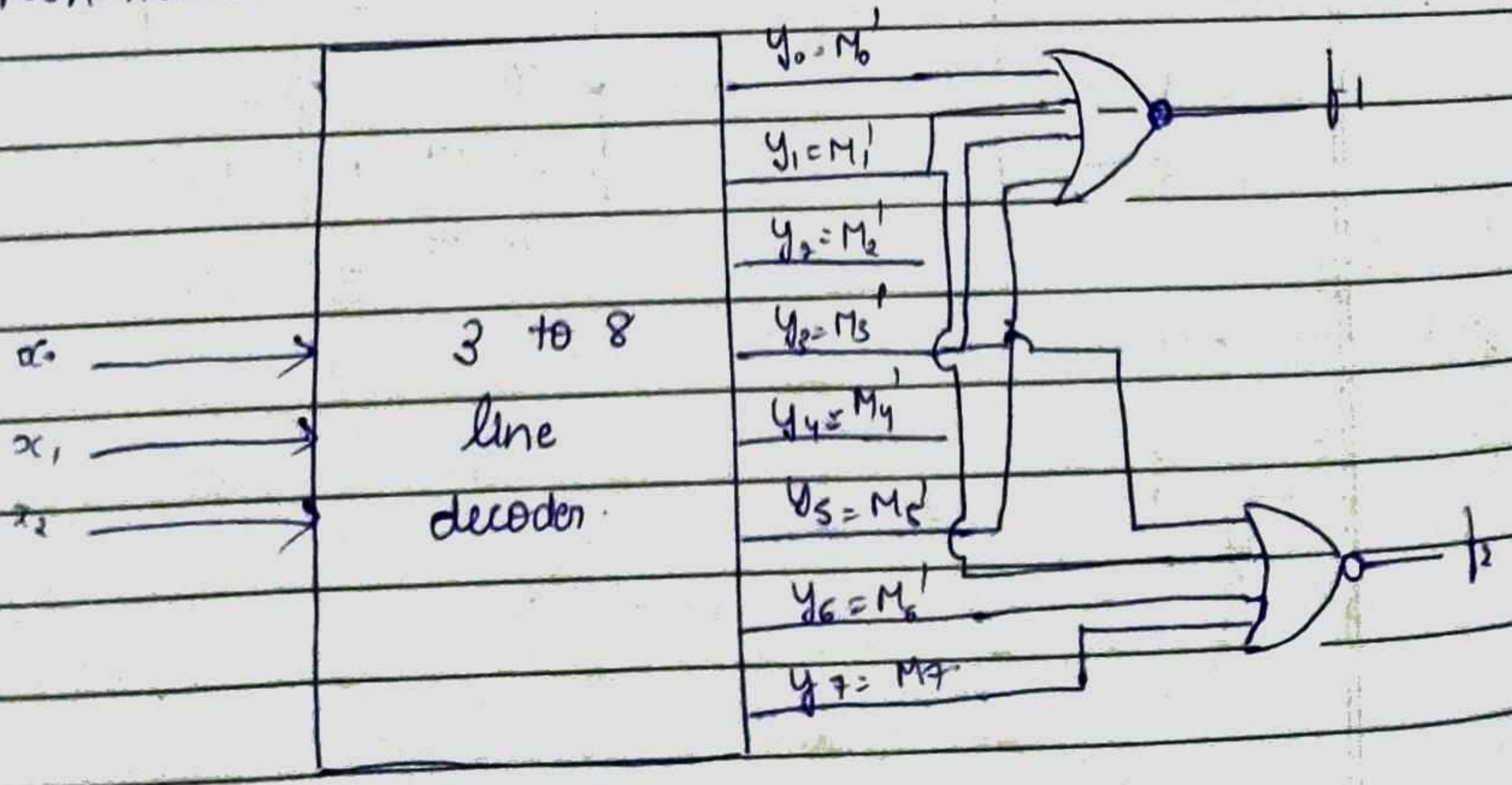
$$f_1 = \pi M(0, 1, 3, 5) = \sum m(2, 4, 6, 7)$$

$$f_2 = \pi M(1, 3, 6, 7) = \sum m(0, 2, 4, 5)$$



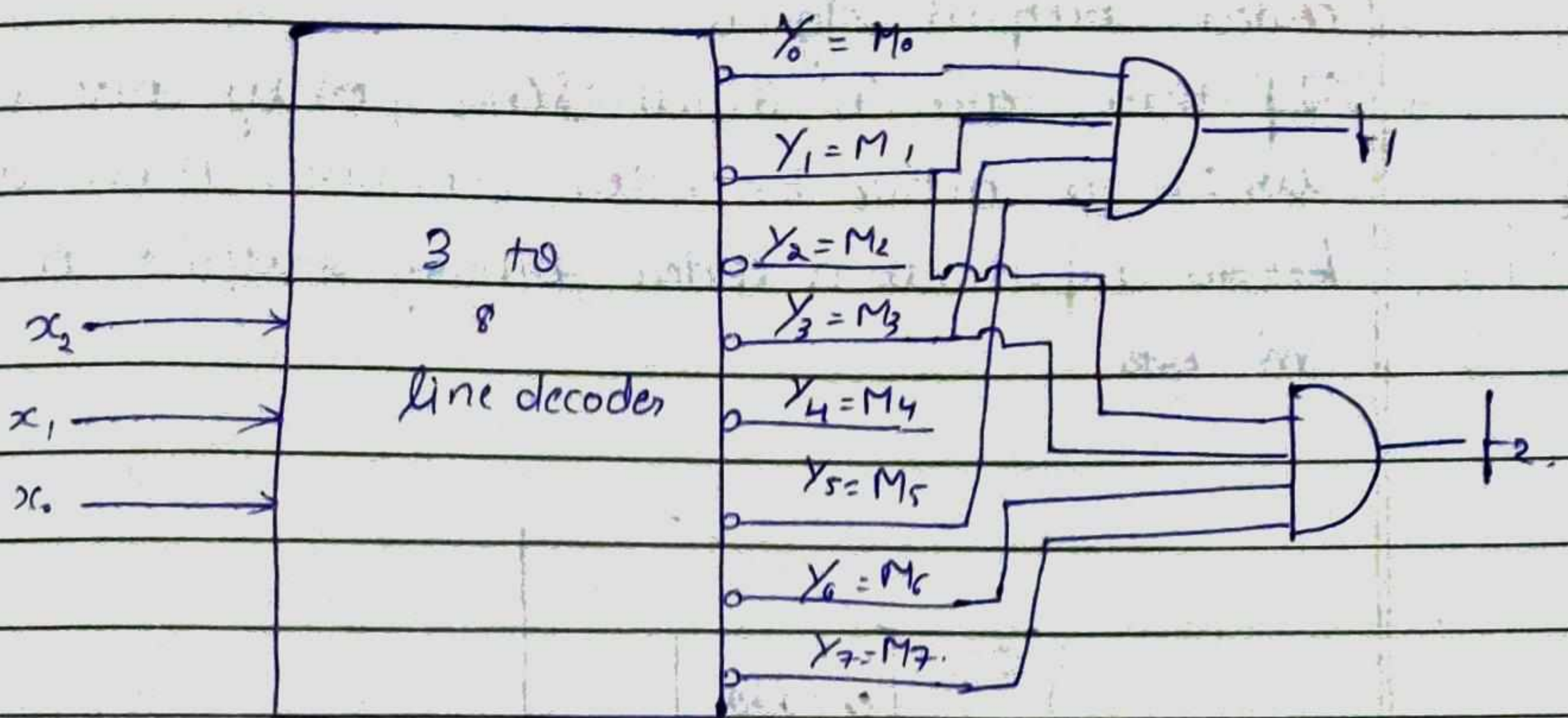
ii) Using NOR gates.

Non-inverted maxterms.



iii) Using AND gates.

Inverted maxterms.

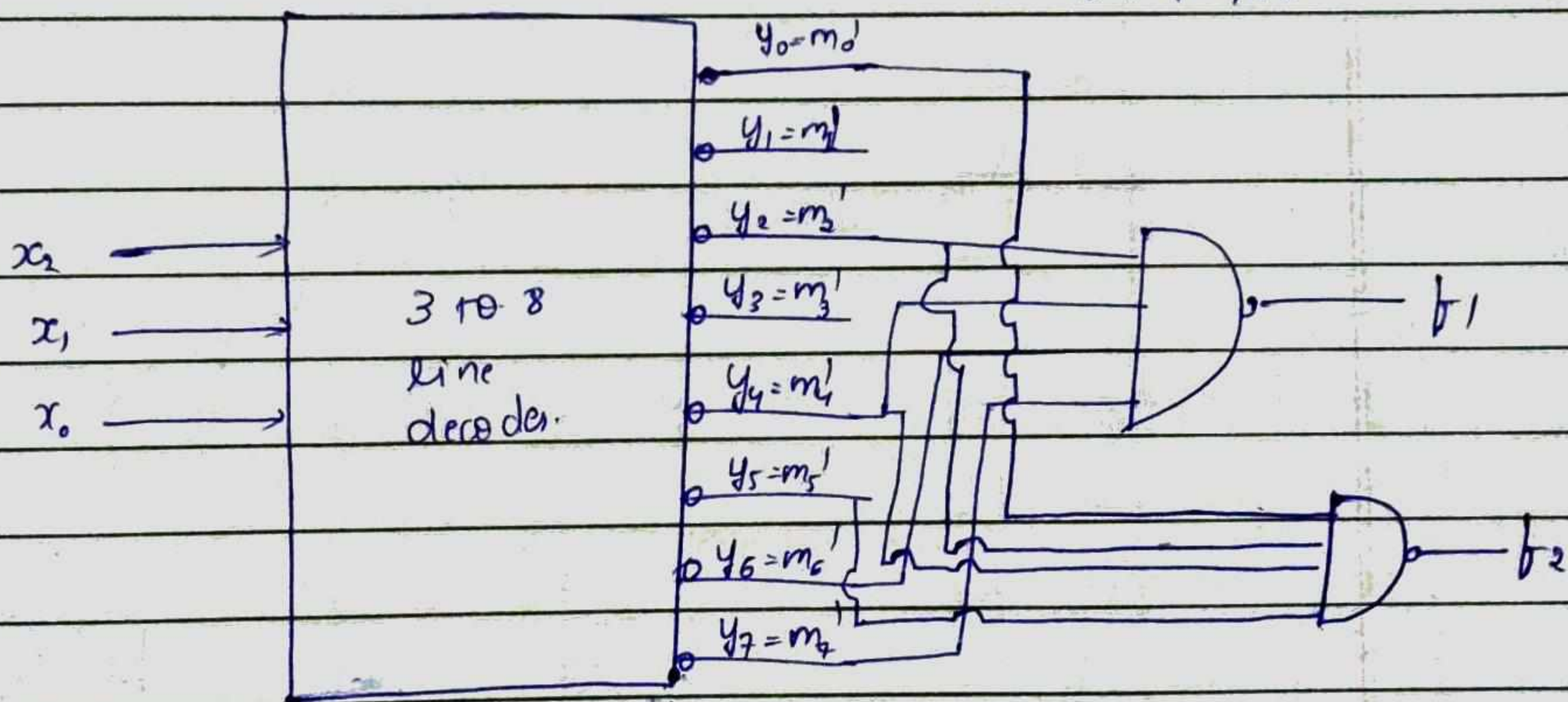


iv) Using Nand gates.

Inverted minterms.

$$f_1 = \sum M(2, 4, 6, 7)$$

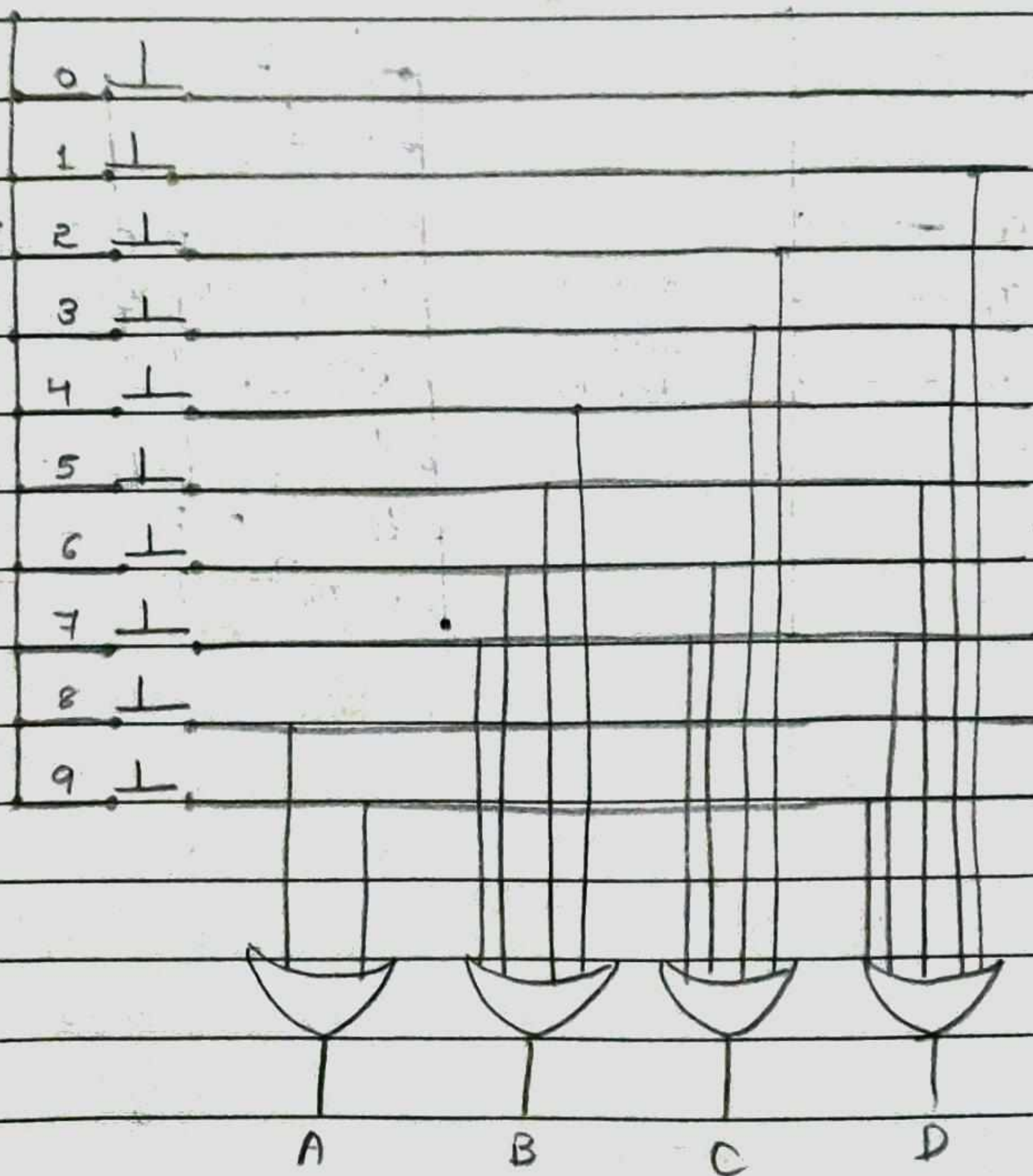
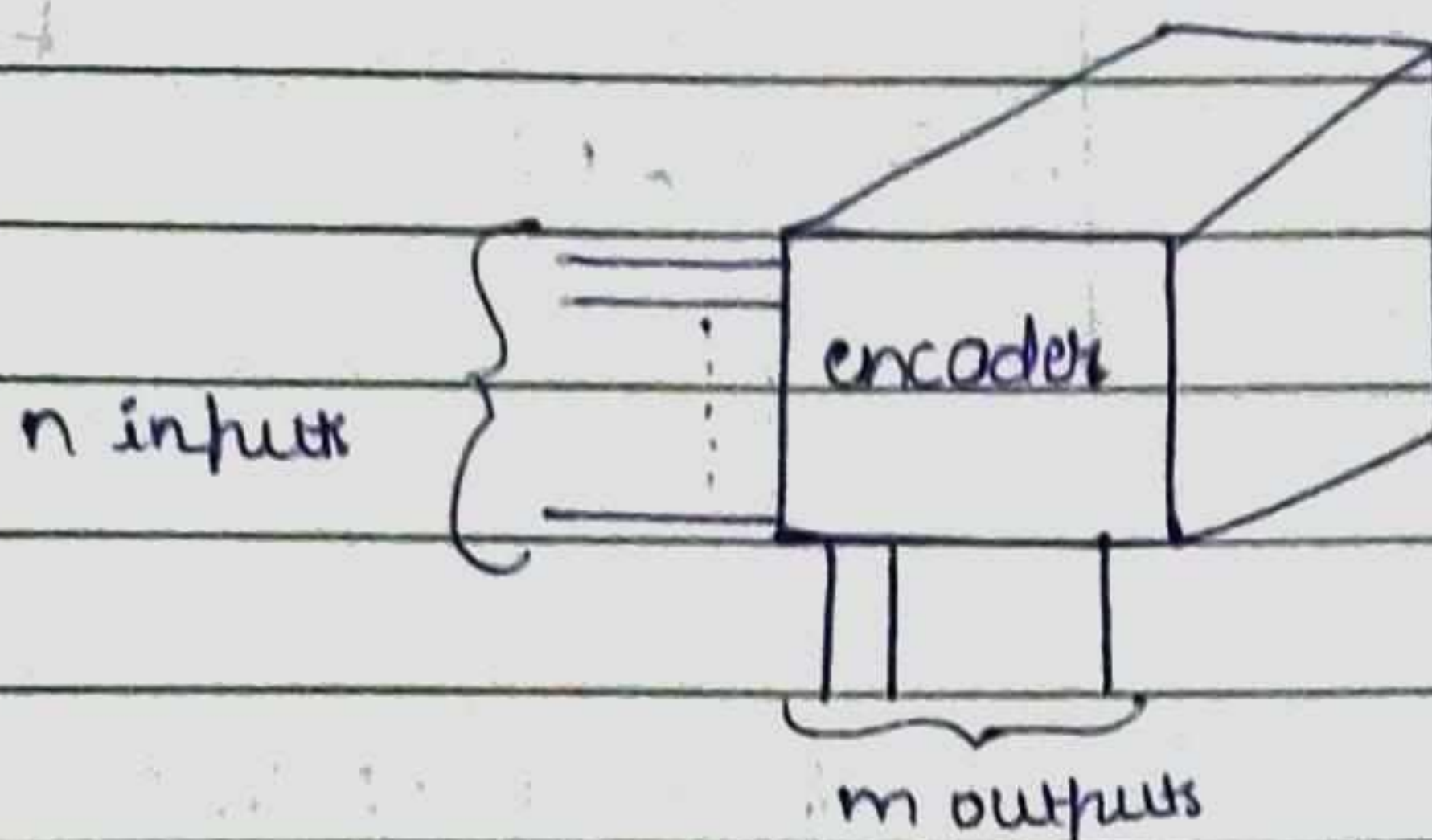
$$f_2 = \sum m(0, 2, 4, 5)$$



⇒ Encoder

It converts all active input signals into a coded output signal.

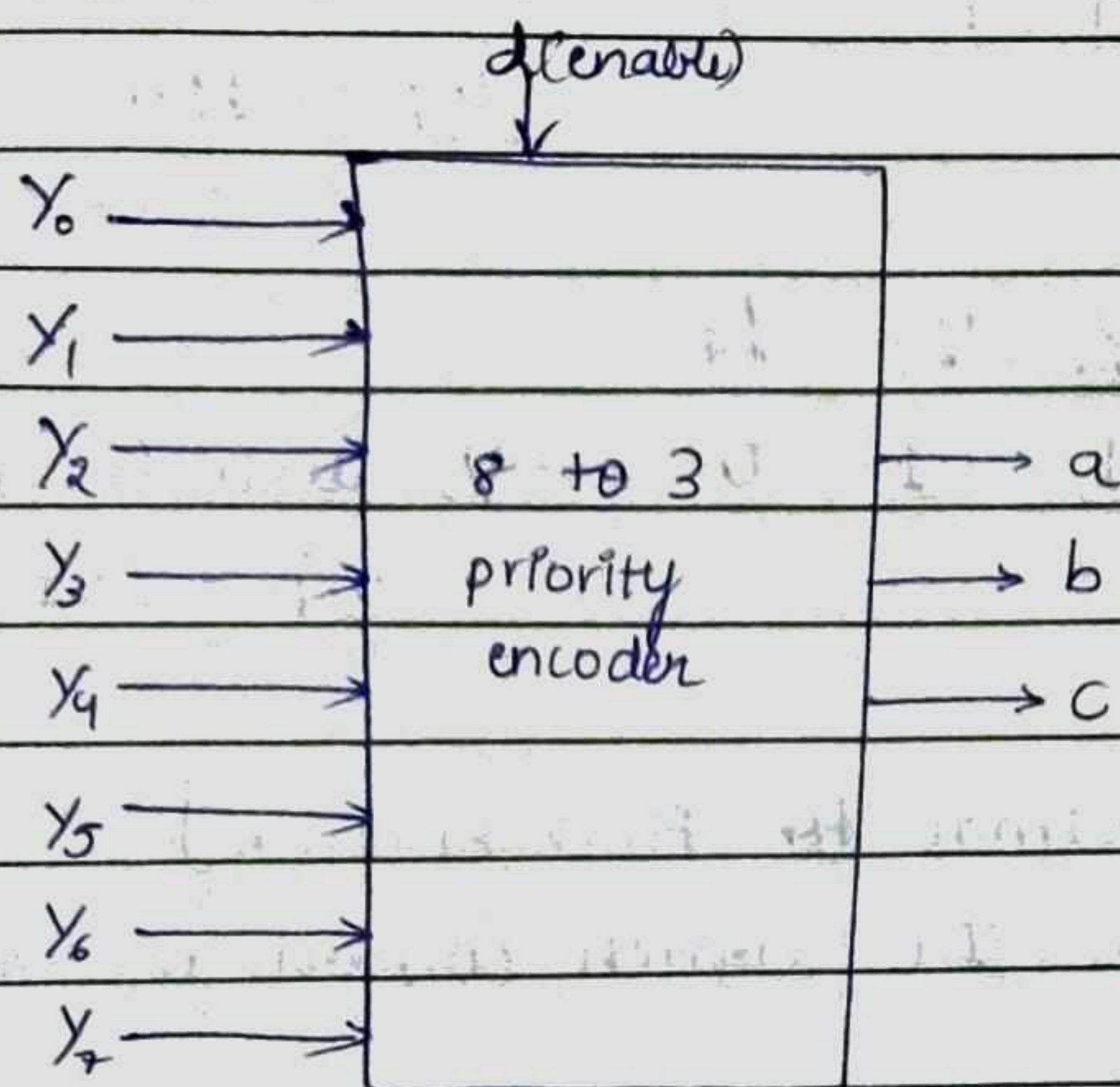
If there are 'n' input lines only one of which is active. Encoder converts this active ~~Encore~~ input to a coded binary output with 'm' bits.



Ex. If Switch 9 is pressed, the values for ABCD will be 1001.

→ 8 to 3 priority Encoder - It gives priority to highest encoded input.

It will work only when enable bit (d) is 1



Truth table.

I/P's								O/P's			enable.
Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	a	b	c	d
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
x	1	0	0	0	0	0	0	0	0	1	1
x	x	1	0	0	0	0	0	0	1	0	1
x	x	x	1	0	0	0	0	0	1	1	1
x	x	x	x	1	0	0	0	1	0	0	1
x	x	x	x	x	1	0	0	1	0	1	1
x	x	x	x	x	x	1	0	1	1	0	1
x	x	x	x	x	x	x	1	1	1	1	1

- If more than one input is 1, then the highest numbered input determines the output.
- In priority encoder, the output will be defined using the priority scheme.

ex.

i) y_0, y_1, y_5, y_6

1 0 0 1 → y_0 will be encoded.

O/p = 110

ii) y_0, y_1, y_2, y_5, y_7

0 1 1 1 0 → y_5 will be encoded.

O/p = 101.

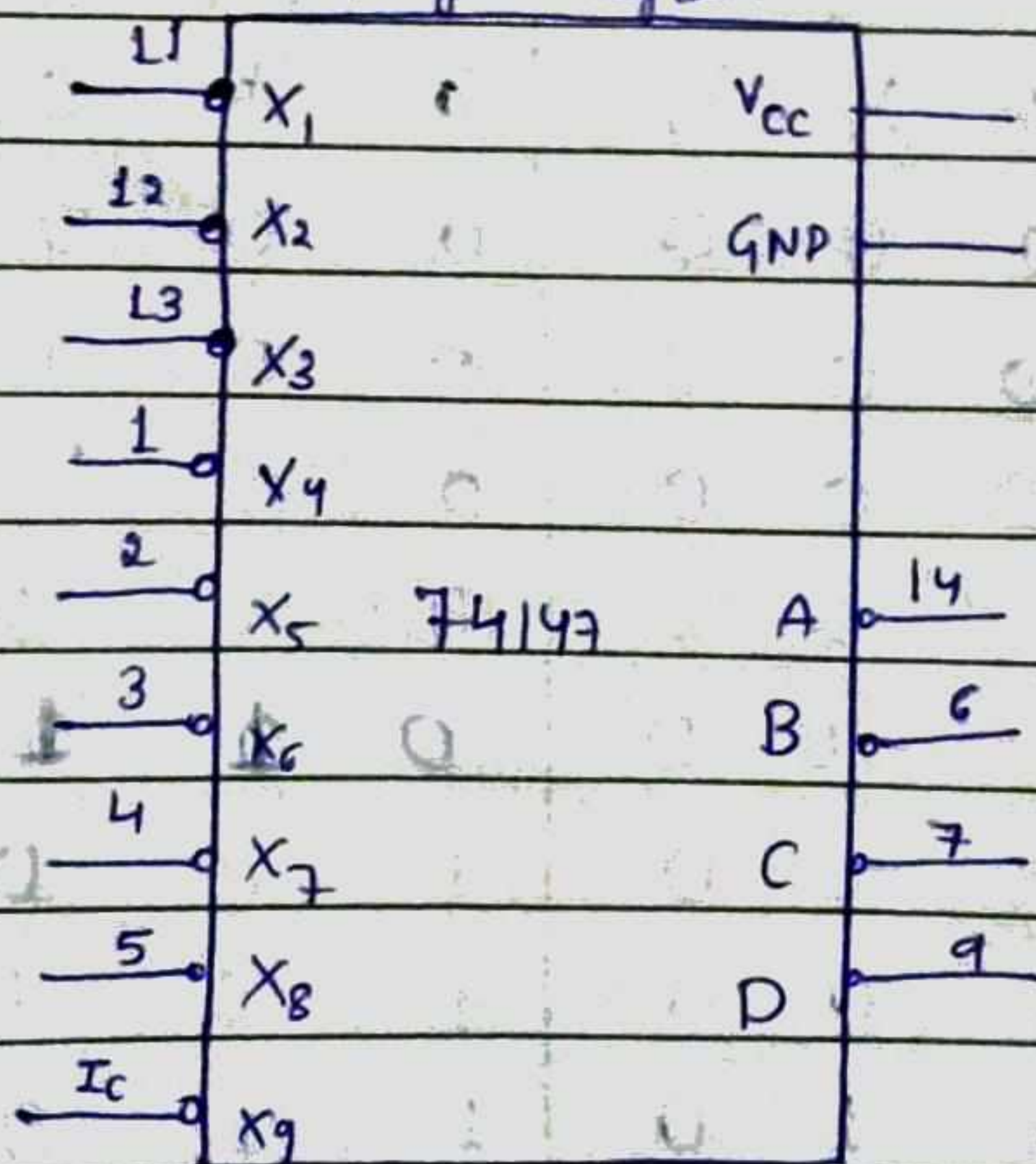
⇒ 74147 [Decimal to BCD Encoder]

74147 is an IC which converts decimal to BCD Encoder.

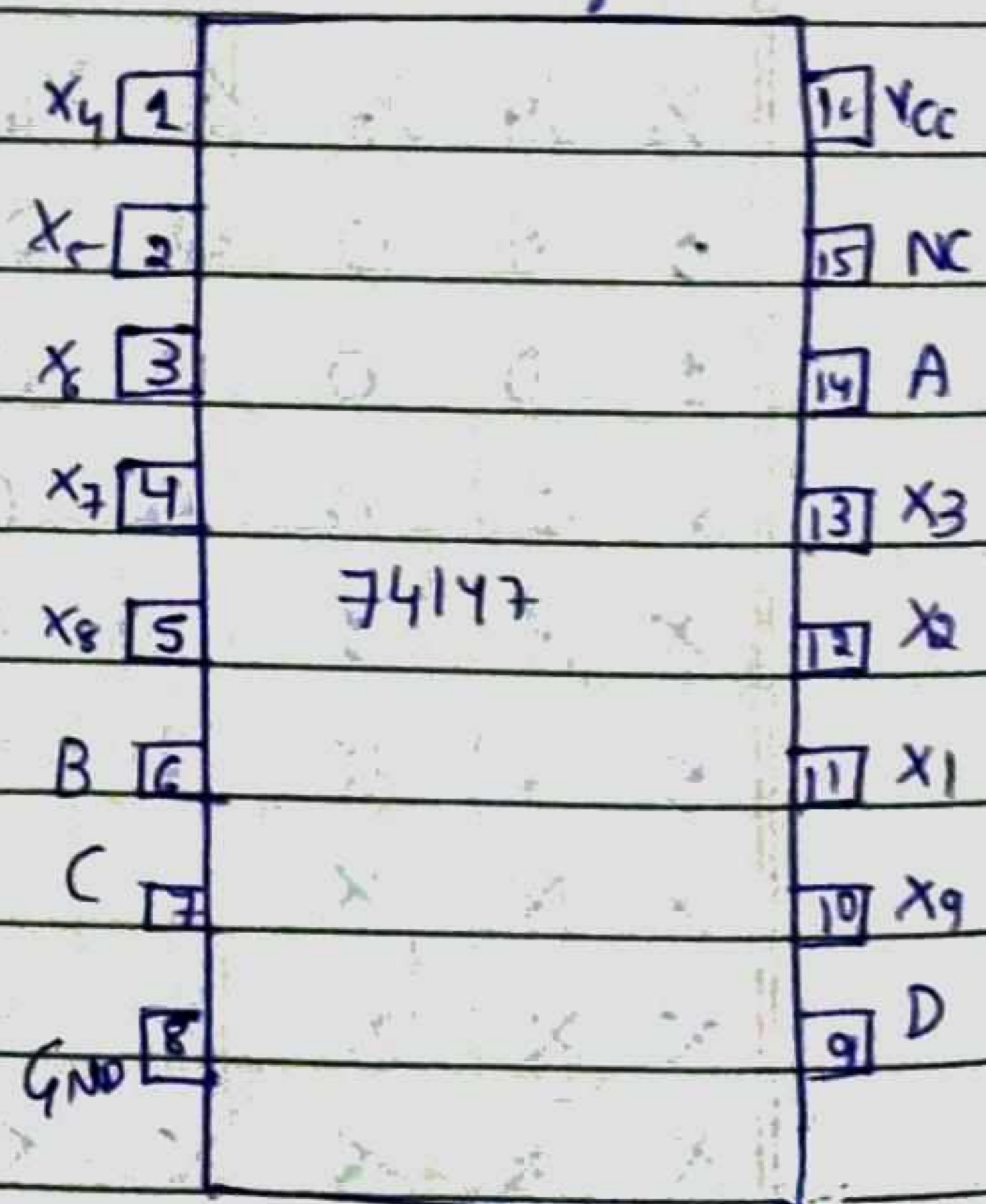
It is a 10 to 4 priority encoder.

It is a priority encoder.

Logic Diagram



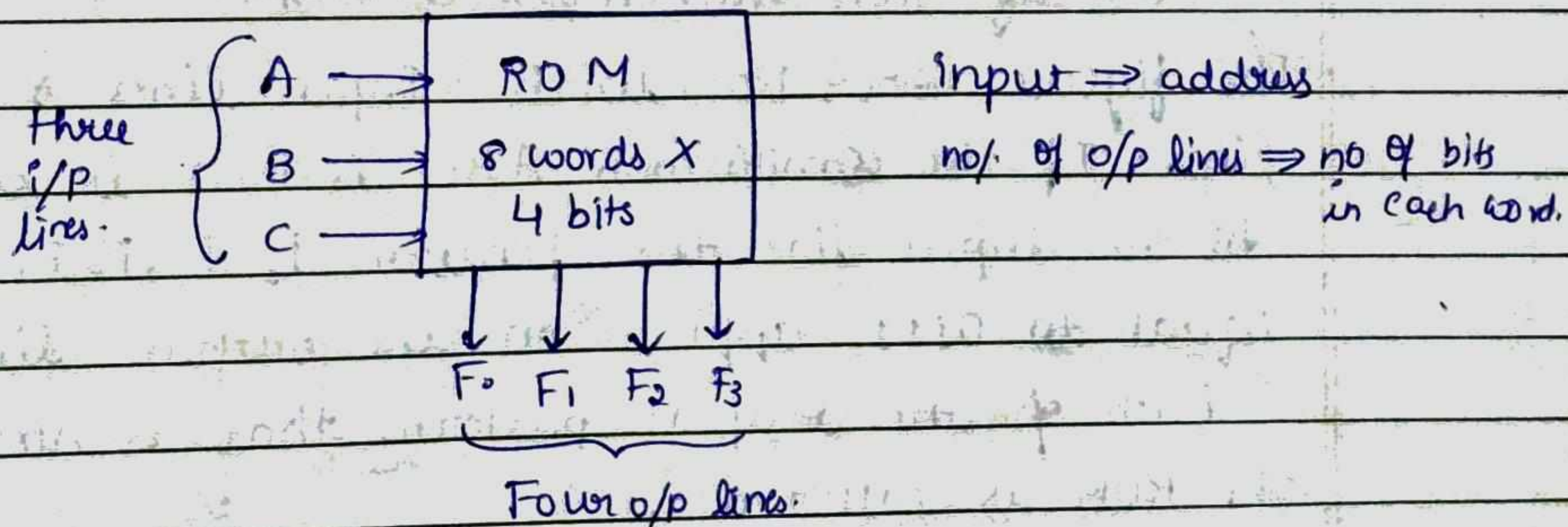
Pinout diagram



x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	A	B	C	D
H	H	H	H	H	H	H	H	H	H	H	H	H
X	X	X	X	X	X	X	X	L	L	H	H	L
X	X	X	X	X	X	X	L	H	L	H	H	H
X	X	X	X	X	X	L	H	H	H	L	L	L
X	X	X	X	X	L	H	H	H	H	L	L	H
X	X	X	X	L	H	H	H	H	H	L	H	L
X	X	X	L	H	H	H	H	H	H	L	H	H
X	X	L	H	H	H	H	H	H	H	H	L	L
X	L	H	H	H	H	H	H	H	H	H	L	H
L	H	H	H	H	H	H	H	H	H	H	H	L

⇒ Read only Memory

ROM consists of an array of semiconductor devices that are interconnected to store an array of (given) binary data. Once the data is stored in the ROM it can be read.



Truth Table

A	B	C	F ₀	F ₁	F ₂	F ₃
0	0	0	1	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	1	1
0	1	1	0	1	0	1
1	0	0	1	1	0	0
1	0	1	0	0	0	1
1	1	0	1	1	1	1
1	1	1	0	1	0	1

Typical data
stored in ROM
(2^3 words - 4 bits
each)

→ General Structure

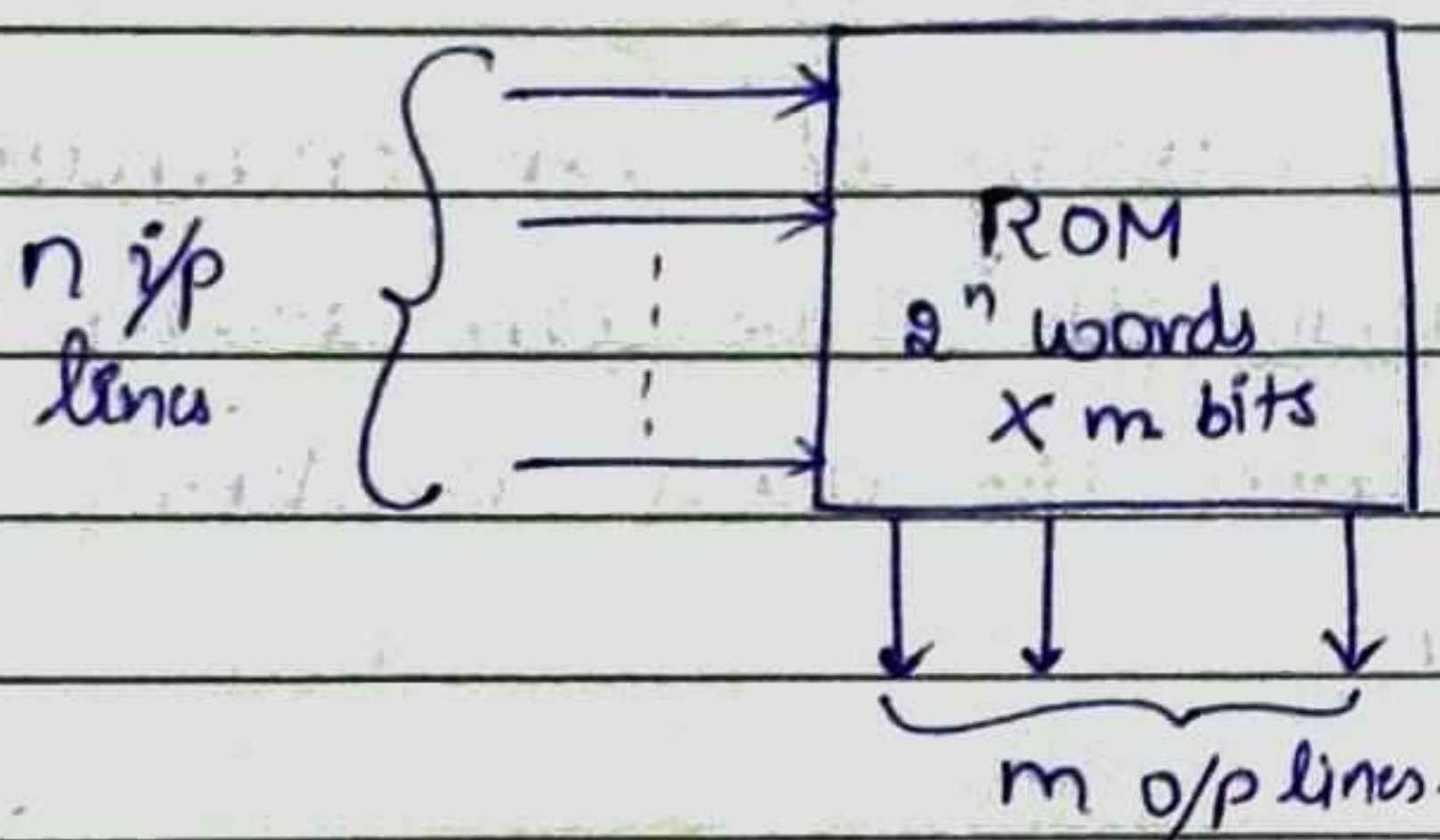


fig: ROM with 'n' i/p's & 'm' o/p's.

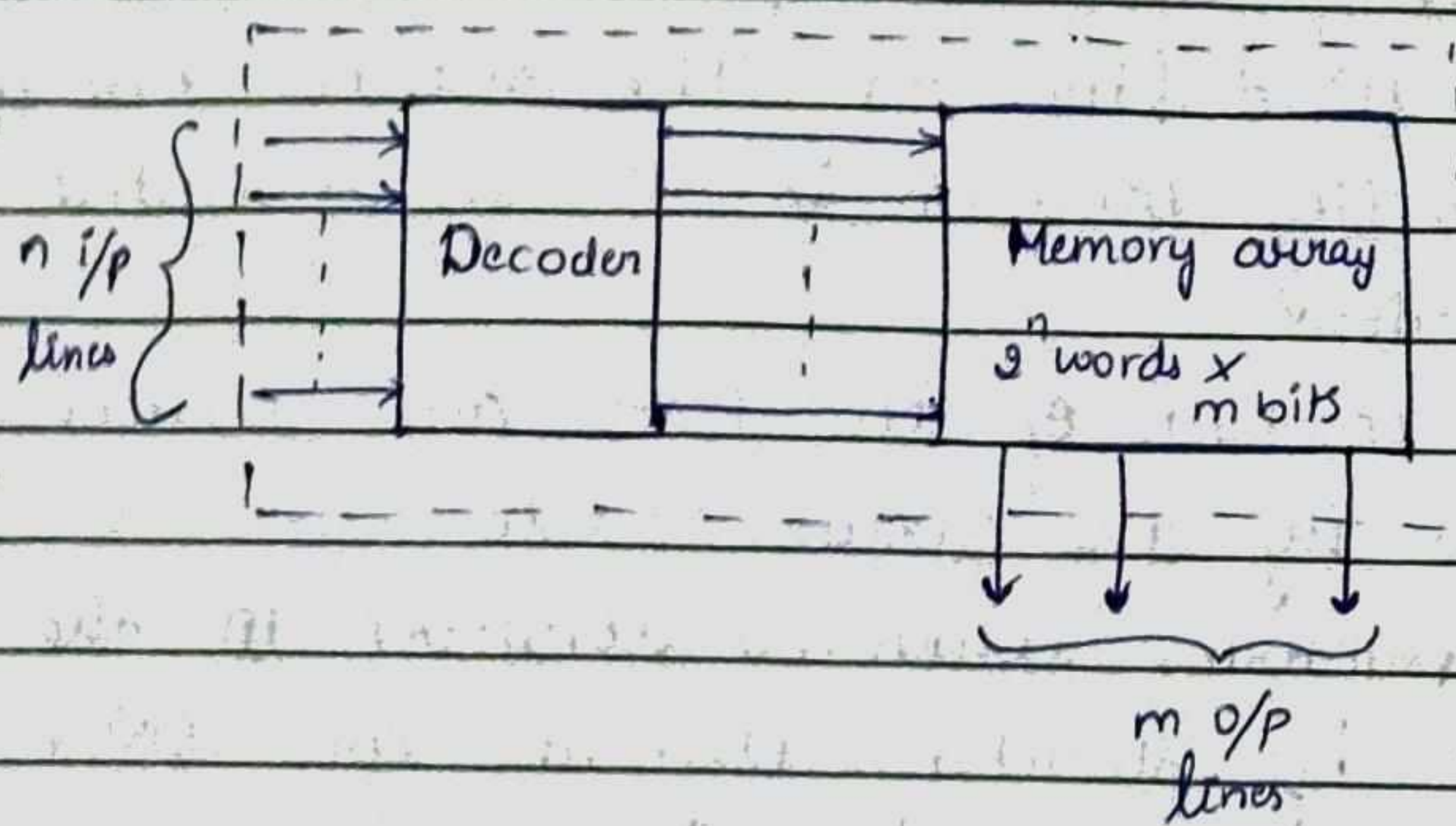
Explanation
For previous diagram (ROM)
The figure shows ROM with 3 input lines & 4 output lines. If the combination ABC is 010 is applied to the input line the pattern F₀, F₁, F₂, F₃ is equal to 0111 appears on the output lines.

Each of the output patterns that is stored in the ROM is called a word.

ROM has 3 i/p lines, $2^3 = 8$ different combinations of input values. Each combination serves as an address which can result one of the eight words stored in the memory.

Since there are 4 o/p lines, each word is 4 bit long & the size of the ROM is 8 words x 4 bits

Basic ROM structure

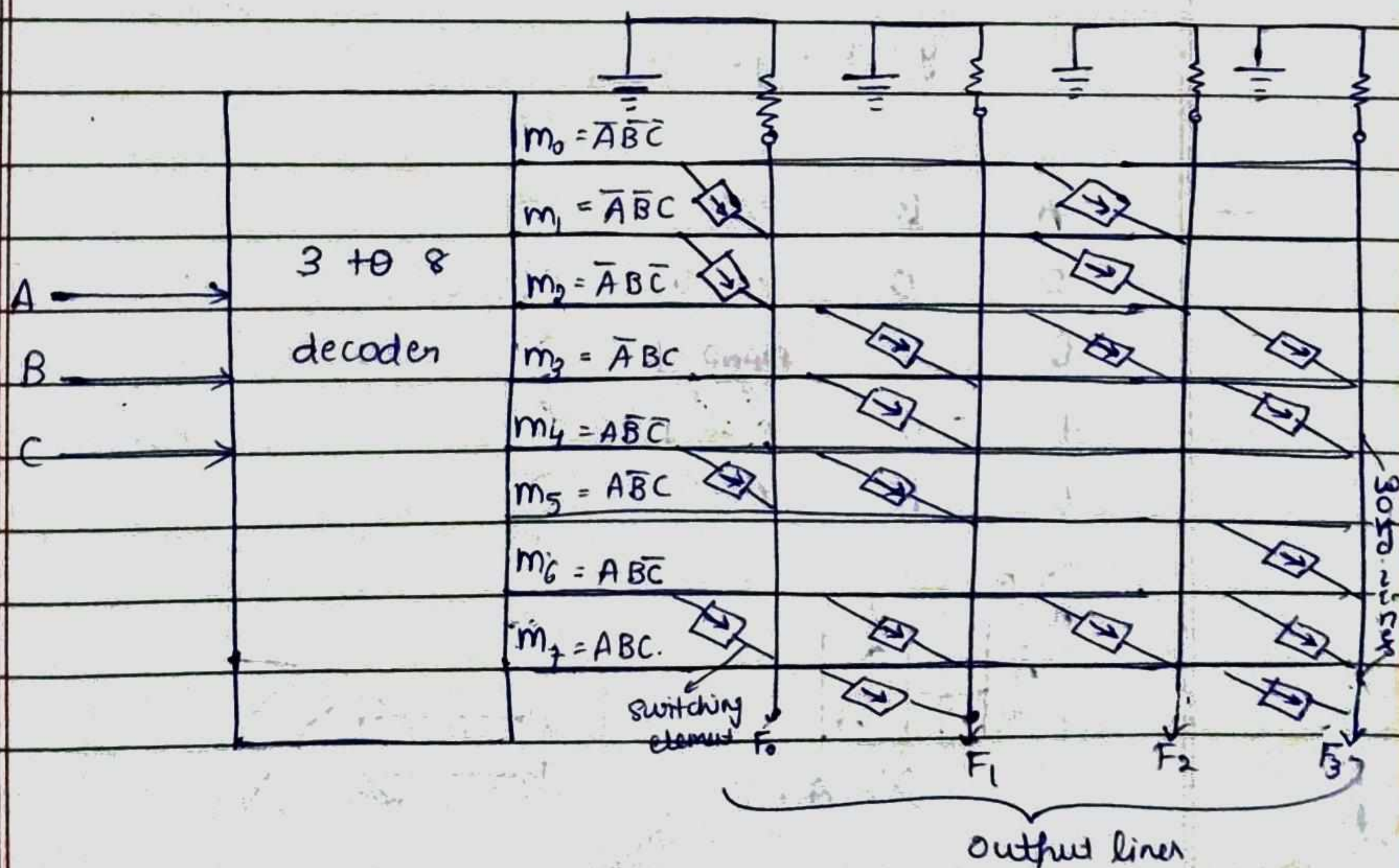


A ROM typically consists of a decoder & a memory array.

When a pattern of ' n ' zeroes & ones are applied to the decoder, exactly one of the 2^n lines decoder^{lines (o/p)} is one.

The decoder results one of the words in the memory array & the bit pattern stored in this word is transferred to the memory output lines.

Internal structure of 8 word X 4 bit ROM



(31)

The decoder generates 8 minterms of 3 i/p variables. A switching element is placed at the intersection of the word line & the output line if the corresponding minterm is to be included in the o/p function.

The contents of the ROM are usually specified by the truth table.

The minterms which are connected to the o/p line F by switching elements are ORed together to form the output F .

$$F_0 = \sum m(0, 1, 4, 6) = \bar{A}\bar{B} + A\bar{C}$$

$$F_1 = \sum m(2, 3, 4, 6, 7) = B + A\bar{C}$$

$$F_2 = \sum m(0, 1, 2, 6) = \bar{A}\bar{B} + B\bar{C}$$

$$F_3 = \sum m(2, 3, 5, 6, 7) = AC + B$$

Multiple output combination circuits can be realised using ROM's.

⇒ Realise a code converter that converts a 4 bit binary number to a hexadecimal digit & outputs the 7 bit ASCII code. [The ASCII value of the hexadecimal zero is 48]

Next Page.

A → 6s

B

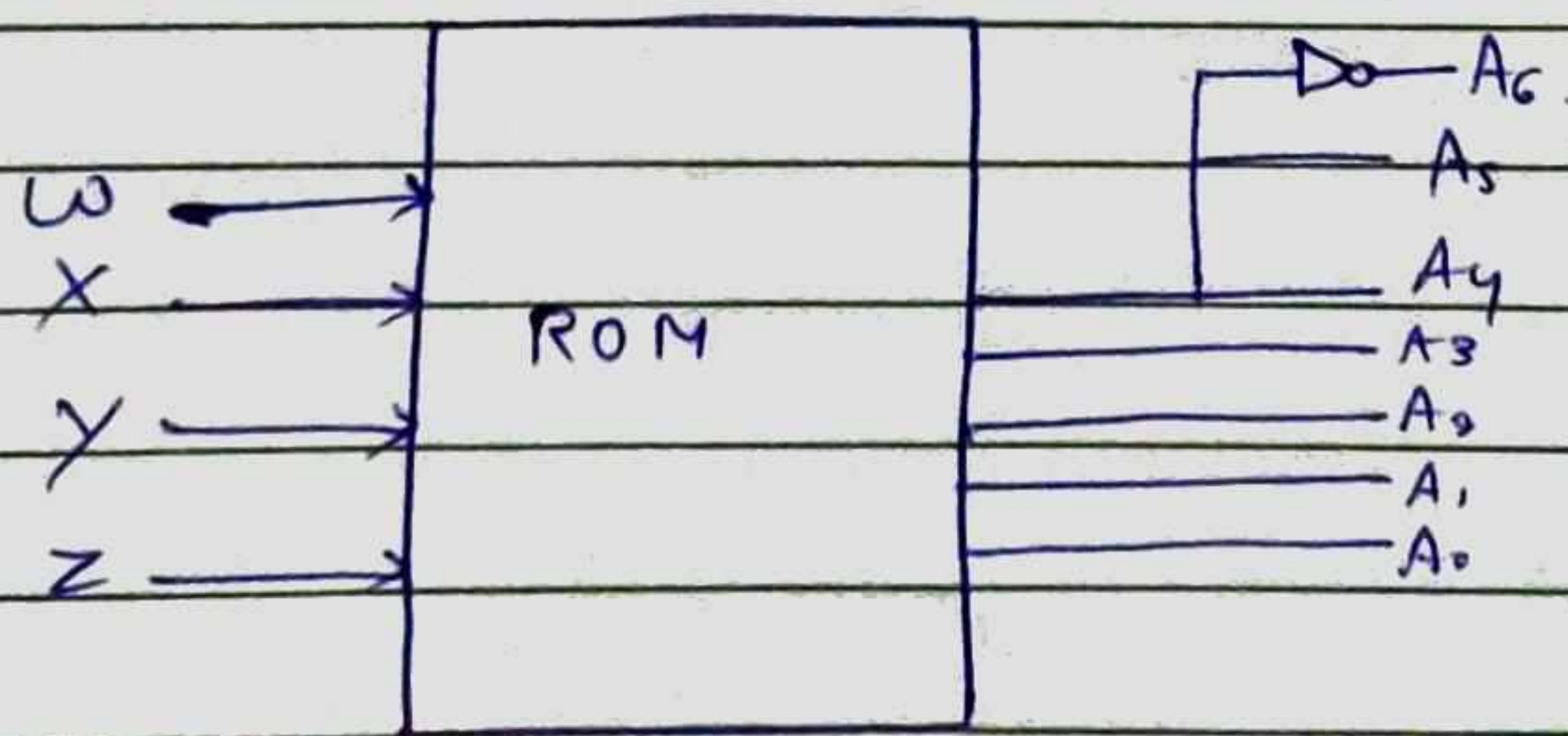
C

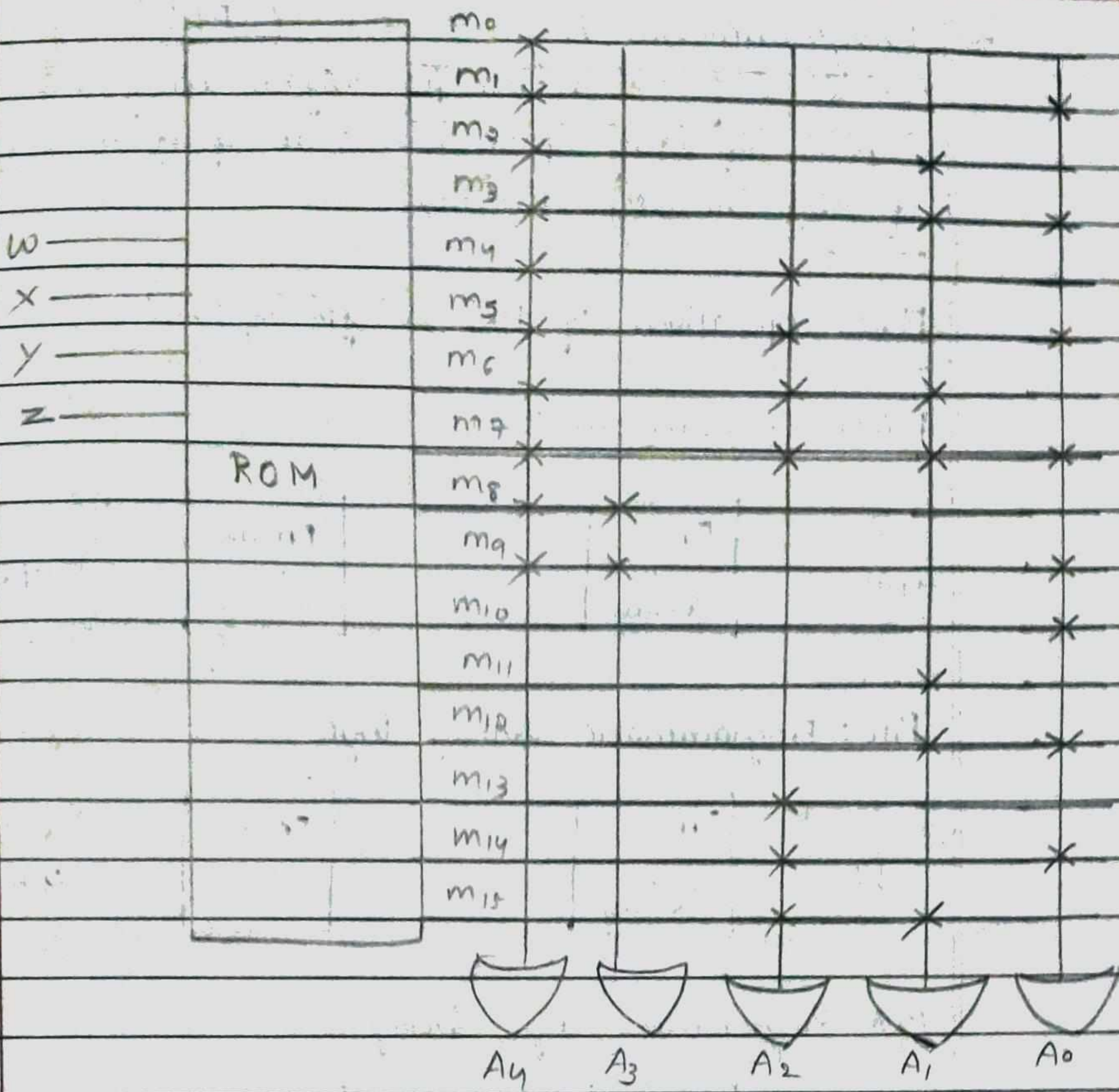
D

E

Truth Table

I/p				HEX	ASCII code for Hex digit						
W	X	Y	Z	DIGIT	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀
0	0	0	0	0	0	1	1	0	0	0	0
0	0	0	1	1	0	1	1	0	0	0	1
0	0	1	0	2	0	1	1	0	0	1	0
0	0	1	1	3	0	1	1	0	0	1	1
0	1	0	0	4	0	1	1	0	1	0	0
0	1	0	1	5	0	1	1	0	1	0	1
0	1	1	0	6	0	1	1	0	1	1	0
0	1	1	1	7	0	1	1	0	1	1	1
1	0	0	0	8	0	1	1	1	0	0	0
1	0	0	1	9	0	1	1	1	0	0	1
1	0	1	0	A	1	0	0	0	0	0	1
1	0	1	1	B	1	0	0	0	0	1	0
1	1	0	0	C	1	0	0	0	0	1	1
1	1	0	1	D	1	0	0	0	1	0	0
1	1	1	0	E	1	0	0	0	1	0	1
1	1	1	1	F	1	0	0	0	1	1	0





⇒ Types of ROM's:

1. Mask programmable ROM.
2. P-ROM [Programmable ROM]
3. Electrically Erasable Programmable ROM [EEPROM]

→ Mask Programmable ROM

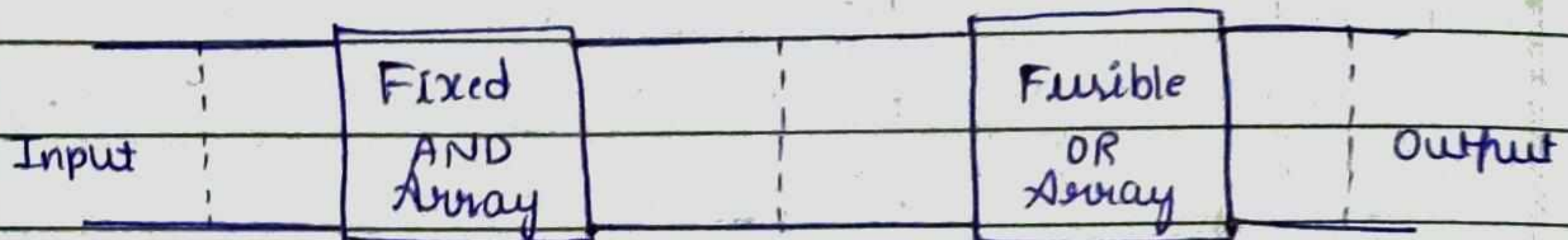
Here the data^{array} is permanently stored. This is accomplished by selectively including or omitting the switching elements at the row column intersections of the memory array. This requires the preparation of a special mask & it is expensive if a small quantity of ROM is required. To overcome this we will use EEPROM.

→ Programmable Logic devices (PLD)

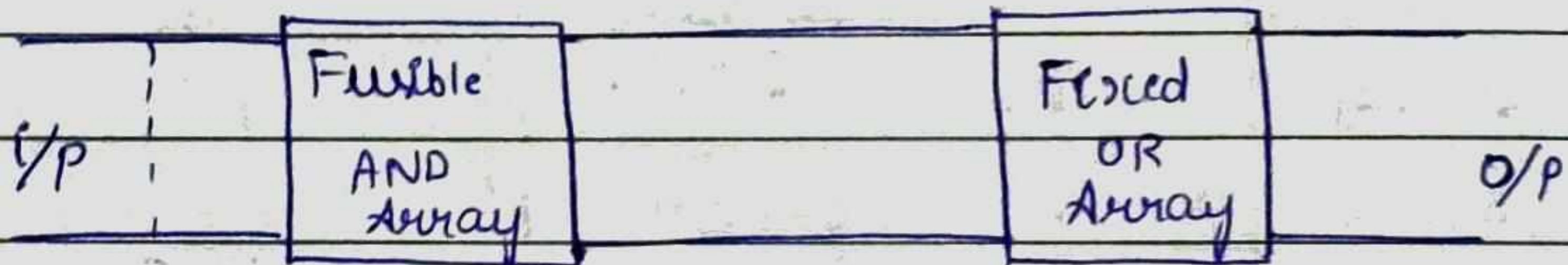
It is a general name for digital integrated circuit capable of being programmed to provide a variety of different logic functions.

Basic operations of PLD (Types of PLD)

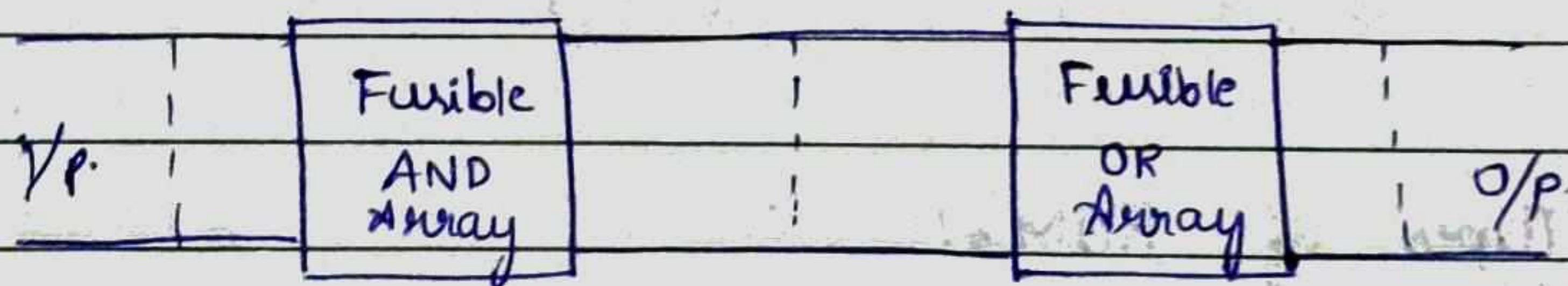
PROM : Programmable ROM



PAL : Programmable Array logic



PLA : Programmable logic Arrays



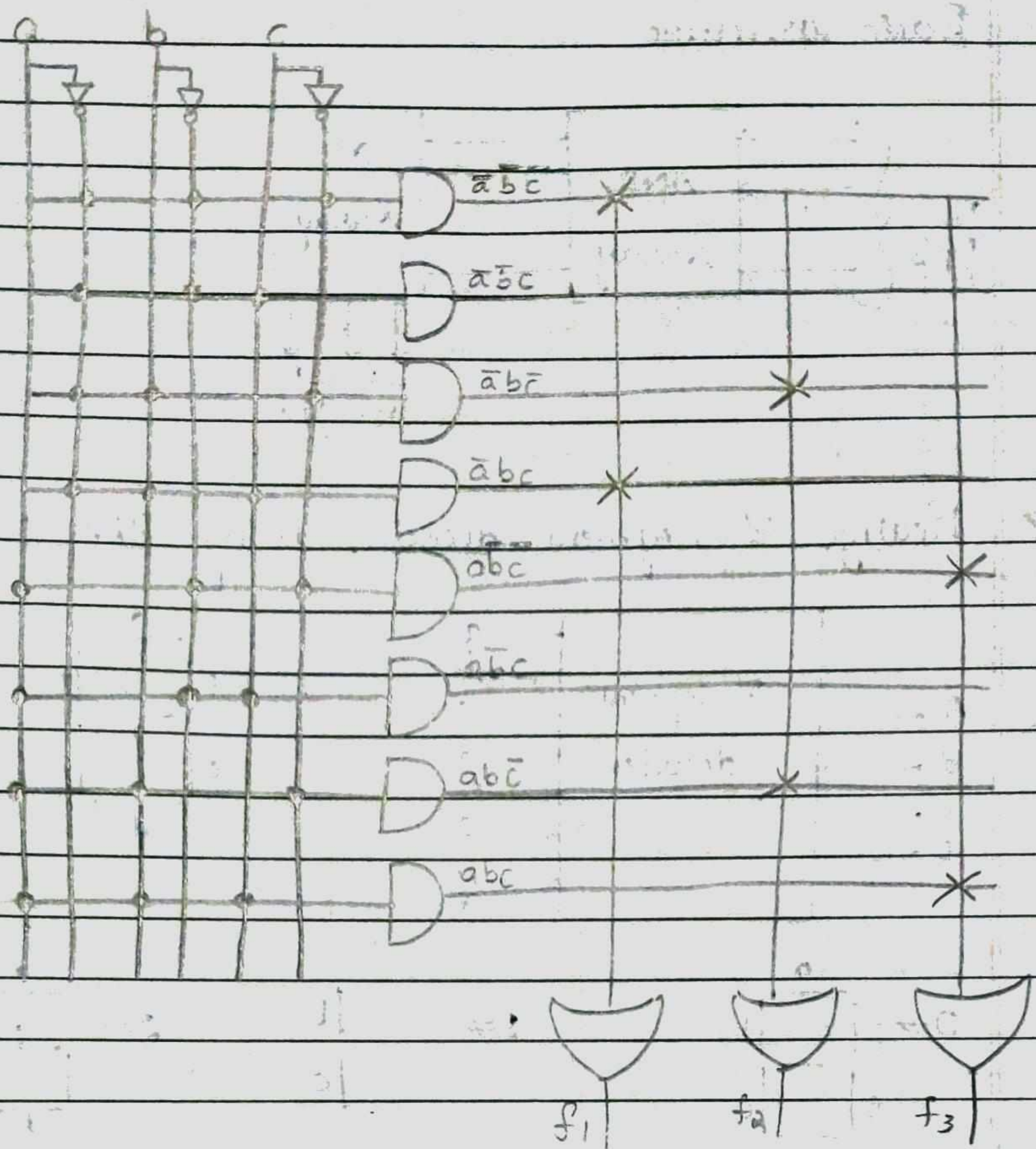
PROM : Fixed AND array - Fusible OR array

Ex:- 1) Realize the given SOP eqⁿ's using PROM

$$f_1(a, b, c) = \bar{a}\bar{b}c + \bar{a}bc$$

$$f_2(a, b, c) = \bar{a}b\bar{c} + abc$$

$$f_3(a, b, c) = abc + a\bar{b}\bar{c}$$

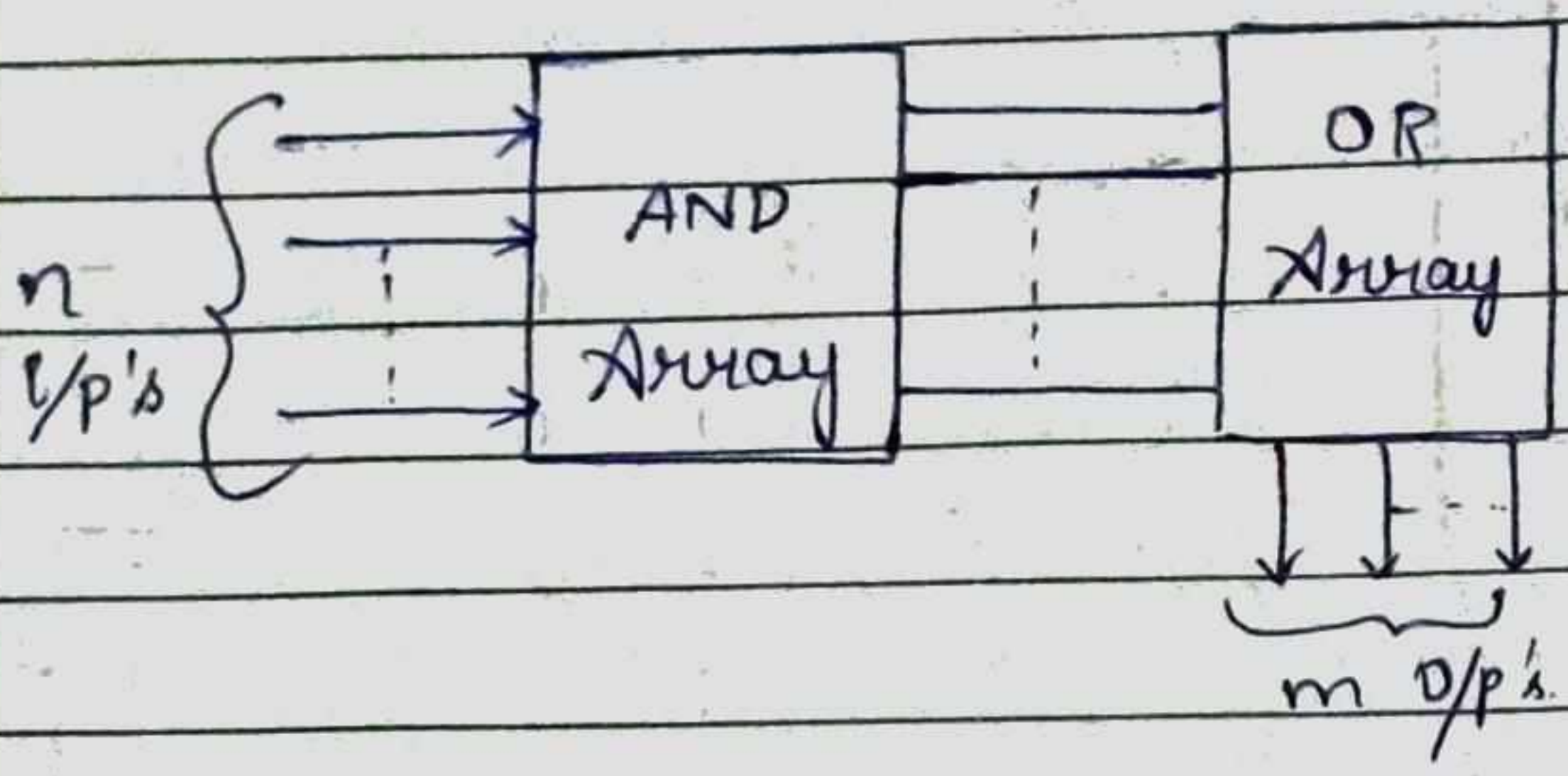


Programmable Logic Arrays

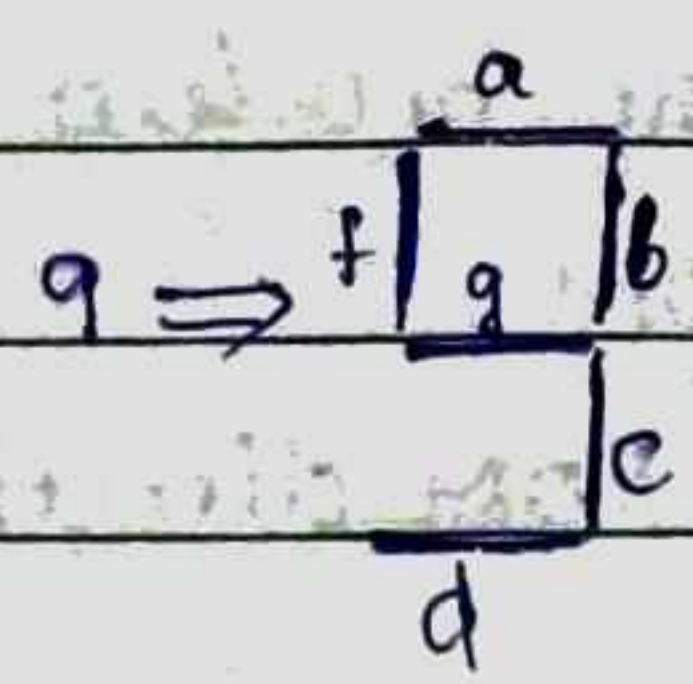
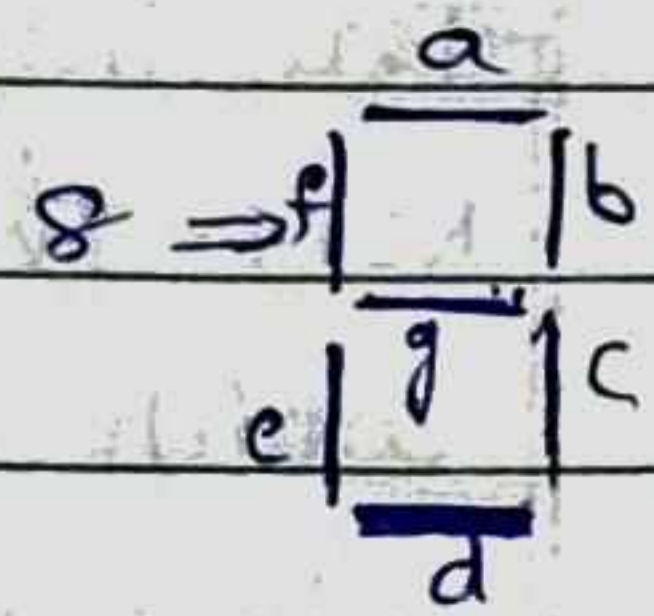
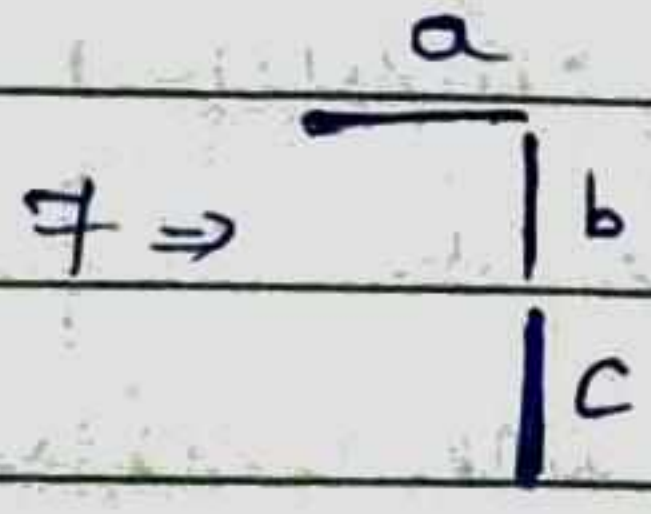
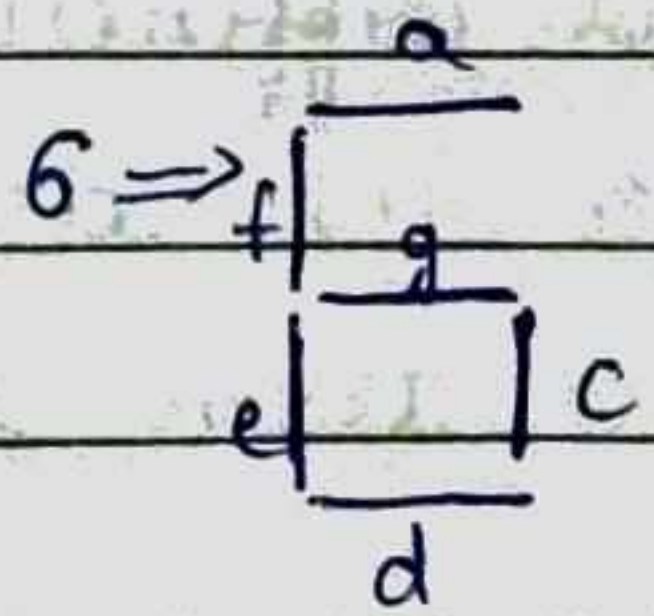
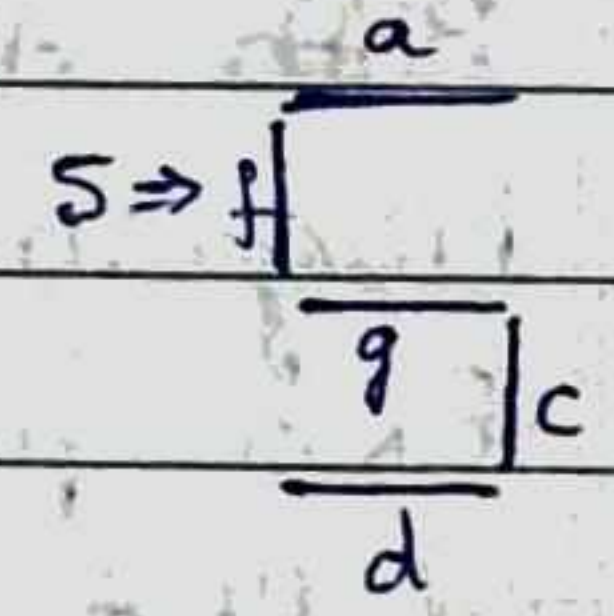
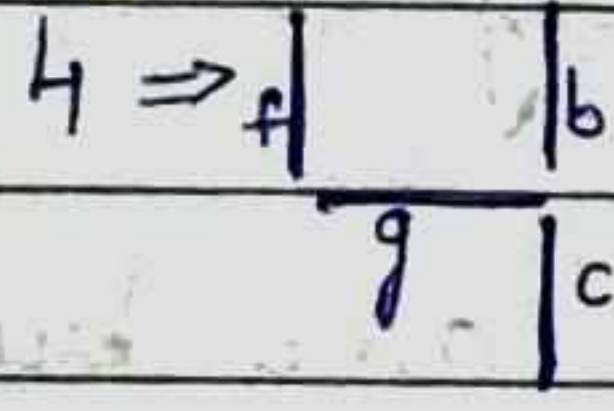
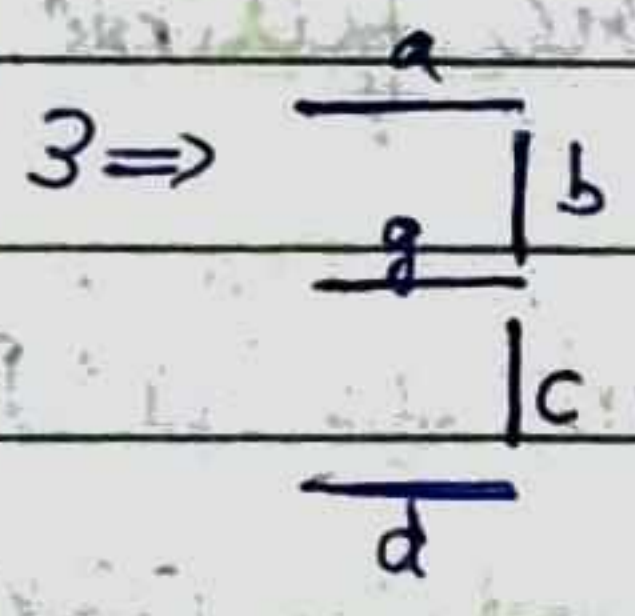
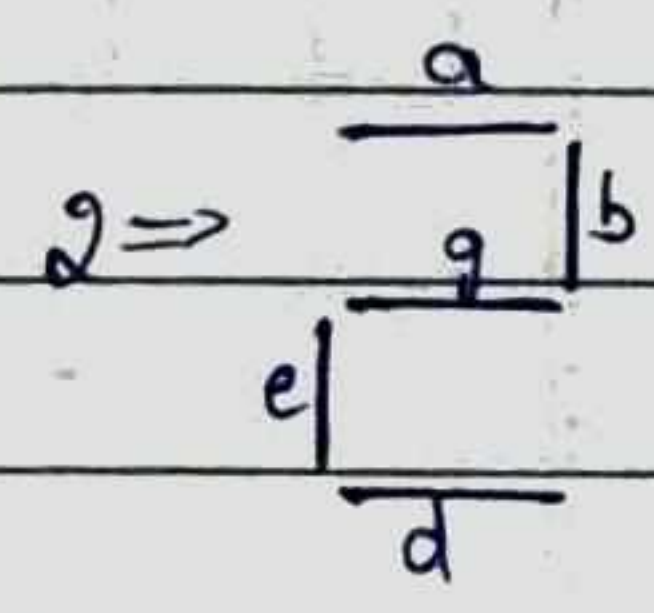
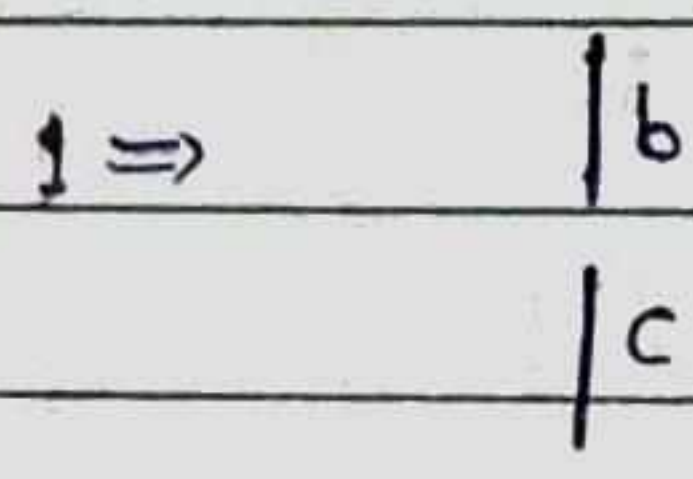
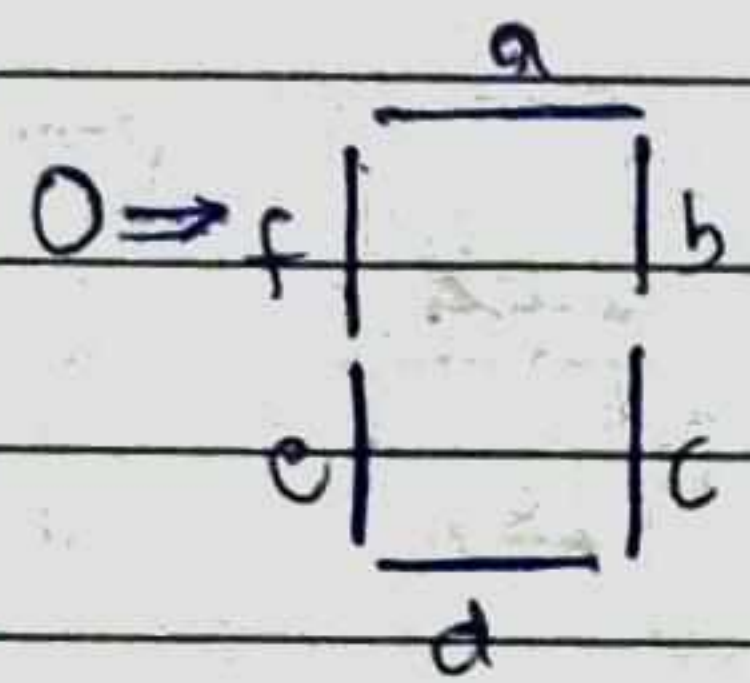
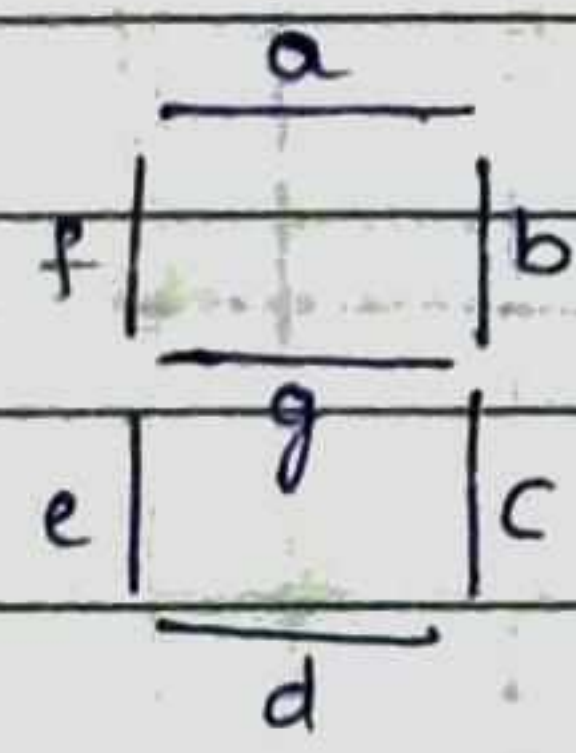
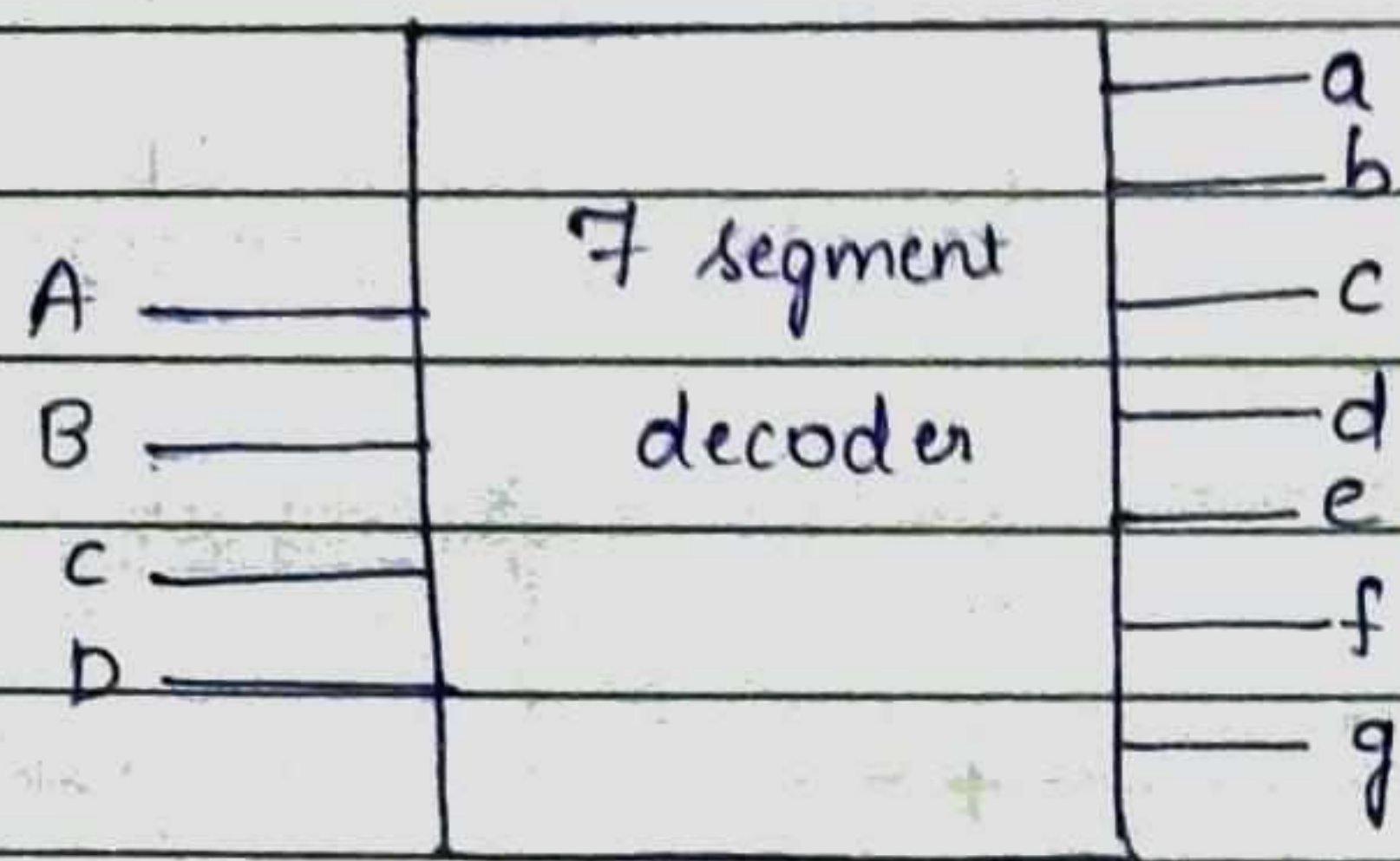
Both AND Array & OR Array are fusible or Programmable.

- PLA performs same basic function as a ROM.
- A PLA with n inputs & m outputs can realize m functions of n variables.
- The internal organization of PLA is different from ROM. The decoder is replaced with AND array which realizes the selected product term of the input variable.
- The OR array OR's together the product terms needed to form the output function.
- PLA implements the SOP expⁿ [ROM directly implements a truth table].

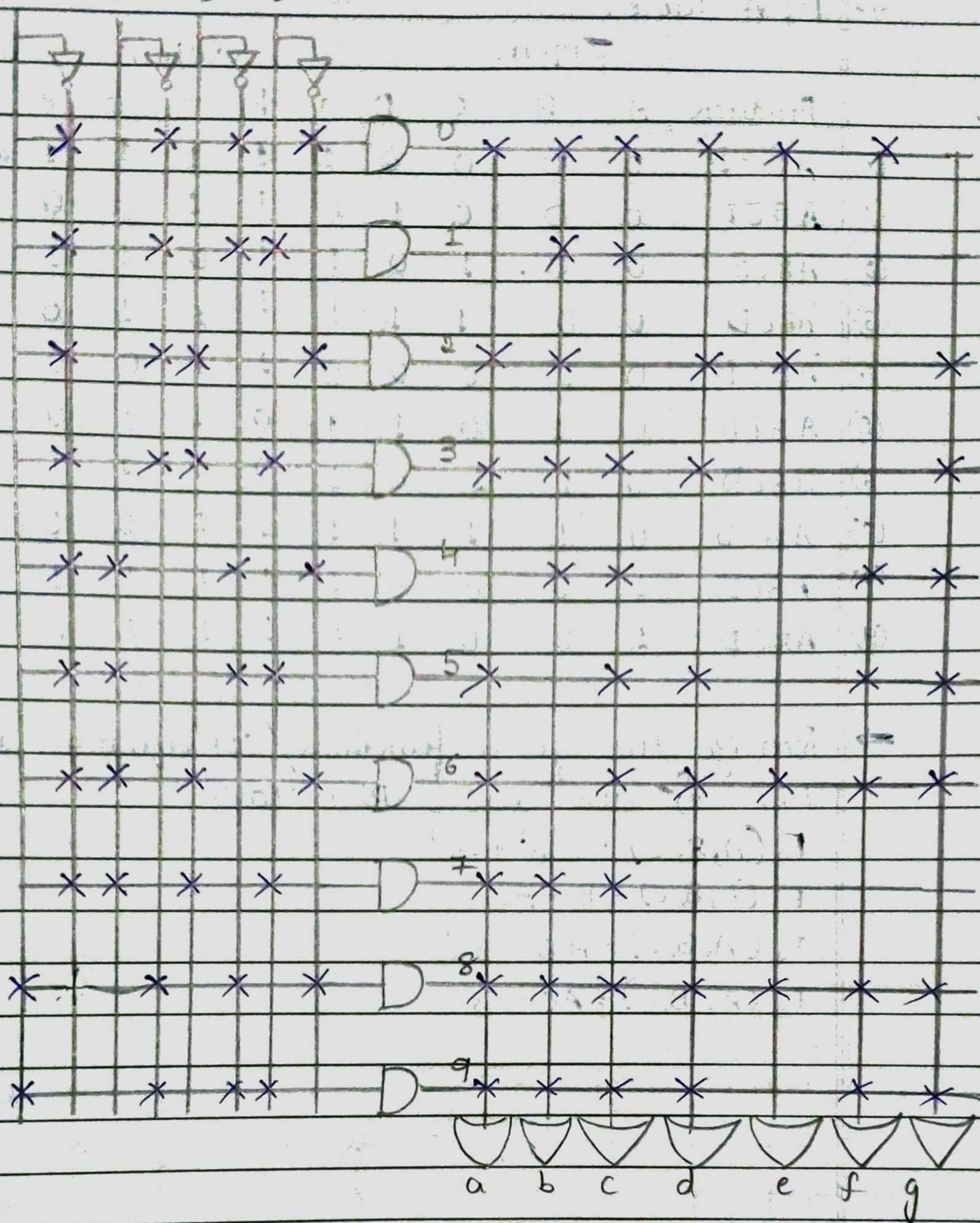
Basic structure



→ Realize 7 segment decoder using PLA.



A B C D



→ PLA Table:

	Products	Inputs				Outputs						
		A	B	C	D	a	b	c	d	e	f	g
(0)	$\bar{A}\bar{B}\bar{C}\bar{D}$	0	0	0	0	1	1	1	1	1	1	1
(1)	$\bar{A}\bar{B}\bar{C}D$	0	0	0	1	0	1	1	0	0	0	0
(2)	$\bar{A}\bar{B}C\bar{D}$	0	0	1	0	1	1	0	1	1	0	1
(3)	$\bar{A}\bar{B}CD$	0	0	1	1	1	1	1	1	0	0	1
(4)	$\bar{A}B\bar{C}\bar{D}$	0	1	0	0	0	1	1	0	0	1	1
(5)	$\bar{A}B\bar{C}D$	0	1	0	1	1	0	1	1	0	1	1
(6)	$\bar{A}BC\bar{D}$	0	1	1	0	1	0	1	1	1	1	1
(7)	$\bar{A}BCD$	0	1	1	1	1	1	1	0	0	0	0
(8)	$AB\bar{C}\bar{D}$	1	0	0	0	1	1	1	1	1	1	1
(9)	$AB\bar{C}D$	1	0	0	1	1	1	1	1	0	1	1

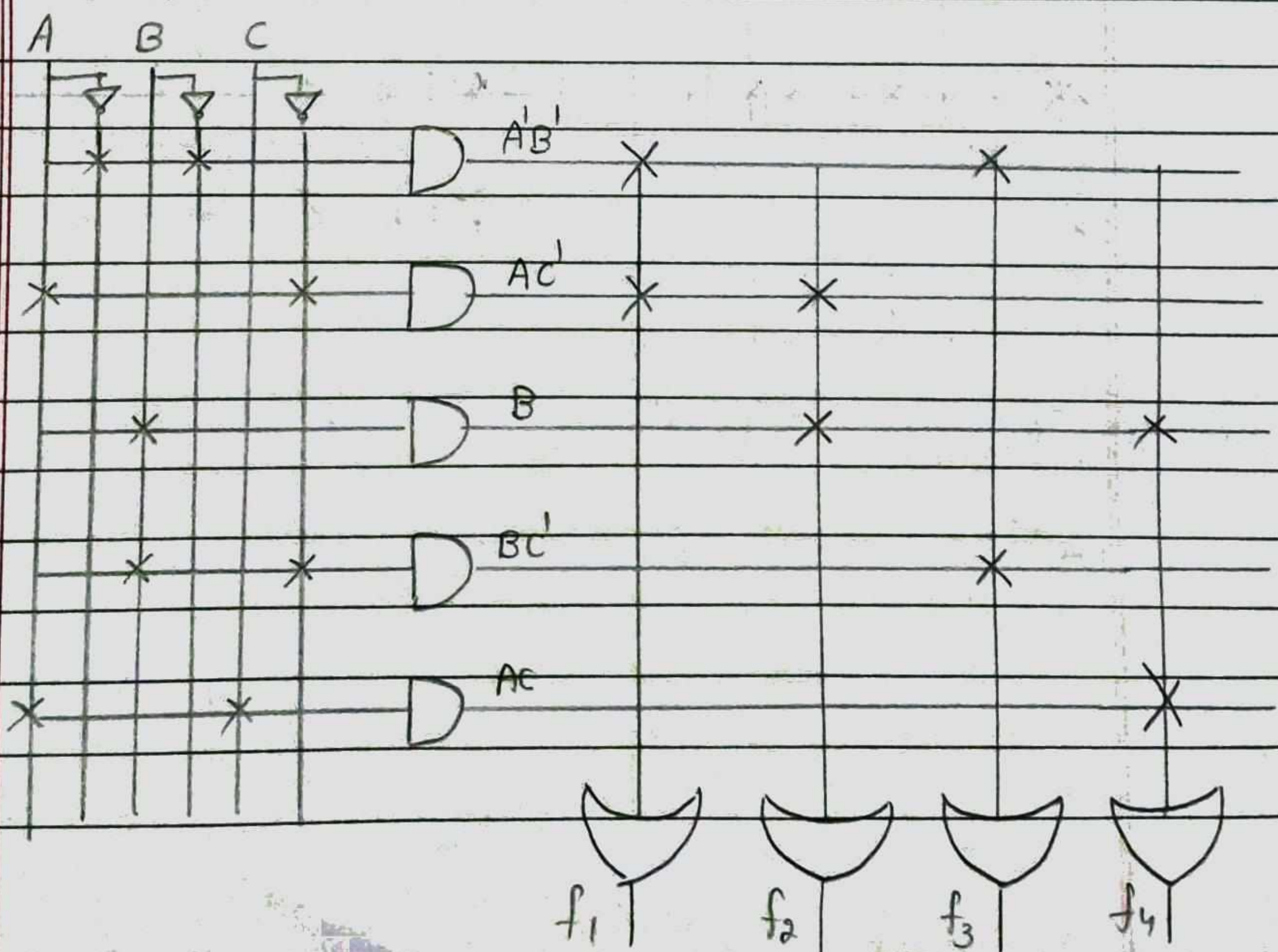
→ Realize the given functions/SOP equation using PLA & also write PLA Table.

$$F_0(A, B, C) = A'B' + AC'$$

$$F_1(A, B, C) = B + AC'$$

$$F_2(A, B, C) = A'B' + BC'$$

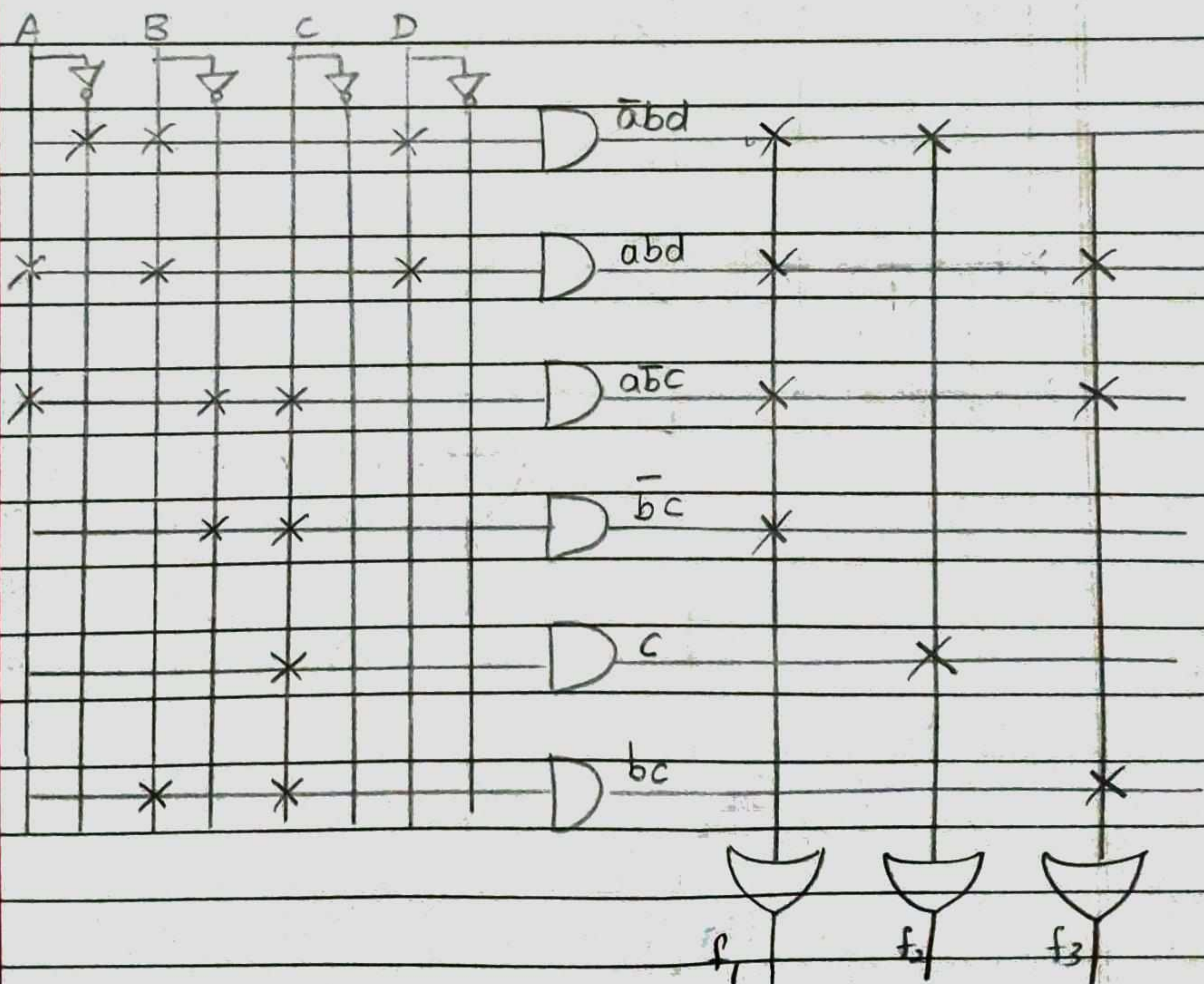
$$F_3(A, B, C) = AC + B$$



	Inputs			Outputs			
Products	A	B	C	f_0	f_1	f_2	f_3
$A'B'$	0	0	—	1	0	1	0
AC'	1	—	0	1	1	0	0
B	—	1	—	0	1	0	1
BC'	—	1	0	0	0	1	0
Ac	1	—	1	0	0	0	1

→ Realize the given PLA table using PLA.

	Inputs				Outputs		
Products	A	B	C	D	f_1	f_2	f_3
$\bar{a}bd$	0	1	—	1	1	1	0
abd	1	1	—	1	1	0	1
$a\bar{b}c$	1	0	0	—	1	0	1
$\bar{b}c$	—	0	1	—	1	0	0
c	—	—	1	—	0	1	0
bc	—	1	1	—	0	0	1



⇒ PAL: Programmable Array Logic

Realize PAL for the given functions

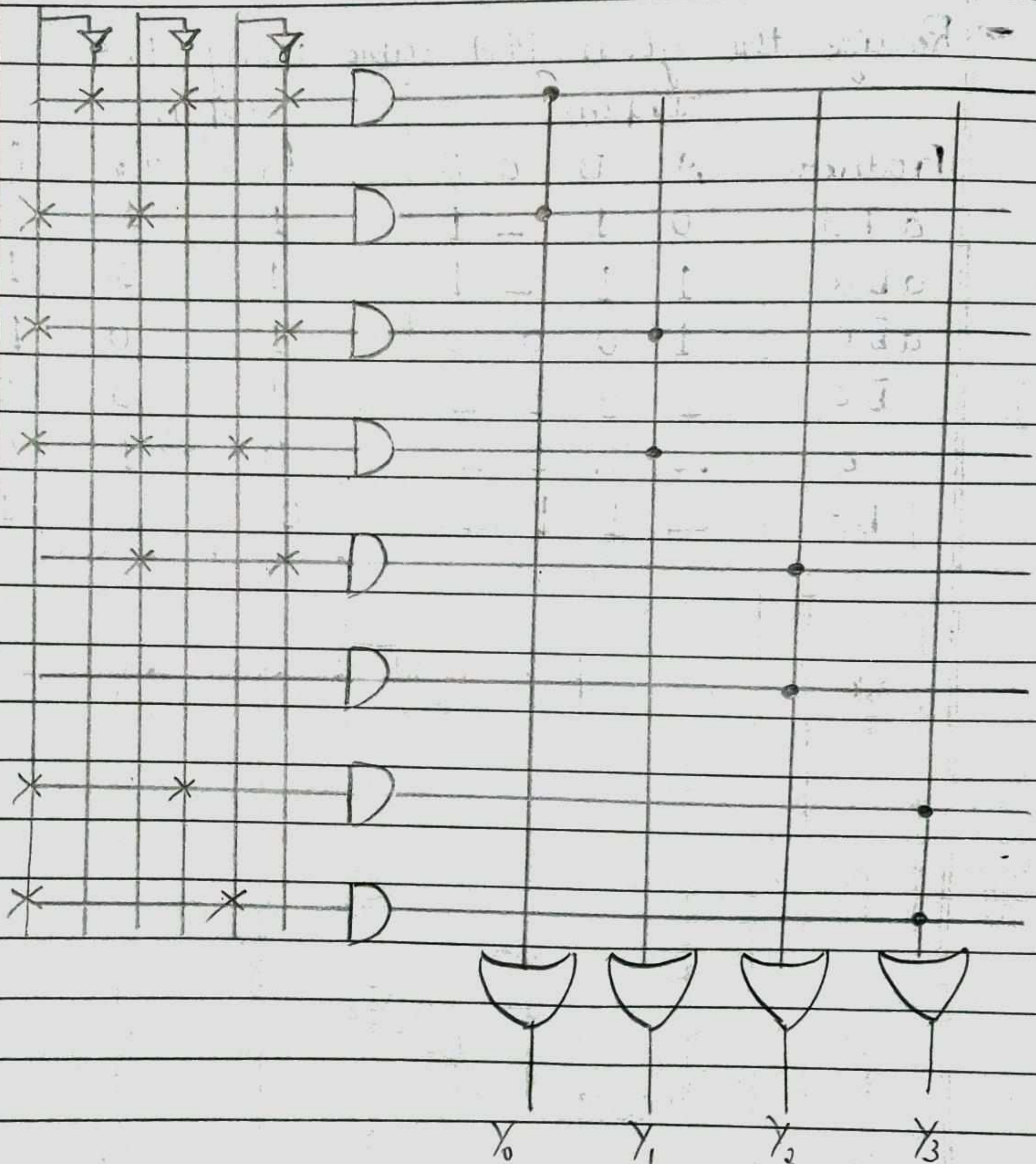
$$Y_0 = \bar{A}\bar{B}\bar{C} + AB$$

$$Y_1 = A\bar{C} + ABC$$

$$Y_2 = B\bar{C}$$

$$Y_3 = A\bar{B} + AC$$

A B C



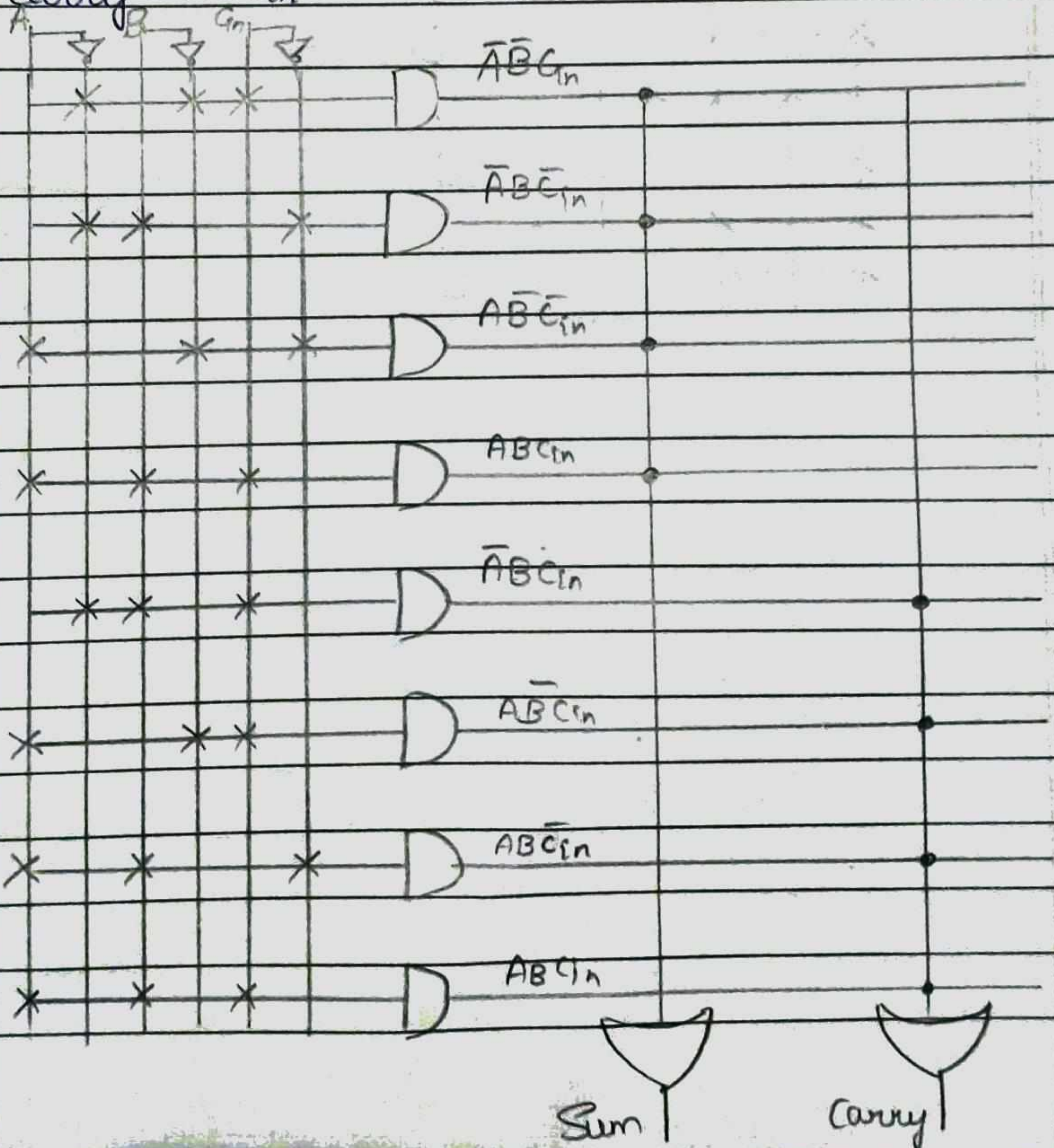
→ Realize Full Adder using PAL

Truth Table.

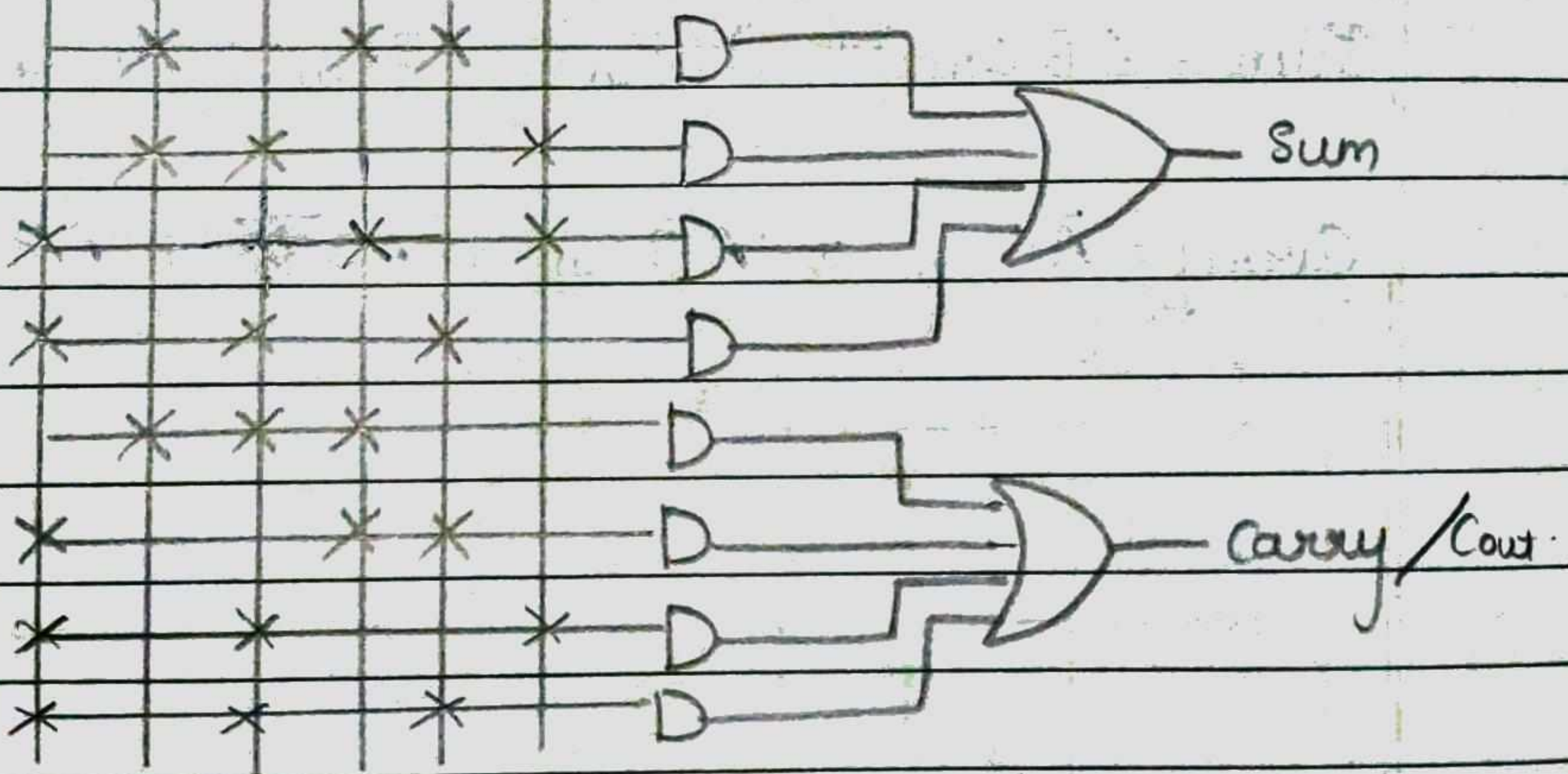
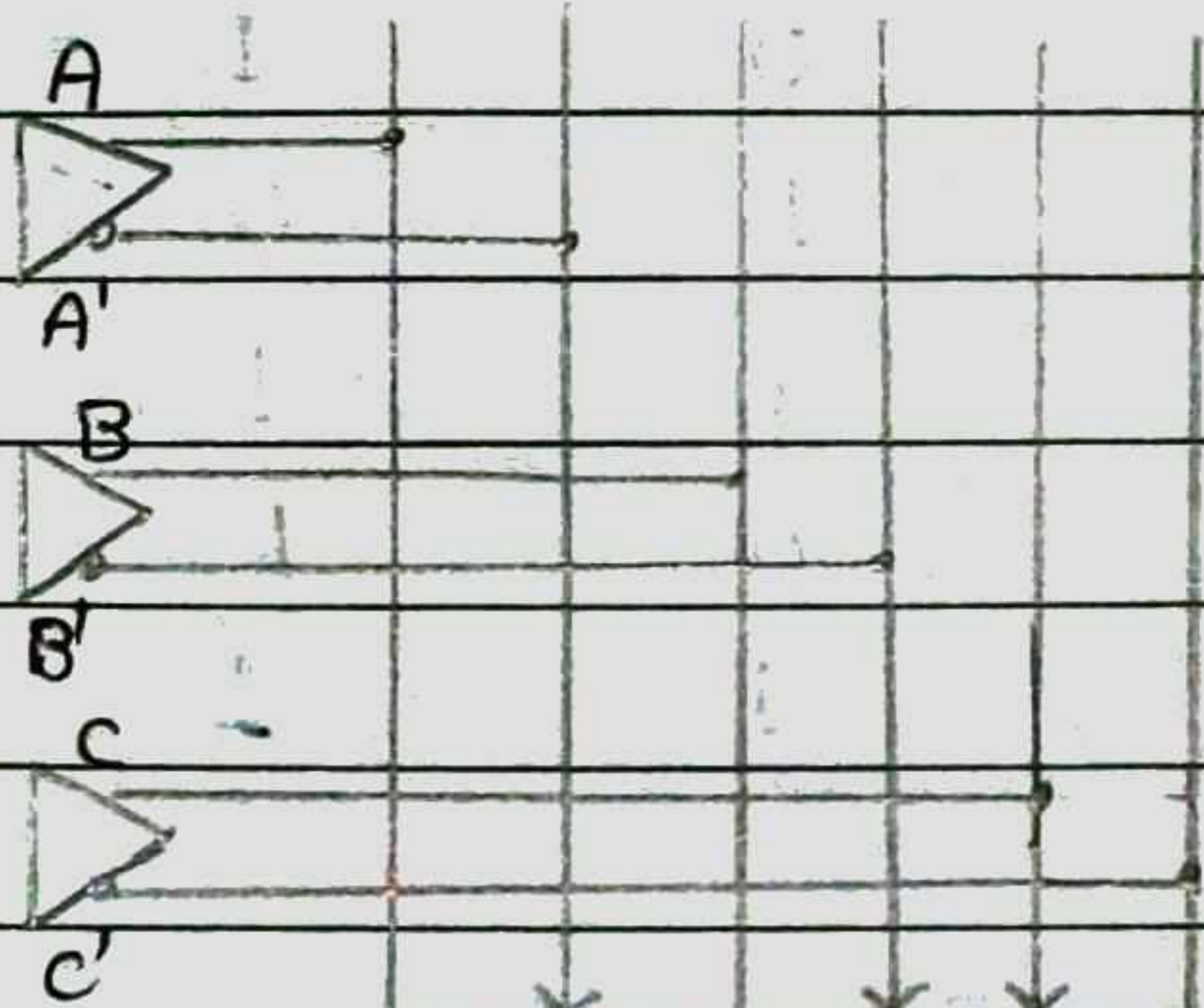
Input			Output	
A	B	Cin	Sum	Carry (cout)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$\text{Sum} = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$$

$$\text{Carry} = \bar{A}BC_{in} + A\bar{B}C_{in} + AB\bar{C}_{in} + ABC_{in}$$



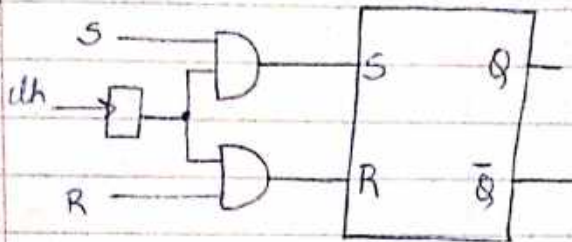
Note:



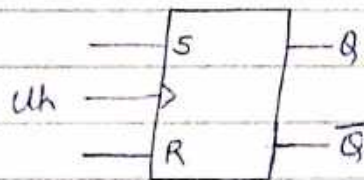
Flip Flops

1. SR flip flop - Positive edge trigger

a. Logic diagram



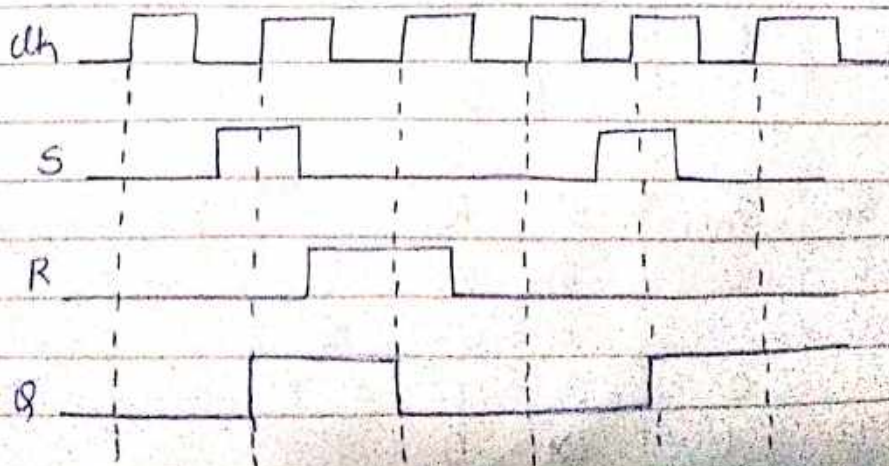
b. Logic symbol / IEEE symbol



c. Truth Table

clk	S	R	Q_{n+1}
↑	0	0	Q_n NC
↑	0	1	0 Reset
↑	1	0	1 Set
↑	1	1	? Illegal

d. Timing diagram



e. Characteristic table

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

f. Characteristic equation

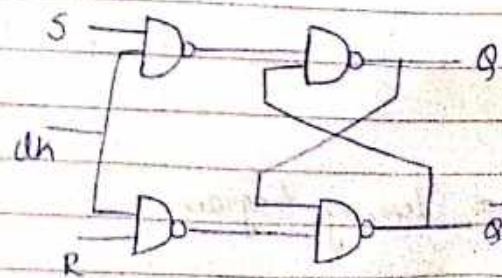
$Q_n \backslash SR$	00	01	11	10
0	0	0	X	1
1	1	0	X	1

$$Q_{n+1} = S + Q_n R'$$

g. Excitation table

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

h. Logic circuit

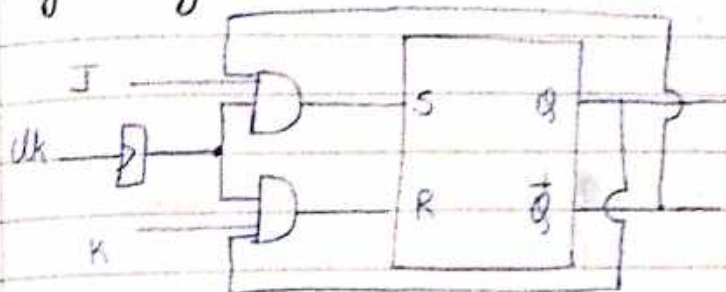


Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

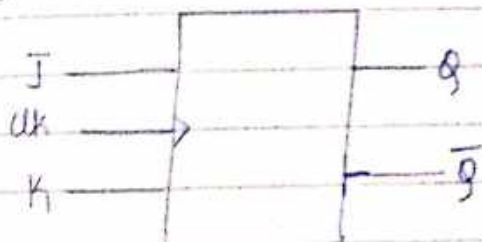
JK flip flop

- (positive edge trigger)

1. Logic diagram



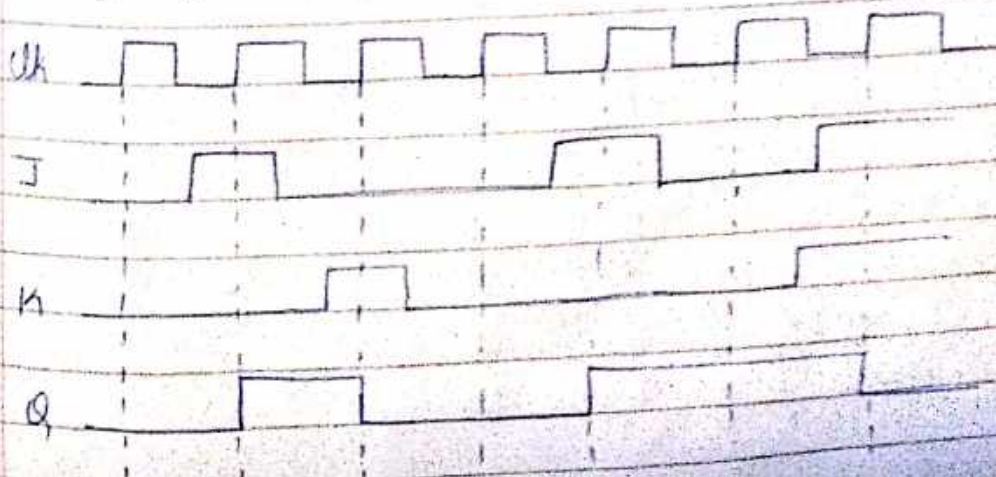
2. Logic symbol / IEEE symbol



3. Truth table

clk	J	K	Q_{n+1}
↑	0	0	Q_n NC
↑	0	1	0 reset
↑	1	0	1 set
↑	1	1	toggle

4. Timing diagram



5 Characteristic table / Next state table

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

6 Characteristic equation

Q_n	$J\bar{K}$	$\bar{J}K$	JK	$\bar{J}\bar{K}$
\bar{Q}_n	0	0	1	1
Q_n	1	0	0	1

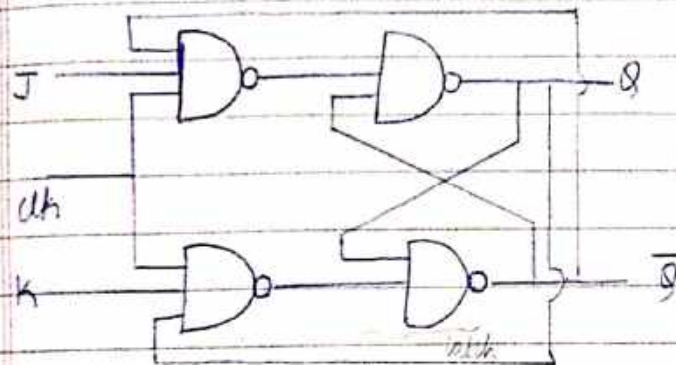
$$Q_{n+1} = \bar{Q}_n J + Q_n \bar{K} = \bar{Q}_n J + Q_n \bar{K}$$

7 Excitation table

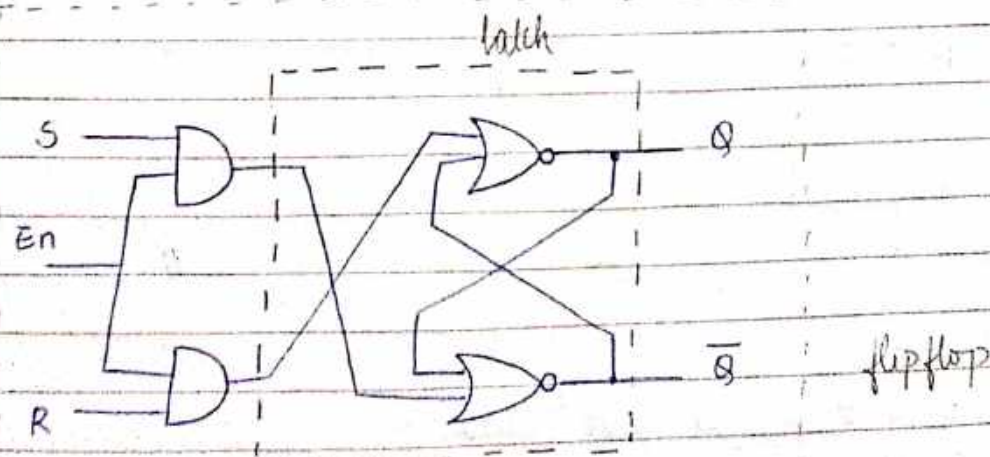
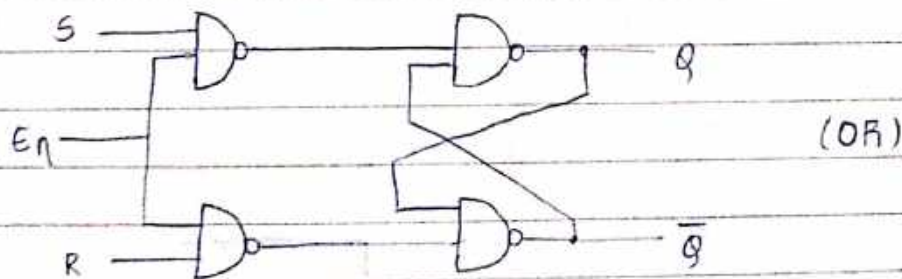
Q	Q_{n+1}	J	K
0	0	0	0
		0	1
0	1	1	0
		1	1
1	0	0	1
		1	1
1	1	0	0
		1	0

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

8 logic unit

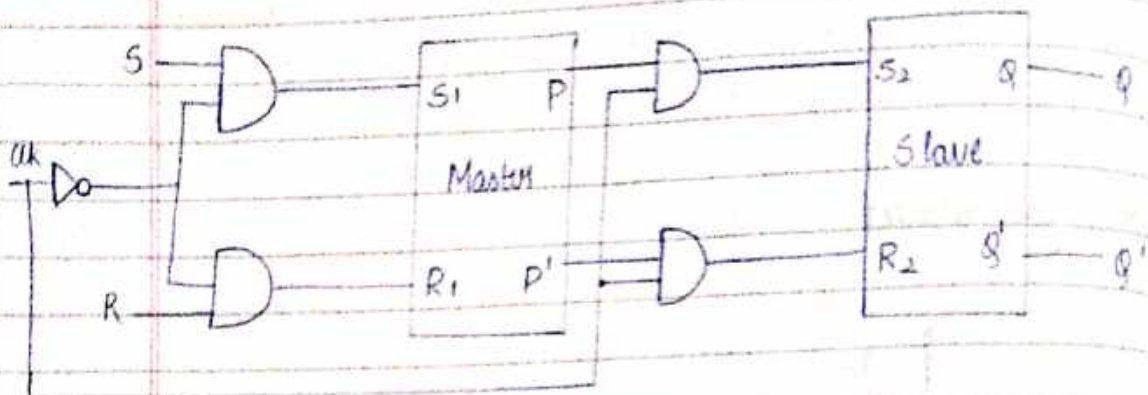


Note -



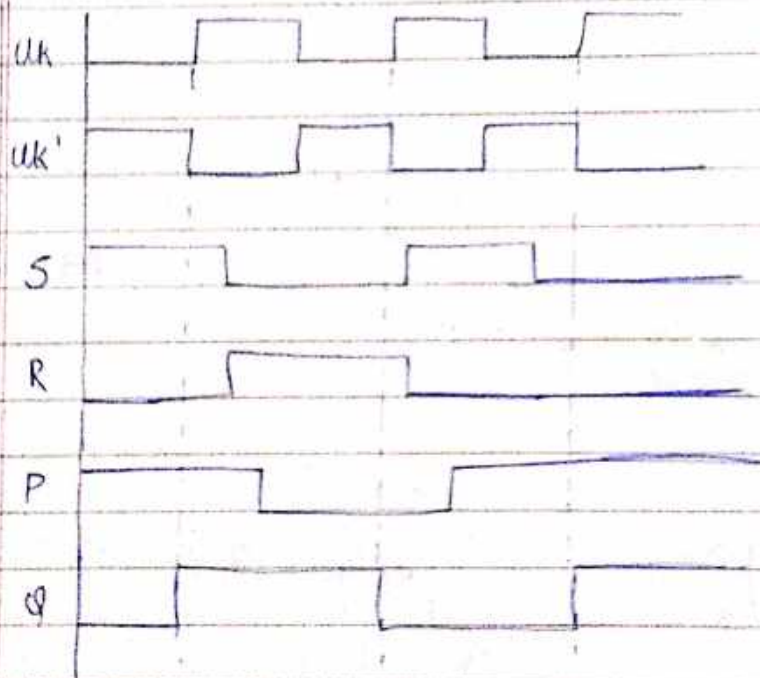
* Whenever clk is connected to a latch it is a flipflop

Implement SR master slave flip flop using two SR latches and gates



here, master - negative edge trigger
slave - positive edge trigger

Timing diagram



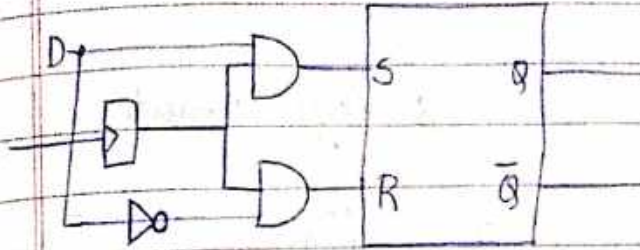
Note -

- > Q_{n+1} or Q^+ is next state
- > Q_n or Q is a current state

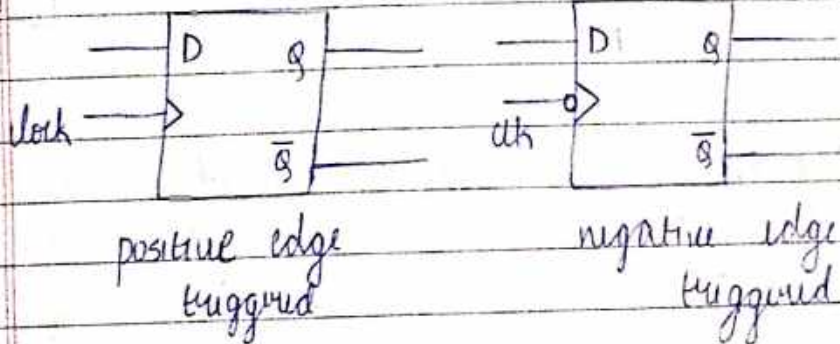
D Flip Flop (D-Data)

- whatever input is given at D is available at the output Q

1. Logic diagram



2. IEEE symbol



3. Truth Table

clk	D	Q_{n+1}
↑	0	0
↑	1	1

4. Characteristic / Next state table

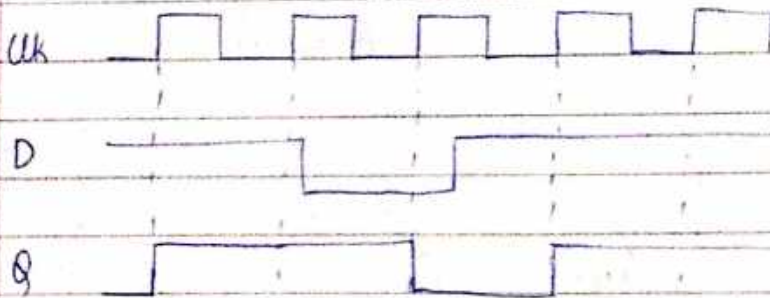
Q_n	D	Q_{n+1}
0	0	0
0	1	1
1	0	0
1	1	1

classmate
Date _____
Page _____

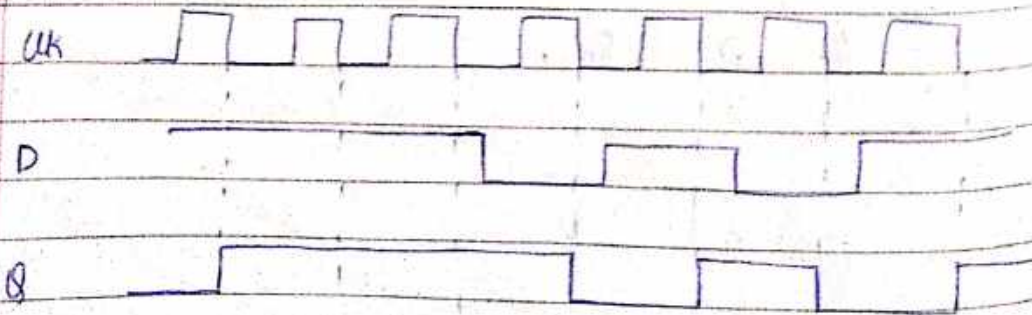
$$Q_{n+1} = D$$

8 Logic unit

7. Timing diagram (positive edge-triggering edge)

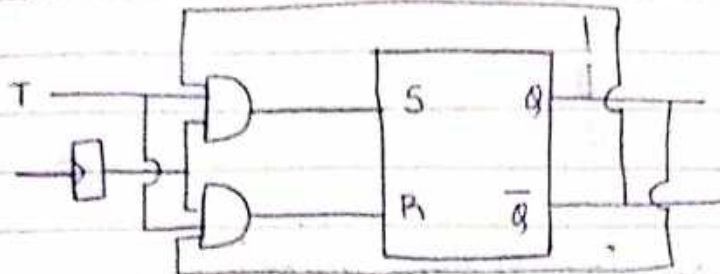


timing diagram (falling edge or negative edge)

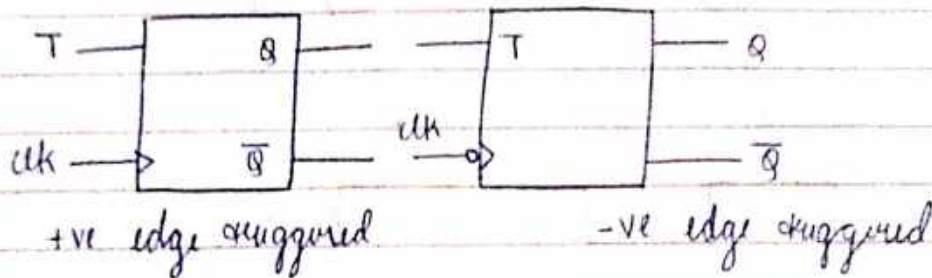


T Flip Flop [Toggle]

1. Logic diagram



2. IEEE symbol



3. Truth Table

clk	T	Q_{n+1}
↑	0	Q_n (No change)
↑	1	\bar{Q}_n (Toggle)

4. Characteristic equation

Q_n	T	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

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

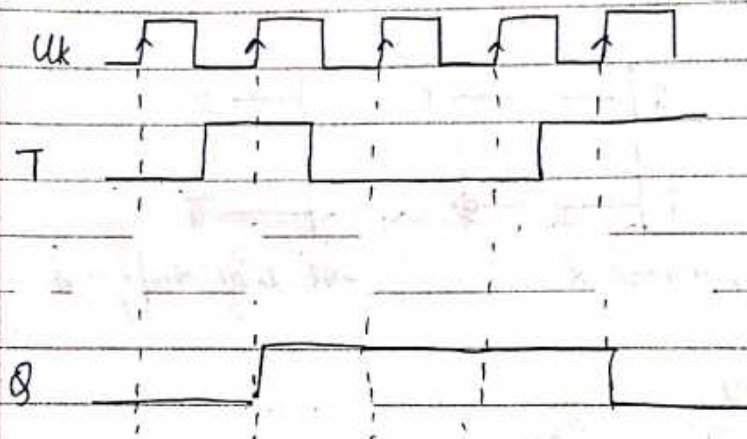
5. Characteristic table

Q_n	T	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

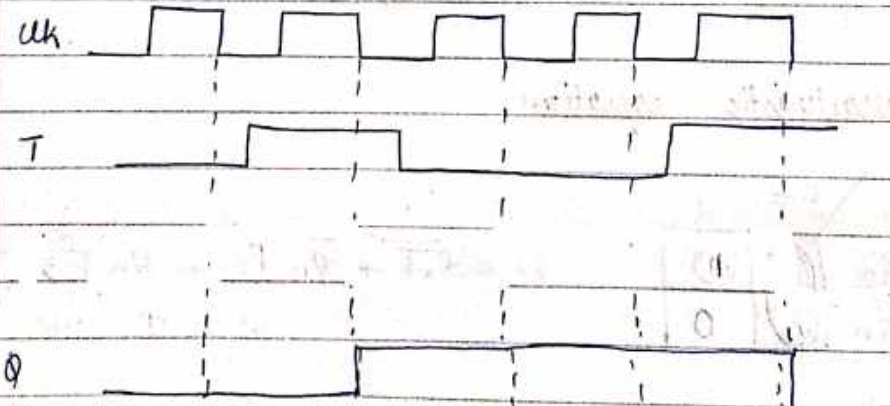
6. Excitation table

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

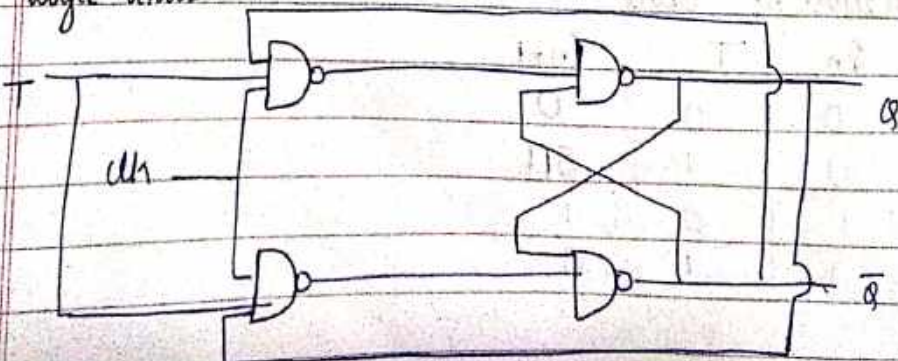
7. Timing diagram [+ve or rising edge]



-ve edge or falling edge



logic circuit



Timing diagram with propagation delay (t_p)

Propagation delay of a flip flop is the time between the active edge of the clock and the resulting change in the output.

If the flipflop input changes at the same time as the active edge, the behaviour is unpredictable.

Consider the timing issues associated with the D flip flop

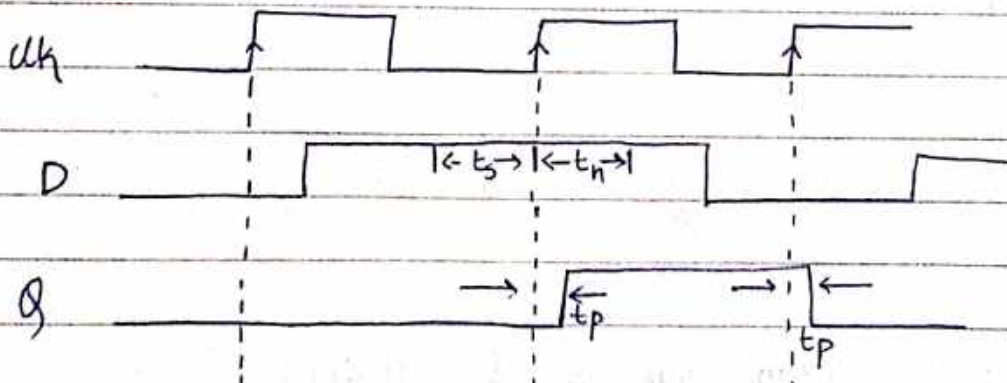
> Setup time (t_{su}) -

The amount of time that D must be stable before the active edge.

> Hold time (t_h) -

The amount of time the D must hold the same value after the active edge.

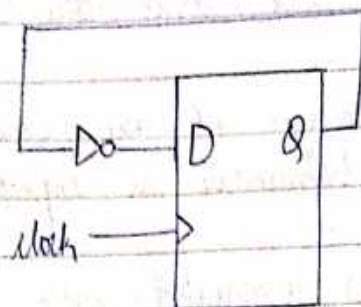
Timing diagram -



(D should not change during the trigger of the clock pulse)

Determination of Minimum Clock Period

a. Simple flip-flop circuit



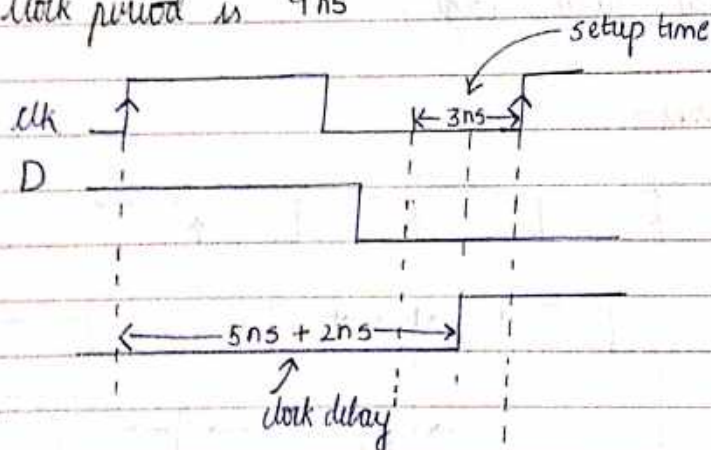
Flip flop delay = 5 ns

not gate delay = 2 ns

setup time = 3 ns

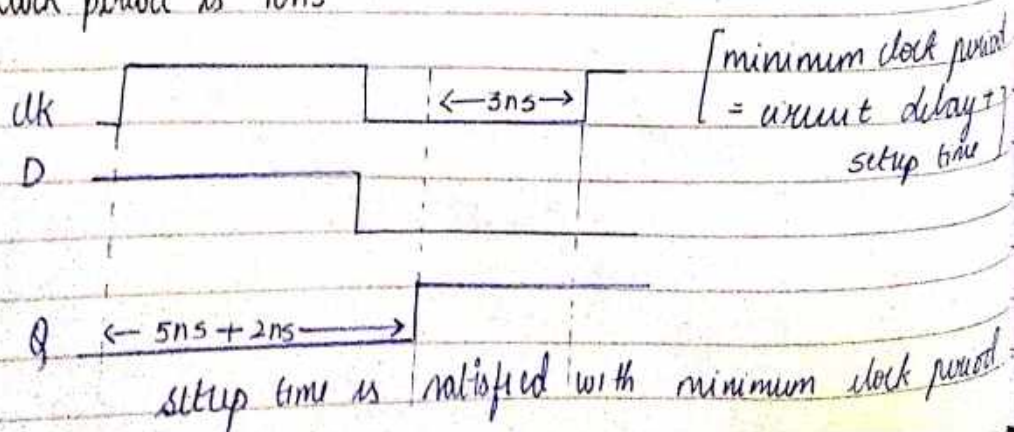
b. i. Setup time is not satisfied

i. Clock period is 9 ns



Setup time is not satisfied

ii. Clock period is 10 ns



setup time is satisfied with minimum clock period

Conversion of flip flops.

1. SR Flip Flop to D flip flop

characteristic table (D) & excitation table for (SR)

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

K-map

$S \rightarrow Q_n$

D	0	1
0	0	1
1	0	X

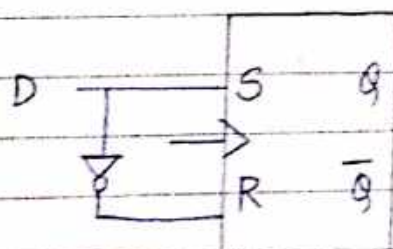
$S = D$

$R \rightarrow Q_n$

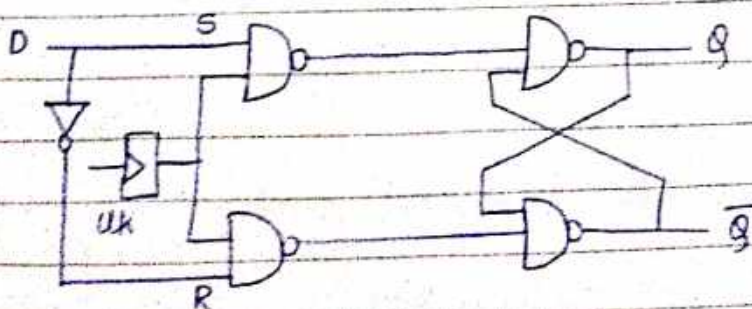
D	0	1
0	X	0
1	1	0

$R = \overline{D}$

logic diagram



logic circuit



2 SR flip flop to JK flip flop

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

K-map

S →

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

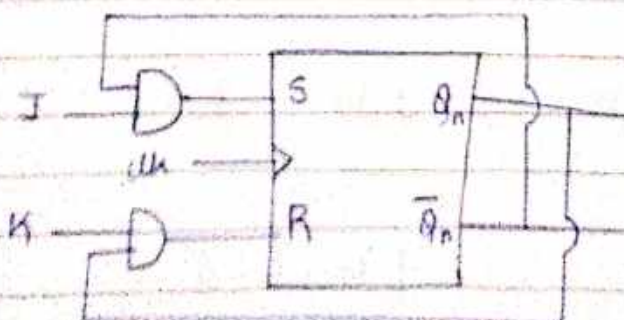
$S = \bar{Q}_n J$

R →

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

$$R = Q_n K$$

Logic diagram



3 convert JK flip flop to T flip flop

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

K-map

J \rightarrow Q_n

T	0	1
0	0	1
1	X	X

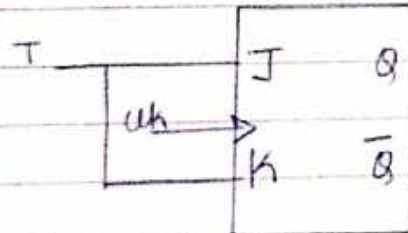
$J = T$

K \rightarrow Q_n

T	0	1
0	X	X
1	0	1

$K = T$

logic diagram



4 convert SR flip flop to T flip flop

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

K-map

S \rightarrow Q_n

T	0	1
0	0	1
1	X	0

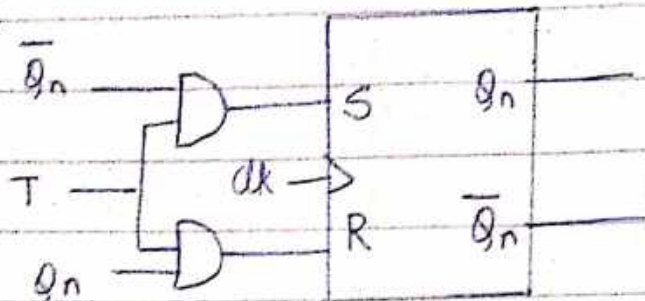
$S = \bar{Q}_n T$

R \rightarrow Q_n

T	0	1
0	X	0
1	0	1

$R = Q_n T$

Logic diagram



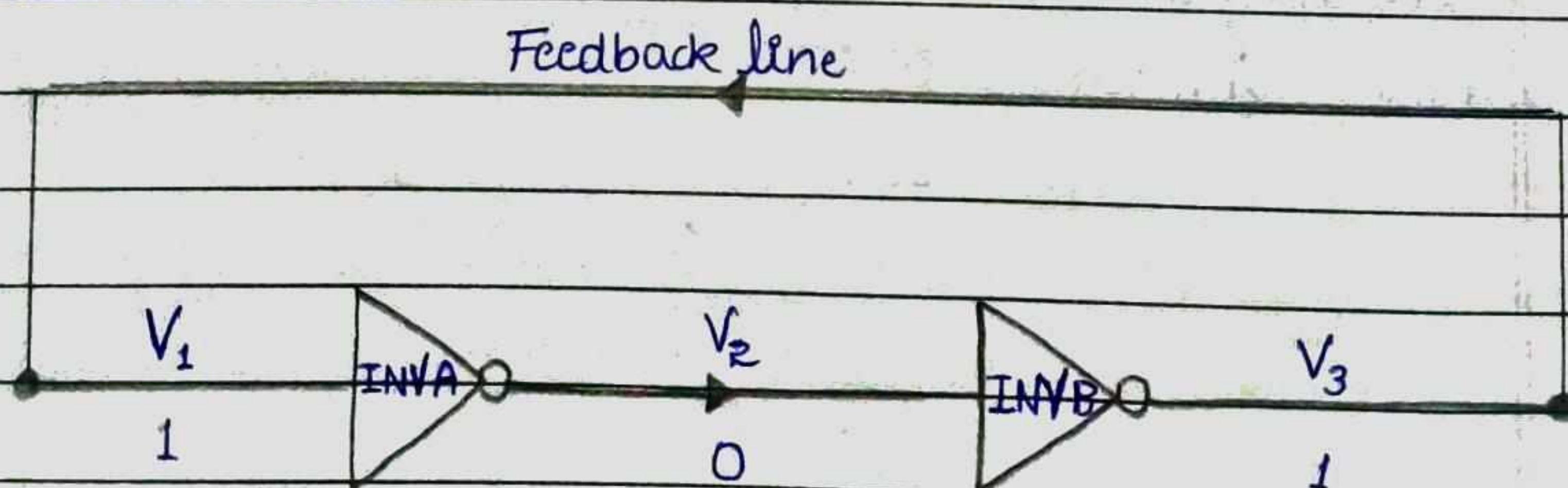
Latches and Flip-Flops

- Sequential circuits have the property that the output depends not only on the present input but also on the past sequence of inputs.
- In effect, these circuits must be able to "remember" something about the past history of the inputs in order to produce the present output.
- Latches and flip-flops are commonly used memory devices in sequential circuits.
- Each of the flip-flops has a clock input, and the flip-flops can only change state in response to a clock pulse.

⇒ RS Flip-Flops
 NOR
 NAND

- A flip-flop is a bistable electronic circuit that has two stable states.
- Any device or circuit that has two stable states is said to be bistable.
- One of the easiest ways to construct a flip-flop is to connect two inverters in a series. The line connecting the output of inverter B (INVB) back to the input of inverter A (INVA) is referred to as the feedback line.

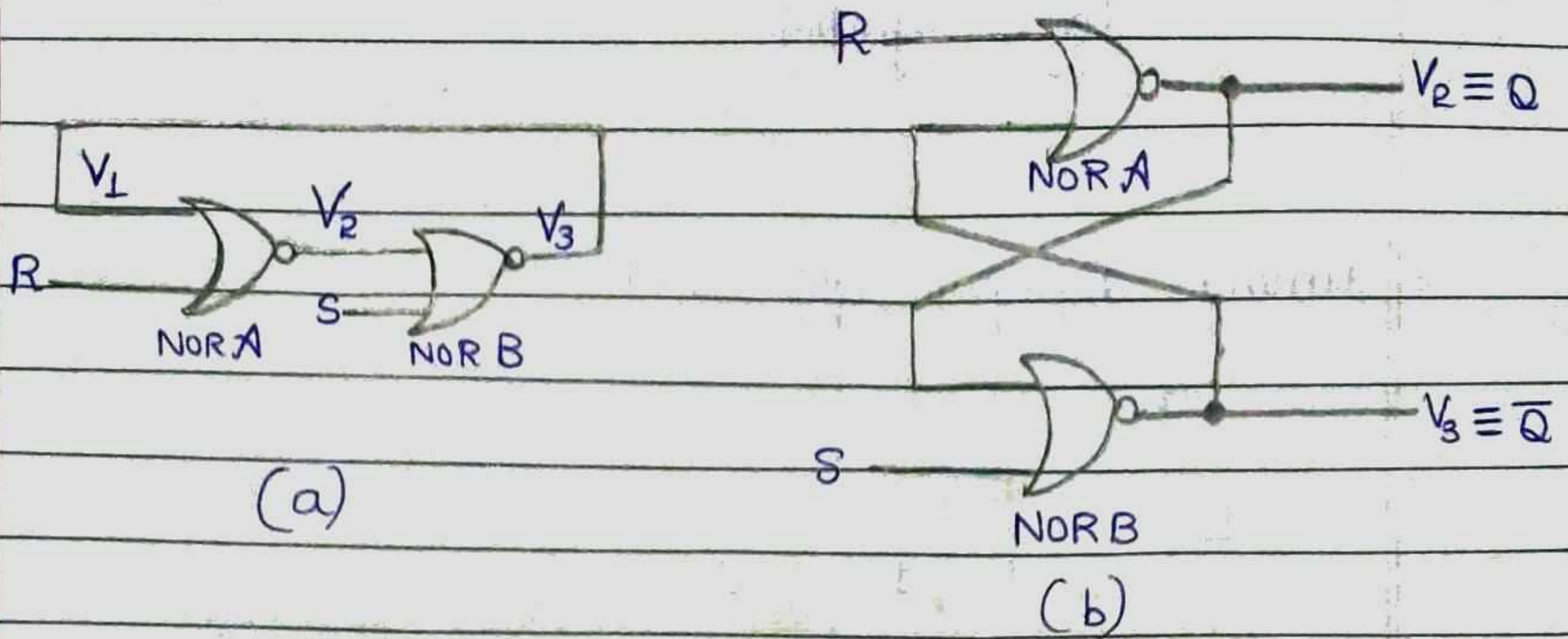
→ Bistable circuit



→ NOR - Gate latch

- The basic flip-flop can be improved by replacing the inverters with either NAND or NOR gates.
- Two 2-input NOR gates are connected to form a flip-flop.

→ NOR - gate flip-flop.

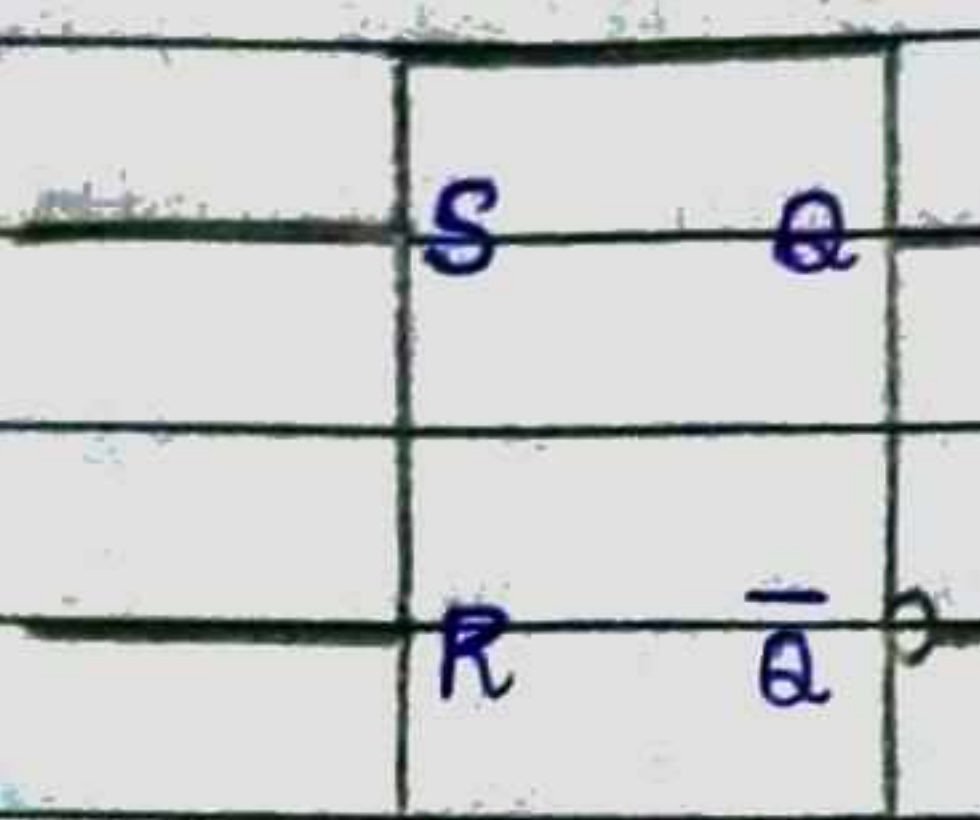


- The flip-flop actually has two outputs, defined in more general terms as Q and \bar{Q} . It should be clear that regardless of the value of Q , its complement is \bar{Q} . There are two inputs to the flip-flop defined as R and S . The input/output possibilities for this RS flip-flop are summarized in the truth table.

→ TRUTH TABLE for a NOR-gate RS flip-flop

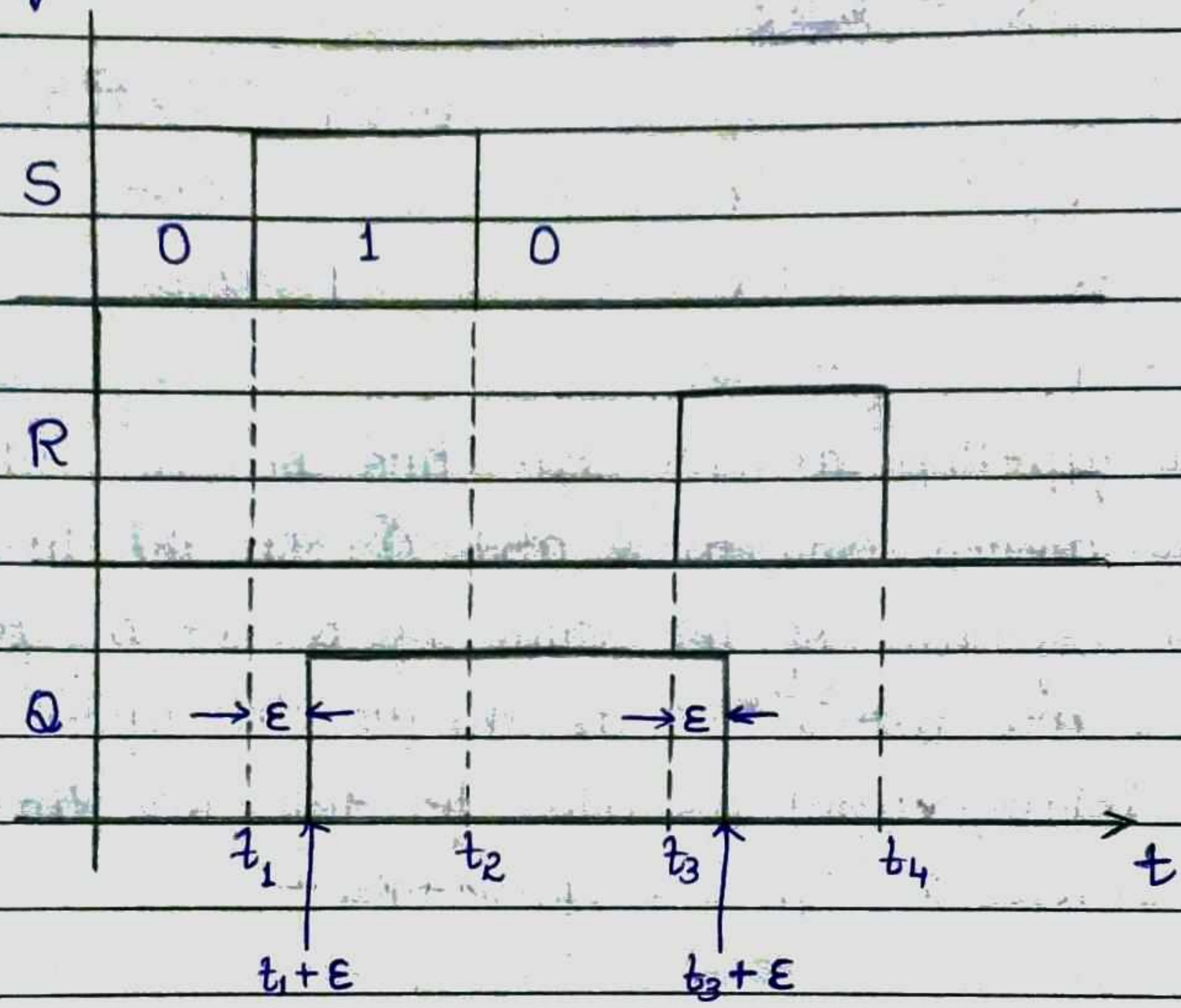
R	S	Q	Action.
0	0	Last state	No change
0	1	1	SET
1	0	0	RESET
1	1	?	Forbidden

→ RS flip-flop.



Logic symbol

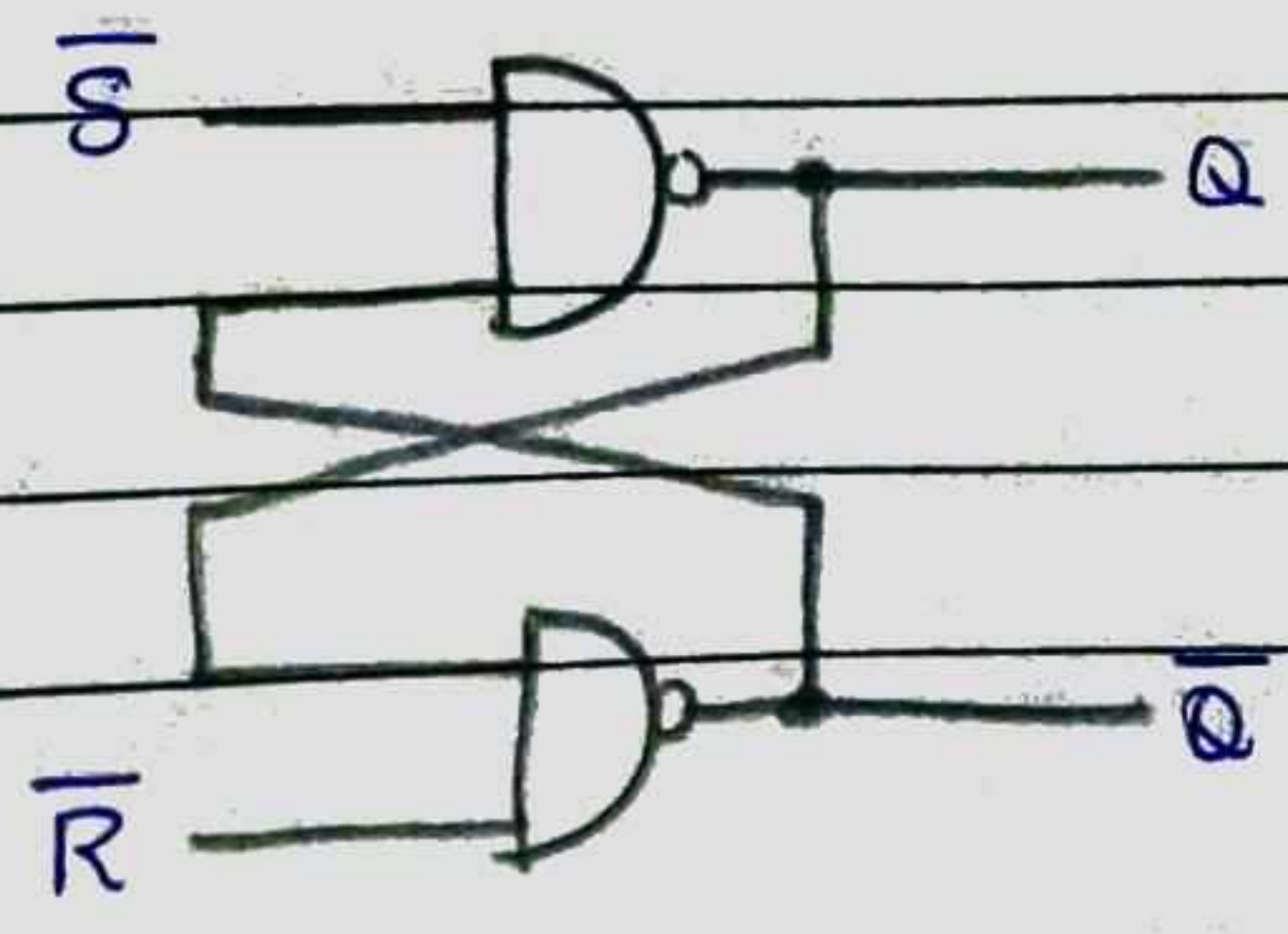
→ Timing Diagram for S-R latch.



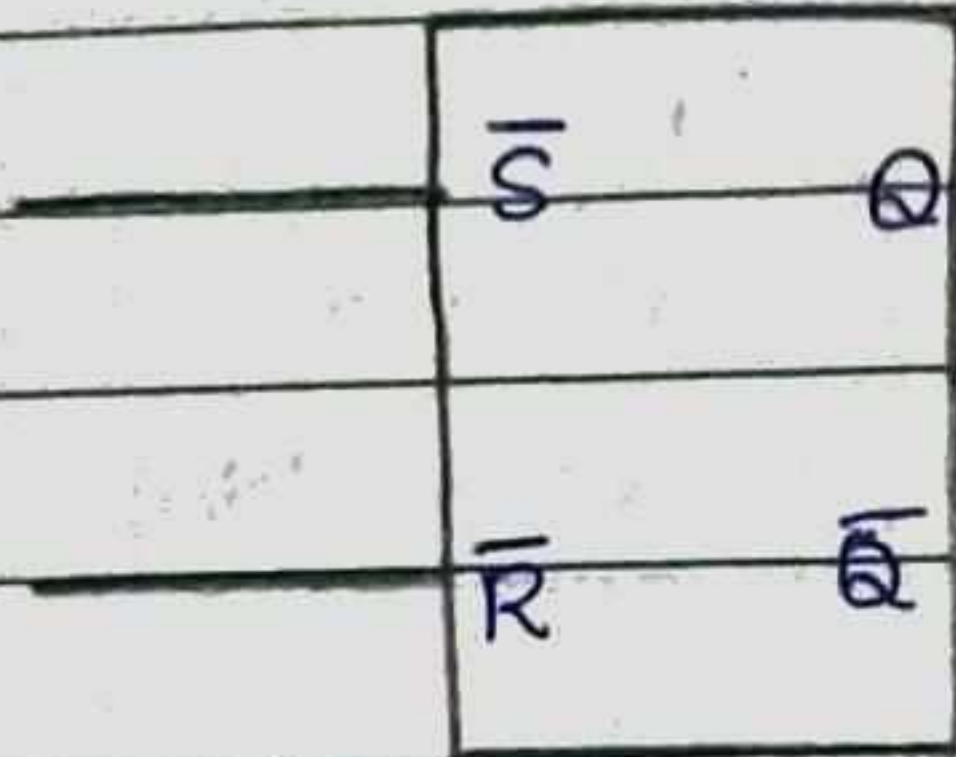
⇒ NAND-Gate Latch

→ An alternative form of the S-R latch uses NAND gates. We will refer to this circuit as an \bar{S} - \bar{R} latch, and the table describes its operation.

→ $\bar{R}\bar{S}$ flip-flop. (a) NAND gate latch.



(b) Logic symbol.

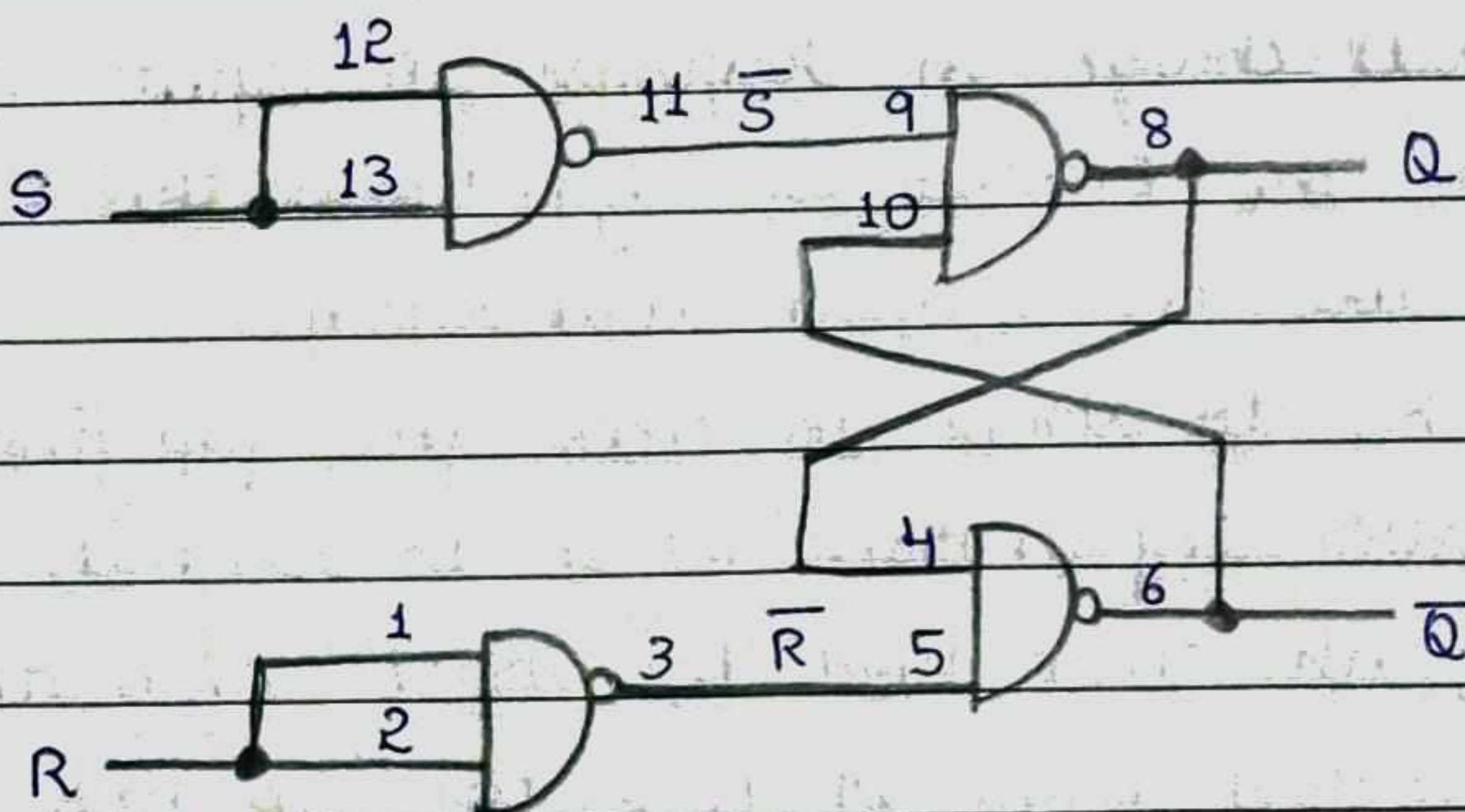


(c) Truth Table.

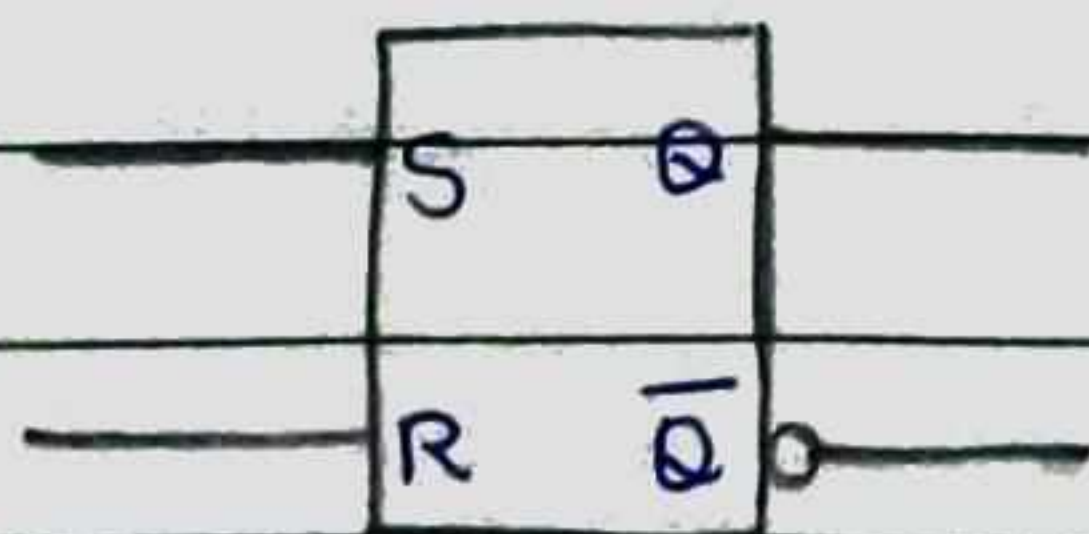
\bar{R}	\bar{S}	Q
1	1	last state
1	0	1
0	1	0
1	0	?(Forbidden)

→ Convert the $\bar{R}\bar{S}$ flip-flop into an RS-Flip-Flop.
 An RS Flip-Flop (latch)

(a) 54/7400



(b) Logic symbol.



(c)	R	S	Q
	0	0	last state
	0	1	1
	1	0	0
	1	1	? (Forbidden)

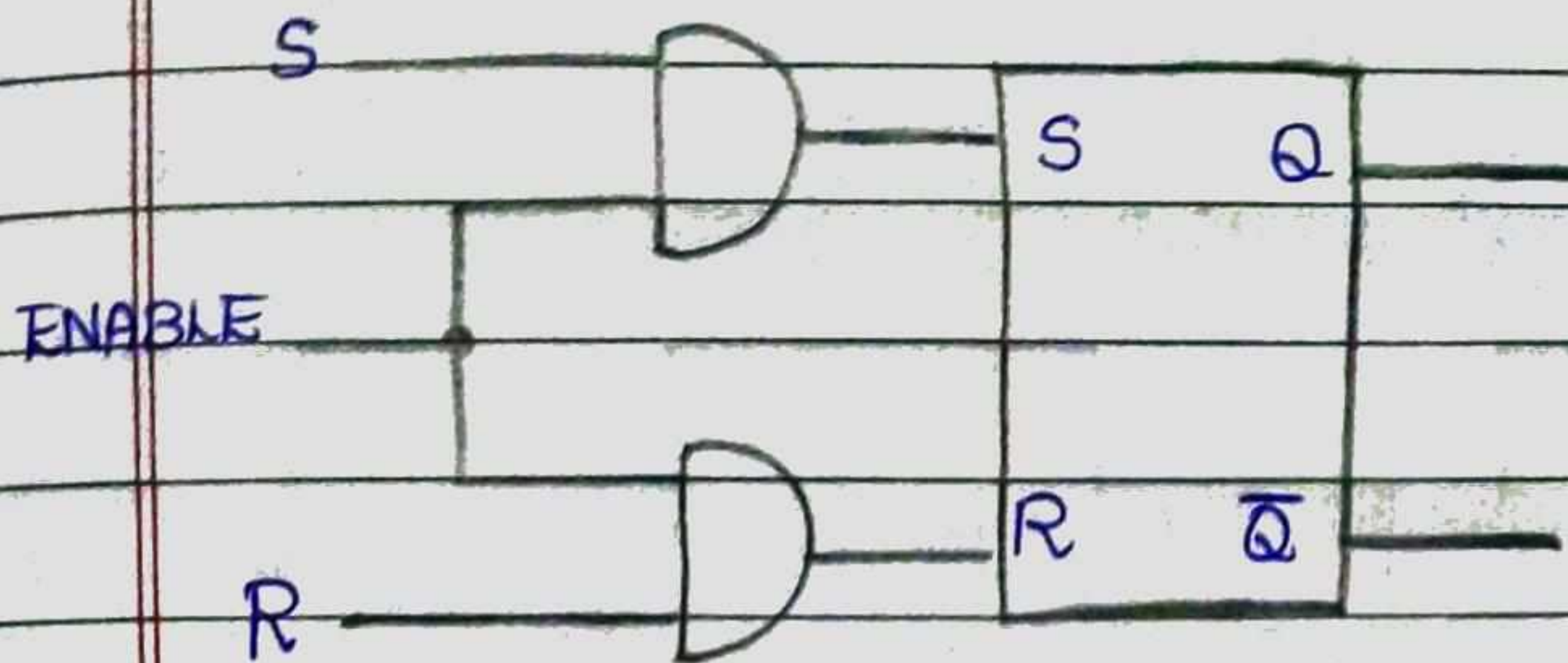
⇒ Gated Flip-Flops

⇒ Clocked RS Flip-Flops

- The addition of two AND gates at the R and S inputs will result in a flip-flop that can be enabled or disabled.
- When the ENABLE input is low, the AND gate outputs must both be low and changes in neither R nor S will have any effect on the flip-flop output Q. The latch is said to be disabled.
- When the ENABLE input is high, information at the R and S inputs will be transmitted directly to the outputs. The latch is said to be enabled. The output will change in response to input changes as long as the ENABLE is high. When the ENABLE input goes low, the output will retain.
- It is possible to strobe or clock the flip-flop in order to store information (set it or reset it) at any time, and then hold the stored information for any desired period of time. The flip-flop is called a gated or clocked RS flip-flop.

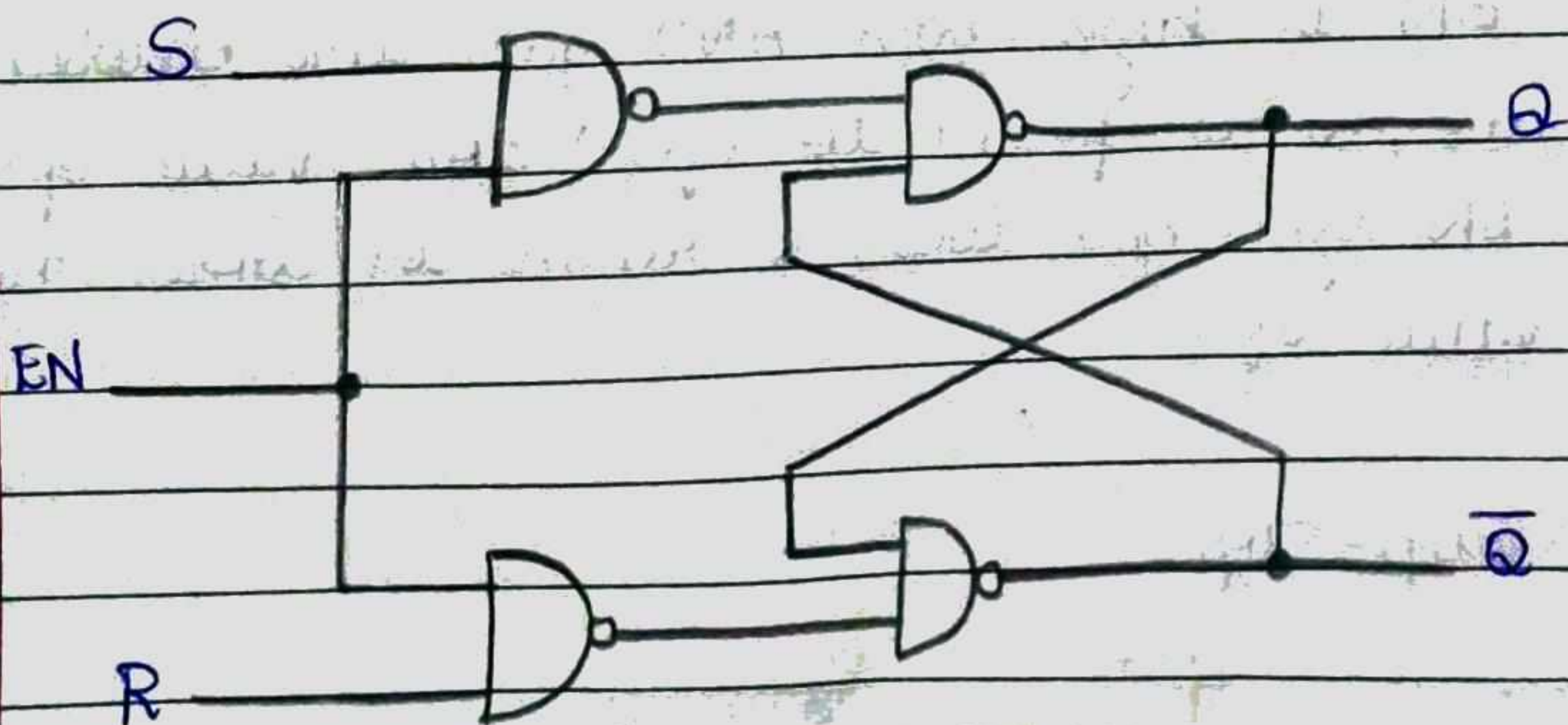
Clocked RS flip-flop

(a) Logic diagram

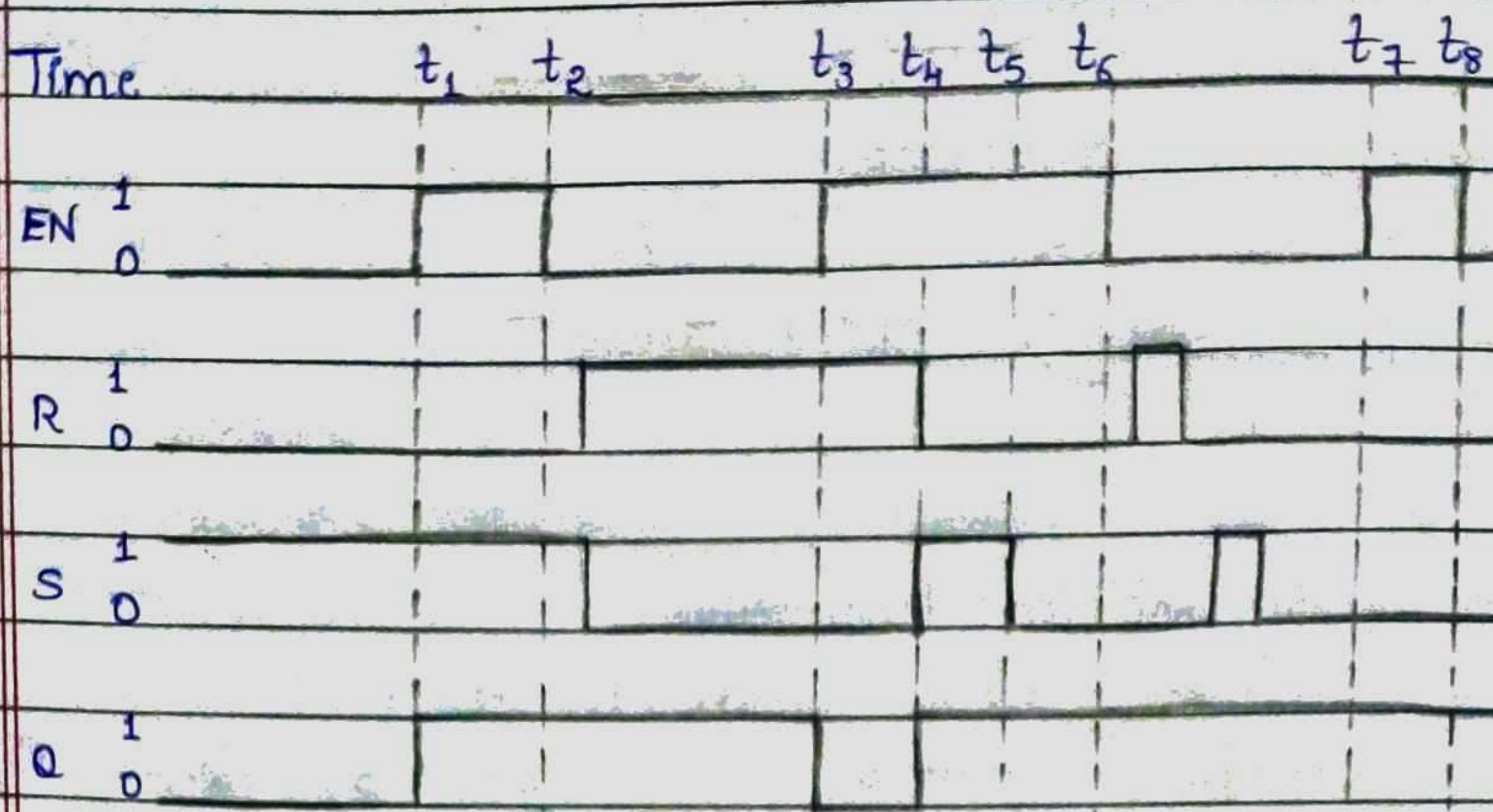


(b) Symbol and truth table

			EN	S	R	Q_{n+1}
	S	Q	1	0	0	Q_n (no change)
	EN		1	0	1	0
	R	\bar{Q}	1	1	0	1
			1	1	1	?(Illegal)
			0	X	X	Q_n (no change)



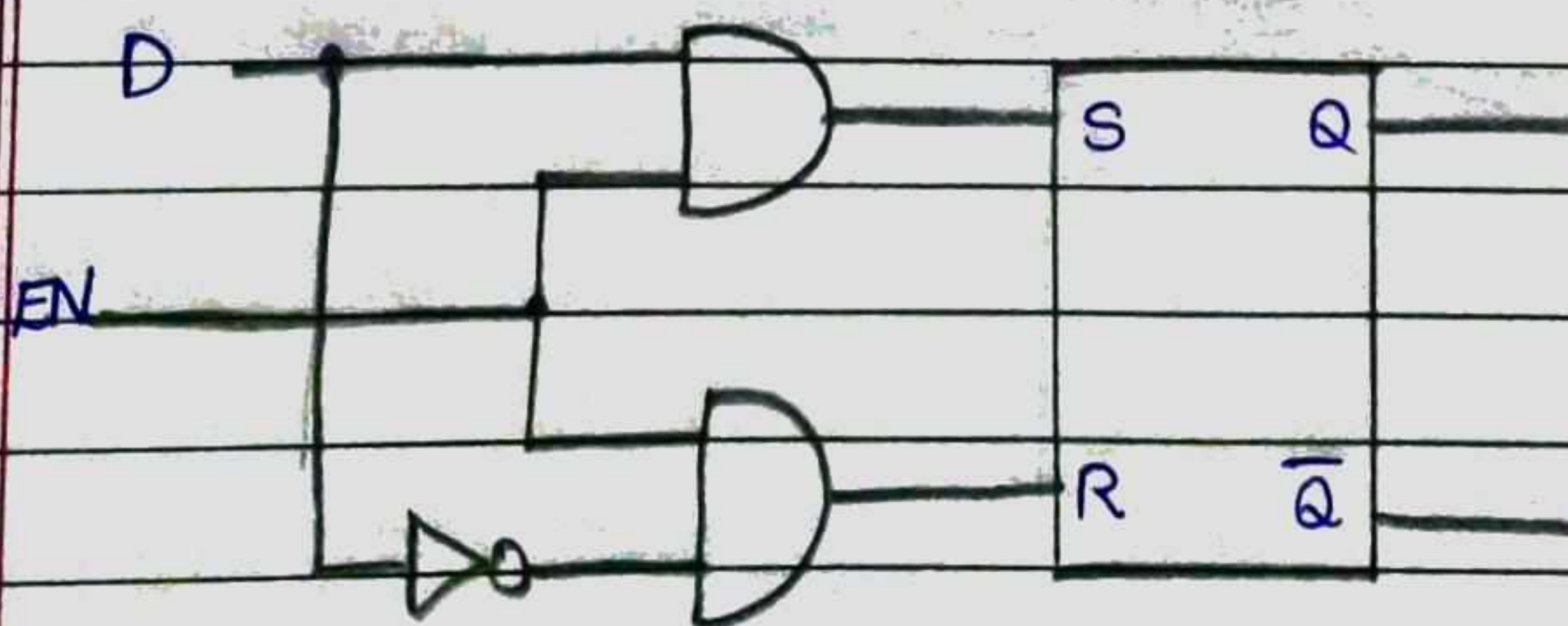
→ Input waveform R, S, EN applied to a clocked RS flip-flop.



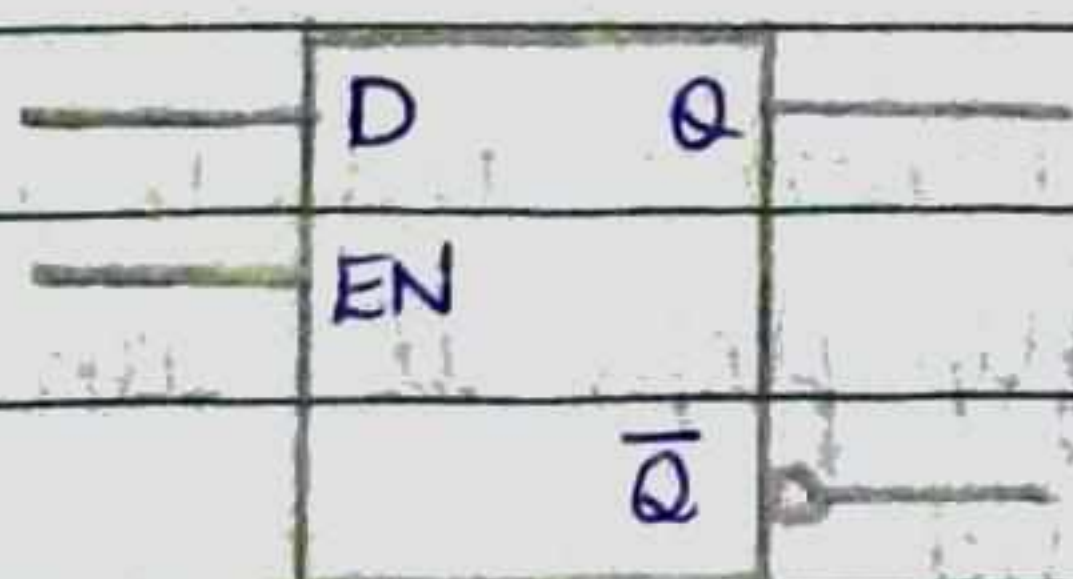
⇒ D Flip-Flops (D-Data)

- The action of the circuit is straightforward.
- When EN is low, both AND gates are disabled; therefore, D can change value without affecting the value of Q.
- When EN is high, both AND gates are enabled. In this case, Q is forced to equal the value of D.
- When EN again goes low, Q retains or stores the last value of D.

A D Flip-Flop.



Logic symbol



Truth Table

EN	D	Q_{n+1}
0	X	Q_n (last state)
1	0	0
1	1	1

⇒ Edge-Triggered Flip-Flop

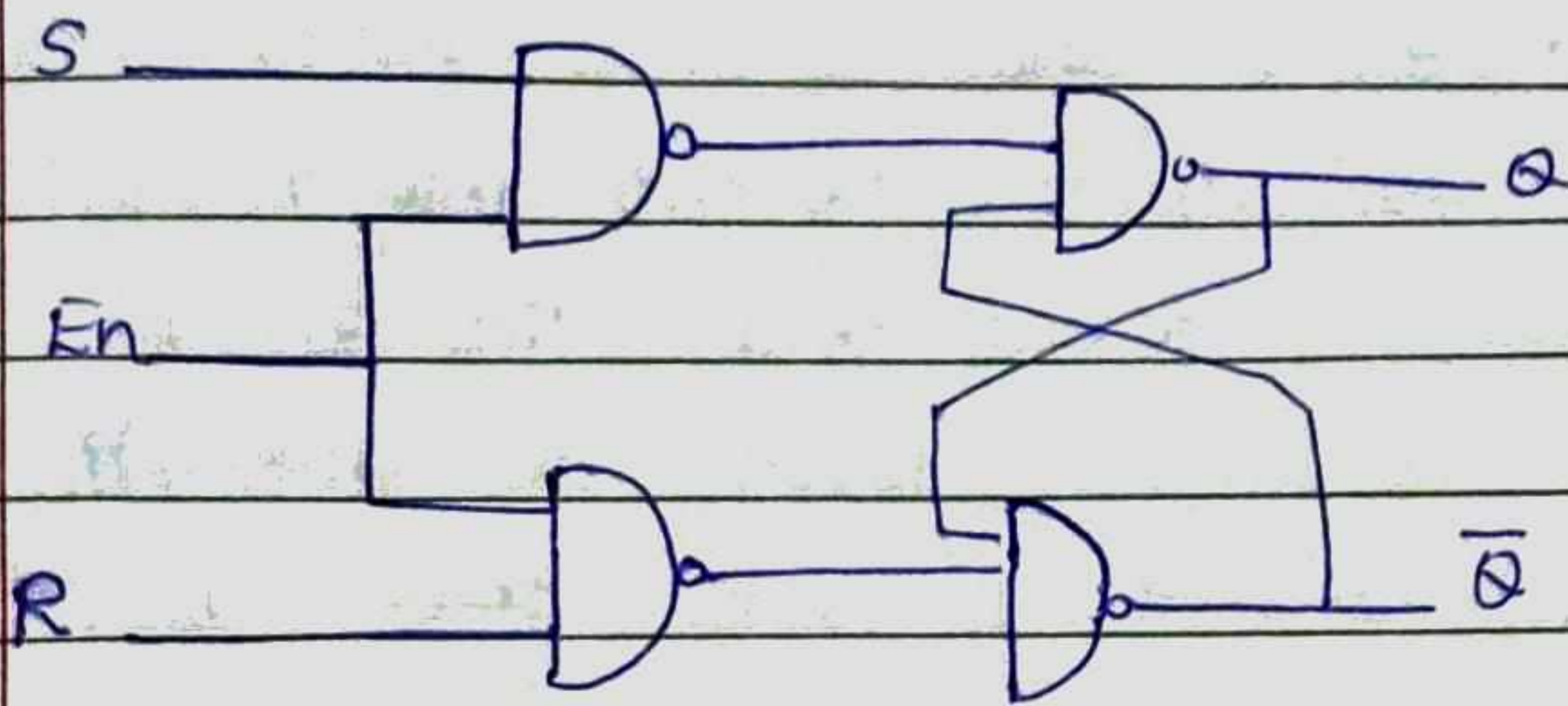
- The flip-flop output changes only in response to the clock, not to a change in i/p.
- If the output can change in response to a 0 to 1 transition on the clock input, we say that the flip-flop is triggered on the rising edge (or positive edge) of the clock.
- If the output can change in response to a 1 to 0 transition on the clock input, we say that the flip-flop is triggered on the falling edge (or negative edge) of the clock.
- An inversion bubble on the clock input indicates a falling-edge trigger, and no bubble indicates a rising-edge trigger.
- The term active edge refers to the clock edge (rising or falling) that triggers the flip-flop state change.

Latches and Flip Flops

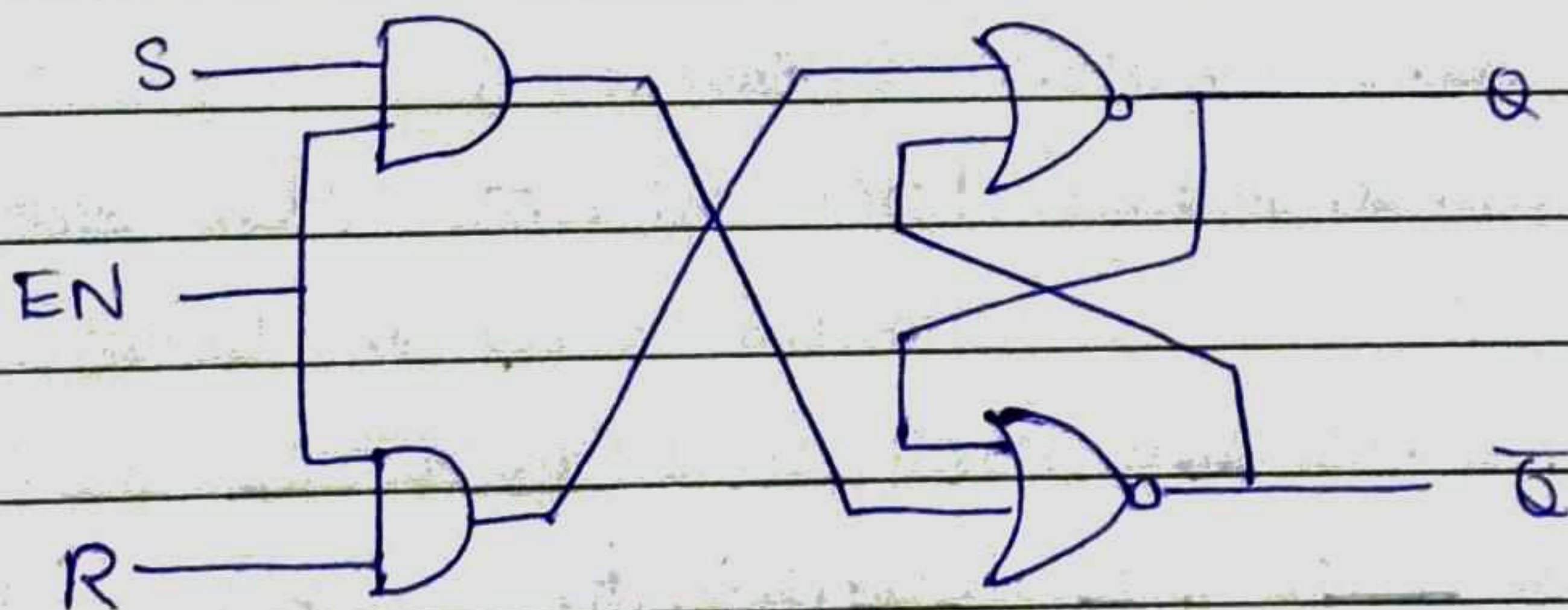
⇒ Types of Flip Flops

- | | | |
|---|-------|--------------------------------|
| 1 | SR FF | Def ⁿ |
| 2 | JK FF | Logic diagram |
| 3 | D FF | IEEE Symbol/Logic symbol |
| 4 | T FF | Truth Table |
| | | Timing diagram |
| | | Characteristic table |
| | | Characteristic eq ⁿ |
| | | Excitation table |
| | | Circuit |

Note:

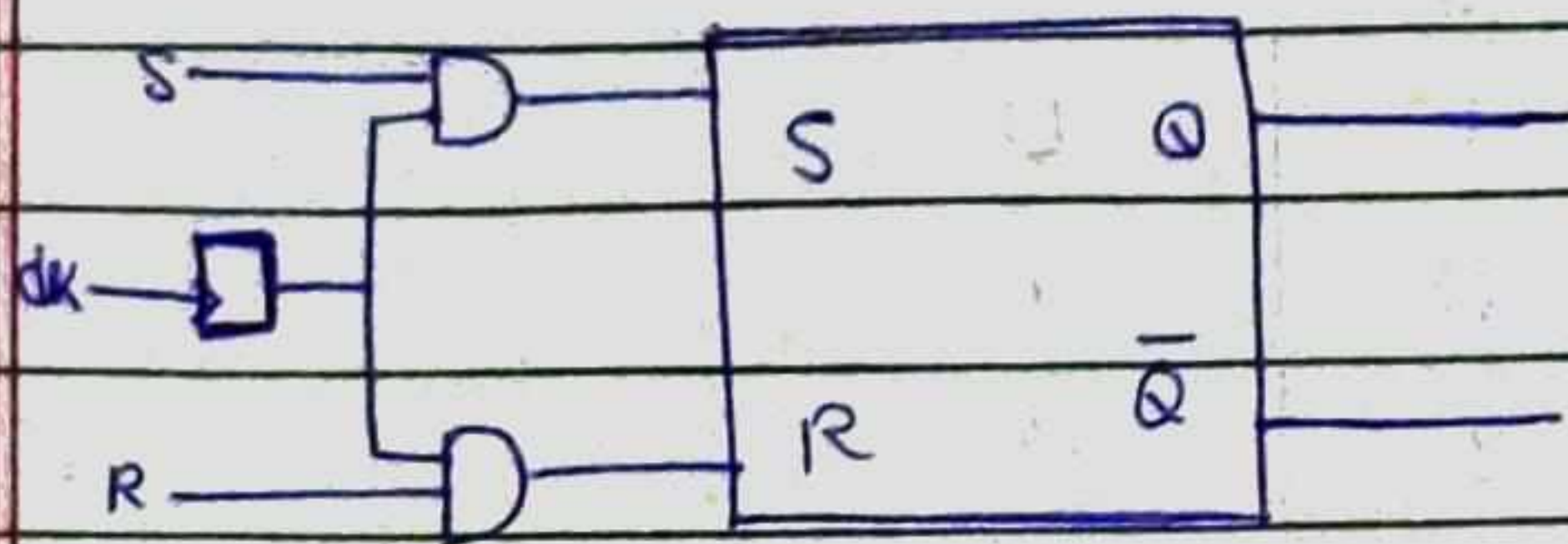


en.

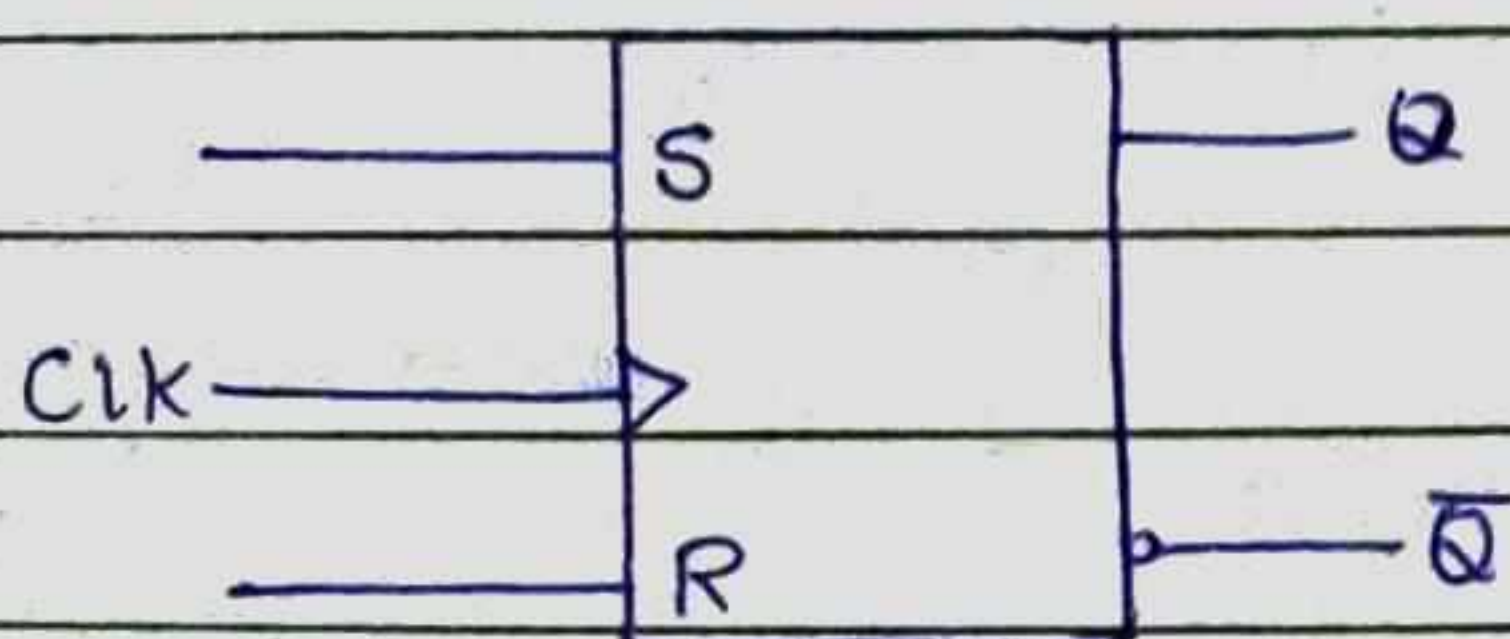


SR- Set Reset FlipFlop. SR FF

i) Logic diagram



ii) IEEE Symbol.



iii) Truth Table.

CLK	S	R	Q_{n+1}
↑	0	0	Q_n No
↑	0	1	0 Reset
↑	1	0	1 Set
↑	1	1	? Illegal.

iv) Timing diagram



v) Characteristic table.

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

vii) Characteristic eqⁿ:

$Q_n \backslash SR$	00	01	11	10
0	0	0	X	1
1	1	0	X	1

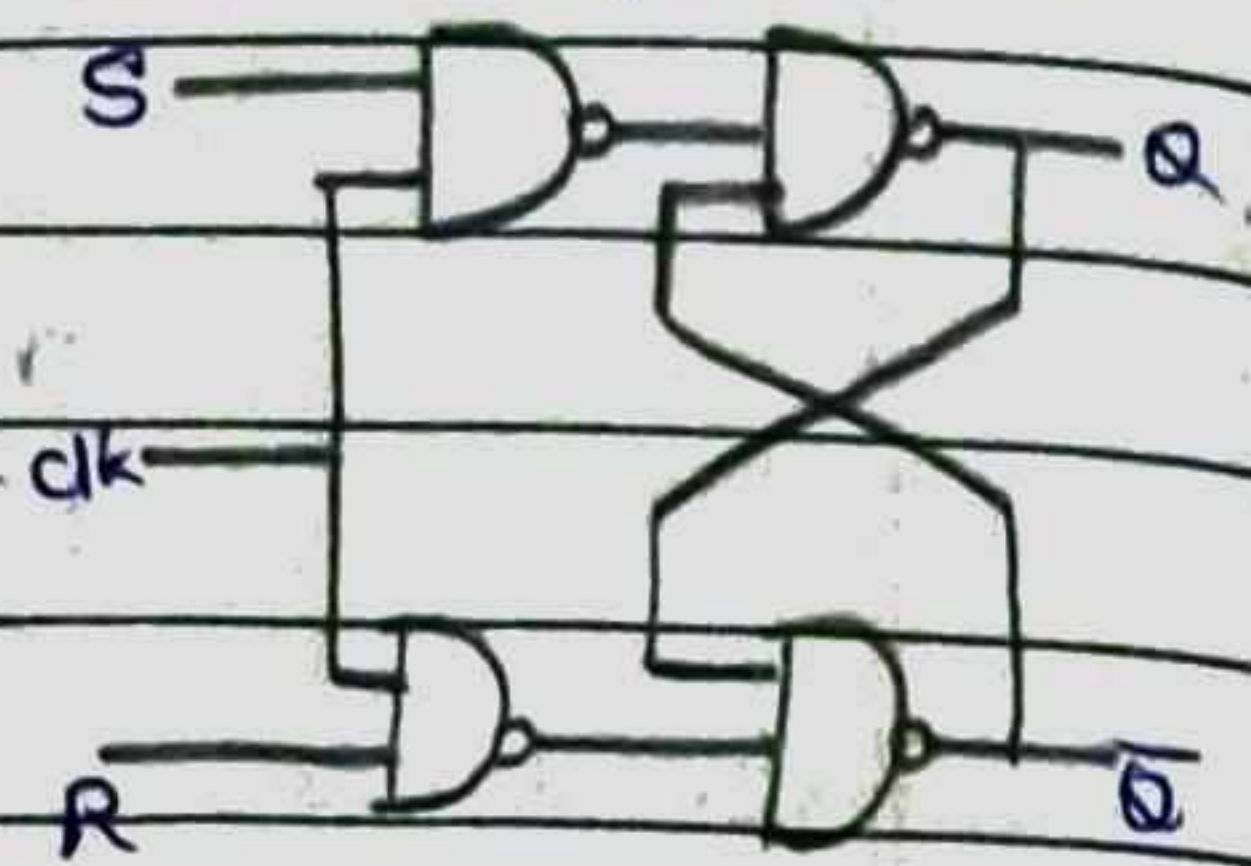
$$Q_{n+1} = S + Q_n R'$$

viii) Excitation table.

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

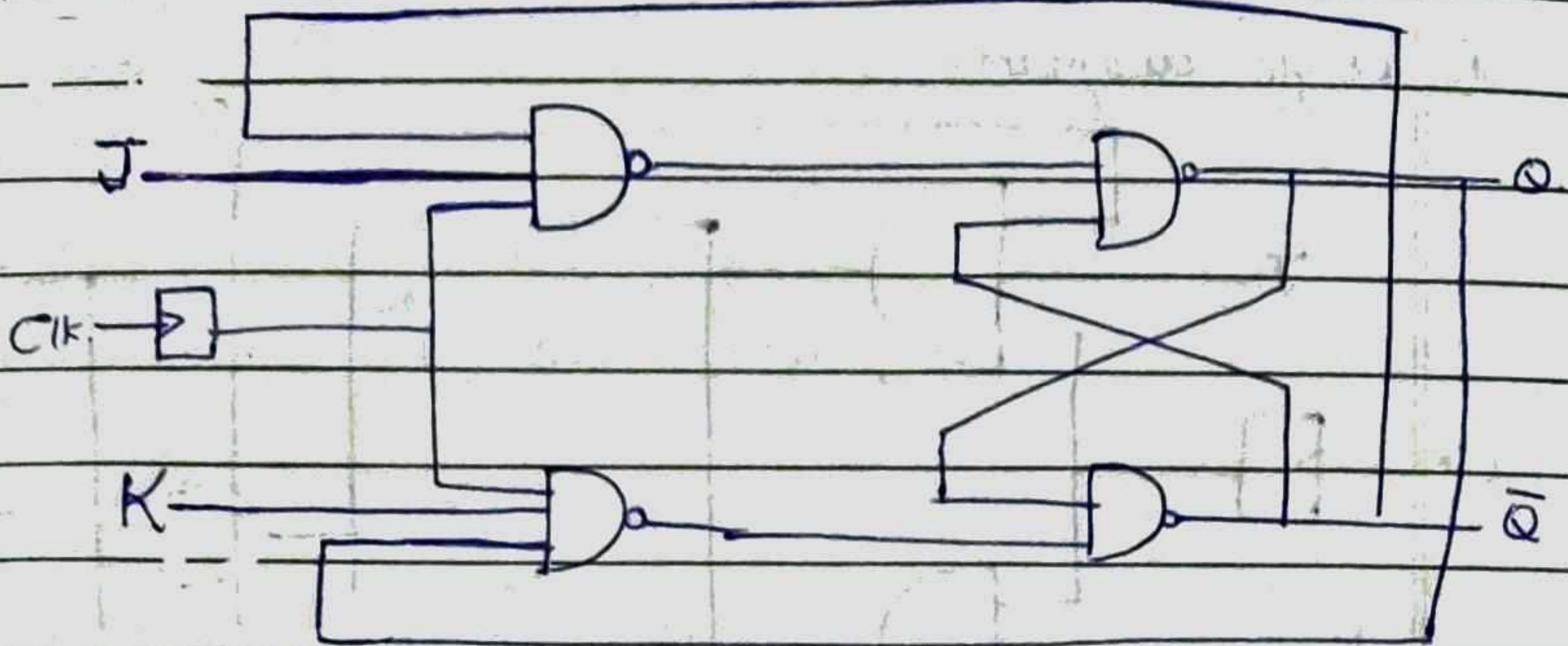
viii) Logic circuit.

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



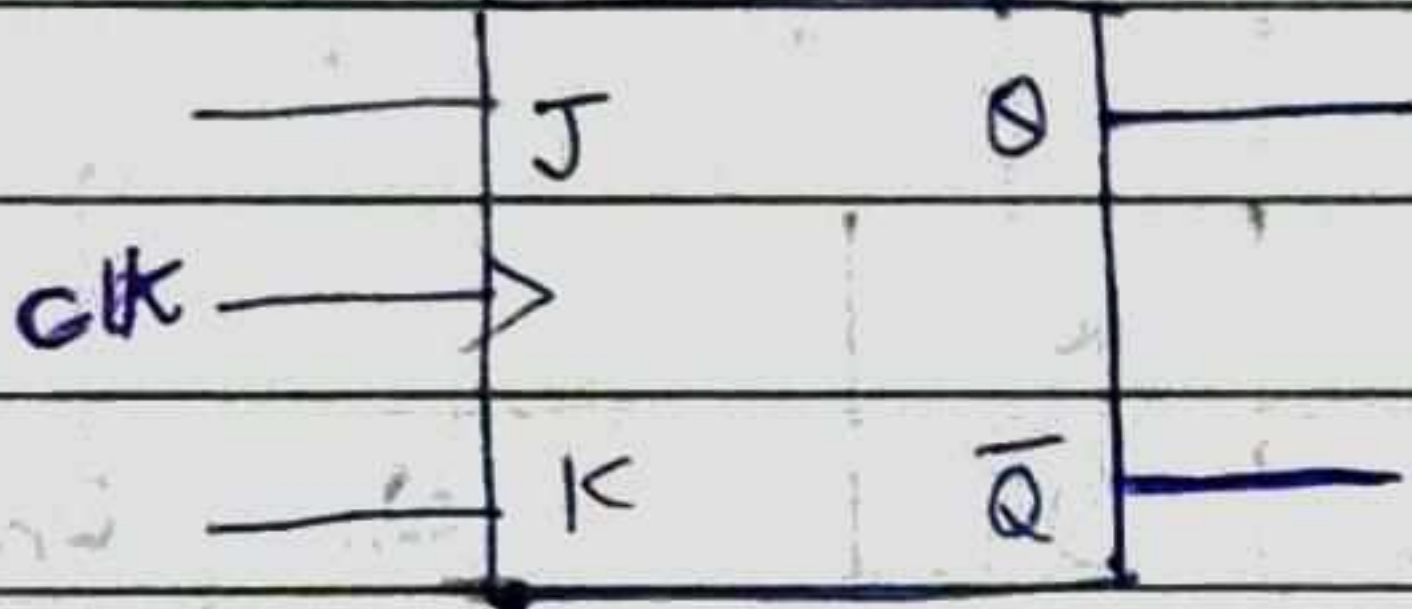
⇒ J K Flip Flop.

1) Logic circuit



2)

Logic symbol.



3) Characteristic table / Next state table.

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

4) Truth Table.

clk	J	K	Q_{n+1}	
↑	0	0	Q_n	NC
↑	0	1	0	Reset
↑	1	0	1	Set
↑	1	1	$\overline{Q_n}$	Toggle

5) Characteristic equation.

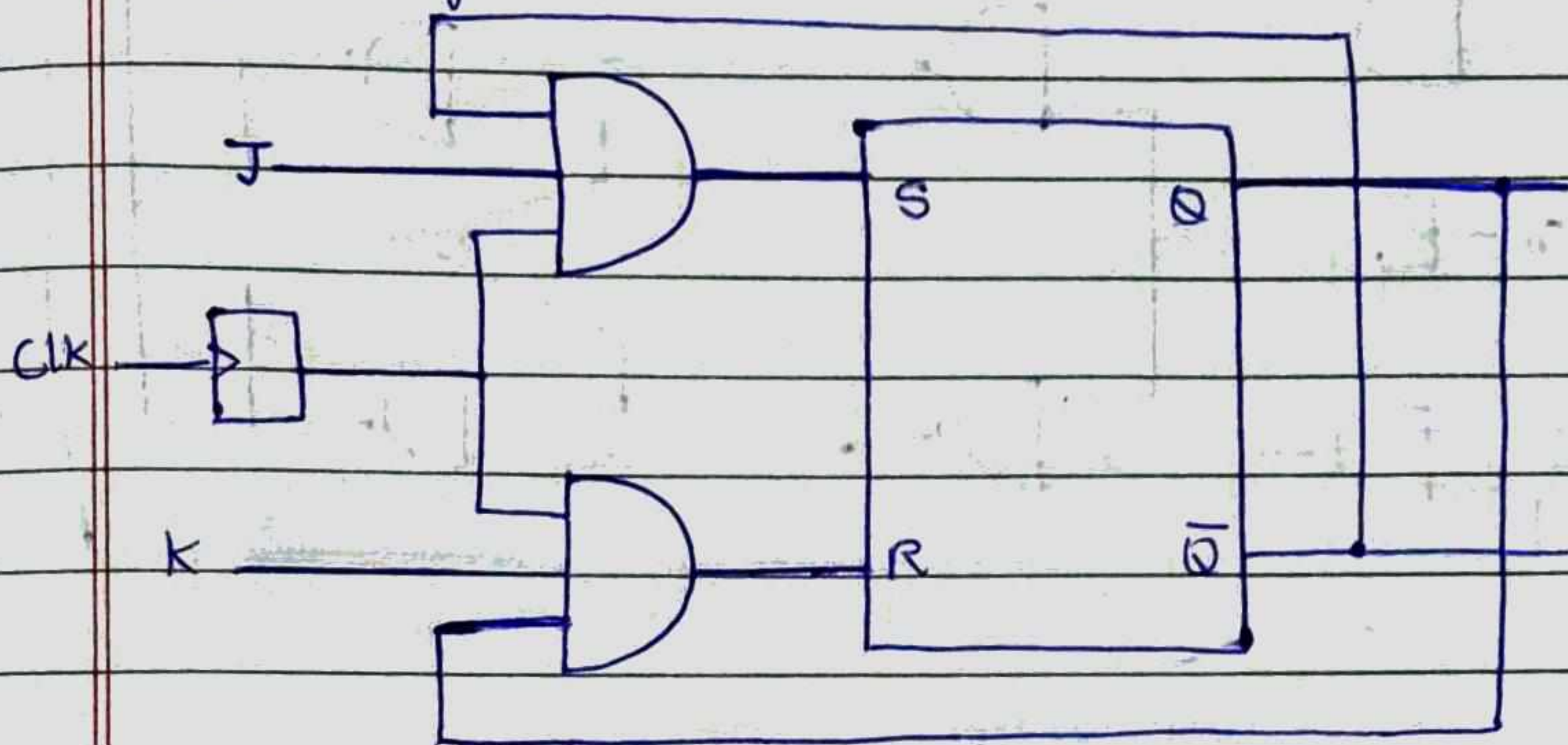
JK

Q_n

	00	01	11	0
0	0	0	1	1
1	1	0	0	1

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

6 Logic diagram.



7 Excitation table.

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

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

Note

1. Q or $Q_n \Rightarrow$ Current State.

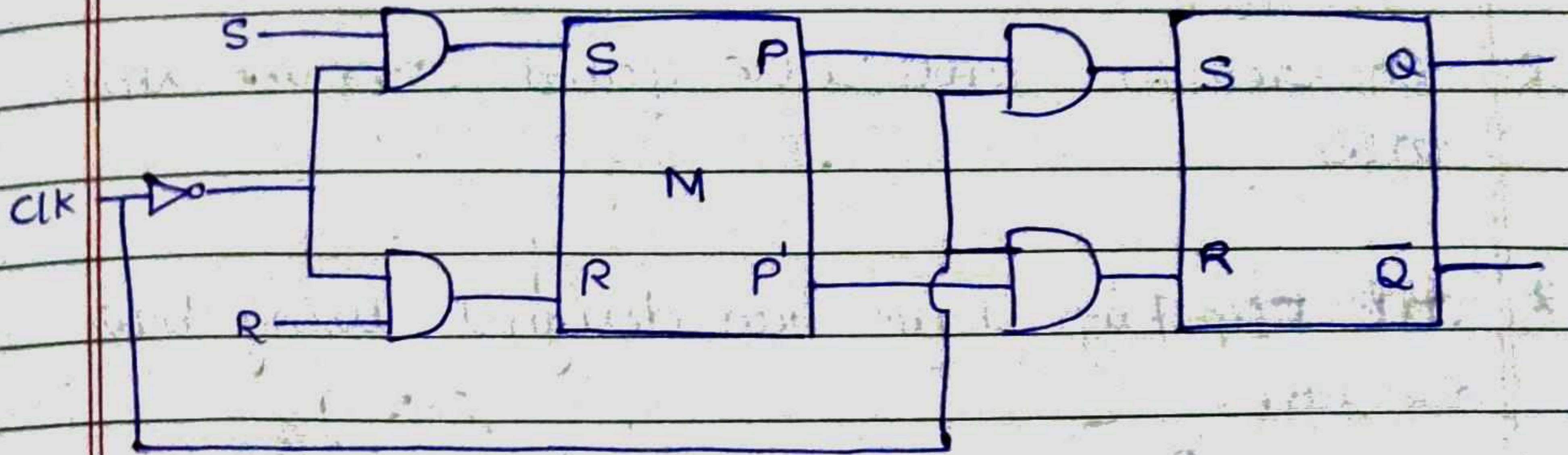
Q^+ or $Q_{n+1} \Rightarrow$ Next State.

2. Characteristic table also called as next state table.

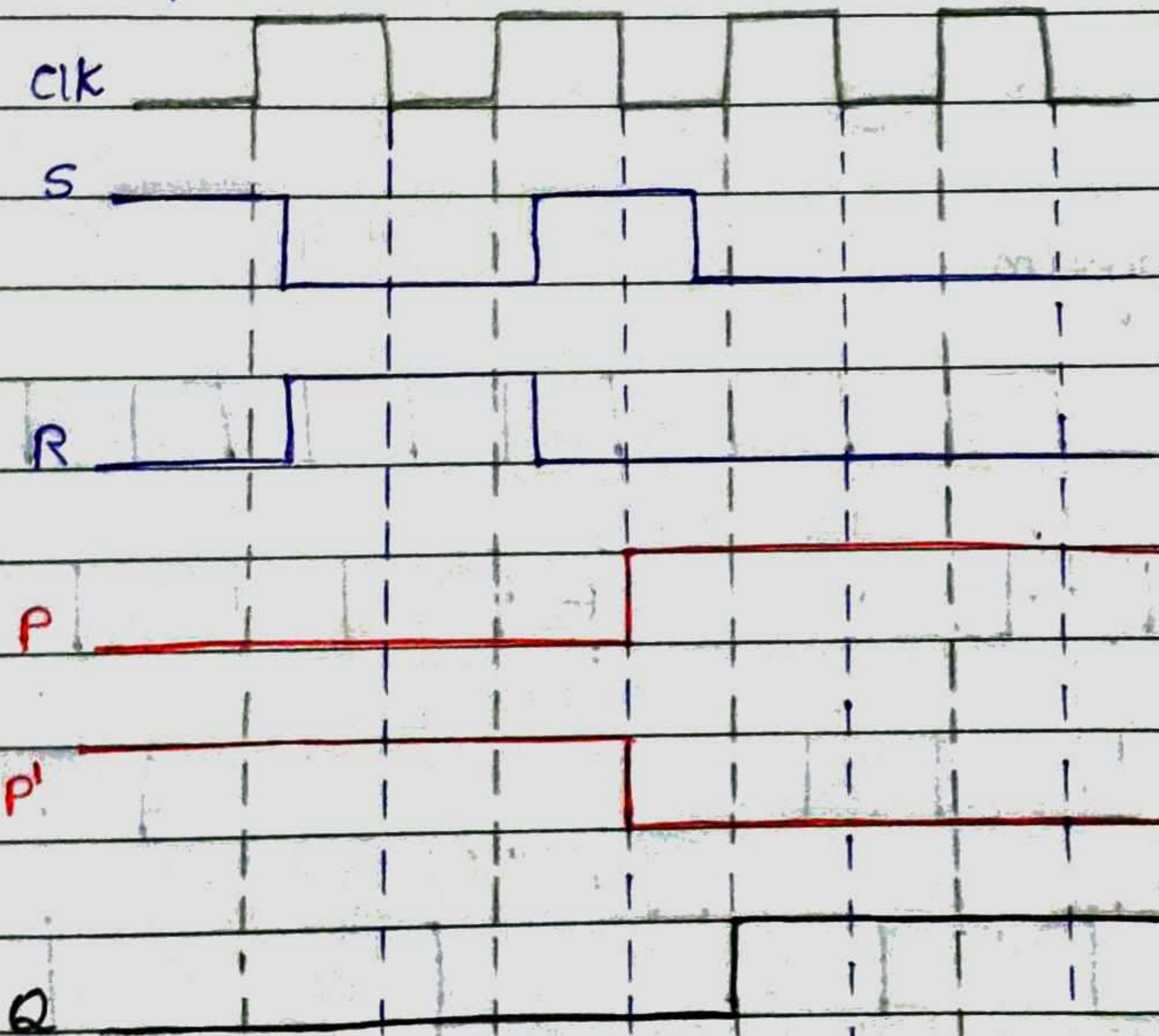
3. All Flip Flop have been designed using basic SR latch.

4. Positive edge or rising edge.
Negative edge or falling edge

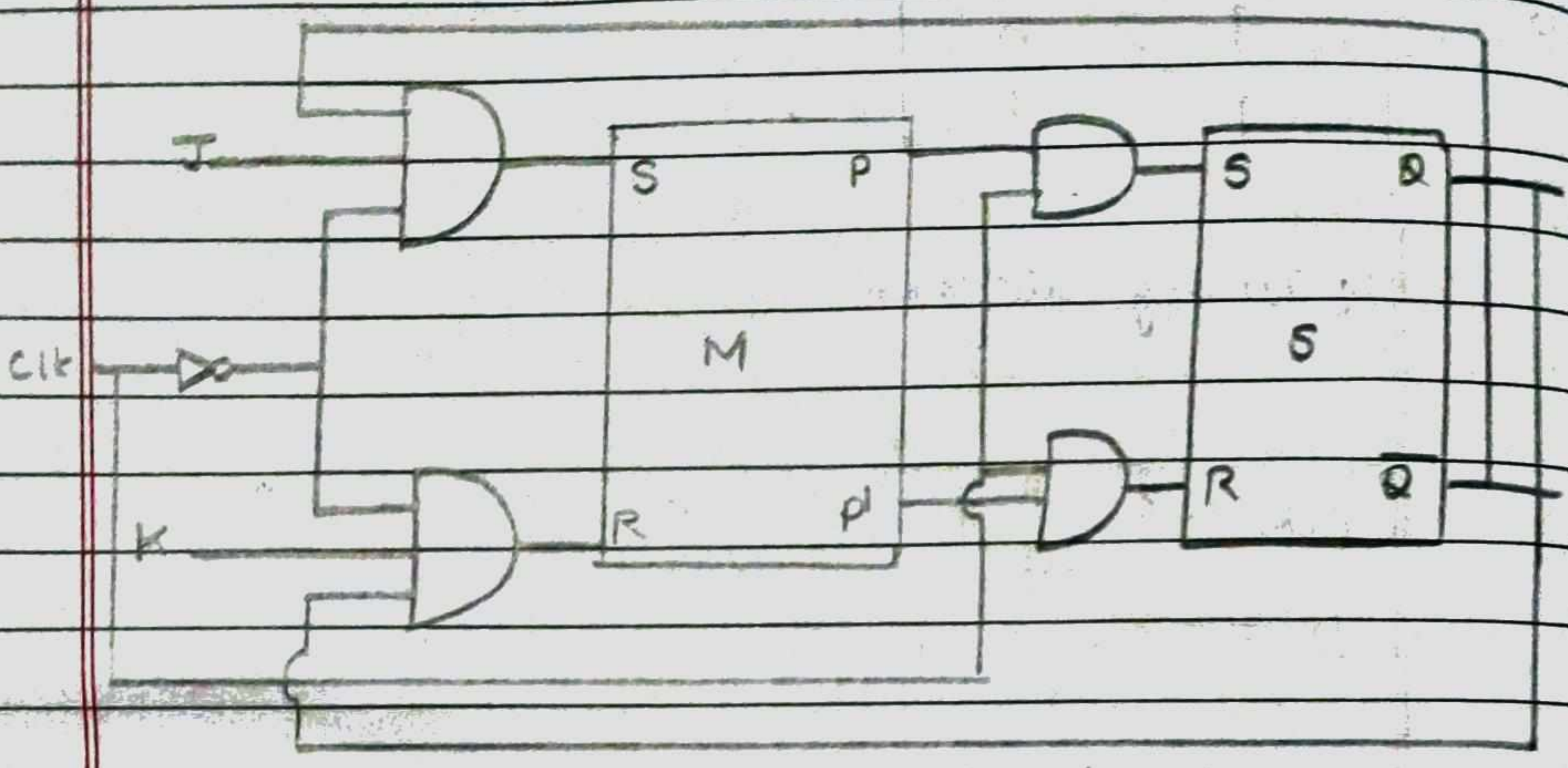
⇒ Implement Master Slave SR FlipFlop using two SR latches & gates.



Timing diagram

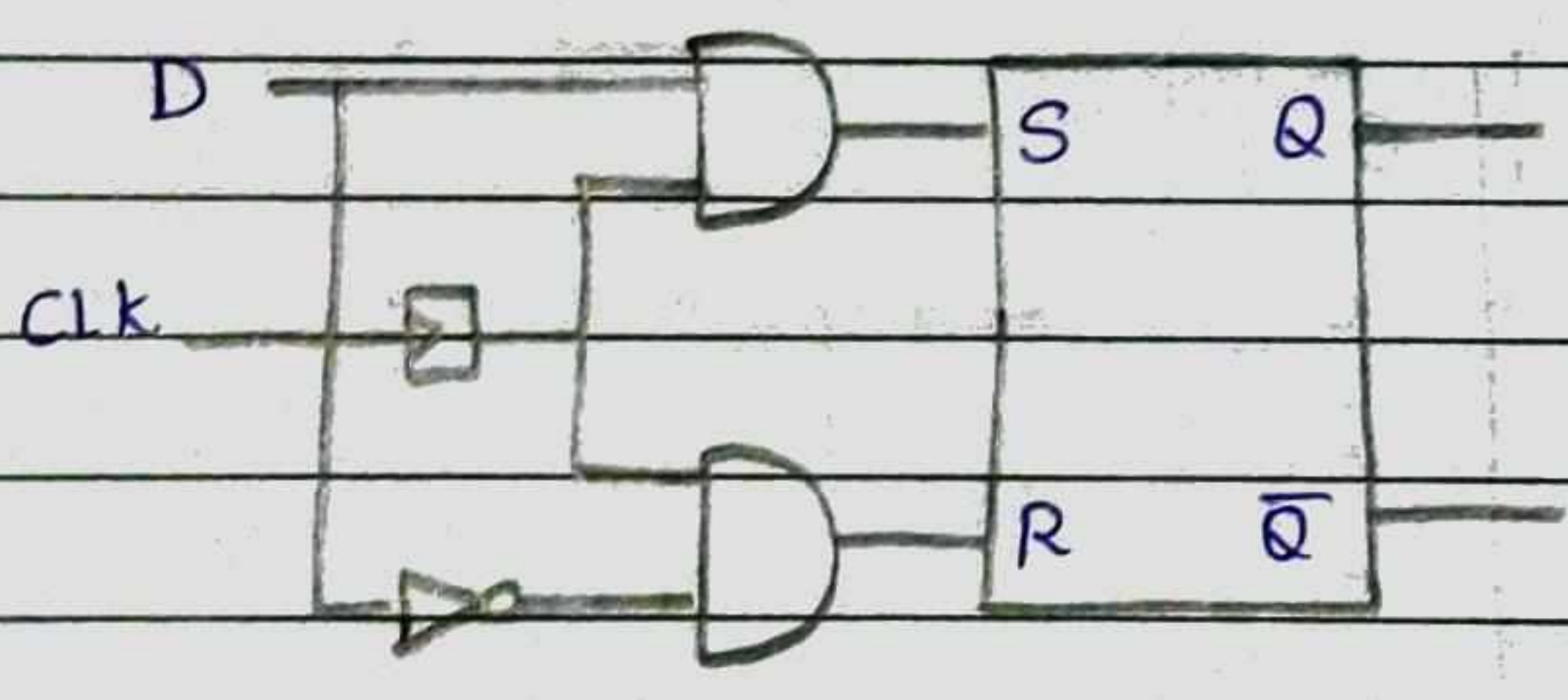


⇒ Implement JK Master slave using 2 SR latches & gates.

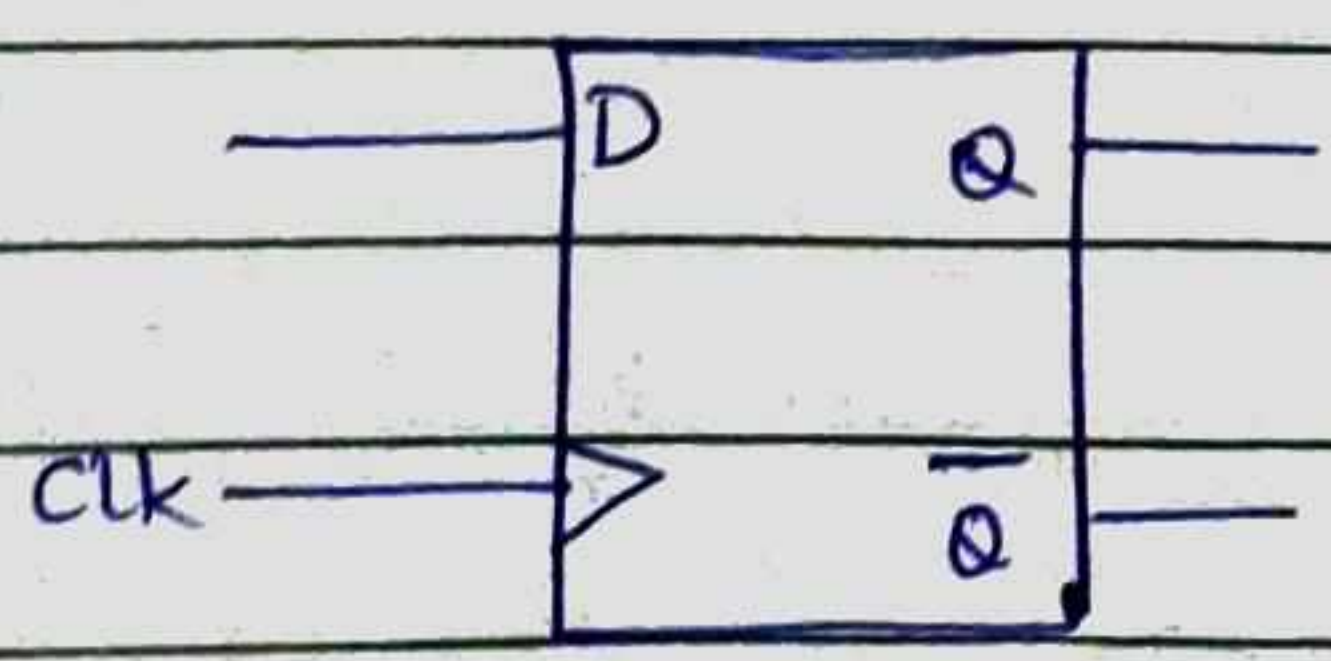


⇒ D Flip Flop : D - Data

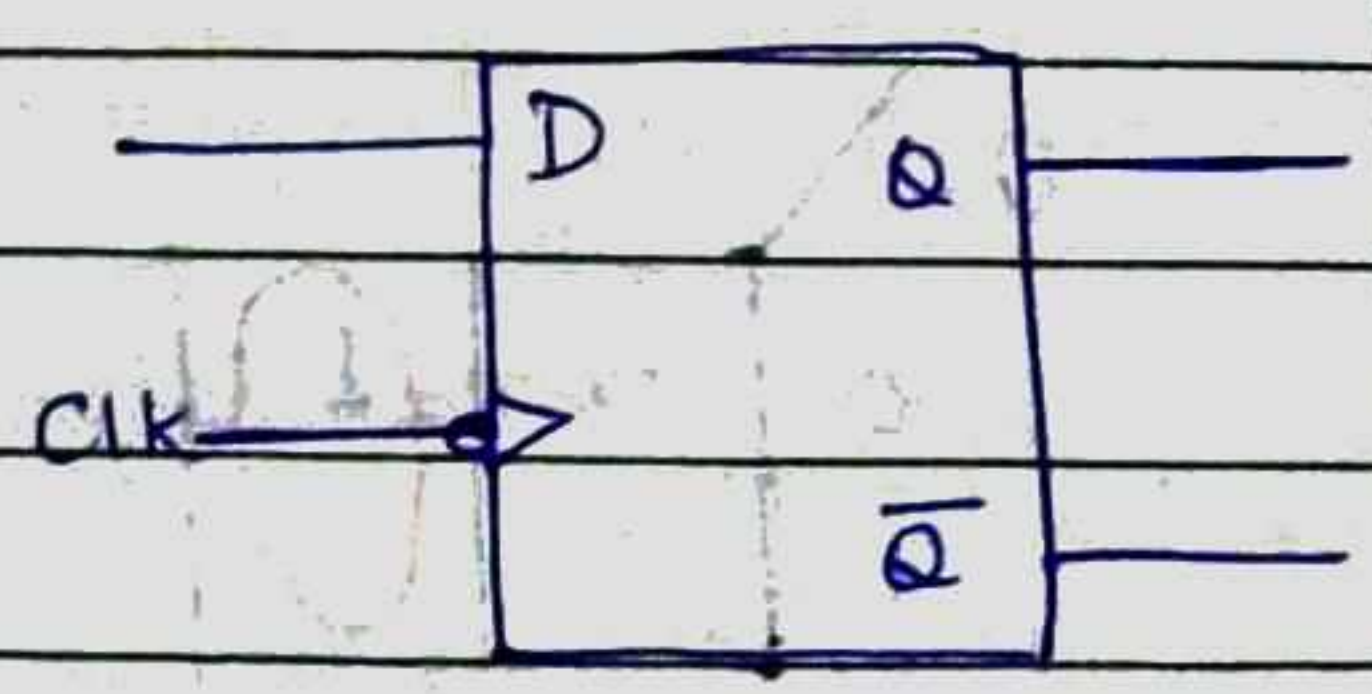
1) Logic diagram.



2) IEEE symbol / Logic.



Positive edge triggered.



Negative edge triggered.

3) Truth Table

CLK	D	Q_{n+1}
↑	0	0
↑	1	1

4) Timing diagram



5) Characteristic/Next State table

Q_n	D	Q_{n+1}
0	0	0
0	1	1
1	0	0
1	1	1

6) Characteristic equation

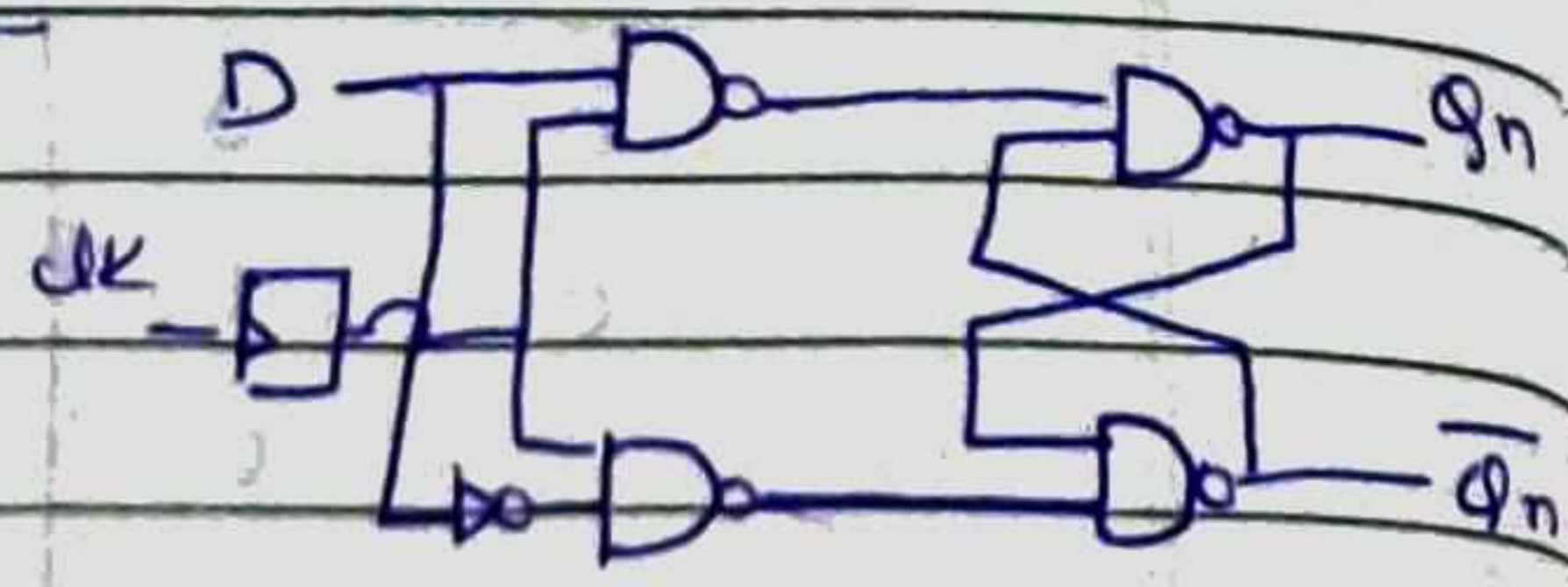
$Q_n \backslash D$	0	1
0	0	1
1	0	1

$$Q_{n+1} = D$$

7) Excitation table

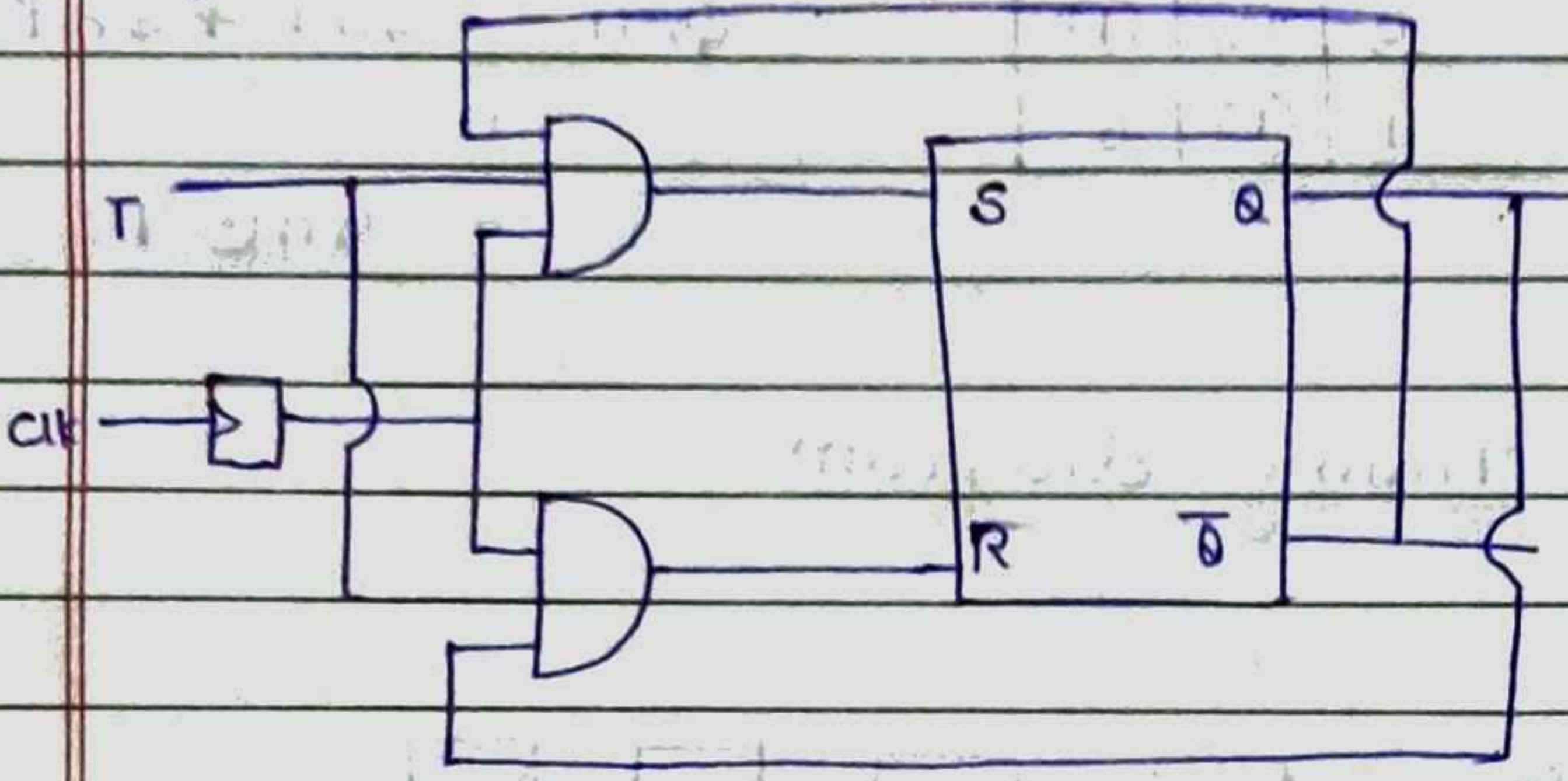
8) Logic CKT

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

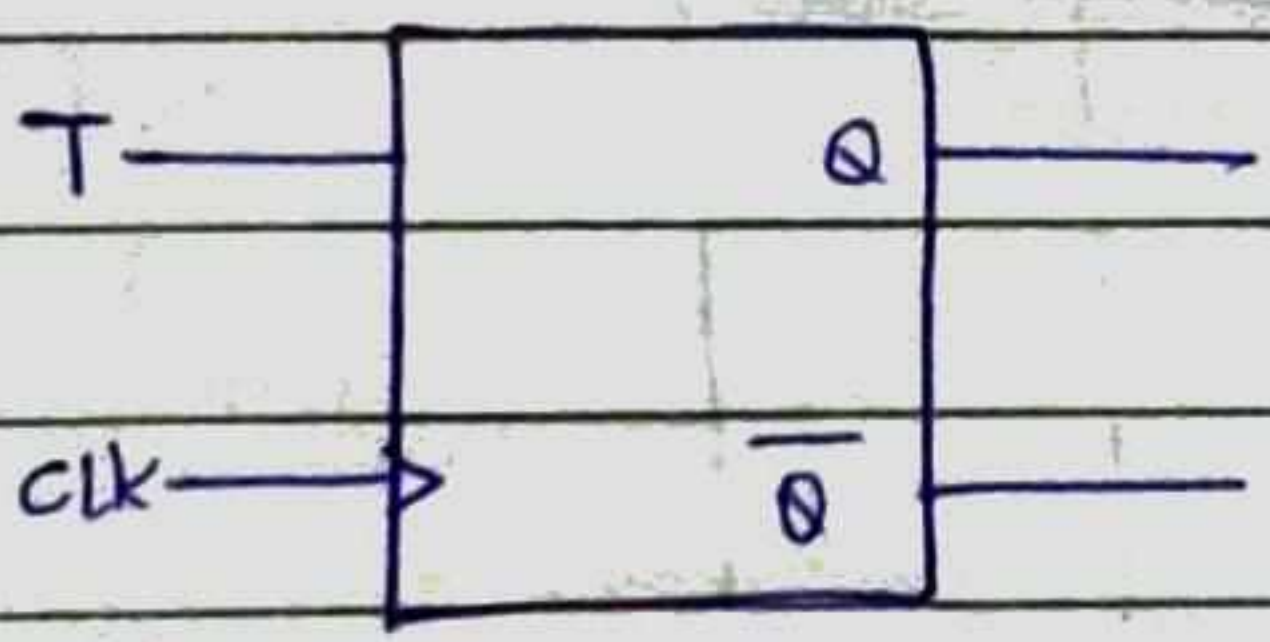


⇒ **T-Flip Flop** T-Toggle

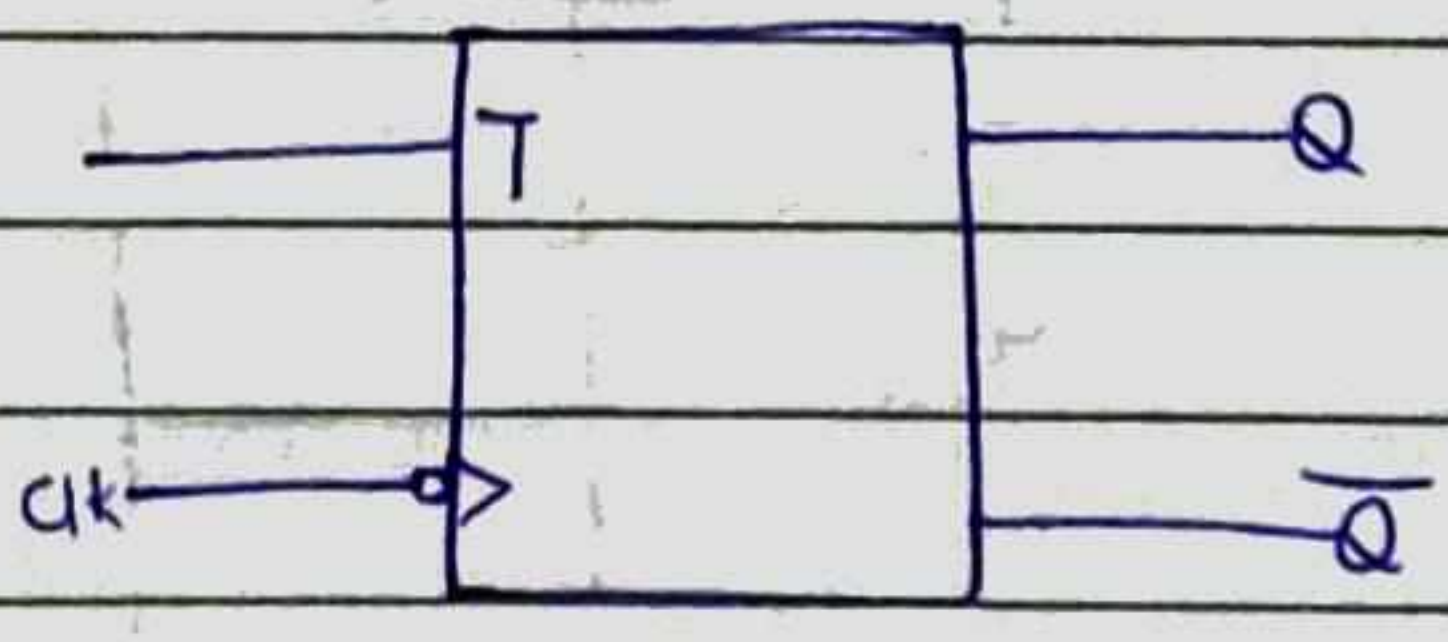
1. Logic diagram



2. IEEE symbol



+ve edge triggered



-ve edge triggered

3. Truth table

clk	T	Q_{n+1}
↑	0	(Nochange) Q_n
↑	1	(Toggle) $\overline{Q_n}$

4) characteristic table.

8) Excitation table

Q_n	T	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

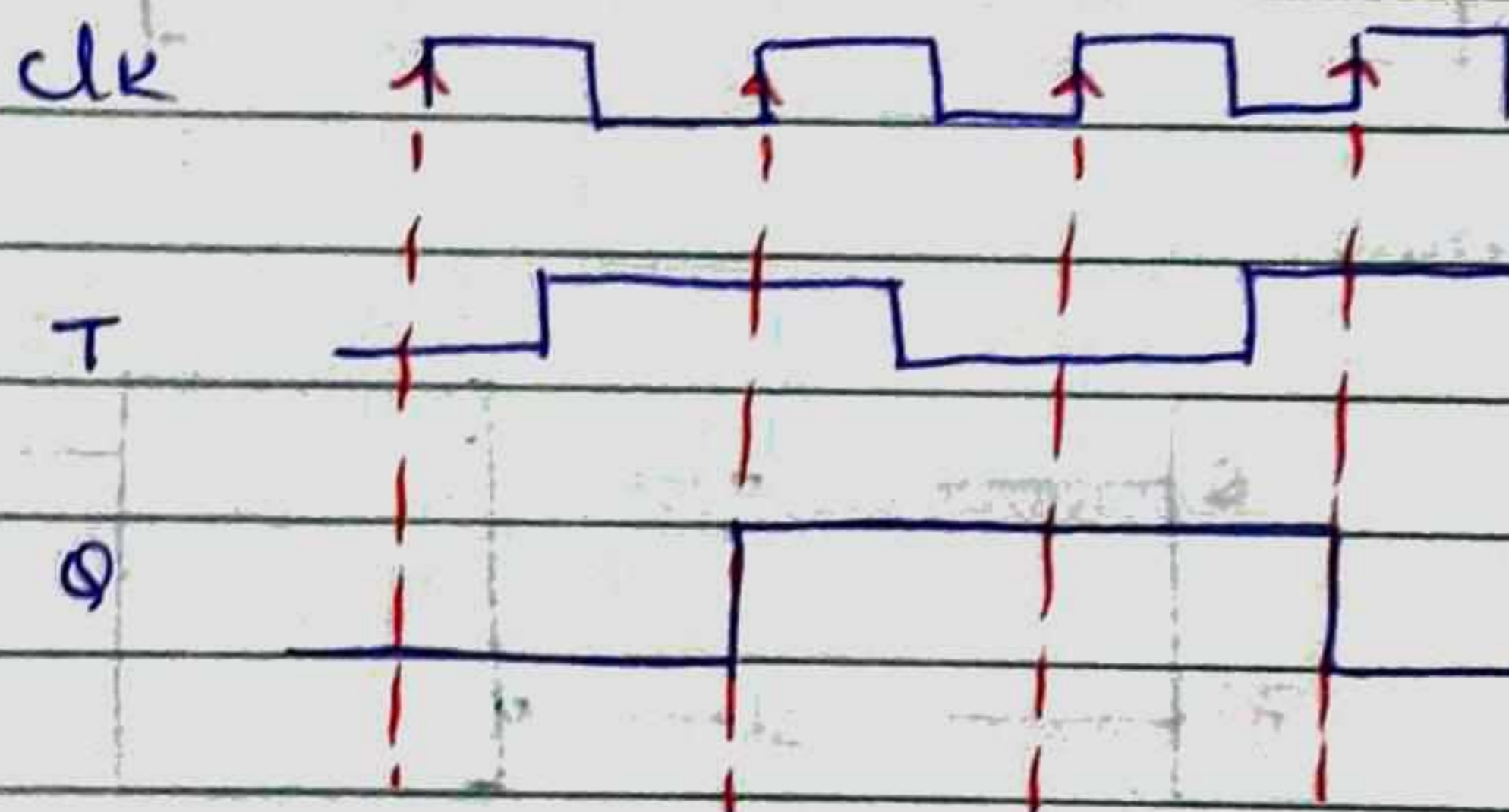
5) characteristic equation

Q_n	T	
	0	1
0	0	0
0	1	1
1	0	1
1	1	0

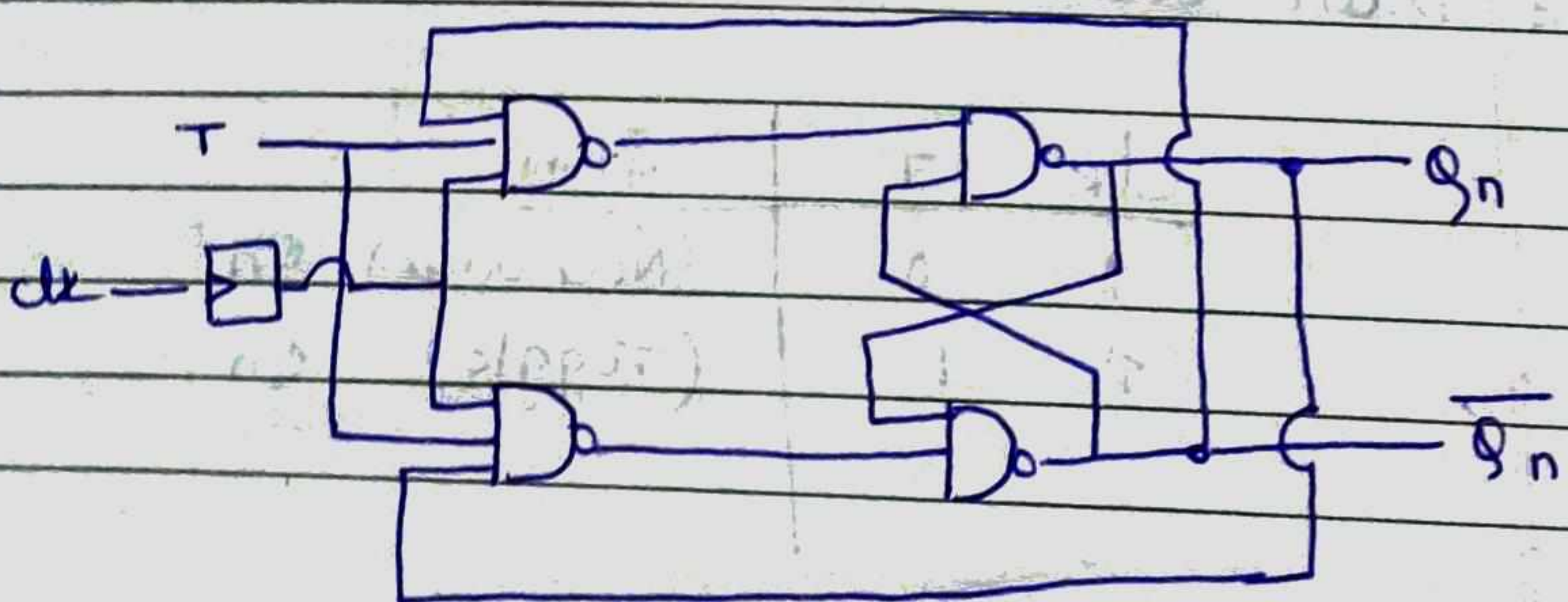
$$Q_{n+1} = \bar{Q}_n T + Q_n \bar{T}$$

$$= Q_n \oplus T$$

6) Timing diagram



7) logic ckt



Note:

Characteristic table: finding the next state.

Excitation table: finding the inputs.

Page No.

Date

⇒ Conversion of flipflops.

ET ↘

↙ CT.

1. SR FF to D FF

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

K-map.

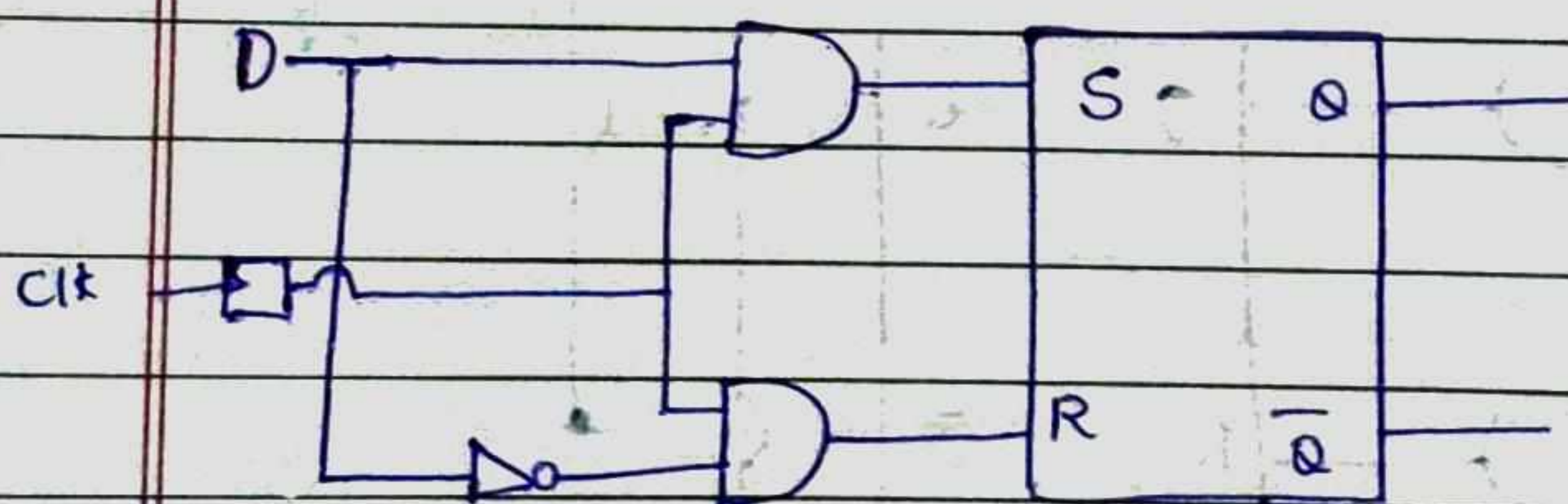
$Q_n \backslash D$	0	1
0	0	1
1	0	X

$$S = D$$

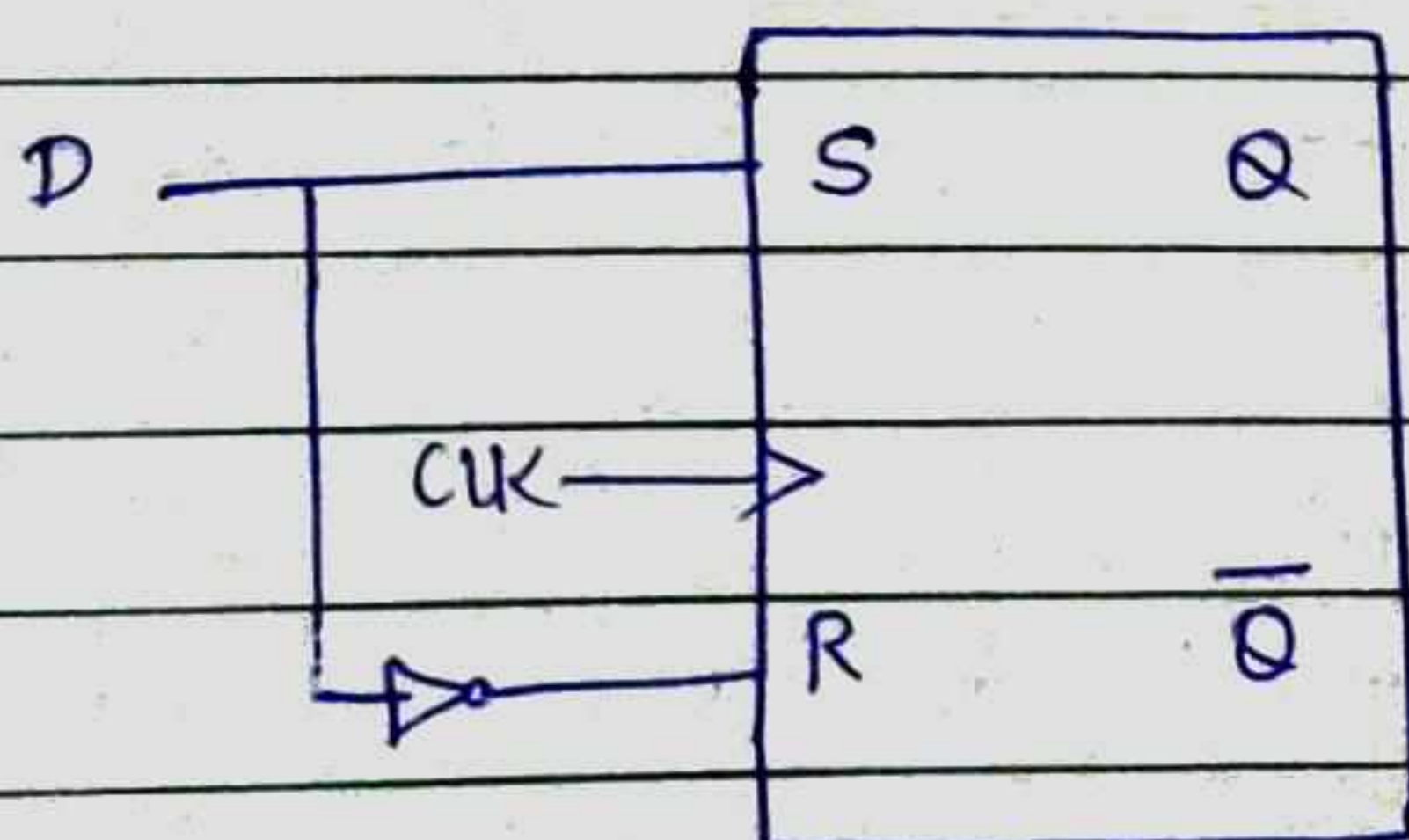
$Q_n \backslash D$	0	1
0	X	0
1	1	0

$$R = \bar{D}$$

Logic diagram



(01)



ET)
 2. JK FF to T.FF

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

K-map

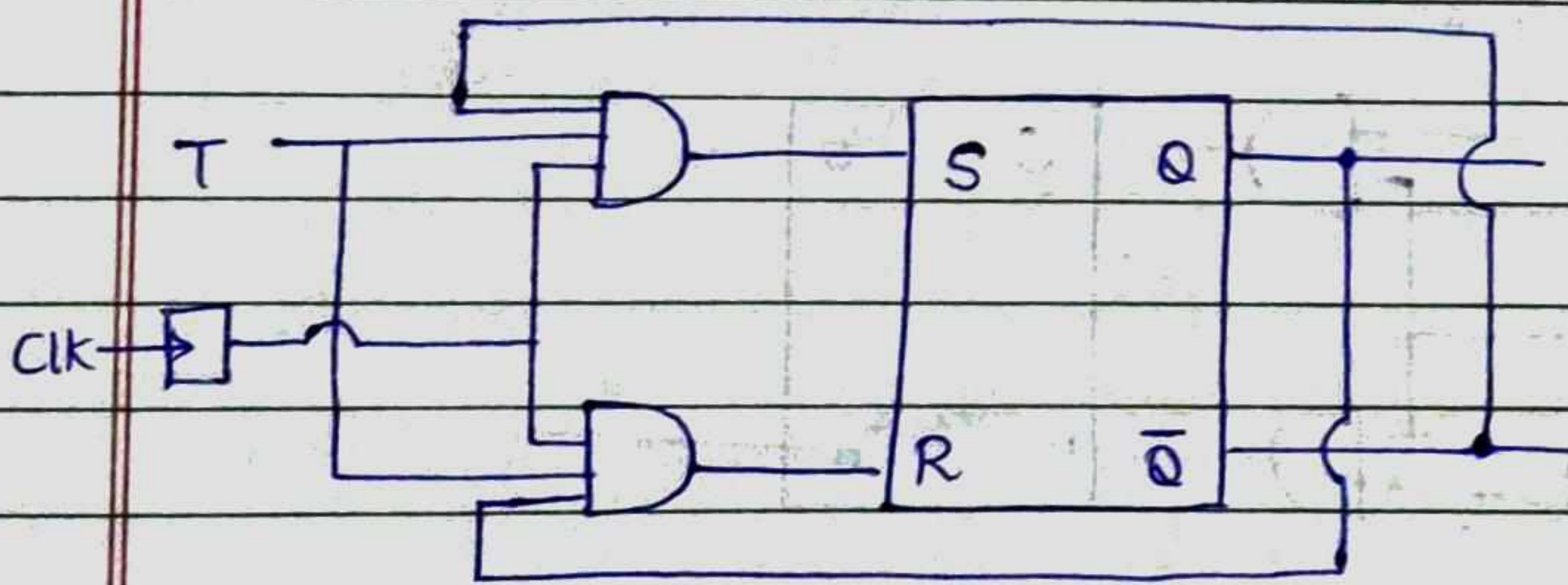
Q_n	T	0	1
0	0	0	1
1	X	X	X

$$J = T$$

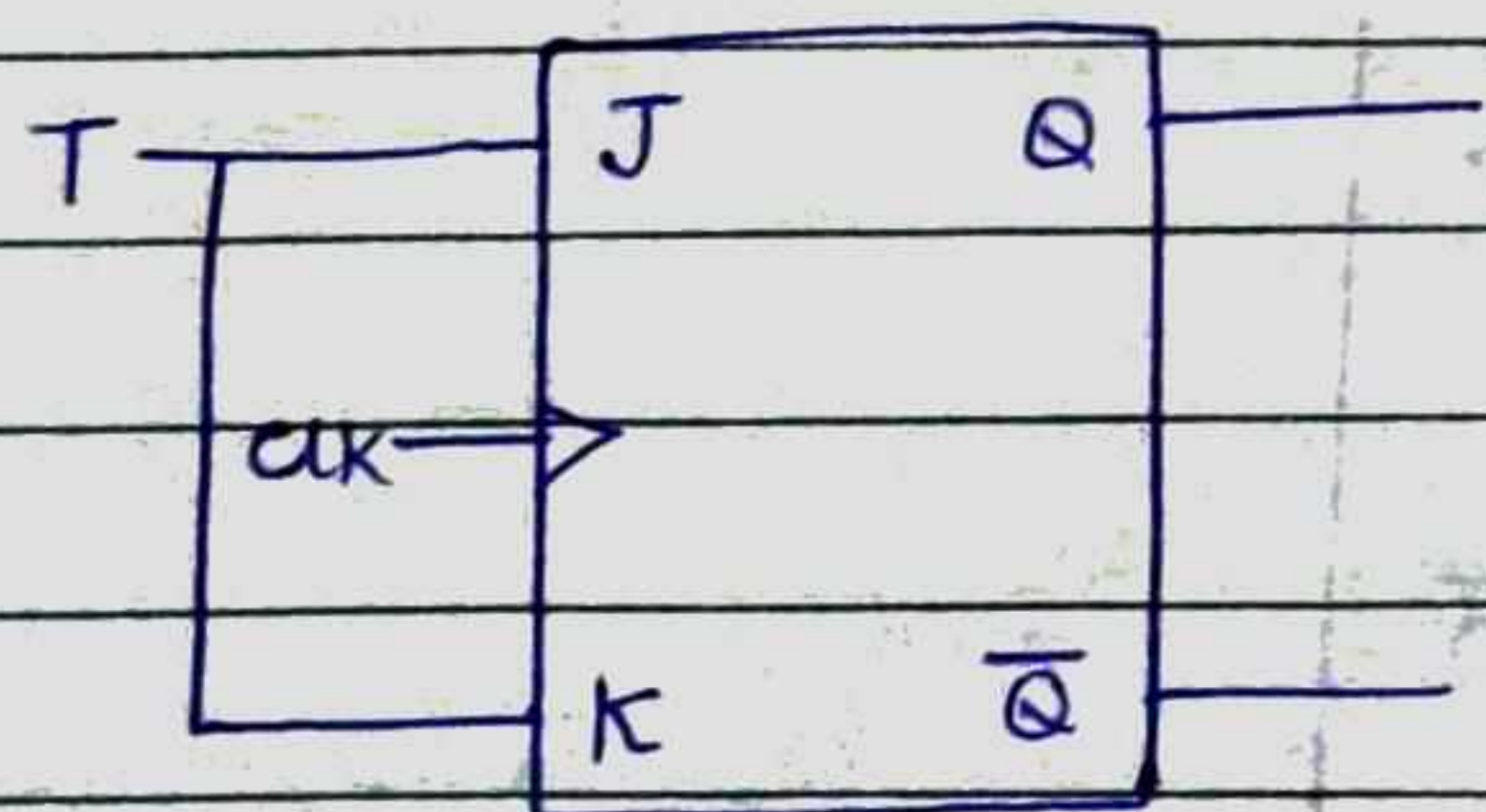
$Q_n \backslash T$	0	1
0	X	X
1	0	1

$$K = T$$

Logic diagram



(or)



3 SR FF to JK FF

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

K-Map

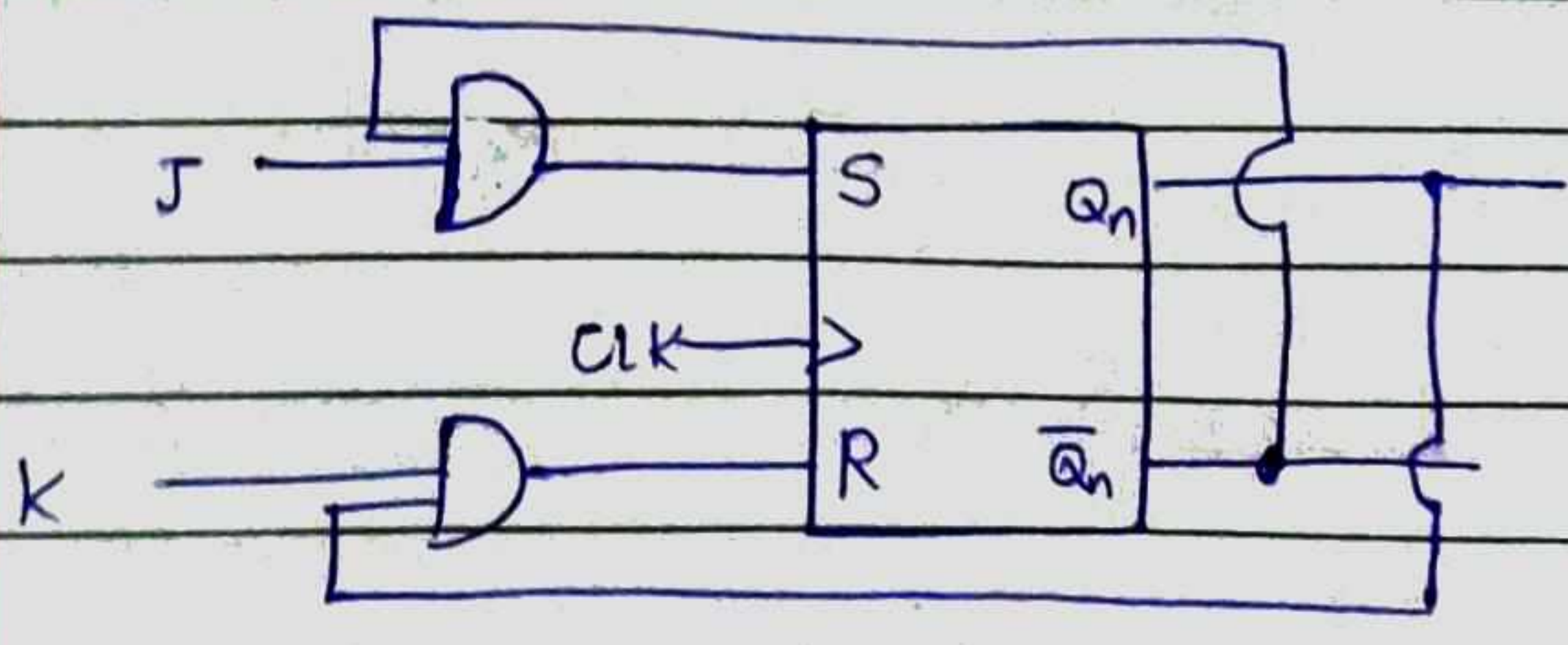
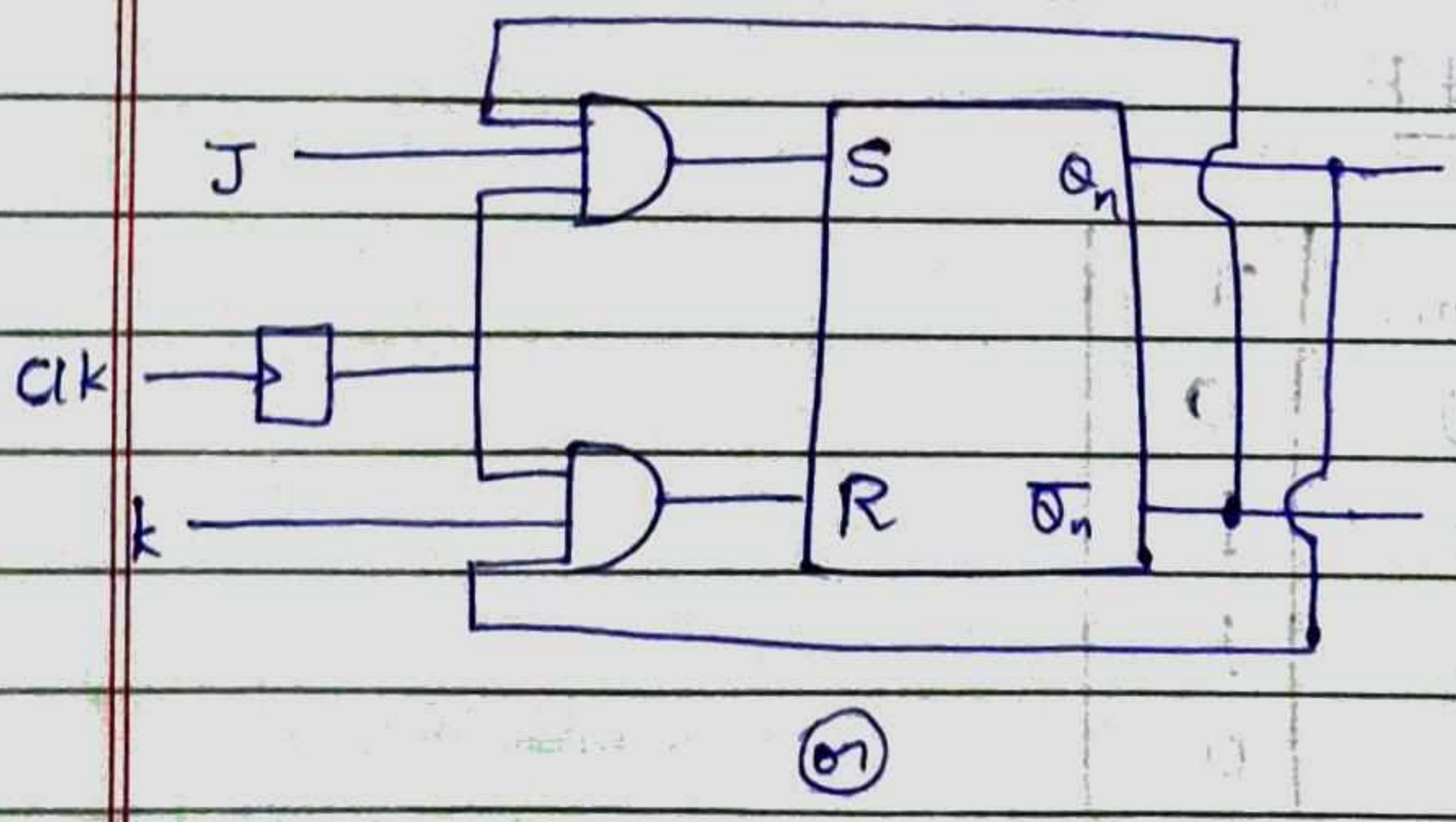
$Q_n \backslash JK$	00	01	11	10
0	0 ₀	0 ₁	1 ₃	1 ₂
1	X ₄	0 ₅	X ₇	0 ₆

$Q_n \backslash JK$	00	01	11	10
0	X	X	0	0
1	0	1	1	0

$$S = \overline{Q_n} J$$

$$R = Q_n K$$

Logic diagram



ET → CT
 ⇒ T-FF to D-FF

Q_n	D	Q_{n+1}	T
0	0	0	0
0	1	1	1
1	0	0	1
1	1	1	0

K-Map

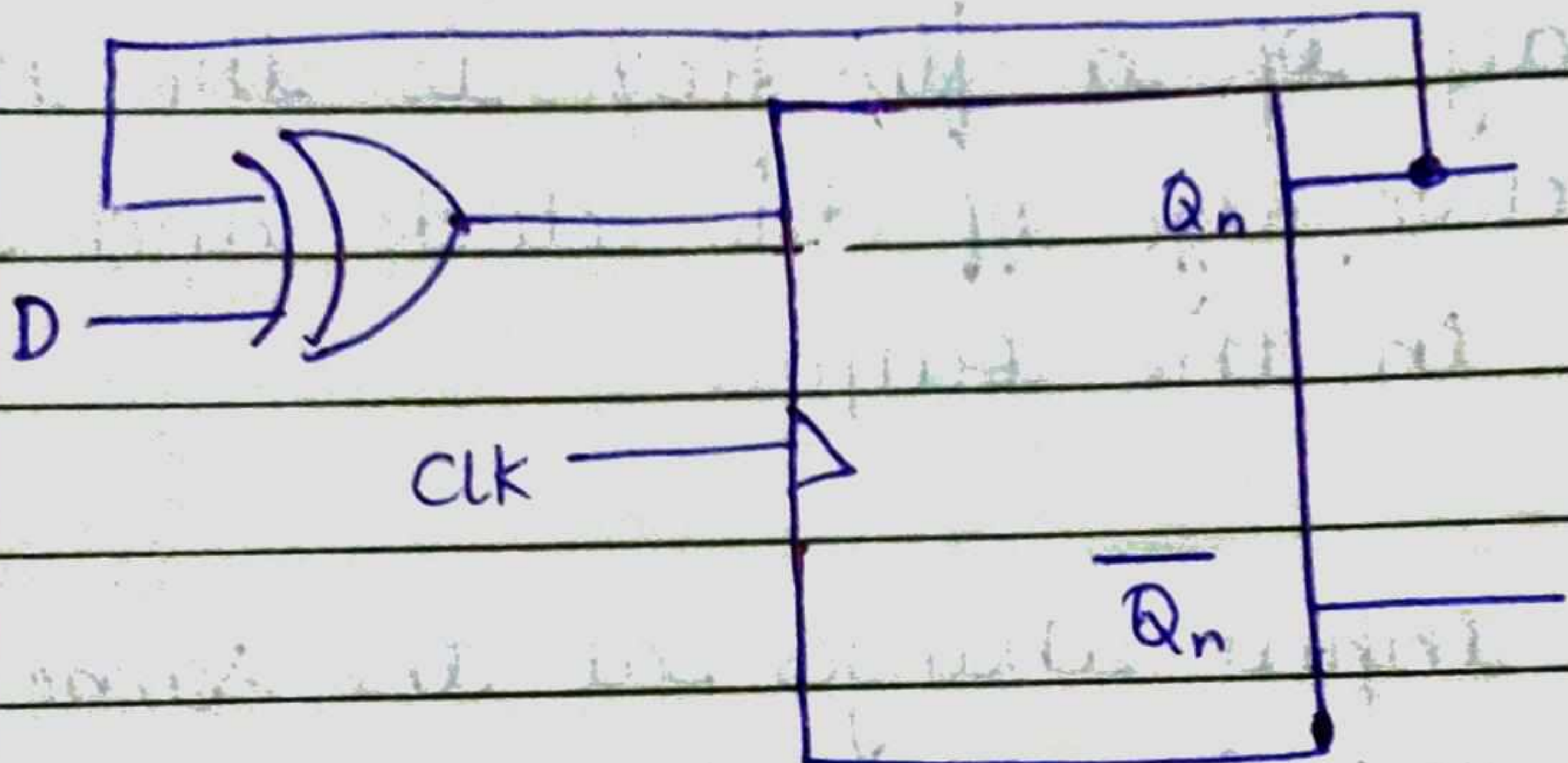
$Q_n \backslash D$	0	1
0	0	1
1	1	0

$$T = \bar{D}Q_n + D\bar{Q}_n$$

$$= Q_n \oplus D$$

25

Logic diagram.



⇒ Timing diagram with propagation delay (t_p)

→ Propagation delay of a flip flop is the time between the active edge of the clock and the resulting change in the output.

→ If the flip-flop input changes at the same time as the active edge, the behaviour is unpredictable.

Consider the timing issues associated with the D-flip flop.

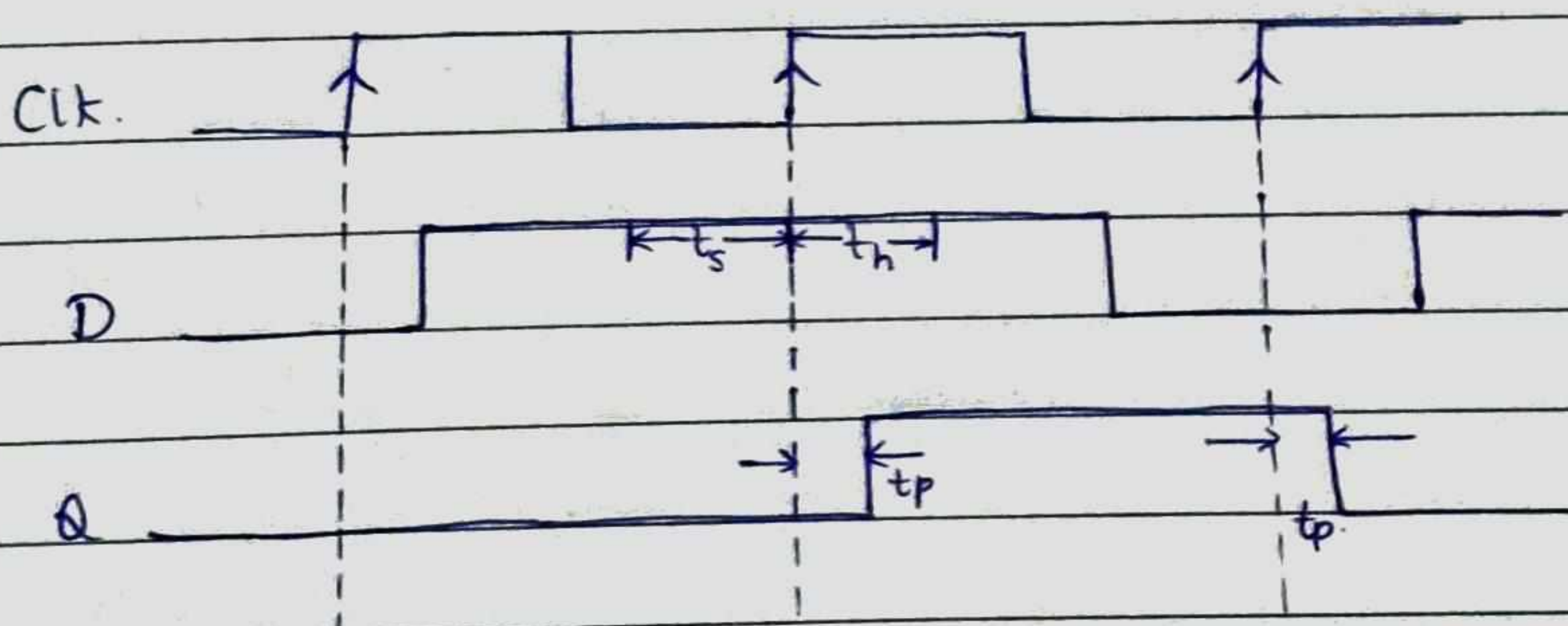
→ Setup time (t_{se})

The amount of time D must be stable before the active edge.

→ Hold time (t_h)

The amount of time the D must hold the same value after the active edge.

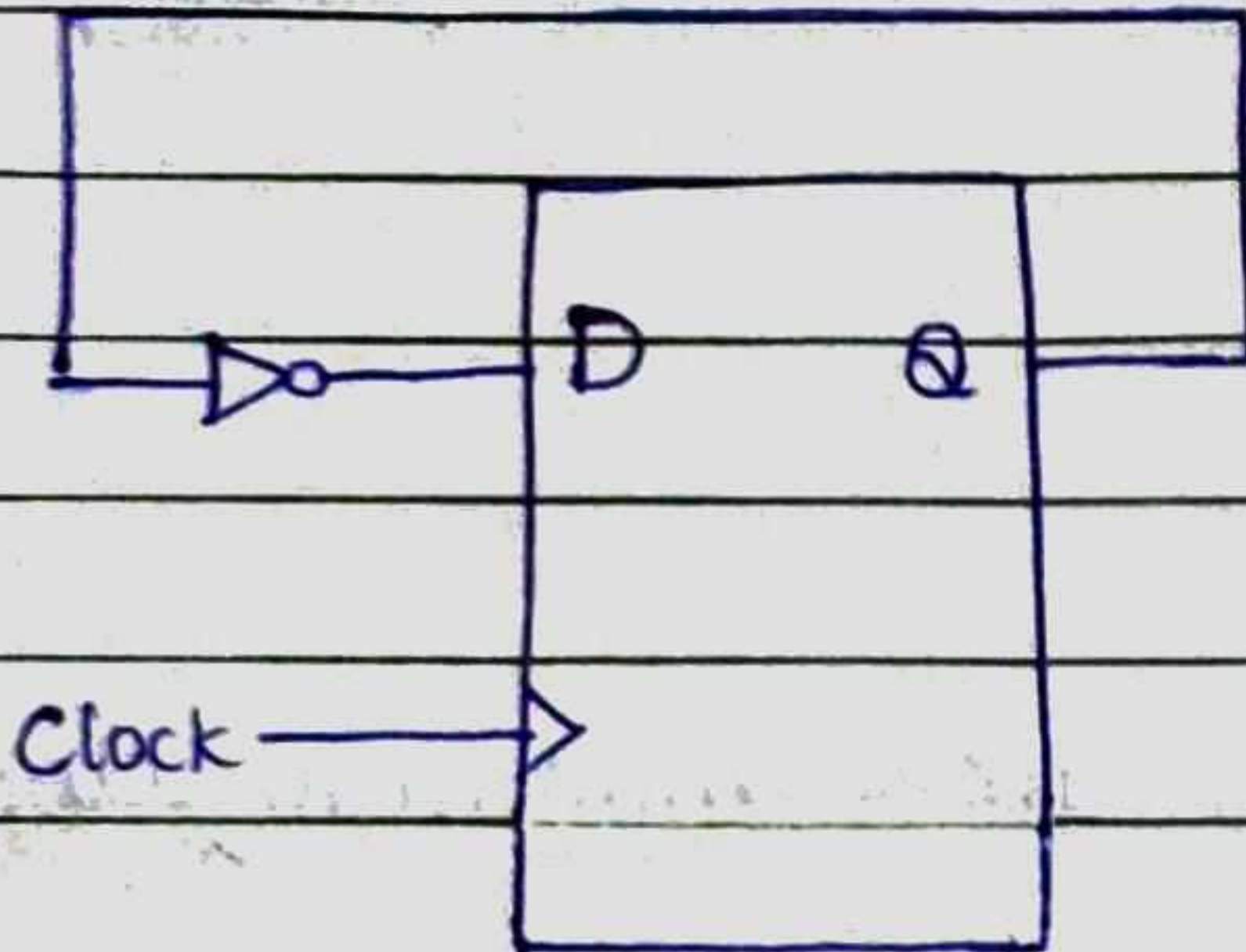
Timing diagram.



(D should not change during the trigger of the clock pulse)

Determination of Minimum clock period

a Simple flip-flop circuit:



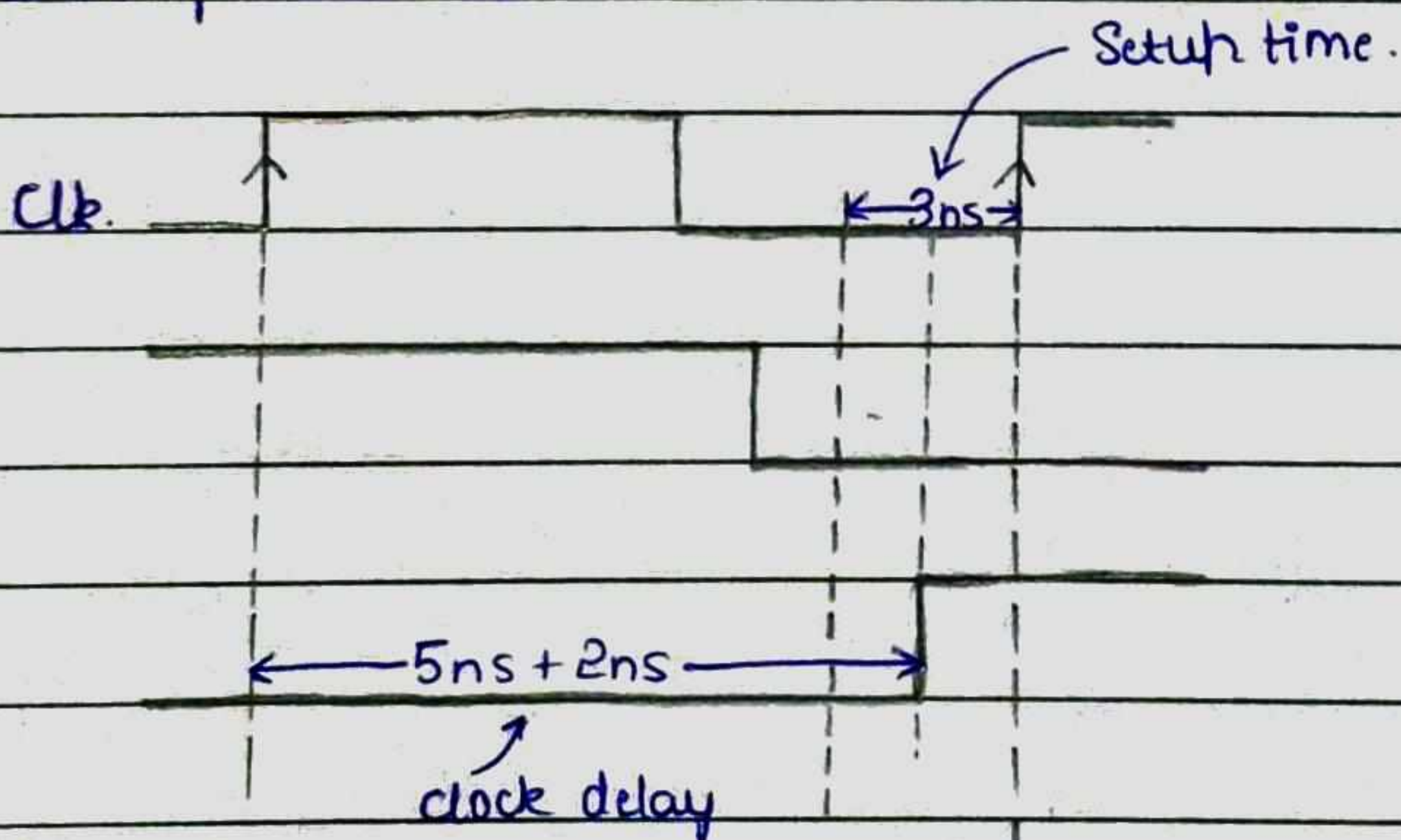
Flip flop delay = 5ns

not gate delay = 2ns

Setup time = 3ns

b Setup time is not satisfied.

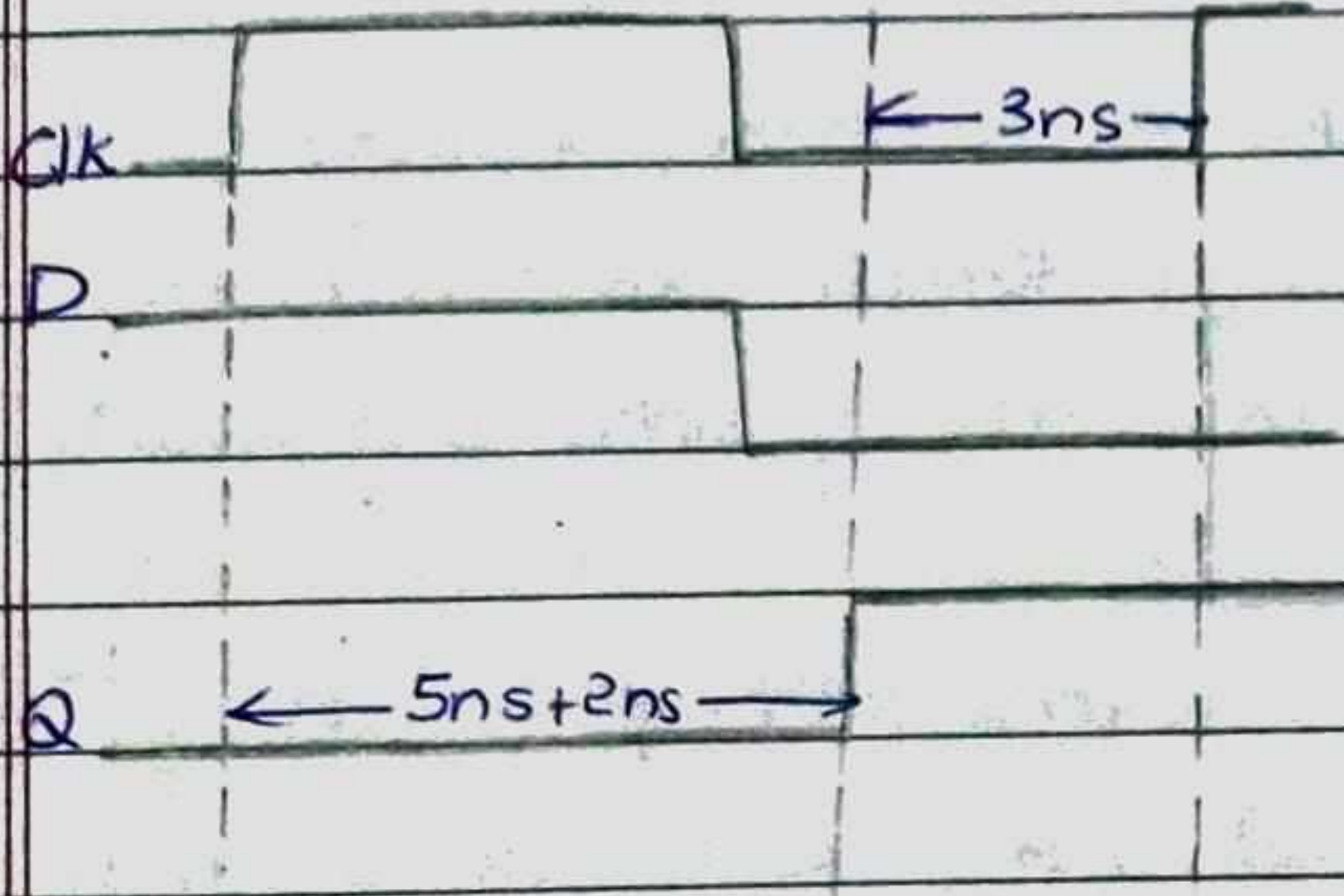
i clock period is 9ns



Setup time is not satisfied.

ii) Clock period is 10 ns

[minimum clock period
= circuit delay +
setup time]

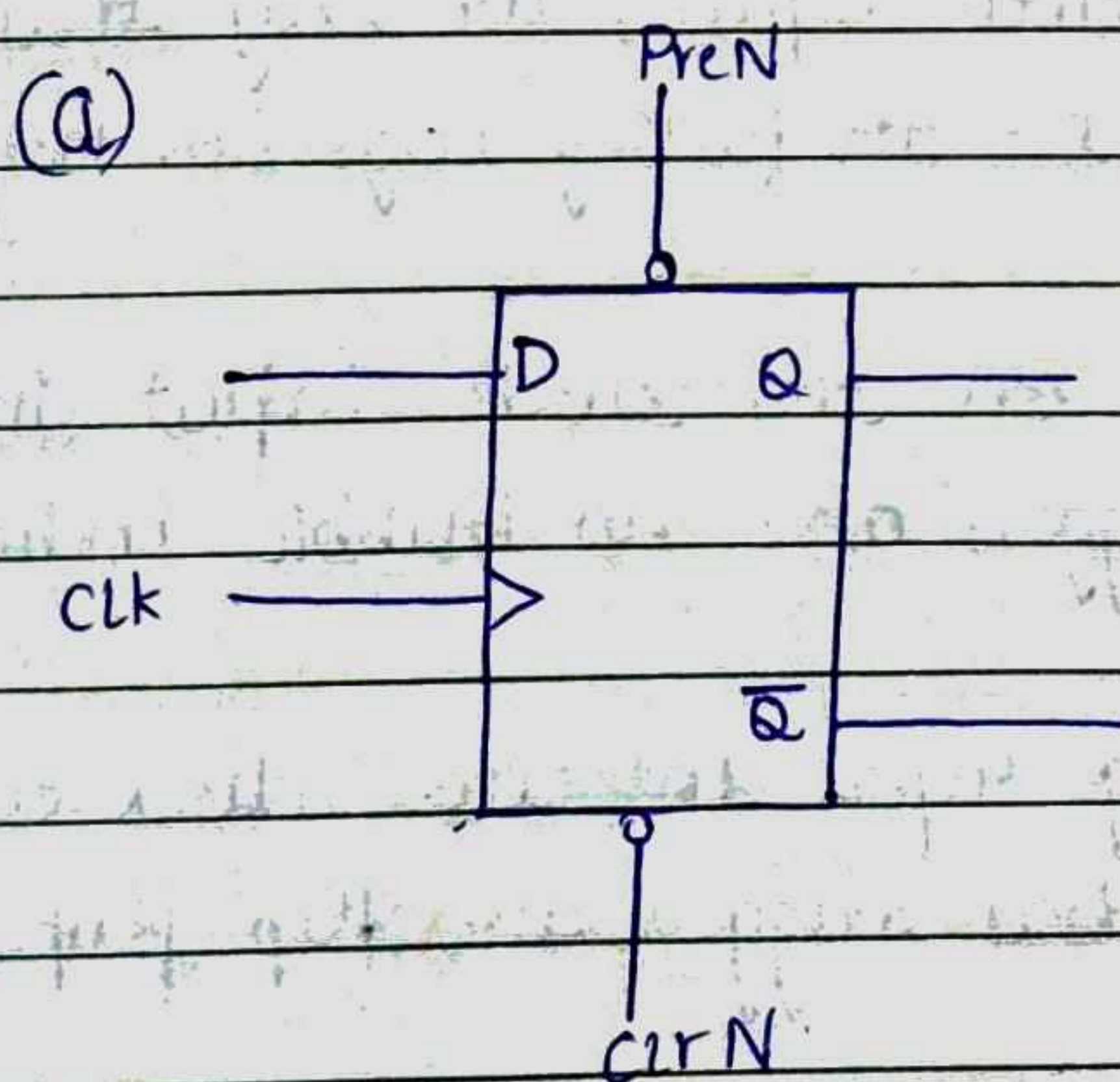


Setup time is satisfied with minimum clock period.

⇒ Flip-Flops with Additional Inputs Preset and Clear

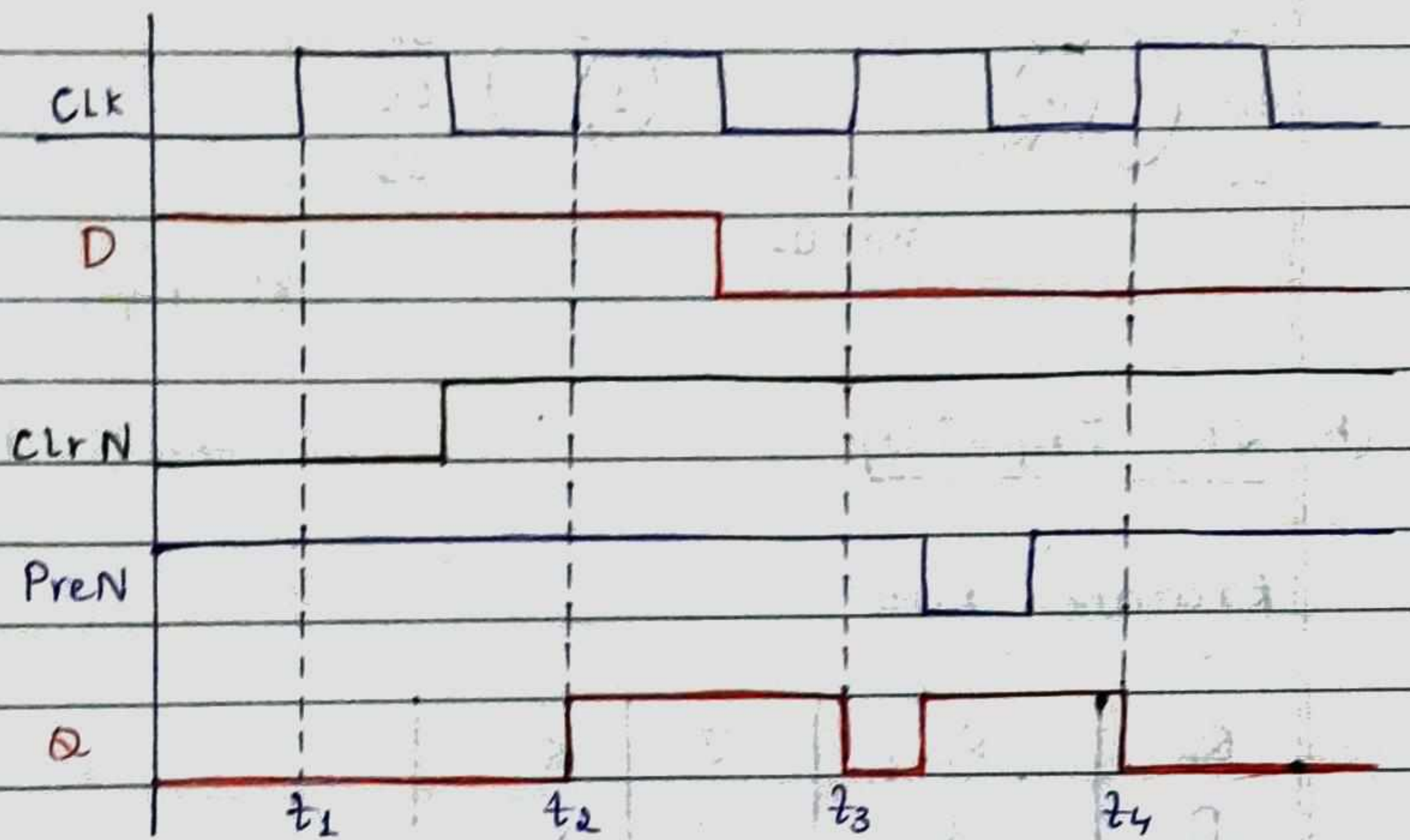
- Flip-flops often have additional inputs which can be used to set the flip-flop to an initial state independent of the clock.
- Figure shows a D flip-flop with clear and preset inputs. The small circles (inversion symbols) on these inputs indicate that a logic 0 (rather than a 1) is required to clear or set the flip-flop.
- We will use the notation ClrN or PreN to indicate active-low clear and preset inputs. Thus, a logic 0 applied to ClrN will reset the flip-flop to $Q=0$, and a 0 applied to PreN will set the flip-flop to $Q=1$. These inputs override the clock and D inputs.
- ClrN and PreN are often referred to as asynchronous clear and preset inputs because their operation does not depend on the clock.
- The table summarizes the flip-flop operation.

D Flip-Flop with Clear and Preset.



(b)	ck	D	PreN	ClrN	Q^+
	X	X	0	0	(not allowed)
	X	X	0	1	1
	X	X	1	0	0
	↑	0	1	1	0
	↑	1	1	1	1
	↓	X	1	1	Q (no change)

Timing Diagram for D Flip-Flop with Asynchronous Clear and Preset



⇒ Flip-Flops as Finite State Machines

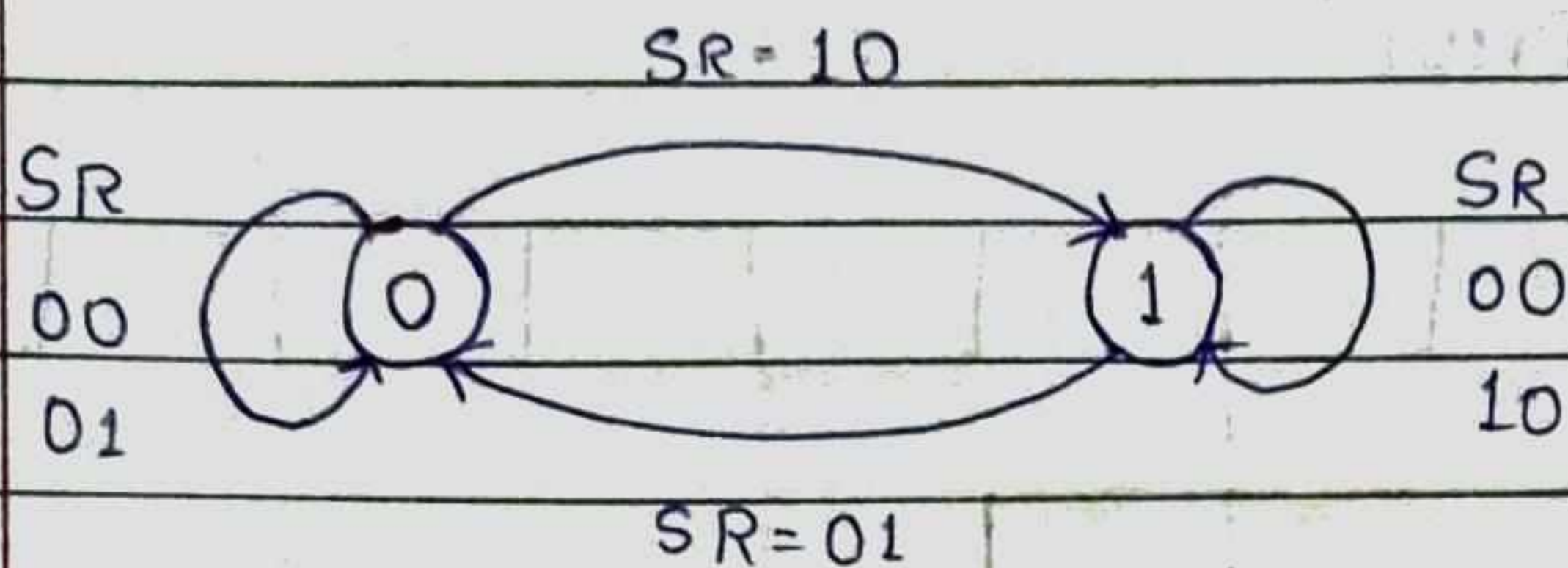
State diagrams

- In a sequential logic circuit Finite State Machine (FSM) the value of all the memory elements at a given time define the state of that circuit at that time.
- In FSM, the functional behaviour of the circuit is represented using the state transition diagram.

(a) SR Flip-Flop

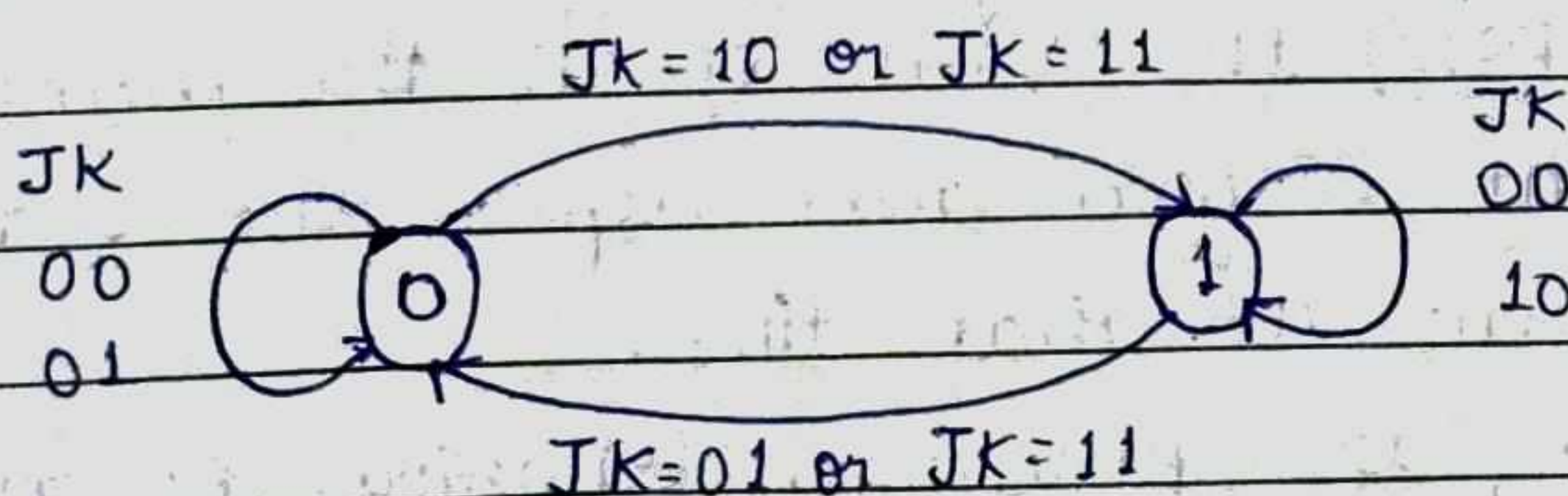
Excitation table

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

(b) JK Flip-Flop

Excitation table

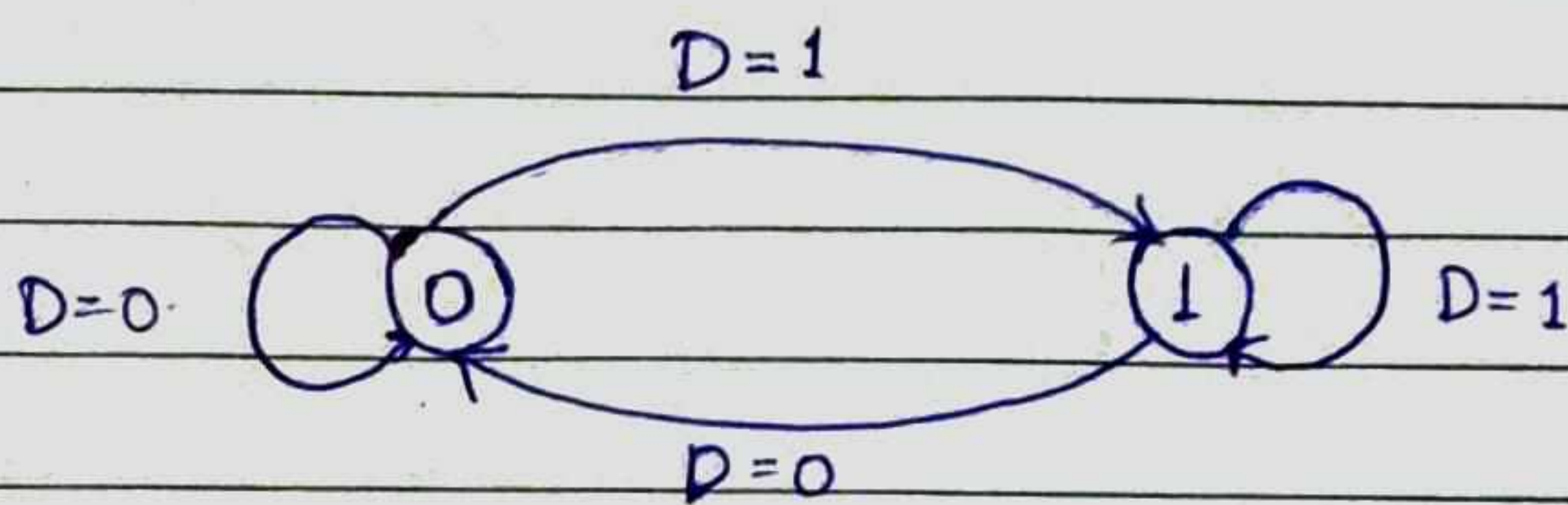
Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0



(c) D Flip-Flop

Excitation Table

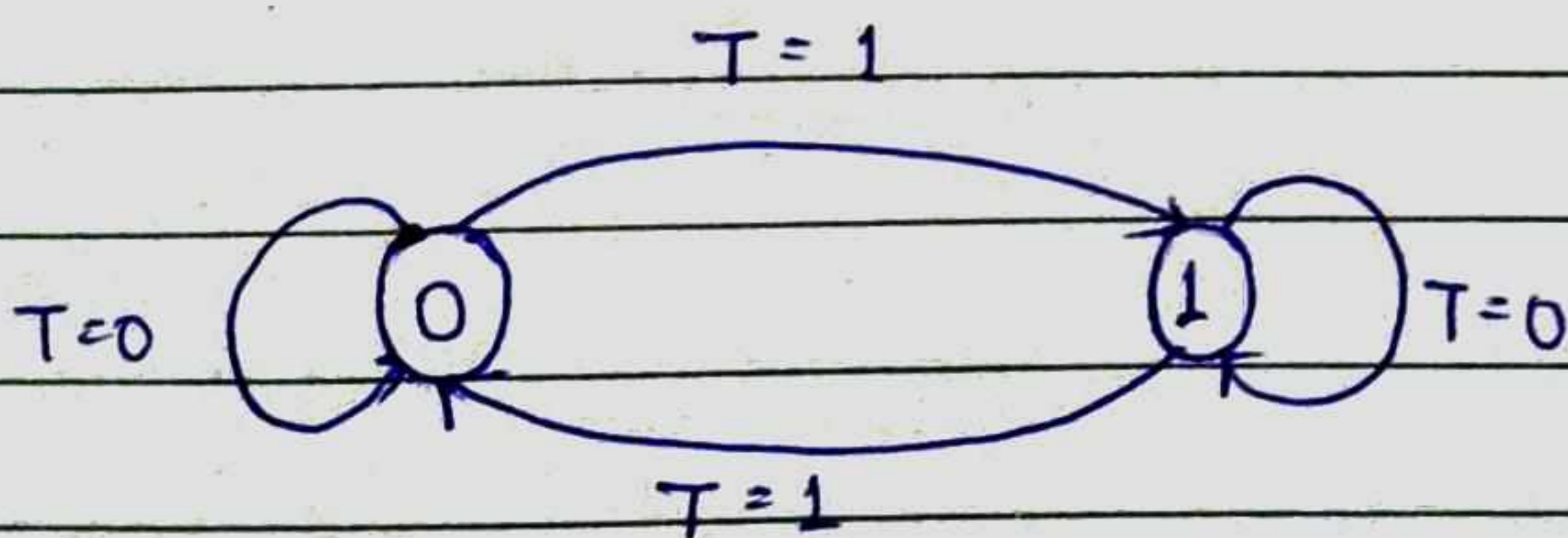
Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1



(d) T Flip-Flop

Excitation Table

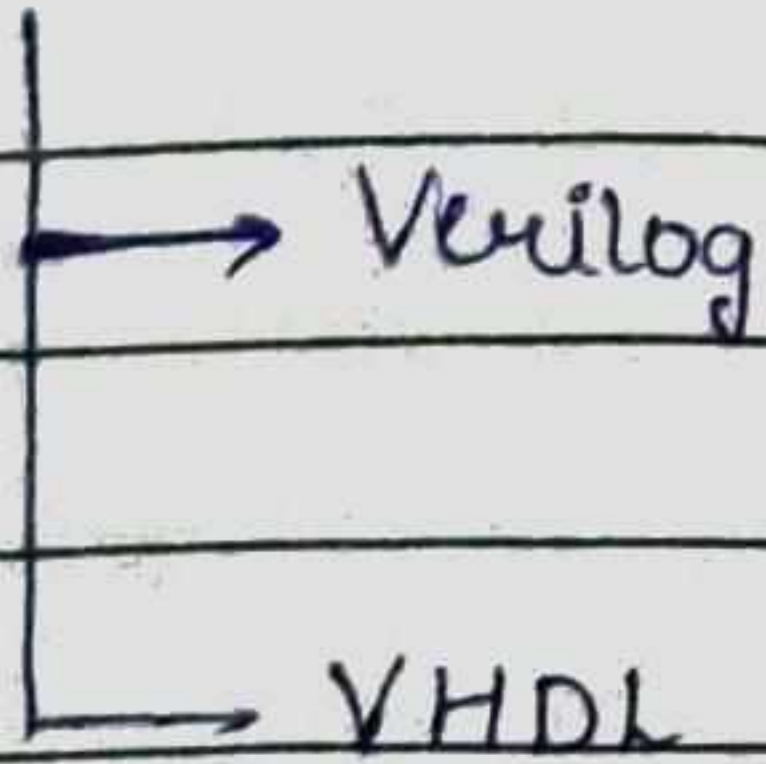
Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0



Module - 4

VHDL

⇒ HDL: Hardware Description Language



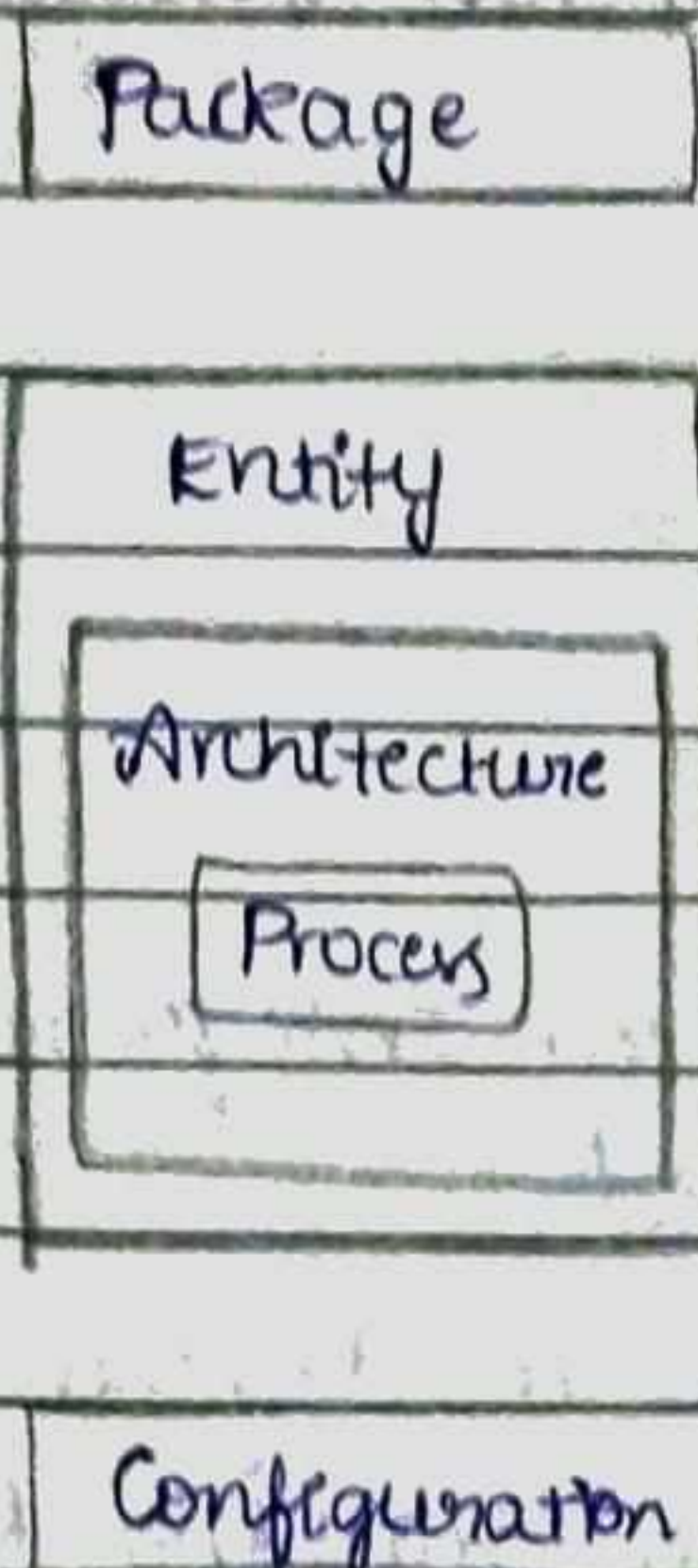
⇒ VHDL [Very High Speed Integrated Circuits Hardware Description Language]

It is used to model a digital system at many levels of abstraction ranging from the algorithmic level to the gate level.

→ Features of VHDL

1. In VHDL, ^{the} simulation of logic operation & timing behaviour of a design is possible.
2. It is a case insensitive language.
3. It has powerful constructs, here the designs may be decomposed hierarchically.
4. Each design element has a well defined interface useful for connecting it to other elements.
5. Each design element has a precise behavioural specification useful for simulating it.
6. It handles asynchronous as well as synchronous sequential circuits.
7. VHDL supports design library.

⇒ Structure of VHDL model [Relationship of VHDL design units]



The components of VHDL:

1. Package
2. Entity
3. Architecture
4. Configuration

- Entity & architecture blocks are compulsorily required.
- Package & Configuration blocks are optional.

→ Package

1. There are some declarations which are common across many design units. A Package is a convenient mechanism to store & share such declarations.
2. It is an optional design unit.
3. It defines items that can be made visible to other design units.
4. A package is represented by

- package declaration.
- package body.

→ Package declaration

It defines the interface to the package.

Syntax:

```
PACKAGE package-name IS
```

```
  type declarations
```

```
  constant "
```

```
  signal "
```

```
  variable "
```

```
  site "
```

```
END package-name;
```

The items declared in the package declaration can be accessed by other design units by using 'library' and 'use' clauses.

→ Package body

It contains the details of a package, i.e., the behaviour of the subprograms & the values of the different constants which are declared in a package declaration.

Syntax:

```
package body package-name is
```

```
  subprogram bodies
```

```
  complete constant declarations
```

```
end package-name;
```

Name of the package must be same as the name of its corresponding package declaration.

⇒ Entity

- It gives the specification of input/output signals to external circuitry. An entity is modelled using an entity declaration and at least one architecture body.
- Entity gives interfacing between device and the other peripherals.
- All information must flow into and out of the entity through the ports. Each port must contain name, data flow direction and type.

Syntax:

entity entity-name **is**

port (
 signal-names: mode signal-type;
 signal-names: mode signal-type;
 :
 :

 signal-name_n: mode signal-type);

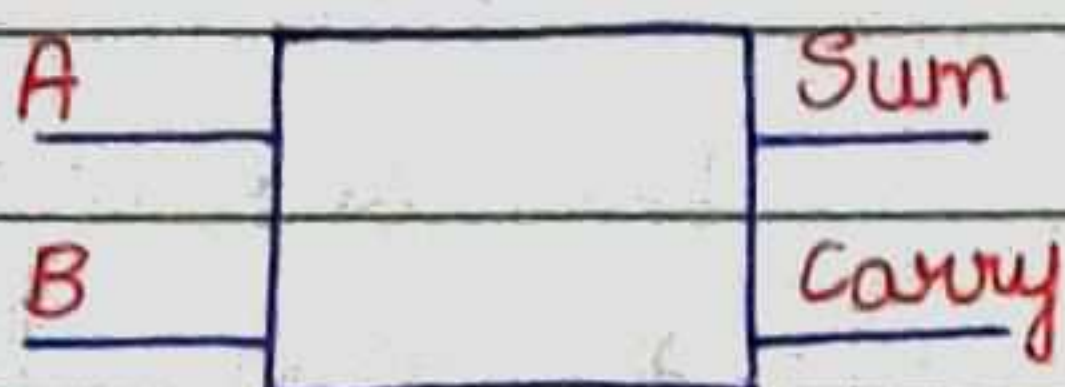
end entity-name;

→ The following section describes the different elements of entity declaration:

- 1 entity-name: It is an identifier selected by the user to name the entity.
- 2 signal-names: It is a list of user selected identifiers to name external interface signals.
- 3 mode: The ports can be declared in four types which specify the signal direction.
 - i in: This mode is used for a signal that is an input to an entity (value is read not written).
 - ii out: It is used for a signal that is an output from an entity. The value of such a signal can not

- be read inside the entity's architecture. But it can be read by other entities that use it.
- iii inout: It is used for a signal that is both, an input to an entity and an output from the entity.
 - iv buffer: The signal is an output from the entity and its value can also be read inside the entity's architecture.
 - v signal-type: It is a built-in or user defined signal type.

Example: Half Adder



Std logic
?

Entity

you!

entity Half-Adder is

port (A, B : in BIT;

SUM, CARRY : out BIT);

end Half-Adder

0,1

⇒ Architecture

- Architecture specifies behaviour, functionality, interconnection or relationship between inputs and outputs.
- It is the actual description of the design.
- An architecture consists of two portions: architecture declaration and architecture body.
- An architecture body specifies the internal details of an entity.
- As a set of concurrent assignment statements (to represent dataflow)

- As a set of interconnected components (to represent structure)
- As a set of sequential assignment statement (to represent behaviour)
- As any combination of above three.

Syntax:

architecture architecture-name of entity-name is

Declarations

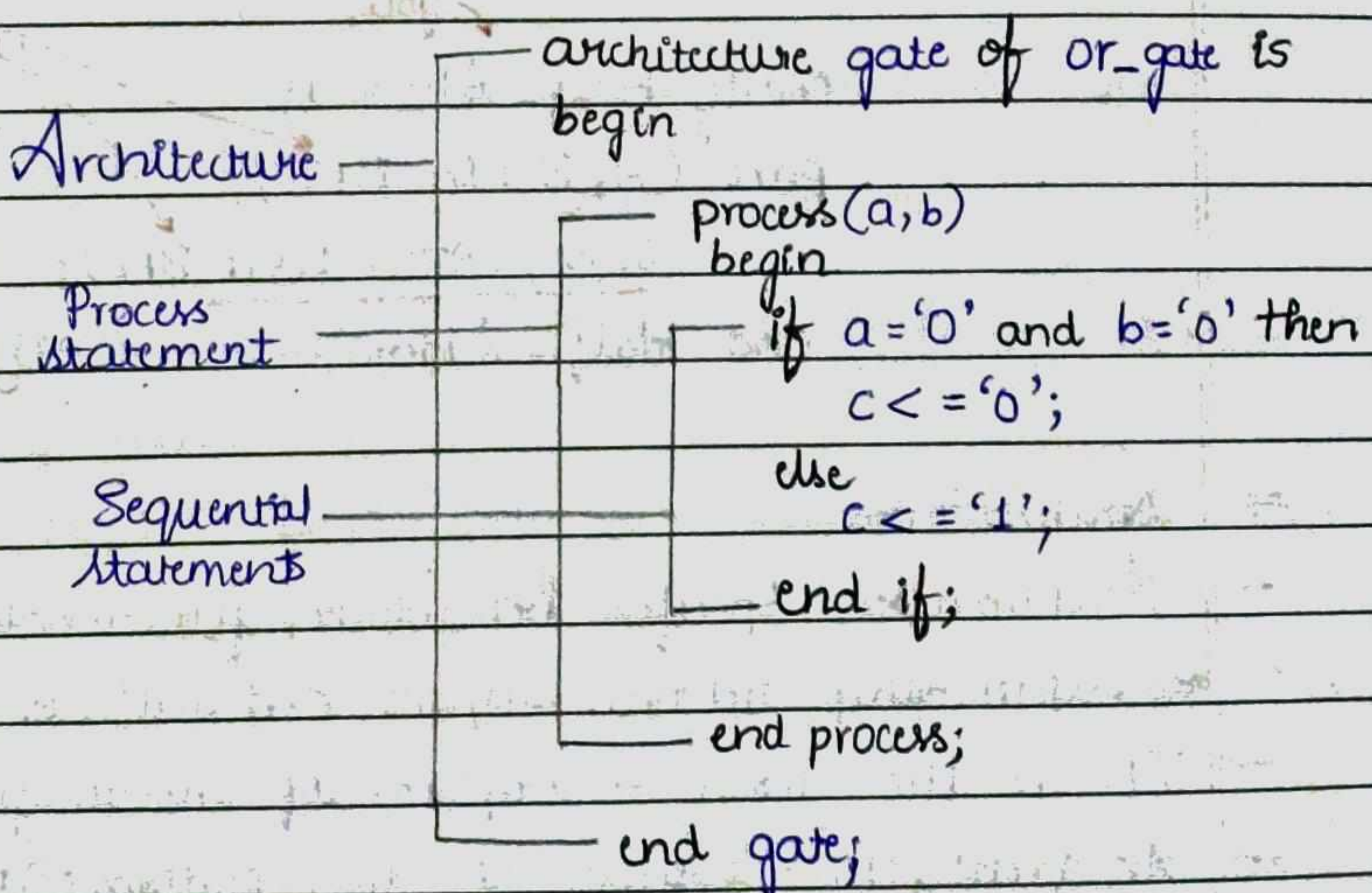
begin

concurrent statements;

sequential statements;

end architecture-name;

Example:



⇒ Configuration

- To associate a particular architecture to an entity
- As their name implies, configuration declarations are used to provide configuration management and project organization for a large design.
- Important points to remember while representing any module using VHDL.
 1. Each statement in VHDL is terminated with a semicolon (;).
 2. The language is case insensitive i.e., the uppercase and lowercase letters are considered as same.
 3. The name should start with an alphabet letter and can include the special character underscore (_).
 4. The name of the ports must be followed by a colon (:).
 5. The architecture body starts with the predefined word begin, followed by statements that detail the relationship between the outputs and inputs.
 6. The comment should begin with two hyphens (--).
 7. Leaving the blank spaces between two words or at the beginning of the line are allowed.
 8. Leaving the blank line(s) is allowed in the module.

⇒ ⇒ The statements in VHDL architecture

1 Concurrent statements (executes in parallel)

- These are used outside a process, but within an architecture.
- It is a short hand way to write a process.
- It is equivalent to a process containing one statement.

Ex: $C \leq A \text{ and } B;$

$E \leq C \text{ or } D;$

2 Sequential statements

These are used in a process & describes the behaviour of an architecture.

Ex: if $a = '0'$ and $b = '0'$ then

$C \leq '0';$

else

$C \leq '1';$

end if;

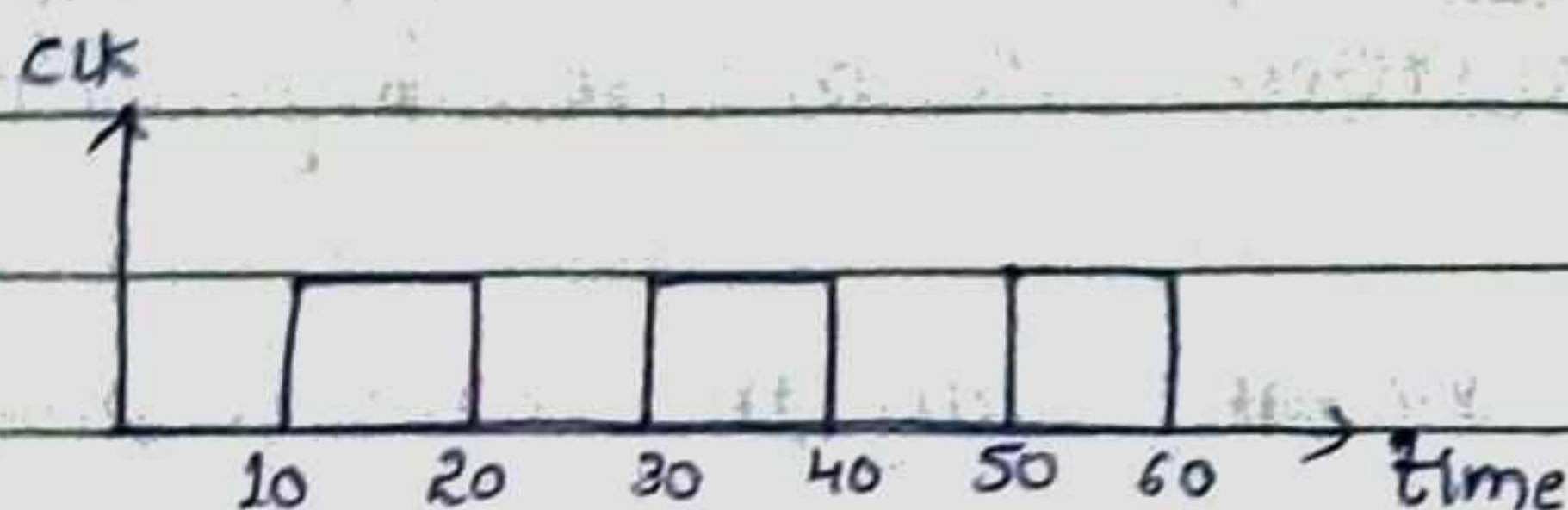
⇒ ⇒ Signal assignment statement [Normal Instructions]

Syntax: signal_name \leq expression [after delay];

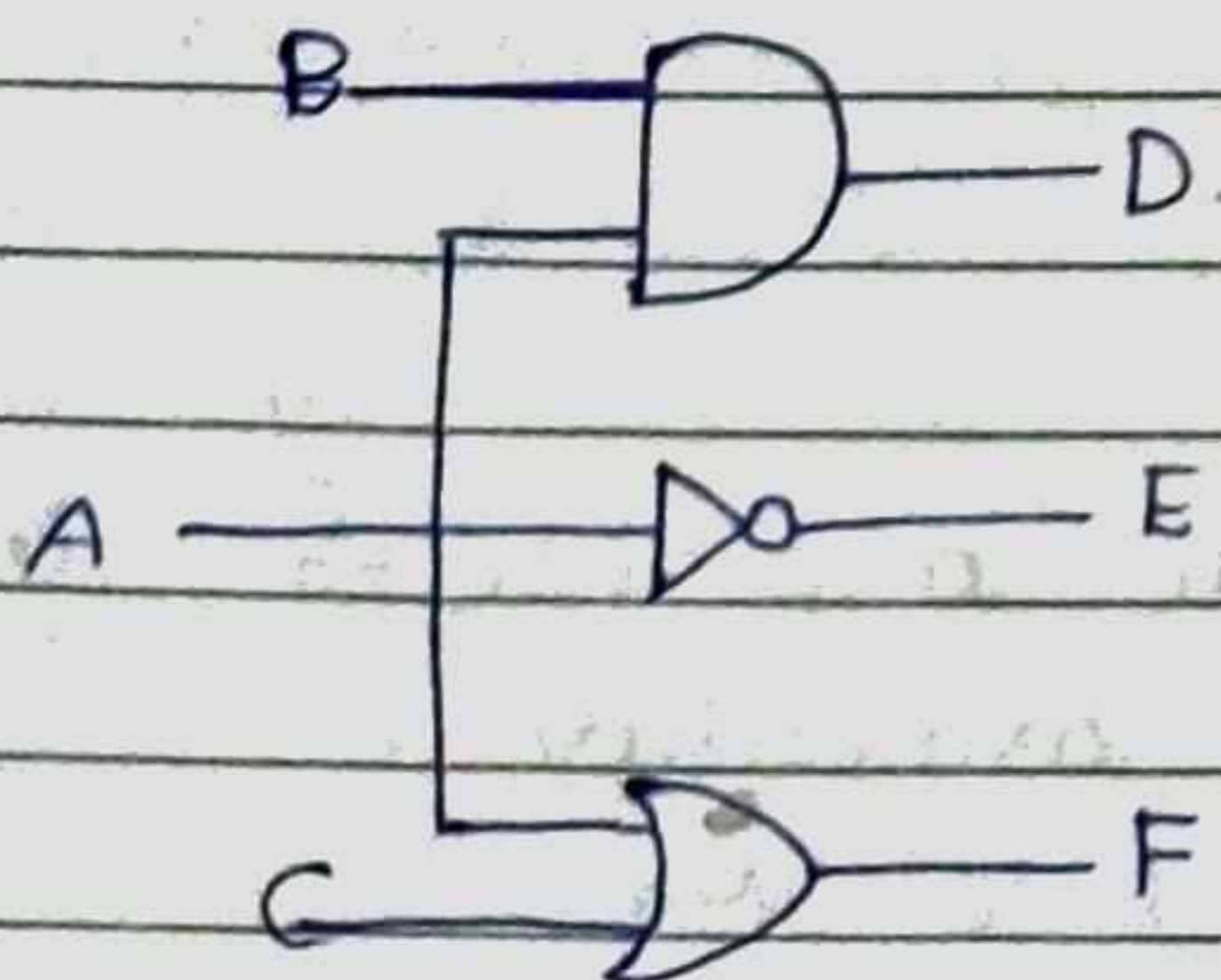
- The expression is evaluated when the statement is executed & the signal on the left side is scheduled to change after delay.
- The square brackets that indicates, after delay is an optional - they are not part of the statement.
- If after delay is omitted, then the signal is scheduled to be updated after a delta delay.
- Delta delay: It is a default signal assignment propagation delay if no delay is explicitly prescribed.

Ex:

1) $clk \leq \text{not } clk \text{ after } 10 \text{ ns};$



2)



$D \leq A \text{ and } B \text{ after } 2 \text{ ns};$

$E \leq \text{not } A \text{ after } 1 \text{ ns};$

$F \leq A \text{ or } C \text{ after } 3 \text{ ns};$

⇒ Conditional signal assignment statement

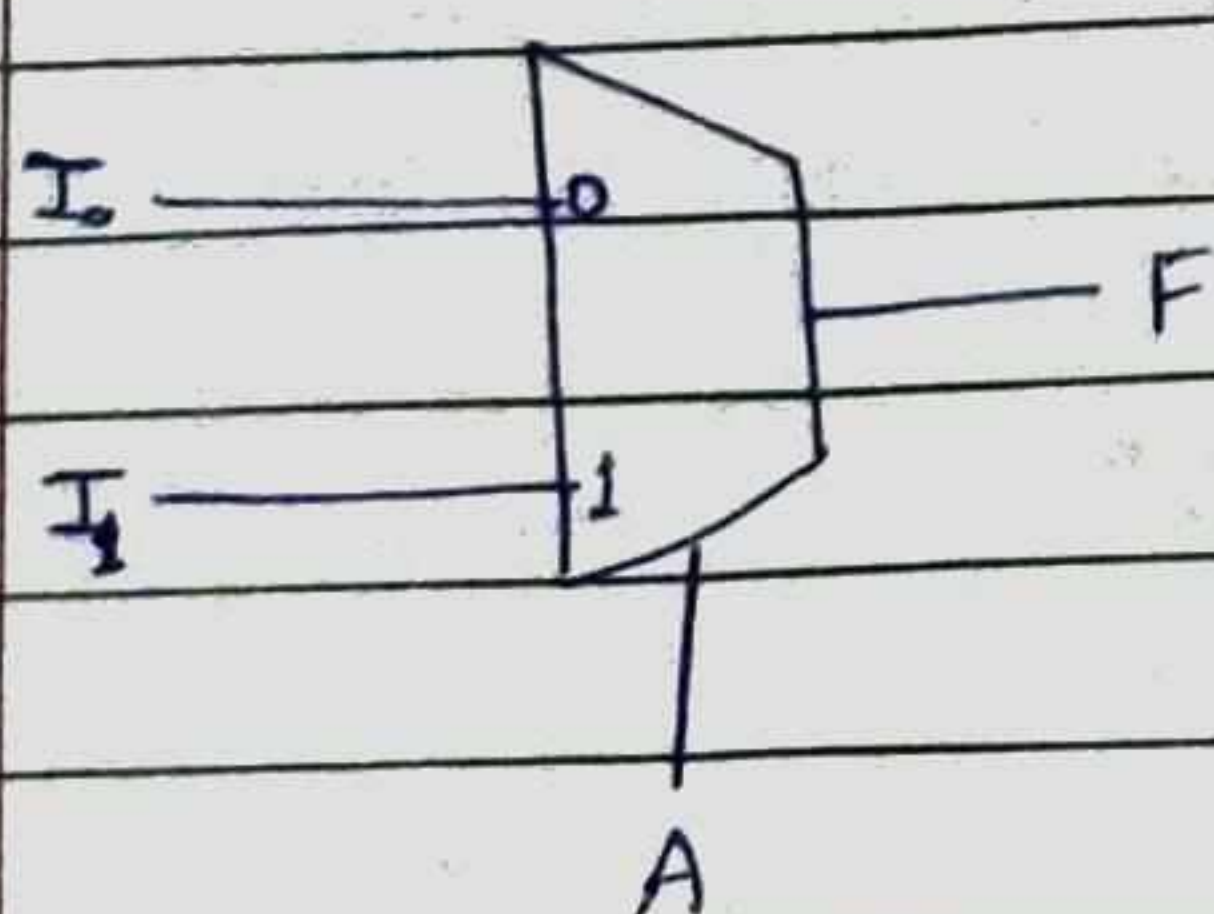
Syntax:

```

signal_name <= expression 1 when condition 1
                else expression 2 when condition 2
                :
                :
                [else expression N];
  
```

If condition 1 is true, signal_name is ^{set} equal to the value of expression 1, or else if condition 2 is true, signal_name is set equal to the value of expression 2 & so on. & the last line in the square bracket is optional.

Ex: 1) 2 to 1 MUX.



$$Y = A' I_0 + A I_1$$

Normal VHDL statement:

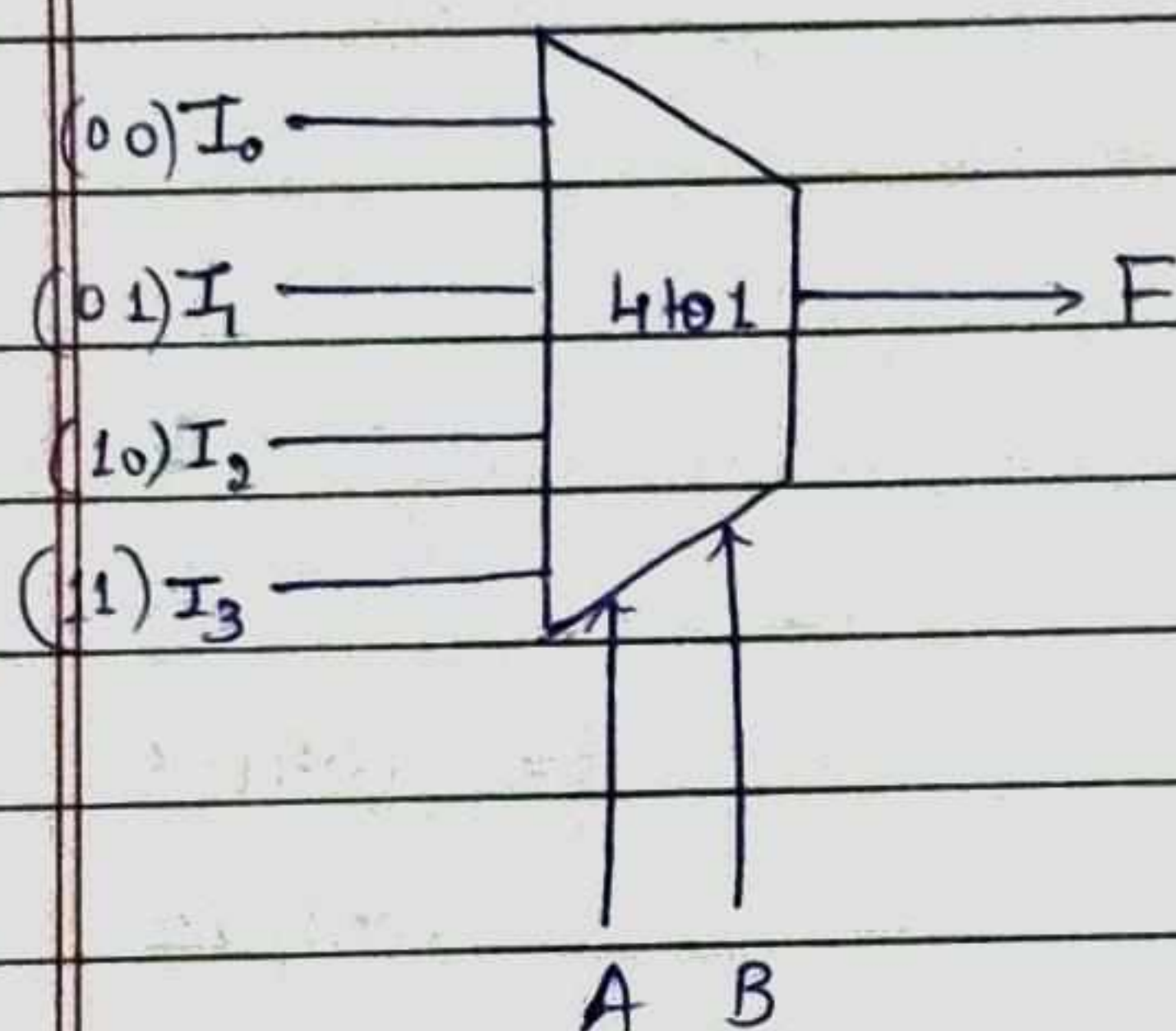
$$F <= (\text{not } A \text{ and } I_0) \text{ or } (A \text{ and } I_1);$$

Conditional signal assignment statement -

$$F <= I_0 \text{ when } A = '0'$$

$$\text{else } I_1;$$

2) 4 to 1 MUX



$$Y = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3$$

Normal VHDL statement -

$$F <= (\text{not } A \text{ and not } B \text{ and } I_0) \text{ or } (\text{not } A \text{ and } B \text{ and } I_1) \text{ or } (A \text{ and not } B \text{ and } I_2) \text{ or } (A \text{ and } B \text{ and } I_3);$$

Conditional signal assignment statement -

$$F <= I_0 \text{ when } A = '0' \text{ and } B = '0'$$

$$\text{else } I_1 \text{ when } A = '0' \text{ and } B = '1'$$

$$\text{else } I_2 \text{ when } A = '1' \text{ and } B = '0'$$

$$\text{else } I_3;$$

⇒ Note:

$F <= I_0$ when $A = '0'$ and $B = '0'$

(or)

$F <= I_0$ when $A \& B = "00"$

⇒ Selected signal assignment

Syntax:

With expression_s select

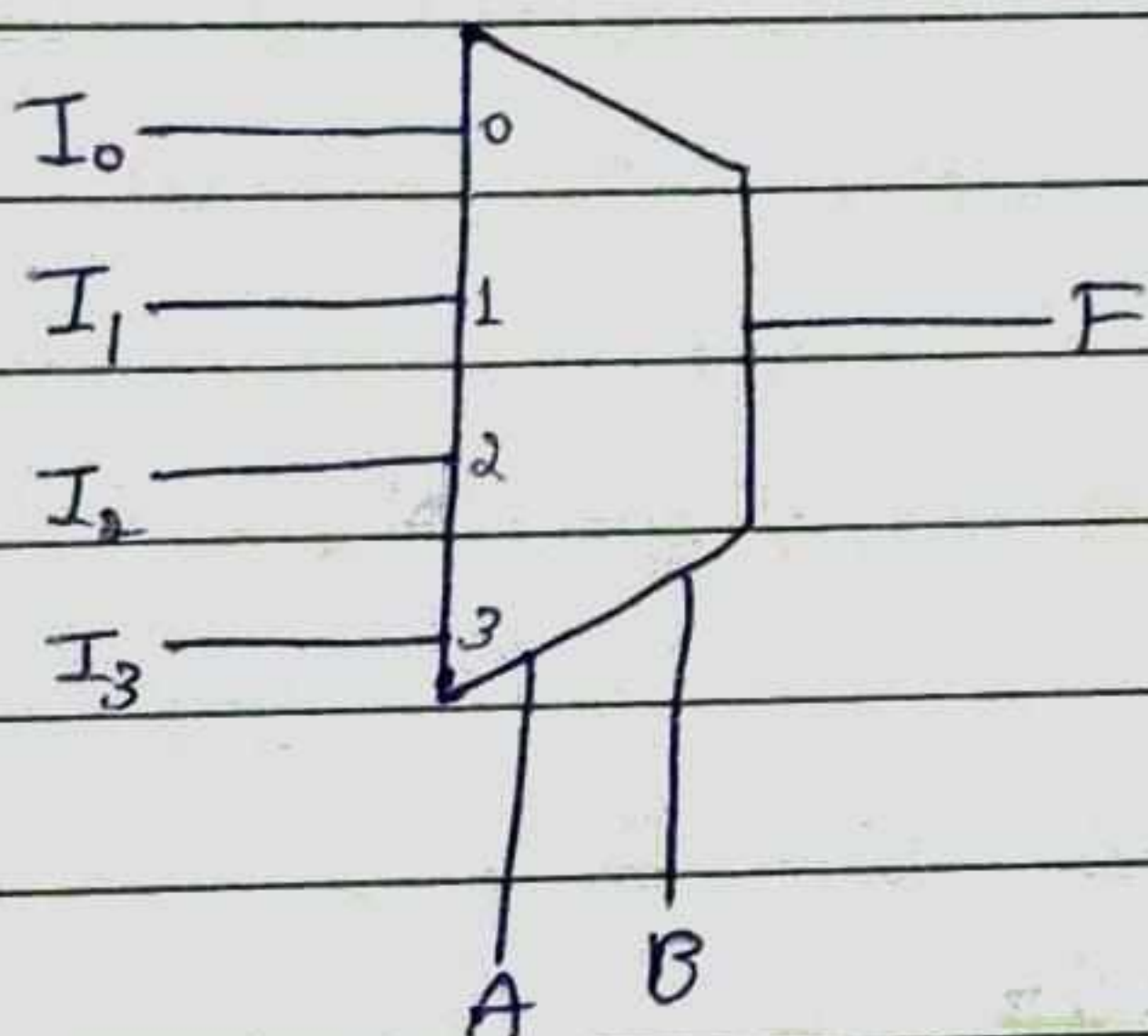
Signal_name $<=$ expression 1 when choice 1;

expression 2 when choice 2;

⋮

[expression N when others];

Example ex: 4 to 1 MUX.



$sel <= A \& B;$

with sel select

$F <= I_0$ when "00",

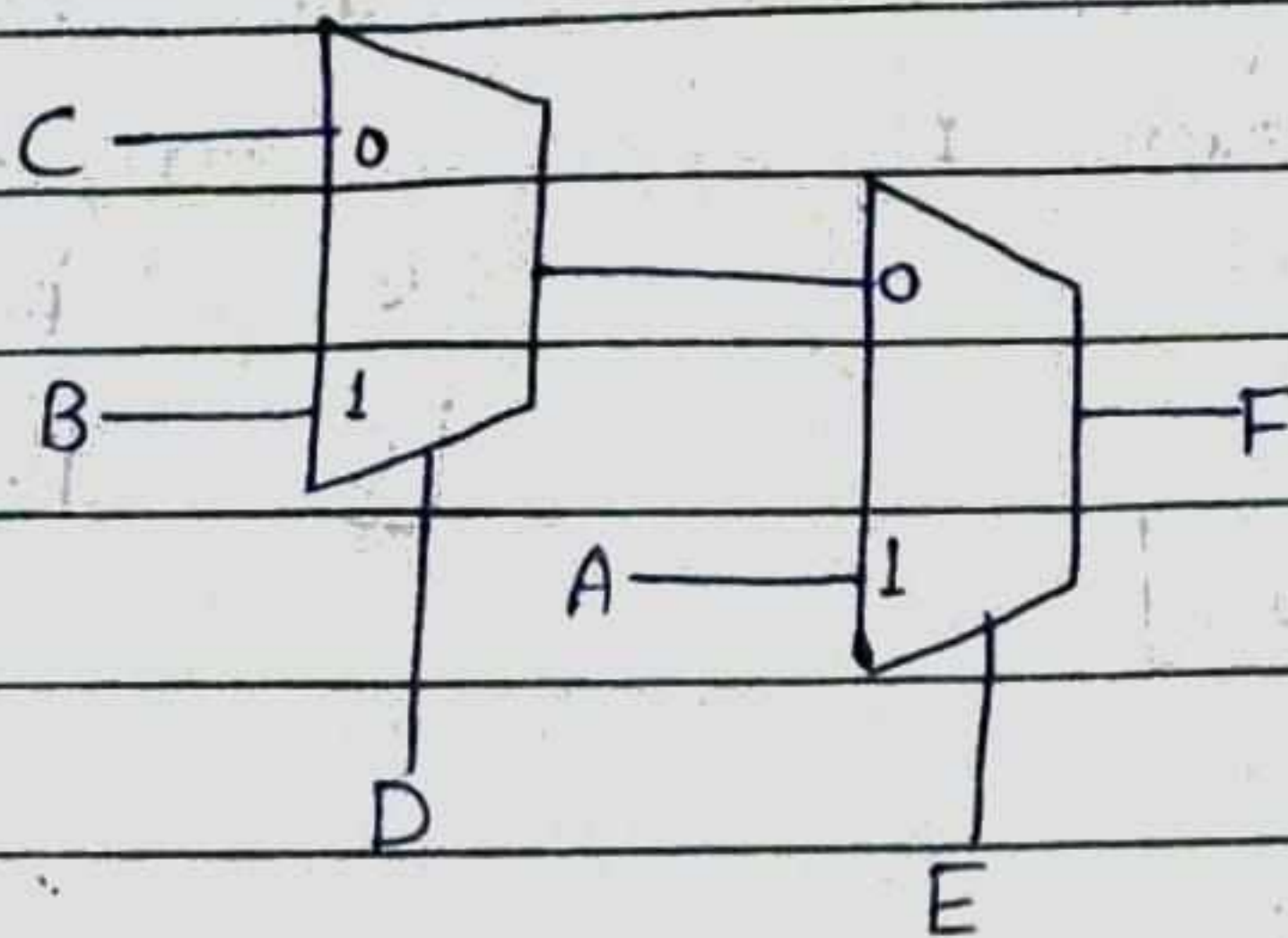
I_1 when "01",

I_2 when "10",

I_3 when "11";

ex: Write a VHDL code using conditional signal assignment.

⇒ Cascaded 2 to 1 MUX.



data i/p: A, B, C
Sel i/p: D, E
o/p: F

$F \leq A$ when $E = '1'$
else B when $D = '1'$
else C ;

⇒ Signals.

→ Signals internal to a module are declared at the start of an architecture before begin & can be used only within that architecture.

Note: A signal used within an architecture must be declared either in a port or in the declaration section of an architecture, but it cannot be declared in both the places.

Syntax:

signal list-of-signal-names: type-name [constraint] [:= initial-value];

ex:

1) signal A, B, C: bit-vector (3 downto 0) := "1111";

A, B & C are 4 bit vectors dimensioned 3 downto 0

& initialized to 1111.

A

3	2	1	0
1	1	1	1

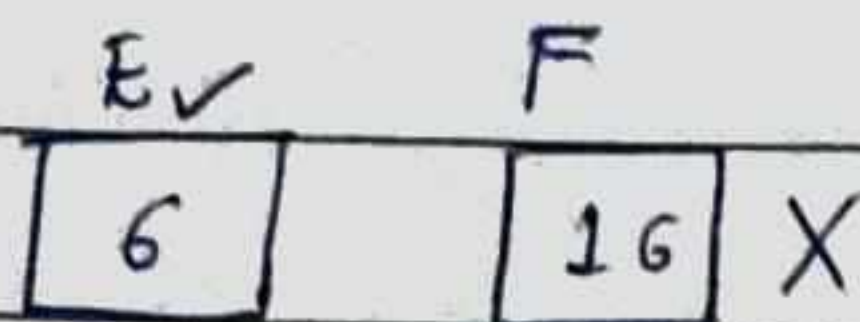
B

1	1	1	1
---	---	---	---

C

1	1	1	1
---	---	---	---

2) signal E, F: integer range 0 to 15;
 E & F are integers in the range 0 to 15,
 initialized by default value 0 (The compiler or
 simulator will flag an error if we attempt to
 assign a value outside the specified range).



⇒ Constant.

These are declared at the start of an architecture
 & can be used anywhere within that architecture.

Syntax:

constant constant_name: type_name [constraint] [:= constant_value];

ex:

1) constant a: integer := 17;

2) constant delay1: time := 5ns;

& it can be used in signal assignment statement
 A <= B after delay1;

VI
HM
VHDL data types.

Datatype	Description
bit	'0' or '1'
boolean	FALSE or TRUE
integer	an integer in the range $-(2^{31}-1)$ to $+(2^{31}-1)$ [some implementation support wider range]

positive	an integer in the range 1 to $2^{31} - 1$
natural	an integer in the range 0 to $2^{31} - 1$
real	floating point number in the range -1.0E38 to +1.0E38
character	any legal VHDL character including upper and lower case characters, digits & special characters; each printable character must be enclosed in single quotes. Eg: 'd', '7'
time	an integer with units fs, ps, ns, us , ms, sec, min, hr.

⇒ Arrays

In order to use an array with VHDL, we must first declare an array type & then ^{declare an} array object

Syntax:

array type ⇒ type array-type-name is array index-range of element-type;

array object ⇒ signal array-name: array-type-name[initial-value];

Example: array type ⇒

type SHORT-WORD is array (15 downto 0) of bit;

It has an integer index with a range from 15 down to 0 & each element of the array of type bit.

array object ⇒

1) signal DATA-WORD: SHORT-WORD;

(DATA-WORD)

It is a signal array of 16 bits & it is initialized to zeros.

2) Signal `ALT-WORD : SHORT-WORD := "0101010101010101";`
`ALT-WORD` is a signal-array of 16 bits which is initialized to alternate zeros & ones

3) Constant `ONE-WORD : SHORT-WORD := (others => '1');`
`ONE-WORD` is constant array of 16 bits, all bits are set to one - because none of the bits have been set individually, in this case `others` applies to all of the bits.

⇒ We can reference individual elements of the array by specifying an index value.

Ex: `ALT-WORD(0)`: Accesses the first bit.

→ We can also specify a portion of the array by specifying an index range.

`ALT-WORD(5 downto 0)` accesses the lower order 6 bits of `ALT-WORD`.

→ Multidimensional array

type `matrix 4x3` is array (1 to 4, 1 to 3) of integer;

signal `matrix A : matrix 4x3 := ((1, 2, 3), (4, 5, 6), (7, 8, 9), (10, 11, 12));`

The signal `matrix A`, will be initialized to

1	2	3
4	5	6
7	8	9
10	11	12

VHDL operators

Pre defined VHDL operators can be grouped into 7 classes:

- 1 Binary Logical operators
and, or, nand, nor, xor, xnor.
- 2 Relational operators
=, !=, <, <=, >, >=
- 3 Shift operator
sll, srl, sla, sra, rol, ror
- 4 Concatenation operators
+, -, &
- 5 Unary sign operators.
+, -
- 6 Multiplying operators.
*, /, mod, rem
- 7 Miscellaneous operators.
not, abs, ^{power} **

⇒ Operator precedence

- Operators in class 7 have highest precedence & are applied first, followed by class 6 then class 5 & so on
- The class 1 have the lowest priority.
- Operators in the same class have the same precedence & are applied from left to right in an expression - left to right (L → R) associative.
- Always parenthesis has the top most priority

2) Relational operators

- They are used to compare 2 expressions & return a value of FALSE or TRUE.
- The 2 expressions must be of same type & size.
- Equal ⁽⁼⁾ & Not equal ^(!=) can be applied to any type.
 ex: if $A=5, B=4$ and $C=3$.
 $(A > B)$ and $(B <= C)$
 TRUE and FALSE
 FALSE

Note

'=' is always relational operator but '<=' also serves as an assignment operator.

3) Shift operators

These are used to shift or rotate a bit vector.

Ex: Consider a eight bit vector A equal to "10010101"

i) sll - Shift left logical filled with zero.

A sll 2

01010100

ii) srl - Shift right logical filled with zero.

A srl 2

00100101

iii) sla - Shift left arithmetic filled with right most bit.

A sla 3

10101111

iv) sra - Shift right arithmetic filled with left most bit.

A sra 3

11110010

v) rol - rotate left

A rol 3

10101100

10010101100

vi) ror - rotate right

A ror 5

10101100

10010101
10101100

4) Concatenation or adding operators

ex:

A = 10010101

A(7) & A(6) & A(7 down to 2)

1 & 0 & 100101

10100101

The '+' & '-' operators can be applied to Integer or real numeric operands.

The '&' operator can be used to concatenate two vectors.

→ IEEE Standard logic

The IEEE Standard 1164 defines a std_logic type that has 9 values

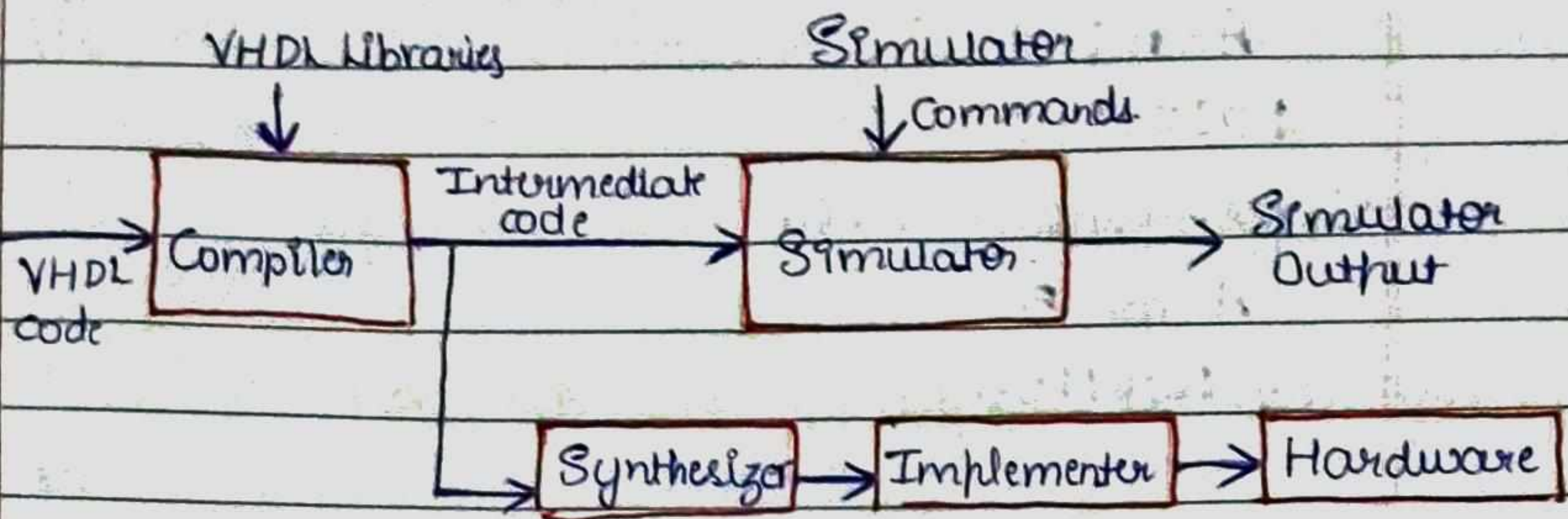
('U', 'X', '0', '1', 'Z', 'W', 'L', 'H' and '-')

We are concerned with only first 5 values.

1. U : U stands for uninitialized.

When a logic circuit is first turned ON & before it is reset, the signals will be uninitialized.

Compilation & Simulation of VHDL Code



- After describing a digital system in VHDL, simulation of the VHDL code is important for two reasons. First, we need to verify the VHDL code correctly implements the intended design, and second, we need to verify that the design meets its specifications.
- The VHDL compiler, also called an analyzer, first checks the VHDL source code to see that it conforms to the syntax and semantic rules of VHDL.
- The compiler also checks to see that references to libraries are correct.
- If the VHDL code conforms to all of the rules, the compiler generates intermediate code which can be used by a simulator or by a synthesizer.
- The VHDL intermediate code must be converted to a form which can be used by the simulator. This step is referred to as elaboration.
- During elaboration, ports are created for each instance of a component, memory storage is allocated for the required signals, the interconnections among the port signals are specified, and a mechanism is established for executing the VHDL statements in the proper sequence.
- After an initialization phase, the simulator enters the execution phase.
- The simulator accepts simulation commands which

control the simulation of the digital system and specify the desired simulator output.

- After the VHDL code for a digital system has been simulated to verify that it works correctly, the VHDL code can be synthesized to produce a list of required components and their interconnections.
- The synthesizer output can then be used to implement the digital system using specific hardware such as a CPLD or FPGA.

⇒ Modeling Styles

There are three modeling techniques which can be used for describing a circuit:

- 1 Gate level modeling / structural modeling.
- 2 Dataflow modeling
- 3 Behavioral modeling.

A circuit can be described in any one of the above techniques or by taking combination of them.

⇒ Structure of Dataflow Description

- Dataflow describes how the circuit signals flow from the inputs to the outputs.
- There are some concurrent statements which allow to describe the circuit in terms of operations on signals and flow of signals in the circuit. When such concurrent statements are used in a program, the style is called a 'dataflow design'.
- Concurrent signal assignment statements are used in this type of modeling style.

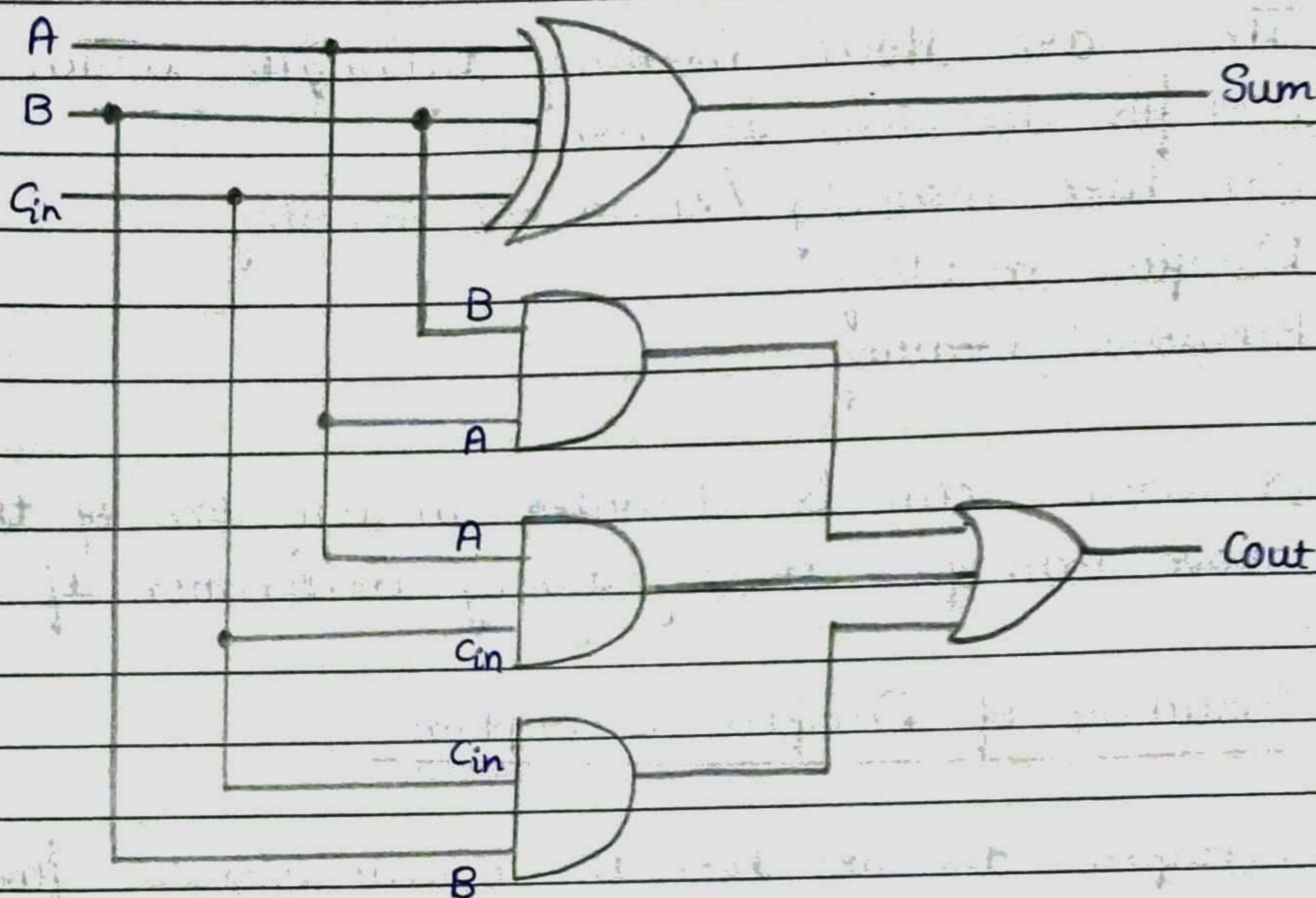
2

classmate

Date _____
Page _____

→ Example of VHDL dataflow description.

→ (1) Implementation of full-adder



entity full-add is

port (A, B, Cin : in bit;

Sum, Cout : out bit);

end full-add;

architecture adder of full-add is

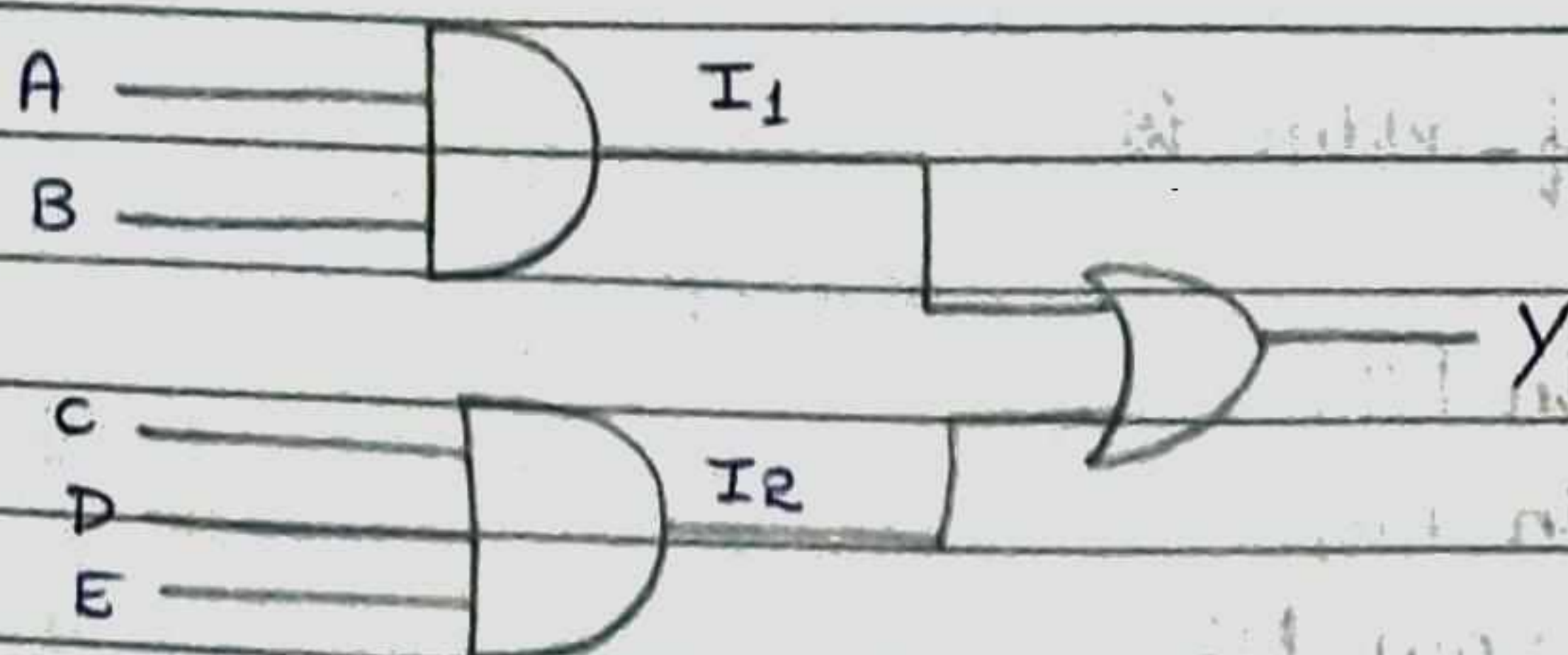
begin

Sum <= A xor B xor Cin;

Cout <= (A and B) or (Cin and A) or (Cin and B);

end adder;

② AND-OR circuit



VHDL code for AND-OR circuit.

```
entity AND_OR is
port (A, B, C, D, E : in bit;
      Y : out bit);
```

```
end;
```

architecture digital-ckt of AND-OR is

```
signal I1, I2;
```

```
begin
```

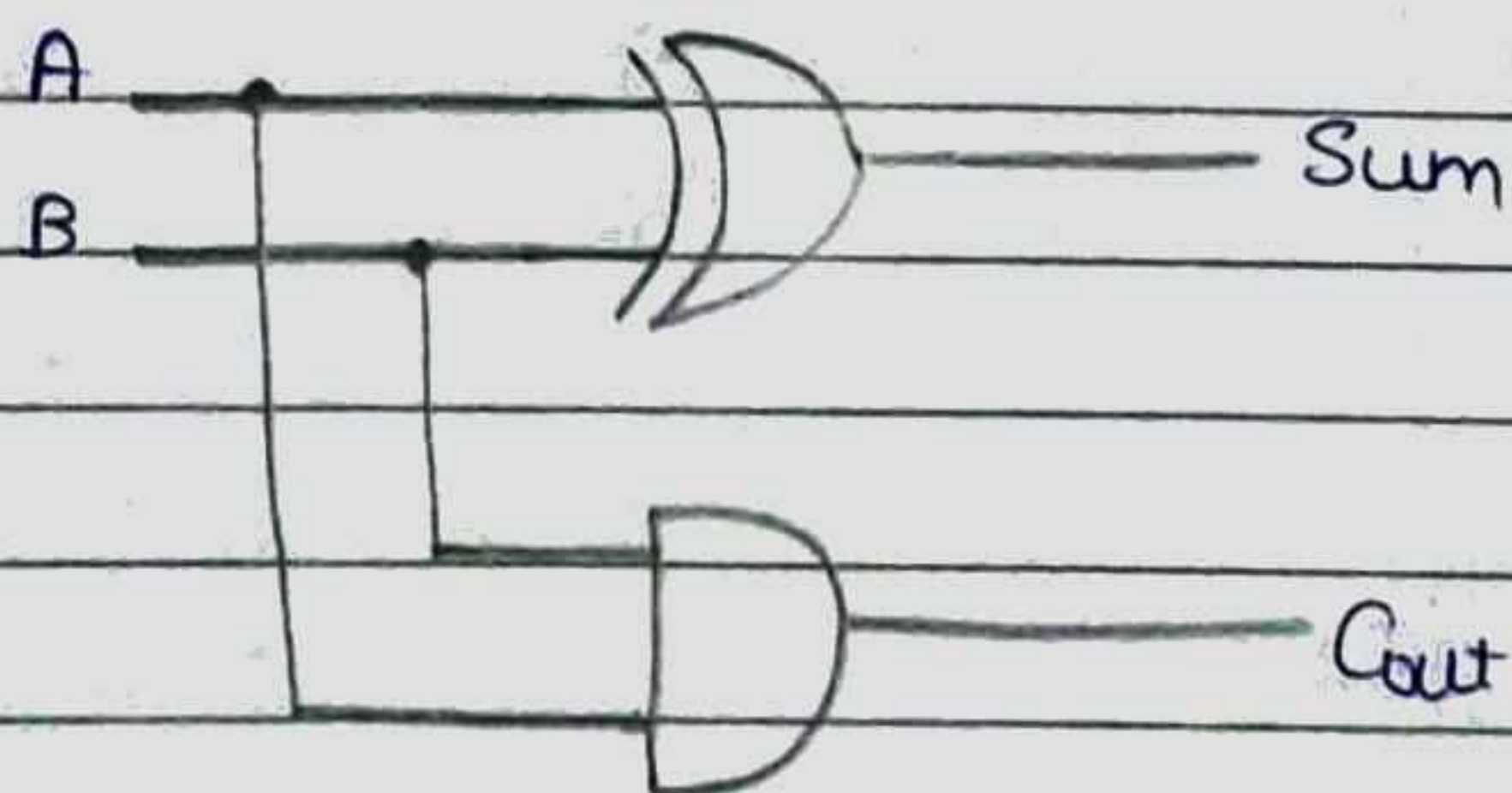
```
st1 : I1 <= A and B after 10ns; -- Time 10ns indicates the propagation
```

```
st2 : I2 <= C and D and E after 10ns; -- delay of gate and word after is
```

```
st3 : Y <= I1 or I2 after 20ns; -- used to specify propagation delay
```

```
end digital-ckt;
```

③ → Dataflow description of a half adder



Logic diagram for half adder.

VHDL half adder description

entity half-adder is
port (

A: in bit;

B: in bit;

Sum: out bit;

Cout: out bit);

end half-adder;

architecture adder of half-adder is

begin

Sum <= A xor B;

-- signal assignment statement

Cout <= A and B;

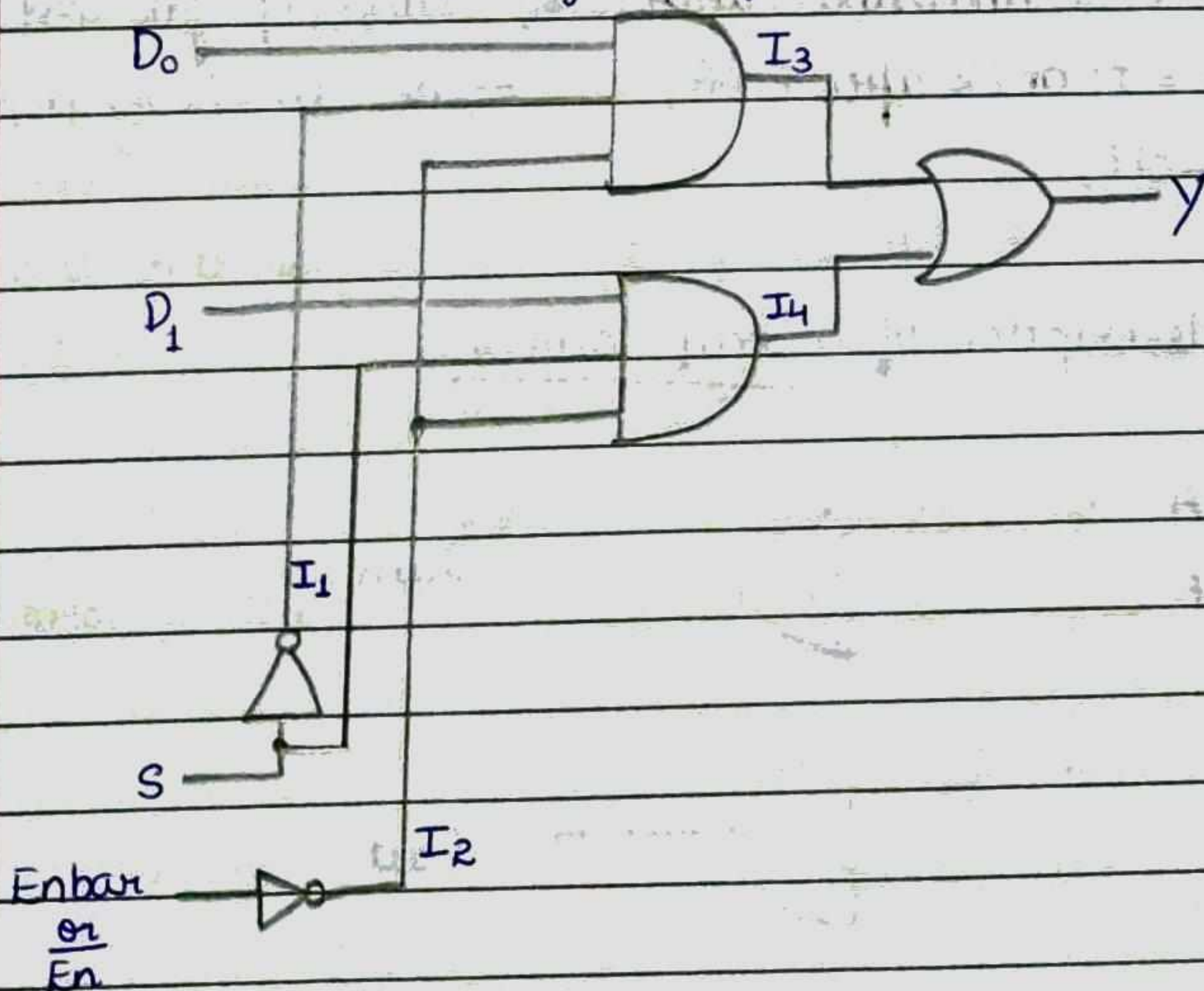
-- signal assignment statement

end adder;

4 → Multiplexer with active low enable

2x1 Multiplexer

(a) Logic diagram



(b) Logic Symbol



Truth table for a 2x1 multiplexer.

Input		Output
S	Enbar	Y
X	1	0
0	0	D ₀
1	0	D ₁

VHDL code of a 2x1 multiplexer

```
library ieee;
use ieee.std-logic-1164.all;
entity mux2x1 is
port (D0, D1, S, Enbar: in std-logic;
      Y: out std-logic);
```

```
end mux2x1;
```

```
architecture MUX of mux2x1 is
```

```
signal I1, I2, I3, I4: std-logic;
```

```
begin
```

-- Assume 10 nanoseconds propagation delay.

-- for all and, or, and not.

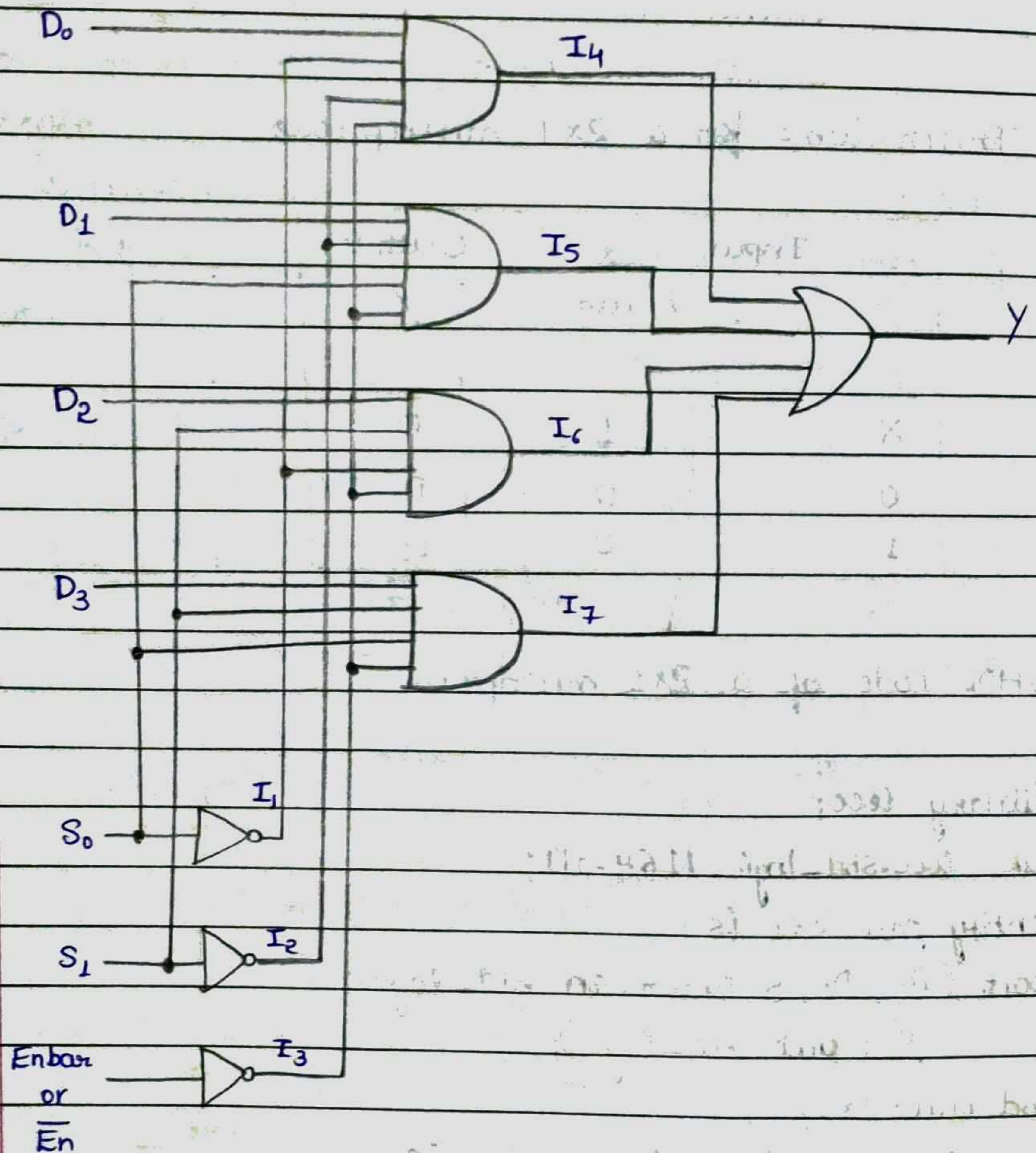
```
I1 <= not S after 10ns;
```

```
I2 <= not Enbar after 10ns;
```

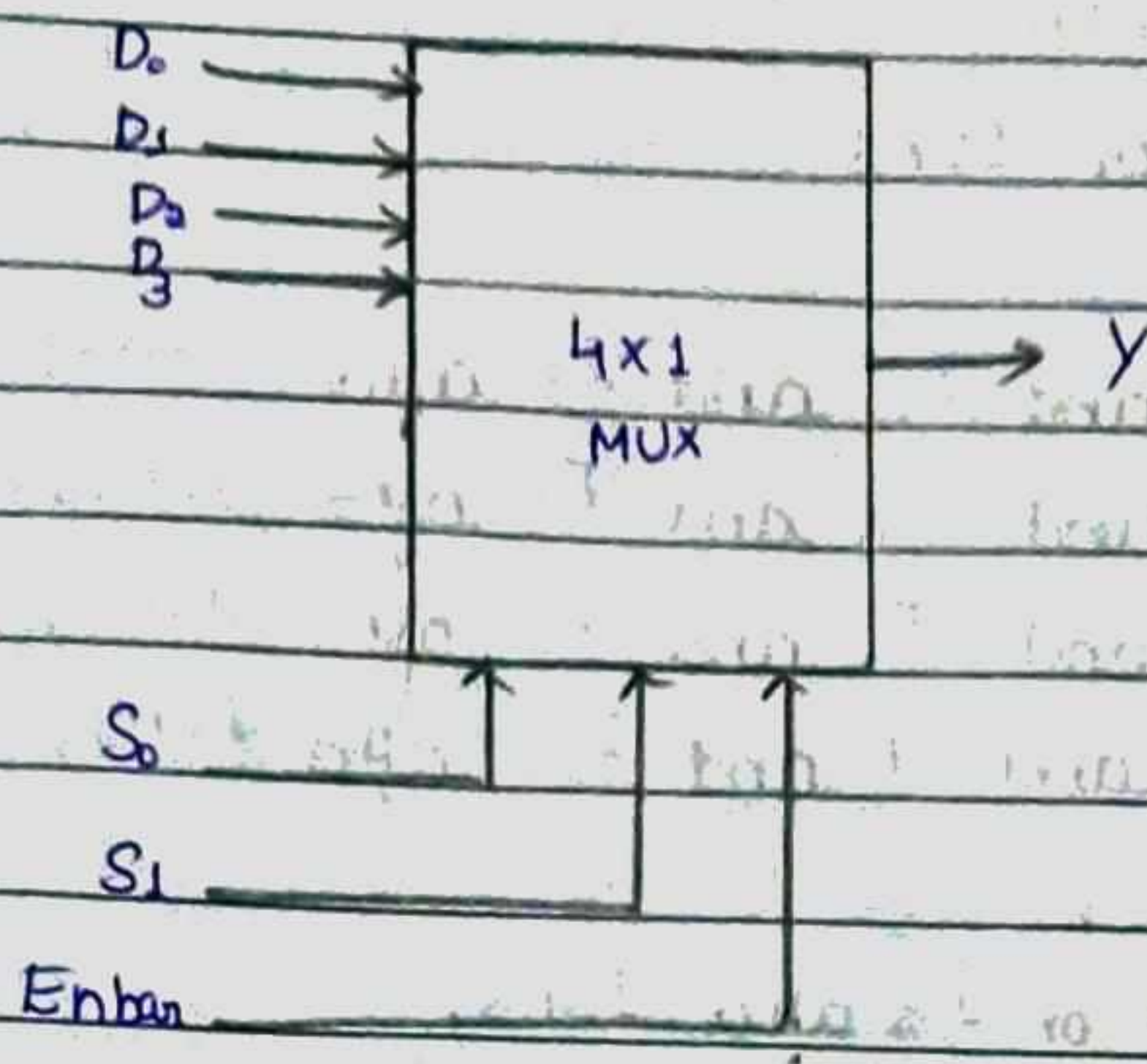

$I_3 \leq D_0$ and I_1 and I_2 after 10ns;
 $I_4 \leq D_1$ and S and I_2 after 10ns;
 $Y \leq I_3$ or I_4 after 10ns;
 end MUX;

5) \rightarrow 4x1 multiplexer.

(a) Logic diagram



(b) Logic symbol



Truth Table

Input			Output
S ₁	S ₀	Enbar	Y
X	X	1	0
0	0	0	D ₀
0	1	0	D ₁
1	0	0	D ₂
1	1	0	D ₃

VHDL code of a 4X1 multiplexer

```

library ieee;
use ieee.std_logic_1164.all;
entity mux4x1 is
port (D: in std_logic_vector(3 downto 0);
      S, Enbar: in std_logic;
      Y: out std_logic);
end mux4x1;
architecture MUX of mux4x1 is
signal I1, I2, I3, I4, I5, I6, I7: std_logic;
begin

```

-- Assume 10 nanoseconds propagation delay
 -- for all and, or, and not.

$I_1 \leq \text{not } S_0 \text{ after } 10\text{ns};$

$I_2 \leq \text{not } S_1 \text{ after } 10\text{ns};$

$I_3 \leq \text{not } \text{Enbar} \text{ after } 10\text{ns};$

$I_4 \leq D_0 \text{ and } I_1 \text{ and } I_2 \text{ and } I_3 \text{ after } 10\text{ns};$

$I_5 \leq D_1 \text{ and } S_0 \text{ and } I_2 \text{ and } I_3 \text{ after } 10\text{ns};$

$I_6 \leq D_2 \text{ and } S_1 \text{ and } I_1 \text{ and } I_3 \text{ after } 10\text{ns};$

$I_7 \leq D_3 \text{ and } S_0 \text{ and } S_1 \text{ and } I_3 \text{ after } 10\text{ns};$

$Y \leq I_4 \text{ or } I_5 \text{ or } I_6 \text{ or } I_7 \text{ after } 10\text{ns};$

end MUX;

⇒ Structure of VHDL Behavioral Description

- The keyword in the behavioral description is process.
- Every behavioral description has to include in the process body.
- The process (A,B) is a concurrent statement; so its execution is initiated by the occurrence of an event.
- The ports A and B included in the process (A,B) statement is called sensitivity list.
- Any change in the state of any element of the sensitivity list is treated as an event.
- The process is activated (initiated) only if an event occurs; otherwise process remains inactive.
- If the process has no sensitivity list, the process is executed continuously.

62

VHDL behavioral description of half adder

```

entity half-adder is
port (A: in bit;
      B: in bit;
      Sum: out bit;
      Cout: out bit);
end half-adder;
architecture adder of half-adder is
begin
process (A, B)
begin
    Sum <= A xor B after 10 ns; -- signal-assignment statement 1
    Cout <= A and B after 10 ns; -- signal-assignment statement 2
                                   -- with 10 nanoseconds delays.
end process;
end adder;

```

⇒ Gate Level / Structural Description

- In this VHDL description, the entity part is same as that of behavioral description. However, architecture part has two components: declaration and instantiation.
- In declaration part all different components used in the system description are declared. For example, following description declares AND gate component.

Component and2

```

port (I1, I2: in std_logic;
      O1: out std_logic);
end component;

```

The and2 component has two inputs: I1 and I2 and one output O1.

- Once the component is declared we can use the same component one or more times in the system description.
- The instantiation part of the code maps the generic inputs/outputs to the actual inputs/outputs of the system.
- For example, the statement `and2 port map (A, B, Cout);` maps A to input I₁ of and2, input B to input I₂ of and2, and output Cout to output O₁ of and2. This mapping means that the logic relationship between A, B and Cout is the same as between I₁, I₂ and O₁.
- VHDL half-adder description.

Ex

```
library ieee;
```

```
use ieee.std-logic-1164.all;
```

```
entity xor2 is
```

```
port (I1, I2: in std-logic;
      O1: out std-logic);
```

```
end xor2;
```

```
architecture xor-gate of xor2 is
```

```
begin
```

```
    O1 <= I1 xor I2;
```

```
end xor-gate;
```

```
library ieee;
```

```
use ieee.std-logic-1164.all;
```

```
entity and2 is
```

```
port (I1, I2: in std-logic; O1: out std-logic);
```

```
end and2;
```

```
architecture and-gate of and2 is
```

```
begin
```

```
    O1 <= I1 and I2;
```

```
end and-gate;
```



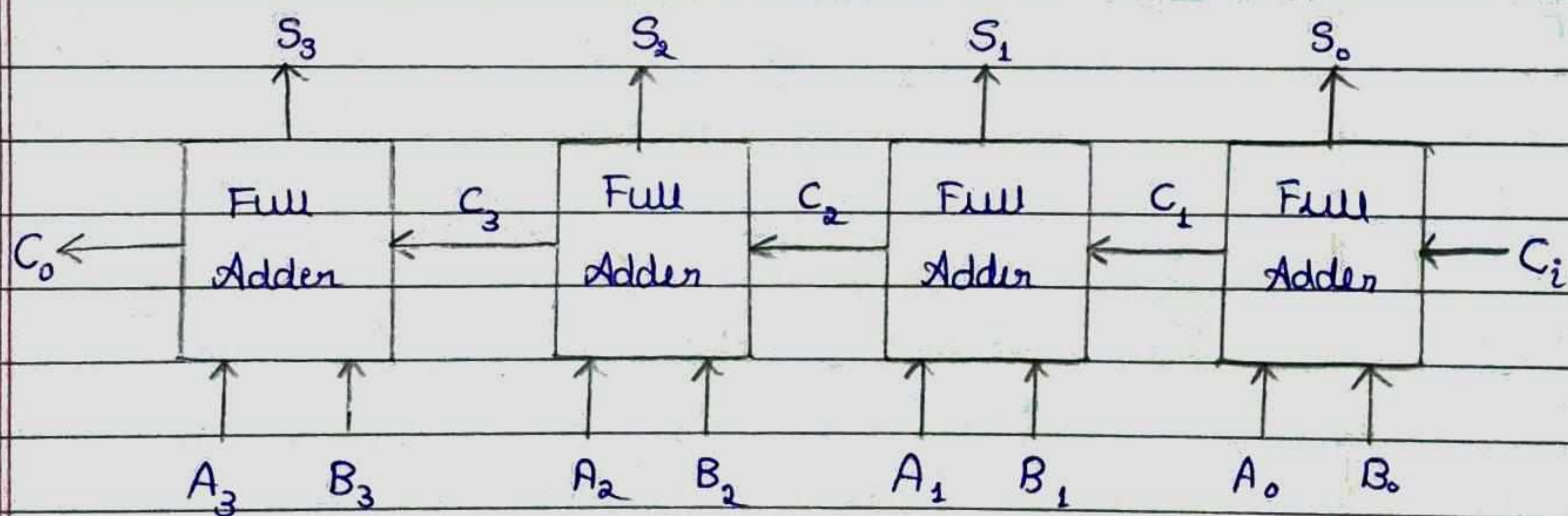
```

library ieee;
use ieee.std_logic_1164.all;
entity half-adder is
port (A, B: in std_logic;
      Sum, Cout: out std_logic);
end half-adder;
architecture adder of half-adder is
component xor2
port (I1, I2: in std_logic;
      O1: out std_logic);
end component;
component and2
port (I1, I2: in std_logic;
      O1: out std_logic);
end component;
begin
X1: xor2 port map (a, b, s);
A1: and2 port map (a, b, c);
end adder;

```

→ Structural description of 4-Bit Adder

4-Bit Binary Adder.



entity Adder4;

architecture Structure of Adder4 is
component FullAdder

port (X, Y, Cin: in bit; - Inputs

Cout, Sum: out bit); - Outputs

end component;

signal C: bit_vector(3 downto 1);

begin - instantiate four copies of the FullAdder

FA0: FullAdder port map (A(0), B(0), C_i, C(1), S(0));

FA1: FullAdder port map (A(1), B(1), C(1), C(2), S(1));

FA2: FullAdder port map (A(2), B(2), C(2), C(3), S(2));

FA3: FullAdder port map (A(3), B(3), C(3), C_o, S(3));

end Structure;

add list A B C_o C C_i S

- put these signals on the output list

force A

- set the A inputs to 1111

force B 0001

- set the B inputs to 0001

force C_i 1

- set C_i to 1

run 50 ns

- run the simulation for 50ns

force C_i 0

force A 0101

force B 1110

run 50ns

Module V

⇒ Registers and Counters

A register is a group of flip-flops that can be used to store a binary number.

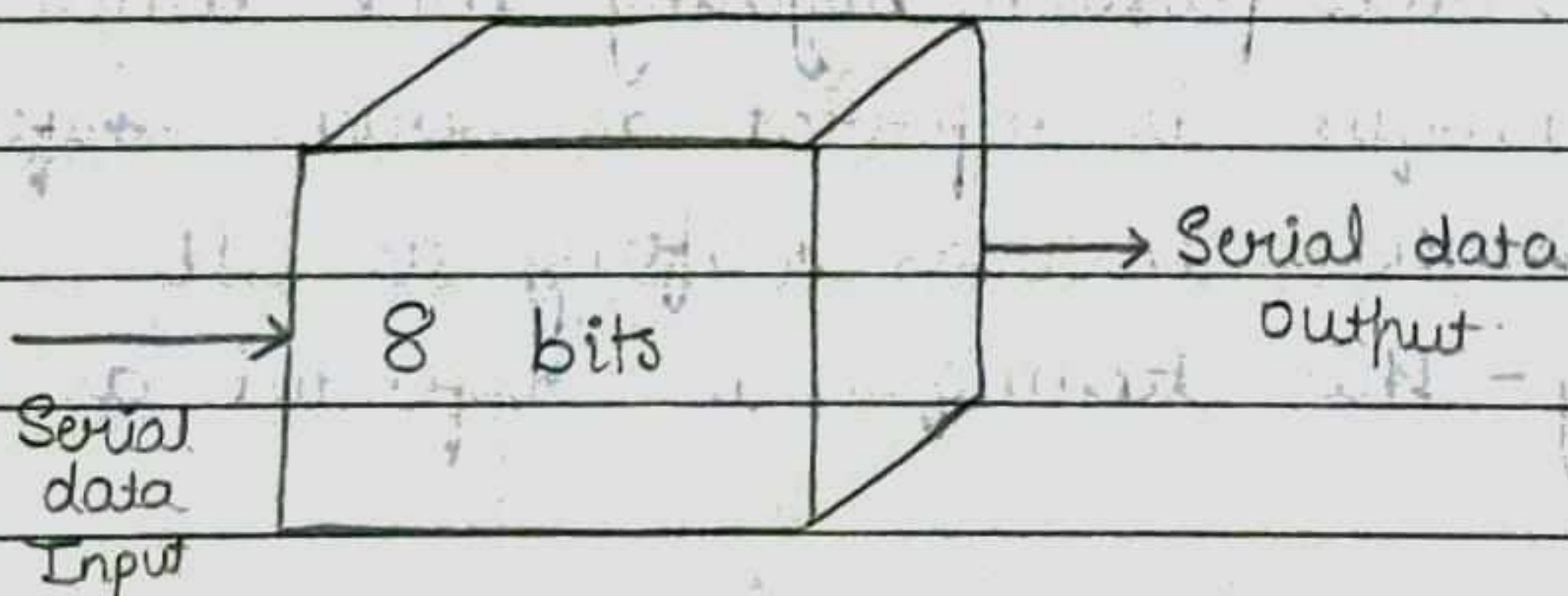
There must be one flip-flop for each bit in the binary number. For example, a register used to store 8 bit binary number must have 8 flip-flops.

The flip-flops must be connected such that the binary number can be entered or shifted into the register & possibly shifted out. A group of flip-flops connected to provide either or both of these functions is called a shift register.

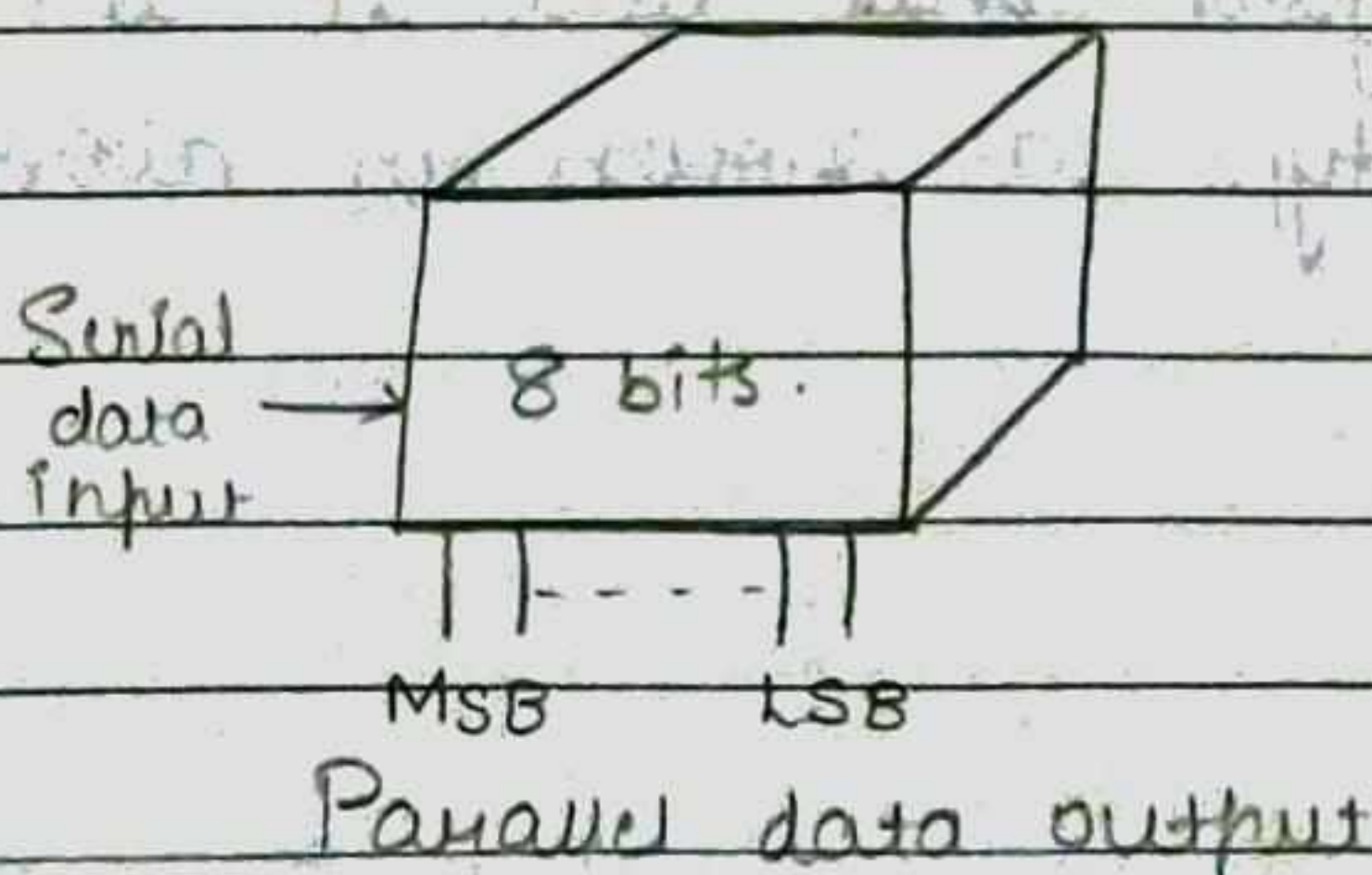
Types of Shift registers:

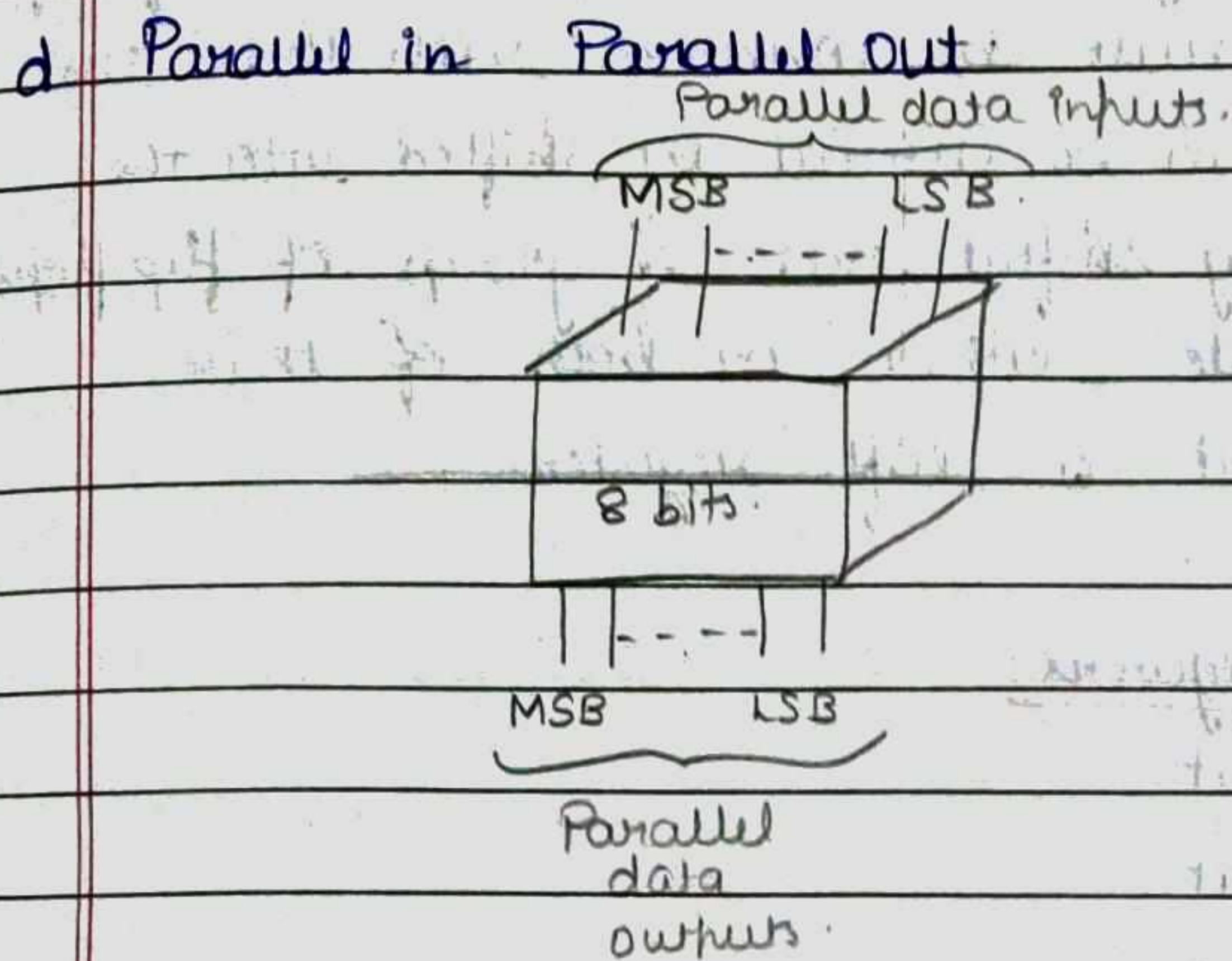
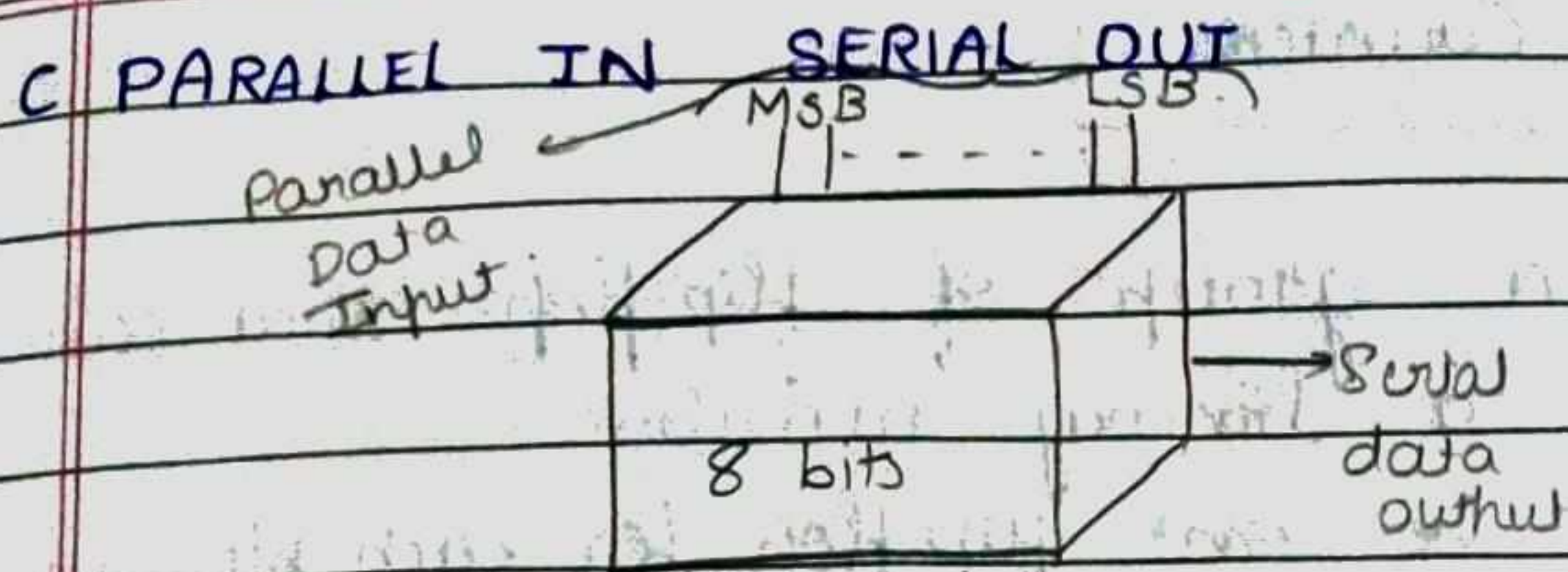
- 1 Serial In Serial Out
- 2 Serial In Parallel Out
- 3 Parallel In Serial Out
- 4 Parallel In Parallel Out

a SERIAL IN SERIAL OUT



b SERIAL IN PARALLEL OUT





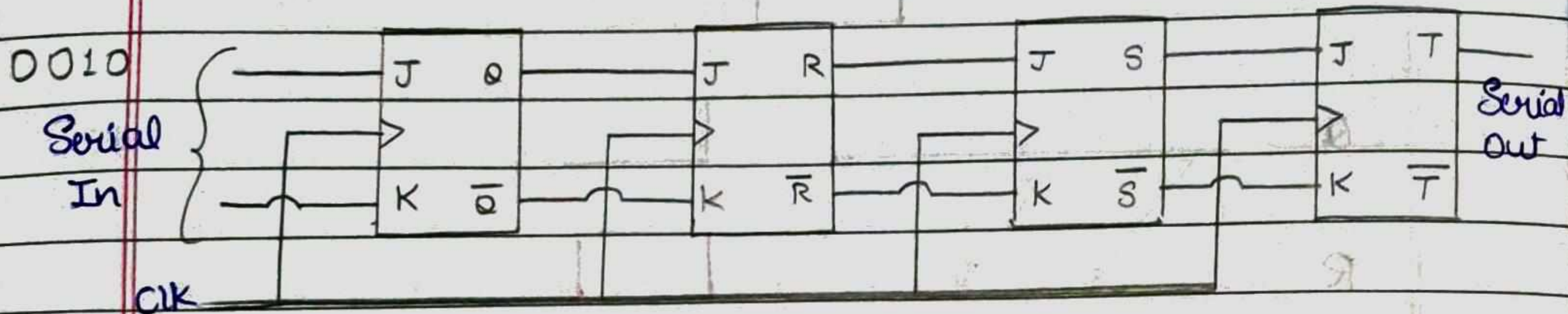
- The bits in a binary number can be moved from one place to another in either of two ways.
- The first method involves shifting the data one bit at a time in a serial fashion, beginning with either the MSB or LSB - This technique is referred as serial shifting.
- The second method involves shifting all the data bits simultaneously - This technique is referred as parallel shifting.
- There are two ways to shift data into a register, serial or parallel & similarly two ways to shift the data out of the register - This leads to the construction of 4 basic register type as shown in above figure.

(3)

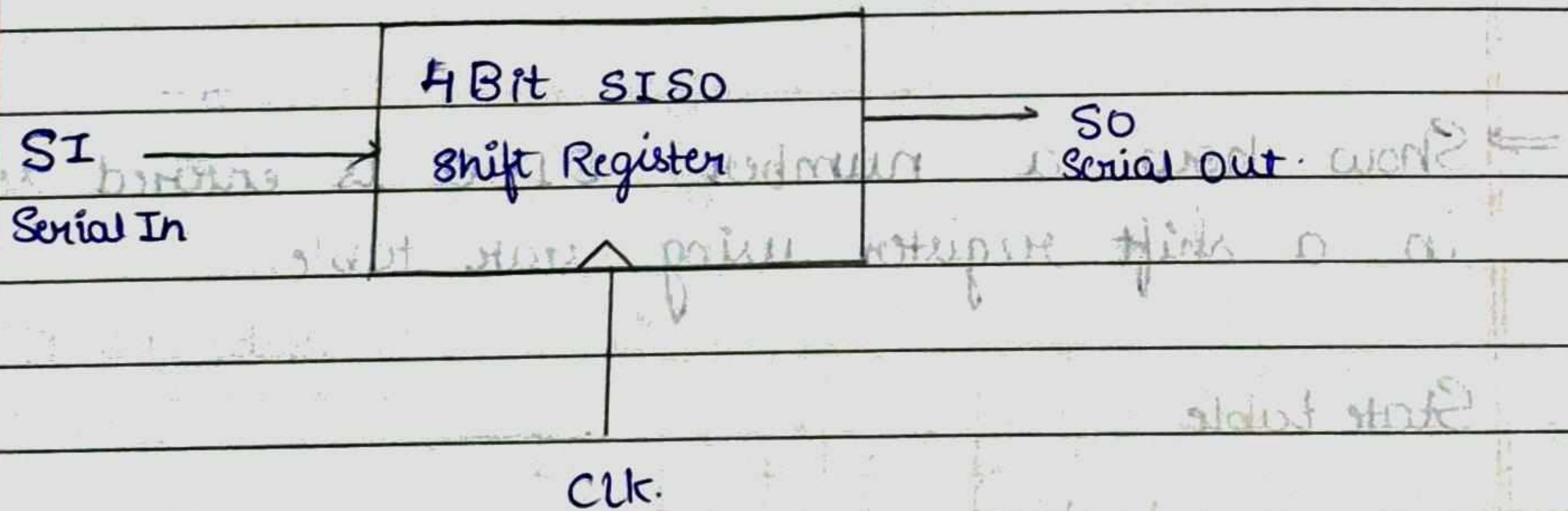
1 SERIAL IN SERIAL OUT (SISO)

→ Design 4 bit SISO Shift Register using JK flipflops

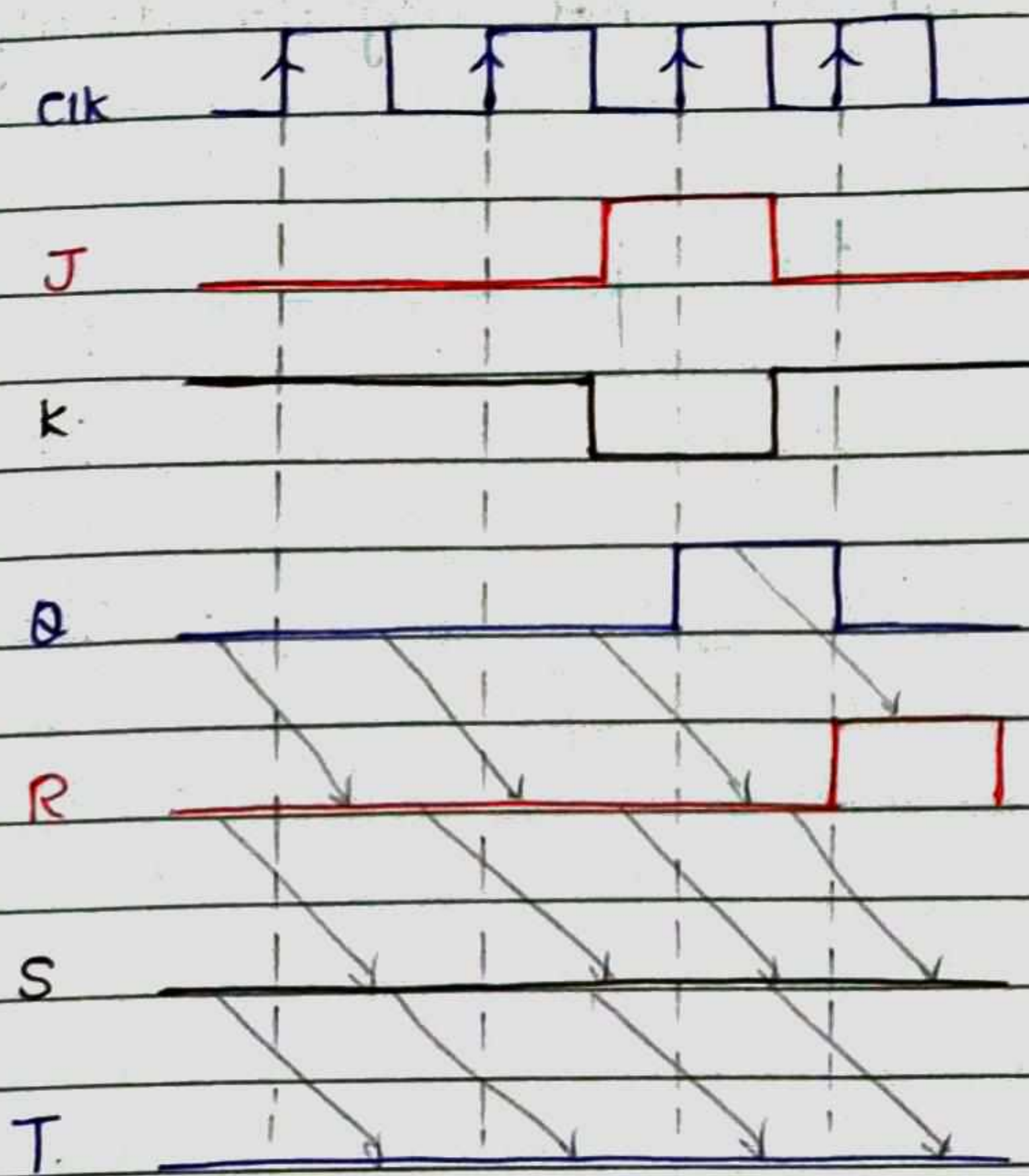
Logic diagram



Block diagram



Timing diagram



⇒ Show how a number 0100 is entered serially in a shift register using state table.

State table.

Clock	SI	Q	R	S	T
0	0	0	0	0	0
1	1	0	0	0	0
2	0	1	0	0	0
3	0	0	1	0	0
4		0	0	1	0

Diagonal arrows in the original image indicate the shift of data from one stage to the next (e.g., from SI to Q, and from Q to R, etc.).

5

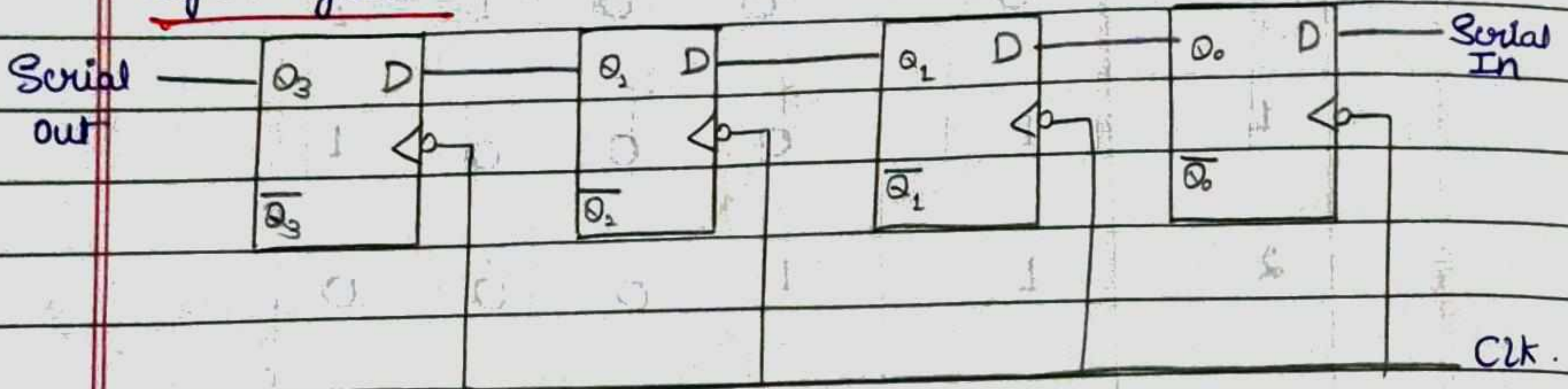
classmate

Date _____

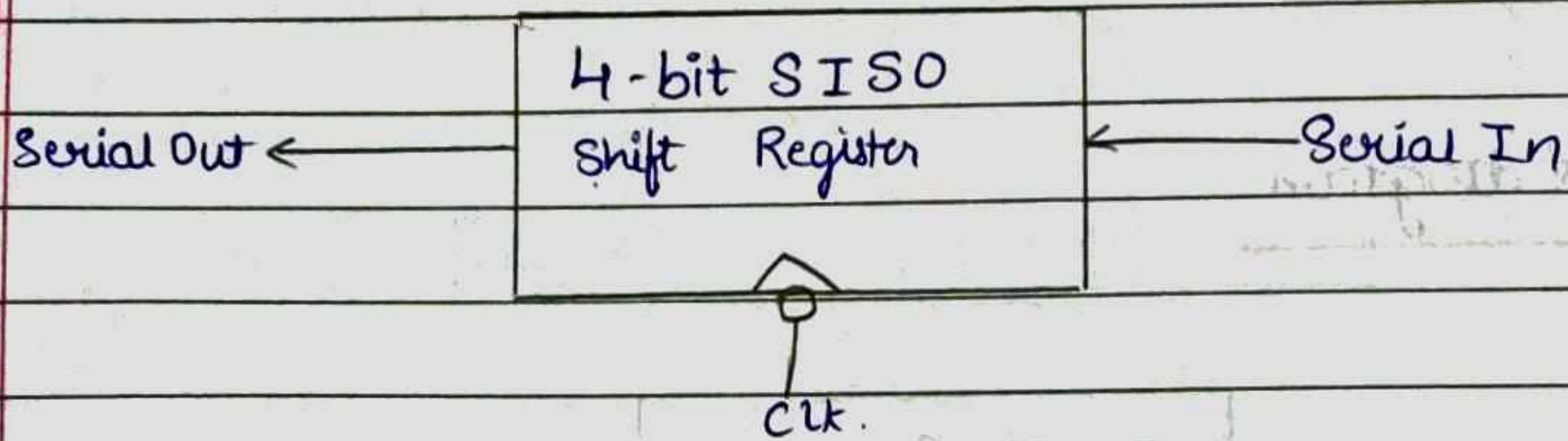
Page _____

⇒ Design 4 bit SISO left shift register using D flip-flop (Negative edge triggered).
Initial contents of the register is 0011 (Q_0, Q_1, Q_2, Q_3)

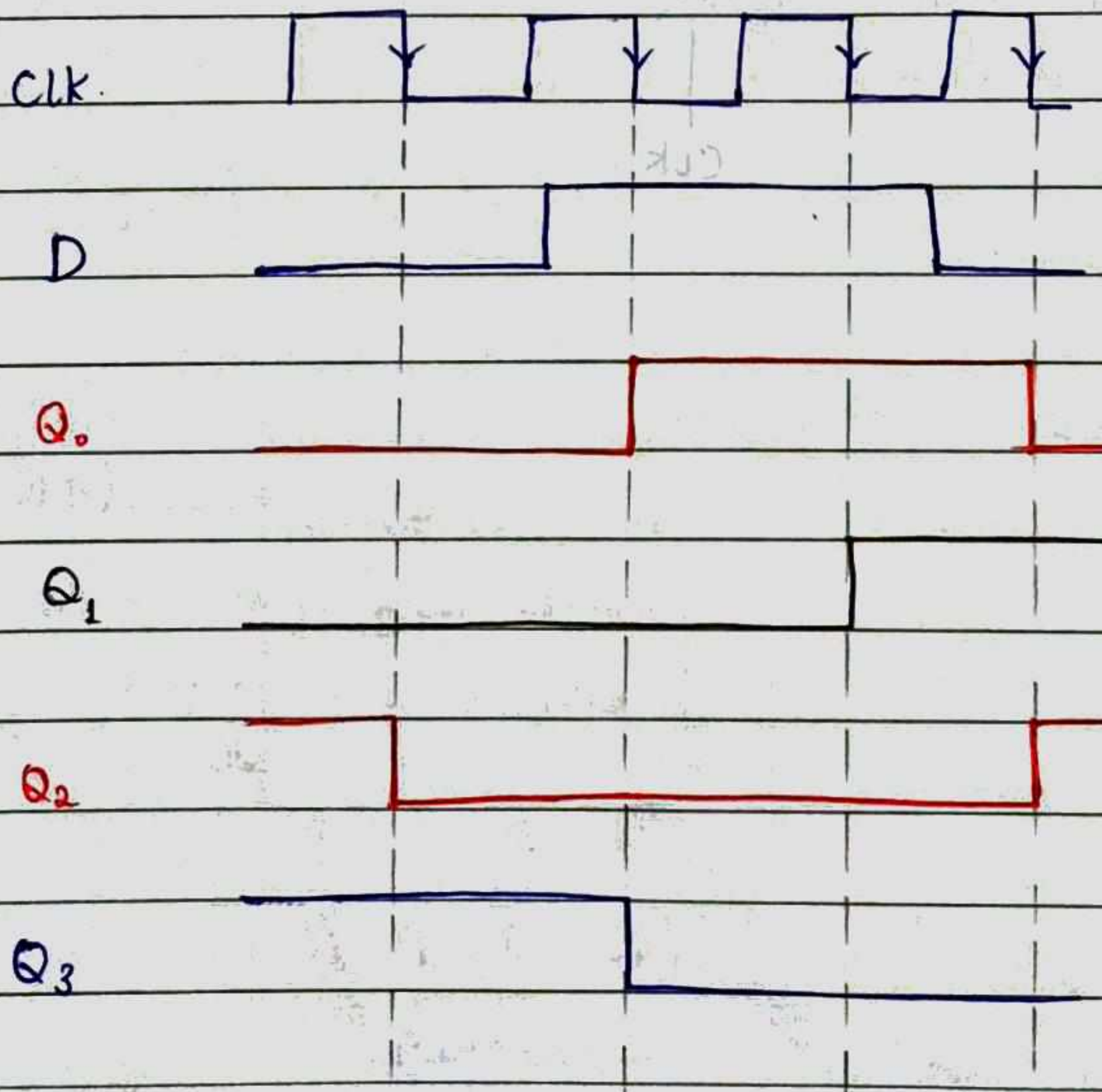
Logic Diagram



Block diagram



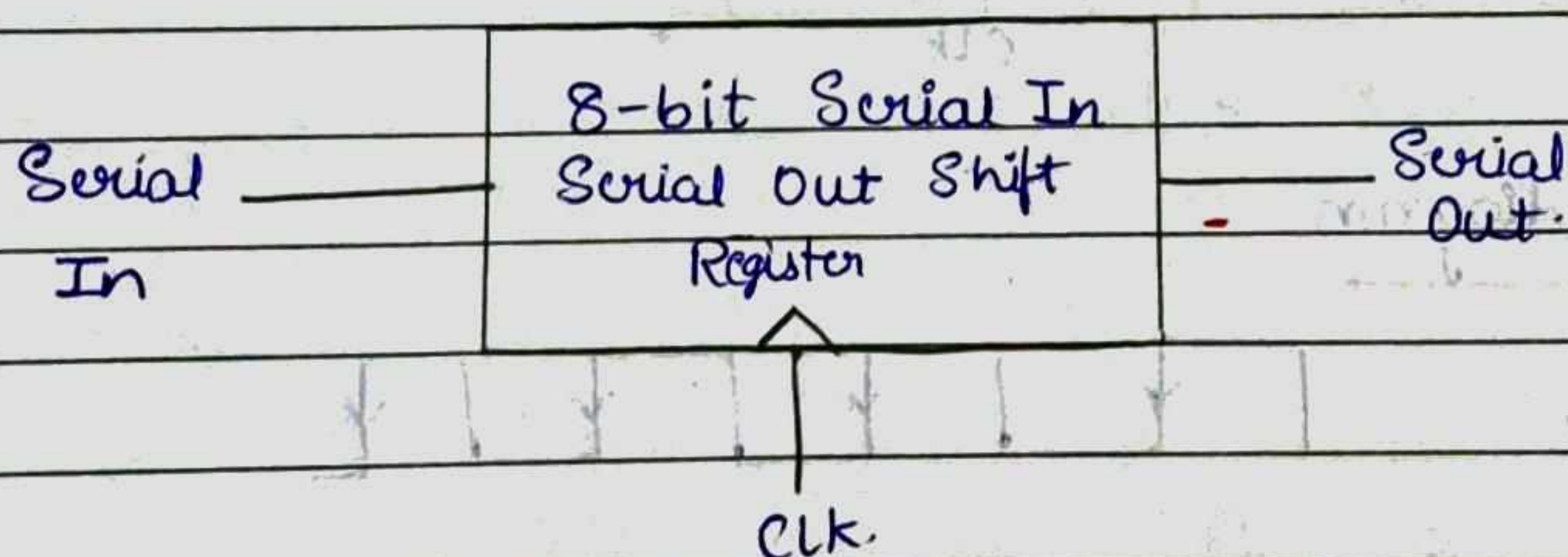
Timing diagram



State table

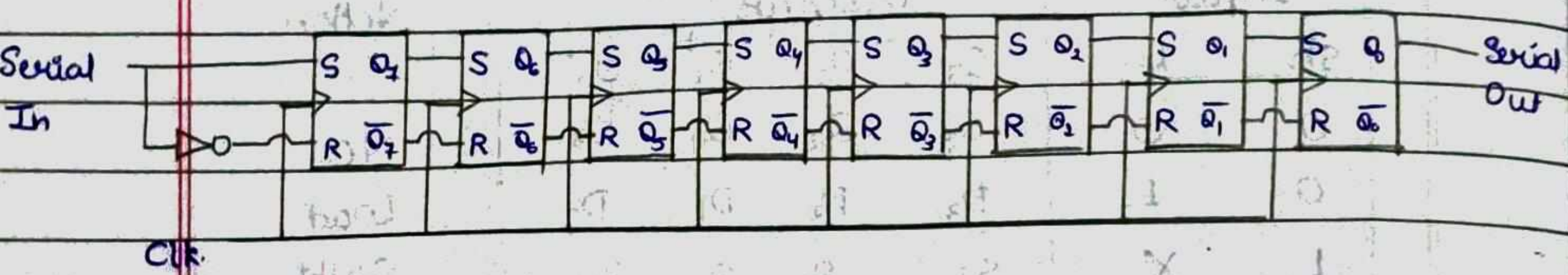
Clk	SI	Q ₀	Q ₁	Q ₂	Q ₃
0	0	0	0	1	1
1	1	0	0	0	1
2	1	1	0	0	0
3	0	1	1	0	0
4		0	1	1	0

⇒

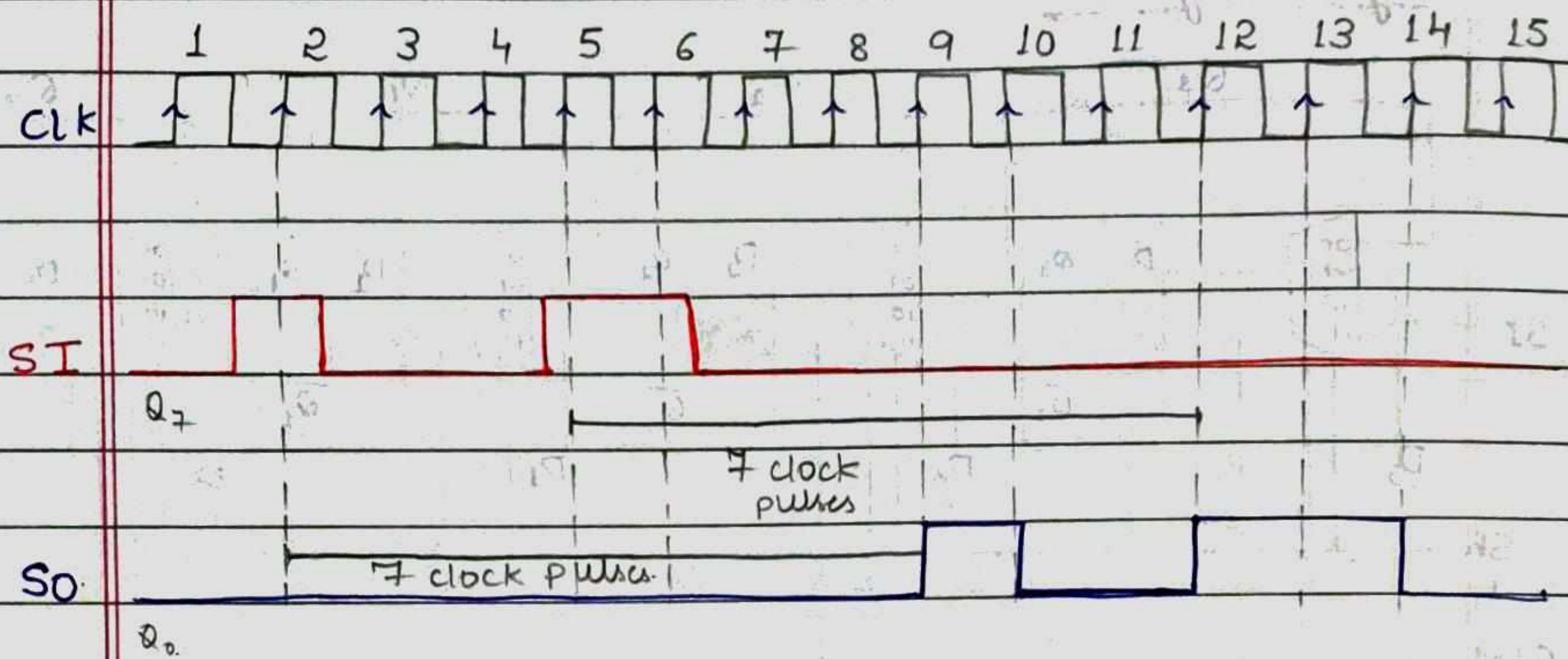
a) Block diagram

(7)

b) Logic diagram

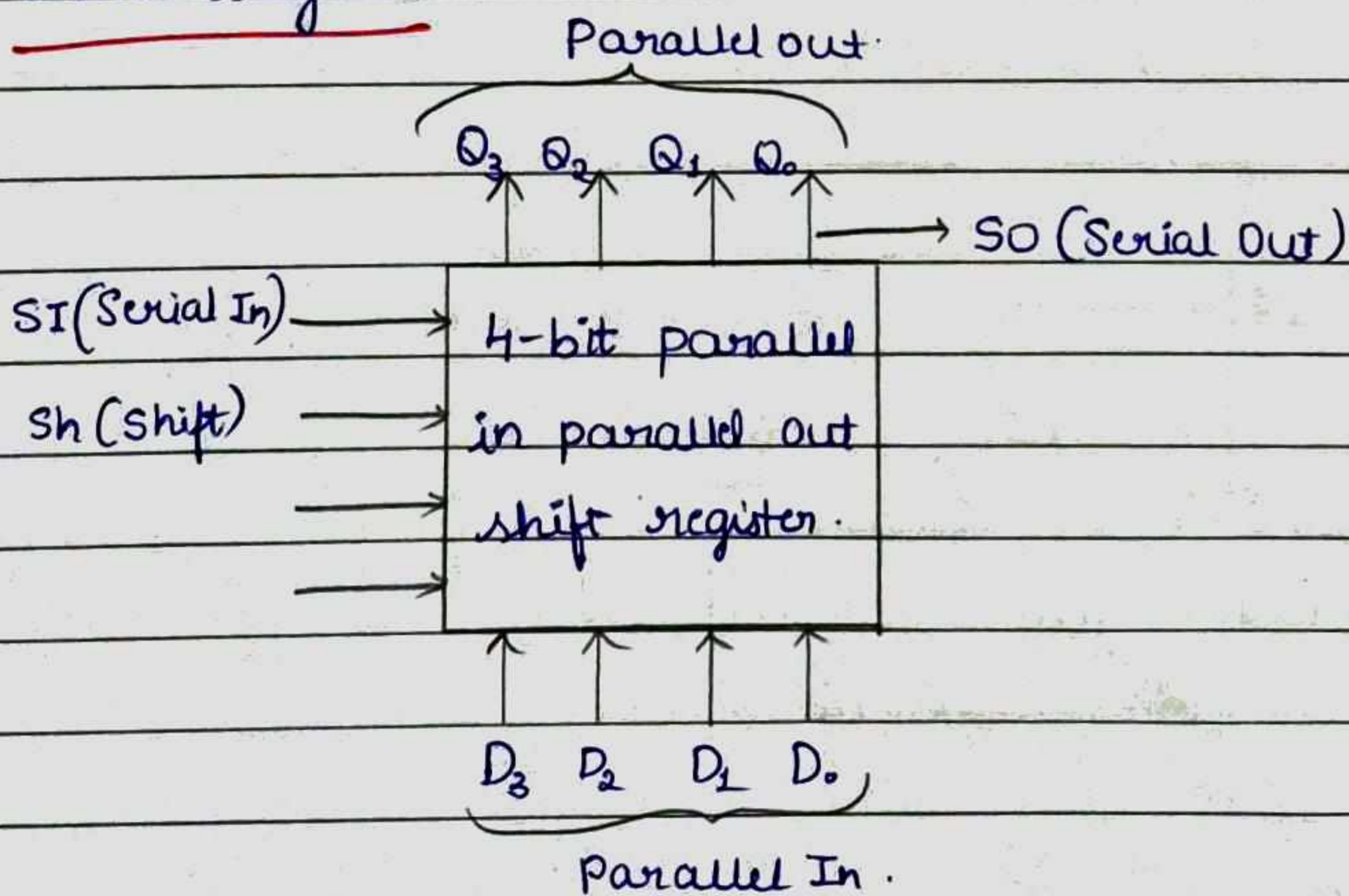


c) Timing diagram



10M
⇒ Parallel In Parallel out Right Shift Register

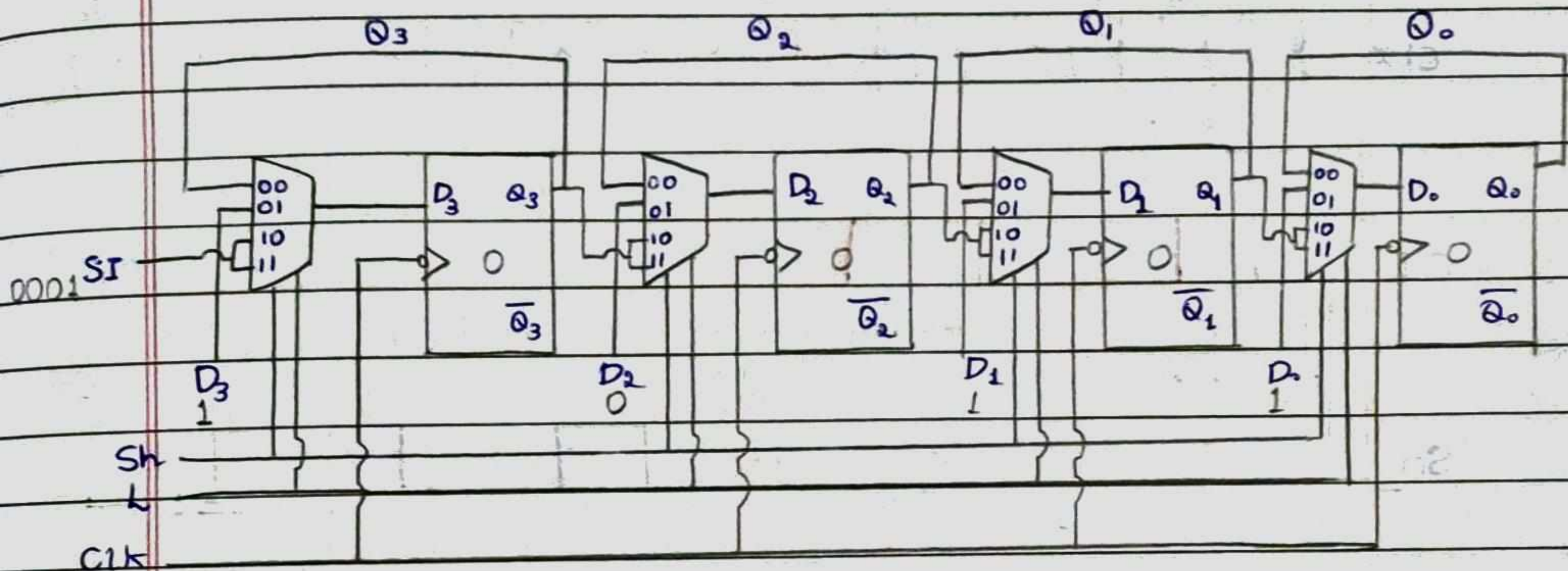
⇒ Block diagram



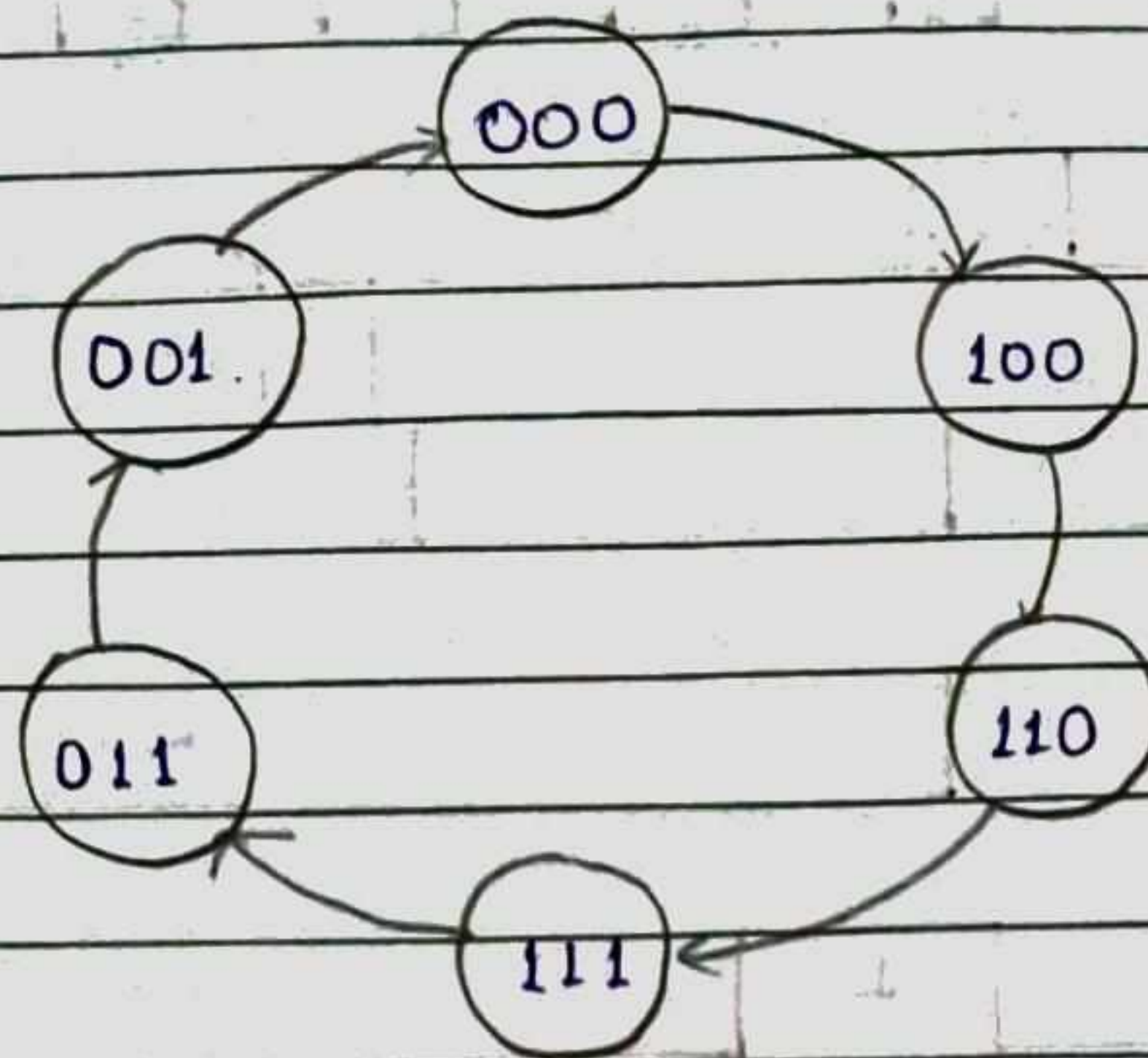
Shift Register operation.

Input		Outputs				Action
Sh	L	Q_3^+	Q_2^+	Q_1^+	Q_0^+	
0	0	Q_3	Q_2	Q_1	Q_0	NC
0	1	D_3	D_2	D_1	D_0	Load
1	X	SI	Q_3	Q_2	Q_1	Shift

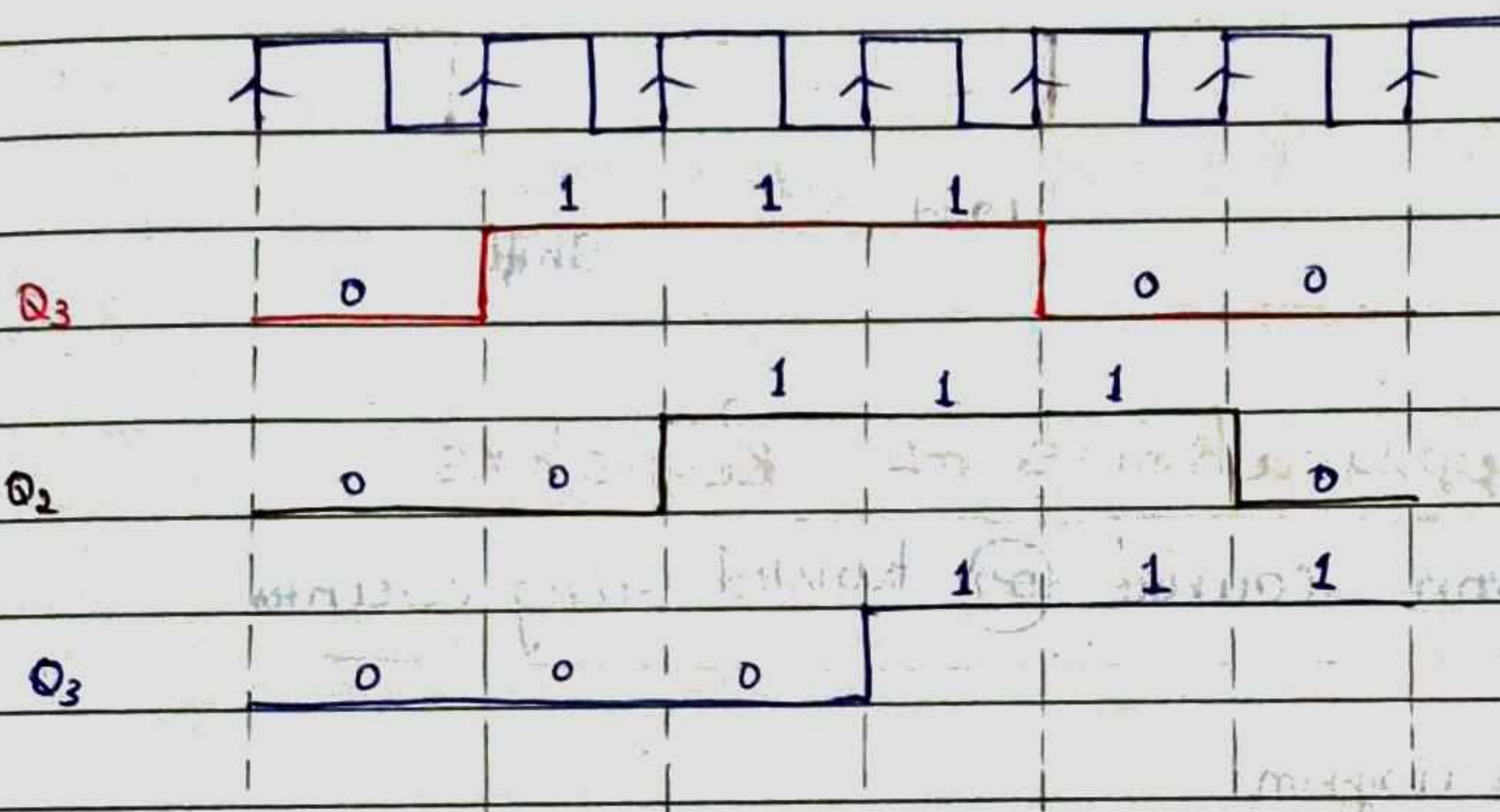
Logic diagram.



Transition Graph or state diagram



Timing diagram



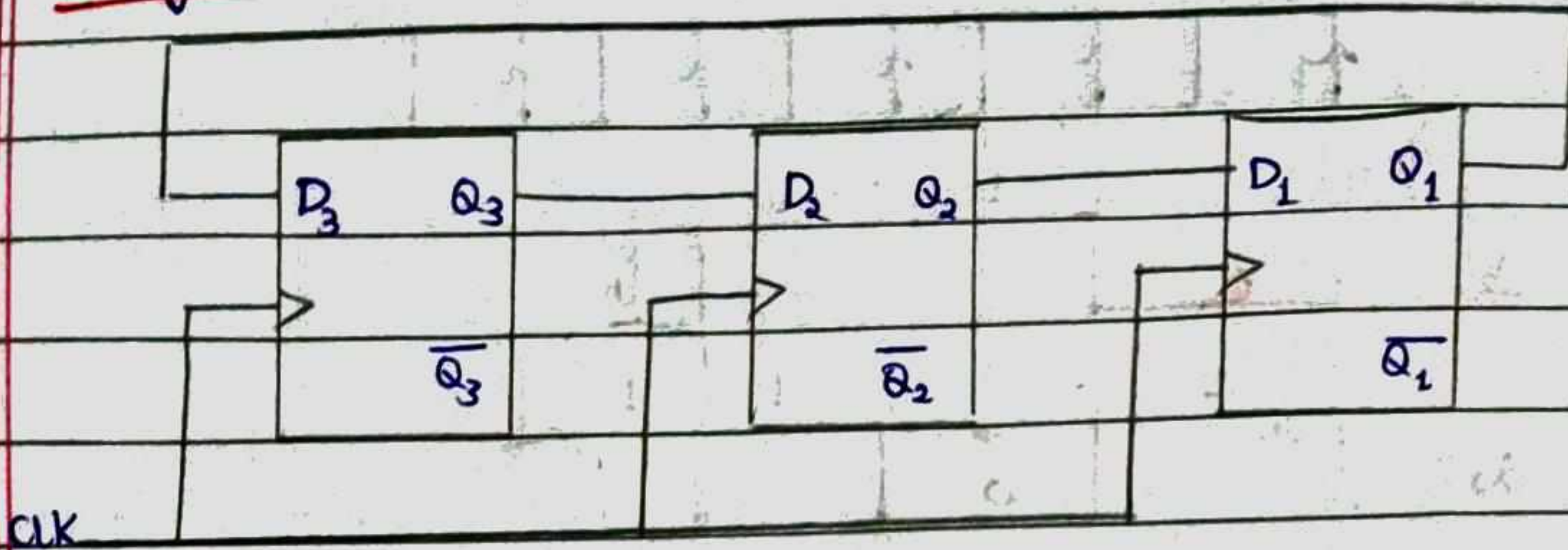
Counter definition:

A circuit that cycles through a fixed sequence of states is called a counter.

Johnson Counter definition:

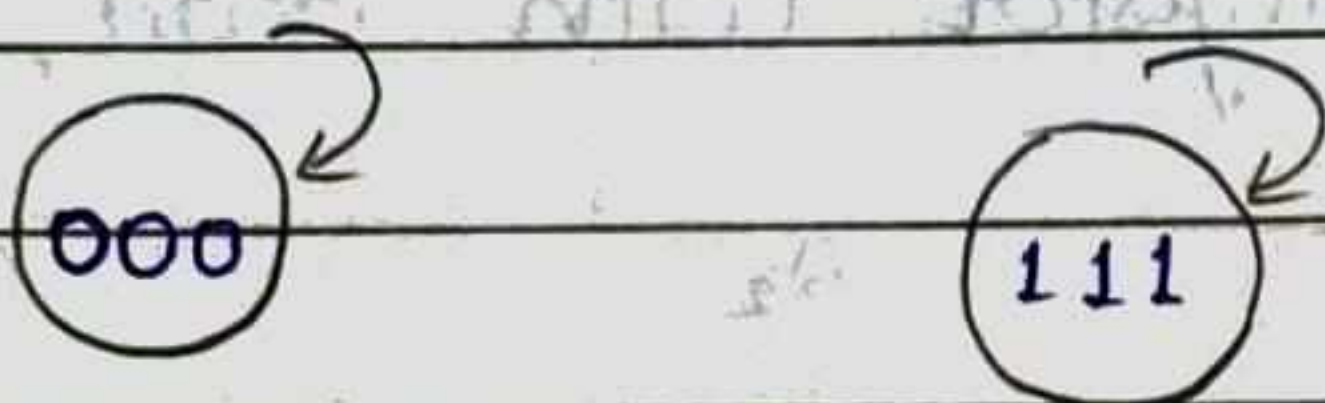
A shift register with inverted feedback is called a Johnson counter or a twisted ring counter.

App of register

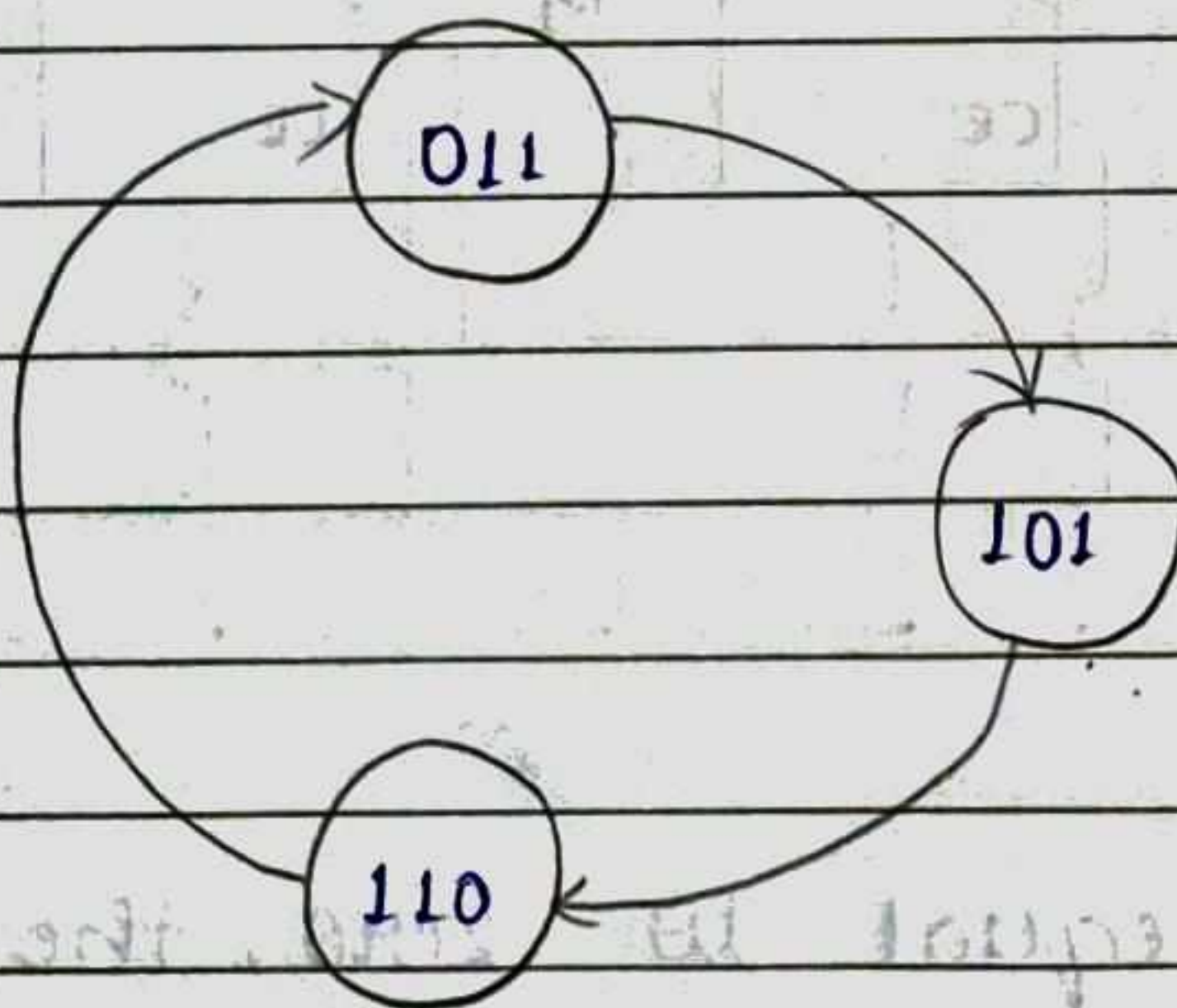
Applications of Registers⇒ Ring counter

- 1 ⇒ If the data in flipflops is 000 (or) 111 then all the states will be similar i.e., 000 (or) 111
 ↳ we can use this for application which is repeatedly produce a same state.

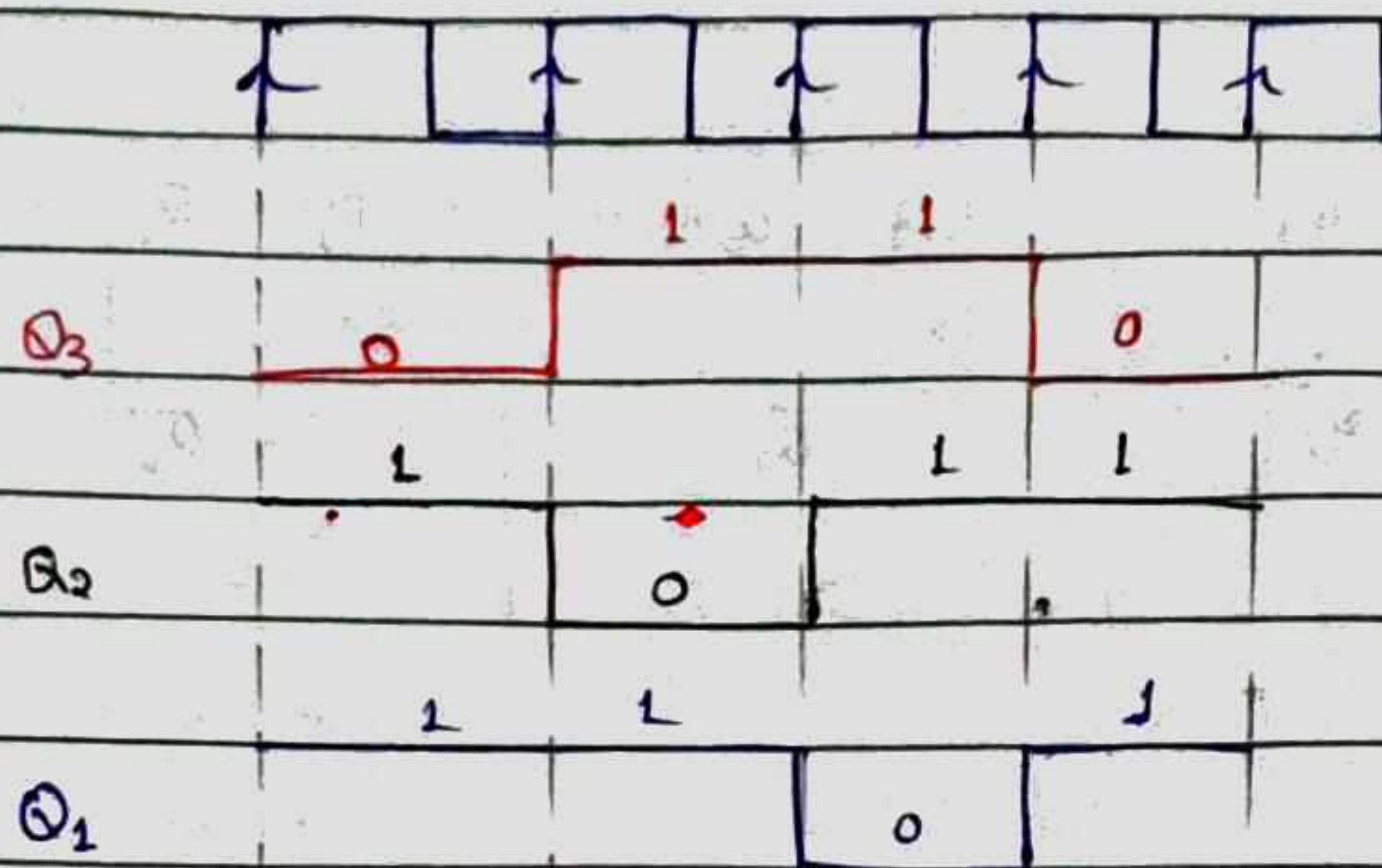
⇒ State or transition diagram



2. If the data in flipflops is 011, then



Timing assuming data in a register at 011.

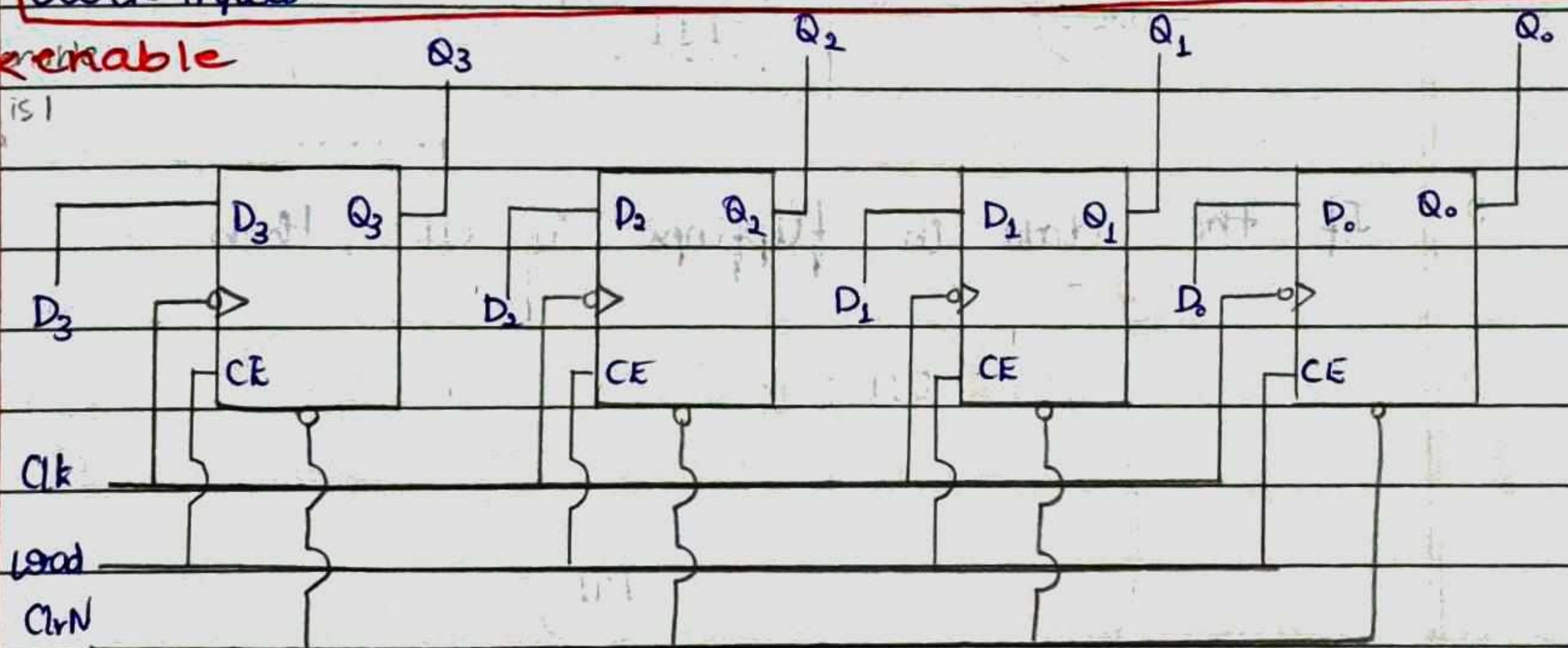


A shift register with non inverted feedback is called a ring counter.

⇒ 4-bit D-flipflop register with data, load, clear & clock inputs.

CE: clock enable

when load is 1
it works

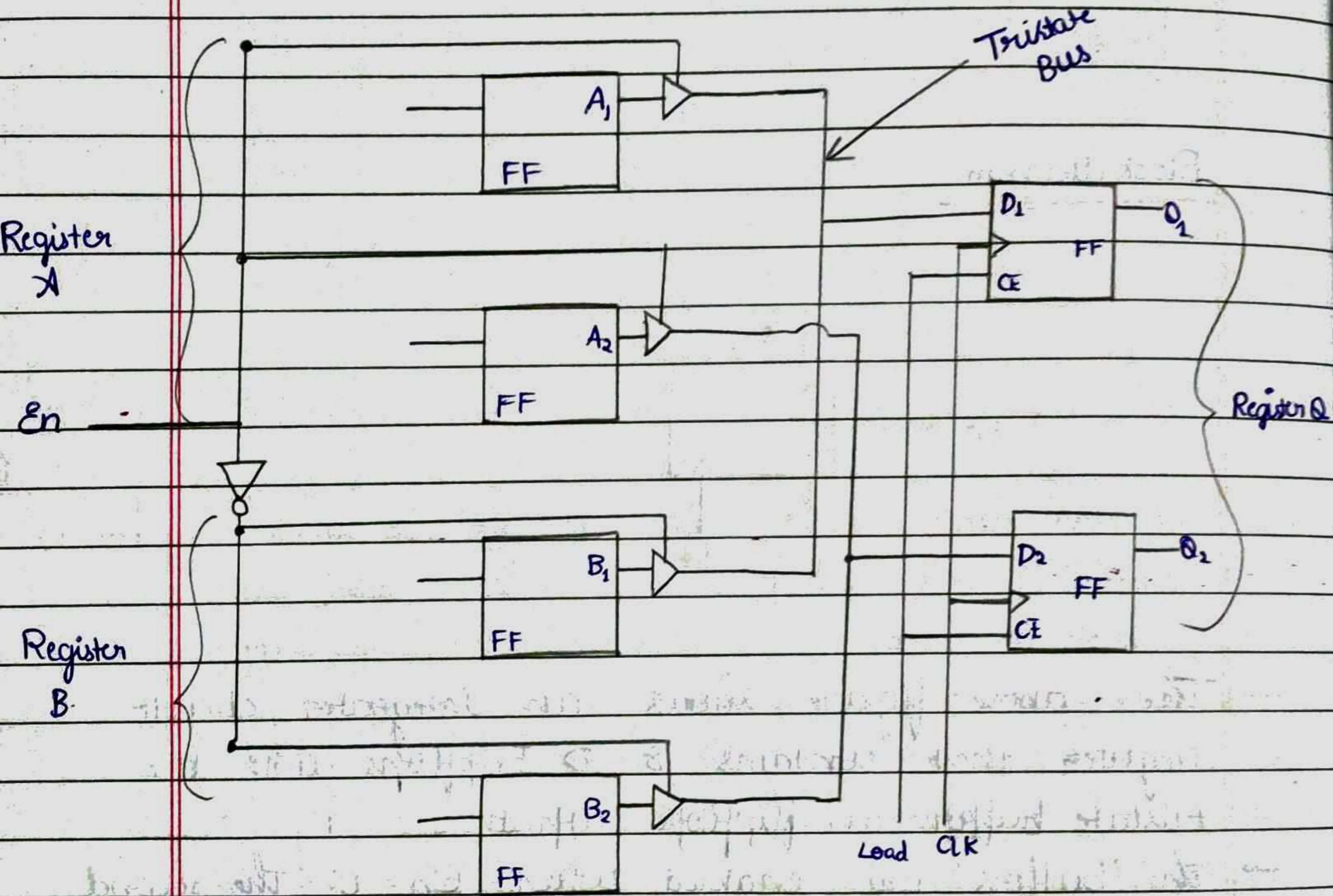


- When load is equal to zero, the register is not clocked & holds its present value.
- When load is equal to one, the clock is transmitted to the flipflop & the data applied to D inputs will be loaded into the flipflops on the falling edge of the clock. For example: if the Q outputs $Q_3 Q_2 Q_1 Q_0$ is 0000 & the data inputs $D_3 D_2 D_1 D_0$ is 1101. After the falling edge of the clock, Q will change from

0000 to 1101.

→ The clr is a asynchronous input, the bubble at the clear input indicates that the logic '0' is required to clear the flipflop.

⇒ Data transfer between the register.



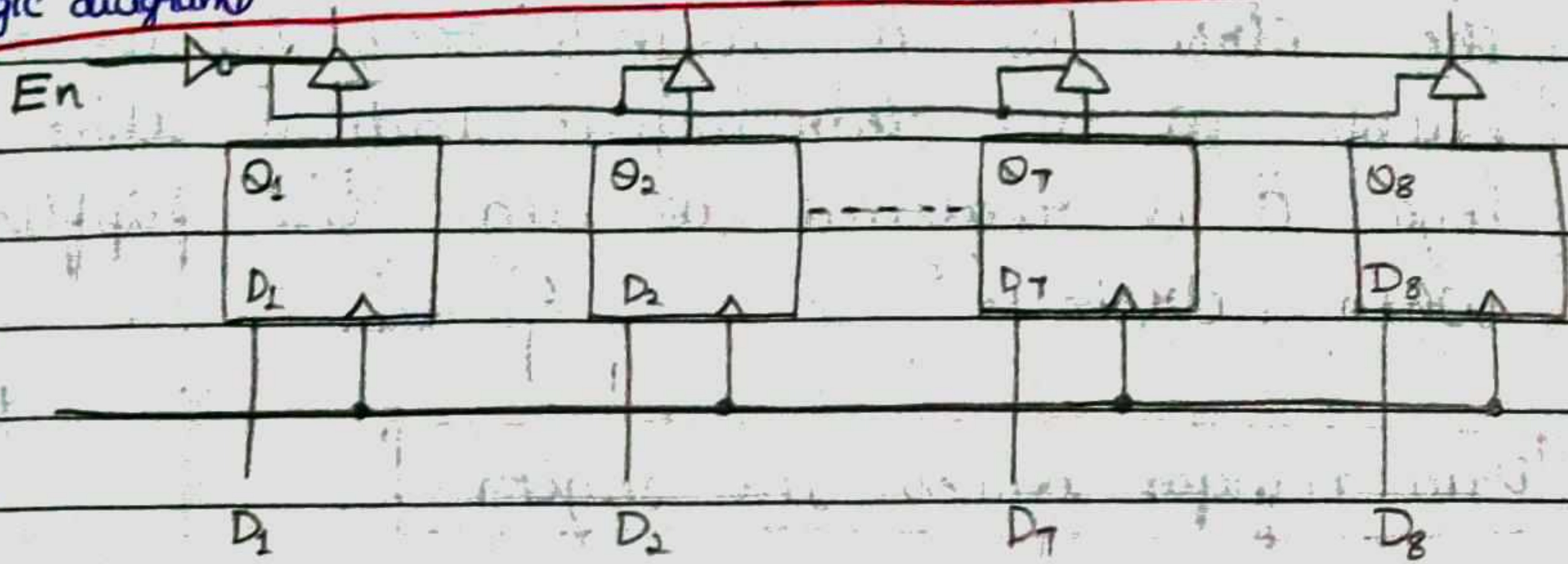
Register A = FF A_1 & A_2

Register B = FF B_1 & B_2

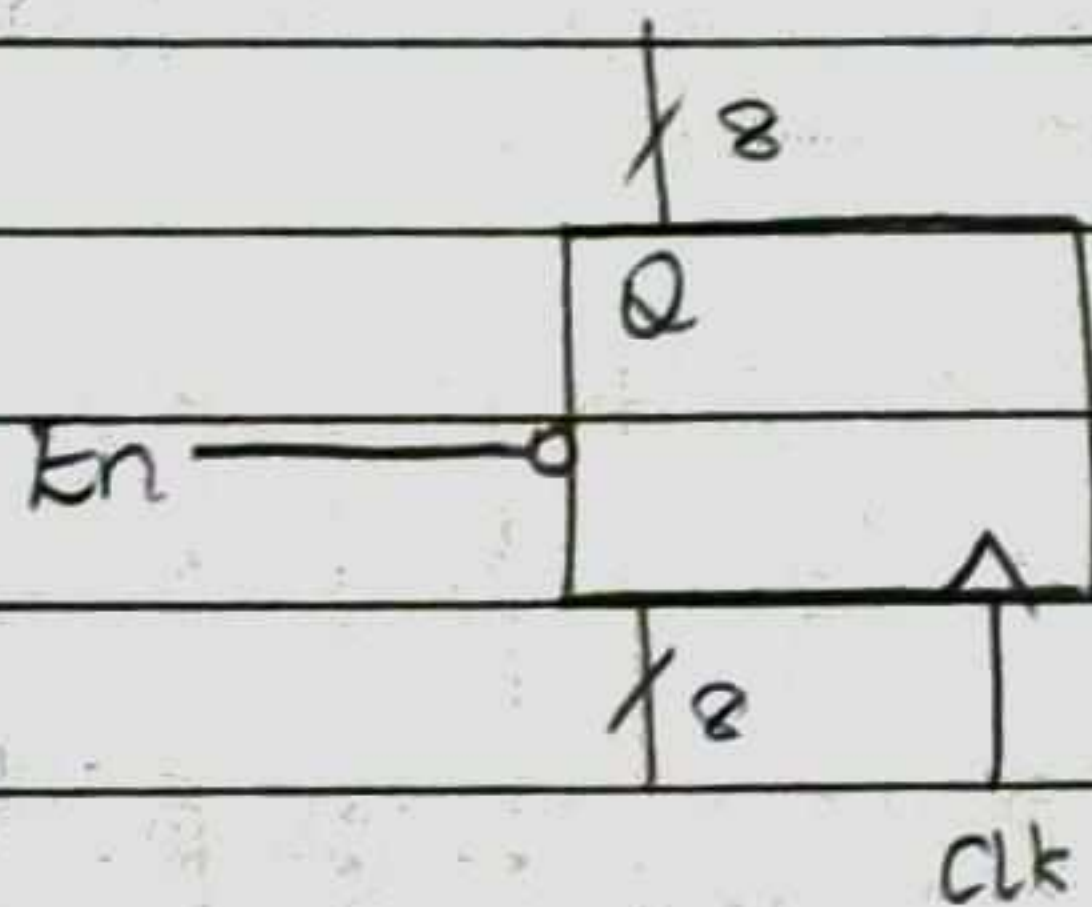
Register Q = FF Q_1 & Q_2

8-bit register with tristate output

Logic diagram

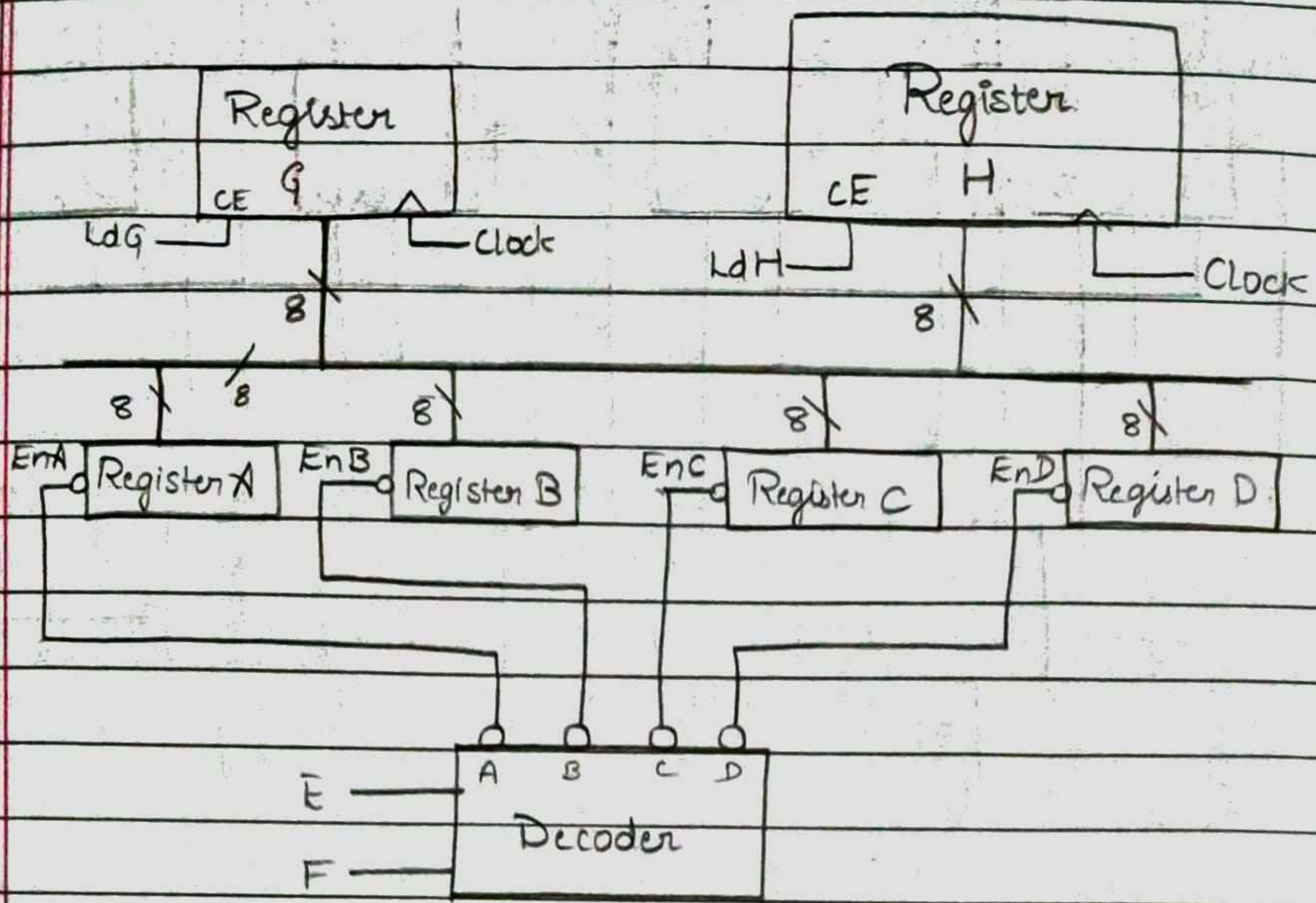


Block diagram



- The above figure shows an integrated circuit register that contains 8 D Flipflops with the tristate buffers at flipflop outputs.
- The buffers are enabled when $En = 0$. The second figure shows the block diagram for 8 bit register.

⇒ Data Transfer using the tristate bus

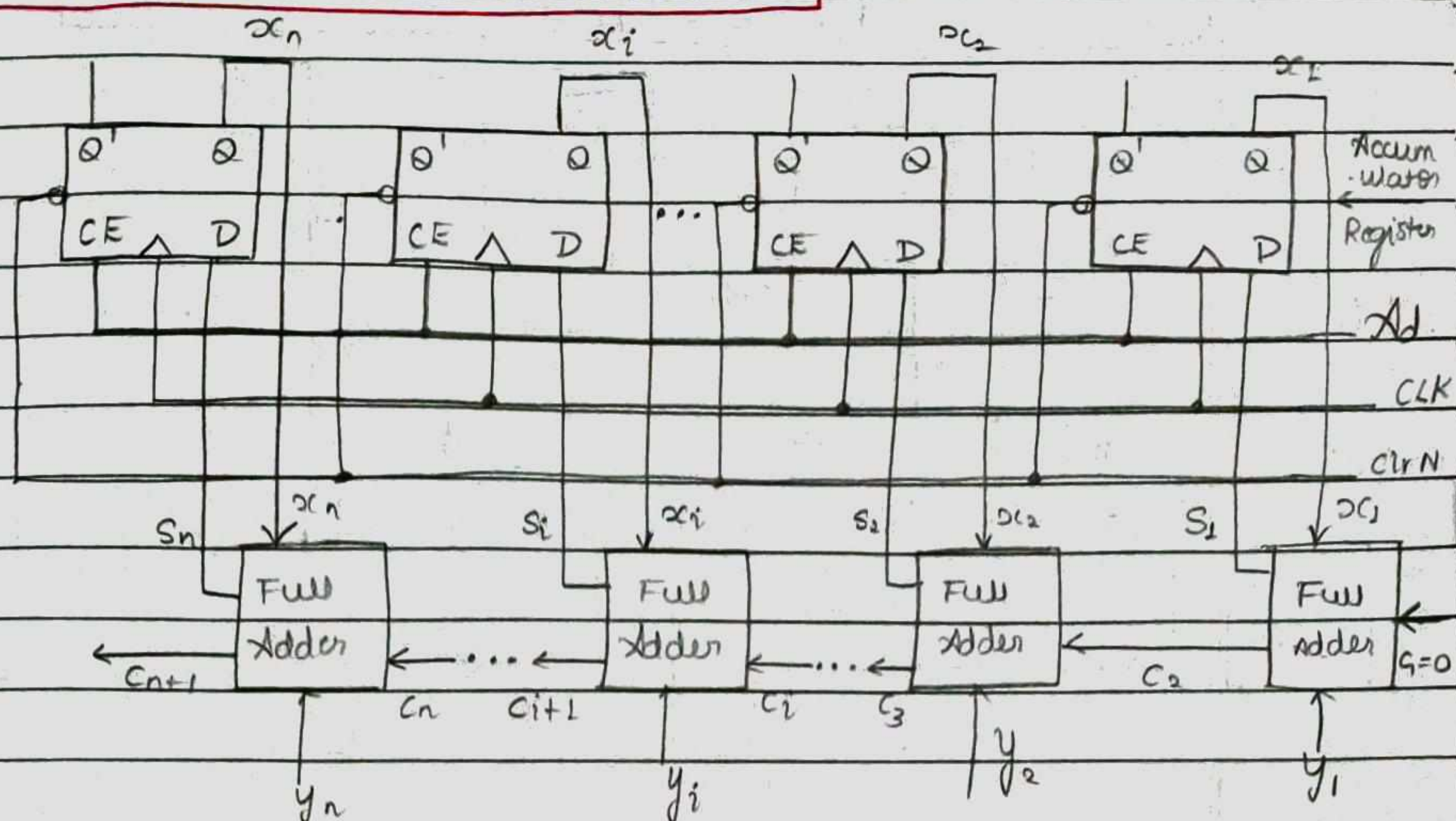


→ The above figure shows how data can be transferred from one of the four 8-bit registers into one of two other registers.

→ The operation can be summarized as follows

- i If $EF = 00$, then 'A' is stored in 'G' or 'H'
- ii If $EF = 01$, then 'B' is stored in 'G' or 'H'
- iii If $EF = 10$, then 'C' is stored in 'G' or 'H'
- iv If $EF = 11$, then 'D' is stored in 'G' or 'H'

Parallel Adder with Accumulator

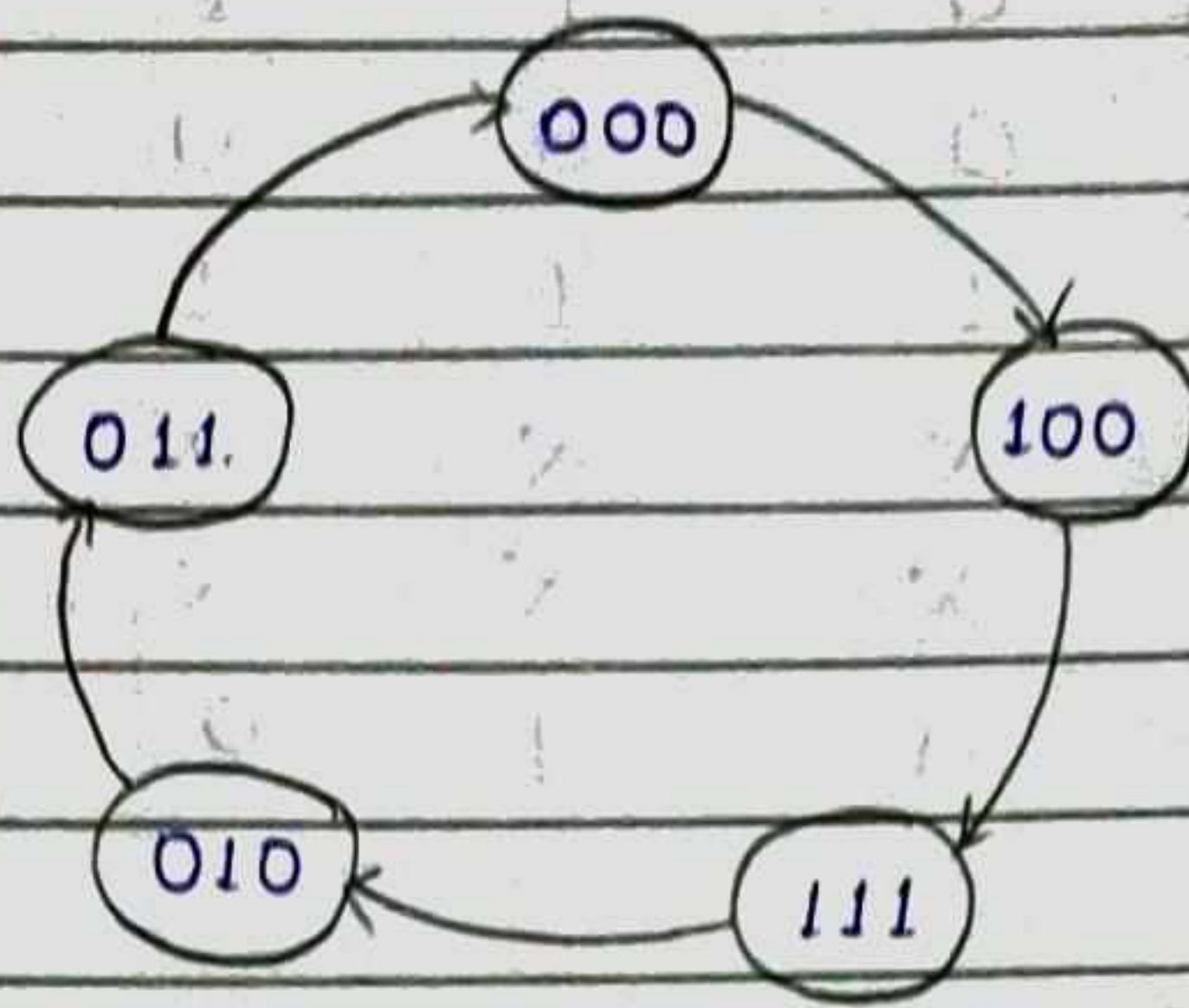


- In computer circuits, it is frequently desirable to store one number in a register of flipflops (called an accumulator) and add a second number to it, leaving the result stored in the accumulator.
- Suppose that the number $X = x_n \dots x_2 x_1$ is stored in the accumulator. Then, the number $Y = y_n \dots y_2 y_1$ is applied to the full adder inputs, and after the carry has propagated through the adders, the sum of X and Y appears at the adder outputs.
- An add signal (Ad) is used to load the adder outputs into the accumulator flip-flops on the rising clock edge.
- If $s_i = 1$, the next state of flip-flop x_i will be 1. If $s_i = 0$, the next state of flip-flop x_i will be 0. Thus, $x_i^+ = s_i$, and if $Ad = 1$, the number X in the accumulator is replaced with the sum of X and Y , following the rising edge of the clock.

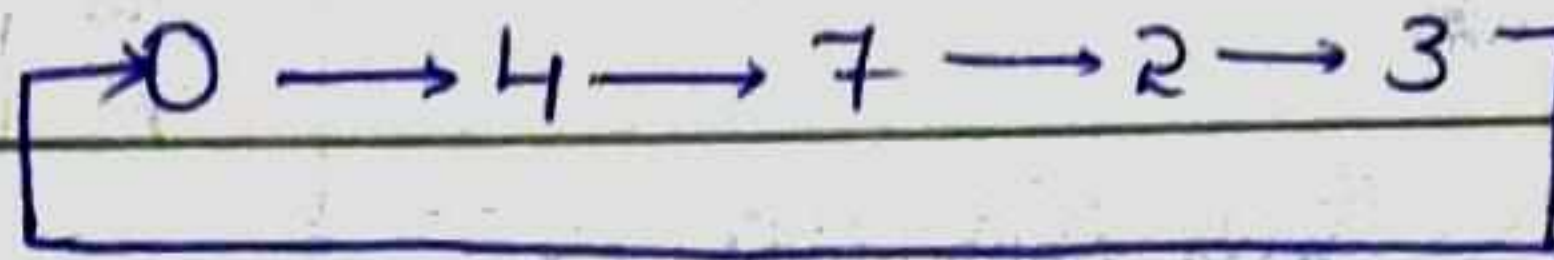
(17)

⇒ Design of irregular counter (counter for other sequence)

⇒ 1. Design an upcounter for given state diagram using T flip-flop.



(07)



i) $T \rightarrow 111 \Rightarrow 3 \text{ FF we need}$

ii) T-FF

iii) Excitation table for T FlipFlop.

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

iv) Next state table.

Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	T_C	T_B	T_A
0	0	0	1	0	0	1	0	0
0	0	1	X	X	X	X	X	X
0	1	0	0	1	1	0	0	1
0	1	1	0	0	0	0	1	1
1	0	0	1	1	1	0	1	1
1	0	1	X	X	X	X	X	X
1	1	0	X	X	X	X	X	X
1	1	1	0	1	0	1	0	1

K-map

Q_C	$Q_B Q_A$			
	$\bar{Q}_B \bar{Q}_A$	$\bar{Q}_B Q_A$	$Q_B \bar{Q}_A$	$Q_B Q_A$
\bar{Q}_C	1	X	0	0
Q_C	0	X	1	X

$$T_C = Q_C' Q_B' + Q_C Q_B$$

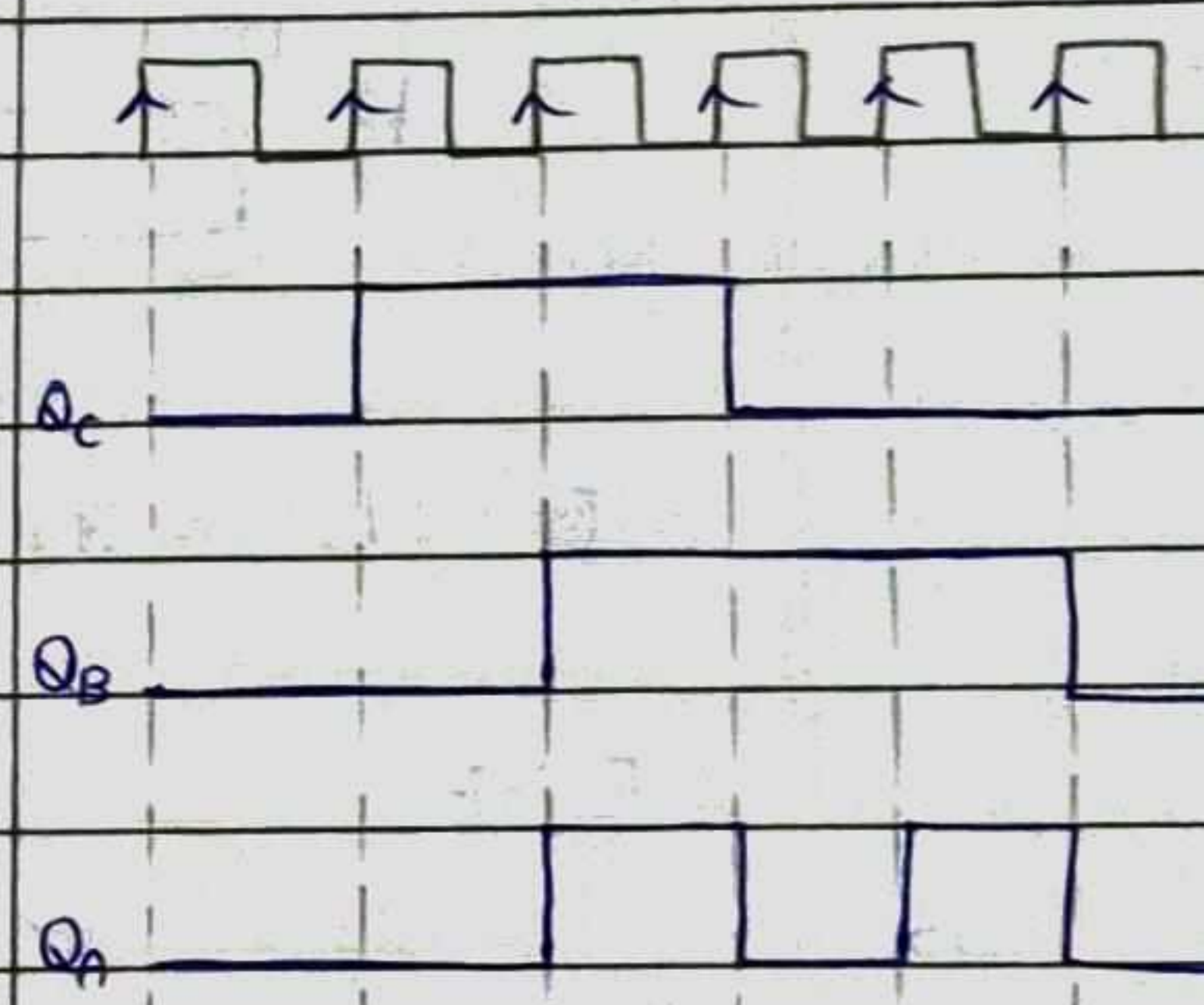
Q_C	$Q_B Q_A$			
	$\bar{Q}_B \bar{Q}_A$	$\bar{Q}_B Q_A$	$Q_B \bar{Q}_A$	$Q_B Q_A$
\bar{Q}_C	0	X	1	0
Q_C	1	X	0	X

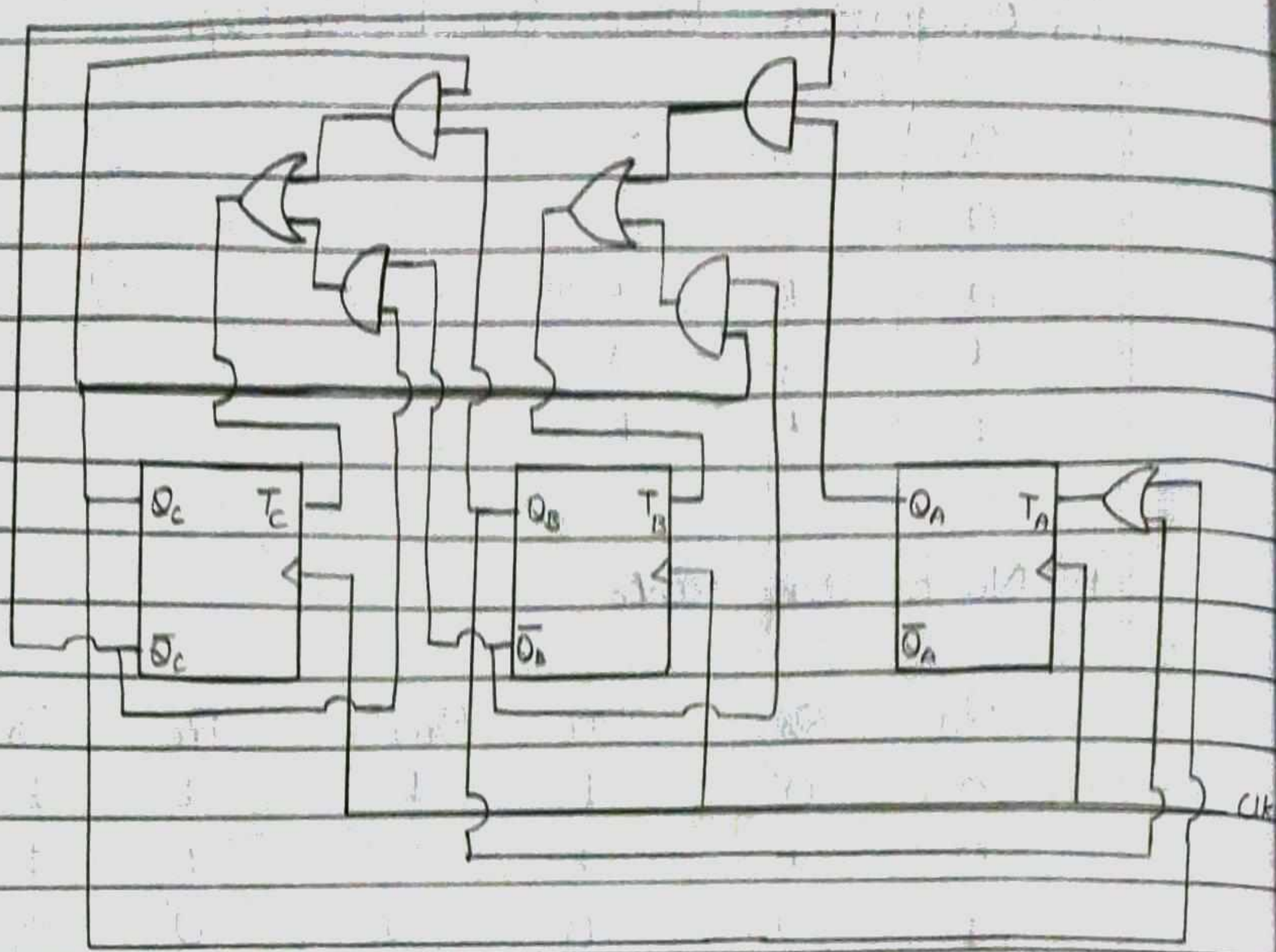
$$T_B = Q_C' Q_A + Q_C Q_B'$$

Q_C	$Q_B Q_A$			
	$\bar{Q}_B \bar{Q}_A$	$\bar{Q}_B Q_A$	$Q_B \bar{Q}_A$	$Q_B Q_A$
\bar{Q}_C	0	X	1	1
Q_C	1	X	1	X

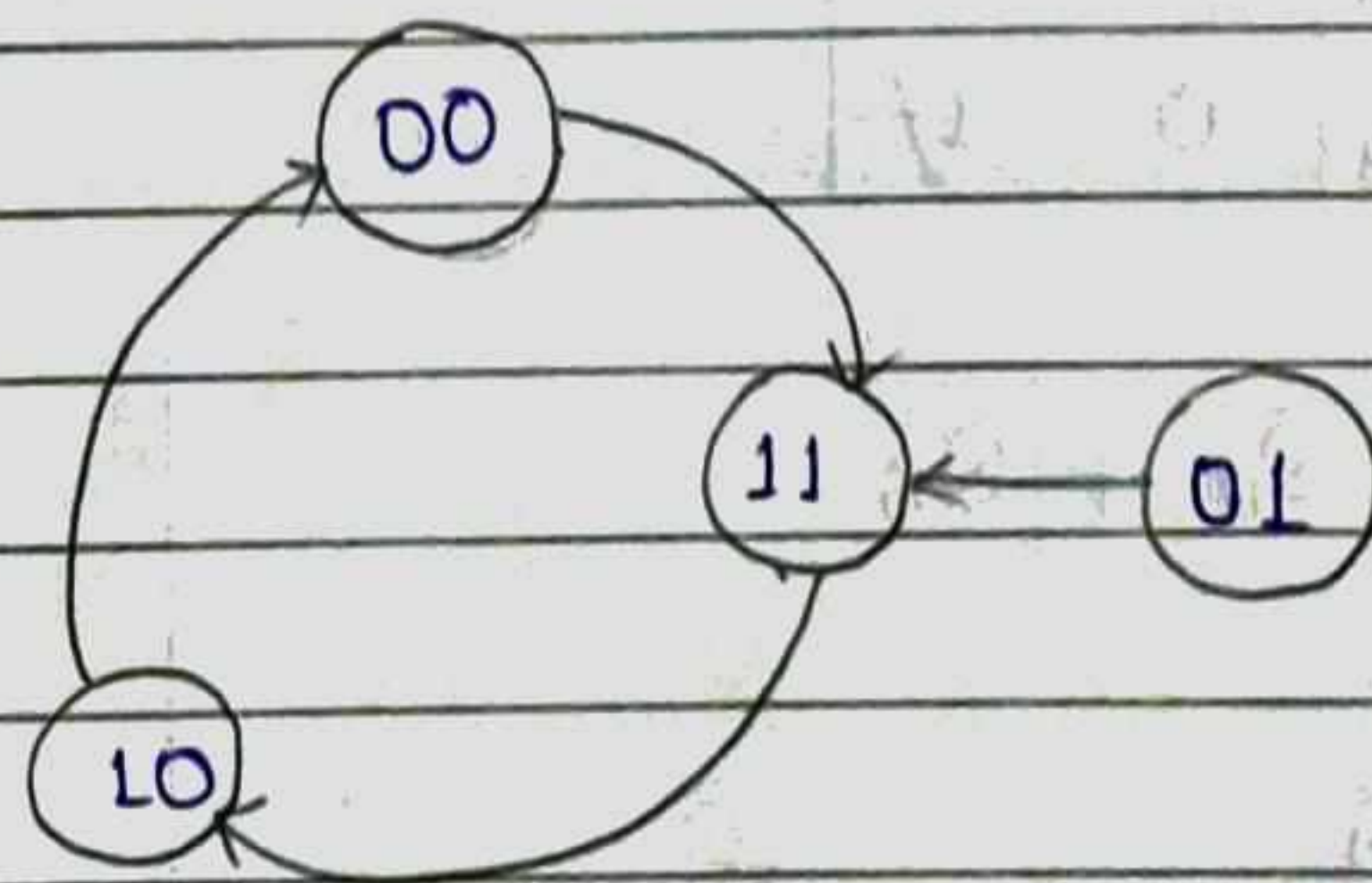
$$T_A = Q_C + Q_B$$

Timing diagram





2. Design an upcounter for given state diagram
using D flip-flop.



i} $D \rightarrow 10 \Rightarrow 2FF$

ii} D-FF

iii) Excitation table for D-Flip Flop.

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

iv) Next state table

Q_B	Q_A	Q_{B+1}	Q_{A+1}	D_B	D_A
0	0	1	1	1	1
0	1	1	1	1	1
1	0	0	0	0	0
1	1	1	0	1	0

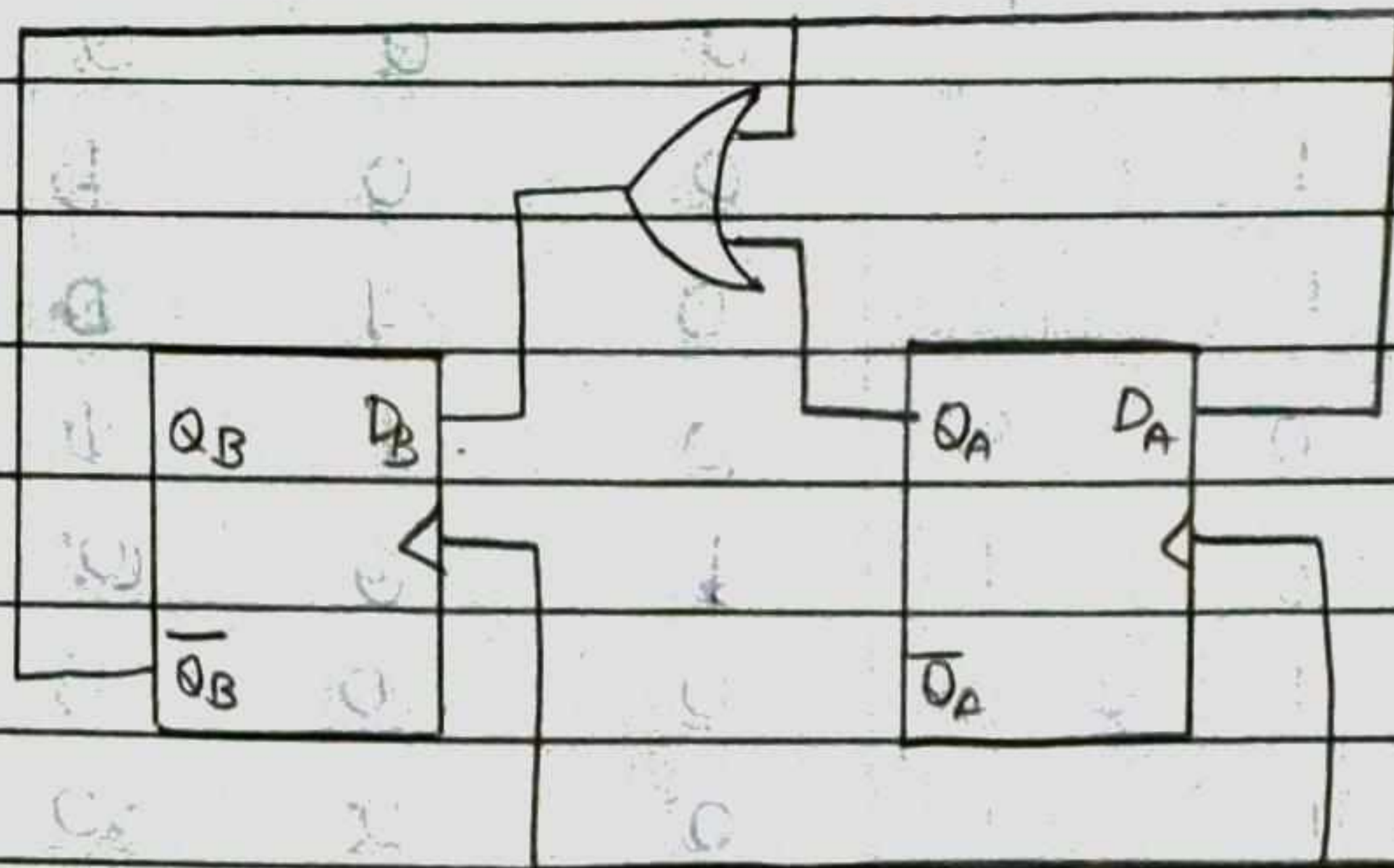
K-map-

		Q_A	\overline{Q}_A
Q_B	\overline{Q}_B	1	1
	Q_B	0	1

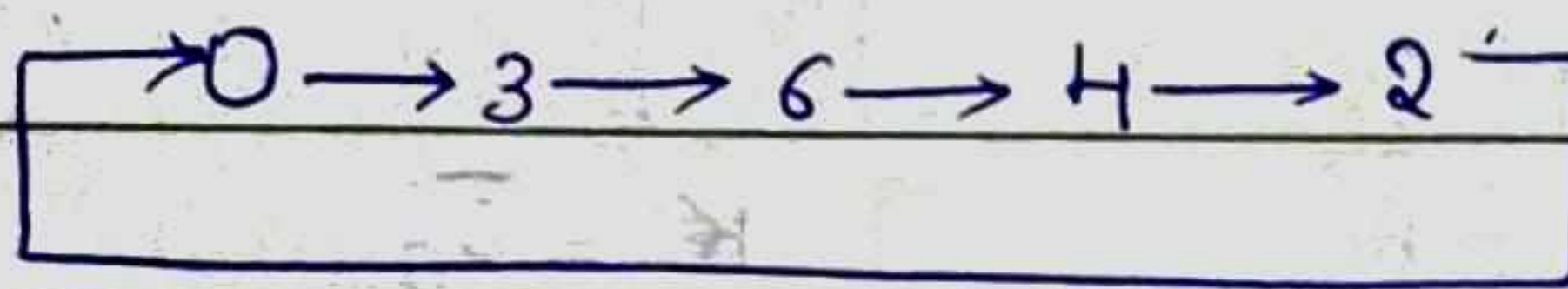
$$D_B = \overline{Q}_B + Q_A$$

		Q_A	\overline{Q}_A
Q_B	\overline{Q}_B	1	1
	Q_B	0	0

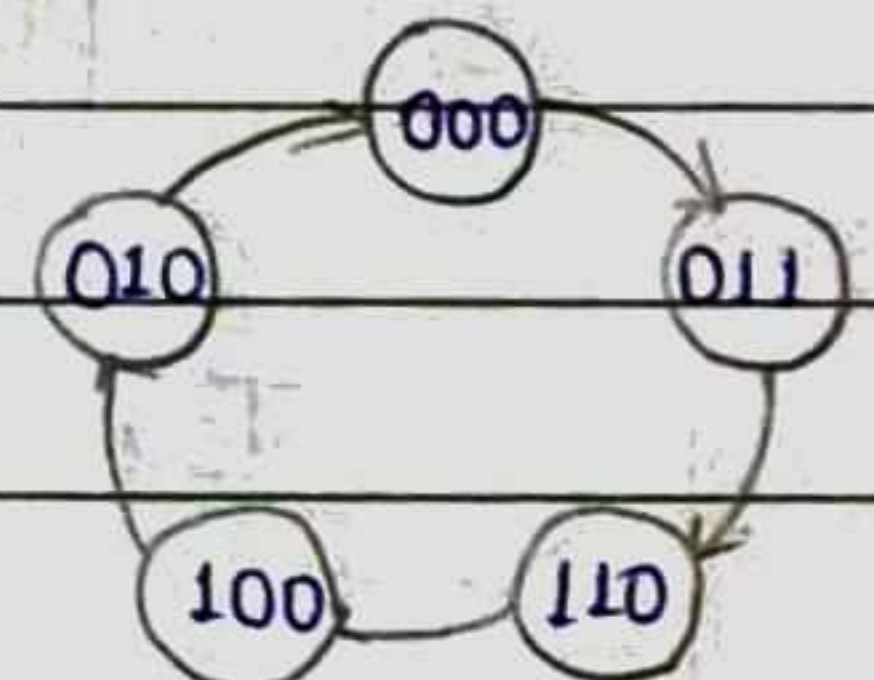
$$D_A = \overline{Q}_B$$



⇒ Design irregular counter using JK Flip Flop for the following sequence.



or



i) JK - 110 - 3FF

ii) JK Flip-Flop.

iii) Excitation table for JK Flip Flop.

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

Next state table

Q_c	Q_B	Q_A	Q_{c+1}	Q_{B+1}	Q_{A+1}	J_c	K_c	J_B	K_B	J_A	K_A
0	0	0	0	1	1	0	x	1	x	1	x
0	0	1	x	x	x	x	x	x	x	x	x
0	1	0	0	0	0	0	x	x	1	0	x
0	1	1	1	1	0	1	x	x	0	x	1
1	0	0	0	1	0	x	1	1	x	0	x
1	0	1	x	x	x	x	x	x	x	x	x
1	1	0	1	0	0	x	0	x	1	0	x
1	1	1	x	x	x	x	x	x	x	x	x

K-Map

$Q_c \backslash Q_B Q_A$	$\bar{Q}_B \bar{Q}_A$	$\bar{Q}_B Q_A$	$Q_B \bar{Q}_A$	$Q_B Q_A$
\bar{Q}_c	0	x	1	0
Q_c	x	x	x	x

$$J_c = Q_A$$

$Q_c \backslash Q_B Q_A$	$\bar{Q}_B \bar{Q}_A$	$\bar{Q}_B Q_A$	$Q_B \bar{Q}_A$	$Q_B Q_A$
\bar{Q}_c	x	x	x	x
Q_c	1	x	x	0

$$K_c = \bar{Q}_B$$

$Q_c \backslash Q_B Q_A$	$\bar{Q}_B \bar{Q}_A$	$\bar{Q}_B Q_A$	$Q_B \bar{Q}_A$	$Q_B Q_A$
\bar{Q}_c	1	x	x	x
Q_c	1	x	x	x

$$J_B = 1$$

$Q_c \backslash Q_B Q_A$	$\bar{Q}_B \bar{Q}_A$	$\bar{Q}_B Q_A$	$Q_B \bar{Q}_A$	$Q_B Q_A$
\bar{Q}_c	x	x	0	1
Q_c	x	x	x	1

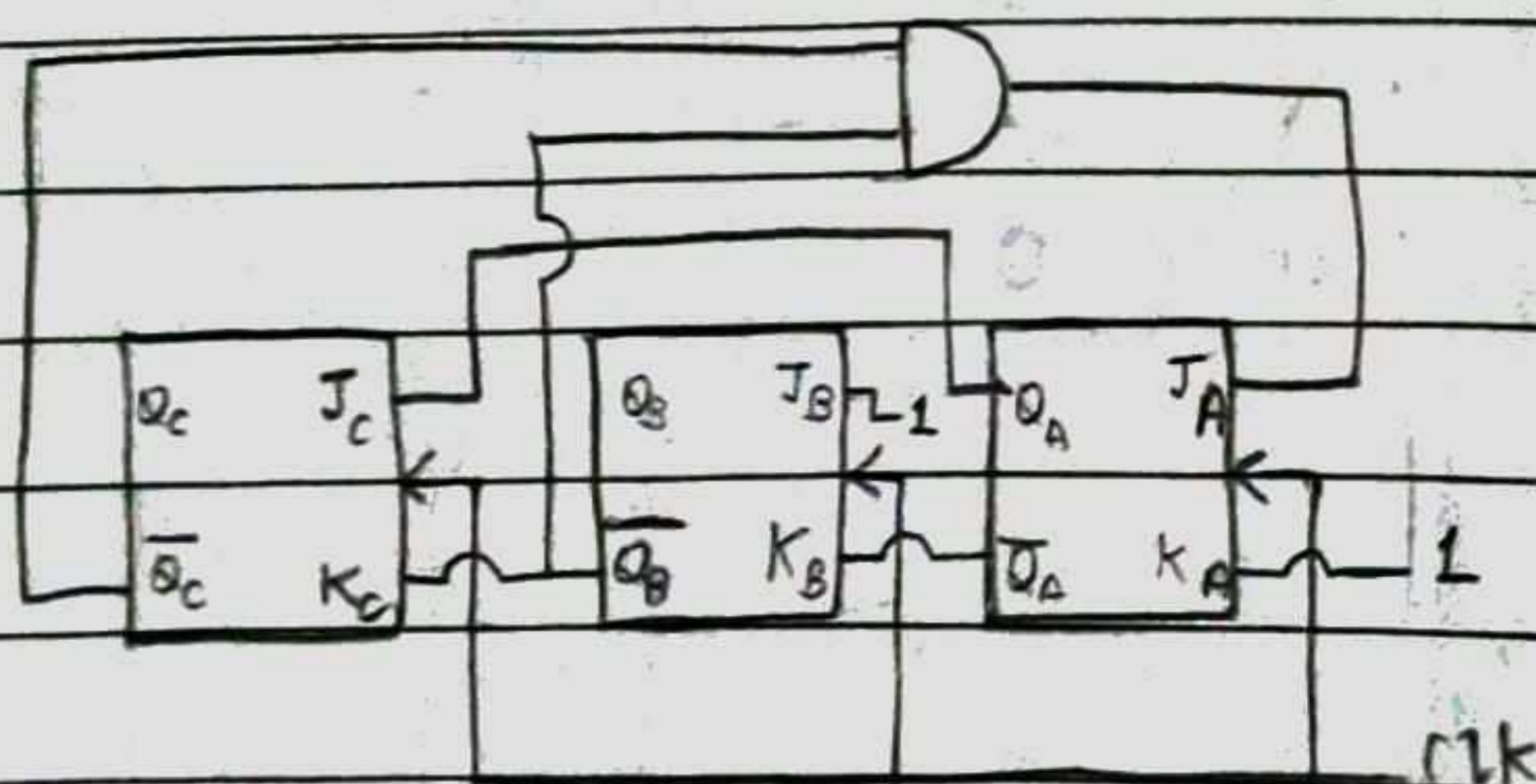
$$K_B = \bar{Q}_A$$

$Q_c \backslash Q_B Q_A$	$\bar{Q}_B \bar{Q}_A$	$\bar{Q}_B Q_A$	$Q_B \bar{Q}_A$	$Q_B Q_A$
\bar{Q}_c	1	x	x	0
Q_c	0	x	x	0

$$J_A = \bar{Q}_c \bar{Q}_B$$

$Q_c \backslash Q_B Q_A$	$\bar{Q}_B \bar{Q}_A$	$\bar{Q}_B Q_A$	$Q_B \bar{Q}_A$	$Q_B Q_A$
\bar{Q}_c	x	x	x	x
Q_c	1	x	1	x

$$K_A = 1$$

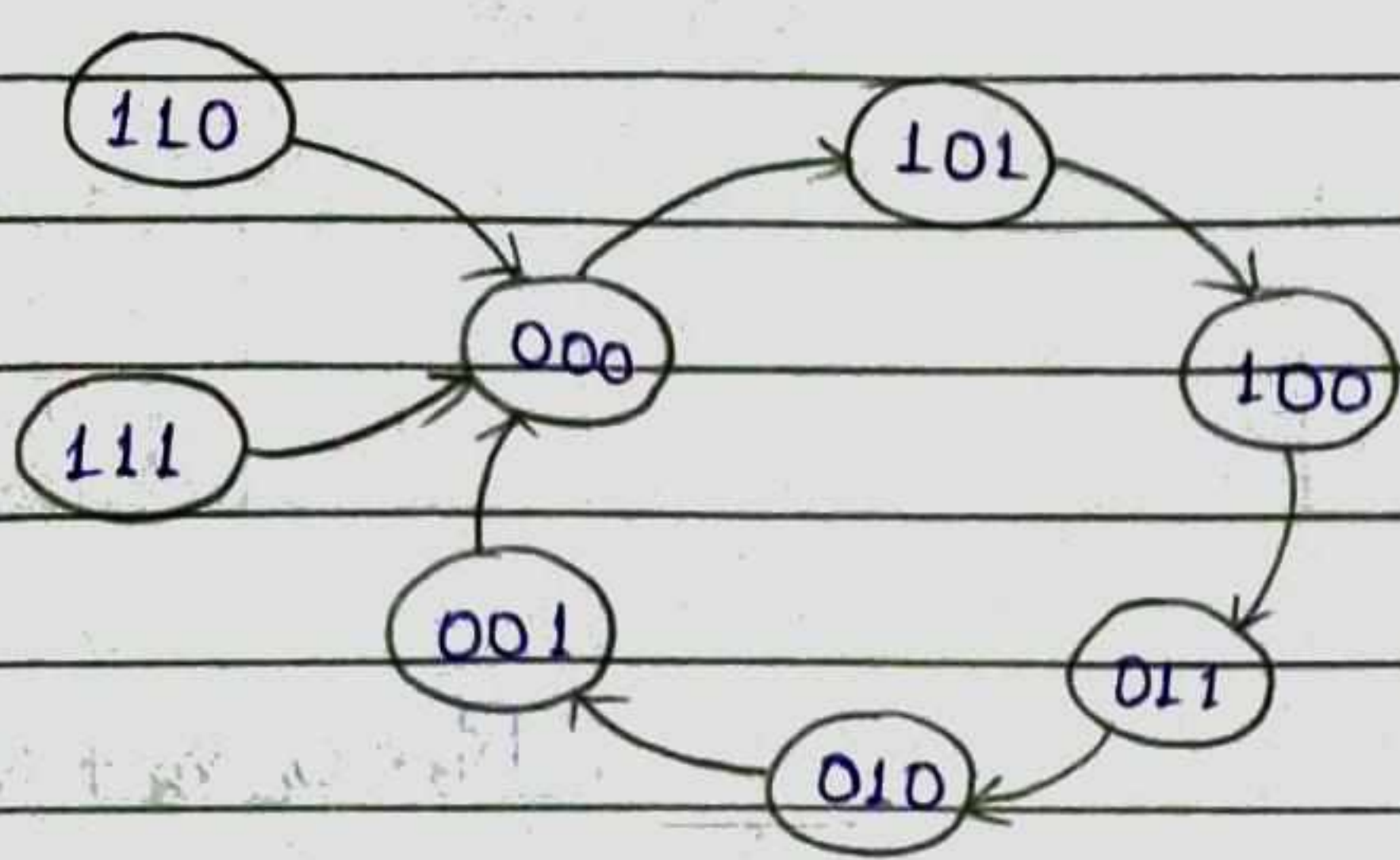


→ Design Mod 6 downcounter using SR flipflop
where all unused state must lead to 000

5 → 0

101 → 000

3 SR Flip Flops.



Excitation table for SR Flip Flop

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

Next state table.

Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	S_C	R_C	S_B	R_B	S_A	R_A
0	0	0	1	0	1	1	0	0	X	1	0
0	0	1	0	0	0	0	X	0	X	0	1
0	1	0	0	0	1	0	X	0	1	1	0
0	1	1	0	1	0	0	X	X	0	0	1
1	0	0	0	1	1	0	1	1	0	1	0
1	0	1	1	0	0	X	0	0	X	0	1
1	1	0	0	0	0	0	1	0	1	0	X
1	1	1	0	0	0	0	1	0	1	0	1

Q_C	$Q_B Q_A$	$\bar{Q}_B \bar{Q}_A$	$\bar{Q}_B Q_A$	$Q_B \bar{Q}_A$
\bar{Q}_C	1	0	0	0
Q_C	0	X	0	0

$$S_C = \bar{Q}_A \bar{Q}_B \bar{Q}_C$$

Q_C	$Q_B Q_A$	$\bar{Q}_B \bar{Q}_A$	$\bar{Q}_B Q_A$	$Q_B \bar{Q}_A$
\bar{Q}_C	0	X	X	X
Q_C	1	0	1	1

$$R_C = Q_B + Q_C \bar{Q}_A$$

Q_C	$Q_B Q_A$	$\bar{Q}_B \bar{Q}_A$	$\bar{Q}_B Q_A$	$Q_B \bar{Q}_A$
\bar{Q}_C	0	0	X	0
Q_C	1	0	0	0

$$S_B = Q_C \bar{Q}_B \bar{Q}_A$$

Q_C	$Q_B Q_A$	$\bar{Q}_B \bar{Q}_A$	$\bar{Q}_B Q_A$	$Q_B \bar{Q}_A$
\bar{Q}_C	X	X	0	1
Q_C	0	X	1	1

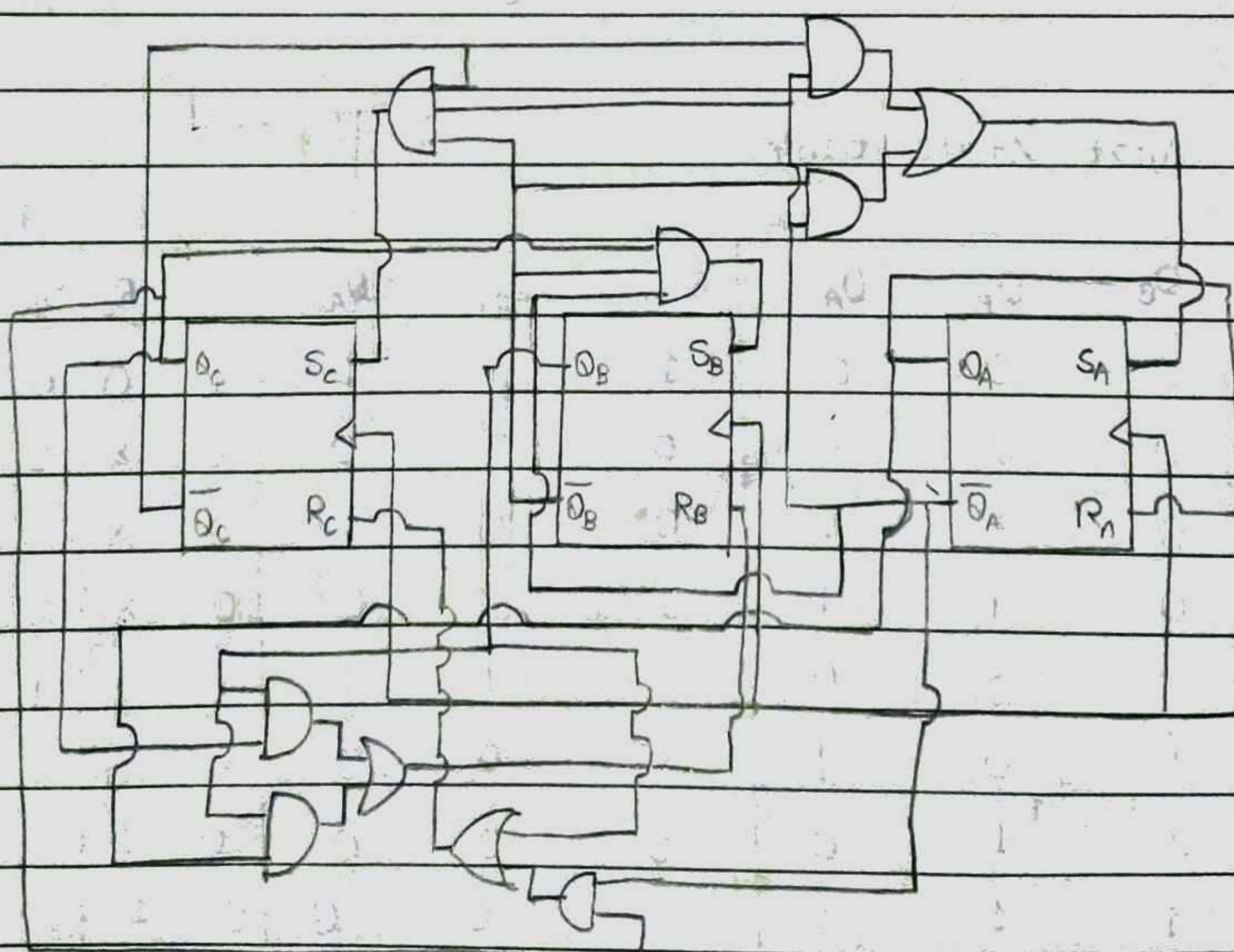
$$R_B = Q_C Q_B + Q_B \bar{Q}_A$$

Q_C	$Q_B Q_A$	$\bar{Q}_B \bar{Q}_A$	$\bar{Q}_B Q_A$	$Q_B \bar{Q}_A$
\bar{Q}_C	1	0	0	1
Q_C	1	0	0	0

$$S_A = \bar{Q}_B \bar{Q}_A + \bar{Q}_C \bar{Q}_A$$

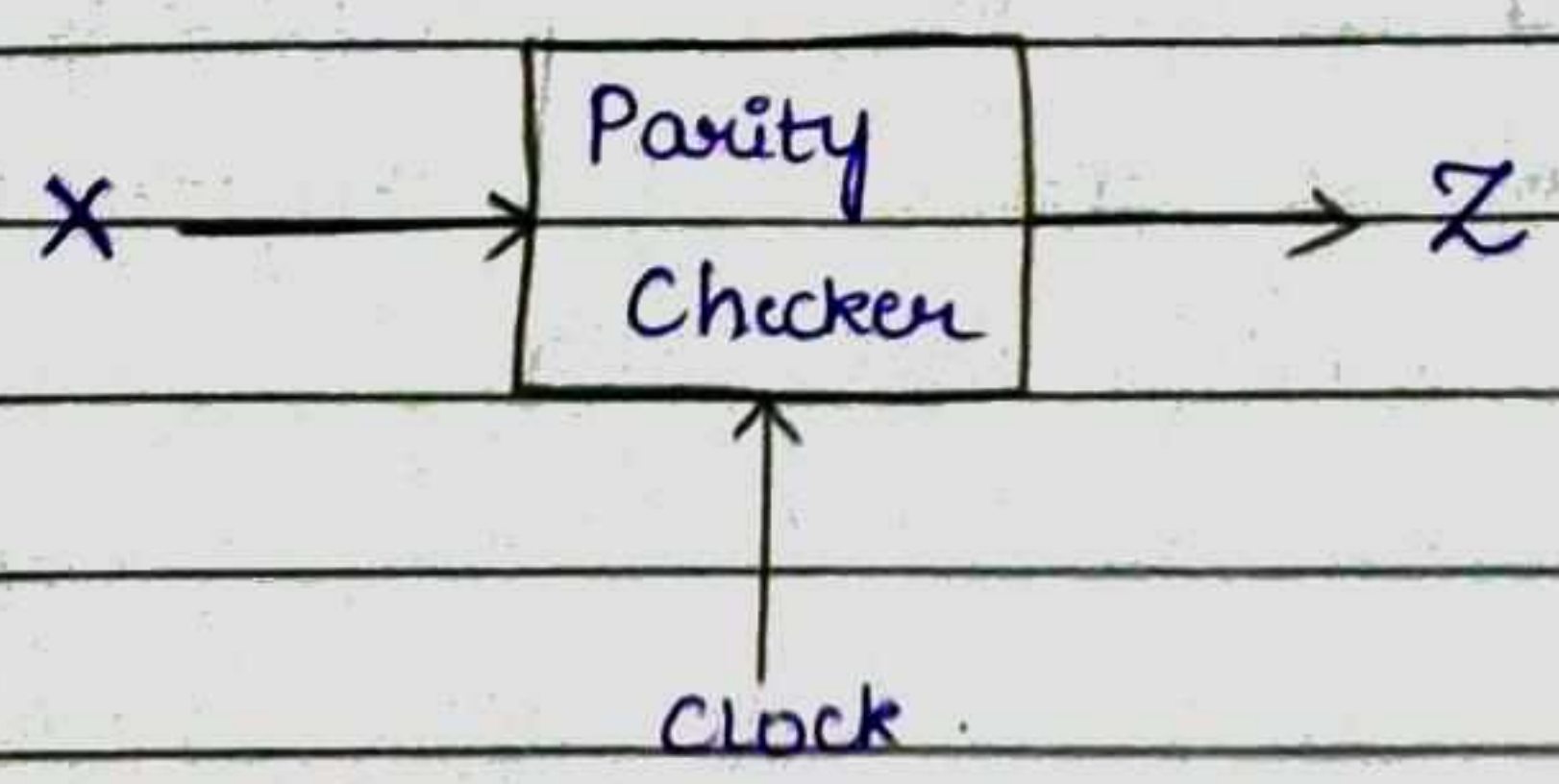
Q_C	$Q_B Q_A$	$\bar{Q}_B \bar{Q}_A$	$\bar{Q}_B Q_A$	$Q_B \bar{Q}_A$
\bar{Q}_C	0	1	1	0
Q_C	0	1	1	X

$$R_A = Q_A$$

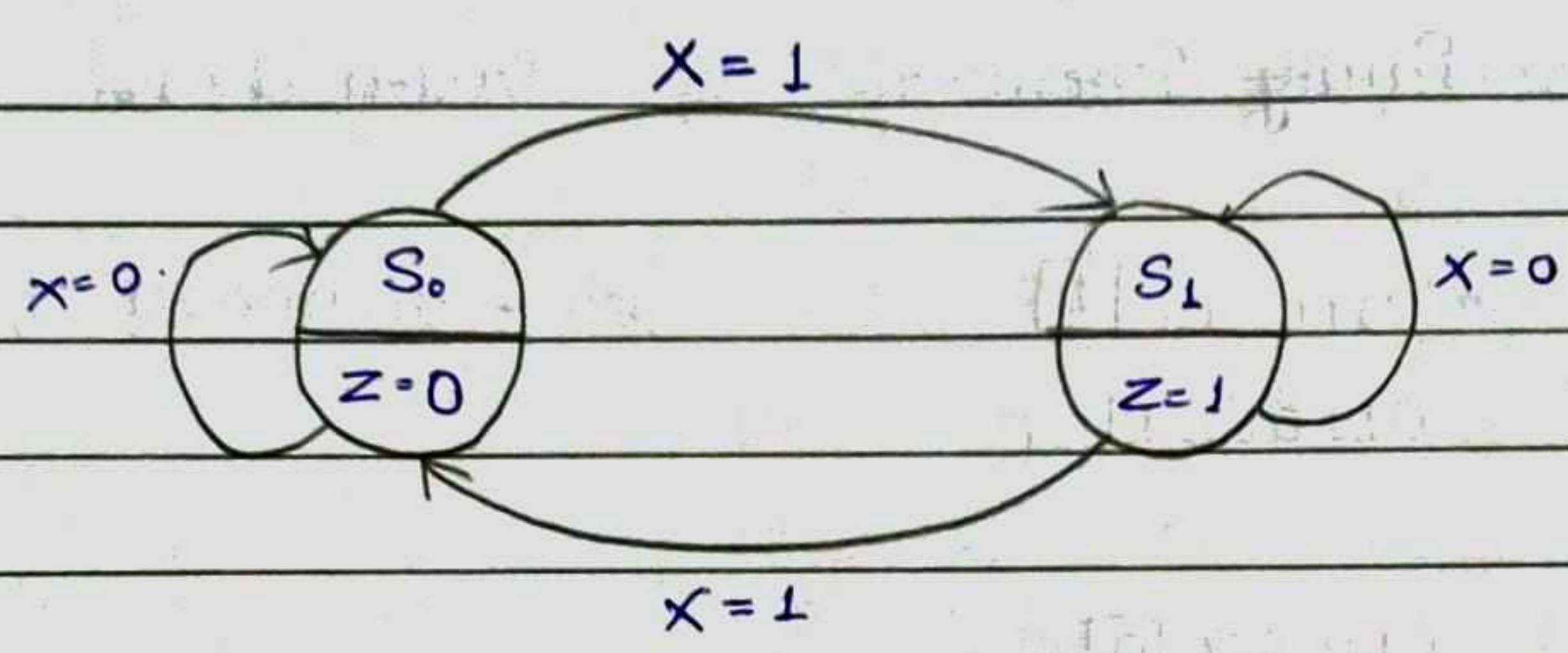


⇒ A sequential parity checker

i) Block diagram for parity checker.



ii) State Graph for parity checker.



iii) State & Transition Table.

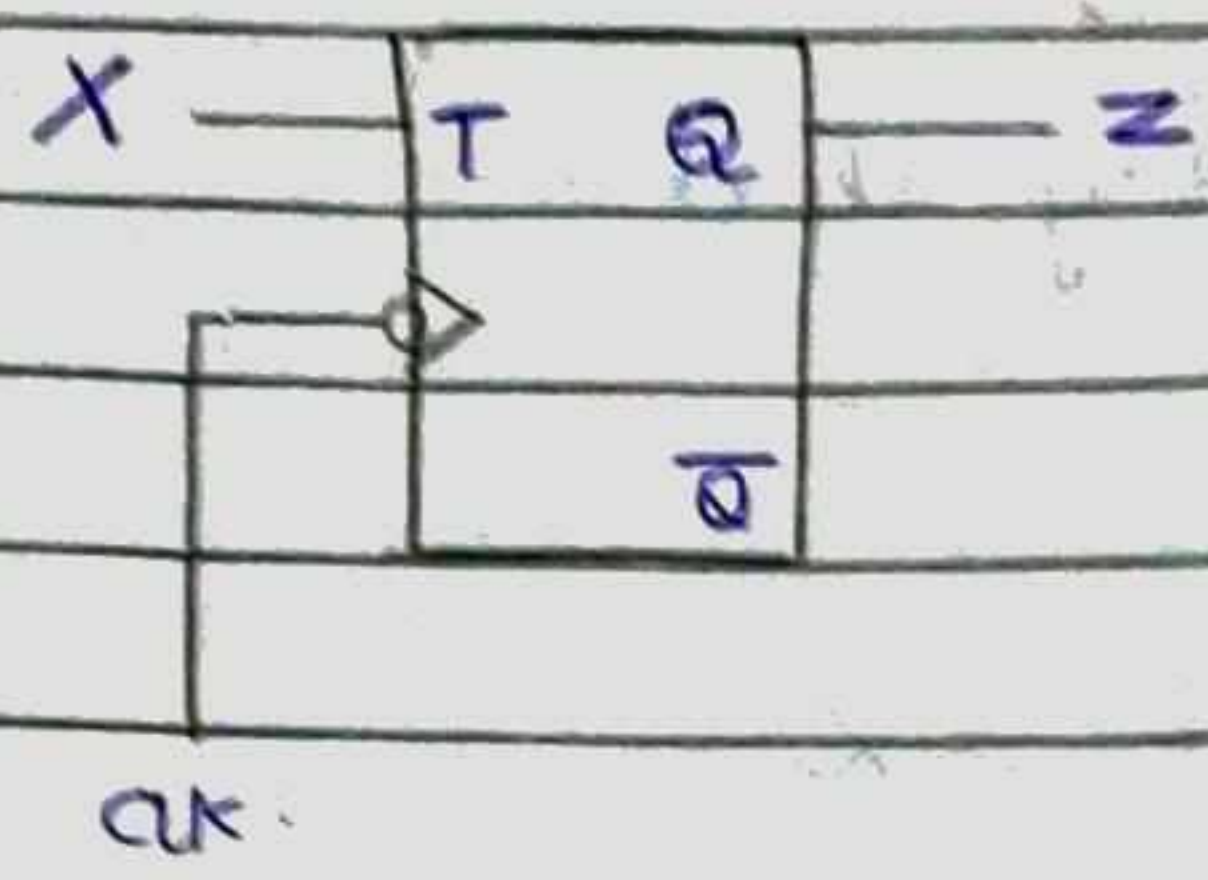
a) State table

Present state	Next state		Present o/p
	X=0	X=1	
S ₀	S ₀	S ₁	0
S ₁	S ₁	S ₀	1

b) Transition table using T flip flop

Q	Q ⁺		T	
	X=0	X=1	X=0	X=1
0	0	1	0	1
1	1	0	0	1

iv sequential parity checker using T FlipFlop



Parity - No of 1's

- Odd parity → 0110001 → odd no of 1's
- Even parity → 0111100 → Even no of 1's

Parity Generator

Parity checker

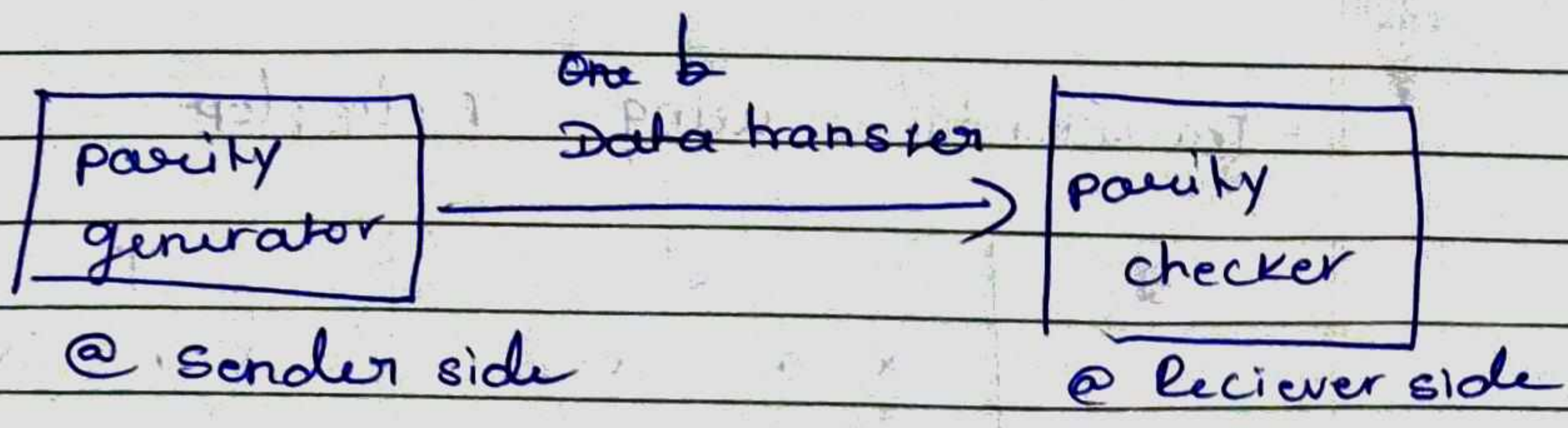
ODD 0110000 1
 0110001 0

ODD → 01100000

EVEN 0110000 0
 0110001 1

Application of parity checker and generator.

→ one bit error detection in data transmission



⇒ A Sequential Parity Checker

- When binary data is transmitted or stored, an extra bit (called a parity bit) is frequently added for purposes of error detection.
- For example, if data is being transmitted in groups of 7 bits, an eighth bit can be added to each group of 7 bits to make the total number of 1's in each block of 8 bits an odd number.
- When the total number of 1 bits in the block (including the parity bit) is odd, we say that the parity is odd.
- Alternately, the parity bit could be chosen such that the total number of 1's in the block is even, in which case we would have even parity.

7 Data Bits							Parity Bits
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	1	1	0	1	1	0	1
1	0	1	0	1	0	1	1
0	1	1	1	0	0	0	0
8-bit word.							

- If any single bit in the 8-bit word is changed from 0 to 1 or from 1 to 0, the parity is no longer odd.
- Thus, if any single bit error occurs in transmission of a word with odd parity, the presence of this error can be detected because the number of 1 bits in the word has been changed from odd to even.

⇒ State Graph for Parity checker [Explanation]

- We will start the design by constructing a state graph.
- The sequential circuit must "remember" whether the total number of 1 inputs received is even or odd, therefore, only two states are required.
- We will designate these states as S_0 and S_1 , corresponding respectively to an even number of 1's received and an odd number of 1's received.
- We will start the circuit in state S_0 because initially zero 1's have been received, and zero is an even number.
- As indicated in the figure, if the circuit is in state S_0 (even number of 1's received) and $X=0$ is received, the circuit must stay in S_0 because the number of 1's received is still even. However, if $X=1$ is received, the circuit goes to state S_1 because the number of 1's received is then odd.
- Similarly, if the circuit is in state S_1 (odd number of 1's received) a 0 input causes no state change, but a 1 causes a change to S_0 because the number of 1's received is then even.
- The output Z should be 1 whenever the circuit is in state S_1 (odd number of 1's received). The output is listed below the state on the state graph.

State Tables and Graphs

→ Rules

The following method can be used to construct the transition table:

1. Determine the flip-flop input equations and the output equations from the circuit.
2. Derive the next-state equation for each flip-flop from its input equations, using one of the following relations:

D flip-flop $Q^+ = D$

D-CE flip-flop $Q^+ = D \cdot CE + Q \cdot CE'$

T flip-flop $Q^+ = T \oplus Q$

S-R flip-flop $Q^+ = S + R'Q$

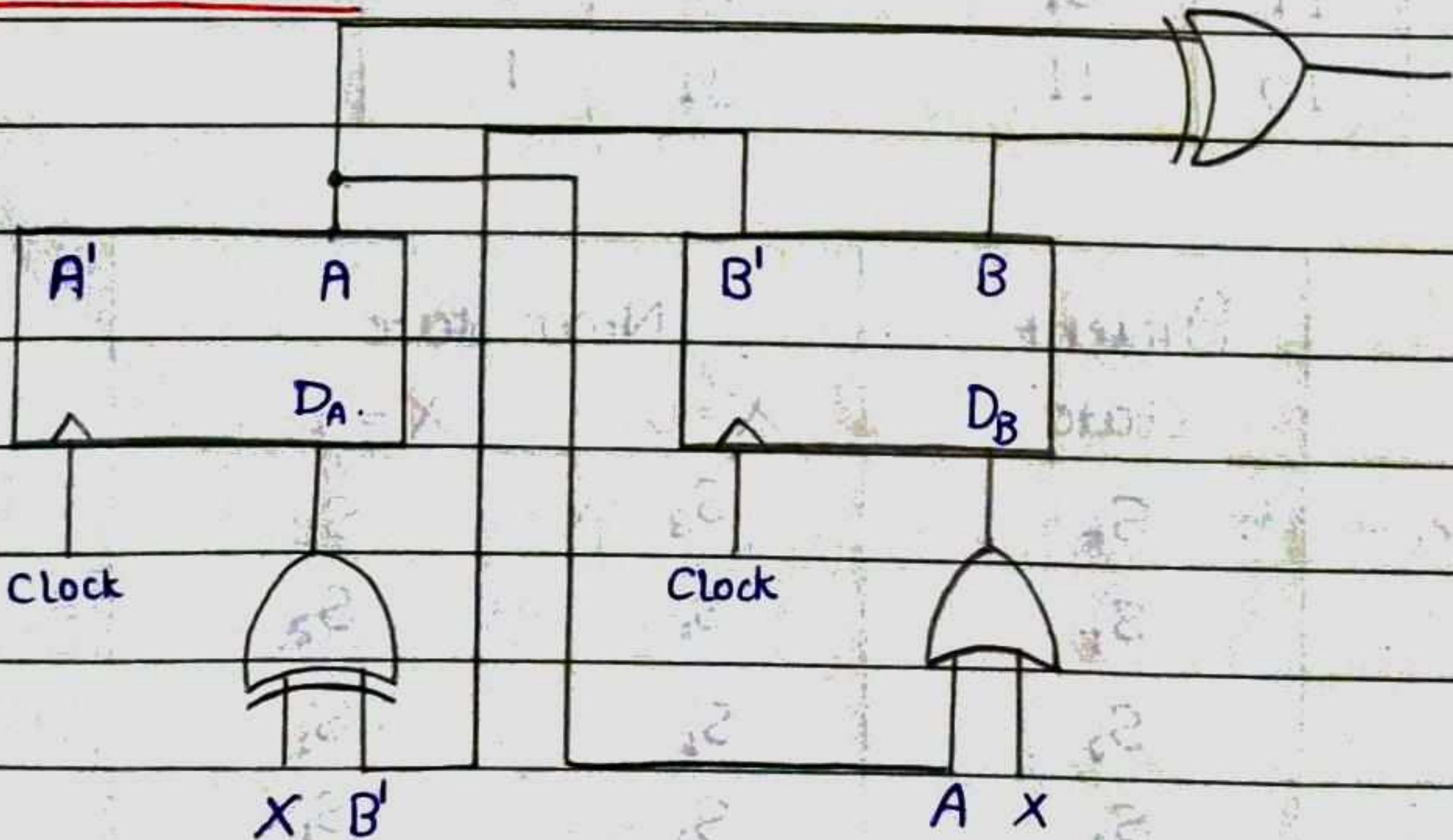
J-K flip-flop $Q^+ = JQ' + K'Q$

3. Plot a next-state map for each flip-flop.
4. Combine these maps to form the transition table.

Such a transition table, which gives the next state of the flip-flops as a function of their present state and the circuit inputs.

Example

1. Moore State Machine



1) The flip-flop input equations and output equation are

$$DA = X \oplus B' \quad DB = X + A \quad Z = A \oplus B$$

2) The next-state equations for the flip-flops are

$$A^+ = X \oplus B' \quad B^+ = X + A$$

3) The corresponding maps are

AB \ X	X	
	0	1
00	1	0
01	0	1
11	0	1
10	1	0

A^+

AB \ X	X	
	0	1
00	0	1
01	0	1
11	1	1
10	1	1

B^+

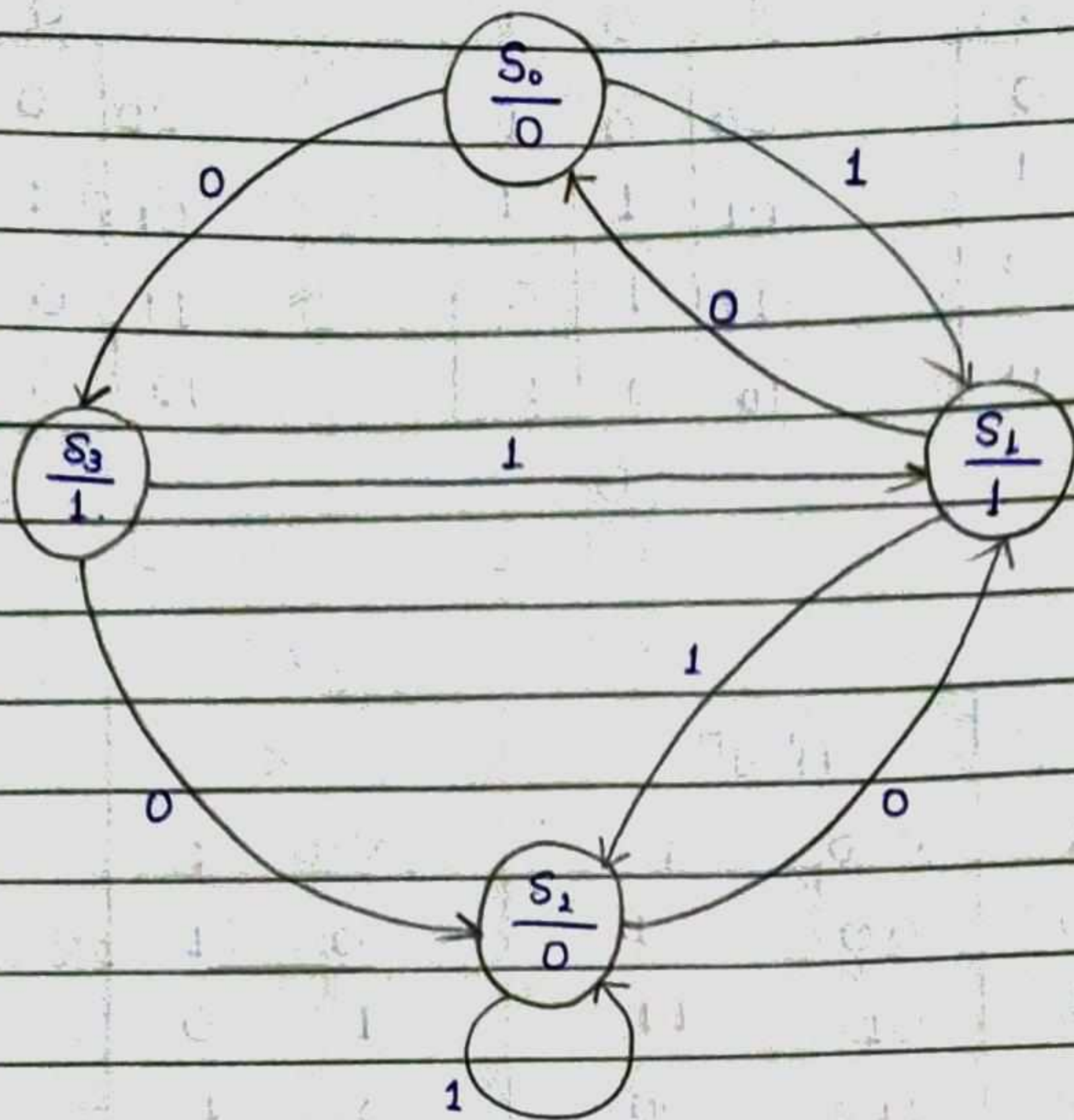
(a)

AB	$A^+ B^+$		Z
	X=0	X=1	
00	10	01	0
01	00	11	1
11	01	11	0
10	11	01	1

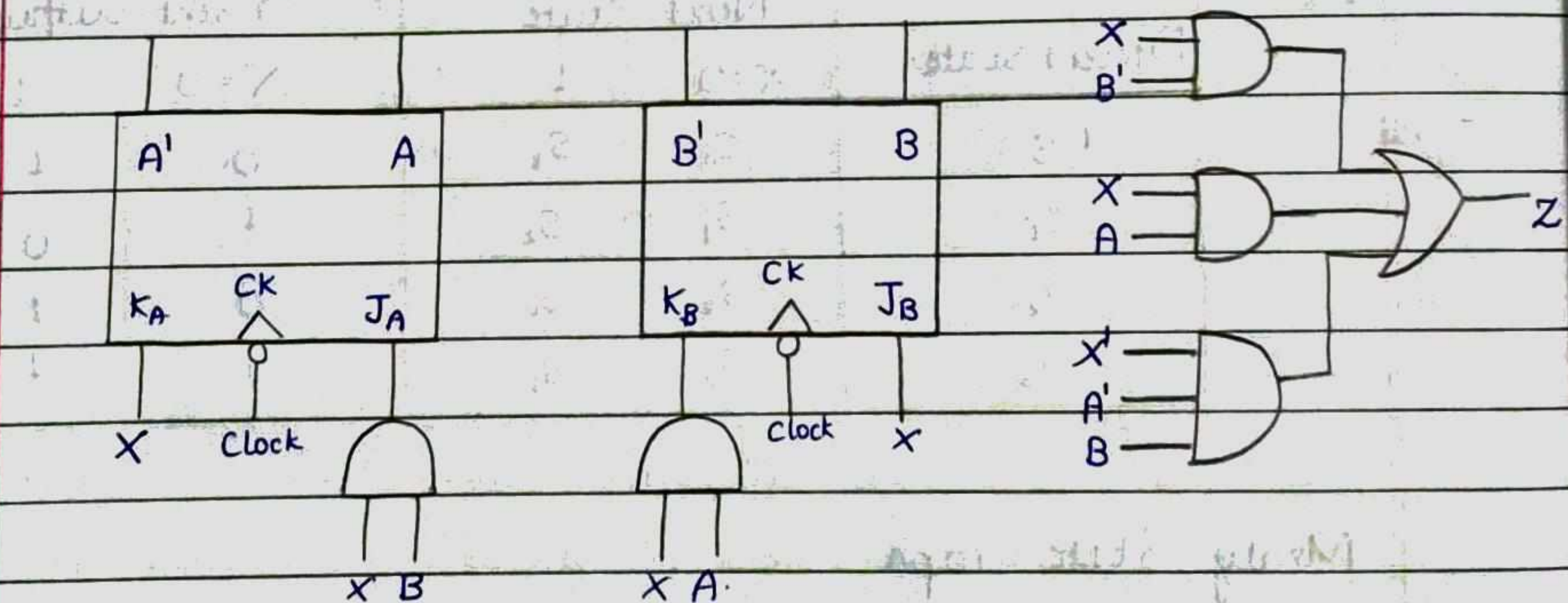
(b)

Present State	Next State		Present Output (Z)
	X=0	X=1	
S_0	S_3	S_1	0
S_1	S_0	S_2	1
S_2	S_1	S_3	0
S_3	S_2	S_0	1

Moore State Graph



2. Mealy State Machine.



$$A^+ = J_A A' + K_A A = XBA' + X'A$$

$$B^+ = J_B B' + K_B B = XB' + (AX)'B = XB' + X'B + A'B$$

$$Z = X'A'B + XB' + XA$$

AB \ X	X	
	0	1
00	0	0
01	0	1
11	1	0
10	1	0

 A^+

AB \ X	X	
	0	1
00	0	1
01	1	1
11	1	0
10	0	1

 B^+

AB \ X	X	
	0	1
00	0	1
01	1	0
11	0	1
10	0	1

Z

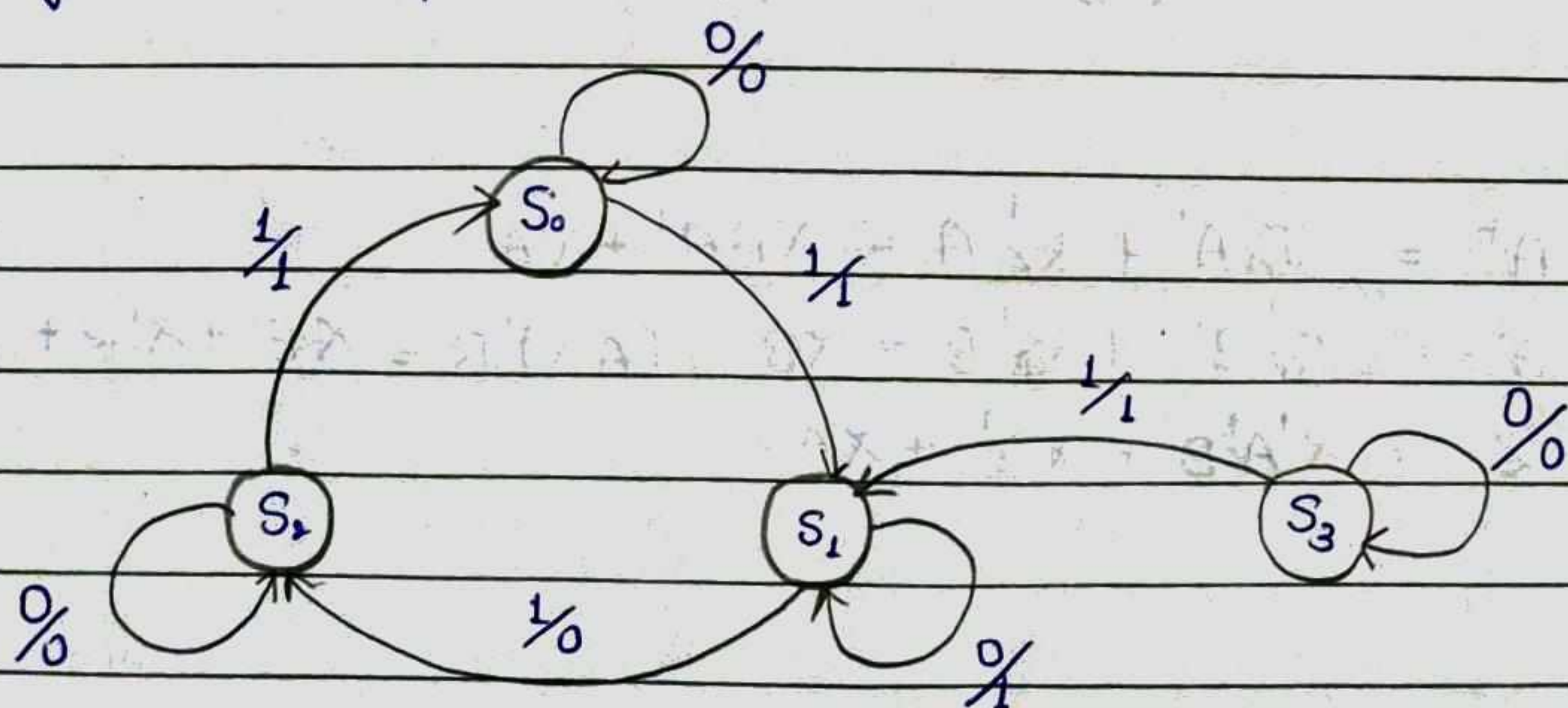
(a)

AB	$A^+ B^+$		Z	
	X=0	1	X=0	1
00	00	01	0	1
01	01	11	1	0
11	11	00	0	1
10	10	01	0	1

(b)

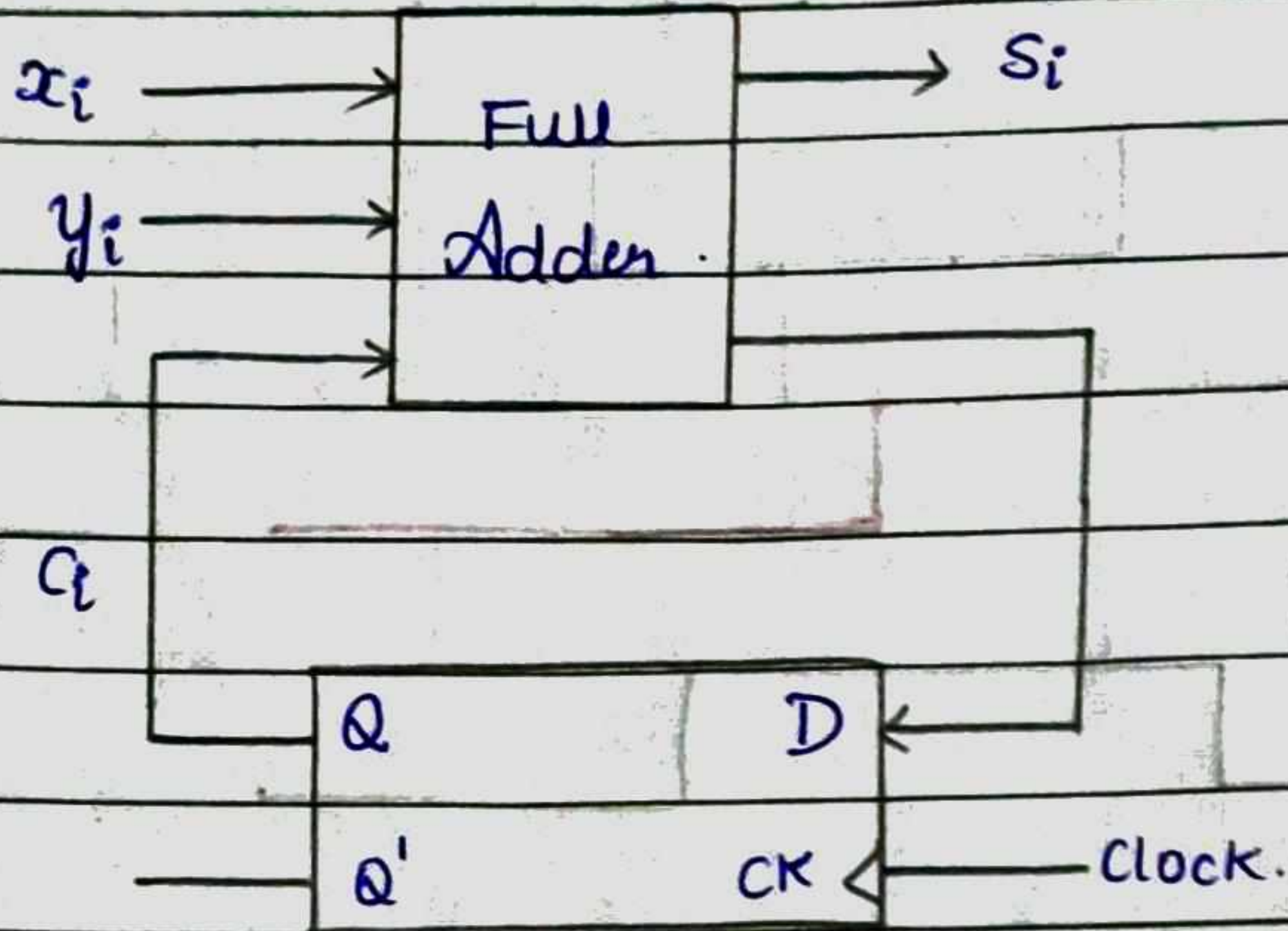
Present State	Next State		Present output	
	X=0	1	X=0	1
S_0	S_0	S_1	0	1
S_1	S_1	S_2	1	0
S_2	S_2	S_0	0	1
S_3	S_3	S_1	0	1

Mealy State Graph.



⇒ Serial Adder

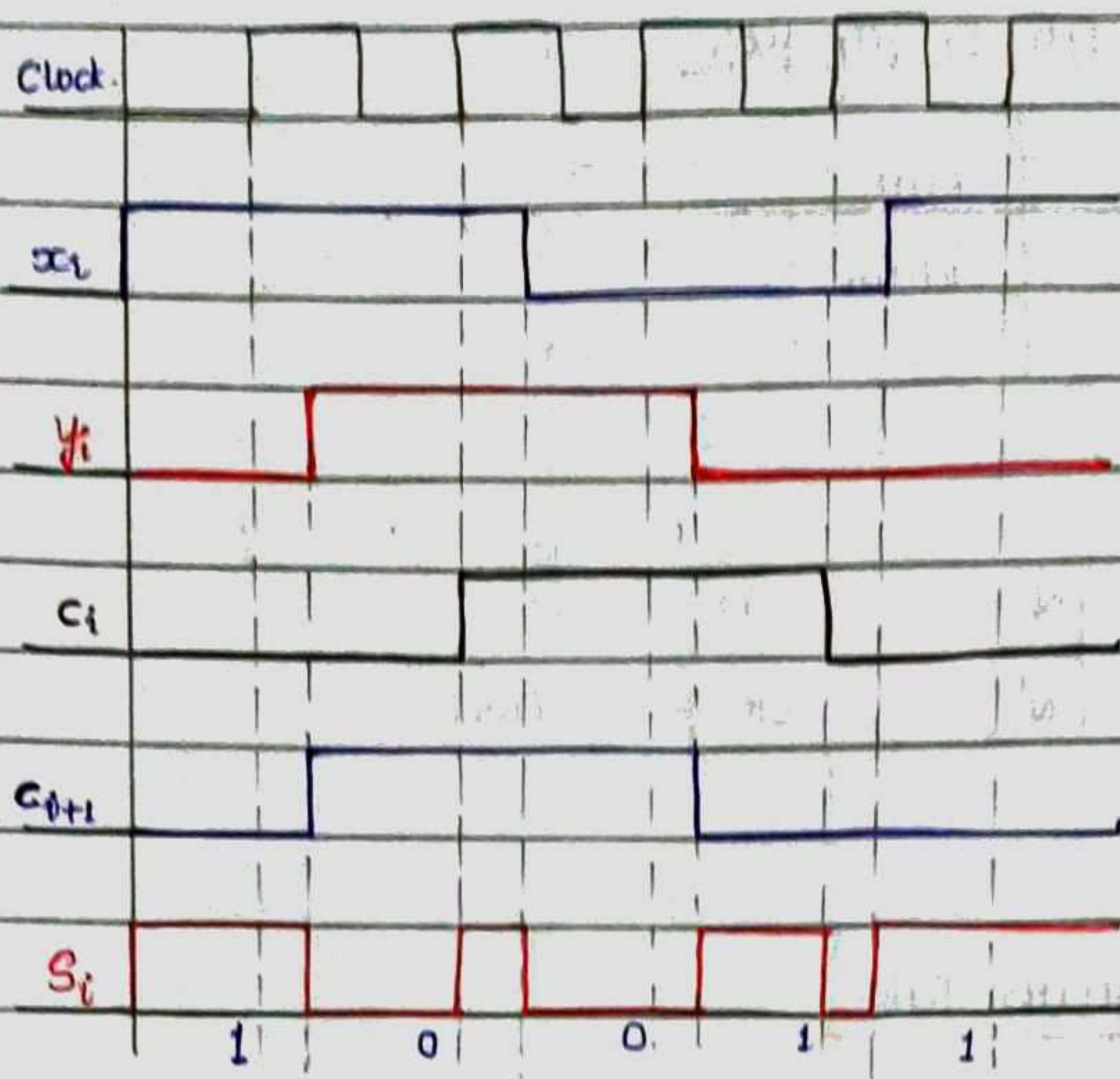
(a) With D Flip-Flop



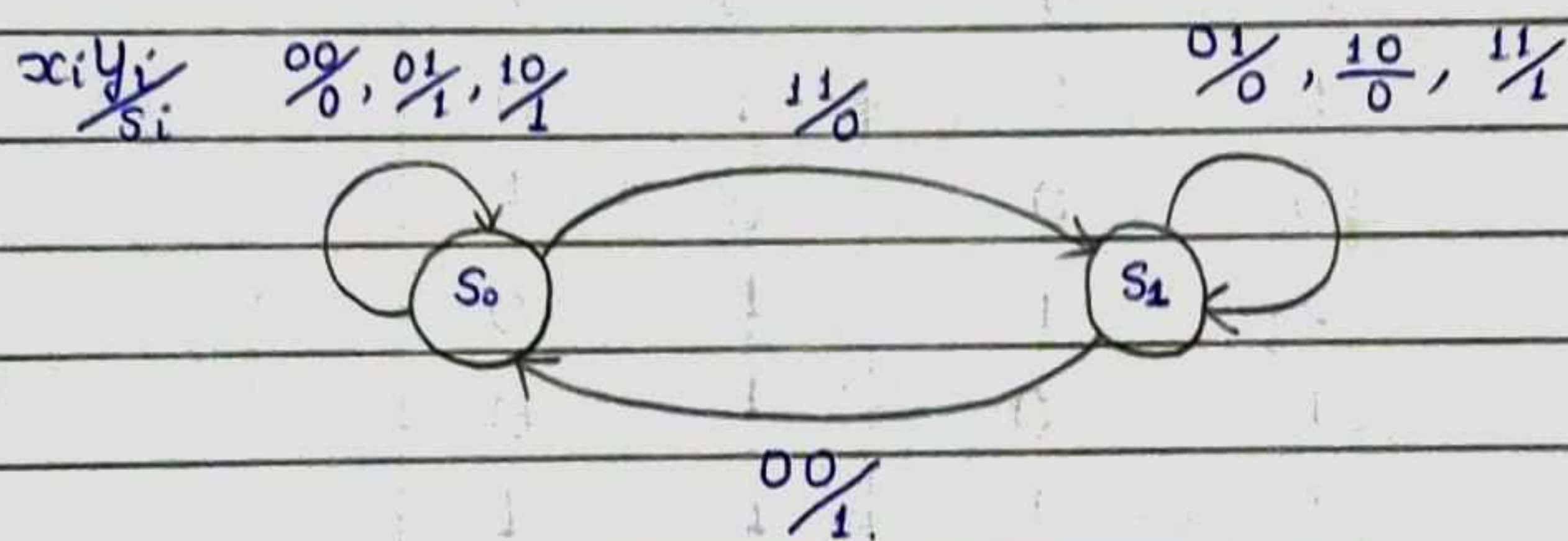
(b) Truth Table

x_i	y_i	C_i	C_{i+1}	S_i
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

Timing diagram for Serial Adder



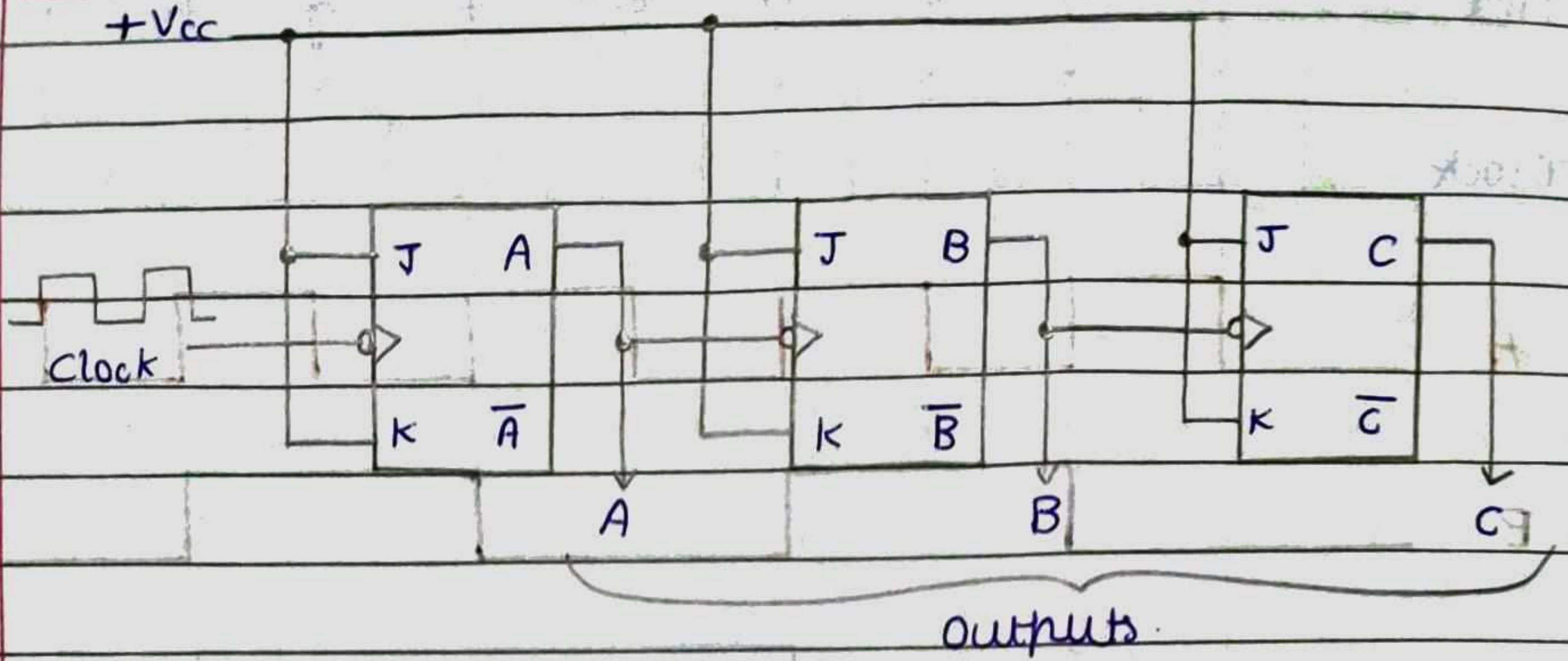
State Graph for Serial Adder



- We can construct a state graph for the serial adder. The serial adder is a Mealy machine with inputs x_i and y_i and output s_i .
- The two states represent a carry (c_i) of 0 and 1, respectively.
- From the table, c_i is the present state of the sequential circuit, and c_{i+1} is the next state.
- If we start in S_0 (no carry), and $x_i y_i = 11$, the output is $s_i = 0$ and the next state is S_1 . This is indicated by the arrow going from state S_0 to S_1 .

⇒ Asynchronous counters

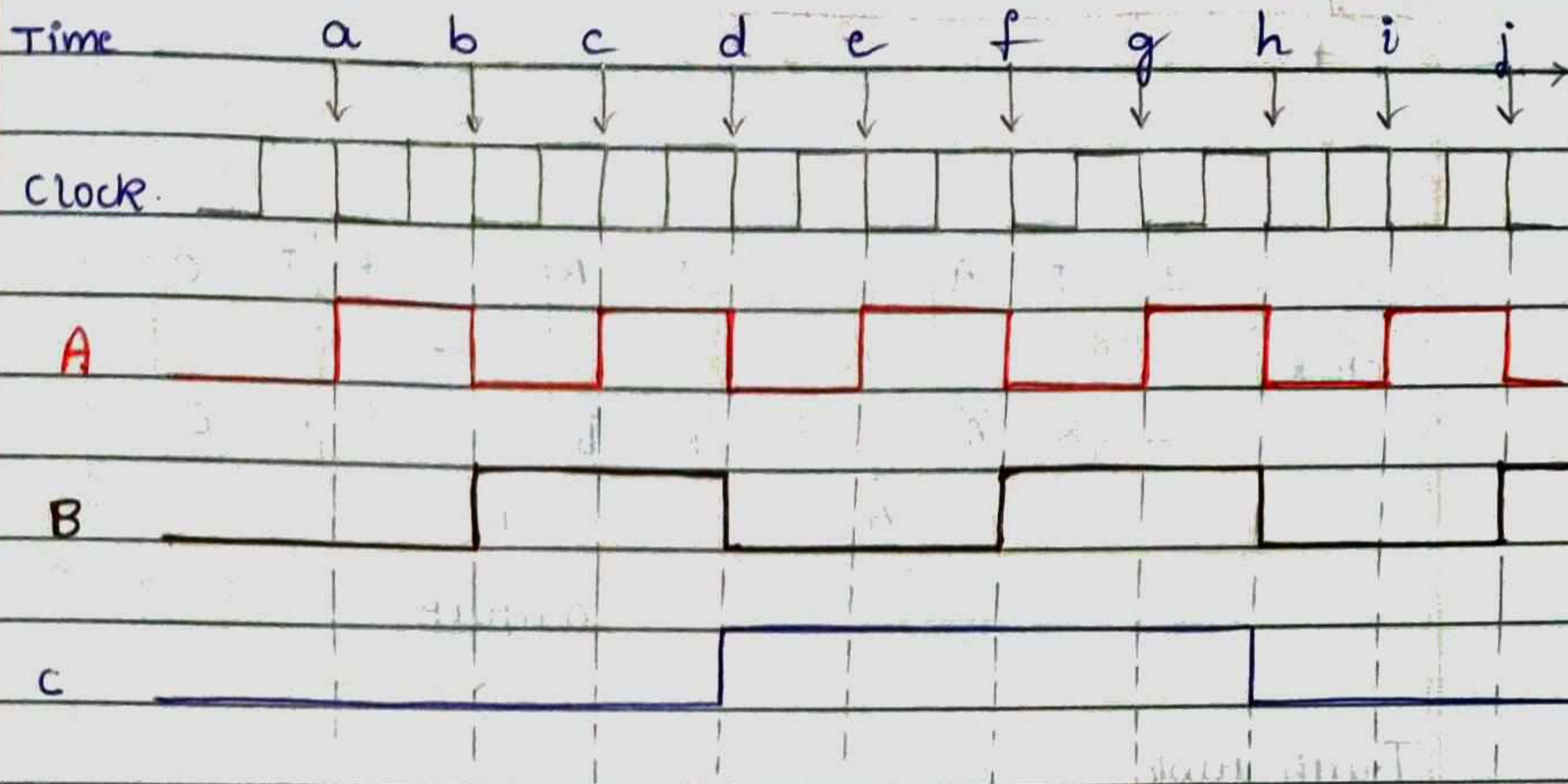
→ Asynchronous UP Counters



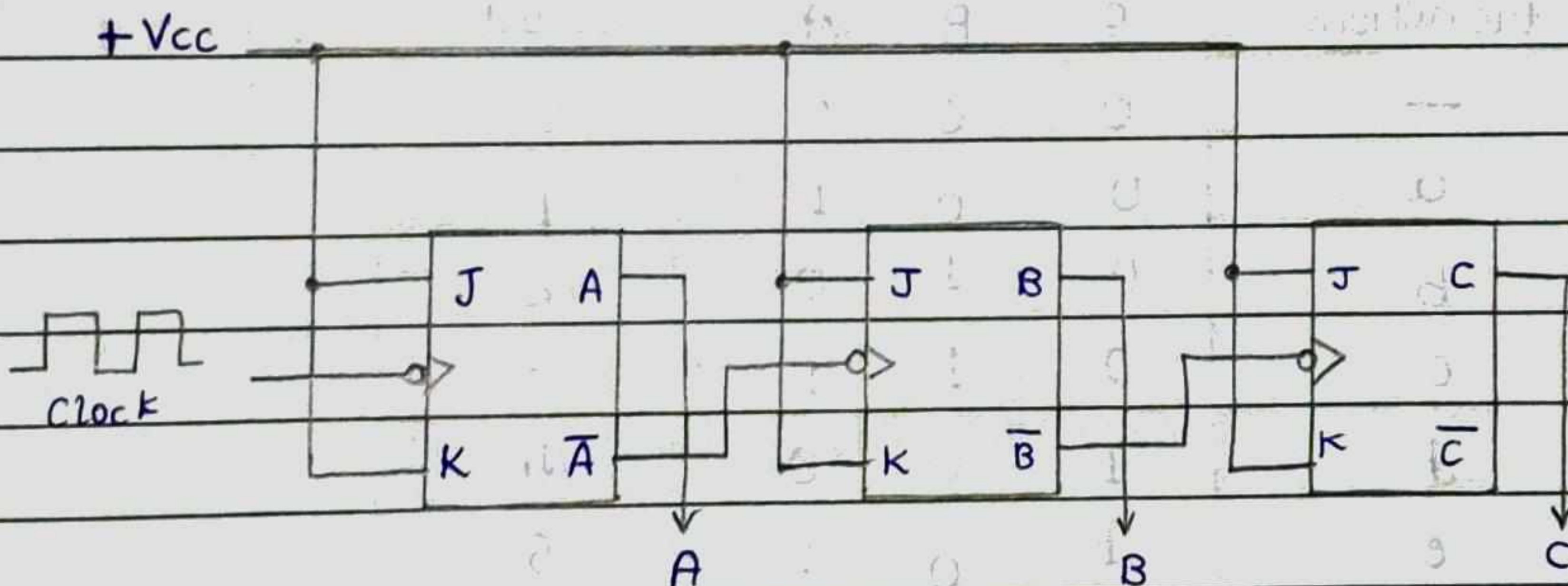
Truth table

Negative clock transitions	C	B	A	State or count
—	0	0	0	0
a	0	0	1	1
b	0	1	0	2
c	0	1	1	3
d	1	0	0	4
e	1	0	1	5
f	1	1	0	6
g	1	1	1	7
h	0	0	0	0

Waveforms



→ Asynchronous Down counter.



(37)

Truth Table

Count	C	B	A
7	1	1	1
6	1	1	0
5	1	0	1
4	1	0	0
3	0	1	1
2	0	1	0
1	0	0	1
0	0	0	0
7	1	1	1

Timing diagram/Waveforms.