Binary Multiplier:

Multiplying a 2x2 bit numbers

let 11 x 11 = 1001

(3) (3) (9)

A_1	A_0	B_1	B_0	P_3	P_2	P_1	P_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

Basic Algorithm of Multiplication:

B_1 B_0

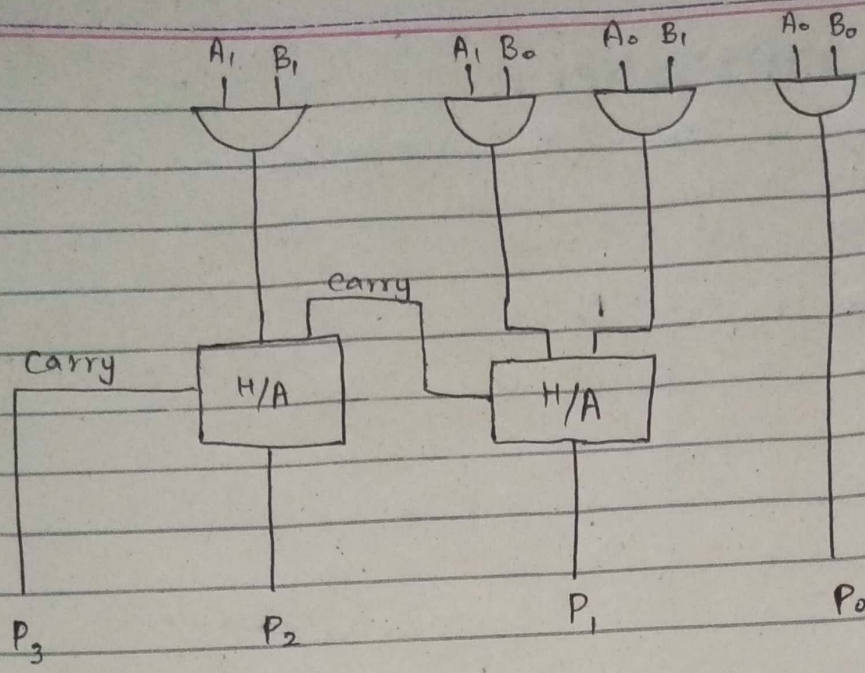
carry ↓ x A_1 A_0

carry ↓ A_0B_1 A_0B_0 ← 1 bit it is (bcz it is simply AND of A_0 and B

A_1B_1 A_1B_0 x

P_3 P_2 P_1 ~~A_0B_0~~ P_0

↳ Circuit Diagram



5 بٹ bits کے لیے 3 output بٹ کی ضرورت ہے

Two Approaches:

(i) 1st Approach: Simplify by truth Table then we'll have to make k-map, we have only learned k-map till 4 variables.

Let: A = 3-bits, B = 4-bits

total bits of inputs are 7, so it has complex one k-map, hence we will use 2nd approach.

(ii) 2nd Approach: Algorithmic Approach

A×B = (m+n) bits A = m bits, B = n bits

B_3 B_2 B_1 B_0

carry A_2 A_1 A_0

A_0B_3 A_0B_2 A_0B_1 A_0B_0

carry carry A_1B_3 A_1B_2 A_1B_1 A_1B_0 0

S_4 S_3 S_2 S_1 S_0

A_2B_3 A_2B_2 A_2B_1 A_2B_0 0

carry + A_2B_2 → P_5

$S_3 + A_2B_2$ → P_4

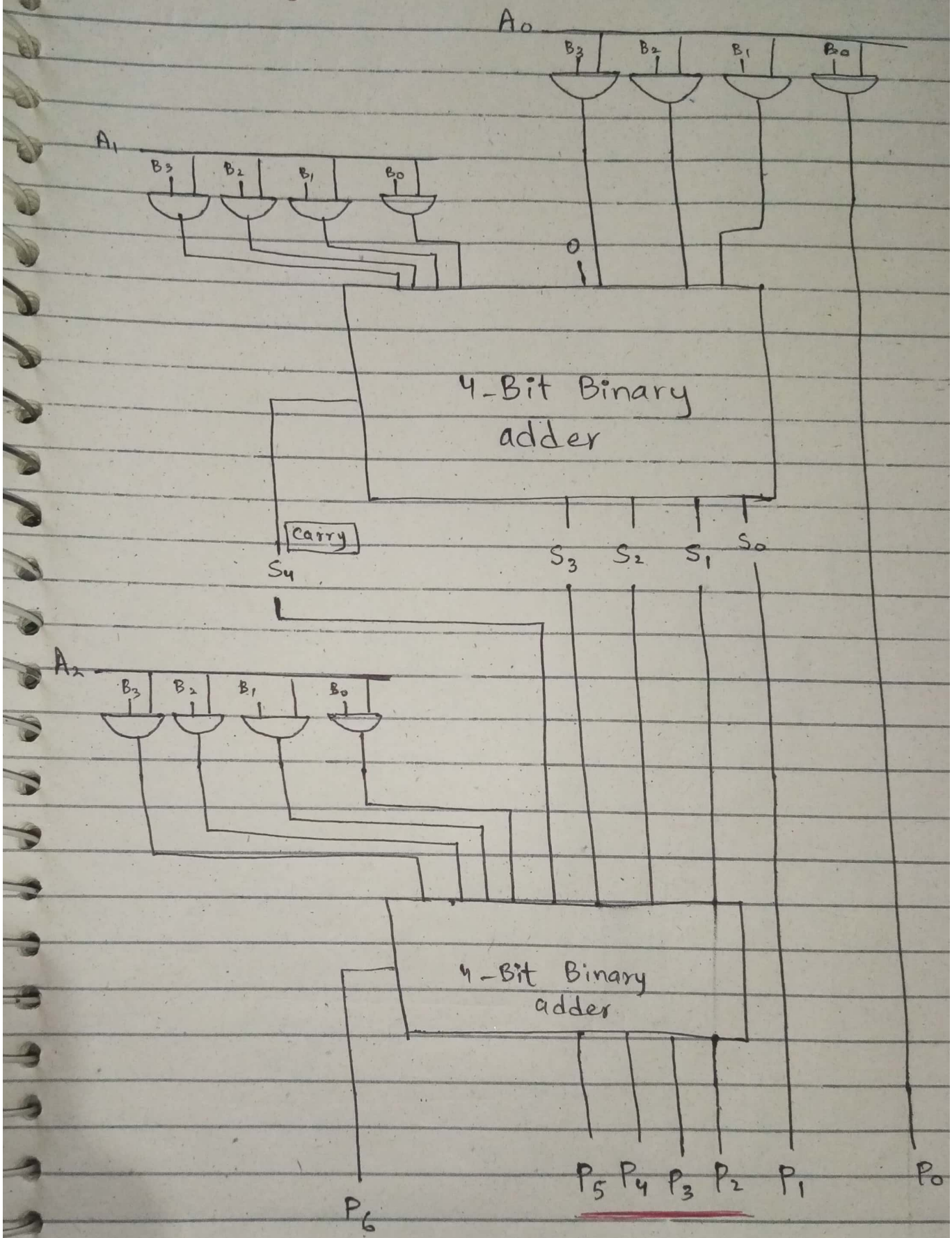
$S_2 + A_2B_1$ → P_3

$S_1 + A_2B_0$ → P_2

$S_0 + 0$ → P_1

P_0

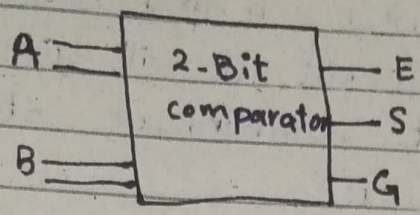
Logic Diagram:



Magnitude Comparator Circuit:

2-Bit magnitude comparator:

Circuit: A, B are 2 bit numbers



A_1	A_0	B_1	B_0	E	S	G
0	0	0	0	1	0	0
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	0	0	1
0	1	0	1	1	0	0
0	1	1	0	0	1	0
0	1	1	1	0	1	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	1	0	0
1	0	1	1	0	1	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	1	0	0

→ Simplificate equations for E, S, G to make a circuit:

→ Which circuit can be used for equality comparison?

A	B	F (XNOR) ⇒ $A \odot B = F$
0	0	1
0	1	0
1	0	0
1	1	1

↳ tells about equality

A_3 A_2 A_1 A_0

B_3 B_2 B_1 B_0

x_3 x_2 x_1 x_0 . $x_0 = A \odot B$

$E = x_3 \cdot x_2 \cdot x_1 \cdot x_0$ (~~dot~~ AND of all outputs (x_0, x_1, x_2, x_3) so if any output is different, it will give that numbers are not equal)

To check greater:

A 1 1 1 0

B 1 1 1 0

$G = A_3 \cdot B_3' + x_3 \cdot A_2 B_2' + x_3 \cdot x_2 \cdot A_1 B_1' + x_3 \cdot x_2 \cdot x_1 \cdot A_0 B_0'$

$A_3 \cdot B_3' = (1)(0) = 0$ → it shows that both of them are same, so logic moves to next bit.

~~$A_3 B_3' = (0)(0) = 0$~~

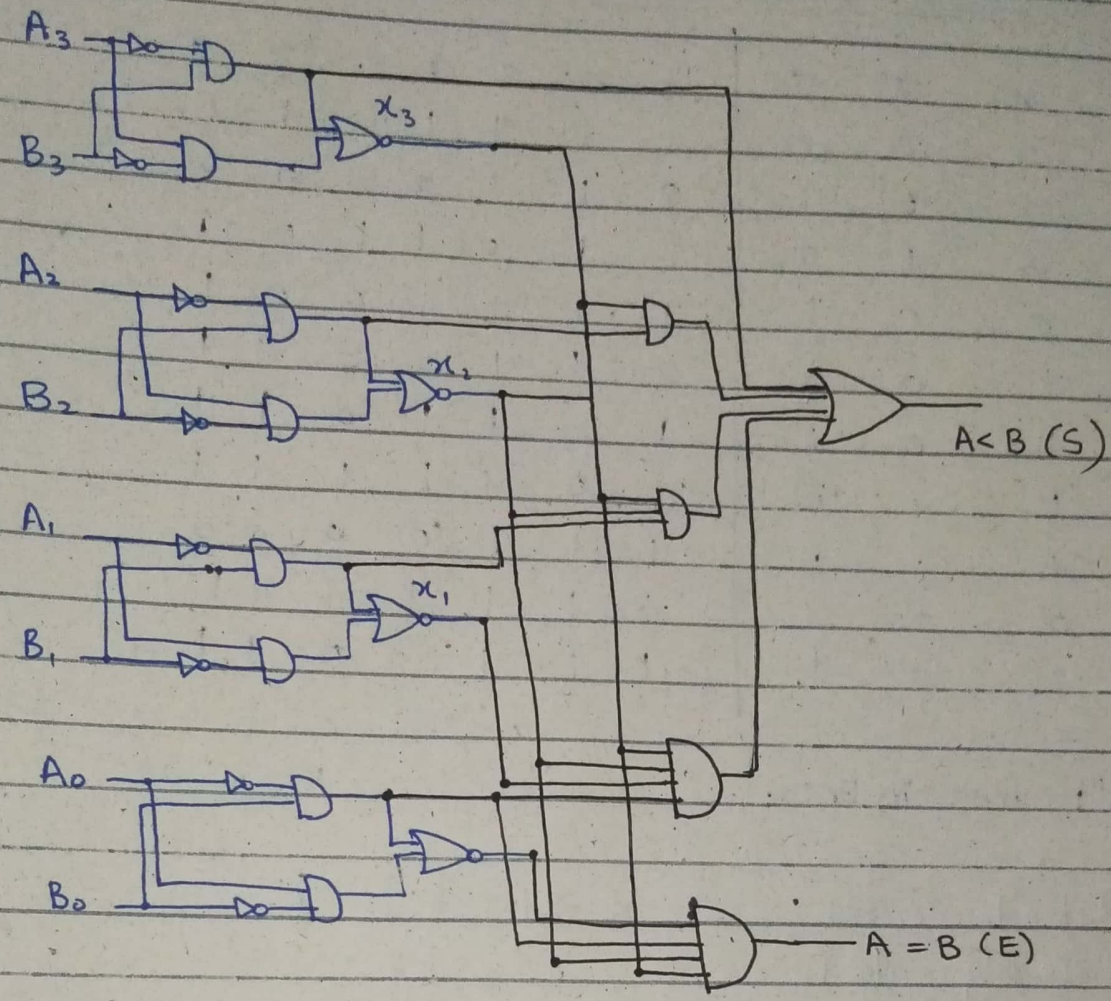
↳ it shows that

To check smaller:

$S = A_3' \cdot B_3 + x_3 \cdot A_2' \cdot B_2 + x_3 \cdot x_2 \cdot A_1' B_1 + x_3 \cdot x_2 \cdot x_1 A_0' B_0$

E, S, G → extenable for higher bits

Circuit Diagram:



- For G, same as S but take $A_0'B_0$ as input instead of A_0B_0'.

$(x'y + xy')'$

f | | b
e | g | c
d

BCD to seven segment Decoder:

w	x	y	z	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1

don't cares in BCD:

→ don't cares are considered in its simplification.

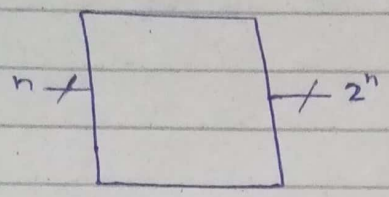
in CPU, controls

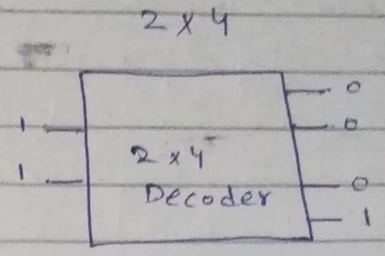
Decoder:

For $n = 2$:

$n \times 2^n$

2×4





x	y	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Q# Design a $n \times 2^n$ decoder where $n = 2$ (or any no.)?

Q# Design a 5×80 decoder? (wrong question- as $n = 5$ and $n \times 2^n = 5 \times 2^5 \neq 5x$

Expression for D_0:

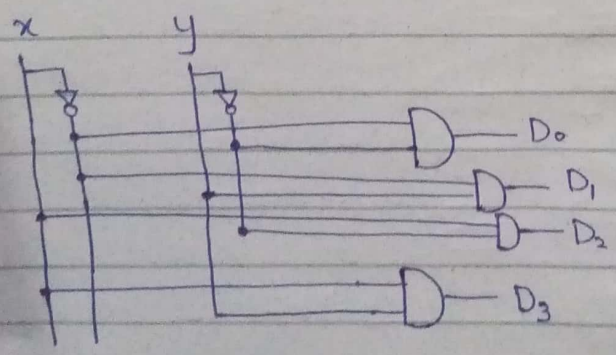
$D_0 = x'y'$

$D_1 = x'y$

$D_2 = xy'$

$D_3 = xy$

Circuit Diagram:



3x8 Decoder:

Design a 3x8 decoder?

$3 \times 2^3 = 3 \times 8$

x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_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

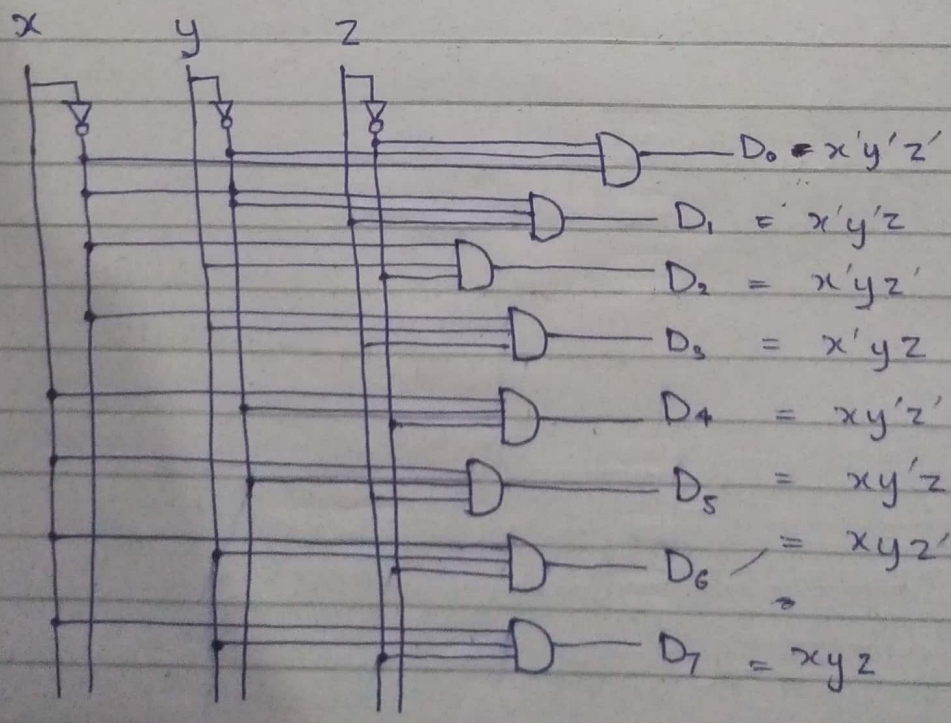
Expressions:

$D_0 = x'y'z'$, $D_3 = x'yz$, $D_6 = xyz'$

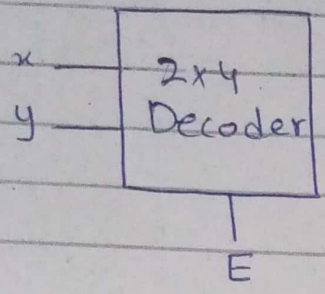
$D_1 = x'y'z$, $D_4 = xy'z'$, $D_7 = xyz$

$D_2 = x'yz'$, $D_5 = xy'z$

Circuit Diagram:



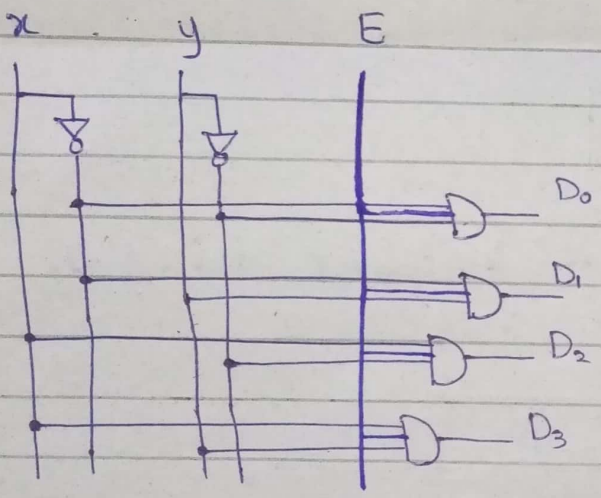
Enable Decoder:



⇒ If Enable (E) has 0, output will always be zero irrespective of inputs (x,y).

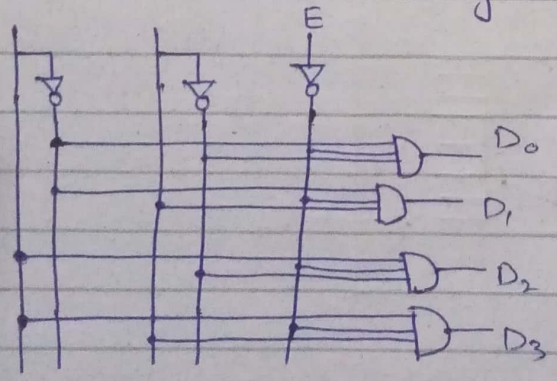
⇒ If E has 1, output will have appropriate output with respect to x,y.

Attach Enable (E) with every AND gate:



Altered Enable Decoder:

→ such that for E = 1, output should be 0 and vice versa. → by NOT of enable (E)

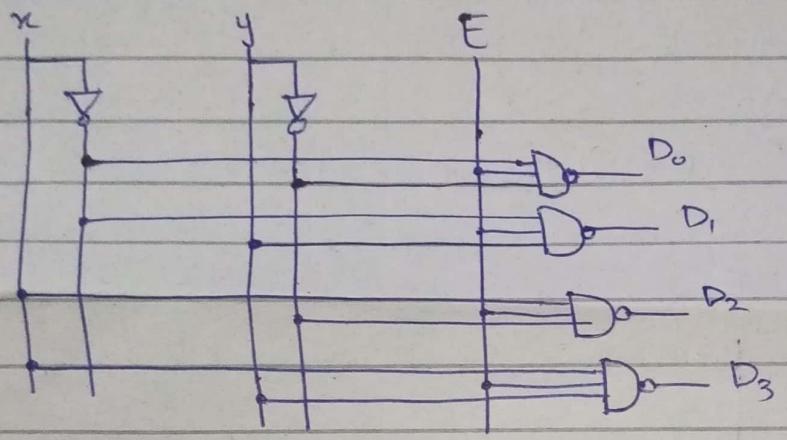


What happens when NAND is implemented in Decoder?

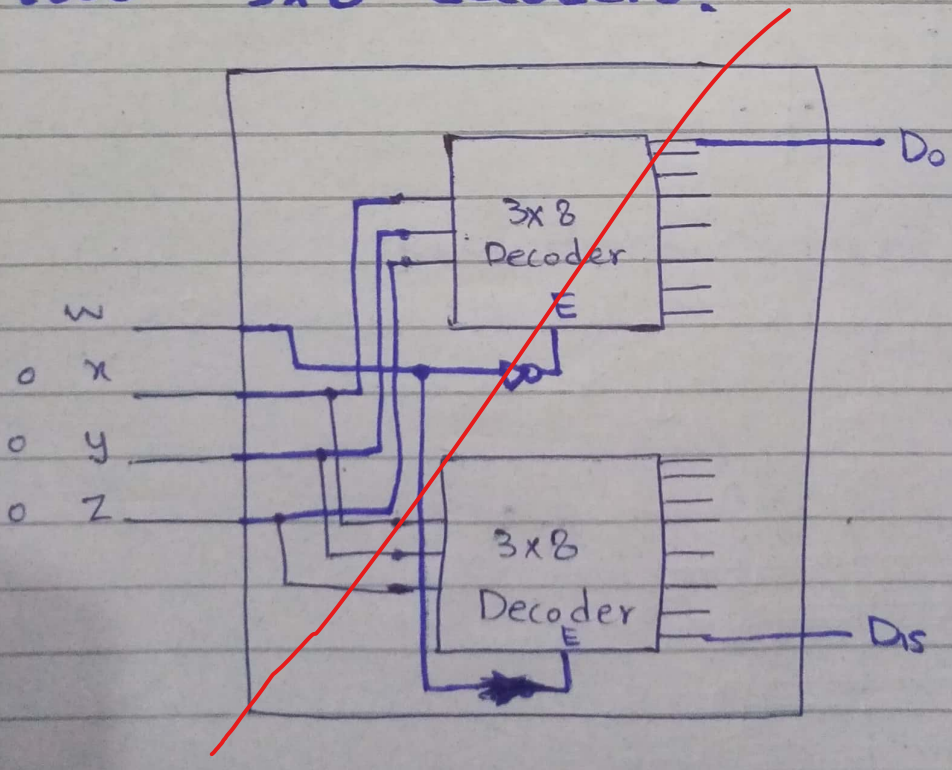
Truth Table:

x	y	D_0	D_1	D_2	D_3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

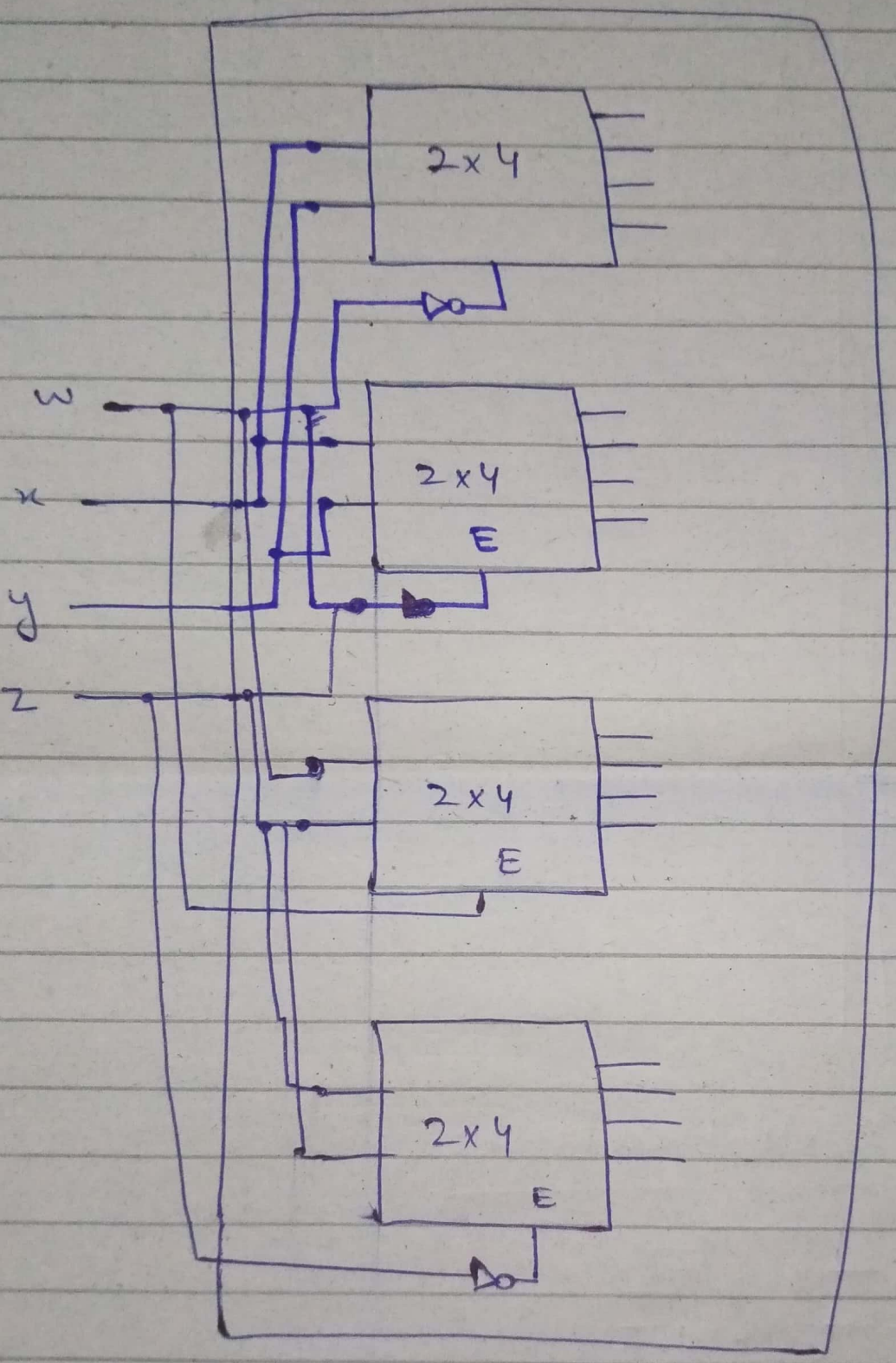
Circuit:



⇒Design a 4x16 decoder using two 3x8 decoders?



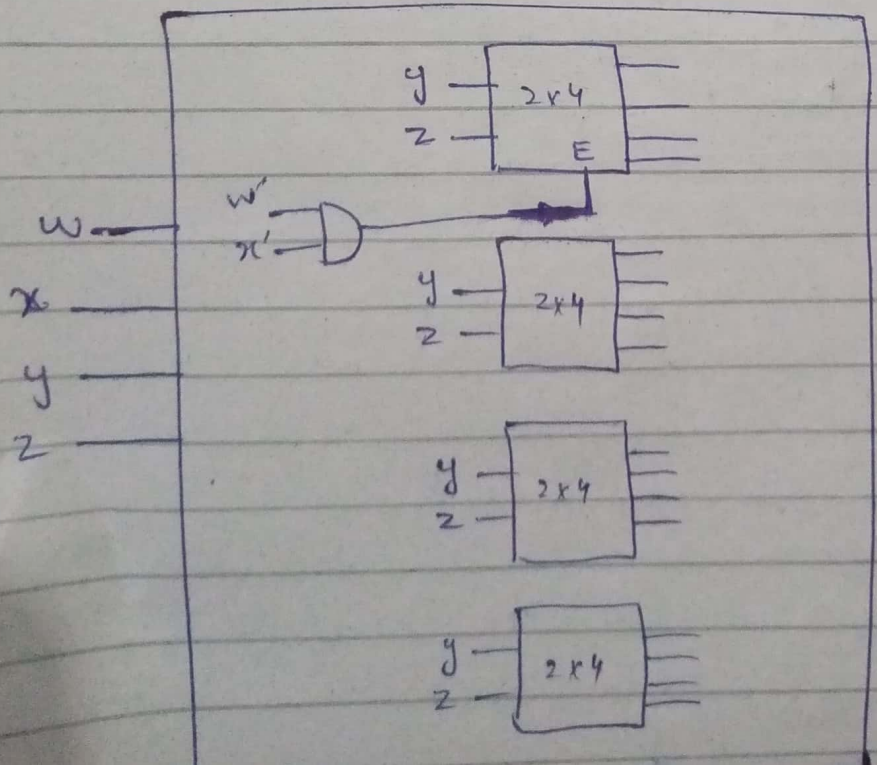
⇒ Design a 4x16 decoder using appropriate no. of 2x4 decoders.



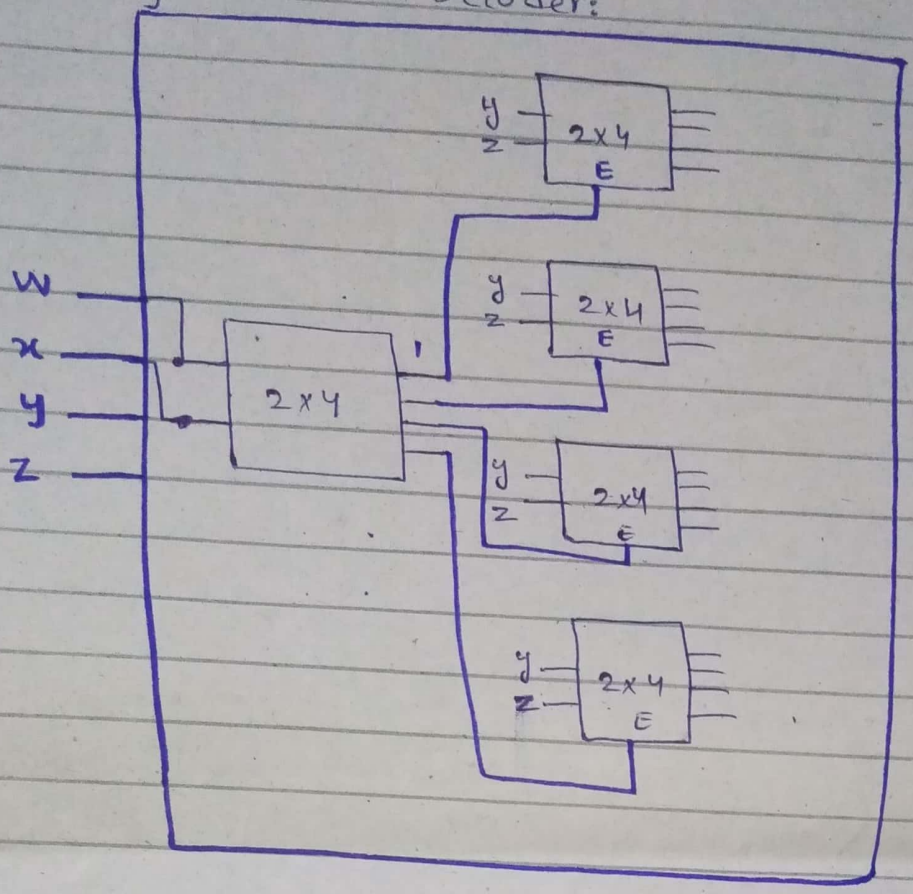
Design a 4x16 Decoder using appropriate no. of 2x4 decoders

w	x	y	z	D_0 ..... D_{15}
0	0	0	0	1
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Using four 2x4 decoders:



Using five 2x4 Decoder:



Full Adder Circuit: (x, y, z) → inputs

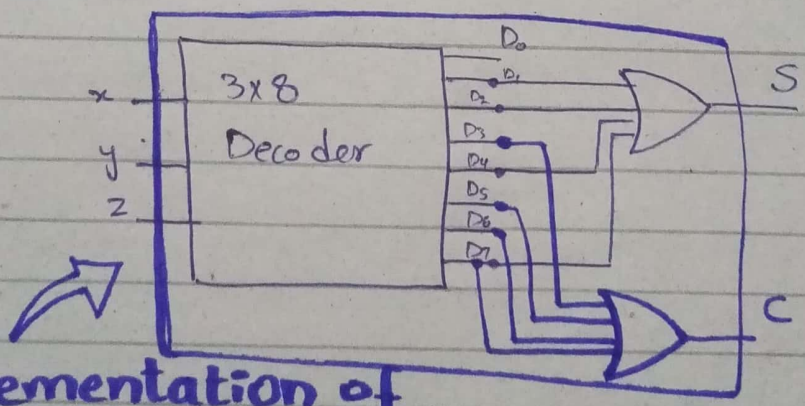
$S = \Sigma(1,2,4,7)$ (sum of minterms)

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

Decoder helps in implementation of functions.

we can implement these functions easily using 3x8 decoder as it has 3 input (x, y, z)

Sum implementation:



Implementation of min-terms using 3x8 decoder:

↓ without simplification (no K-map required)

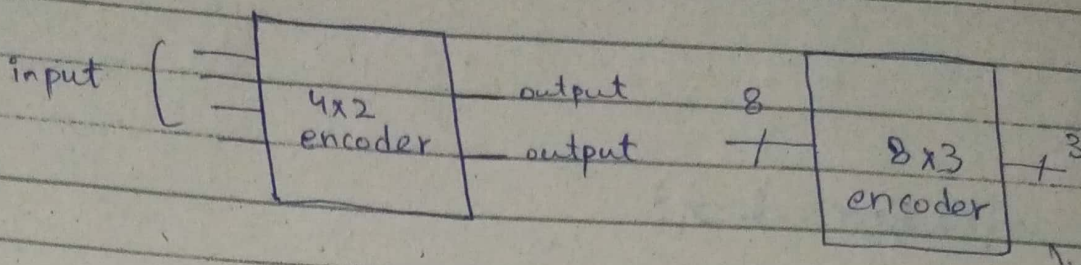
Designing a decoder

Decode a decoder using smaller decoder.

Design using min. no. of decoder

implement functions (find minterms → then implement)

Encoder:



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

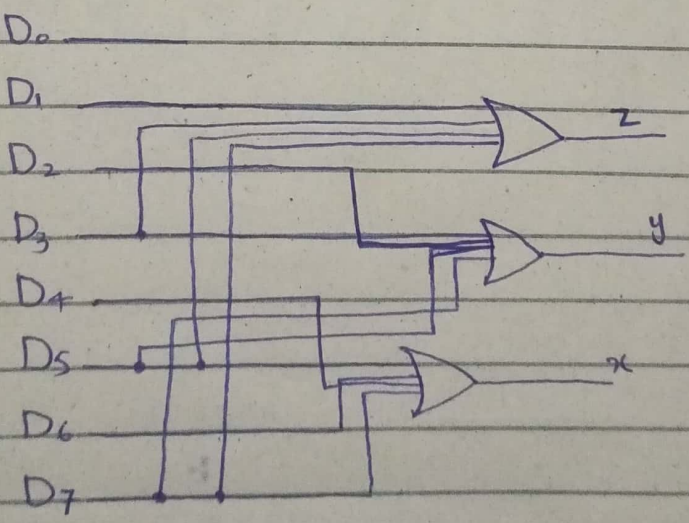
Expression in the form of Min-terms:

$x = D_4 + D_5 + D_6 + D_7$

$y = D_2 + D_3 + D_5 + D_7$

$z = D_1 + D_3 + D_5 + D_7$

Circuit Diagram:



If D_6 and D_7 both are 1, then output is 1 1 1.

(s due to priority order ⟶

Priority Order Encoder ⟶

2 Things:
(i) No "D" is 1
(ii) Multiple D's have 1

highest / lowest

What to do?
(ii) selection of priority by encoder
(i) Validity output → to check that either input is related to encoder or not.

→ If V is 0, it means ~~out~~ input is invalid.
→ If V is 1, it means ~~in~~ put is valid.

Priority Encoder:

D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	x	x	0
1	0	0	0	0	0	1
x	1	0	0	depends on priority encoder		1
0	1	0	0	0	1	1
x	x	1	0	1	0	1
0	0	1	0	1	0	1
x	x	x	1	1	1	1
0	0	0	1	1	1	1

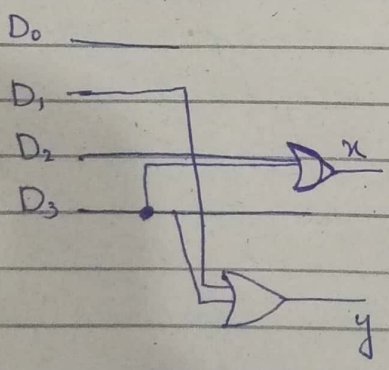
For high priority:

for high priority: x = 0, y = 1

For high-Priority 4x2 encoder:

D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	x	x	0 → invalid input
x	1	0	0	0	1	1
x	x	1	0	1	0	1
x	x	x	1	1	1	1

Will it be solved by ORing? NO



Combinations:

For x = 1;

xx10

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

xxx1

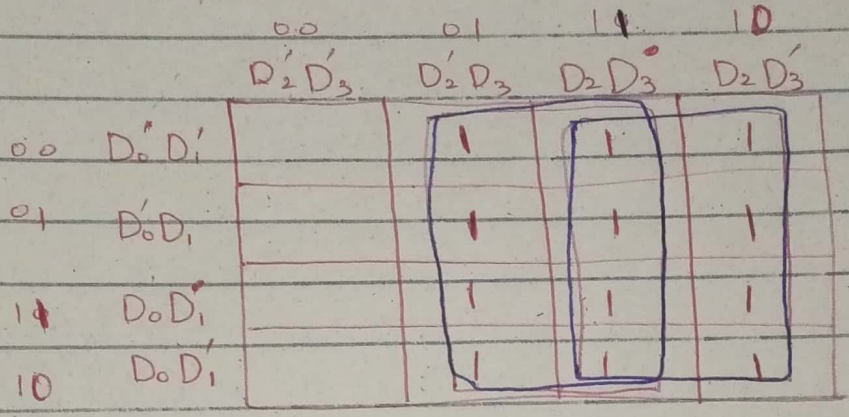
1
x
x
1

XXX1

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

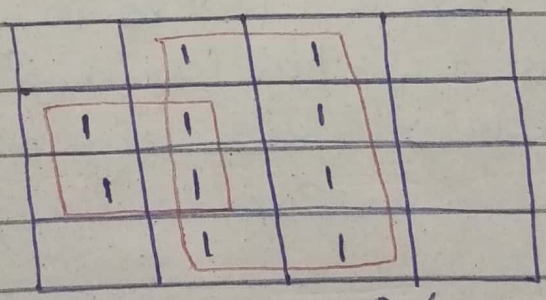
K-Map for x:

For y:



$x = D_3 + D_2$

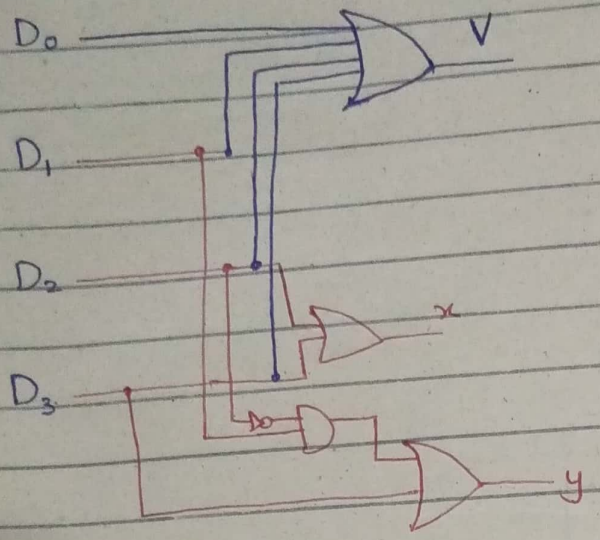
For y:



$y = D_3 + D_1D_2'$

Circuit Diagram:

∫ 4x2 Priority Encoder with Validity bit:



Draw an encoder such that Validity bit gives zero for invalid input that is hard coded? (Data is valid only for one 1 in input) here

↓ no priority:

Hence, only four inputs will be valid/ legal. Remaining 12 inputs will be illegal/ invalid in this case. So, what to do? find V by using k-map and x, y will be found easily.

D_0	D_1	D_2	D_3	x	y
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

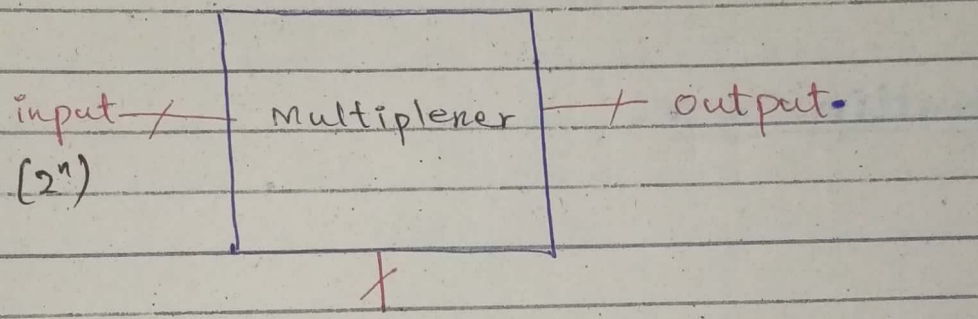
Least Priority Encoder:

D_0	D_1	D_2	D_3	x	y
1	x	x	x	0	0
0	1	x	x	0	1
0	0	1	x	1	0
0	0	0	1	1	1

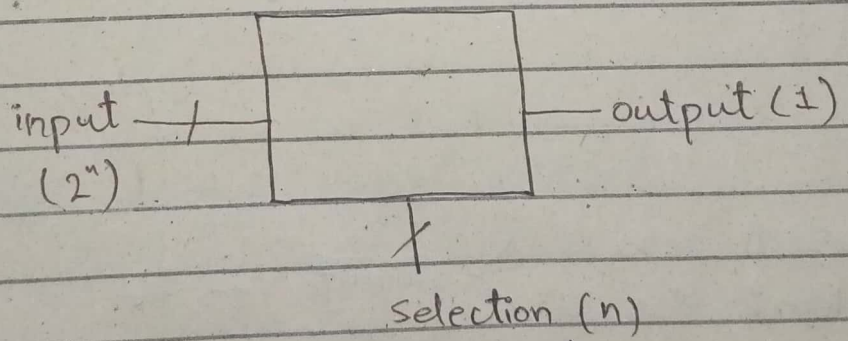
Multiplexers:

↳ in ALU

used in Arithematics to select an operation



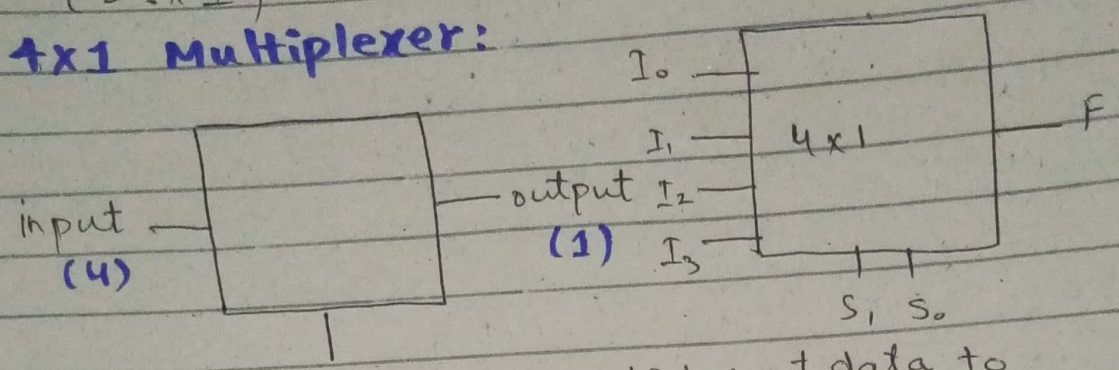
General:



- multiple inputs (2^n)
- one output
- multiple selection lines (n)

$(2^n \times 1)$

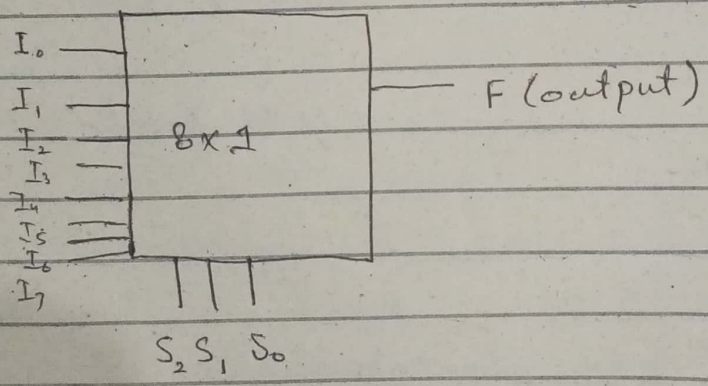
4x1 Multiplexer:



If multiplexer selects ~~out~~ input no. 1, so it sends its data to output.

If it selects ~~a~~ input no. 3, so it sends its respective data to output.

8x1 Multiplexer:

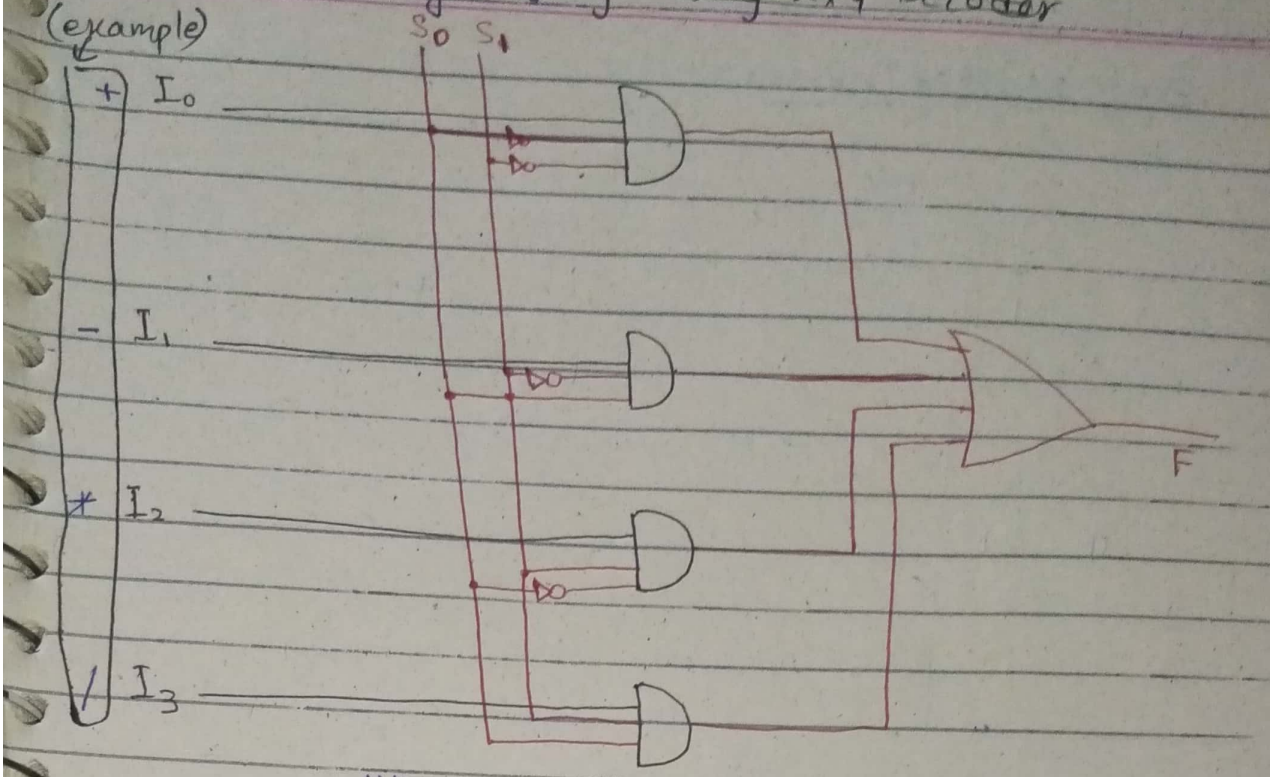


OR Gate: At least one '1' > output can be predicted
AND Gate: At least one '0' >

S_0	S_1	F
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Circuit Diagram: By using 2x4 Decoder

(example)



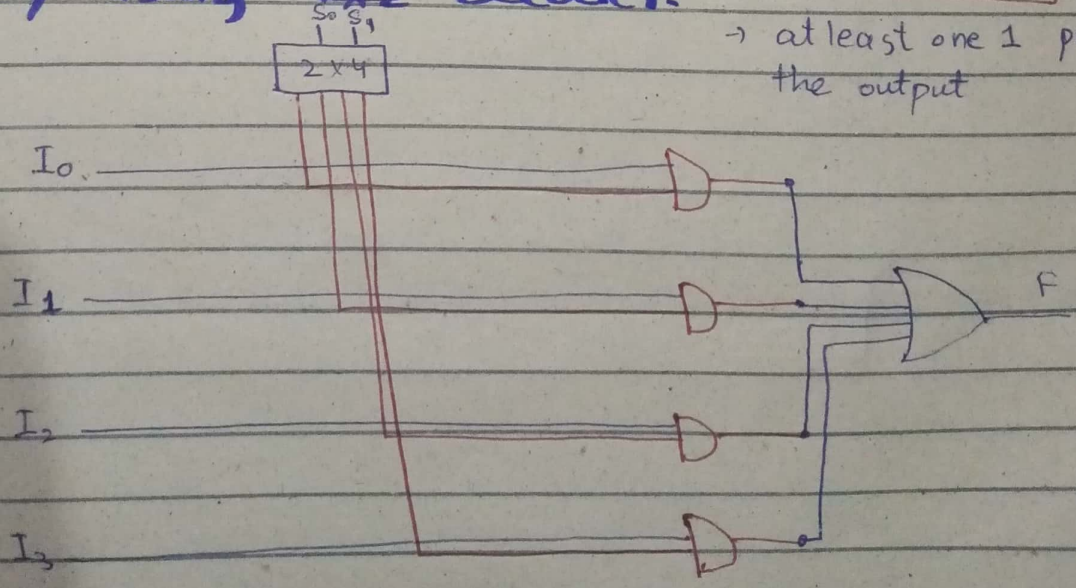
These Four AND gates are not going to give 1 as output at the same time. ~~No~~ Only one AND gate will produce 1 at a time for a diff. combination of S_0, S_1.

S_0	S_1	F
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Property of AND:
→ at least one 0 predicts that AND gate is going to produce 0 as a result

Property of OR:
→ at least one 1 predicts the output

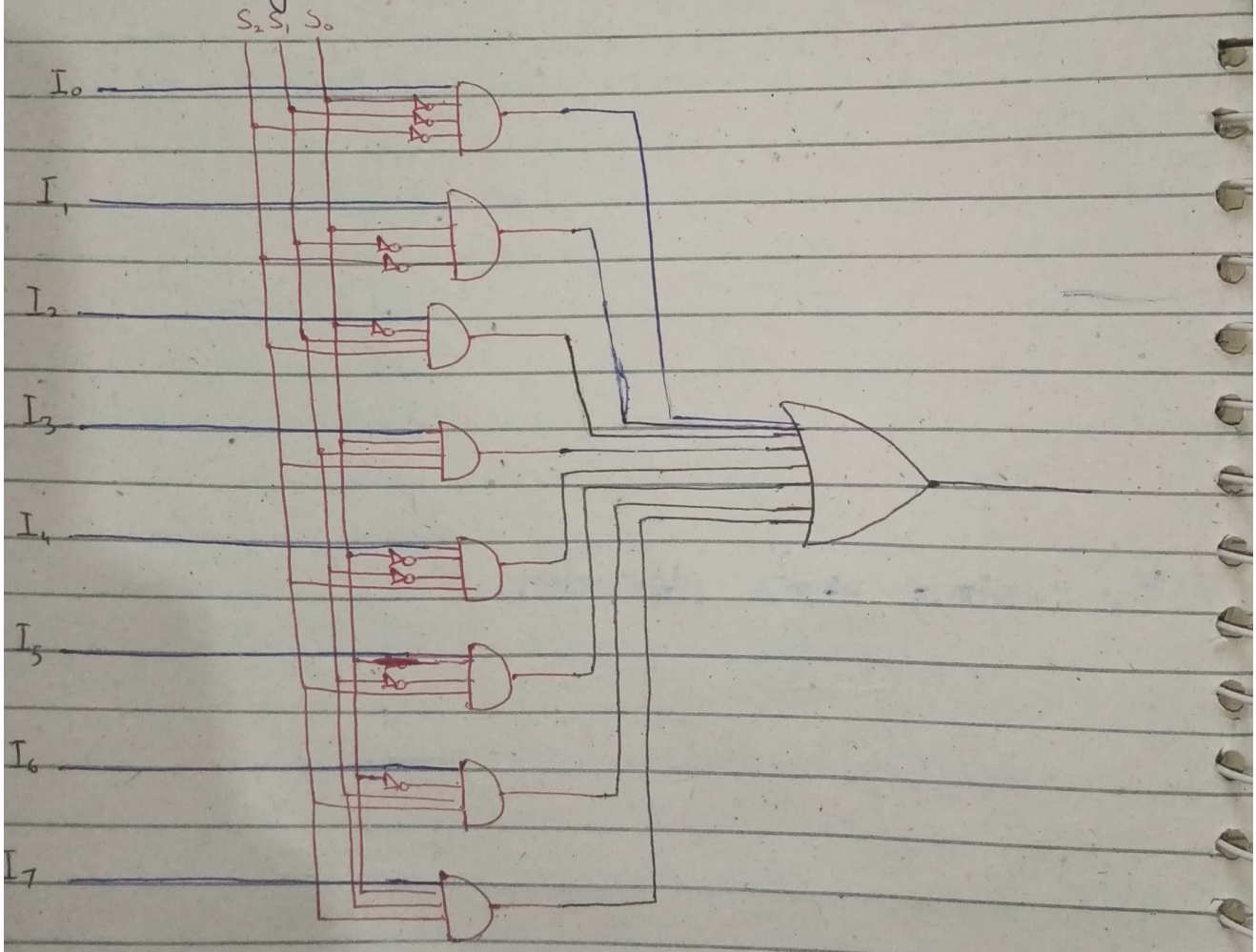
By using ~~1x2~~ 2x4 decoder:



8x1 Multiplexer:

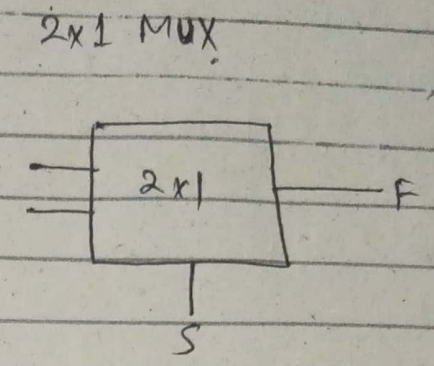
S_2	S_1	S_0	F
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

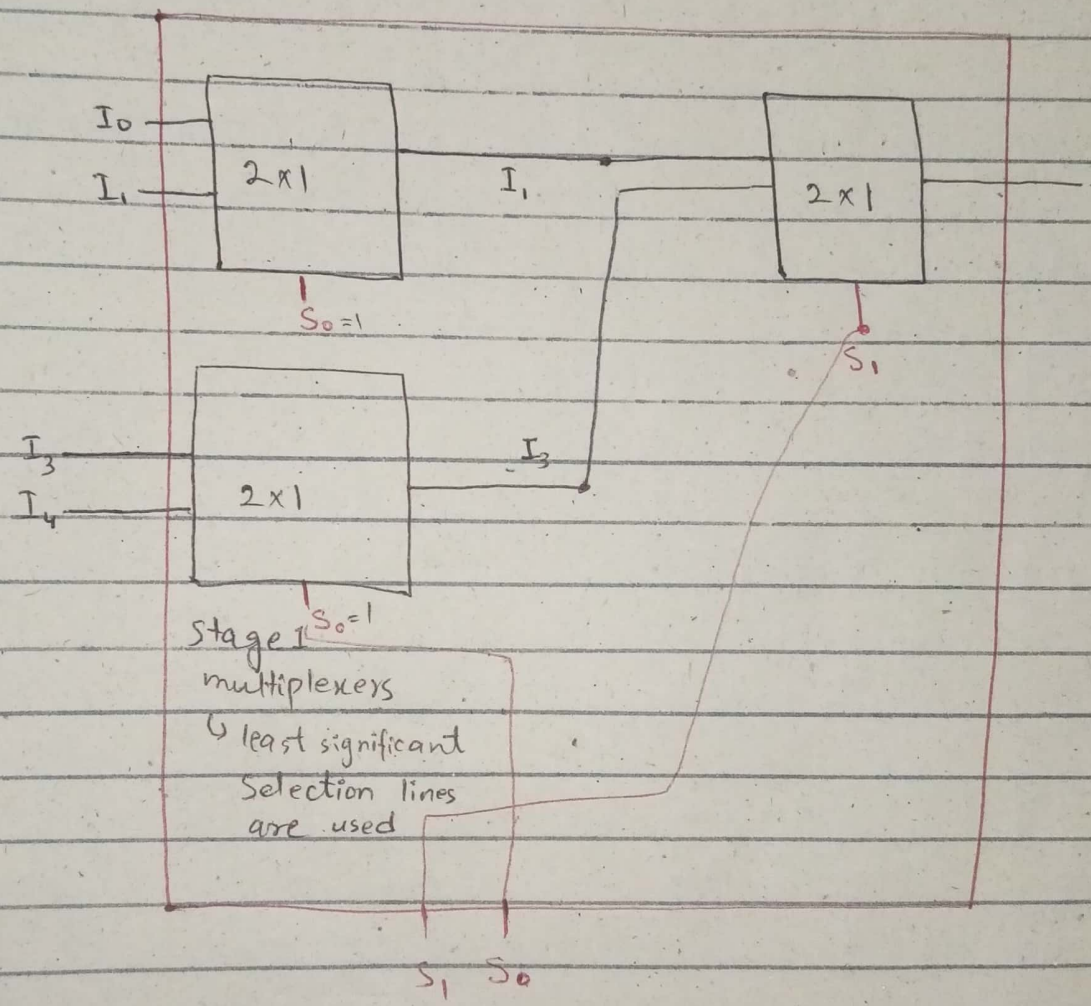
Circuit Diagram:



"Design a 4x1 multiplexer by using appropriate number of 2x1 multiplexer".

S_1	S_0	F
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3





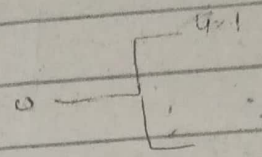
Two Stages:

(i) Least significant Selection lines at stage 1

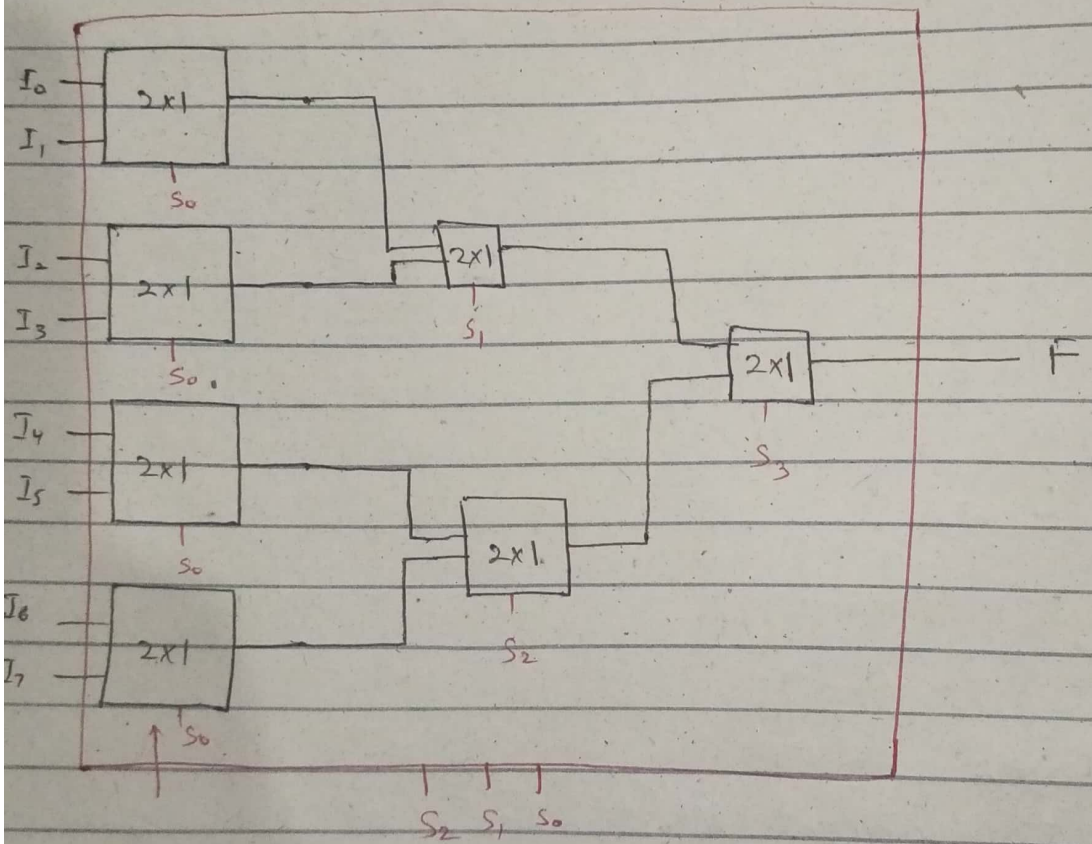
(ii) Most significant Selection lines at last stage

Design a 8x1 multiplexers using appropriate number of 2x1 :

S_2	S_1	S_0	F
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

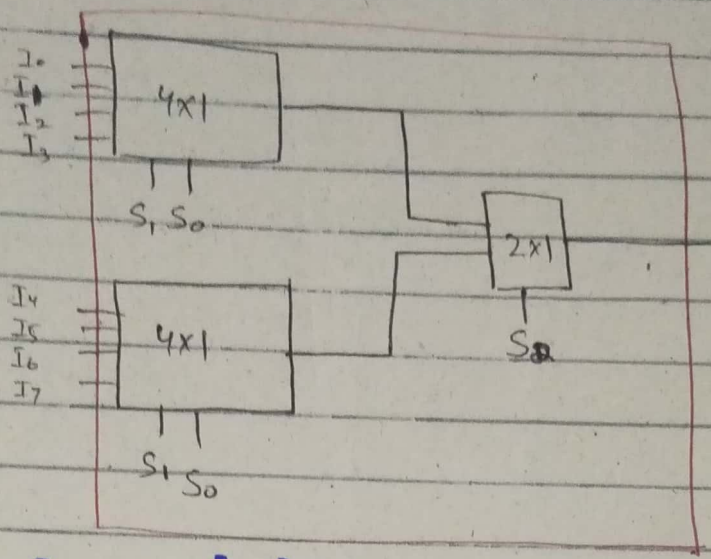


Circuit:



level 1/
stage 1
(least significant selection line)

8x1 Multiplexer by using two 4x1 and one 2x1:



Implementation of Boolean Functions by using multiplexer:

Step 1: For n inputs, use (n-1) lines of selection wala multiplexer.

$F(A,B,C) = \Sigma(1,3,5,6)$

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

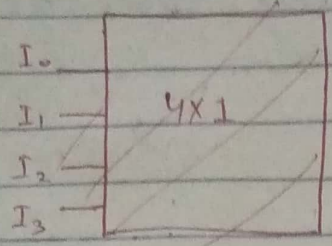
Implementation Table:

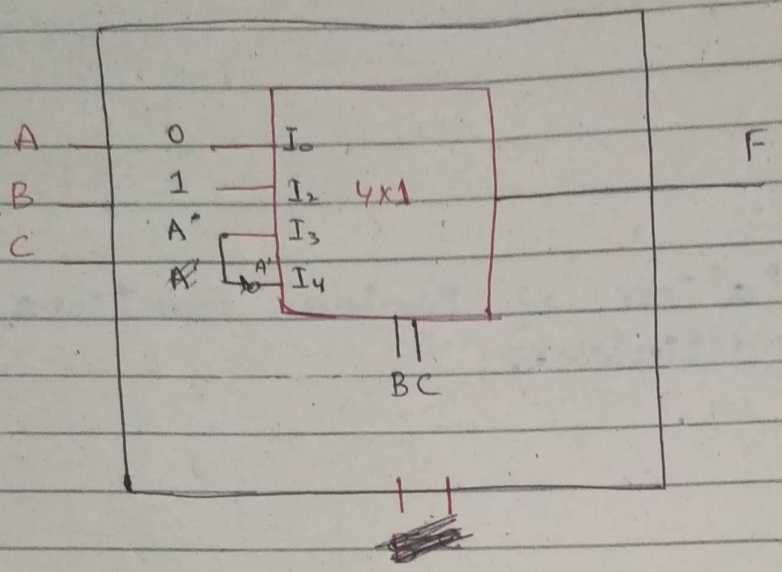
	I_0	I_1	I_2	I_3
A	0	(1)	2	(3) ← min-term
A'	4	(5)	(6)	7
	0	1	A'	A

↑ as no circled digit due to being not in Σ as minterms

↳ due to single circled digit.

↔ Circuit



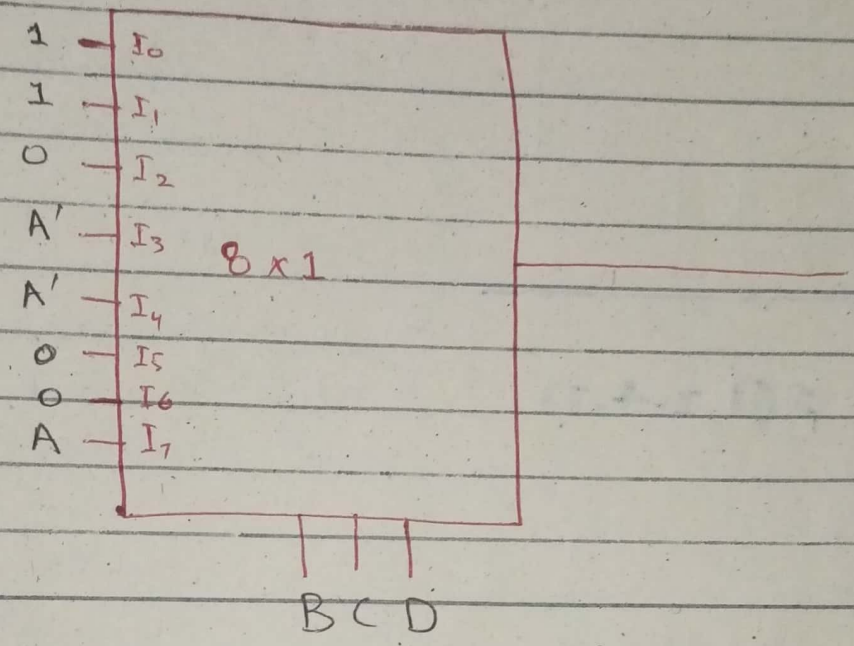


$F(A,B,C,D) = \Sigma(0,1,3,4,8,9,15)$

A	B	C	D	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Implementation Table:

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
A′	(0)	(1)	2	(3)	(4)	5	6	7
A	(8)	(9)	10	11	12	13	14	(15)
	1	1	0	A′	A′	0	0	A

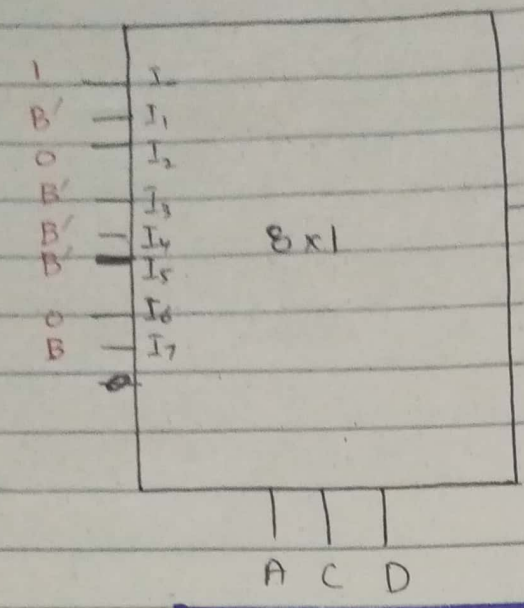


For ACD as selection line:

So, now as B selection line is left, toh table will be implemented on B.

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
B′	(0)	(1)	2	(3)	(8)	(9)	10	11
B	(4)	5	6	7	12	13	14	(15)
	1	B′	0	B′	B′	B′	0	B

Circuit:



$F(A,B,C) = \Sigma(1,2,4,7)$

↓

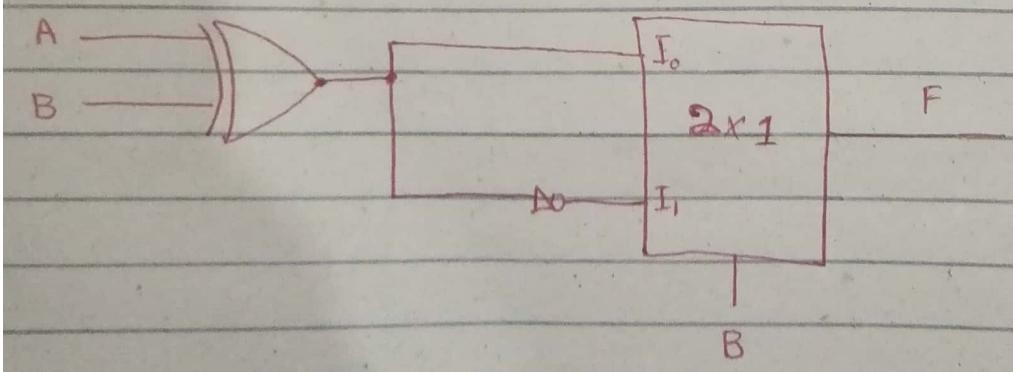
Implement with C as selection line:

	I_0	I_1
$A'B'$	0	1
$A'B$	2	3
AB'	4	5
$A'B'$	6	7

$= A'B + AB'$ $\quad$ $A'B' + AB$

$= A \oplus B$ $\quad$ $= A \odot B$

Circuit:



* If It is mentioned in question to use one specific selection (such as C), now we know that 2x1 multiplexer has only one selection line.

* If it is mentioned in Q to use a specific multiplexer (such as 2x1) now we know that only one input will be taken as selection line.

With B as selection line

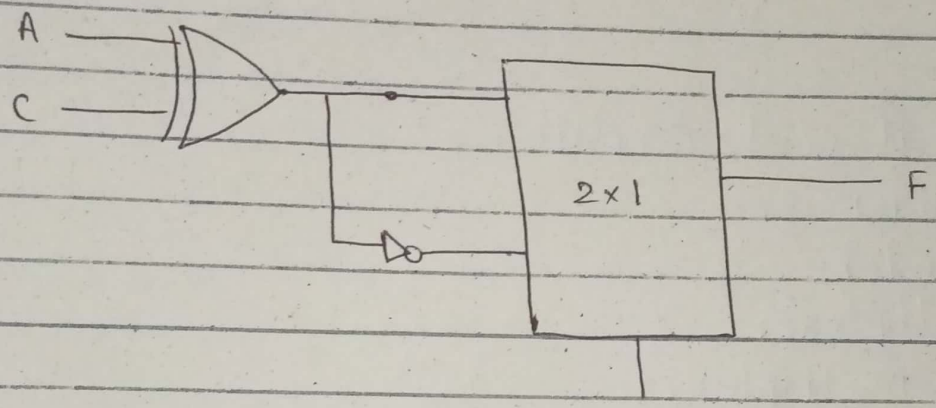
	I_0	I_1
$A'C'$	0	1
$A'C$	1	3
AC'	4	6
AC	5	7

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

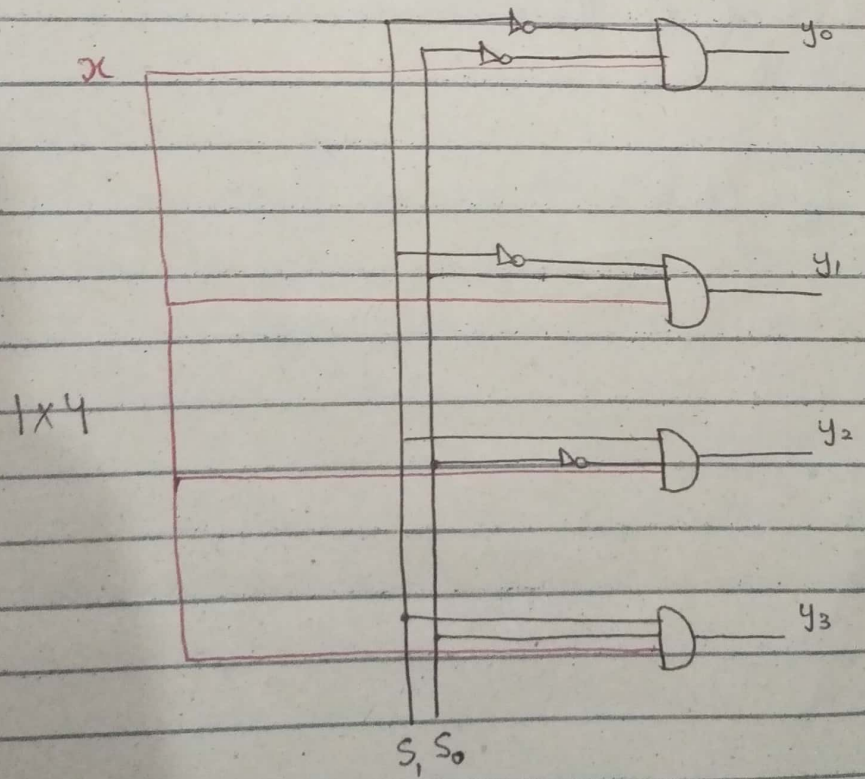
$= A'C + AC' = A \oplus C$

$= A'C' + AC = A \odot C$

Circuit:



De-Multiplexer:



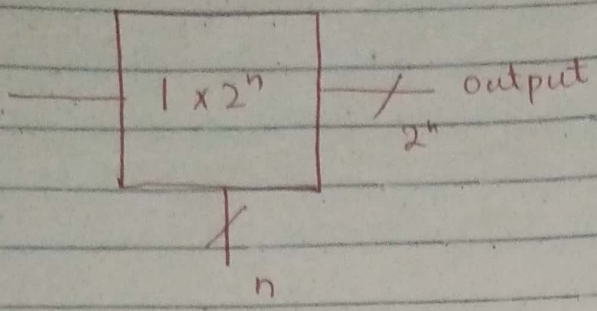
* This is somehow like "Decoder with an Enable".

← This is called "de-Multiplexer"

↓

opposite of Multiplexer

Circuit: De-Multiplexer



~~8x3~~

~~4x16~~

Combinational Circuits:

→ Decoder
→ Encoder
→ Multiplexer
→ De_Multiplexer

Identify on the base of their sizes:
i.e, 8x3, 4x16, 1x32, 32x1

End of Ch # 4