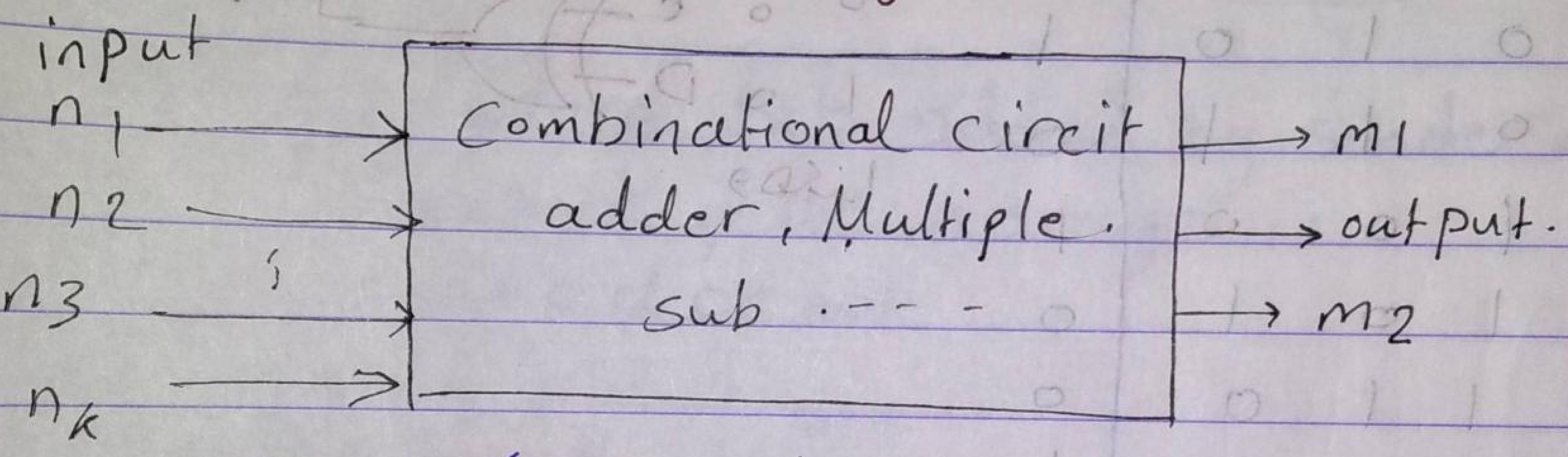
Chapter 4

Combinational logic circit (Design).

* There are two types of digital circit :-

① Combinational logic circit :- (ch4).

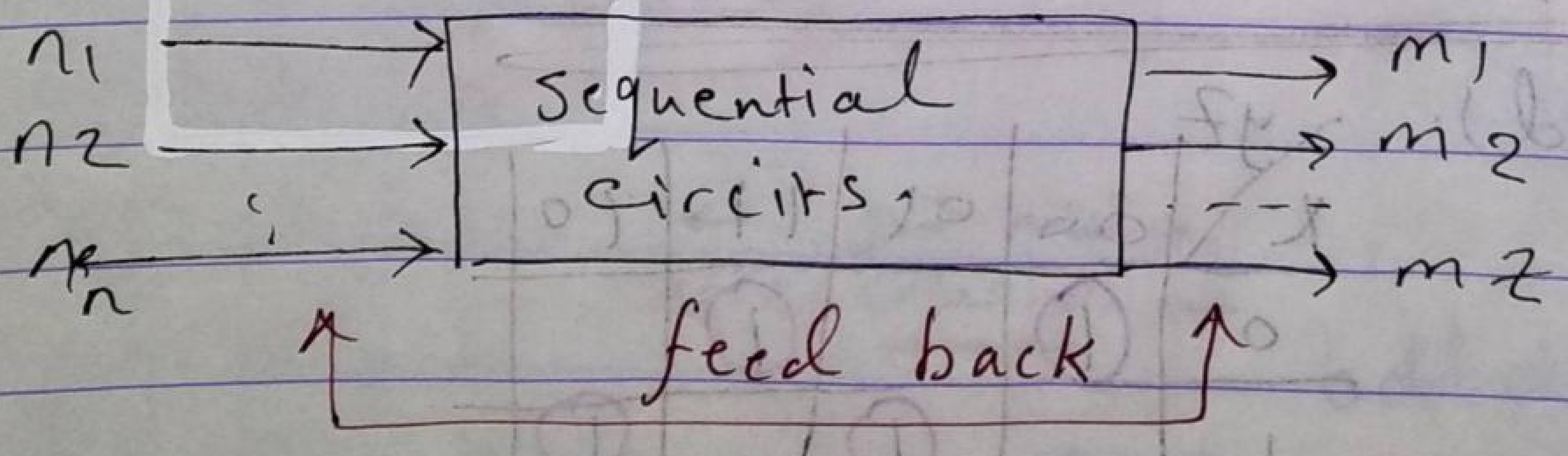
↳ output depends only in inputs.



$$\underset{output}{m_1} = \underbrace{(n_1 \cdot n_2) + n_3}_{inputs.}$$

② sequential circit (ch5, ch6).

output depend on input and previous output



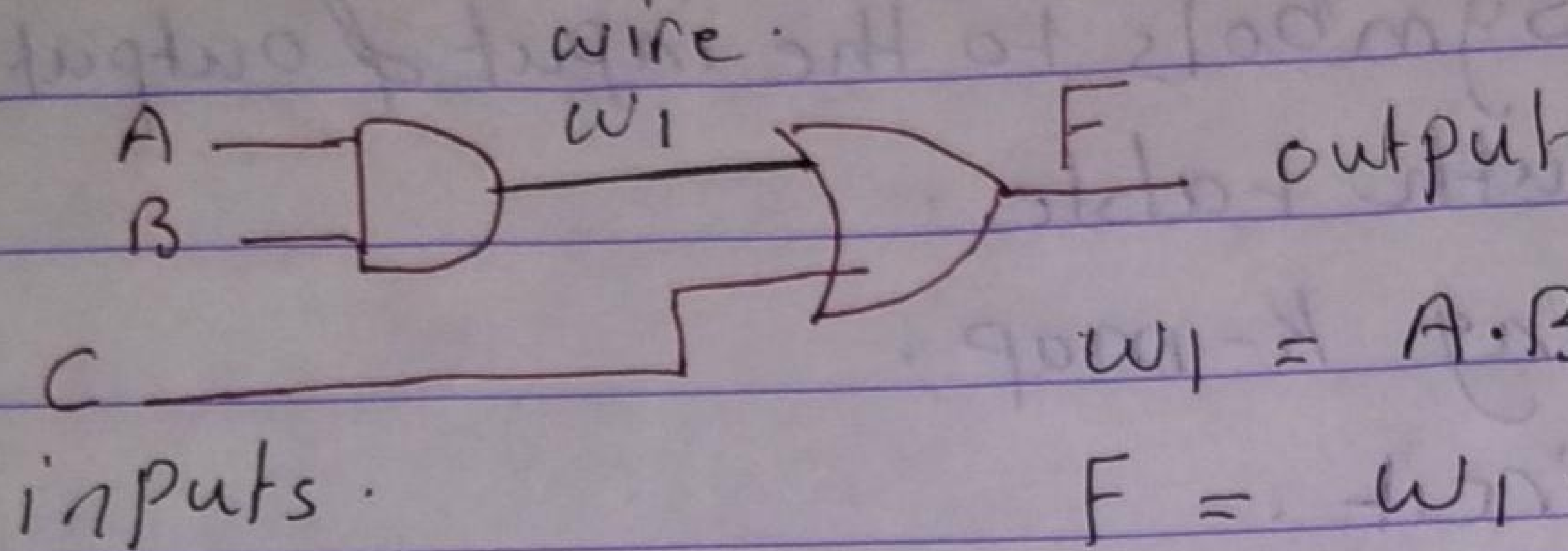
$$\underset{output.}{\underline{m_1}} = \underbrace{(n_1 + n_2 + m_3)}_{input} \cdot \underset{output}{\underline{m_2}}$$

Scanned by TapScanner

Combinational circits :-

Analysis ⇒ output $\xrightarrow{\quad}$ بدلالة inputs

ex

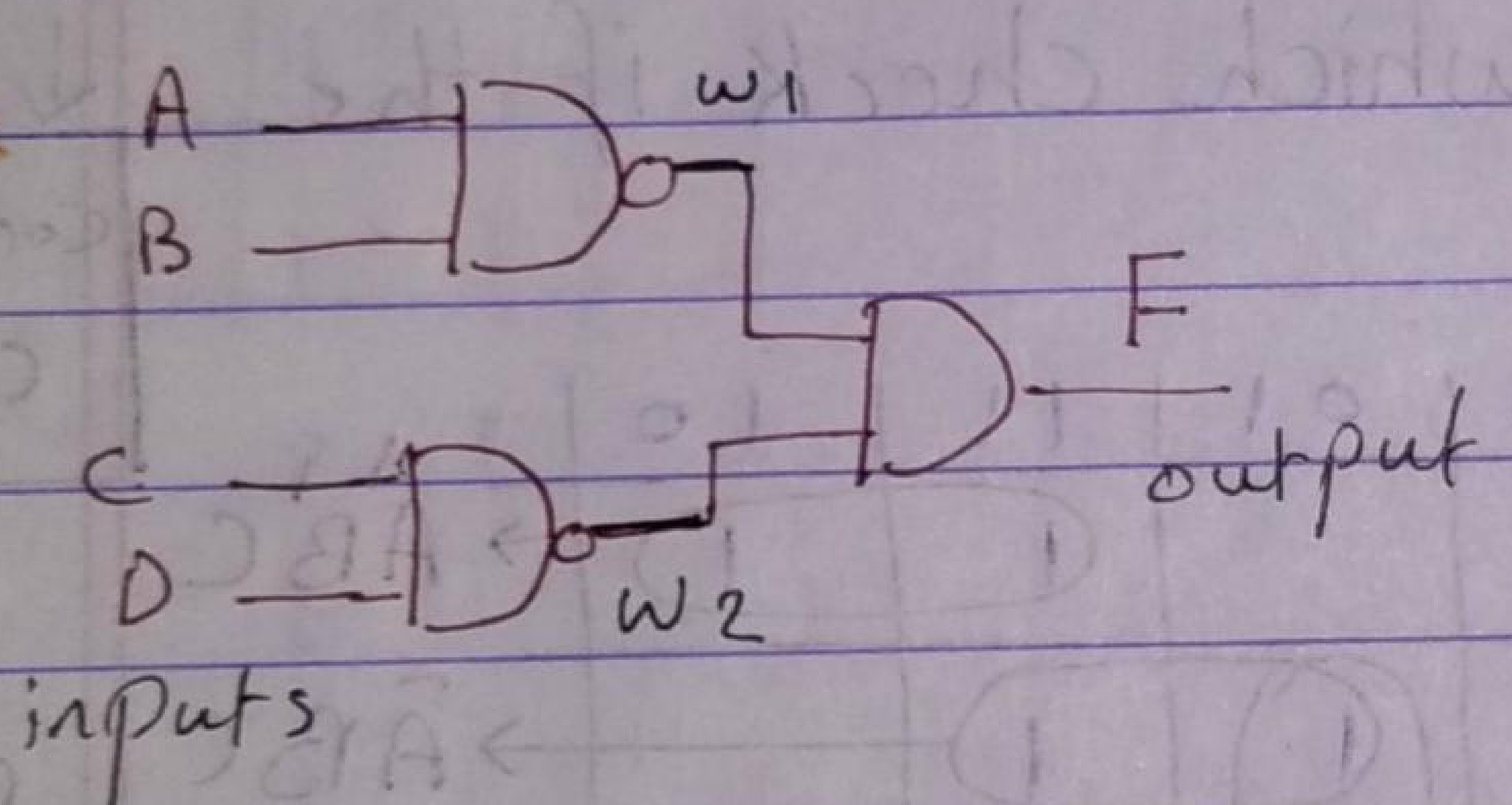


$w_1 = A \cdot B$.

$F = w_1 + C$.

$= \boxed{(A \cdot B) + C}$ → Analysis.

ex



$w_1 = (A \cdot B)'$

$= A' + B'$

$w_2 = (C \cdot D)'$

$= C' + D'$.

$F = w_1 \cdot w_2$

$F = (A' + B') \cdot (C' + D')$.

$F = \boxed{A'C' + A'D' + B'C' + B'D'}$. ← Analysis .

Scanned by TapScanner

Design Procedure :-

① - Read the problem carefully.

② - Determine the number of inputs and number of outputs

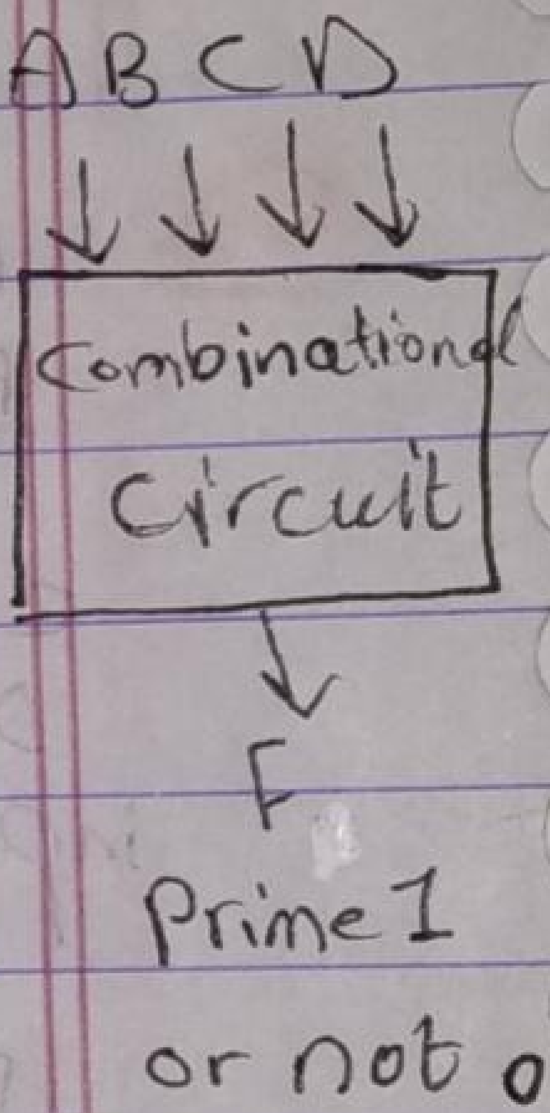
③ - Assign letter symbols to the input & output.

④ - Derive the truth table.

⑤ - simplify using k-map.

⑥ - Draw the circit.

ex Design a combinational circit that takes 4 bit input number which check if the number is prime ?



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

AB \ CD	00	01	11	10
00			1	1
01		1	1	
11		1		
10			1	

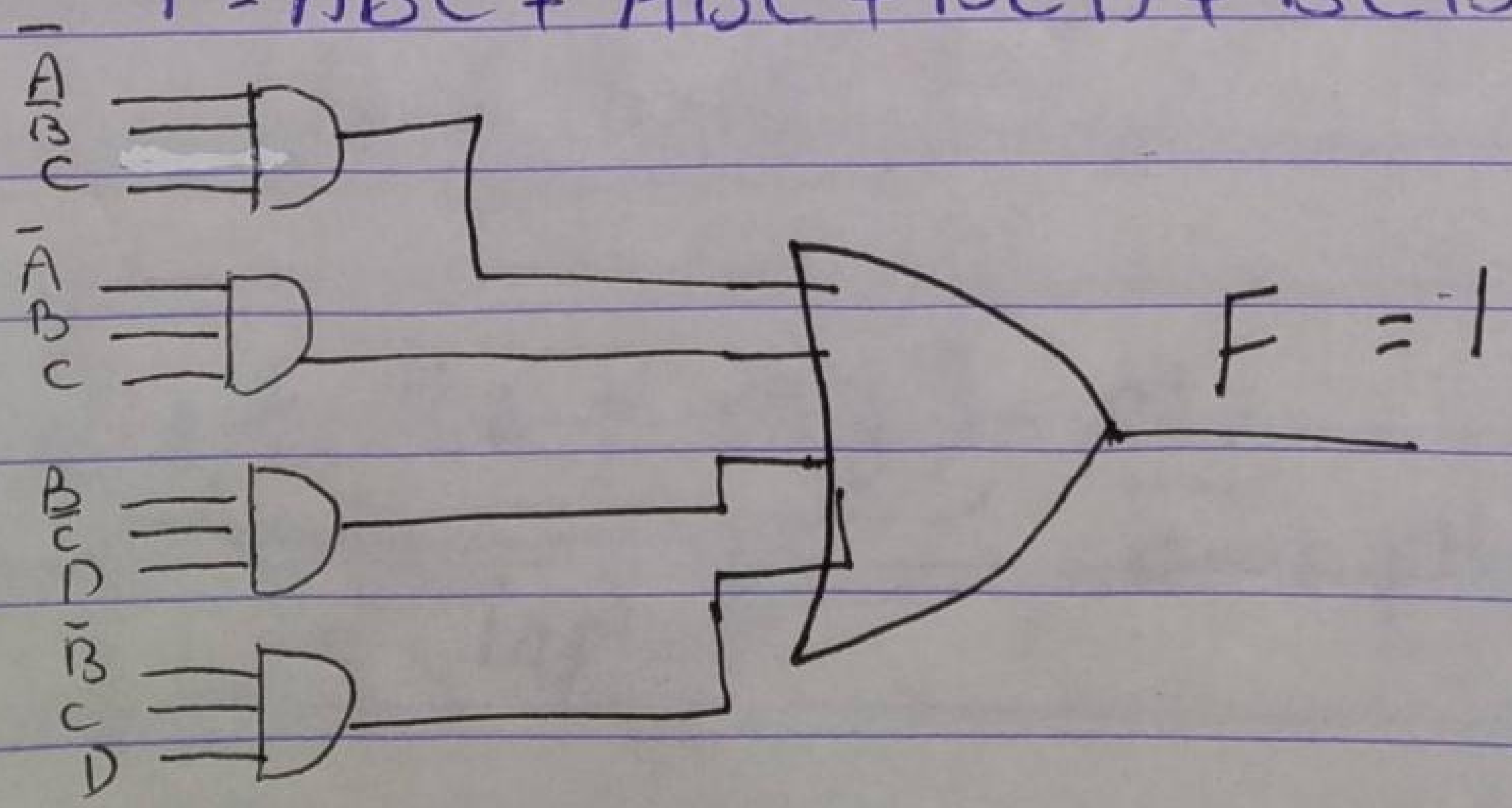
$\rightarrow \bar{A}\bar{B}C$

$\rightarrow \bar{A}BC$

$B\bar{C}D \leftarrow$

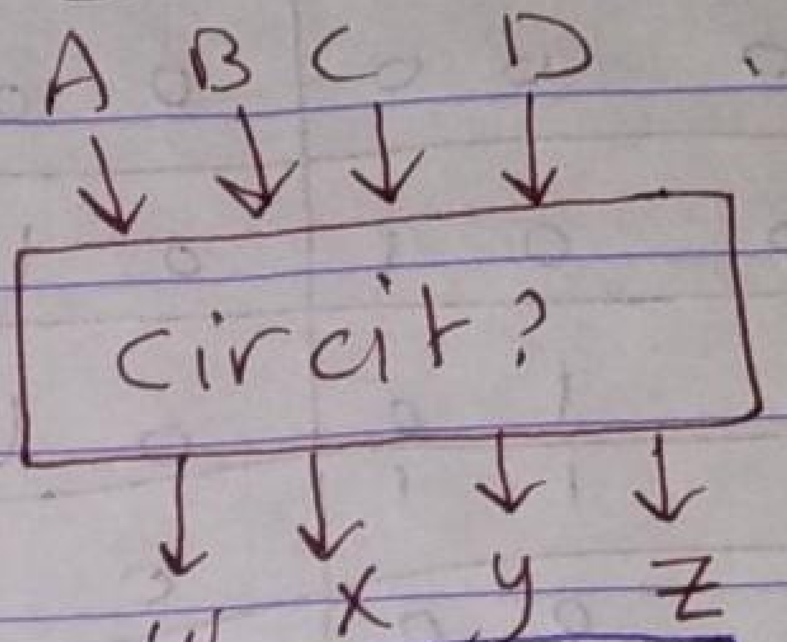
$\rightarrow \bar{B}CD$

$$F = \bar{A}\bar{B}C + \bar{A}BC + B\bar{C}D + \bar{B}CD .$$



ex Design acombinational circit that gererate the 9's complement of a BCD digit ?

A	B	C	D	W	X	y	Z
0	0	0	0	1	0	0	1
0	0	0	1	1	0	0	0
0	0	1	0	01	1	1	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	0	0
0	1	1	0	0	0	1	1
0	1	1	1	0	0	1	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0
x	x	x	x	x	x	x	x



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

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

$X = B\bar{C} + \bar{B}C$

$\bar{B}C$

$B\bar{C}$

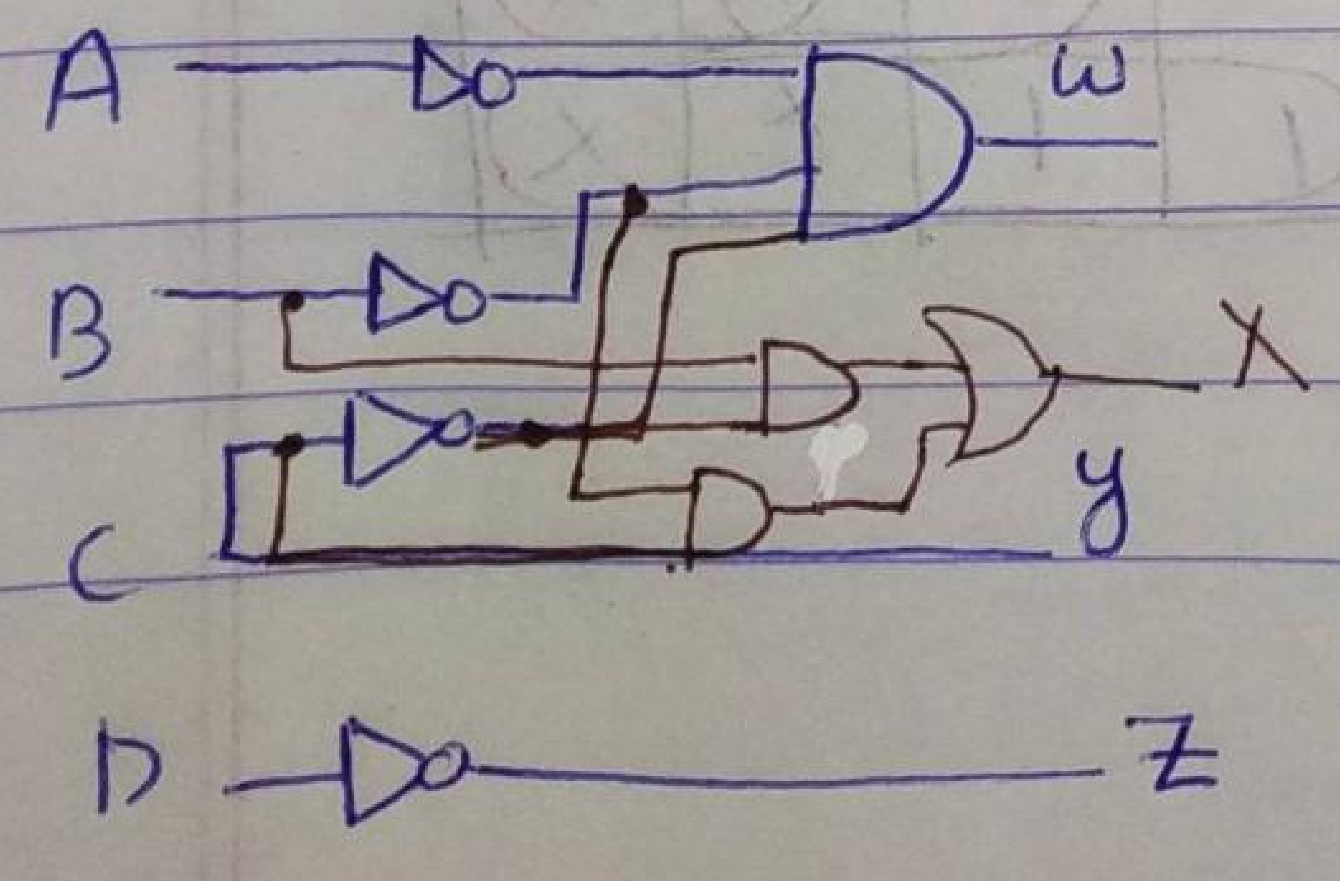
AB \ CD	00	01	11	10
00			1	1
01	1	1		
11	X	X	X	X
10			X	X

$y = C$

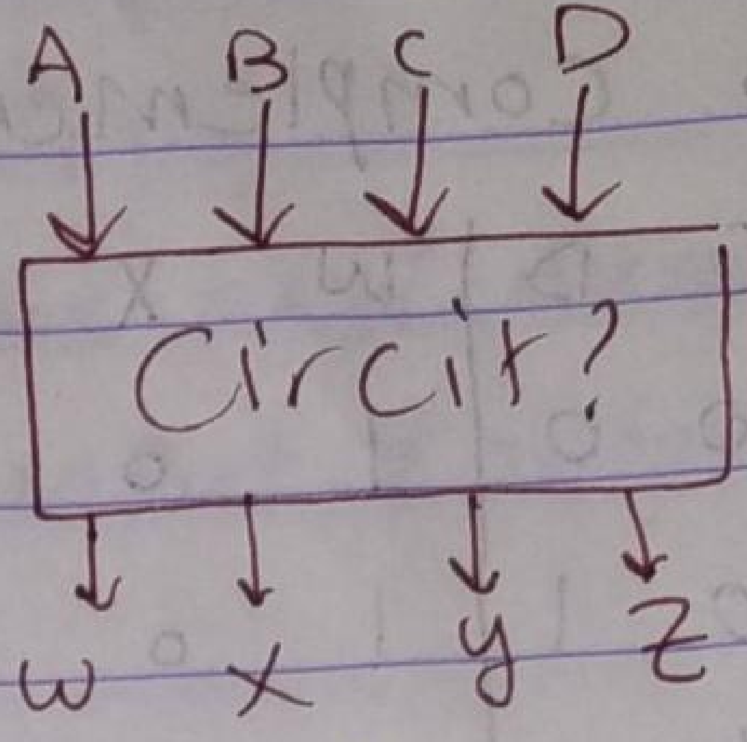
AB \ CD	00	01	11	10
00			1	1
01			1	1
11	X	X	X	X
10			X	X

$Z = \bar{D}$

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



ex Design a BCD to excess-3 Code Conversion?



A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
x	x	x	x	x	x	x	y
x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x
x	x	x	x		x	x	x

$Z = D'$

	00	01	11	10
00	1			1
01	1			1
11	X	X	X	X
10	1		X	X

$y = C'D' + CD$

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

$X = B'C + B'D + BC'D'$

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

$\bar{B}C$

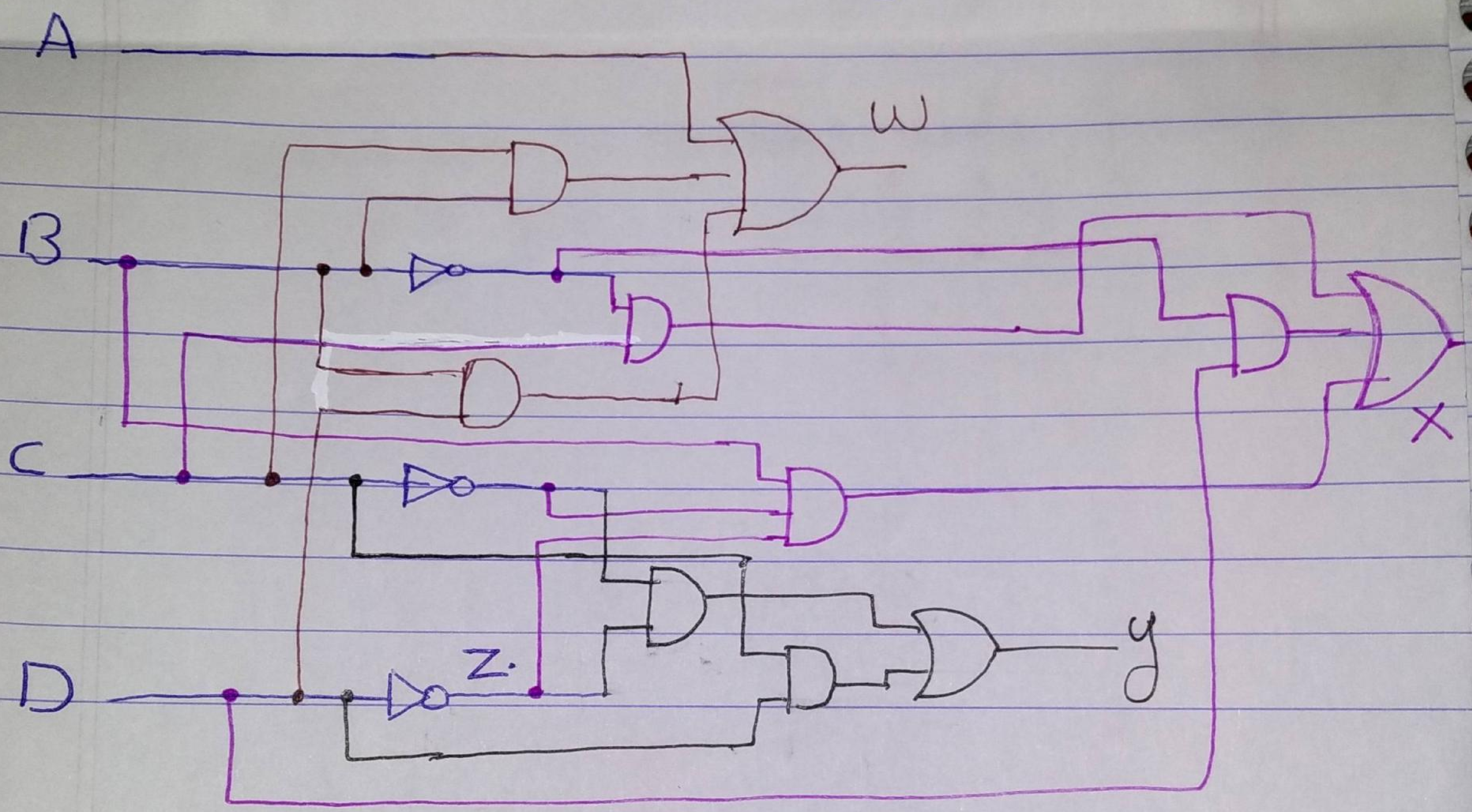
$\bar{B}D$

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

$w = A + BC + BD$

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

الرسمة للمثال السابق .



Scanned by TapScanner

Binary Adder & Subtractor :-

4 Possible operation for addition two Binary digit.

$\begin{array}{r} 0 \\ +0 \\ \hline 0\;0 \end{array}$ Carry sum

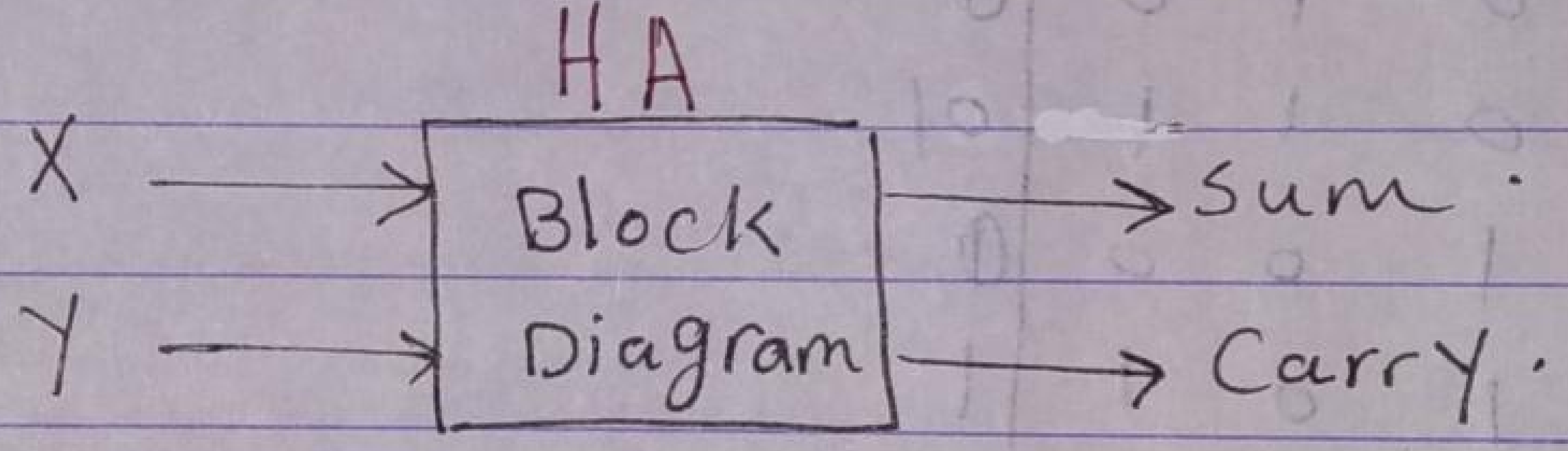
$\begin{array}{r} +0 \\ 1 \\ \hline 0\;1 \end{array}$ Carry sum

$\begin{array}{r} +1 \\ 0 \\ \hline 0\;1 \end{array}$ carry sum

$\begin{array}{r} +1 \\ 1 \\ \hline 1\;0 \end{array}$ Carry sum.

Half Adder (HA) :- Combinational curcuit that perform addition of two digits.

X	y	sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



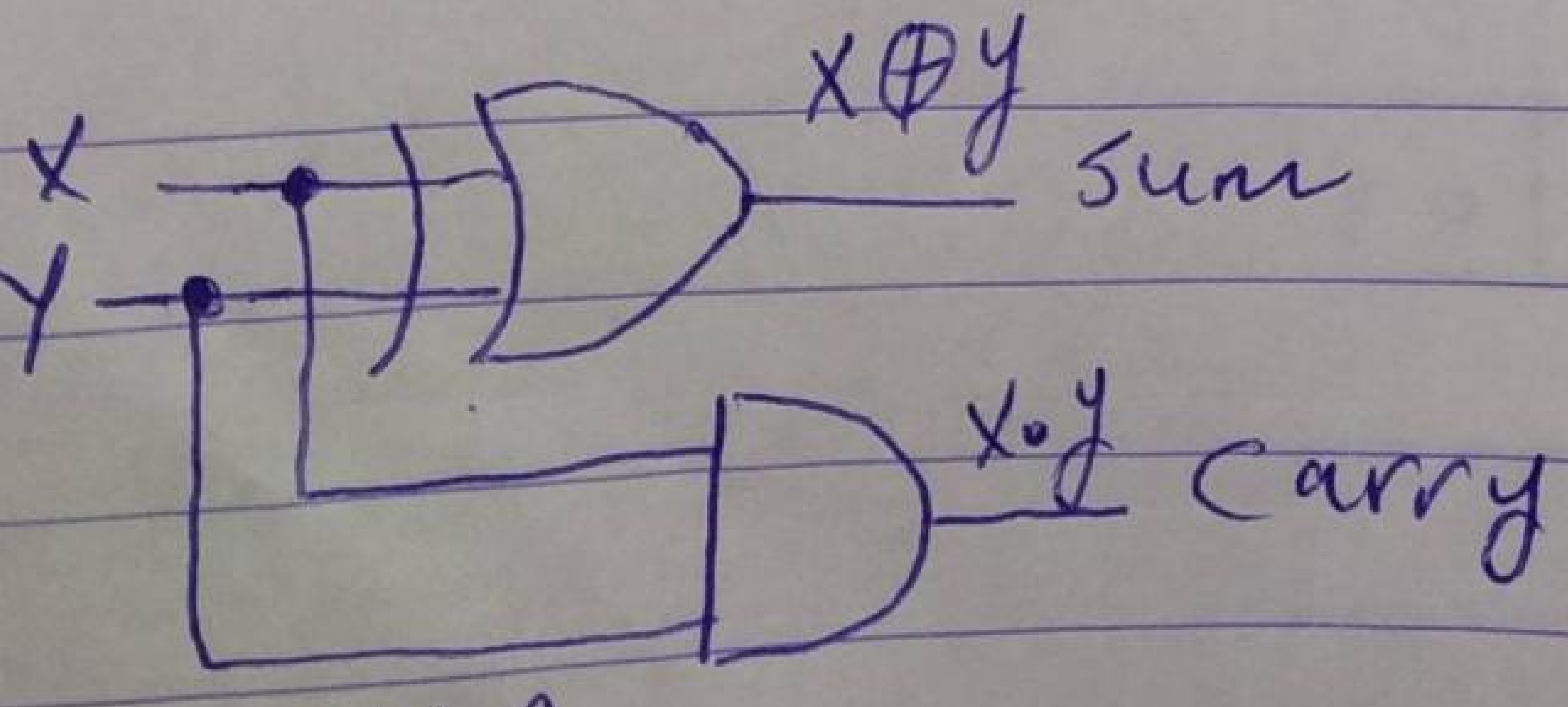
X \ Y	0	1
0		1
1	1	

$Sum = X'y + XY'$

$Sum = X \oplus Y$

X \ Y	0	1
0		
1		1

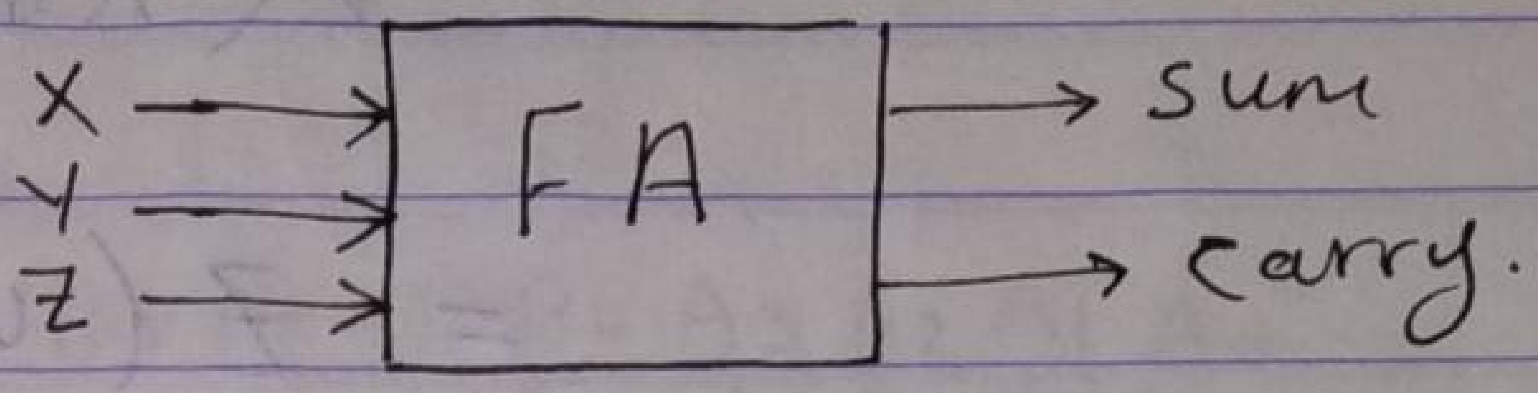
$Carry = X \cdot y$



HA

Full Adder (FA) :- Combinational circuit that perform addition of three bits.

x	y	z	sum	carry
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

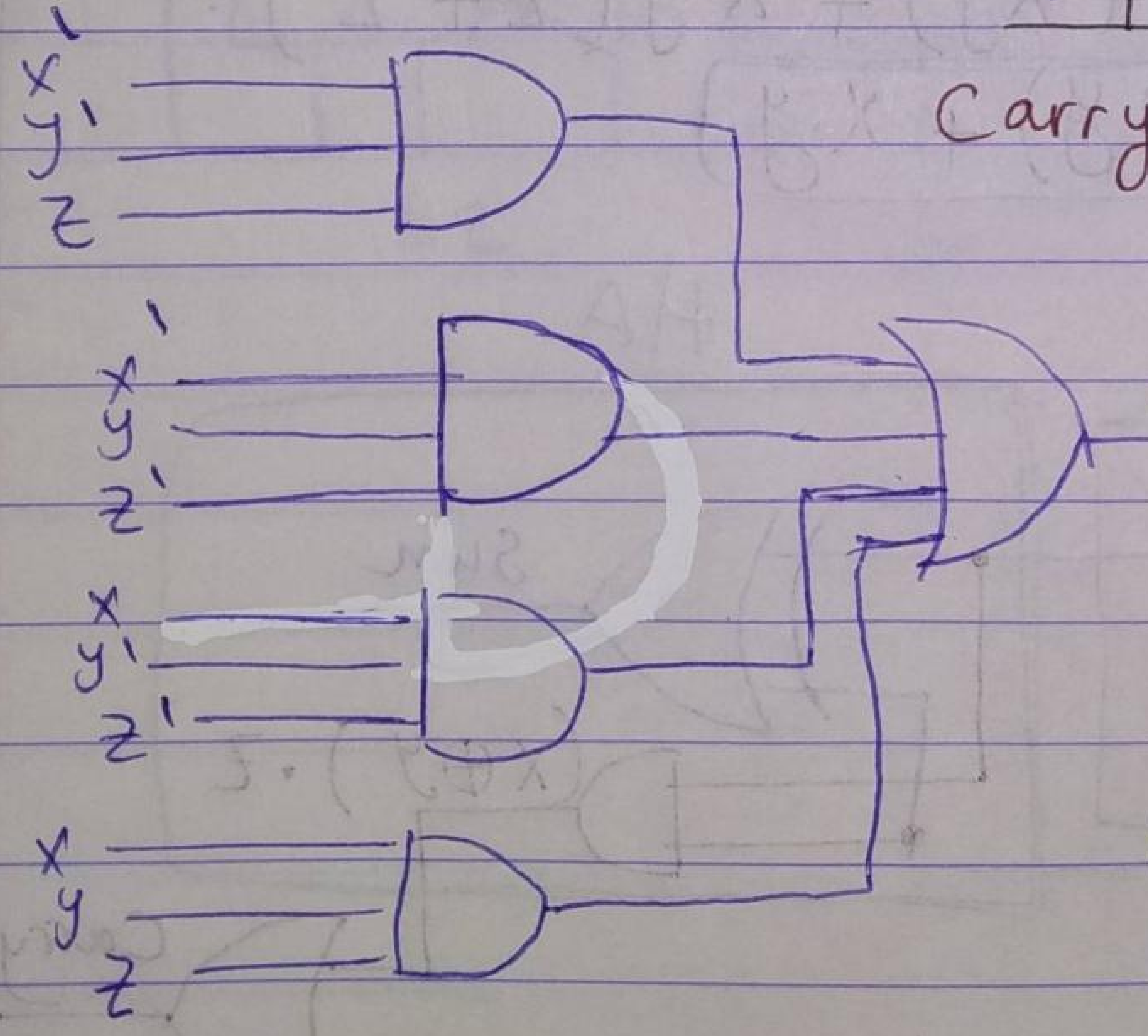


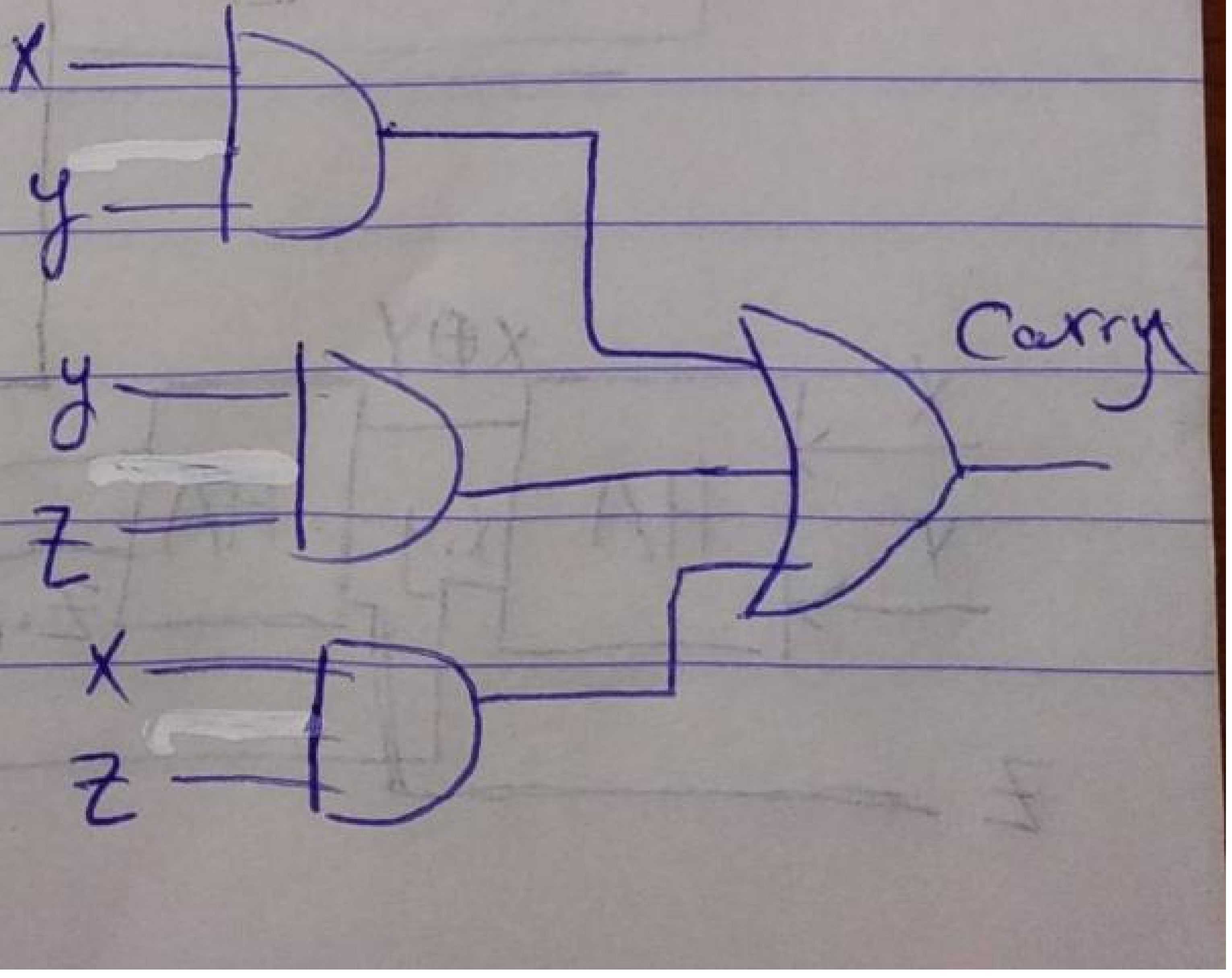
x \ yz	00	01	11	10
0		1		1
1	1		1	

Sum = $x'y'z + x'yz' + xy'z' + xyz$

x \ yz	00	01	11	10
0			1	
1	.	1	1	1

Carry = $x.y + y.z + x.z$





Scanned by TapScanner

للتبسيط :-

تكملة ⟹ المثال

$$Sum = z(x'y' + xy) + z'(x'y + xy')$$

$$= z(x \oplus y)' + z'(x \oplus y).$$

$$= z(w)' + z'(w).$$

$$= z \oplus w$$

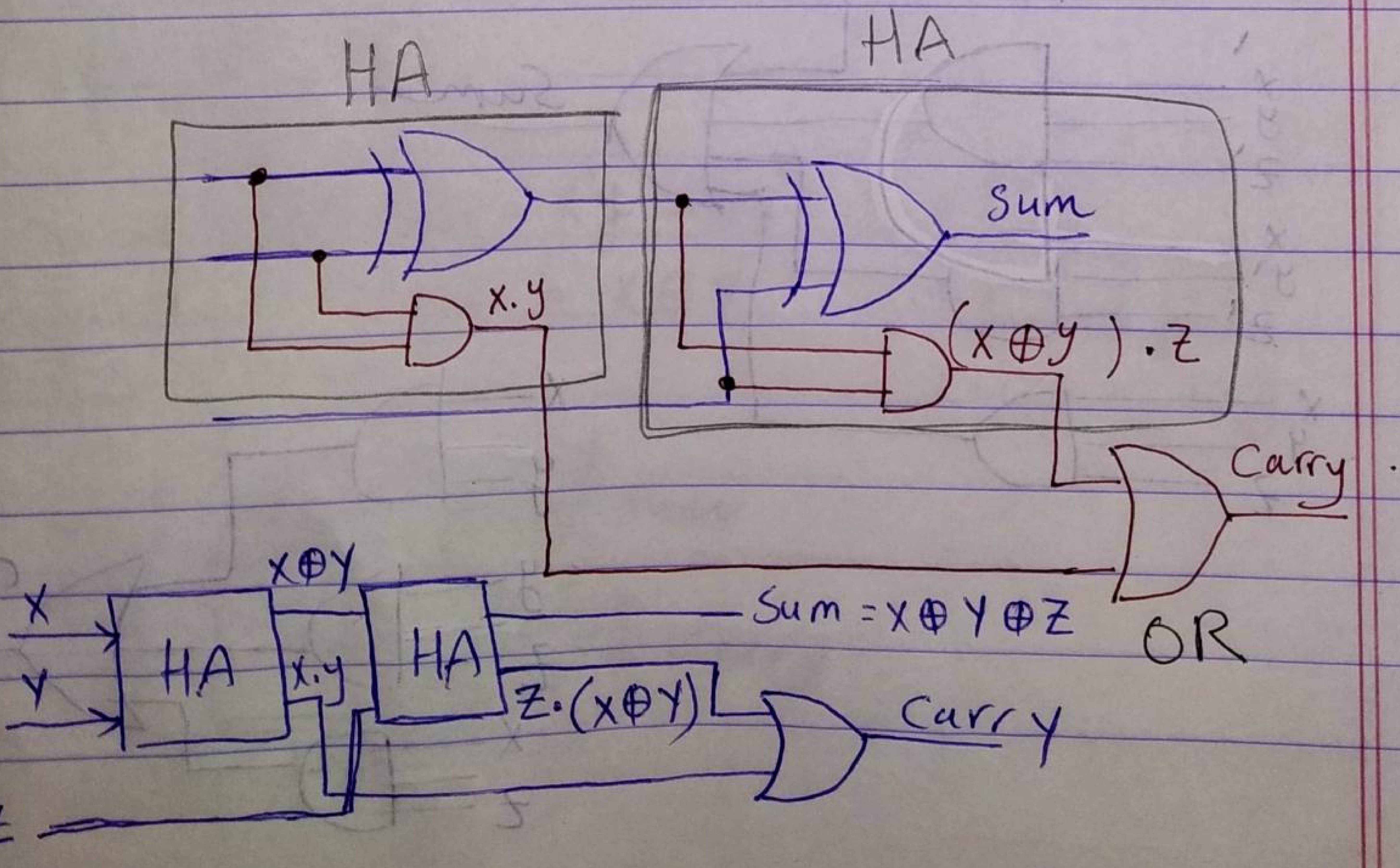
$$\boxed{sum} = z \oplus x \oplus y = \boxed{x \oplus y \oplus z}$$

Carry = بنوخذ كل رقم لحاله

$$Carry = x'\underline{yz} + x\underline{y'z} + xyz + xyz'.$$

$$= z(x'y + xy') + x.y\,\overset{1}{\cancel{(z + z')}}.$$

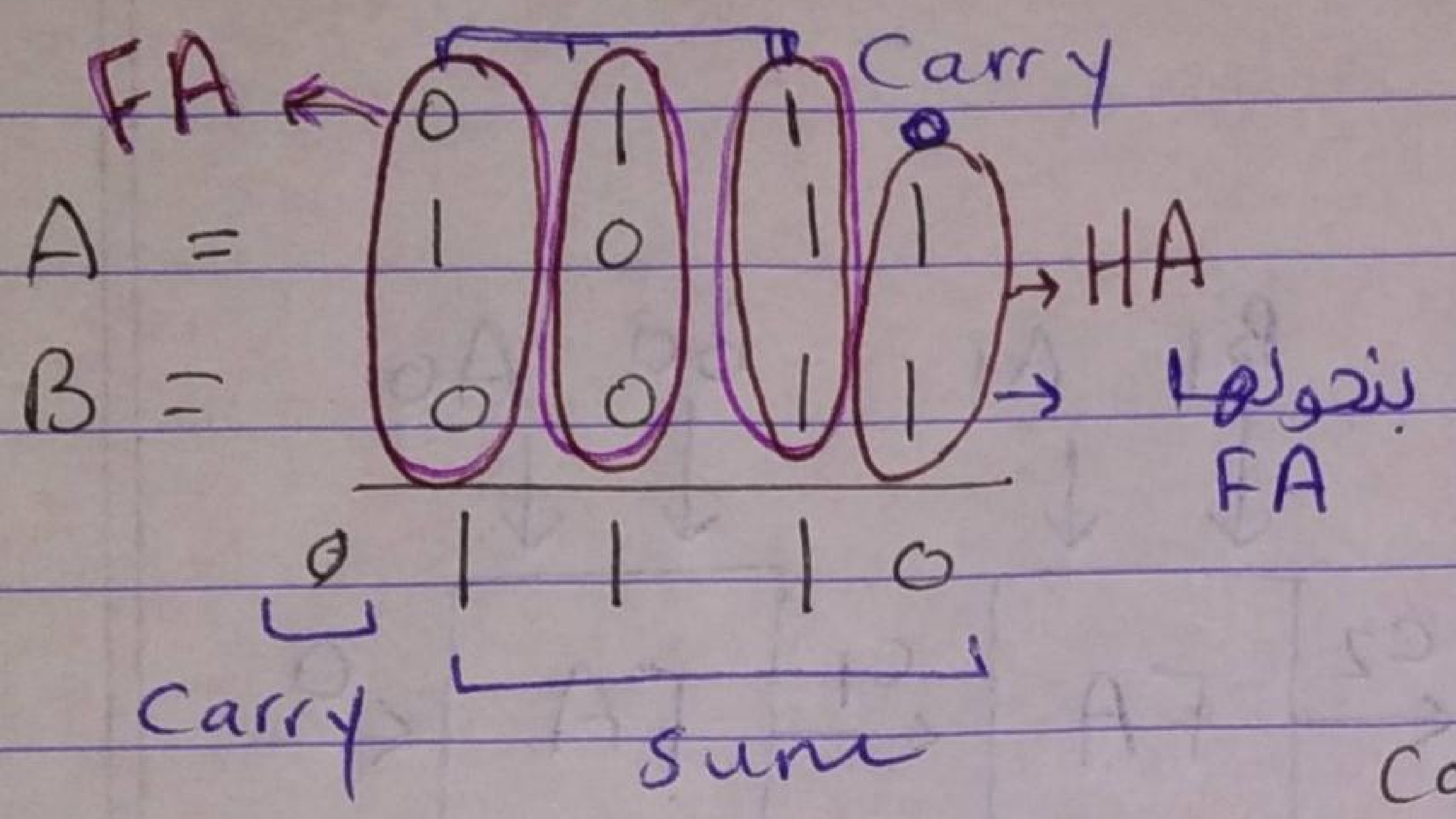
$$\boxed{Carry = z.(x \oplus y) + x.y}$$

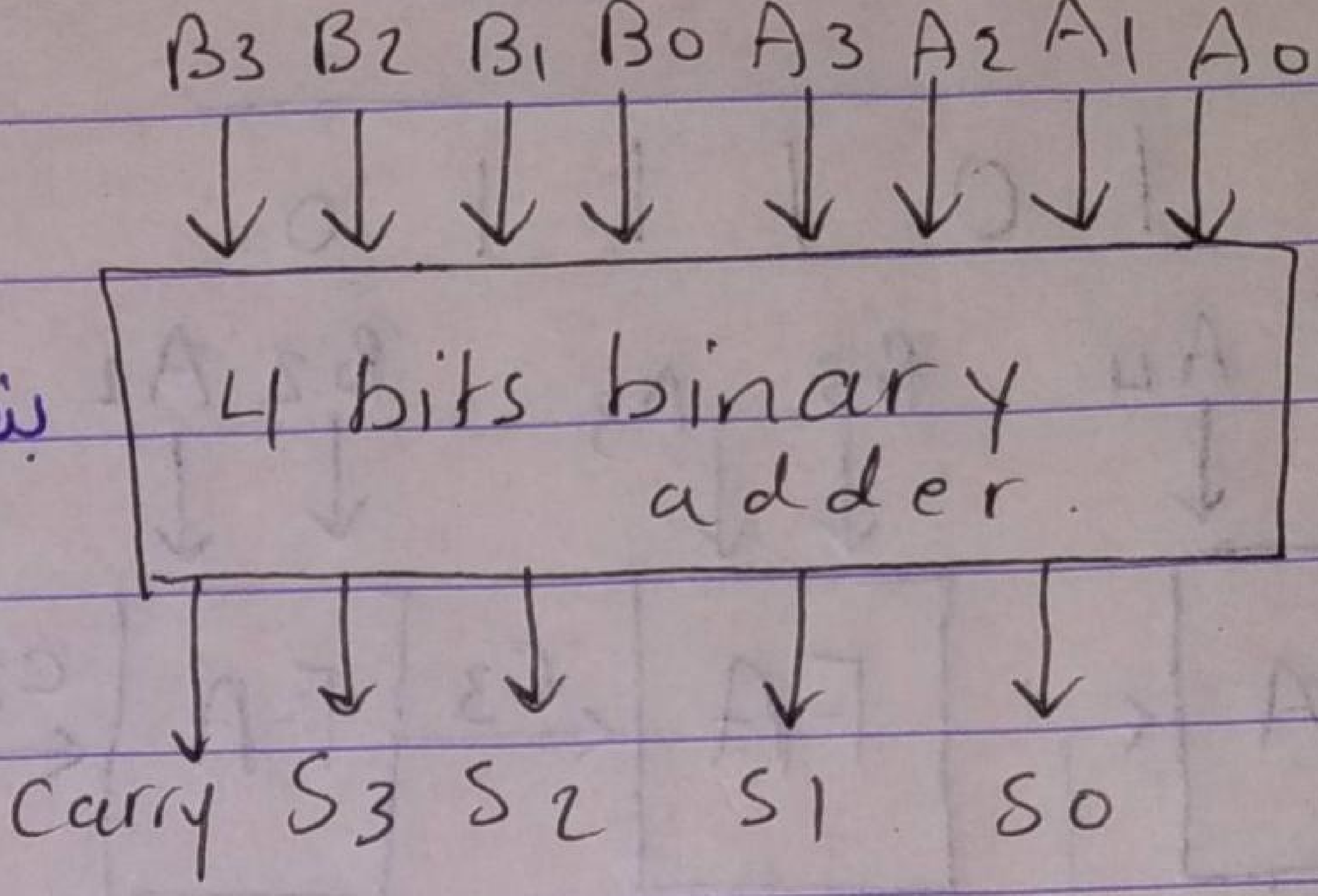


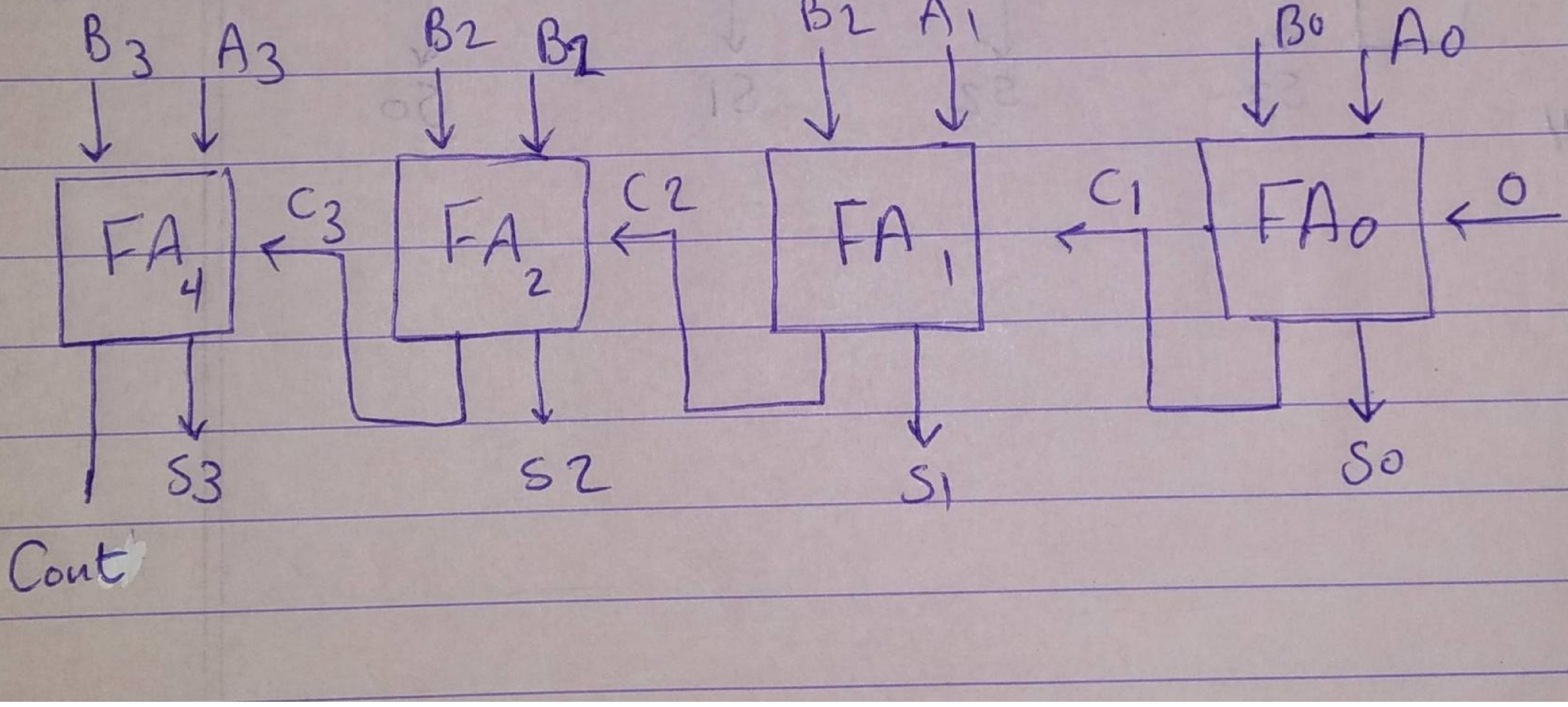
Binary Adder :- 4 bits binary adder.

ex $A = A_3 A_2 A_1 A_0$.

$B = B_3 B_2 B_1 B_0$.

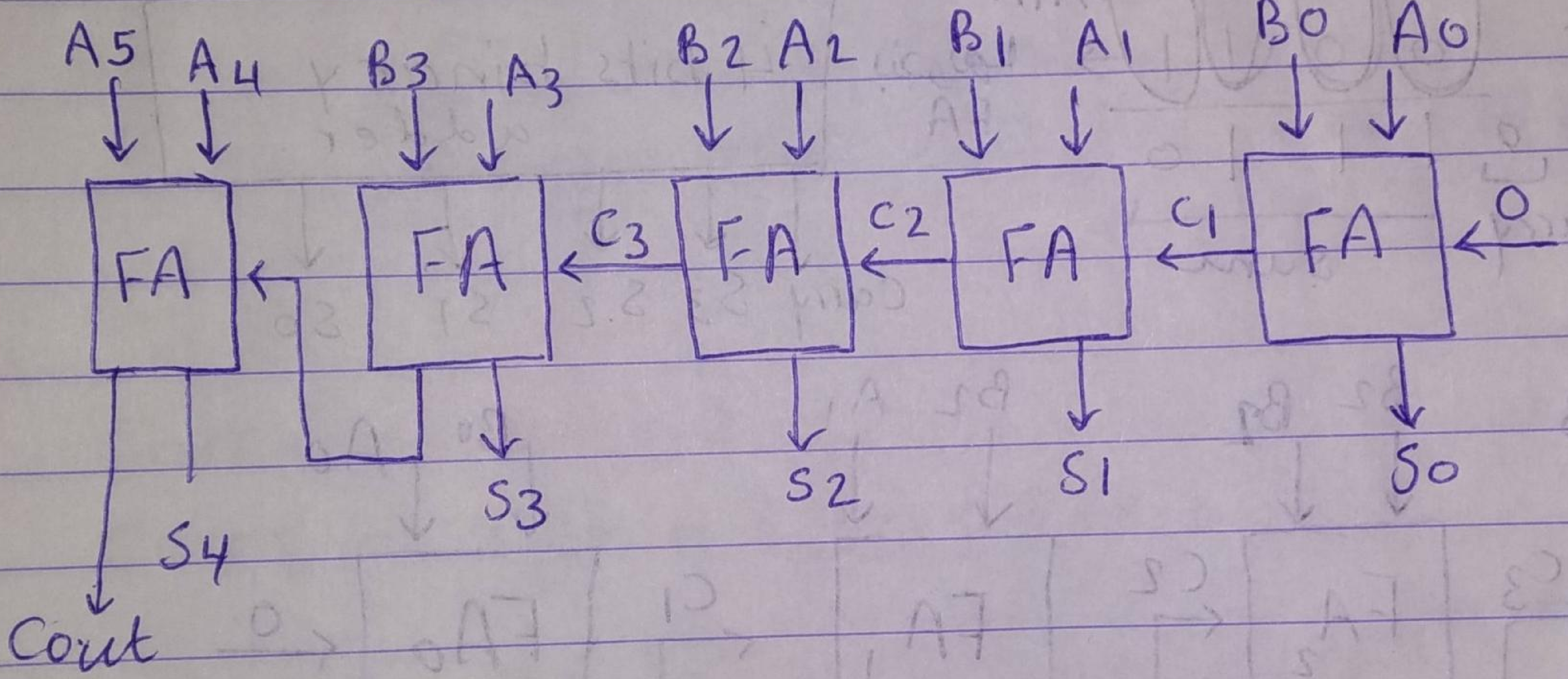






* Design 5 bits binary adder:-

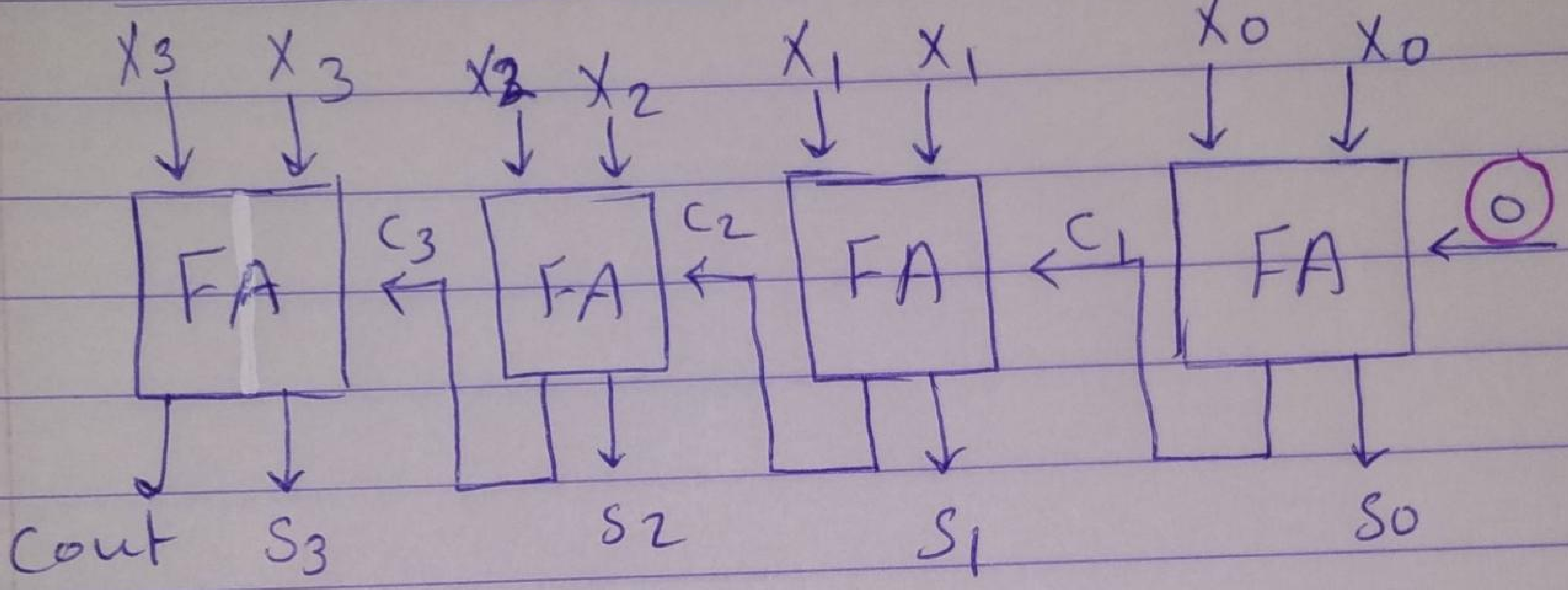
```
         1   1   1   0
     1   0   1   1   1    FA
 +   1   0   1   1   1
 ---------------------
 1   0   1   1   1   0
```



Scanned by TapScanner

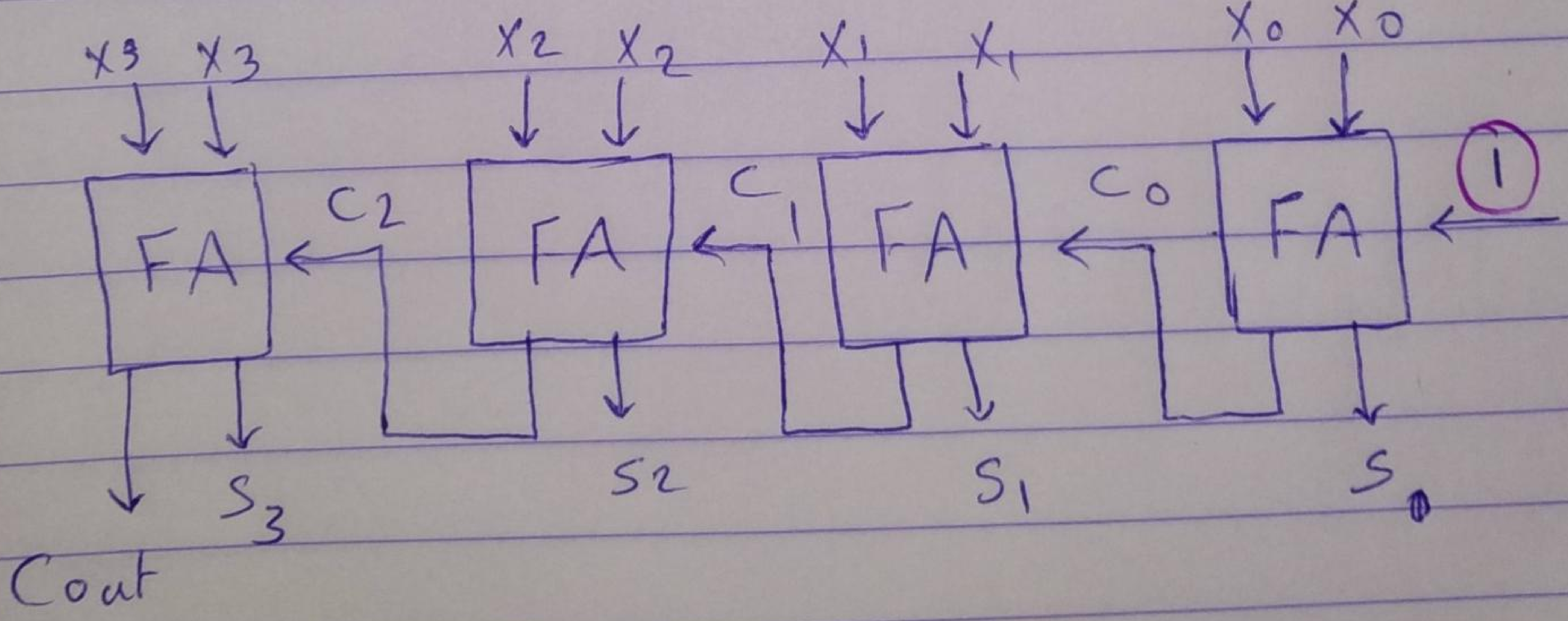
* Design Circuit that do the following operation 2X $\hookrightarrow$ = X + X. X = 4 bits.

$$= \begin{array}{r} X_3\ X_2\ X_1\ X_0 \\ +\ X_3\ X_2\ X_1\ X_0 \\ \hline \end{array}$$



ex Design acircit that do the following. X = 4 bits.

2X + 1



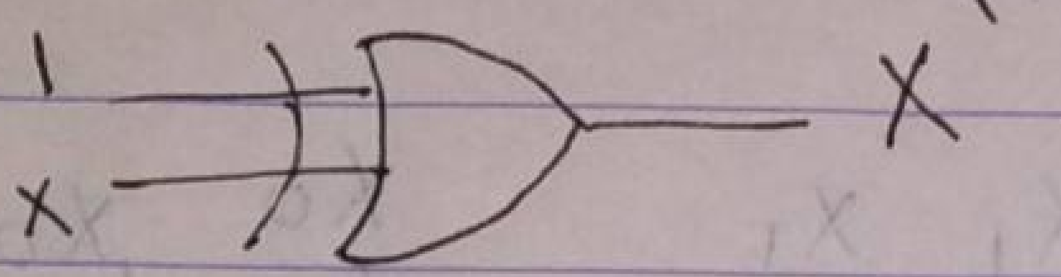
Scanned by TapScanner

Binary Subtractor

$A - B = A + 2's$ Compl.

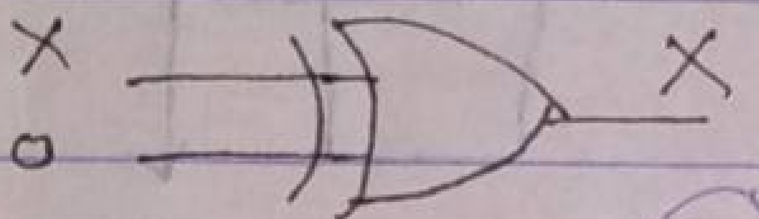
$= A + (1^{st}$ Compl $+ 1)$.

Note



$1 \oplus x \Rightarrow 1 \oplus 0 = 1$

$\quad \Rightarrow 1 \oplus 1 = 0$ x'



$1 \oplus 0 = 1$

$0 \oplus 0 = 0$

ex Design 4 bit binary subtraktor

A = 1011 $\qquad$ S = A - B = A + 2's compl.

B = 0011 $\qquad\qquad\qquad$ = A + 1st Compl + 1

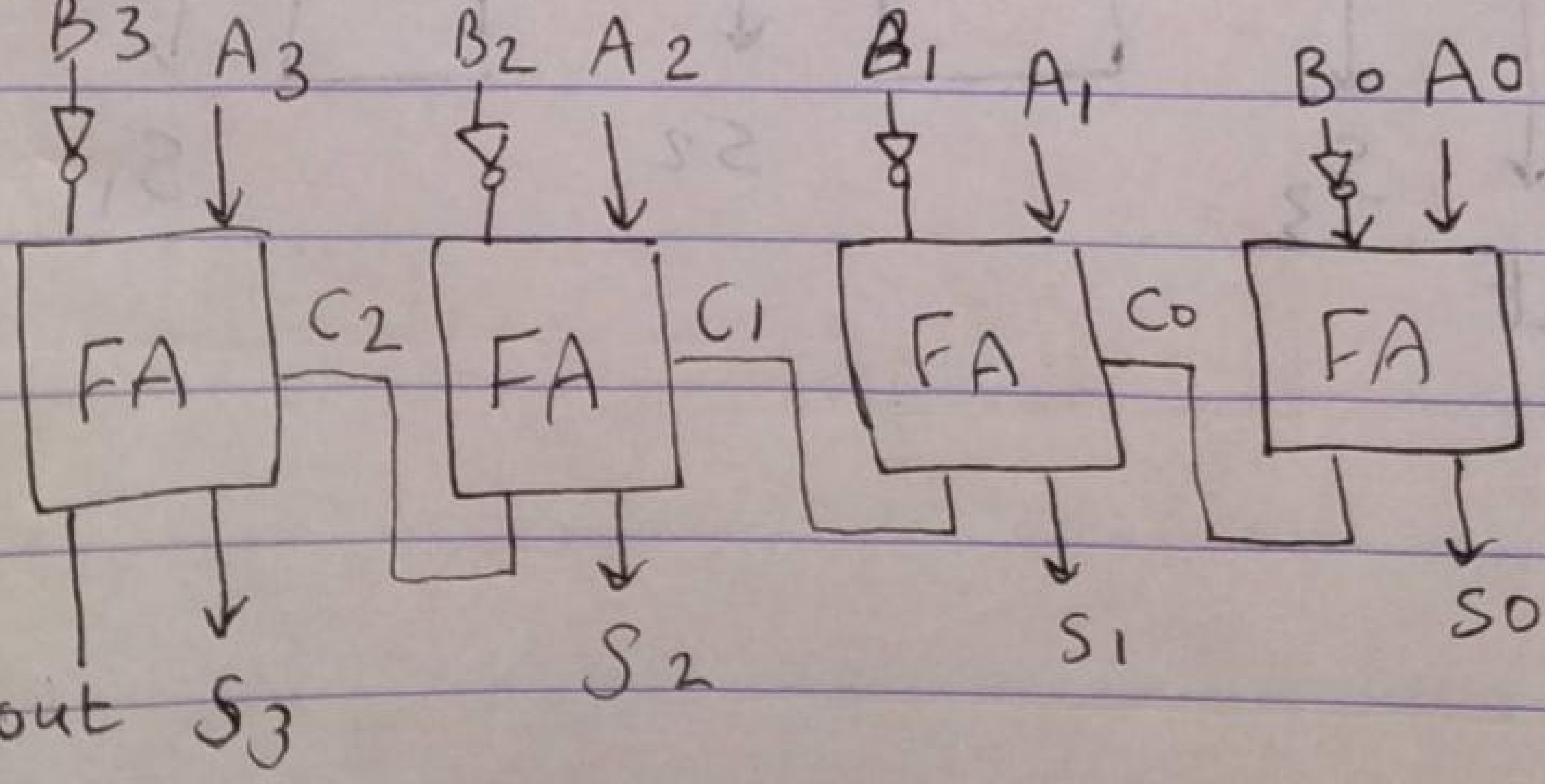
2's compl B = 1100 + 1 = 1101

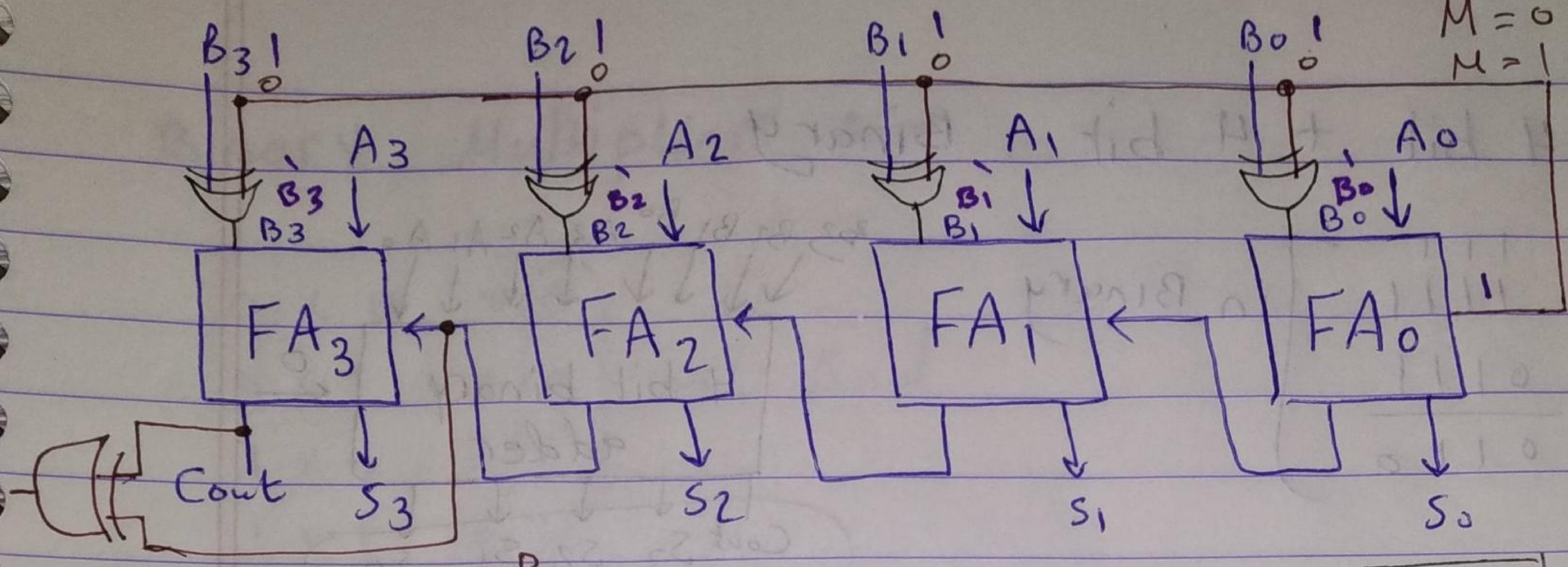
$$\begin{array}{r} {}^{1\,1\,1\,0} \\ 1011 \\ +\ 1101 \\ \hline 11000 \end{array}$$

A + B' + 1

A + 2's Compl

A - B'





A + B

$M = 0$ $A_3\ A_2\ A_1\ A_0$

Adder $B_3\ B_2\ B_1\ B_0$

Adder.

$M = 1$ $\boxed{A - B}$

$A_3\ A_2\ A_1\ A_0$

$B_3'\ B_2'\ B_1'\ B_0'$

Adder + Subtractor.

بنحط XOR بالآخر عشان نفحص اذا في overflow أو لا .

(إذا كانوا متشابهات ما في over flow واذا مختلفات في) .

Decimal Adder :— (BCD adder).

BCD → 0 - 9 valid 10 - 15 unvalid.

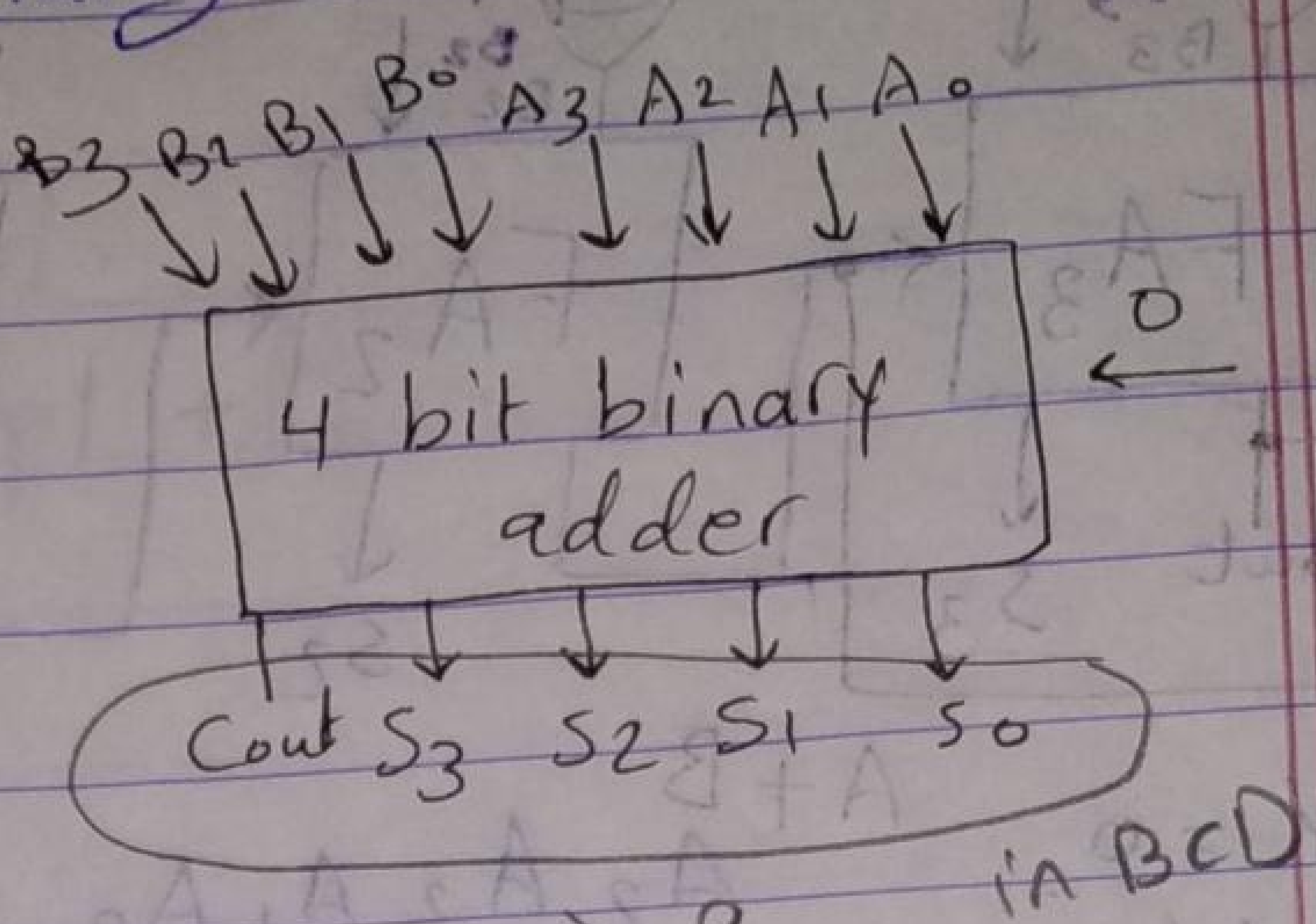
BCD + BCD Check sum < 9 OK.

BCD + BCD check sum > 9 + 6 .

* 4 bit + 4 bit Binary

```
    111
A   1111
B   0111
   10110
   ↓
  Cout
```

in Binary



> 9

+6

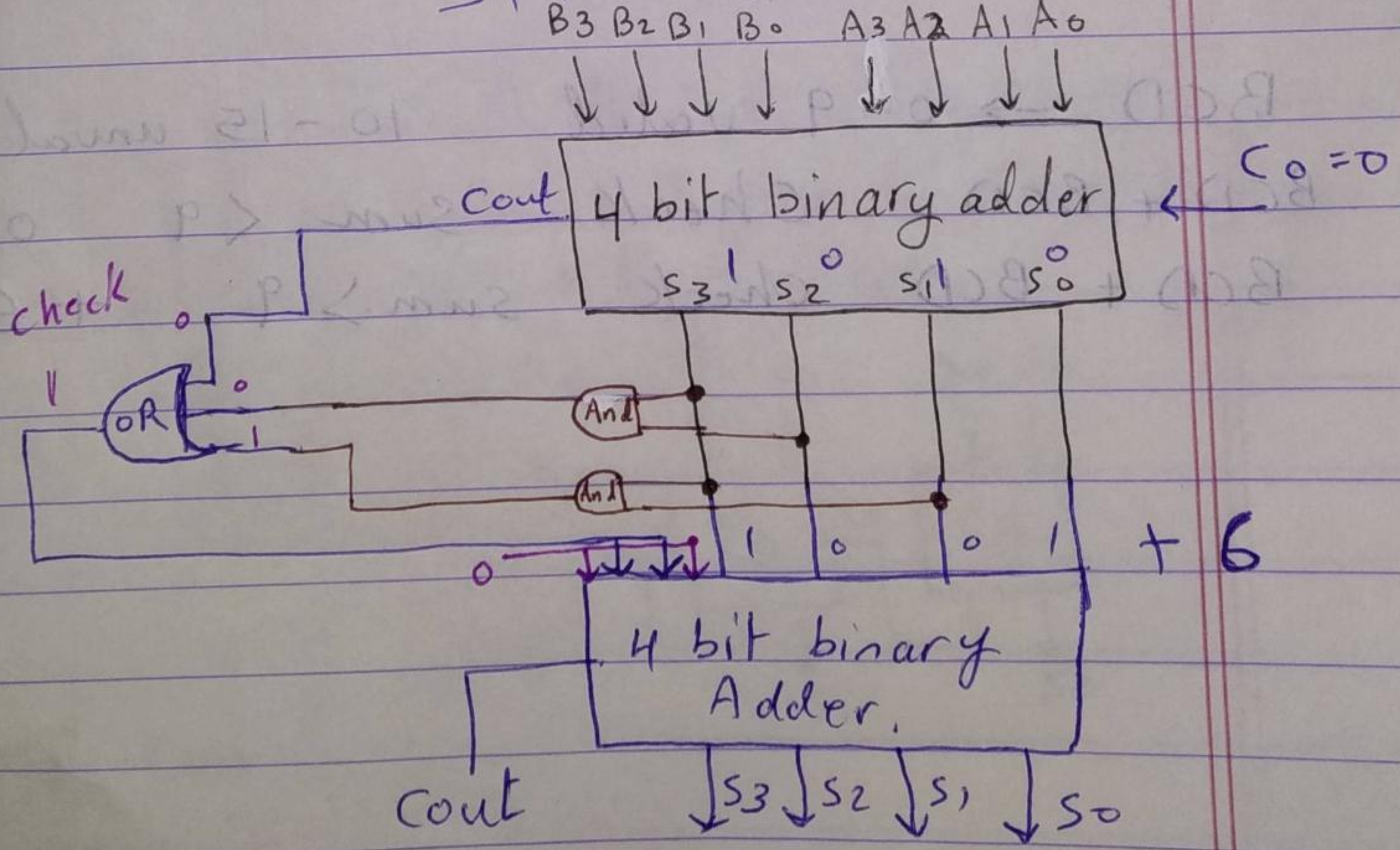
A(BCD) + B(BCD) =

```
    A
 +  B
9 < ① - - - -
```

```
  4    0100
+ 4    0100
      ⓪1000

  9    1001
+ 9    1001
      ①00:10
>9   1 0 0 1    1 0 0 1
```

```
   1
 1001
 1001
10010
 =
```



Scanned by TapScanner

Binary Multiplier :-

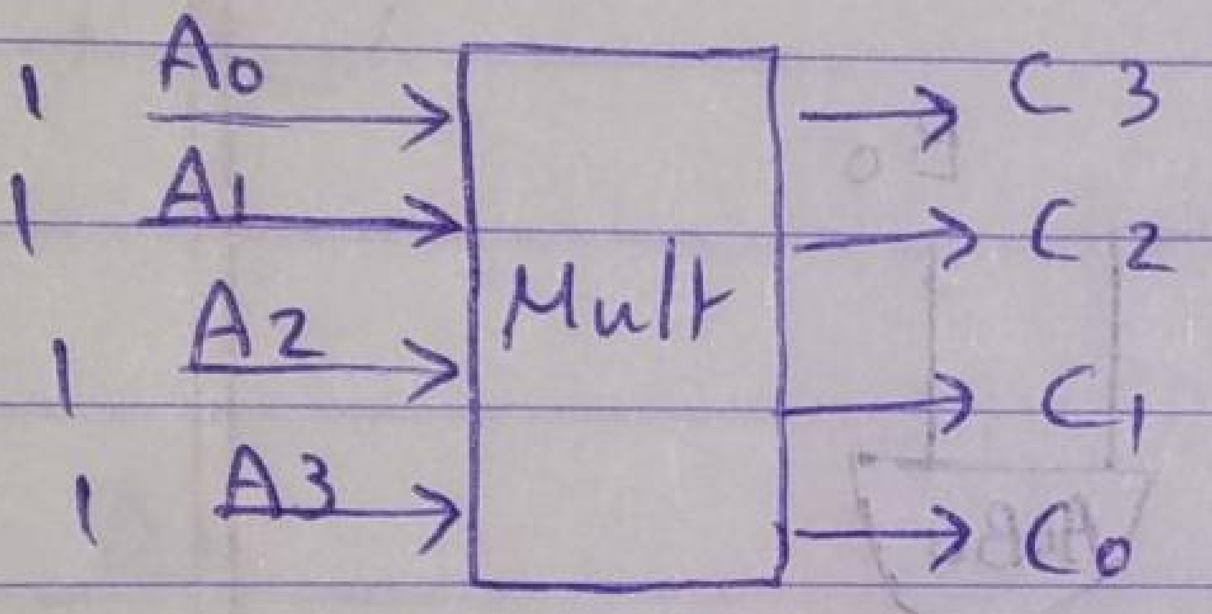
$0 \times 0 = 0$

$0 \times 1 = 0$ ← And.

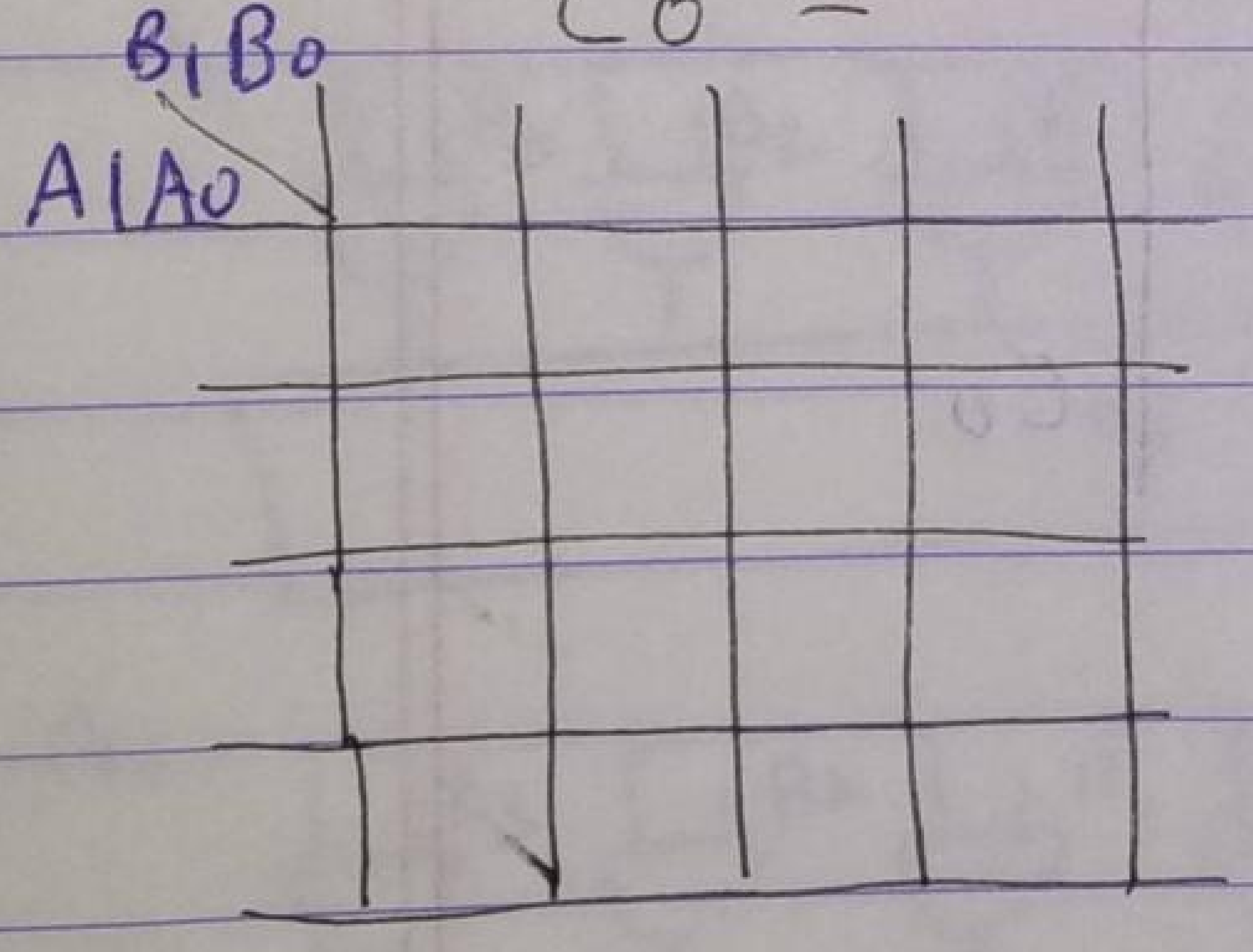
$1 \times 0 = 0$

$1 \times 1 = 1$

ex Design 2 bit by 2 bit binary multiplier.



$C_0 =$



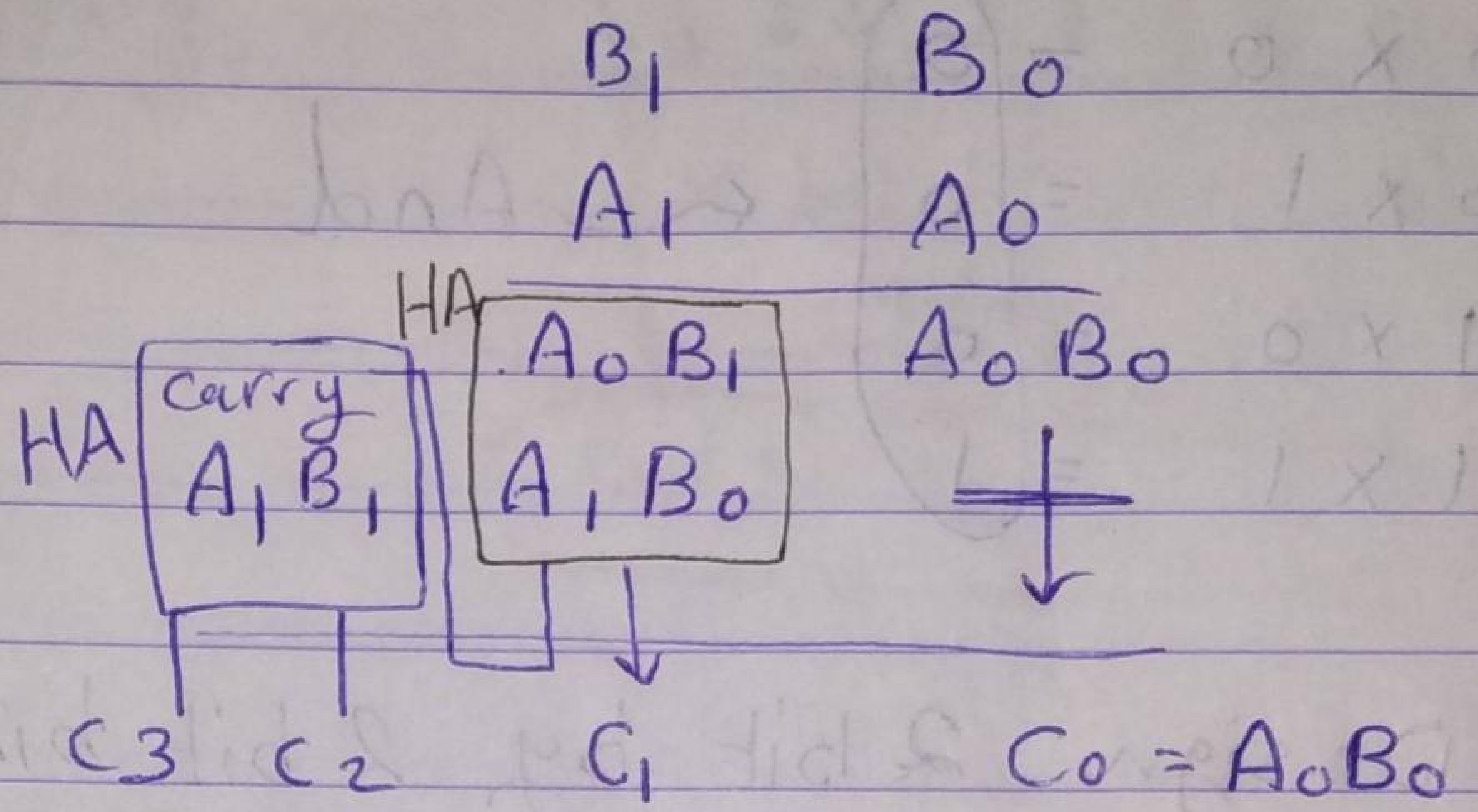
A_1	A_0	B_1	B_0	C_3	C_2	C_1	C_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1				
1	0	0	0				
1	0	0	1				
1	0	1	0				
1	0	1	1				
1	1	0	0				
1	1	0	1				
1	1	1	0				
1	1	1	1	1	0	0	1

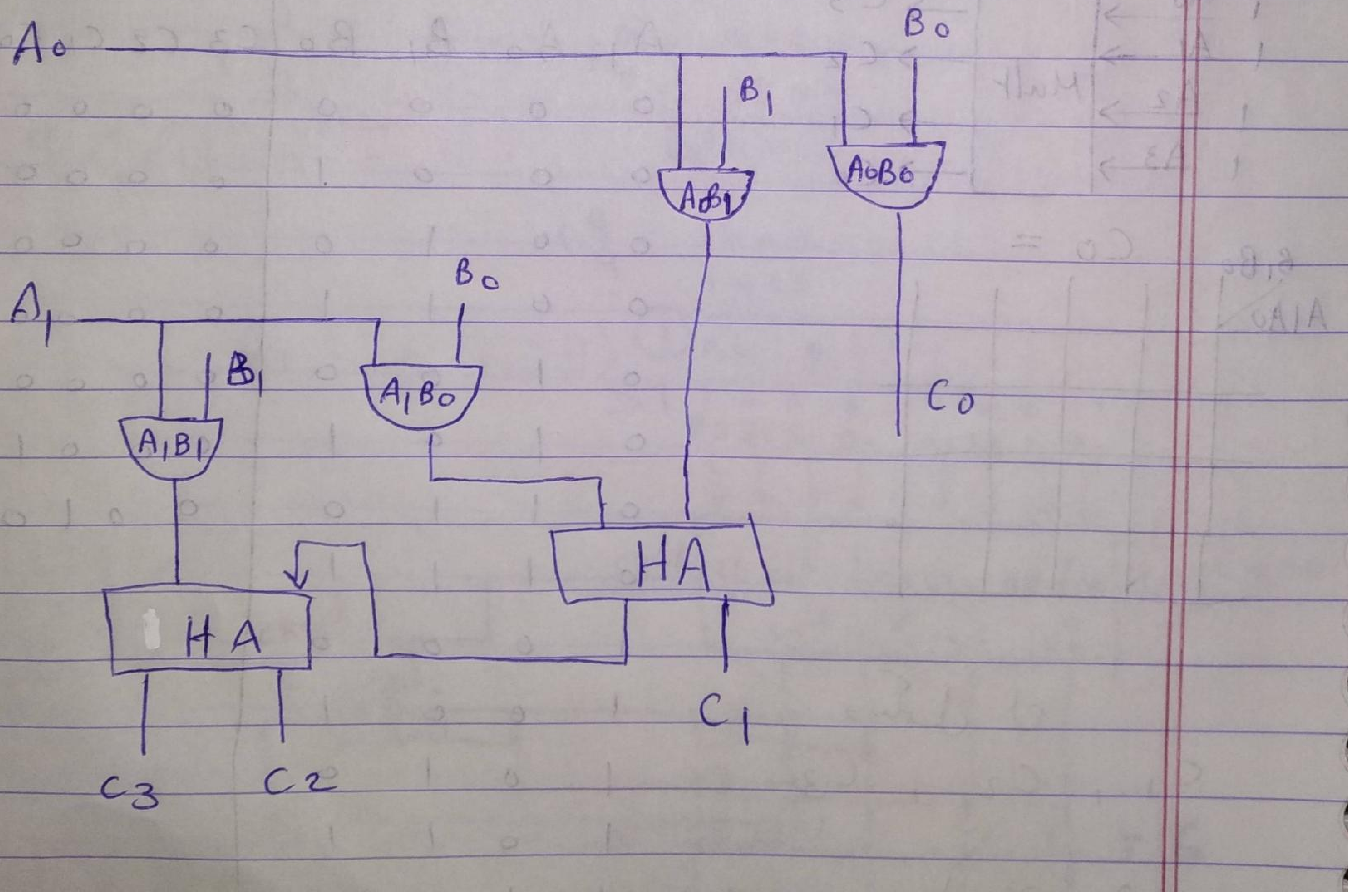
وبنكمل ل

C_1 , C_2 , C_3

بس هاي الطريقة مش
efficient
(فعّالة) .

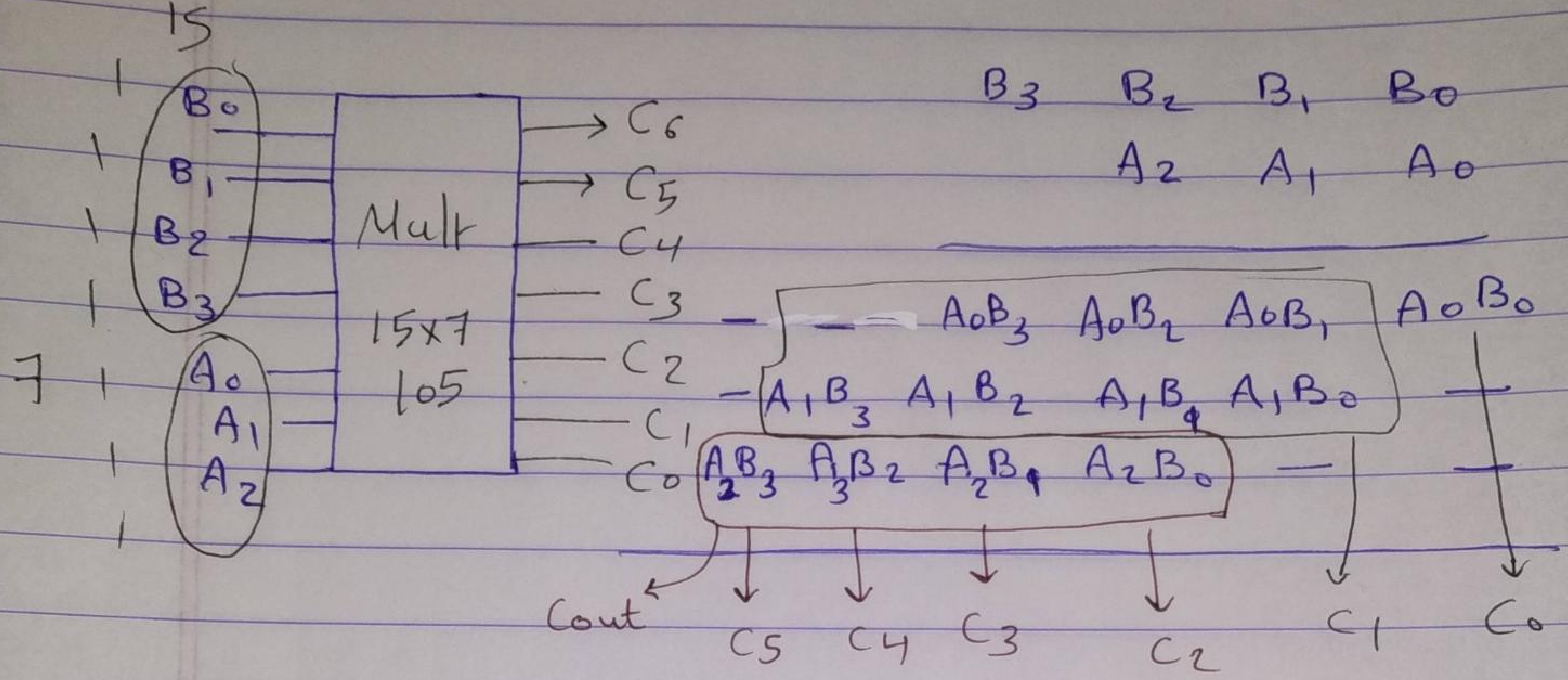
ex Design a 2 bit $\overset{X}{\text{by}}$ 2 bit binary multiplier.

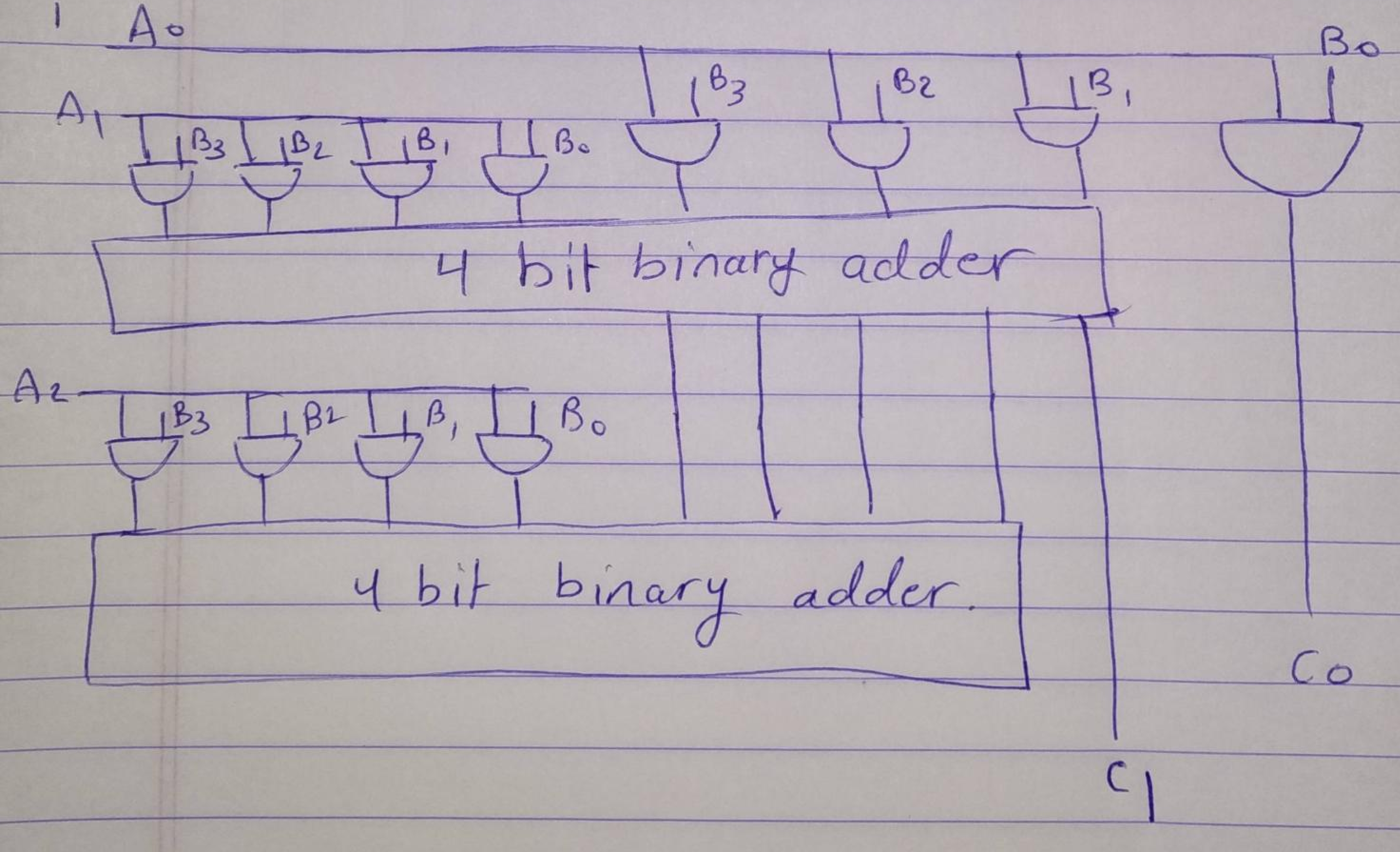




Scanned by TapScanner

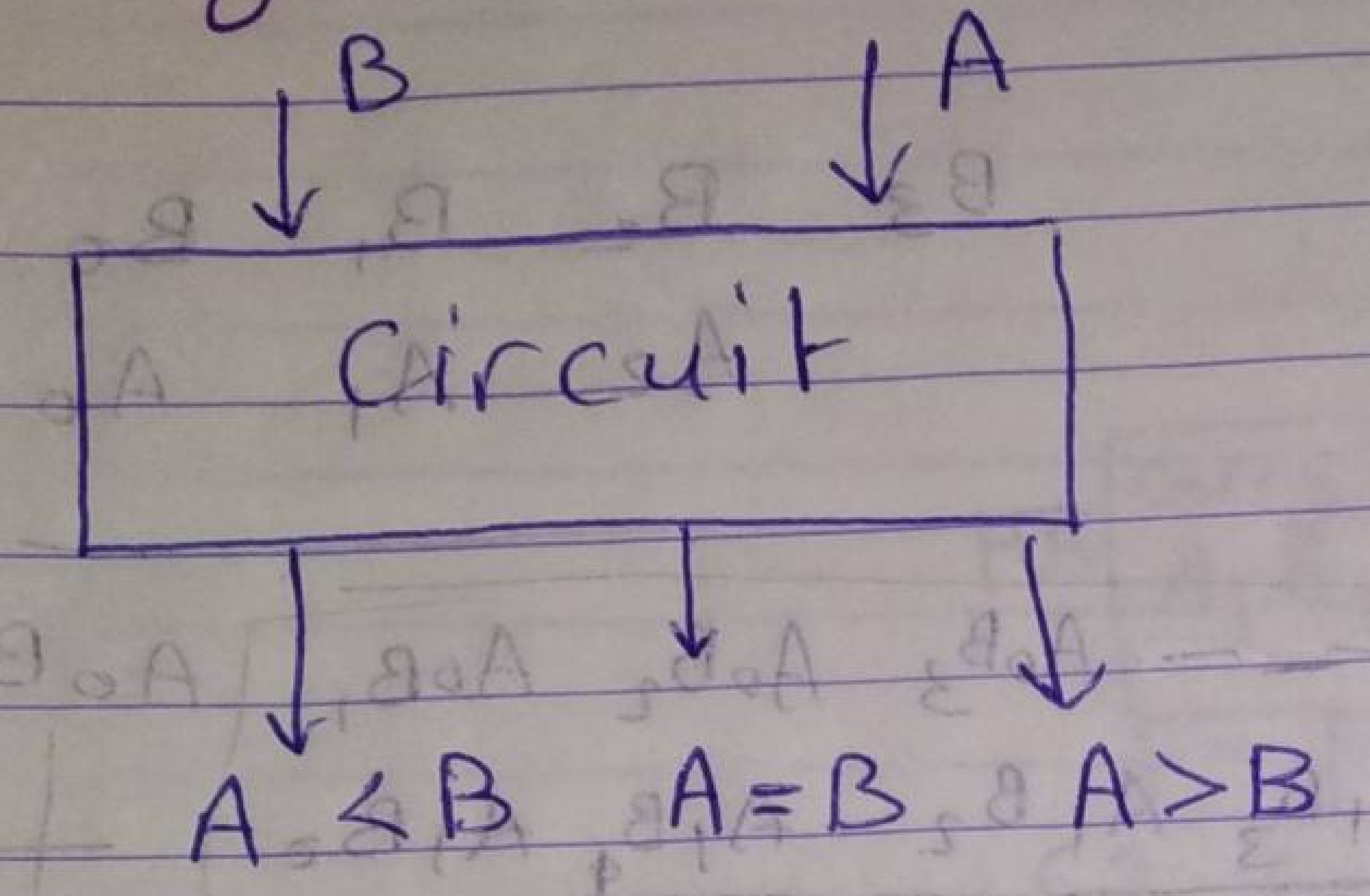
ex Design 4 bit x 3 bit multiplier.



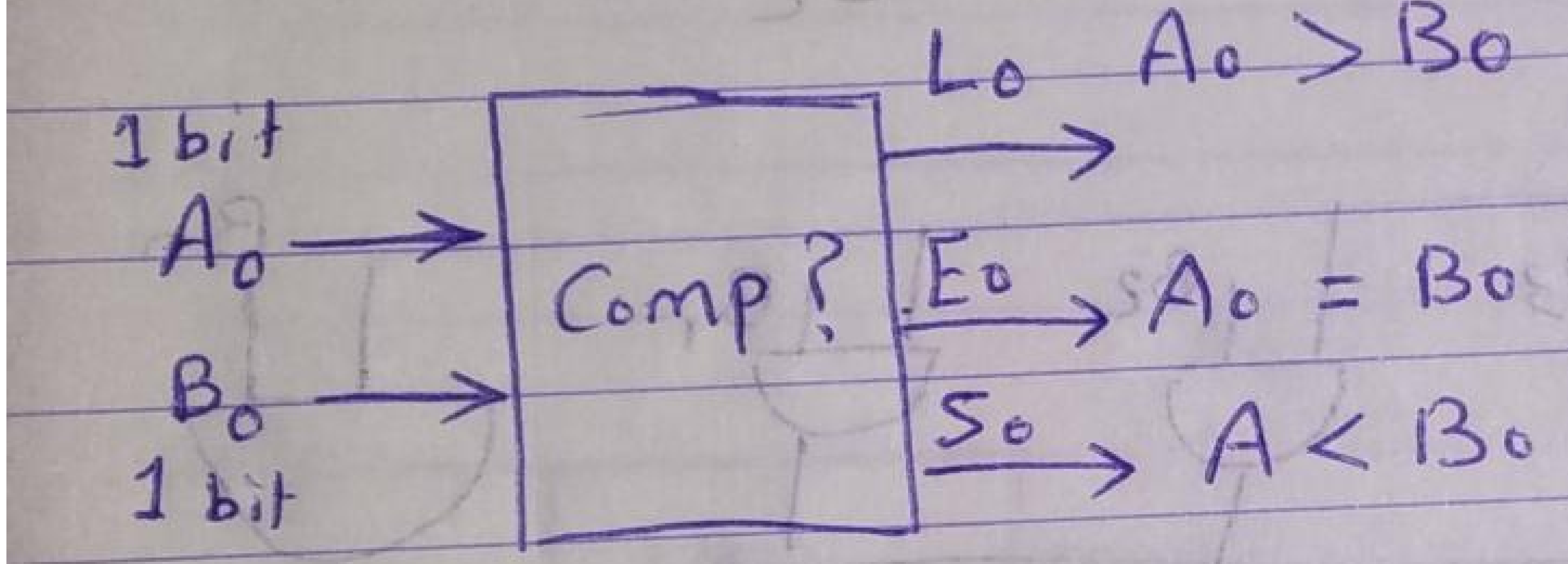


Scanned by TapScanner

Magnitude Comparator :-



Design 1 bit magnitude Computer.



$L_o = A_o B_o'$

A_o \ B_o	0	1
0		
1	1	

A_o	B_o	L_o	S_o	E_o
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	0	0	1

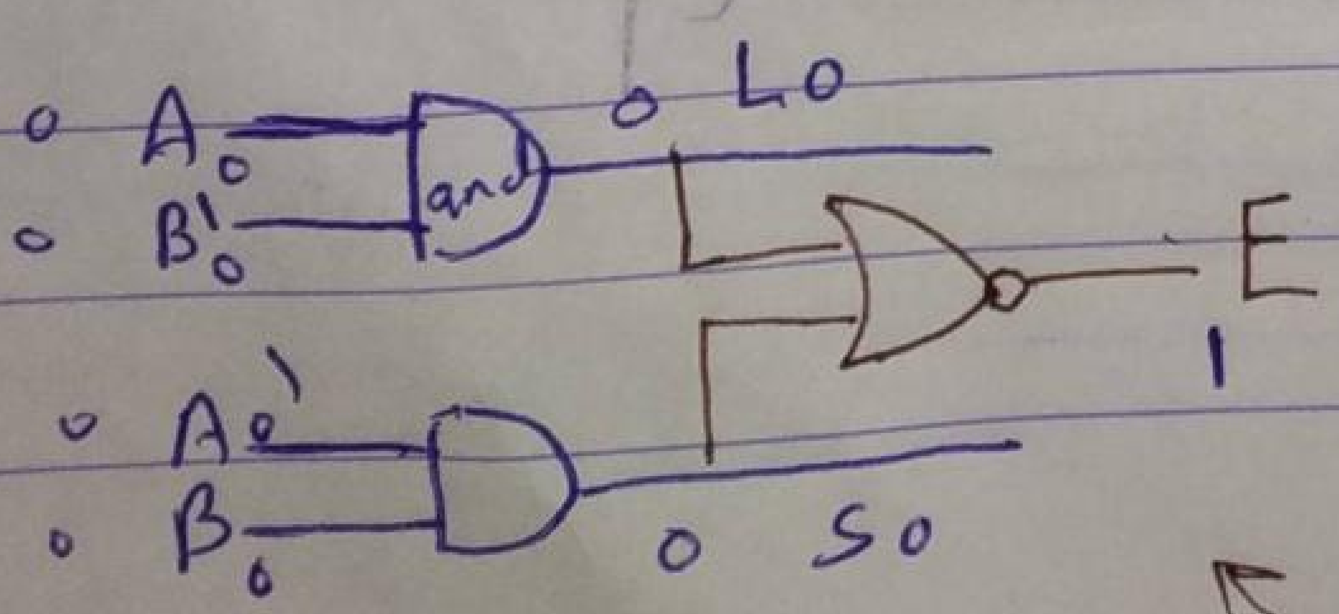
$S_o = A_o' B_o$

$L_o + S_o = A_o B_o' + A_o' B_o$

$E_o = A_o' B_o' + A_o B_o$

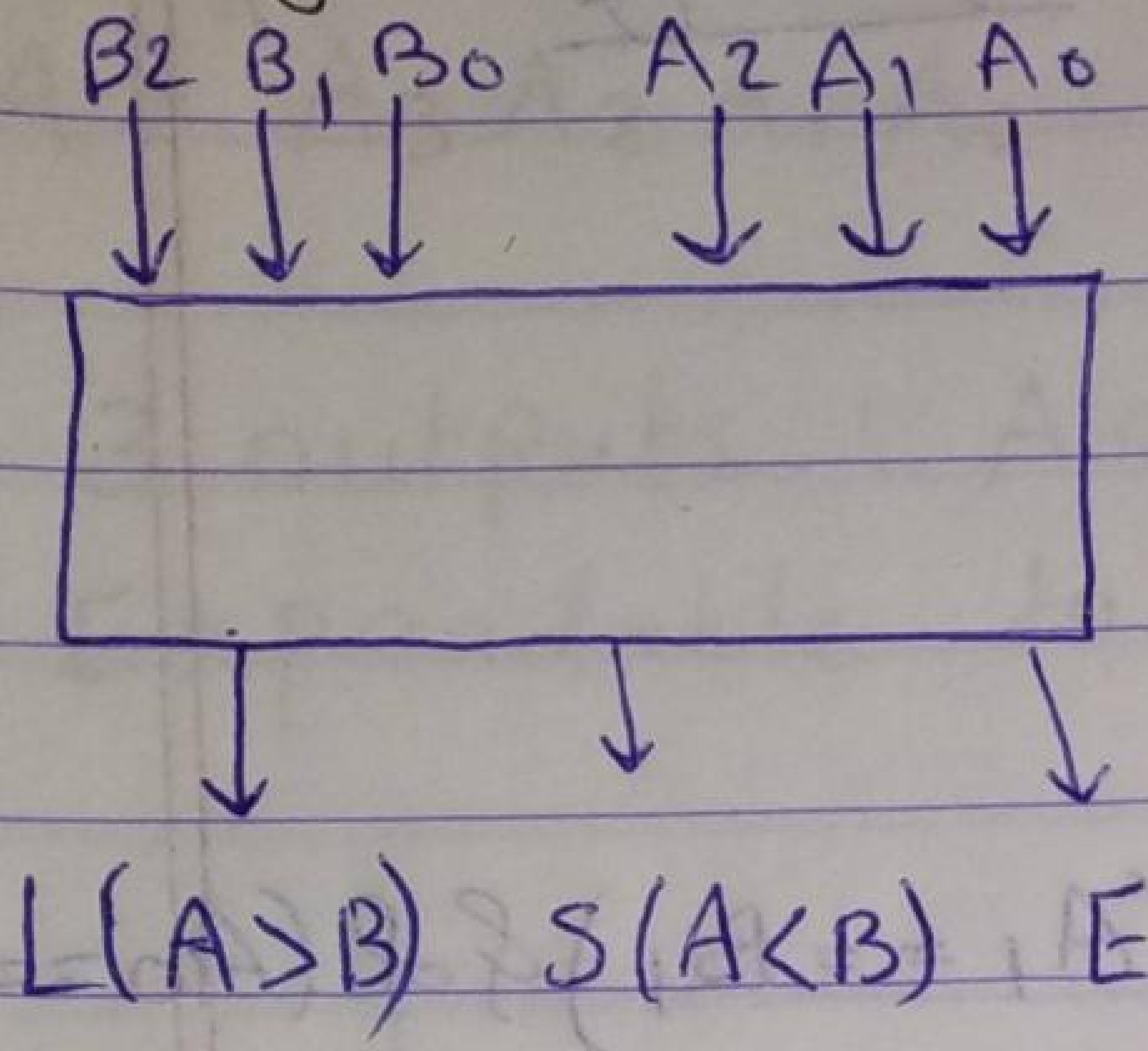
$E_o = (L_o + S_o)'$

$= (A_o + B_o)'$



← 1 bit magnitude Comparator.

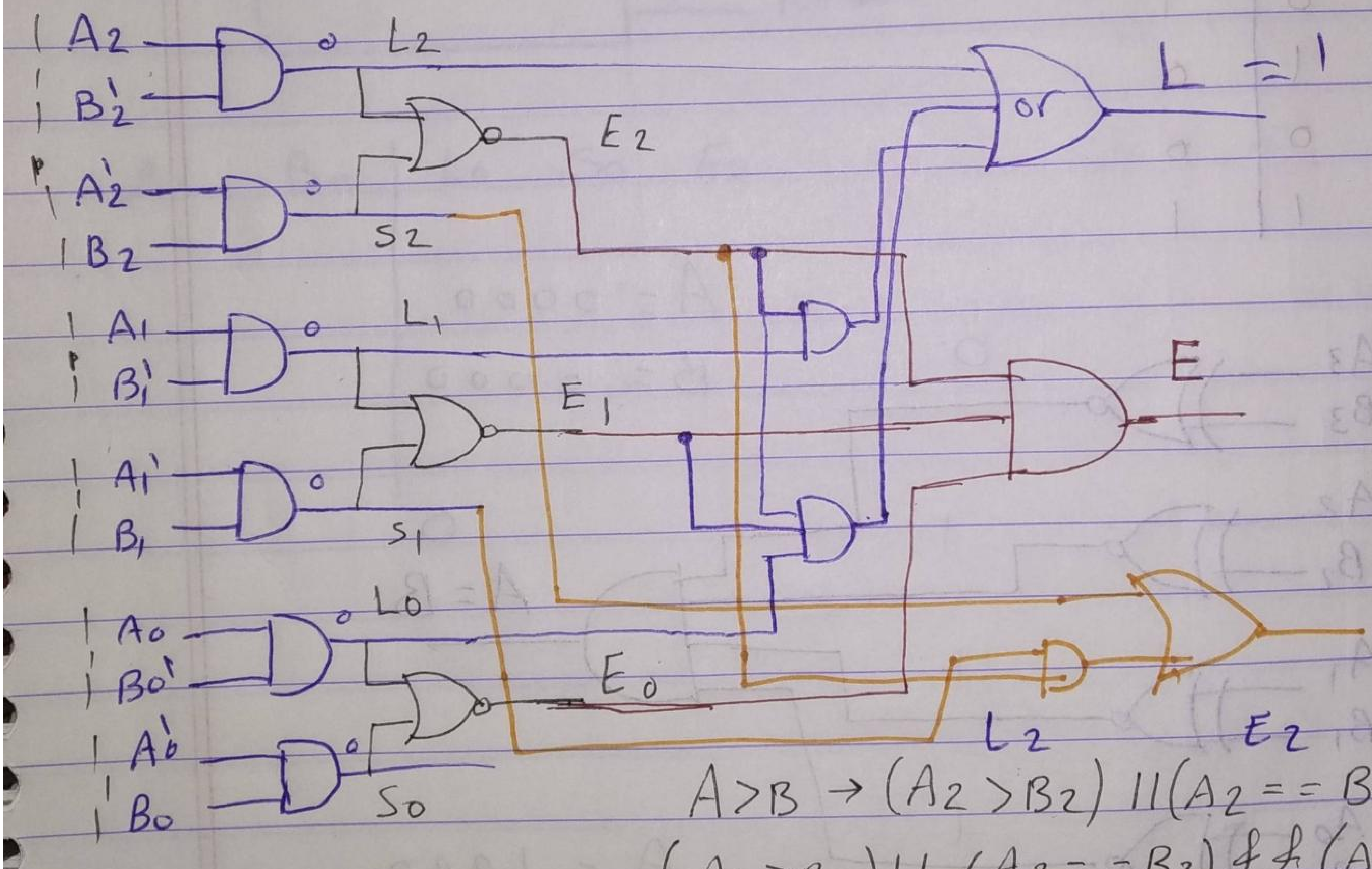
Design 3 bit magnitude Comparator.



$A = A_2 A_1 A_0$.

$B = B_2 B_1 B_0$.

$A = B \rightarrow (A_2 == B_2) \& (A_1 == B_1) \& (A_0 == B_0)$.



$A > B \rightarrow (A_2 > B_2) \,||\, (A_2 == B_2) \&\& (A_1 > B_1) \,||\, (A_2 == B_2) \&\& (A_1 == B_1) \&\& (A_0 > B_0)$.

$(A < B) \rightarrow (A_2 < B_2) \,||\, (A_2 == B_2 \&\& A_1 < B_1) \,||\, (A_2 == B_2) \&\& (A_1 == B_1) \&\& (A_0 < B_0)$.

Design a 4 bit magnitude equality Comparator.

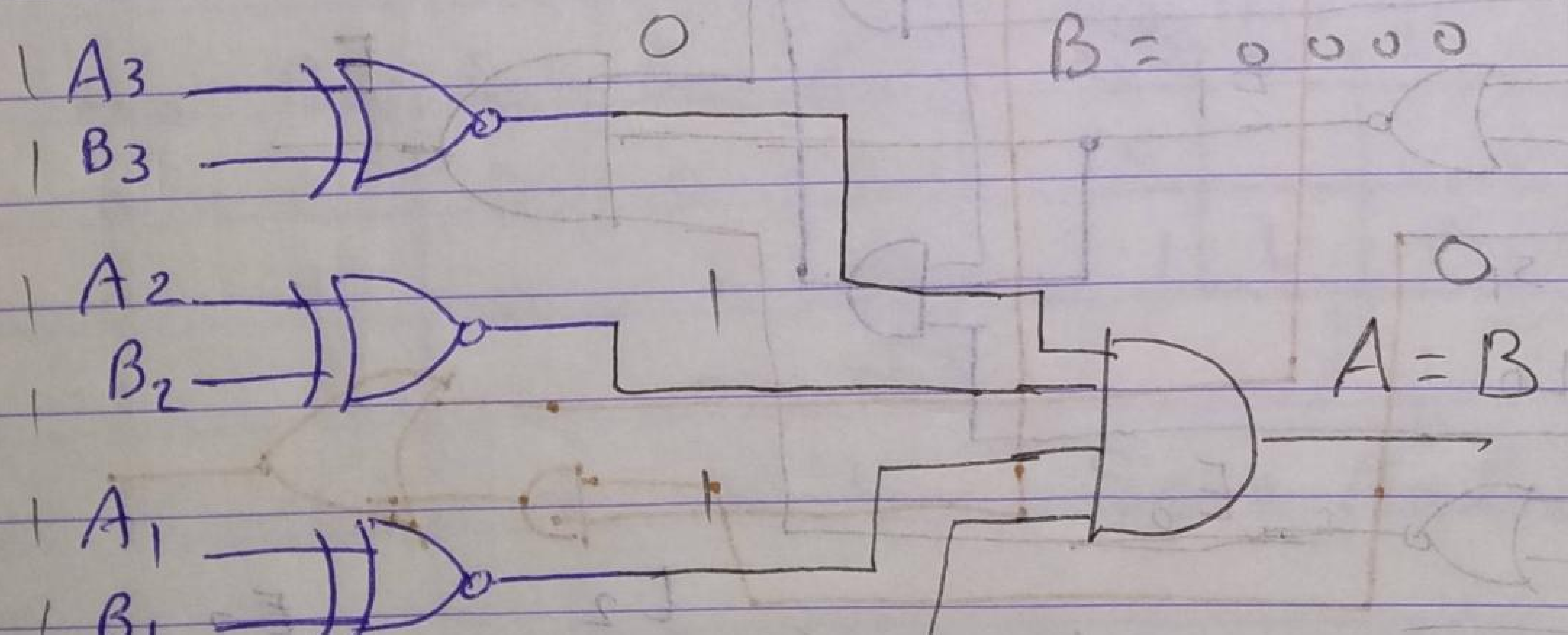
$A = A_3 A_2 A_1 A_0$
$B = B_3 B_2 B_1 B_0$

$A == B$
$(A_3 == B_3)$ && $(A_2 == B_2)$ && $(A_1 == B_1)$ && $(A_0 == B_0)$

XNOR

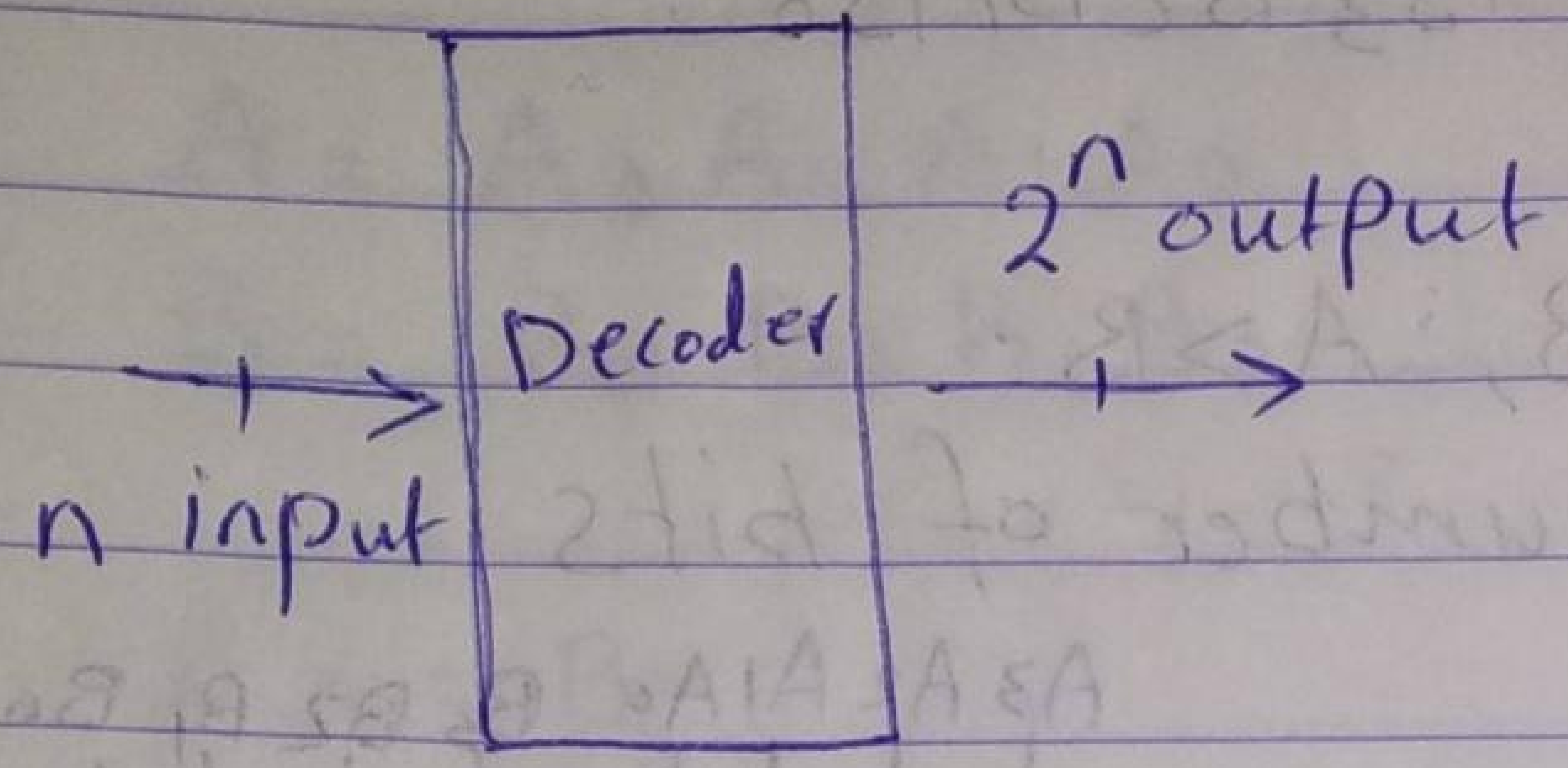
A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

A = 0000
B = 0000

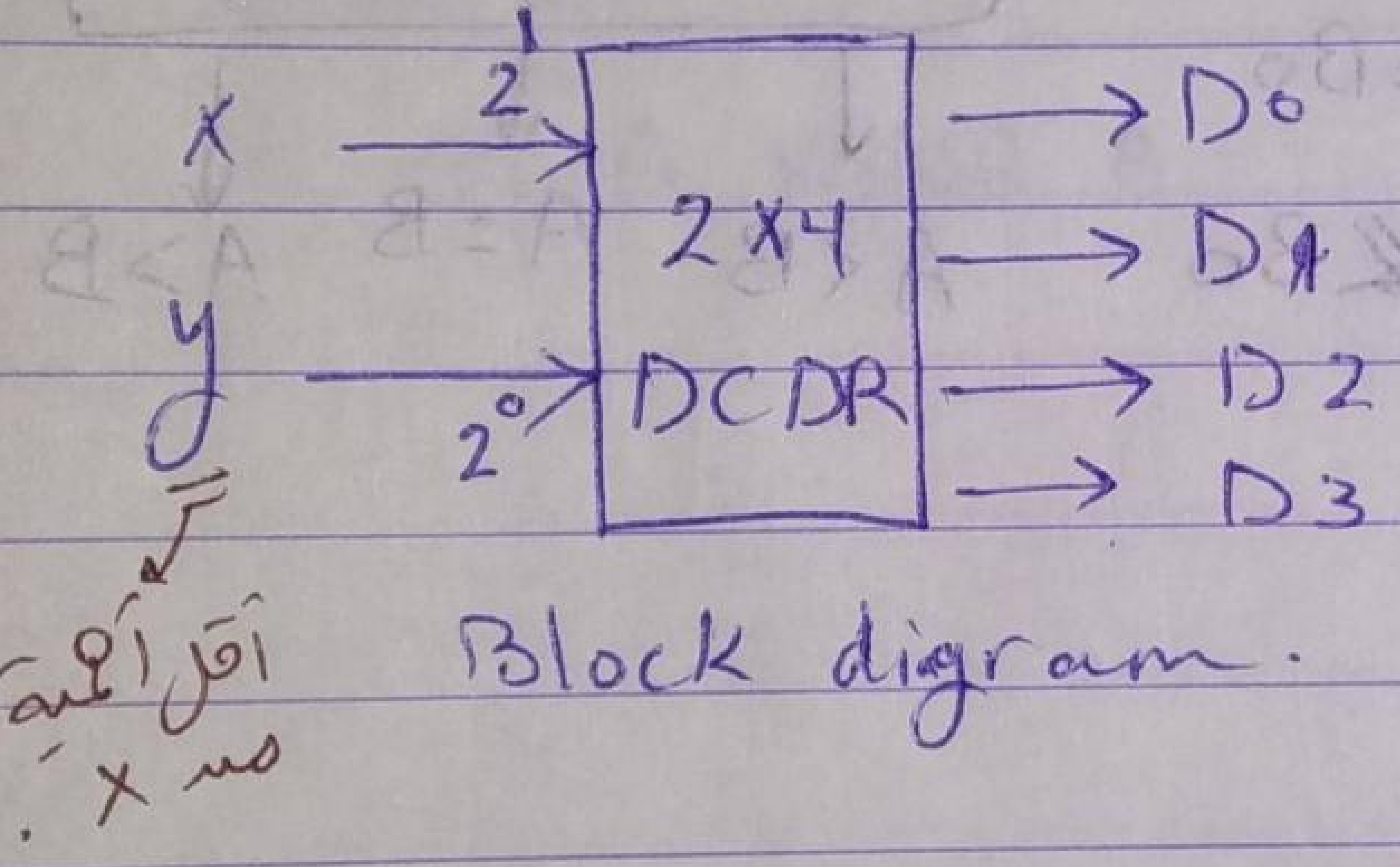


A = 1000
B = 0000

Decoder : Combinational circit .



example :- Decoder 2X4 .



Block digram.

X	y	D0	D1	D2	D3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

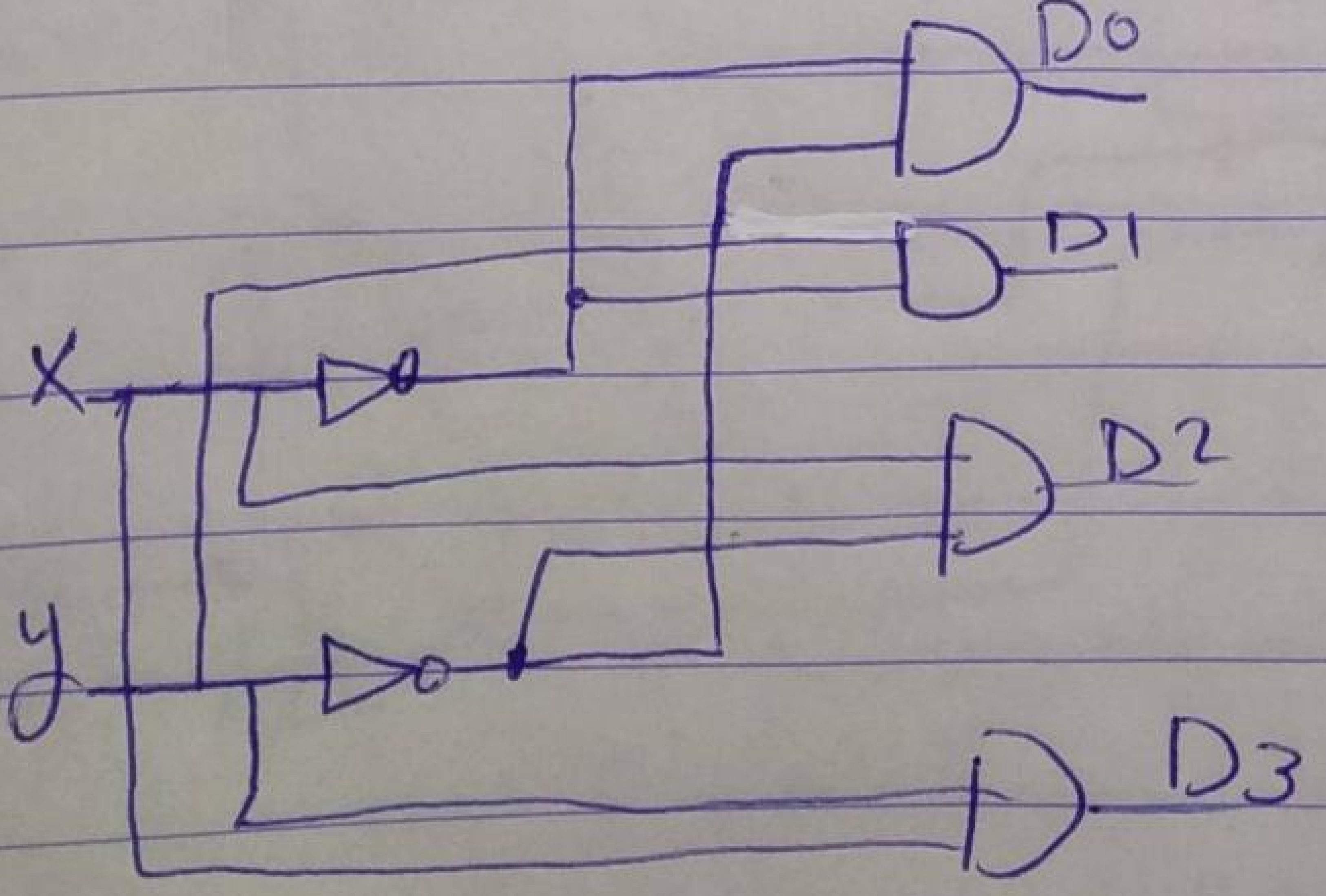
$D_0 = x'y' = m_0$.

$D_1 = x' \cdot y = m_1$.

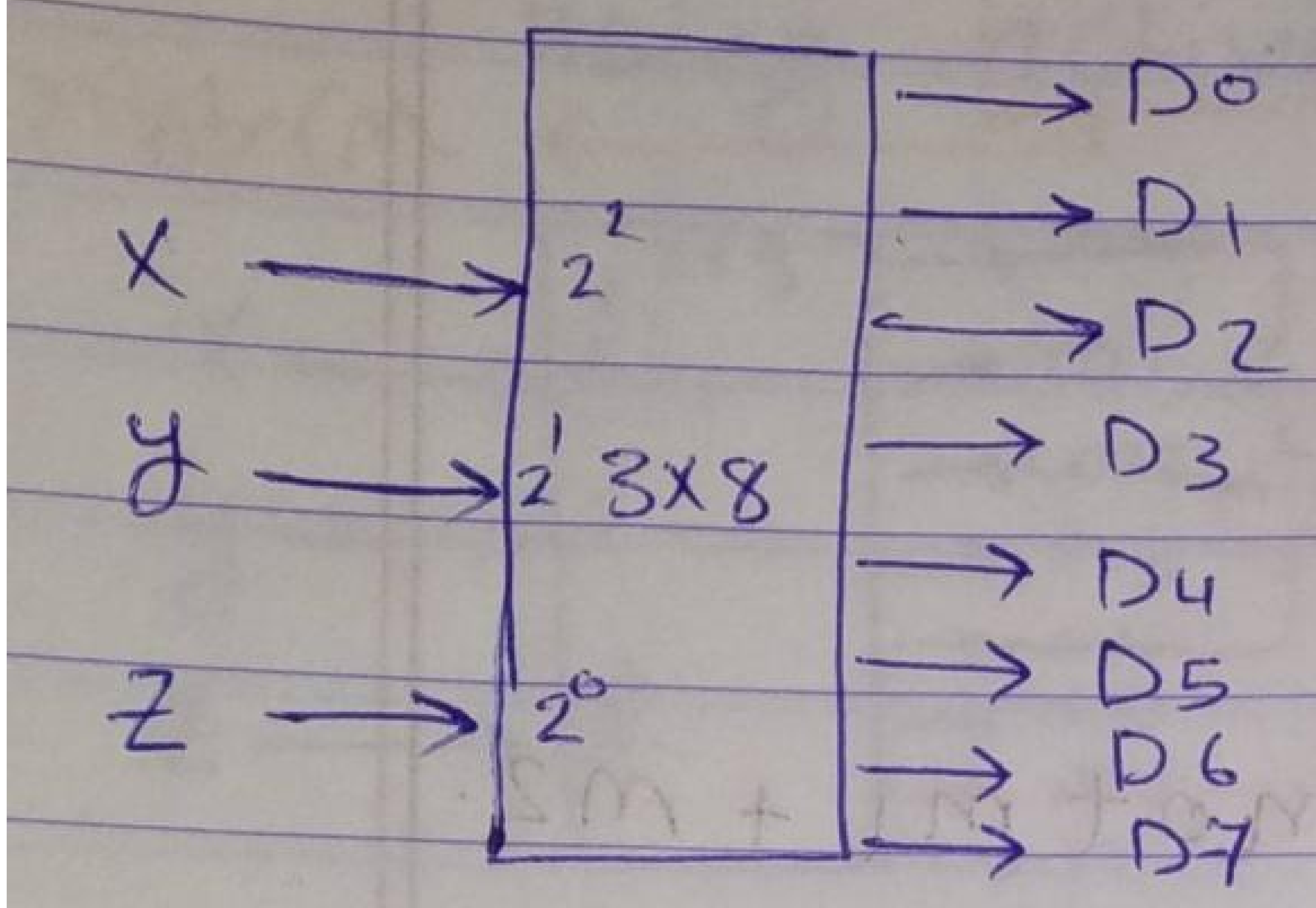
$D_2 = xy' = m_2$.

$D_3 = xy = m_3$.

x \ y	0	1
0	1	
1		



example 3 X 8 Line Decoder :

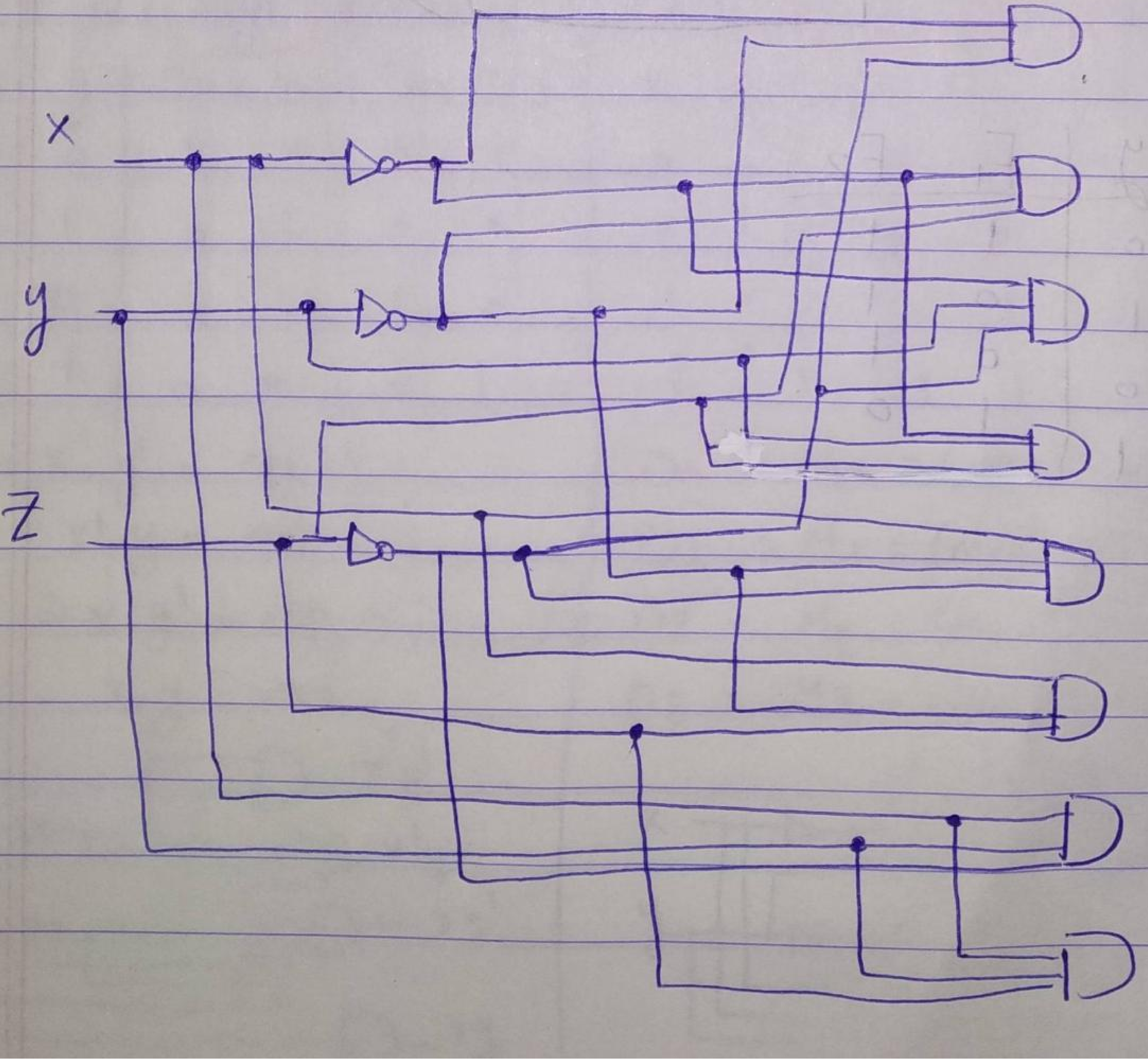


x	y	z	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

$D0 = x'y'z'$ $D1 = x'y'.z$ $D2 = x'yz'$

$D3 = x'yz$ $D4 = xy'z'$ $D5 = xy'z$

$D6 = xyz'$ $D7 = xyz$.

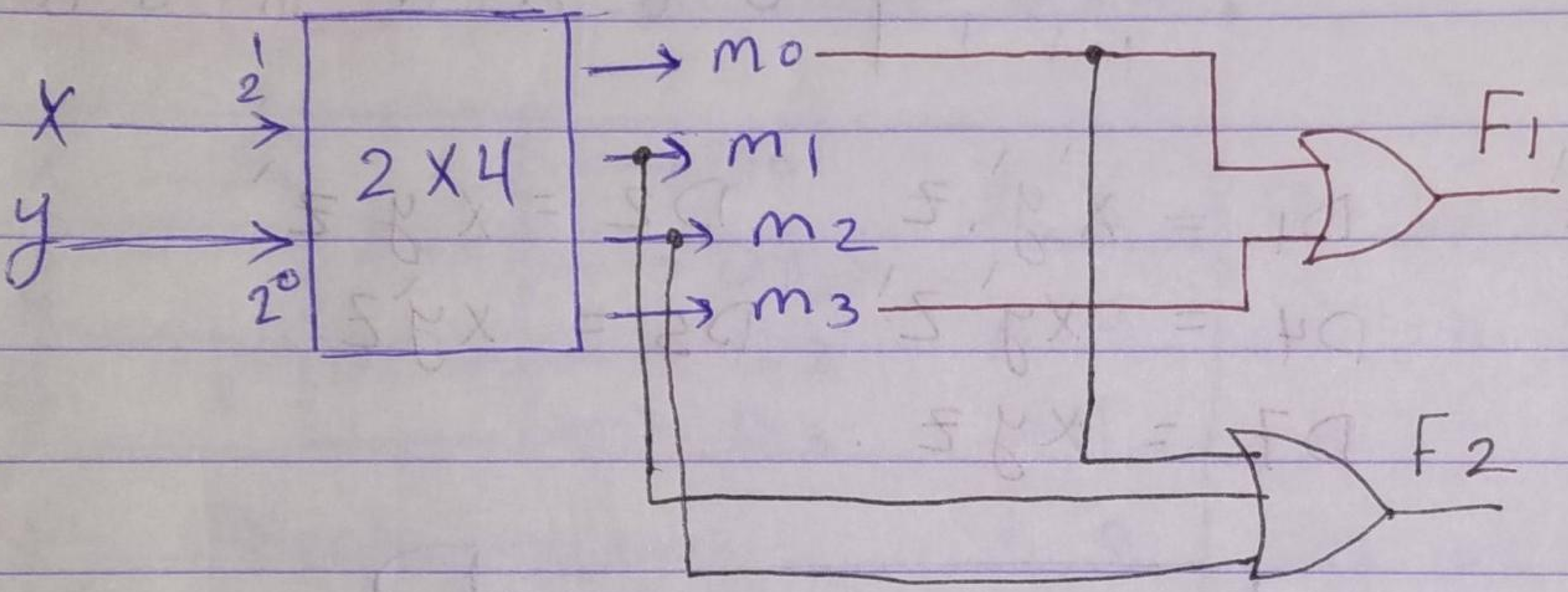


example 1- Implement the Following Function using Decoder ?.

$F_1(x, y) = \sum 0, 3.$

$F_2(x, y) = \sum 0, 1, 2.$

$F_1 = m0 + m3.$ $F_2 = m0 + m1 + m2.$

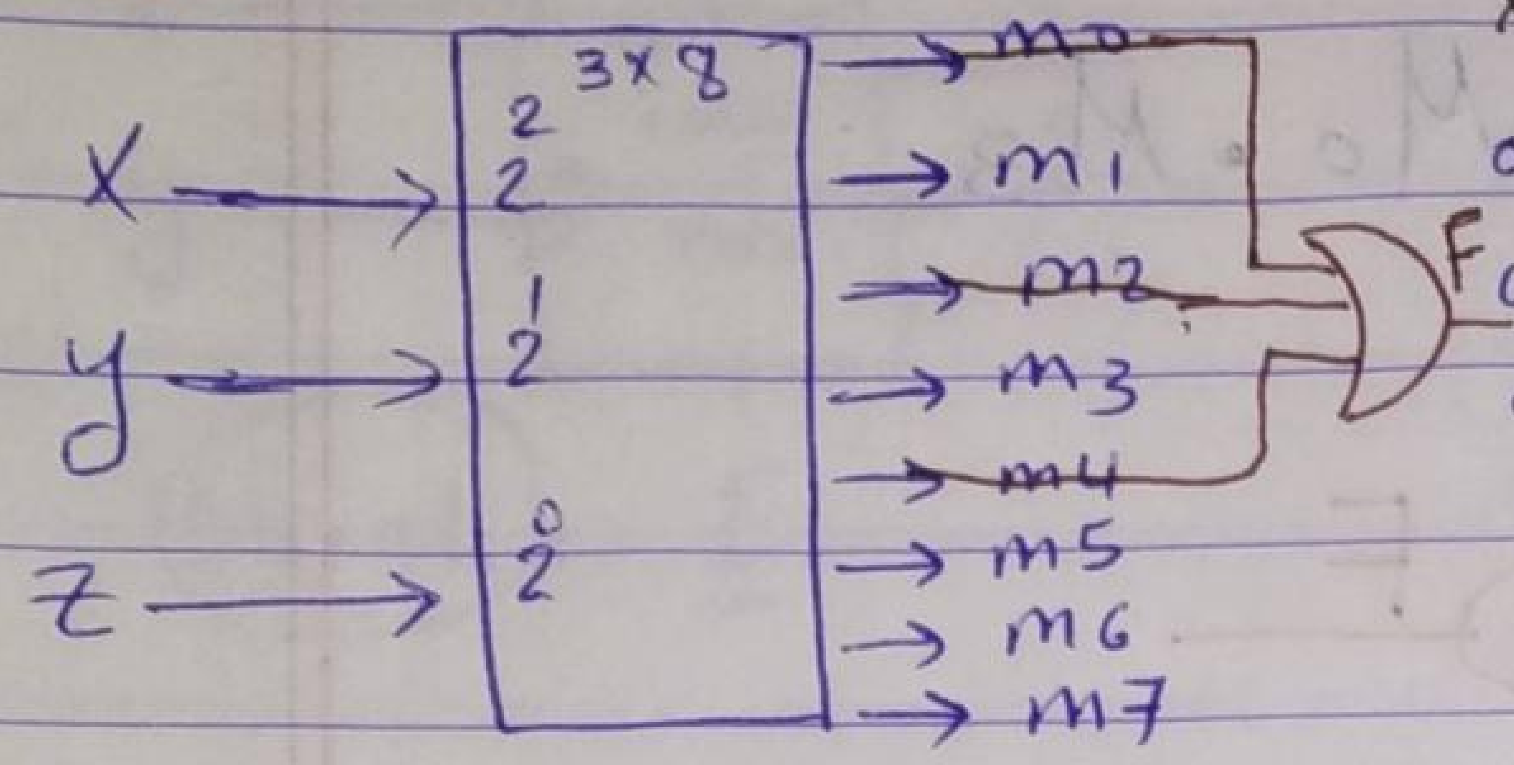


X	y	F_1	F_2
0	0	1	1
0	1	0	1
1	0	0	1
1	1	1	0

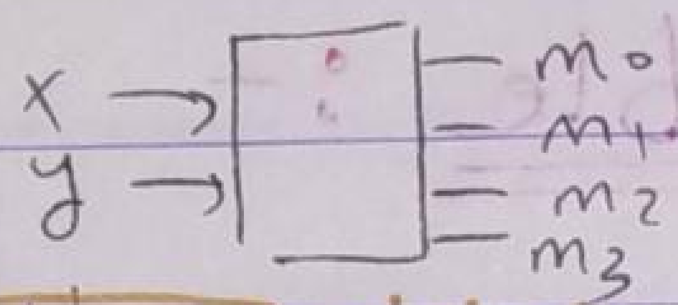
Scanned by TapScanner

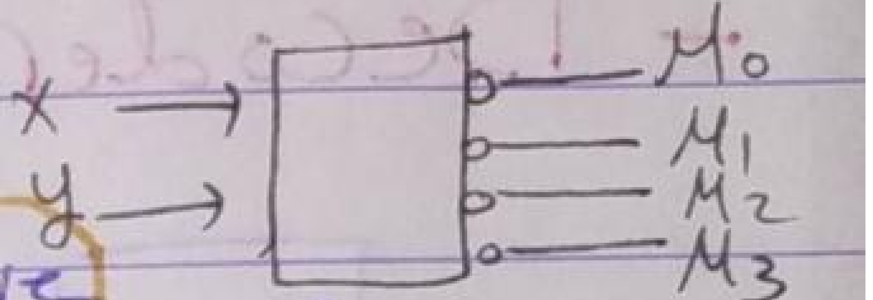
example :- Implement $F(x,y,z) = \sum 0, 2, 4$ using active high decoder.

minterm ←



x	y	z	D0	D1	D2	D3	D4	D5	D6	D7
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





example :-

2 x 4 active high decoder Minterm

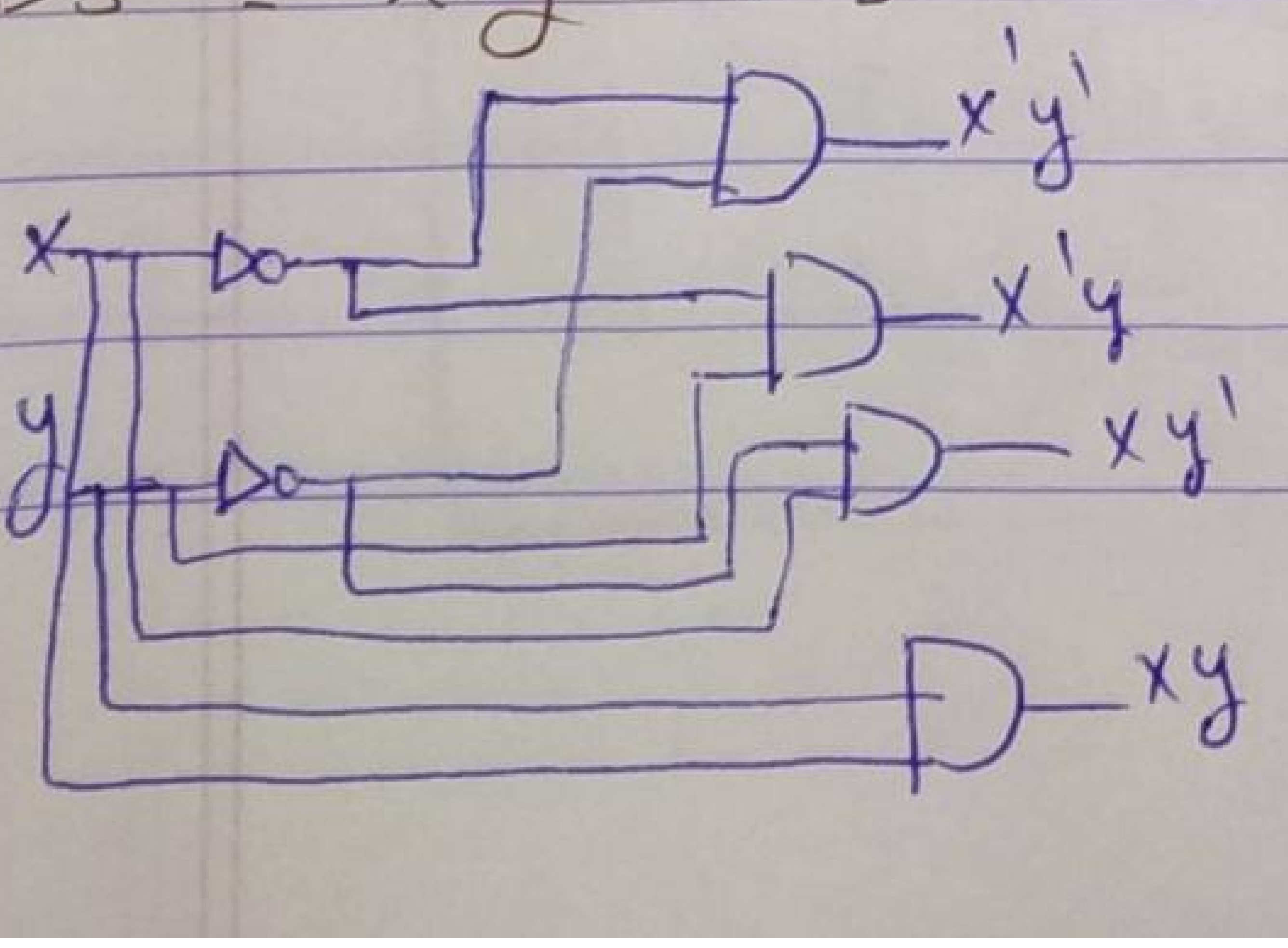
x	y	D0	D1	D2	D3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

$D_0 = x'.y' = m_0$

$D_1 = x'.y = m_1$.

$D_2 = x.y' = m_2$.

$D_3 = x.y = m_3$.



Maxterm

2 x 4 active low decoder.

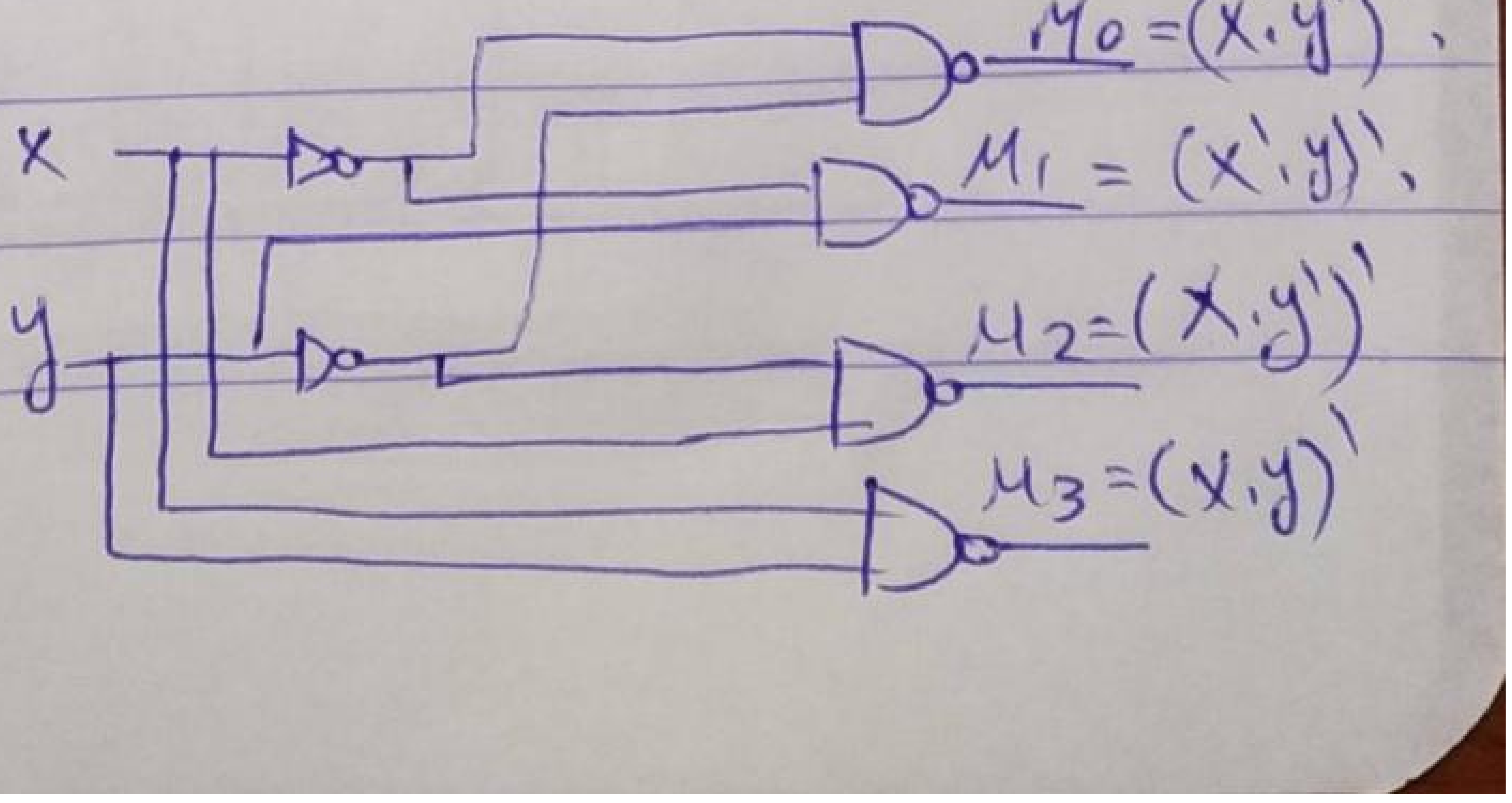
x	y	D0	D1	D2	D3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

$D_0 = M_0 = (m_0)' = (x'.y')' = x+y$

$D_1 = M_1 = (m_1)' = (x'.y)'$

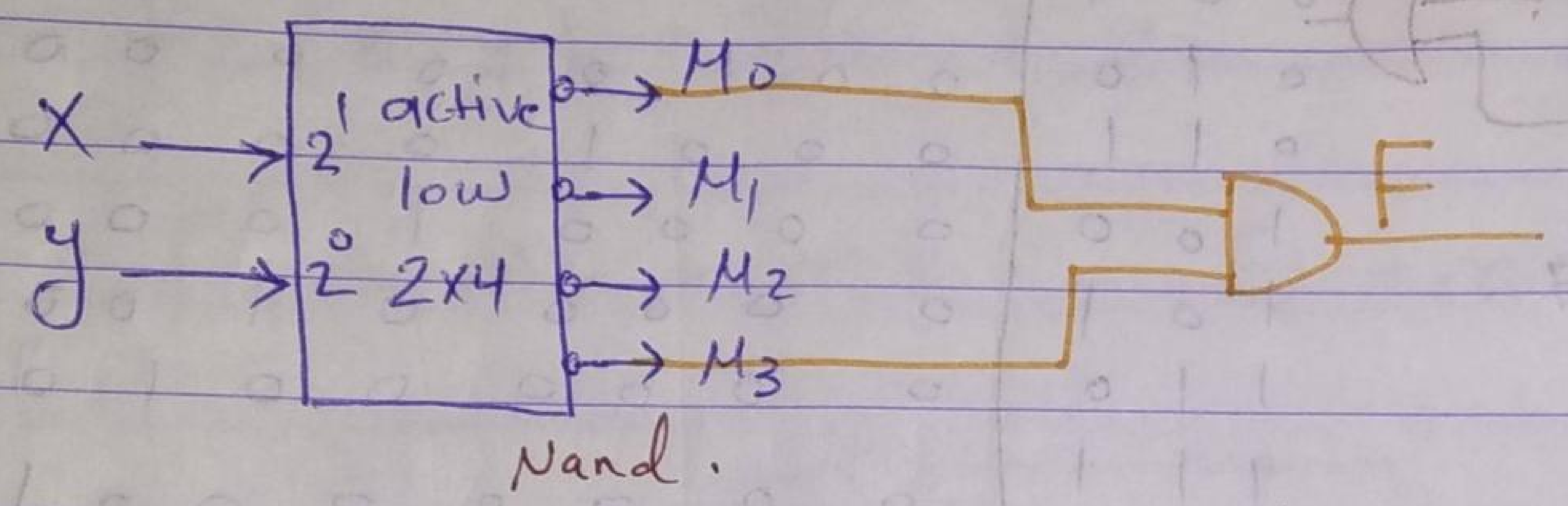
$D_2 = M_2 = (m_2)' = (x.y')'$

$D_3 = M_3 = (m_3)' = (x.y)'$



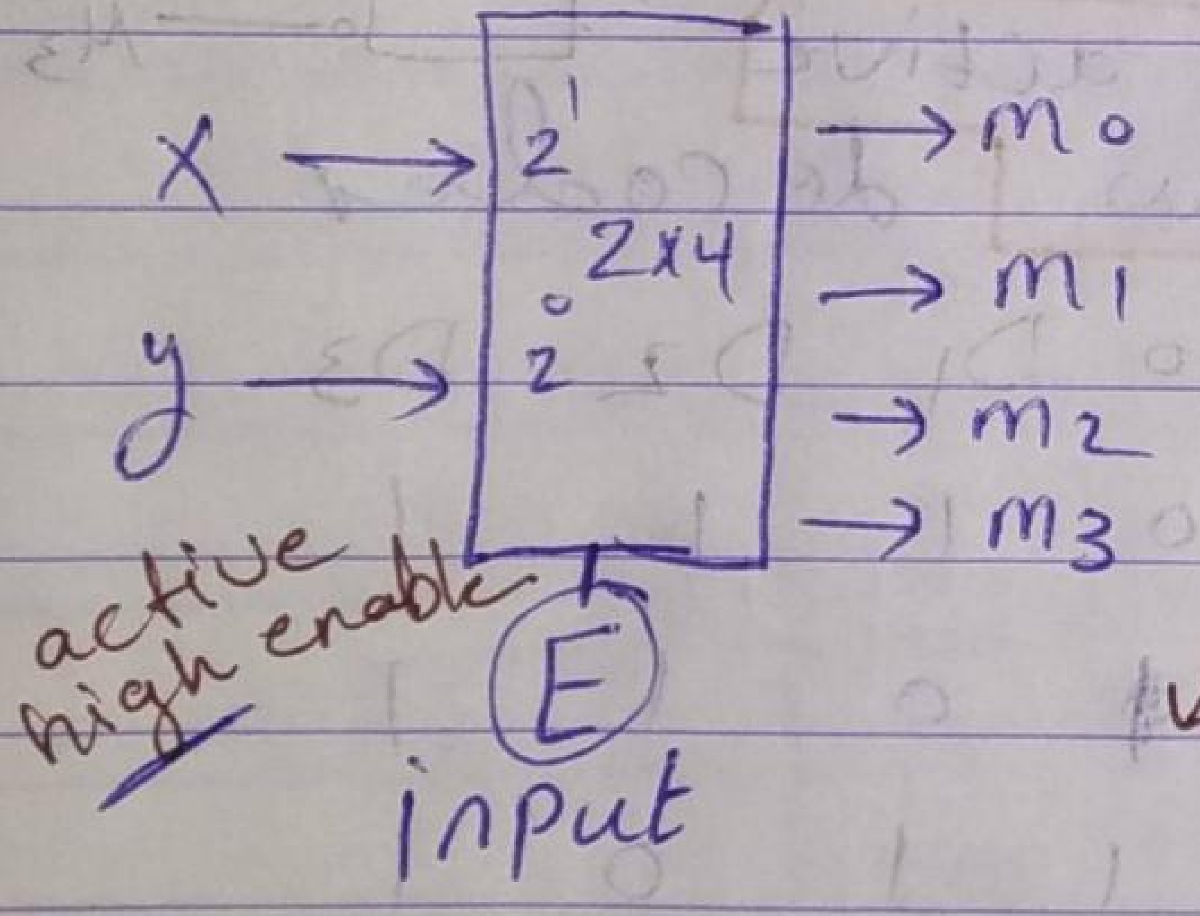
example :- Implement the Following Function using decoder.

$$F(x,y) = \pi(0,3) = M_0 \cdot M_3$$



E

- Decoder with enable :-



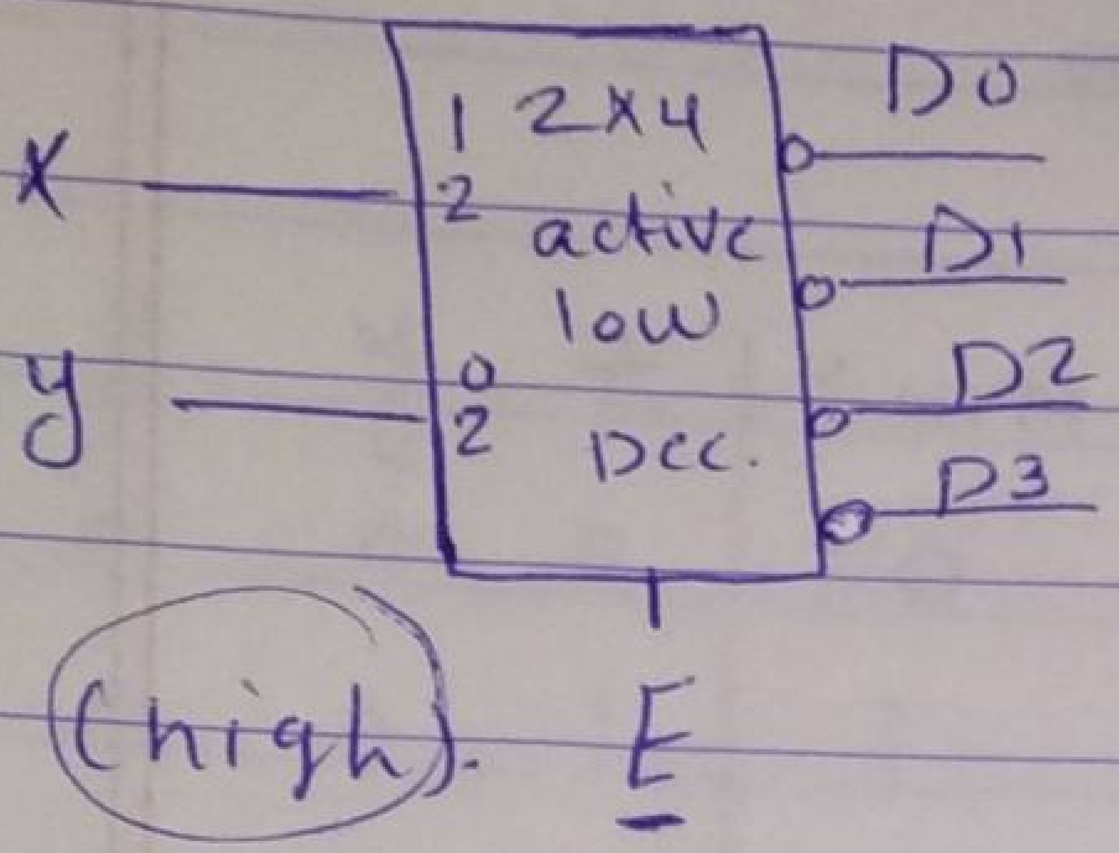
E	X	y	D0 m0	D1 m1	D2 m2	D3 m3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

اذا كان صفر بغض النظر عن
قيمة x,y فبنحط أصفار.
ولما يكون 1 ← بنشوف كيف
كنا نحل.

active high enable.

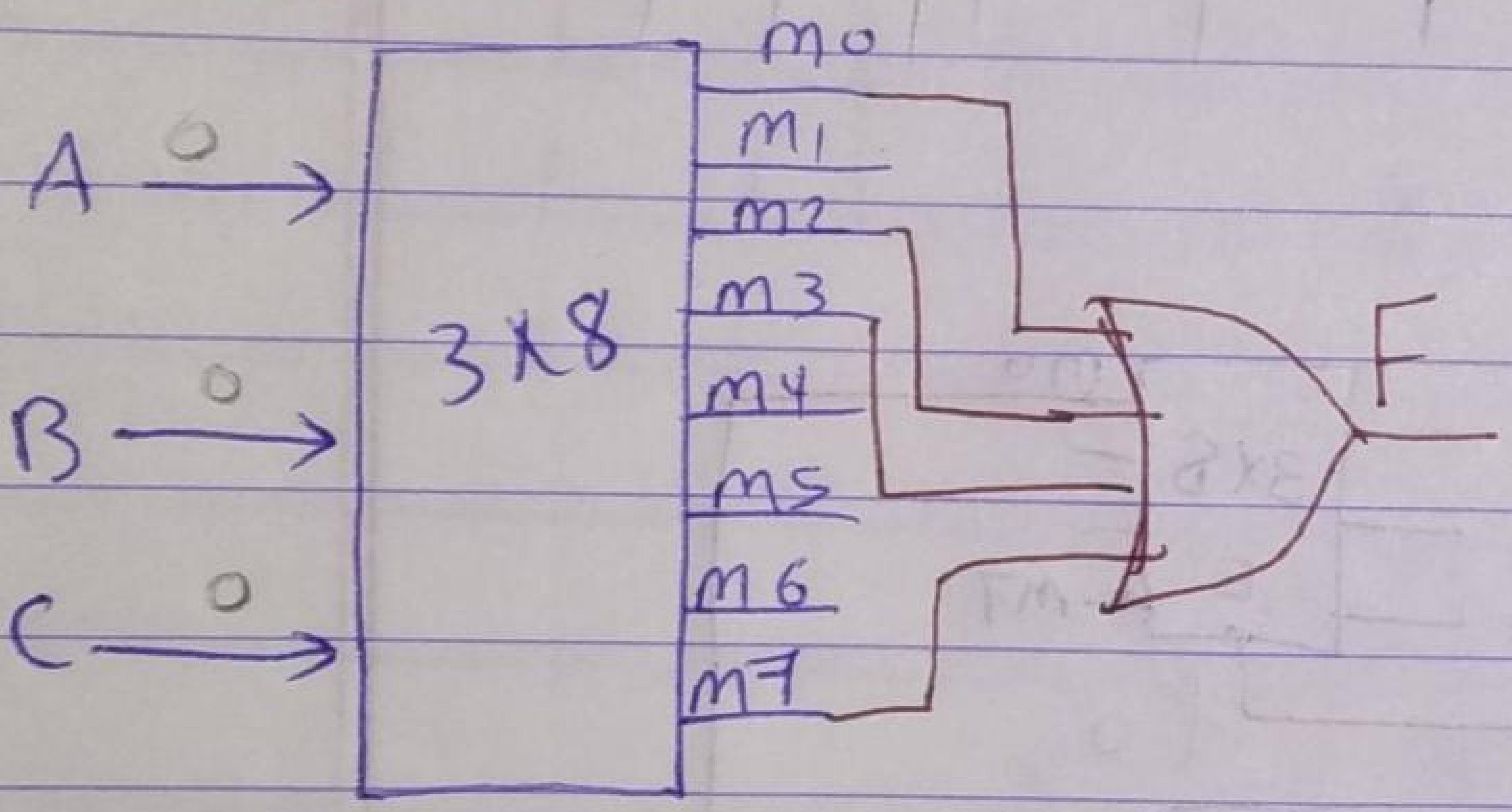
active high decoder.

example :- 2 X 4 decoder.

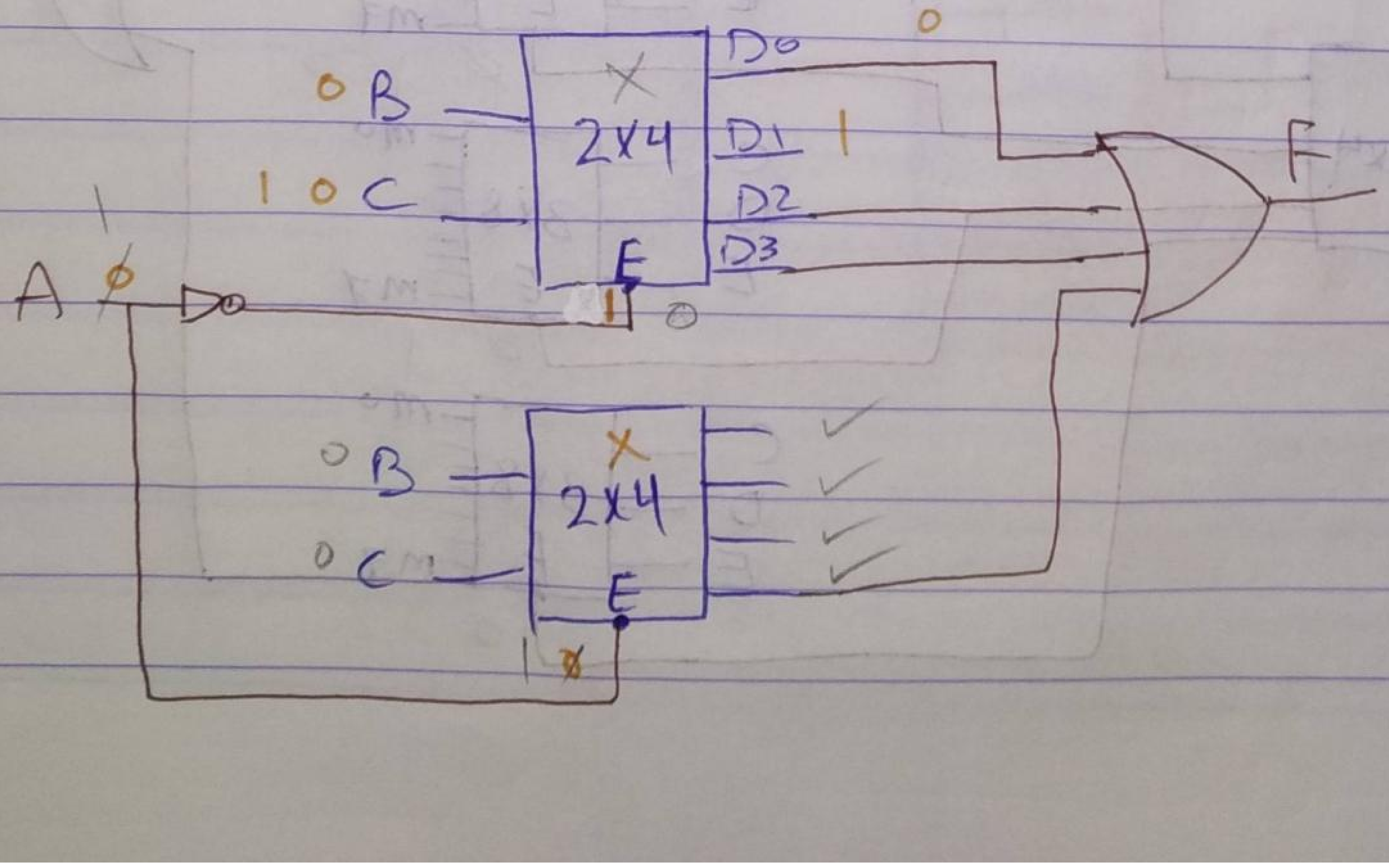


			M0	M1	M2	M3
E	x	y	D0	D1	D2	D3
0	X	X	1	1	1	1
1	0	0	0	1	1	1
1	0	1	1	0	1	1
1	1	0	1	1	0	1
1	1	1	1	1	1	0

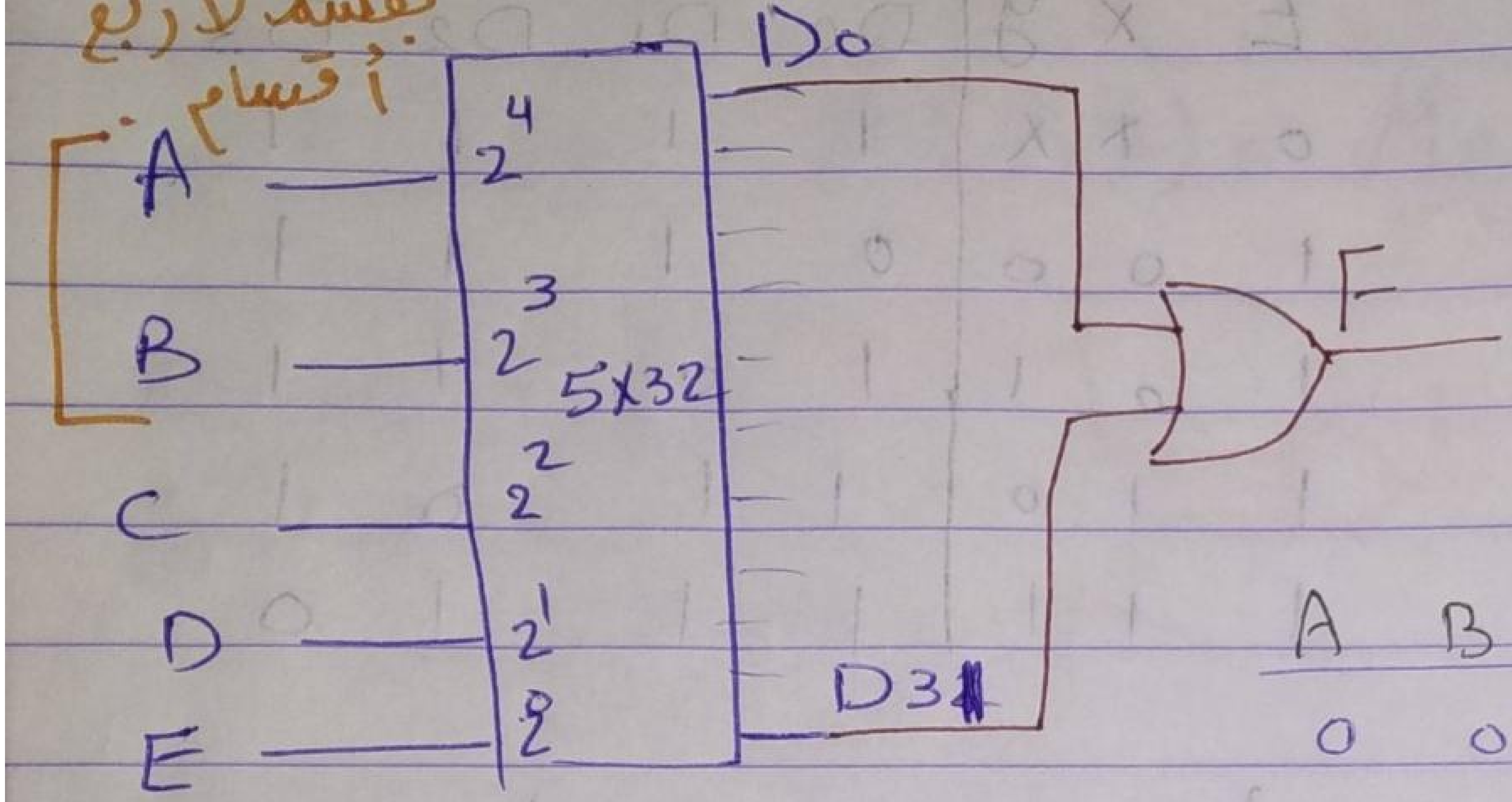
example :- $F(A,B,C) = \Sigma(0,2,3,7)$ Decoder.



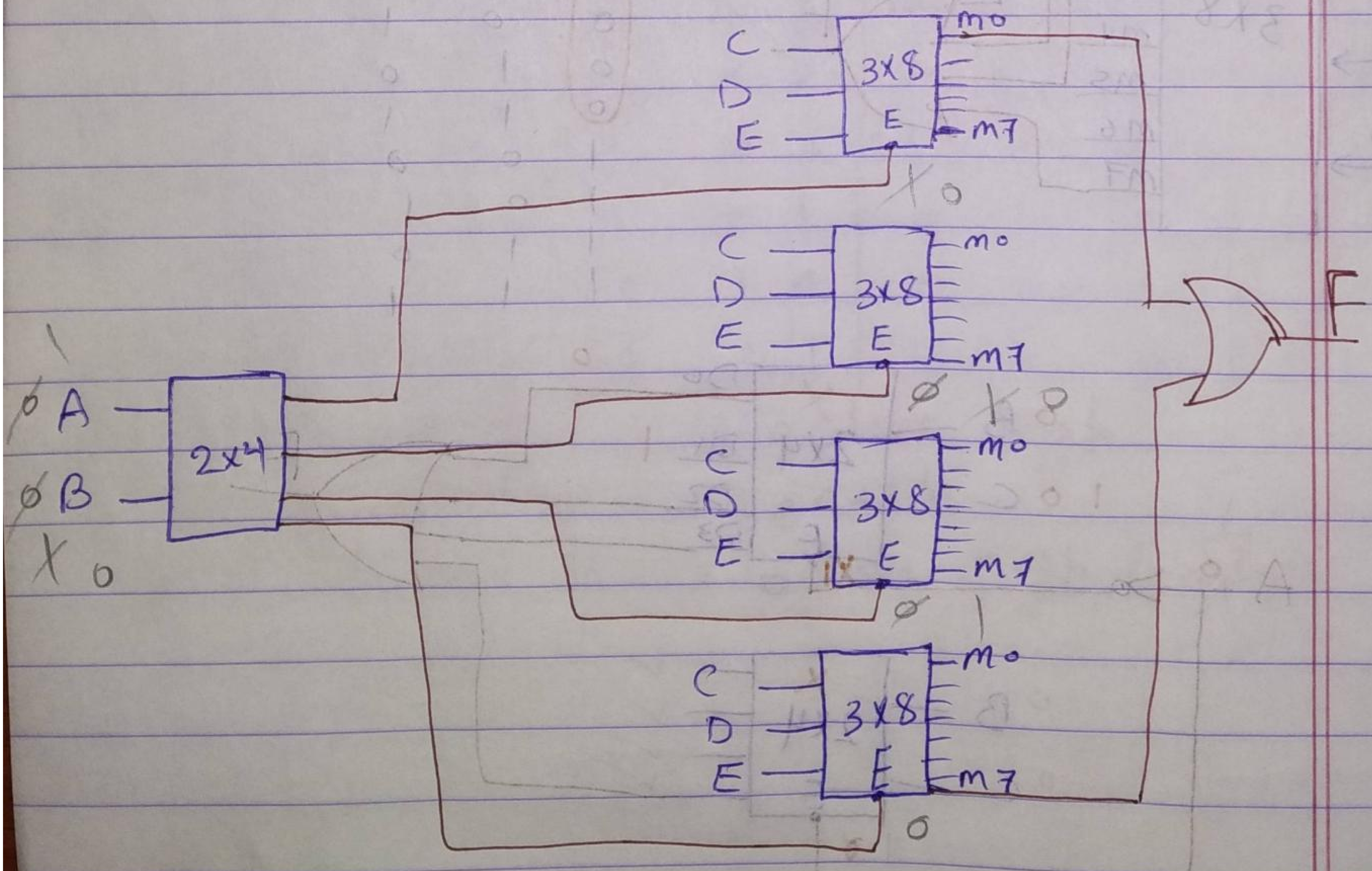
A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1



example :- F(A, B, C, D, E) = Σ0, 31



A	B	C	D	E
0	0	0	0	0
1	1	1	1	1



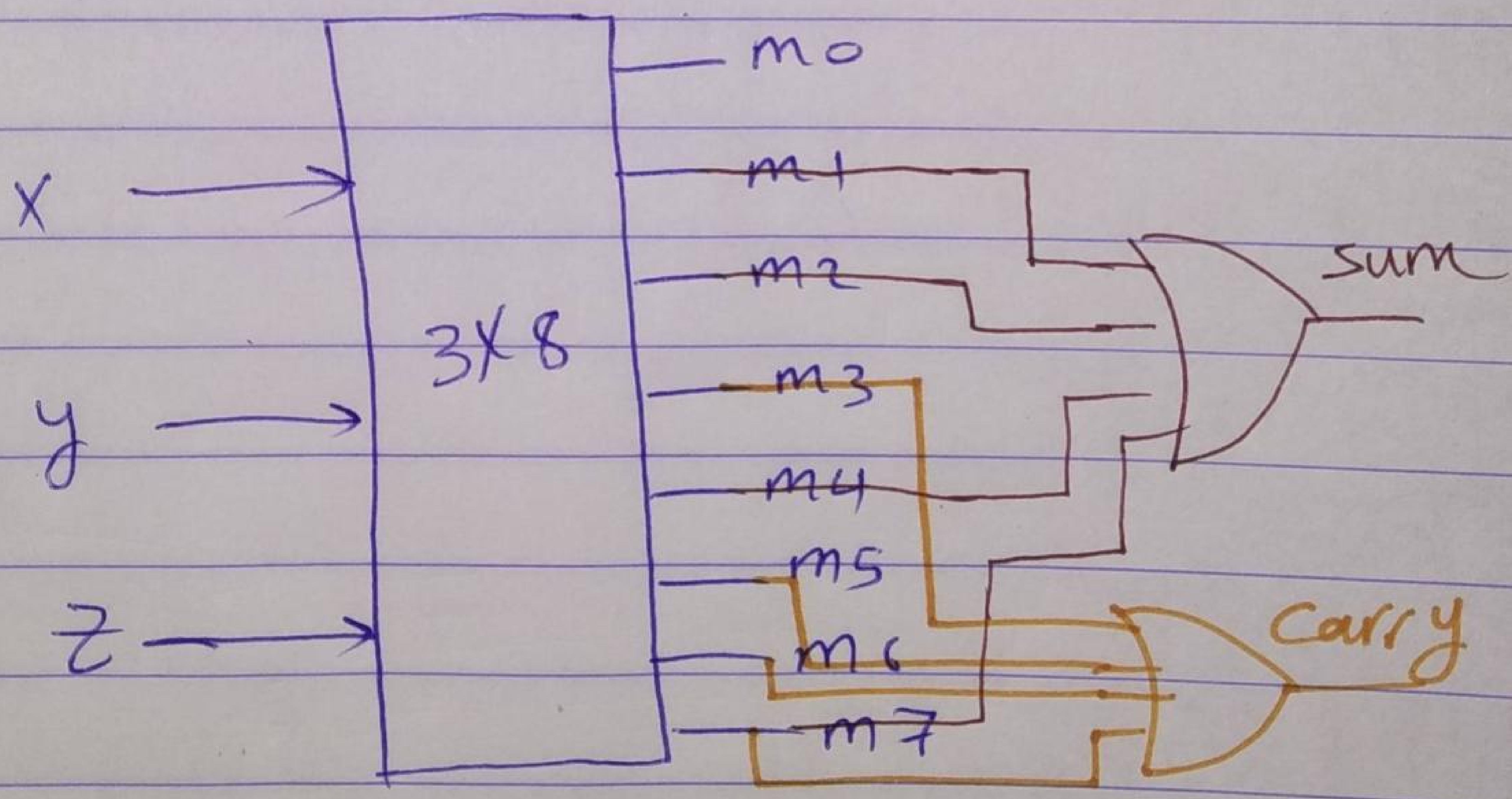
Scanned by TapScanner

example :- Implement full adder using decoder . . (بوخذ الواحدات)

x	y	z	sum	carry.
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

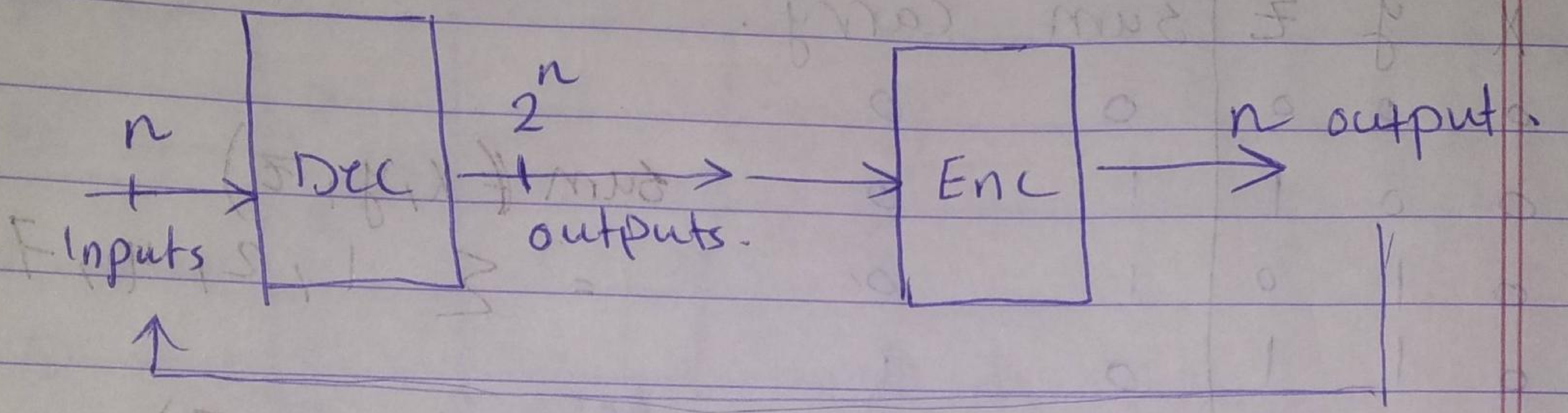
sum ~~f~~ (x, y, z)
$= \sum 1, 2, 4, 7.$

carry (x, y, z)
$= \sum 3, 5, 6, 7.$



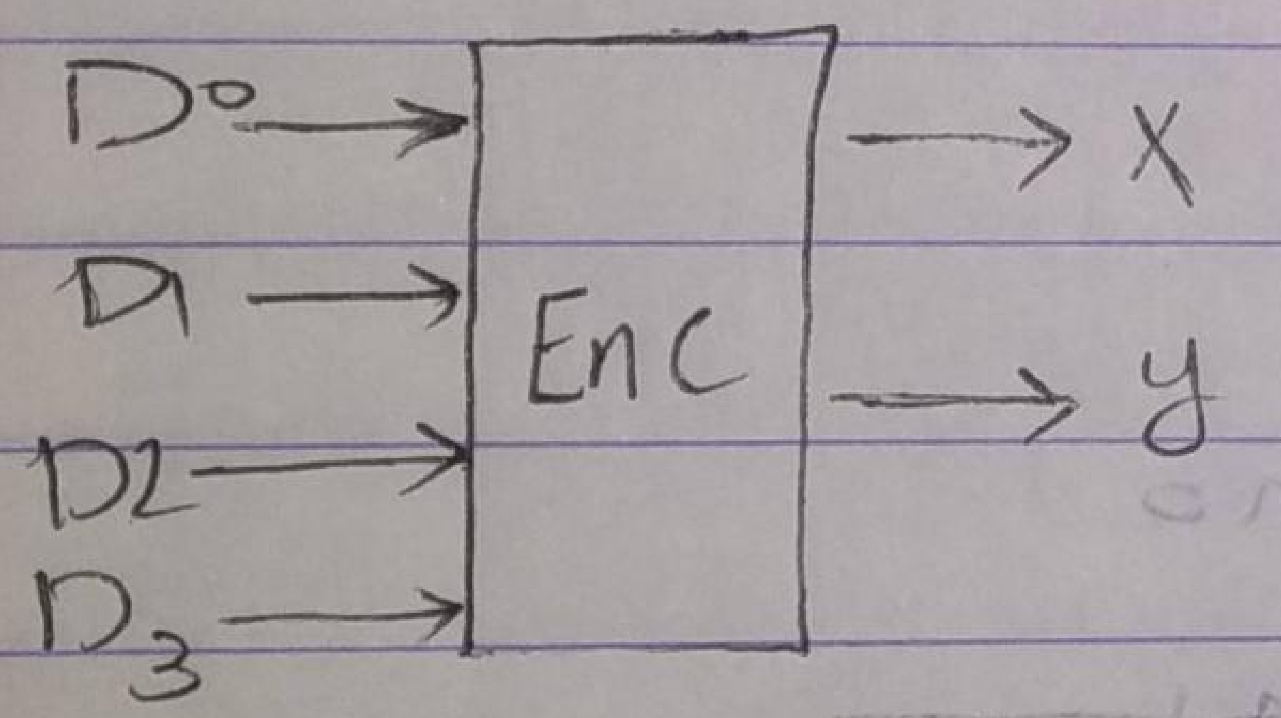
Encoder :- Combinational circuit

Inverse operation of decoder.



example : 4 x 2 Encoder.

Inputs

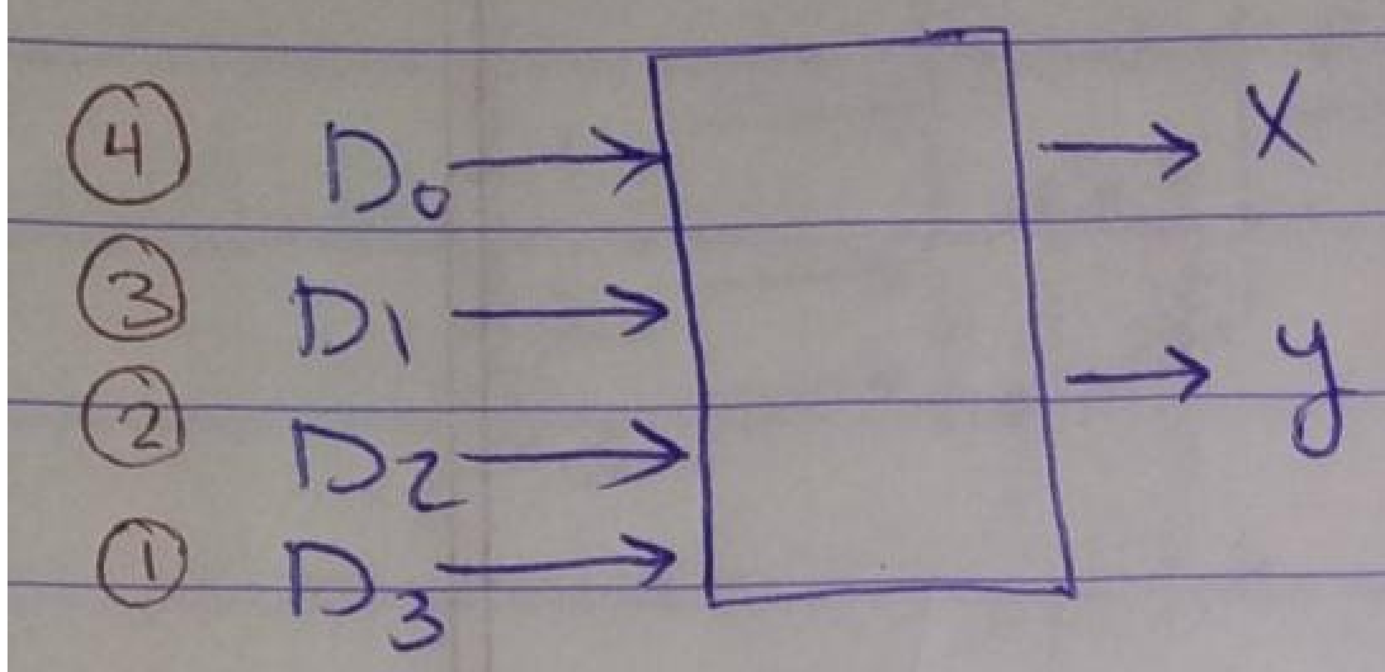


D0	D1	D2	D3	x	y
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

Scanned by TapScanner

Priority Encoder :-

4 X 2 priority Encoder.



←

D_0	D_1	D_2	D_3	X	y
1	0	0	0	0	0
X	1	0	0	0	1
X	X	1	0	1	0
X	X	X	1	1	1

D_0D_1 \ D_2D_3	00	01	11	10
00		1	1	1
01		1	1	1
11		1	1	1
10		1	1	1

$X = D_2 + D_3$

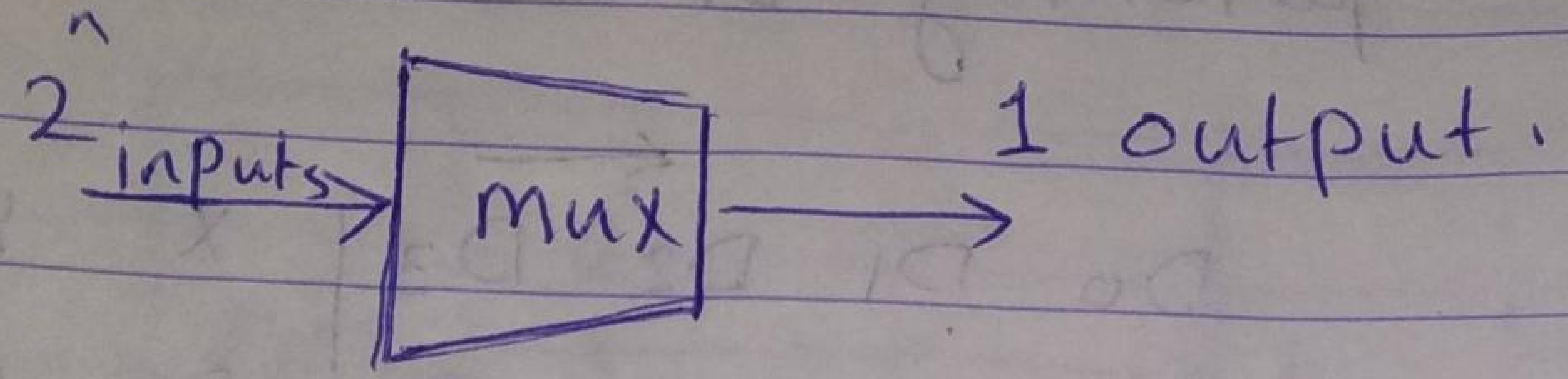
D_0D_1 \ D_2D_3	00	01	11	10
00		1	1	
01	1	1	1	
11	1	1	1	
10		1	1	

$y = D_3 + D_1D_2'$

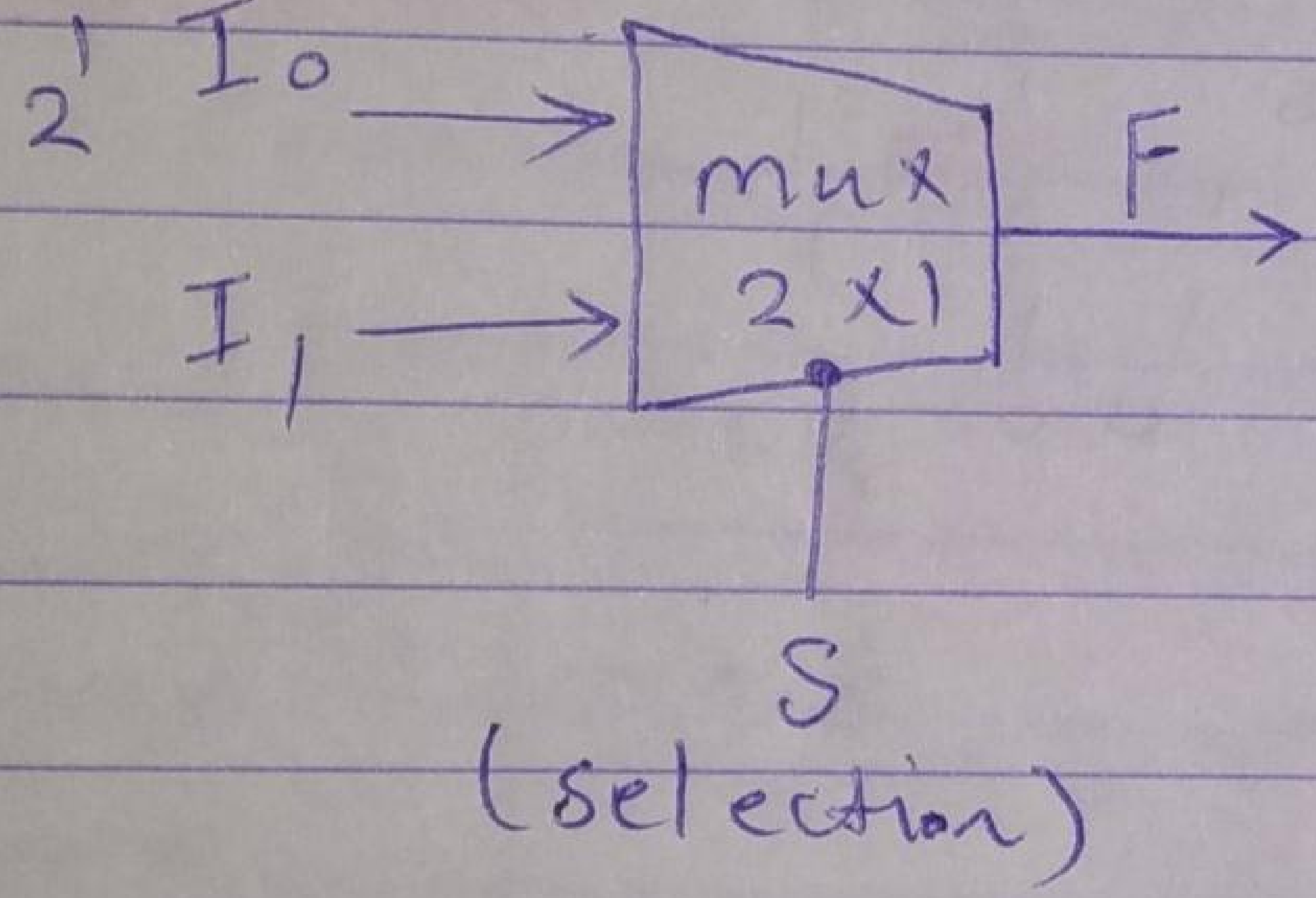
(كتبت ال inputs بدلالة ال outputs).

اختيار من متعدد .

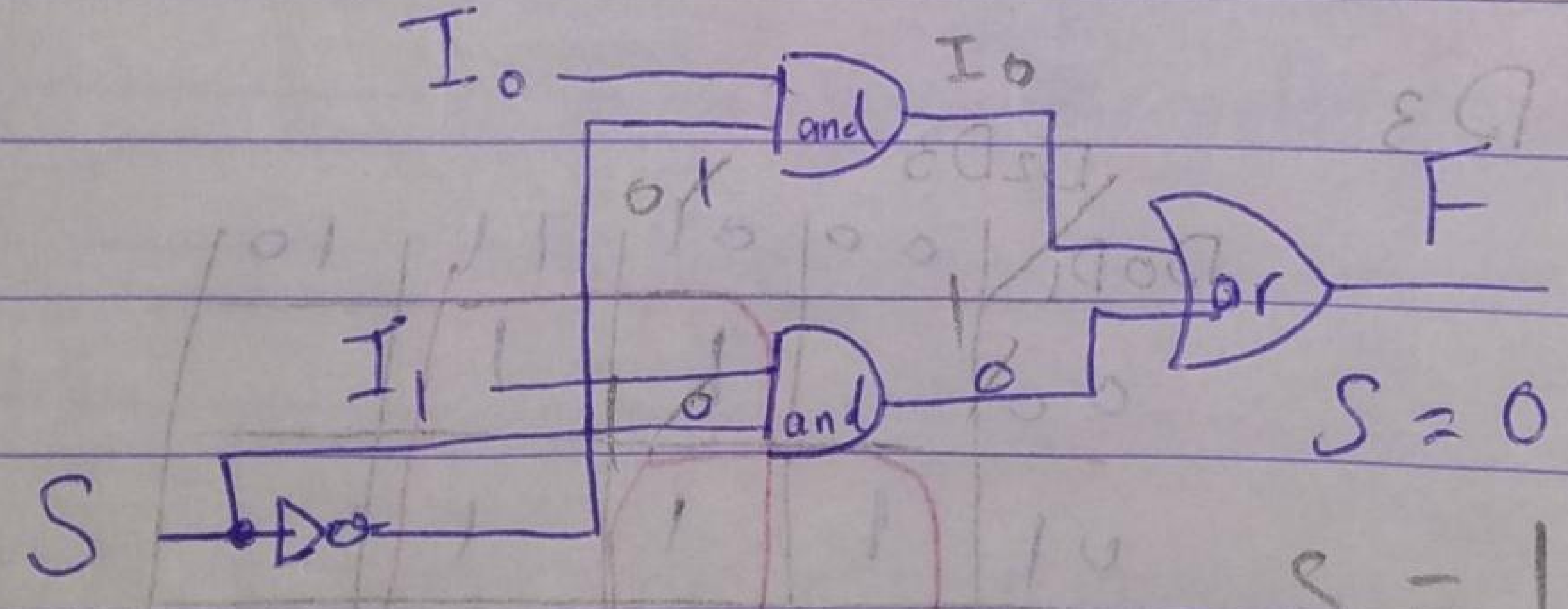
Multiplexer(Mux) :- Combinational circit .



example :- Mux 2 x 1



if (S == 0)
F = I_0
else S = =1
F = I_1

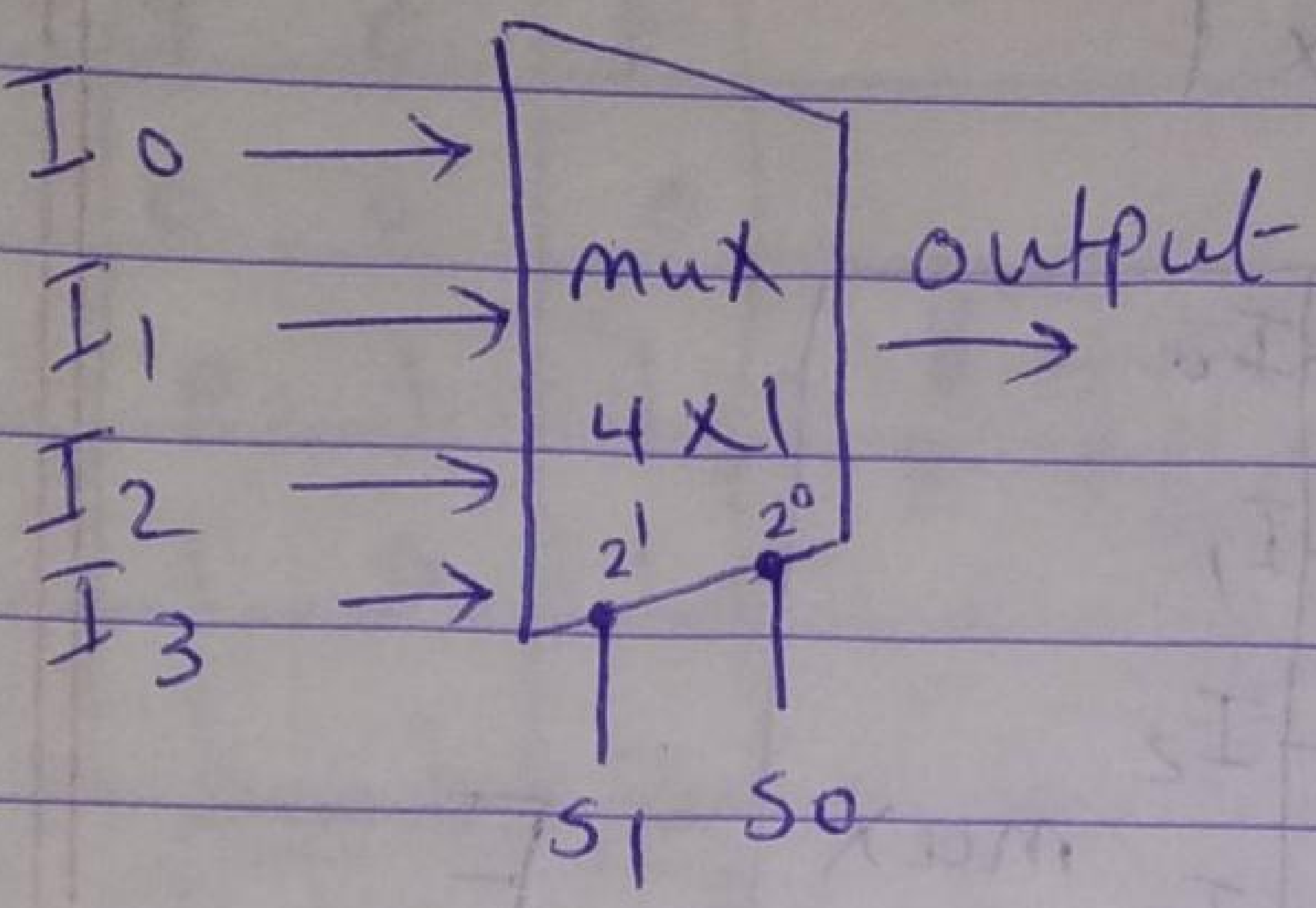


S = 0 F = I_0

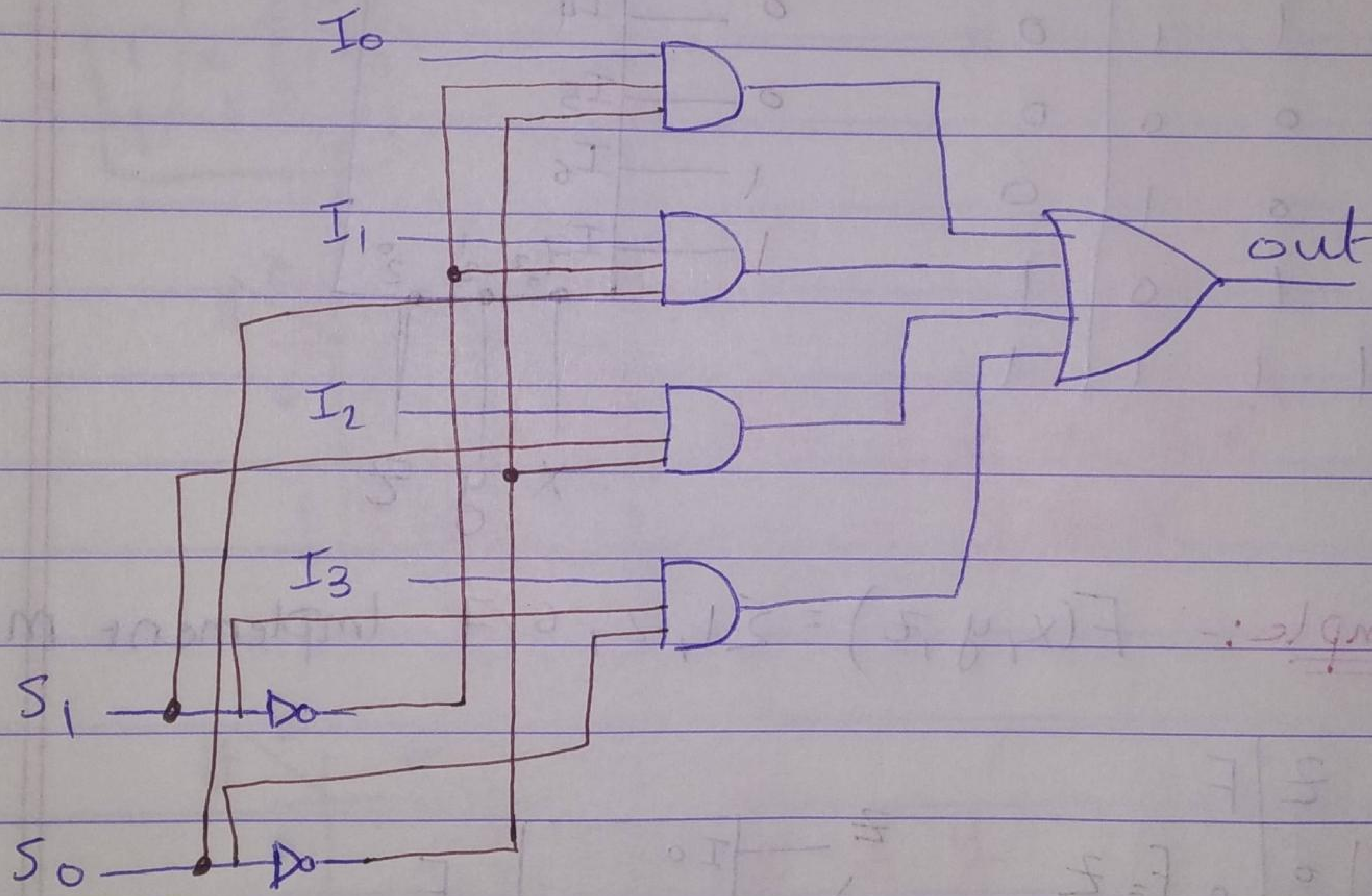
S = 1 F = I_1

Scanned by TapScanner

example:- Mux 4 x 1



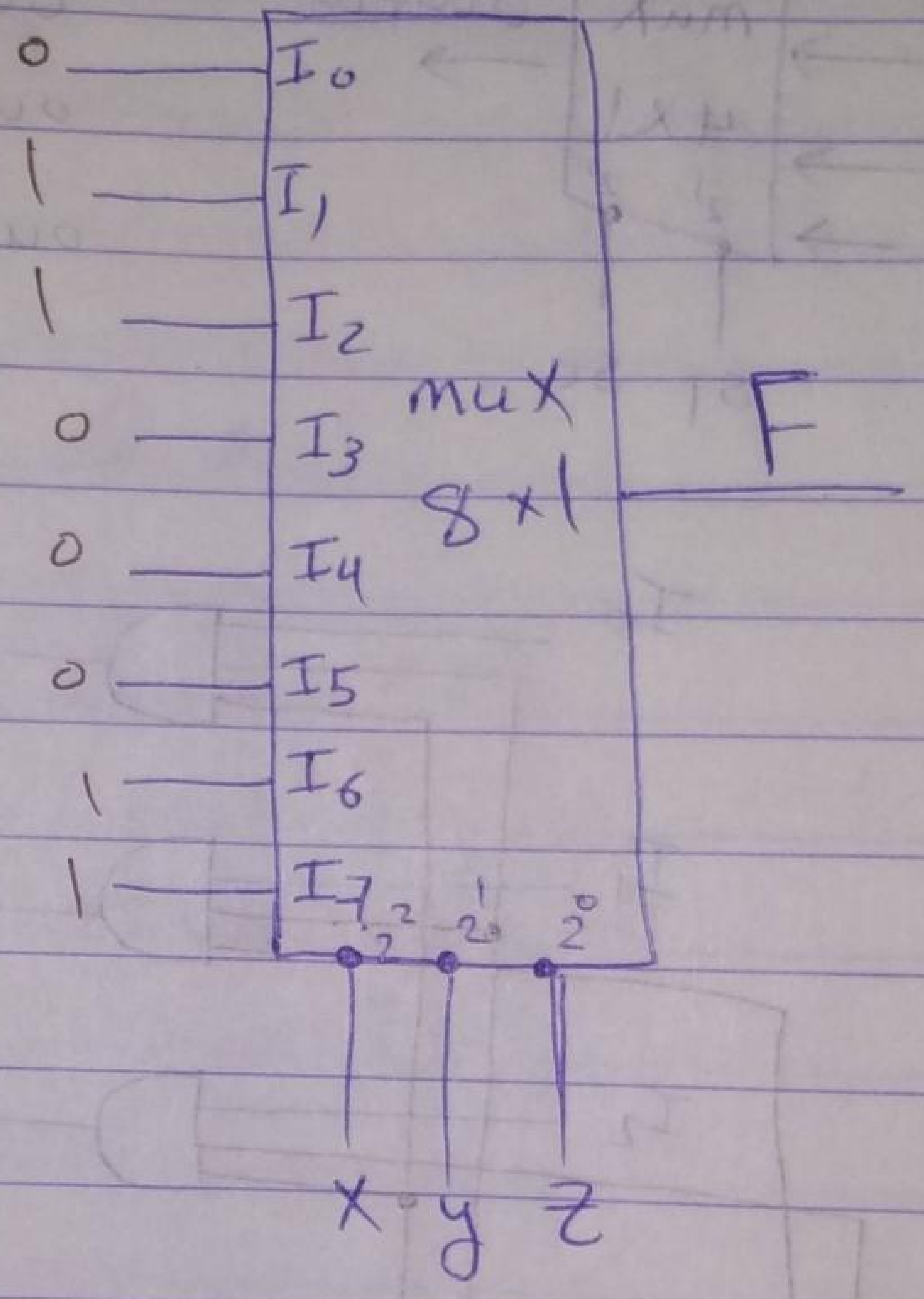
out = I_0	S_1S_0 = 00
out = I_1	S_1S_0 = 01
out = I_2	S_1S_0 = 10
out = I_3	S_1S_0 = 11



example:- $F(x, y, z) = \Sigma 1, 2, 6, 7$

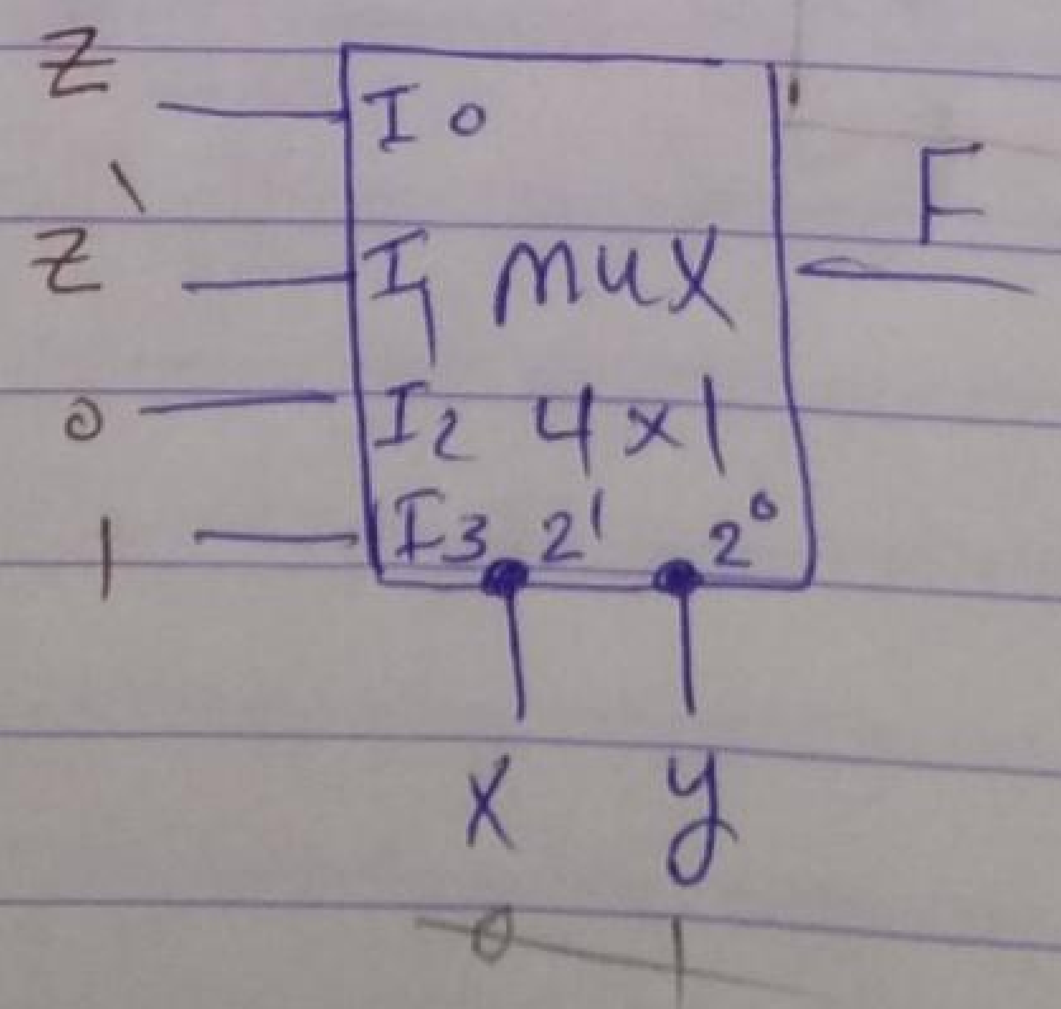
Implement the Function using mux 8 x 1

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



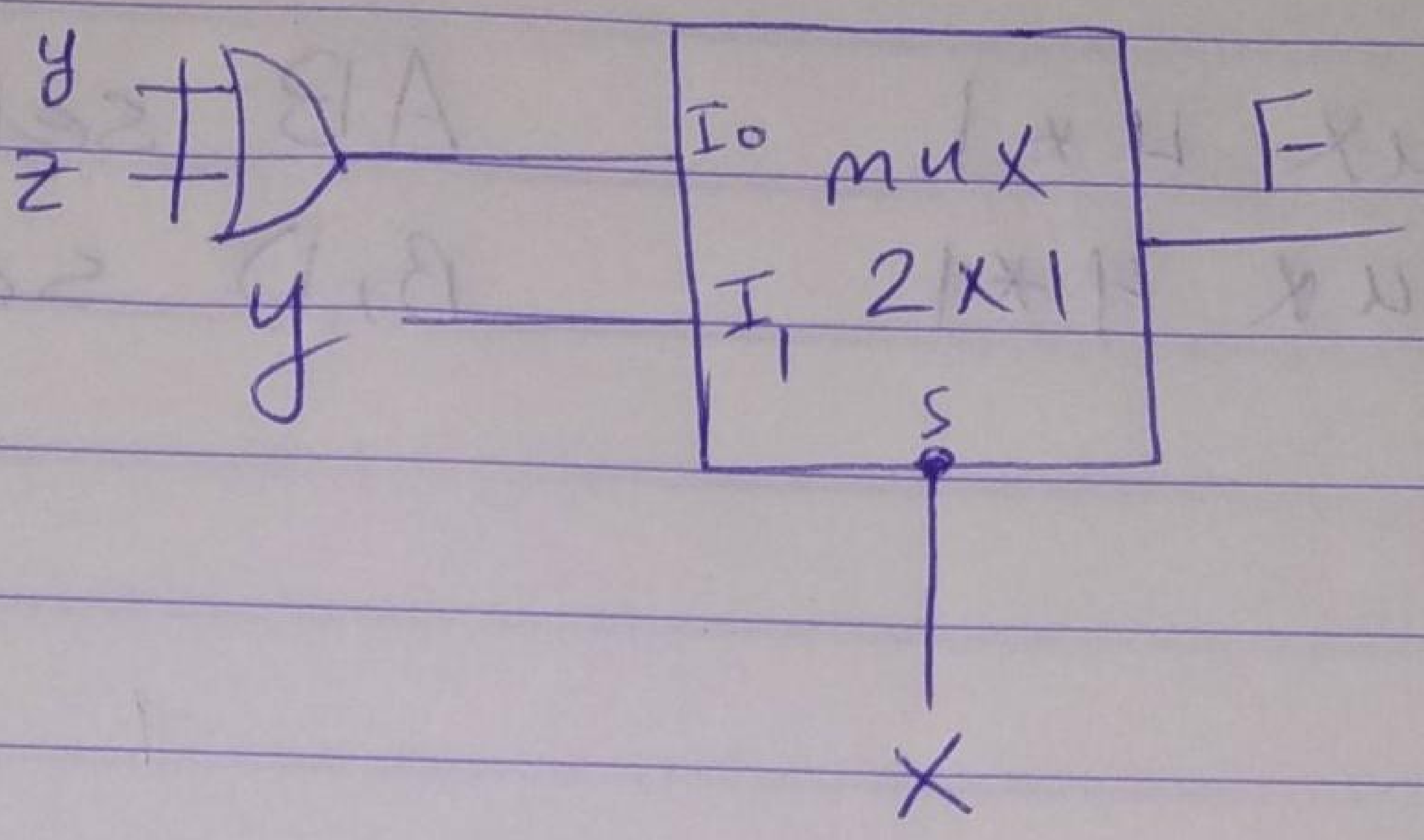
example:- $F(x, y, z) = \Sigma 1, 2, 6, 7$ Implement mux 4x1

x	y	z	F	
0	0	0	0	F = z
0	0	1	1	
0	1	0	1	F = z'
0	1	1	0	
1	0	0	0	F = 0
1	0	1	0	
1	1	0	1	F = 1
1	1	1	1	



example $F(x, y, z) = \sum 1, 2, 6, 7$ Implement mux 2X1

X	y	Z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



y \ z	0	1
0		1
1	1	

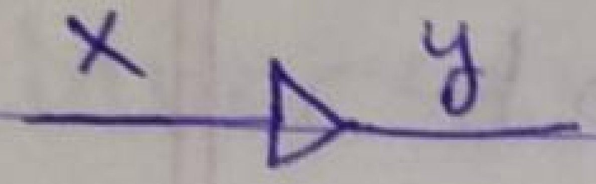
$F = y'z + yz'$
$= y \oplus z$

y \ z	0	1
0		
1	1	1

$F = y$

Scanned by TapScanner

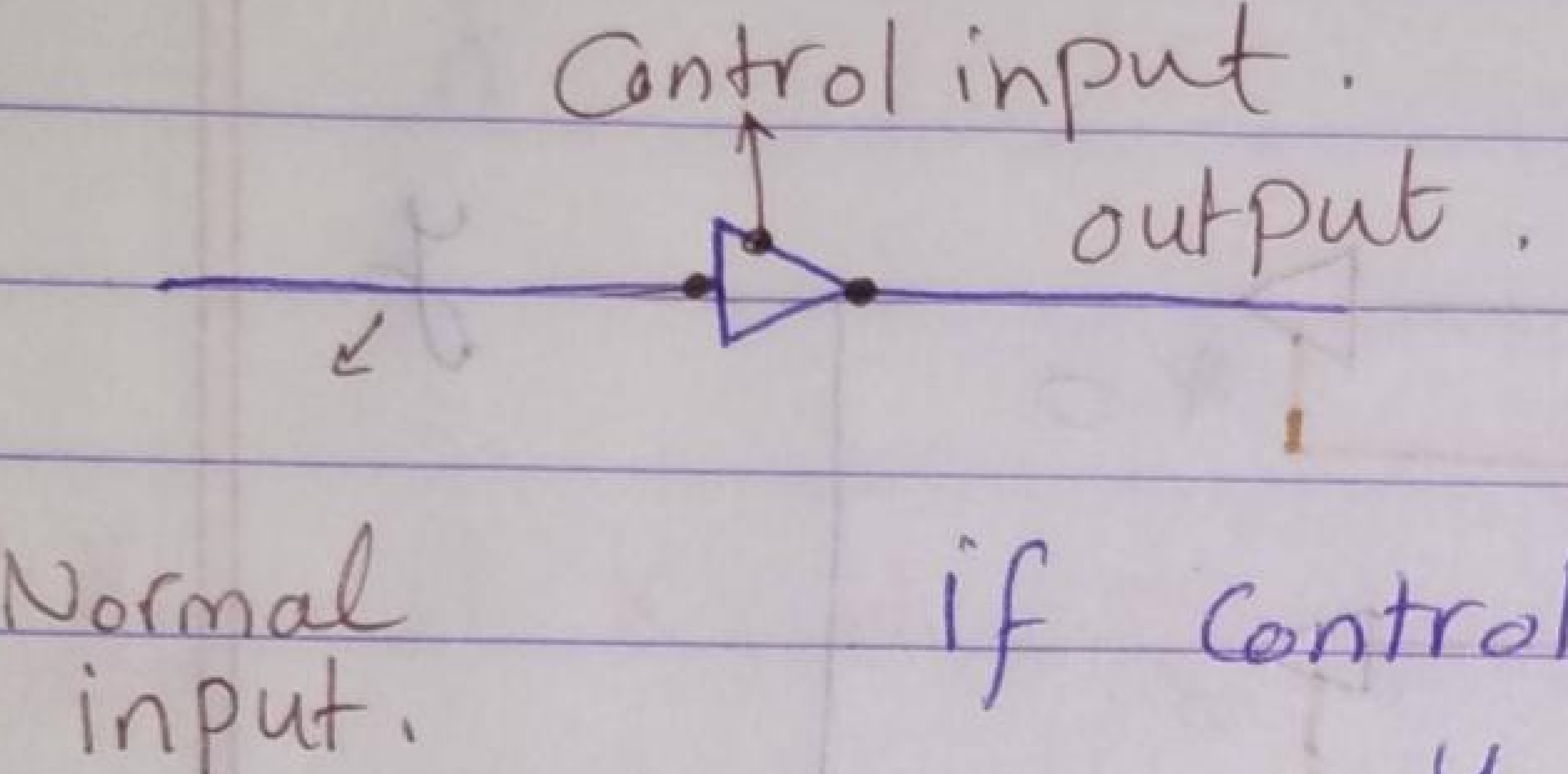
Three (Tri) State buffer



Normal buffer.

$y = x$ (delay).

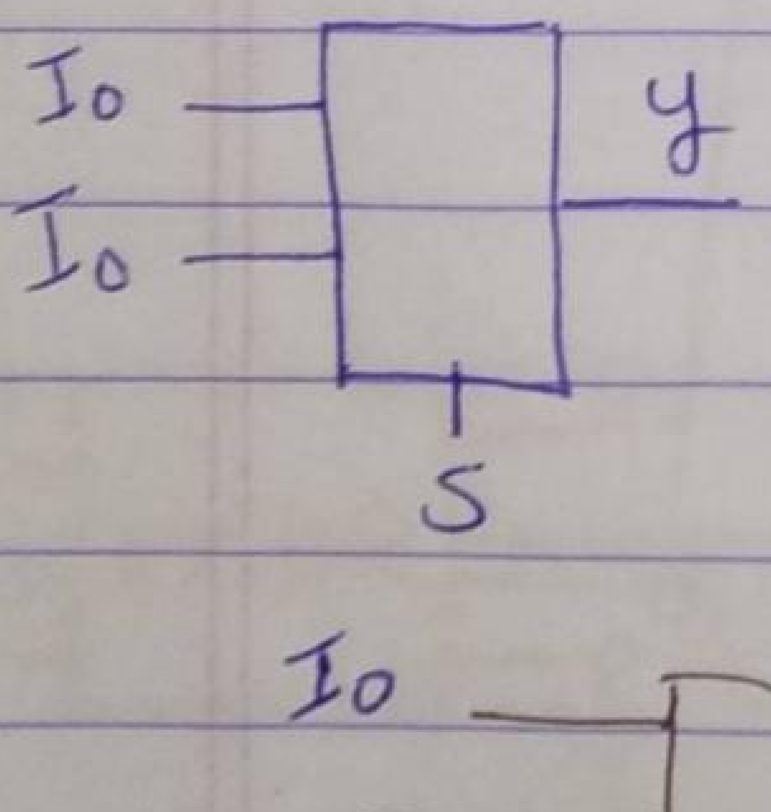
x	y
0	0
1	1



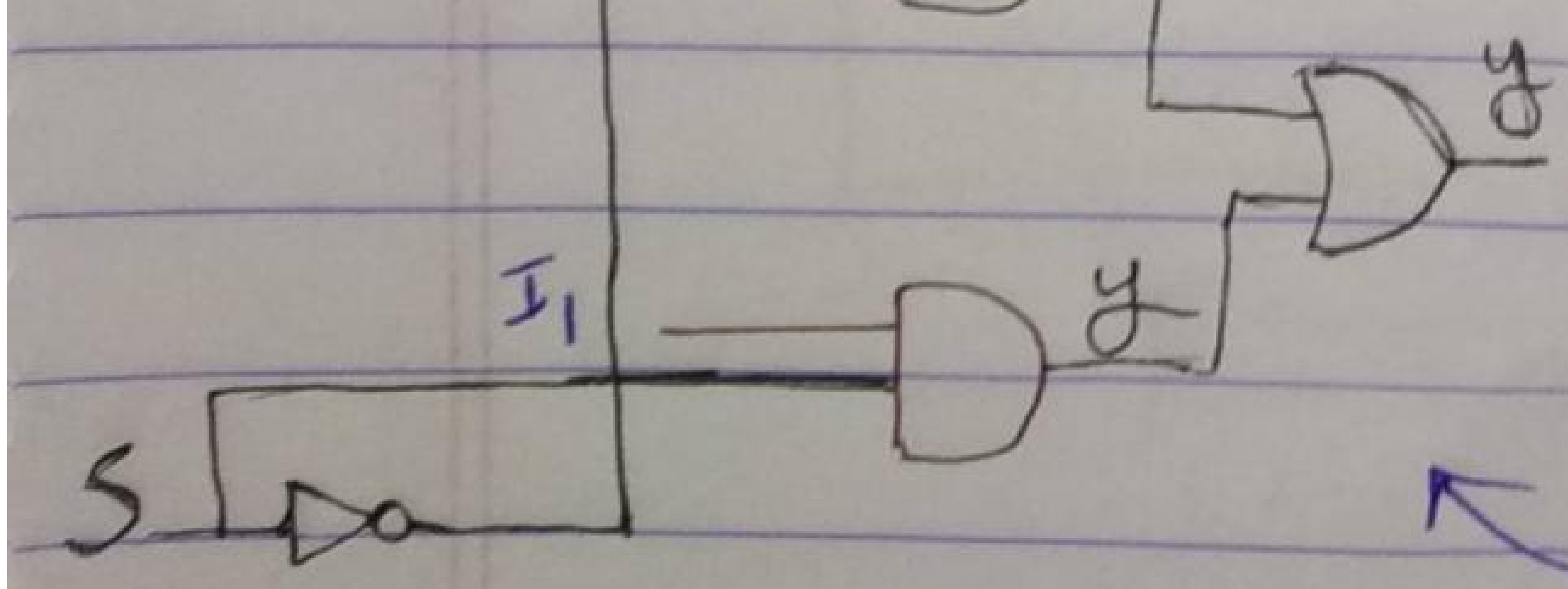
if Control input (c) = 1
$y = \alpha$.

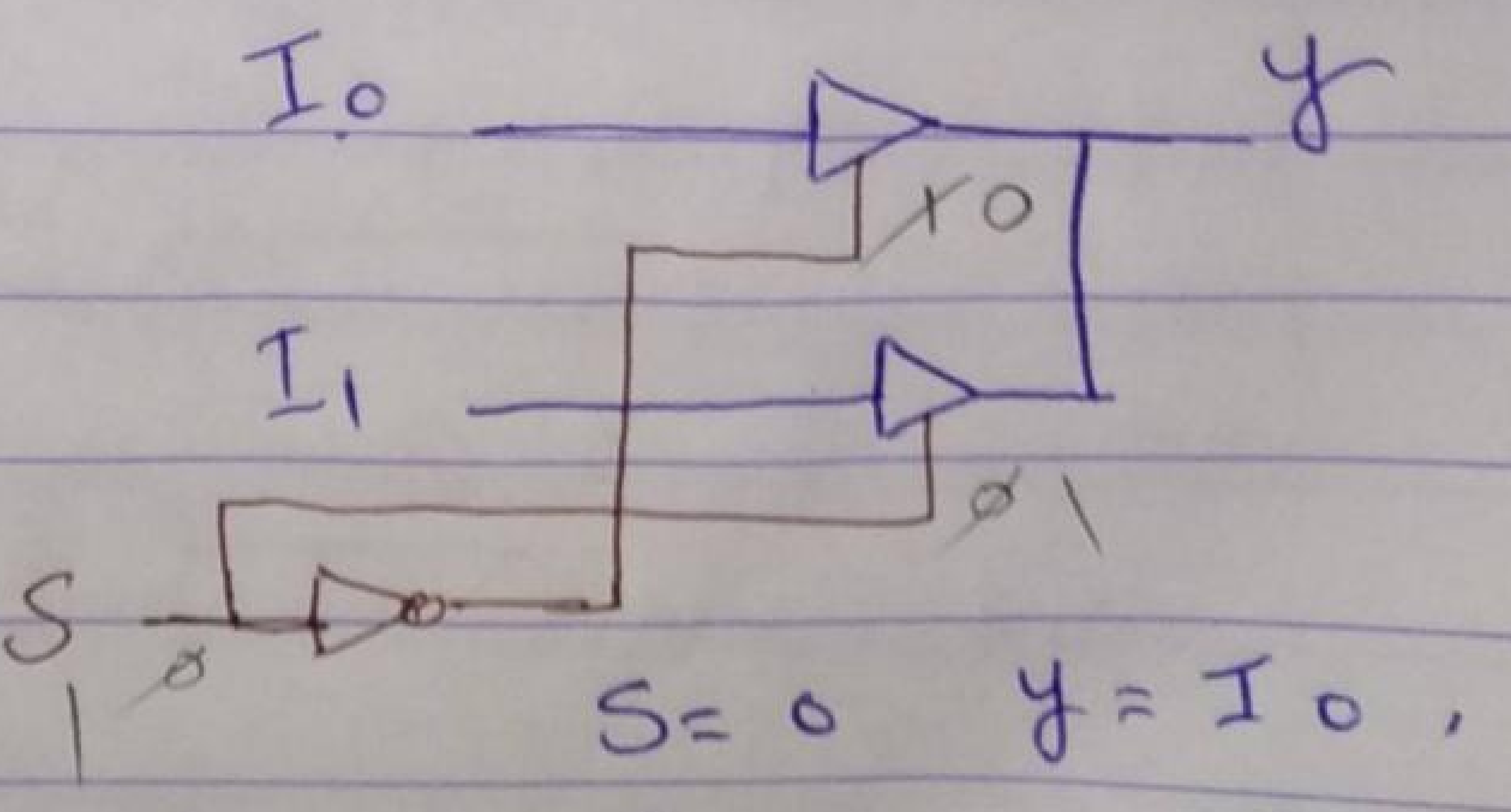
if $c == 0$ (open circit high imp

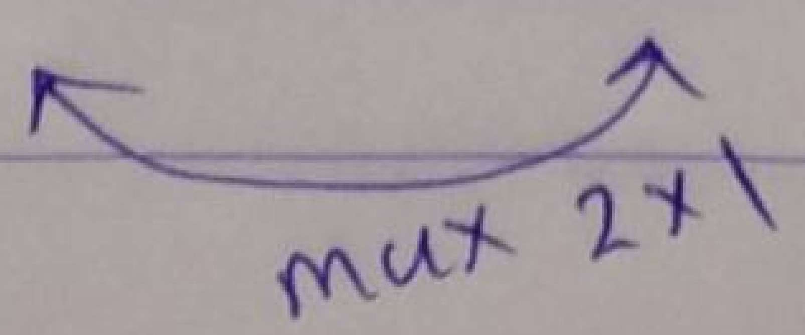
ex 2x1 mux.



$y = I_0 , S = 0$
$y = I_1 , S = 1$

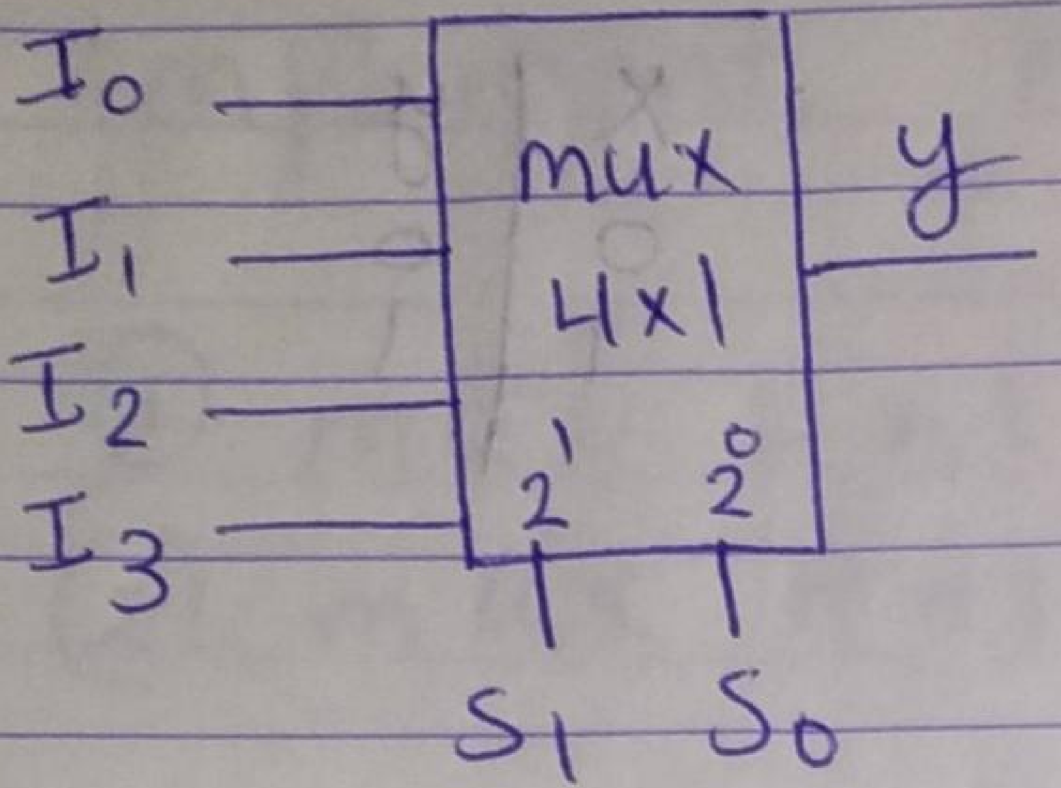






S = 0 $y = I_0$.
S = 1 $y = I_1$.

ex mux 4x1

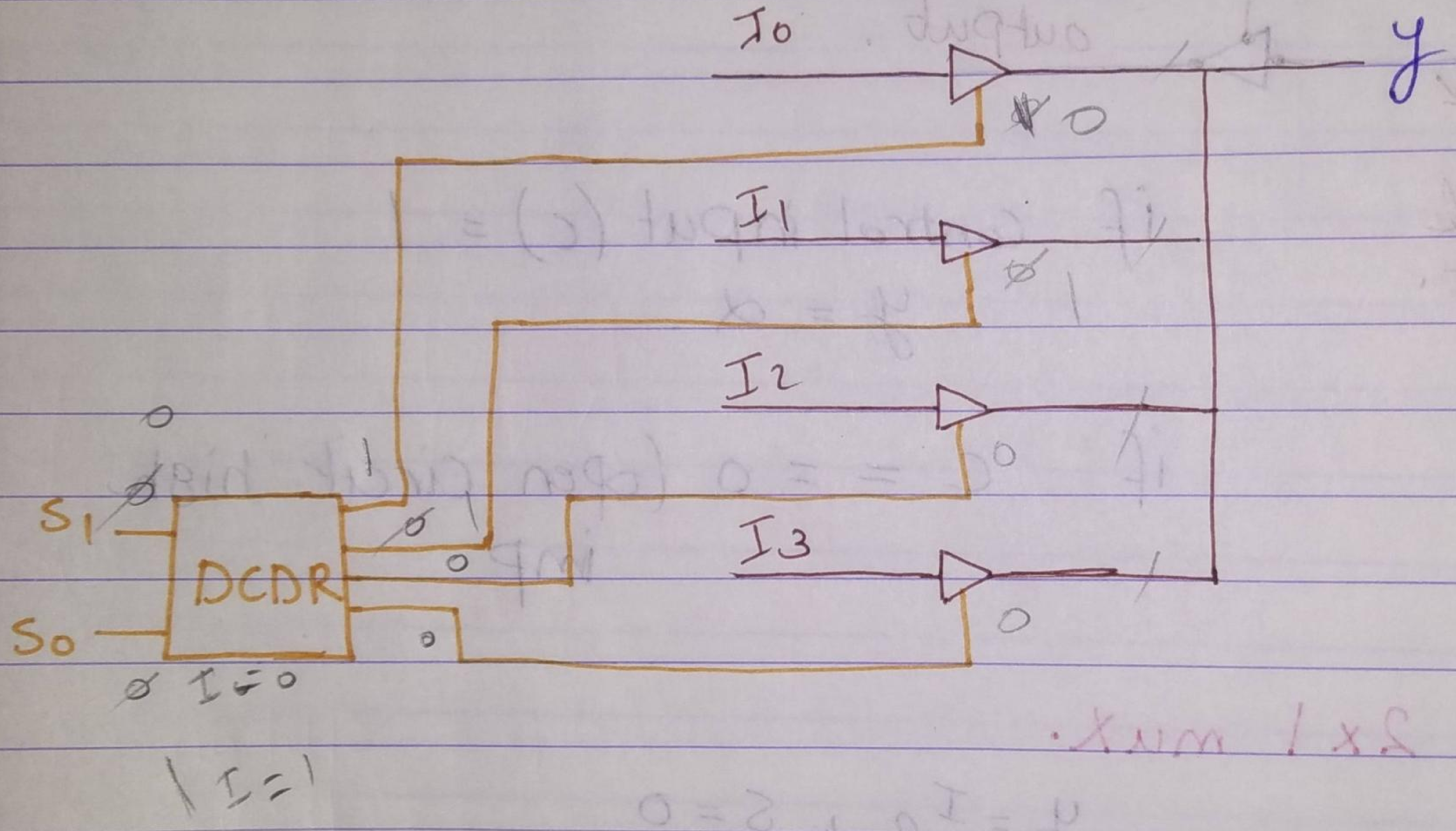


$y = I_0$, $S_1S_0 = 00$

$y = I_1$, $S_1S_0 = 01$

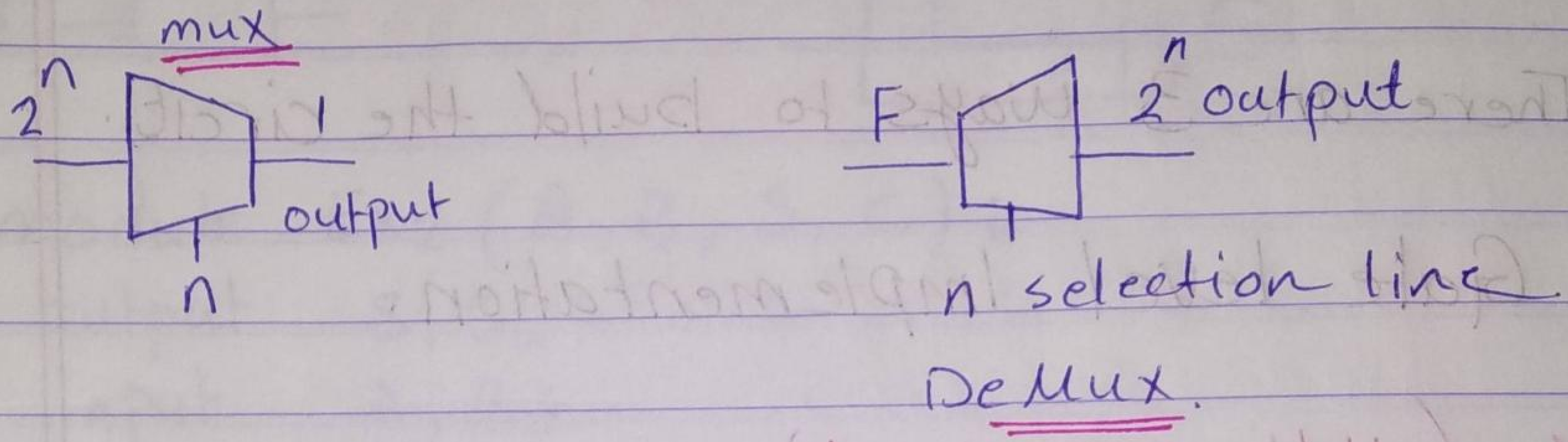
$y = I_2$, $S_1S_0 = 10$

$y = I_3$, $S_1S_0 = 11$

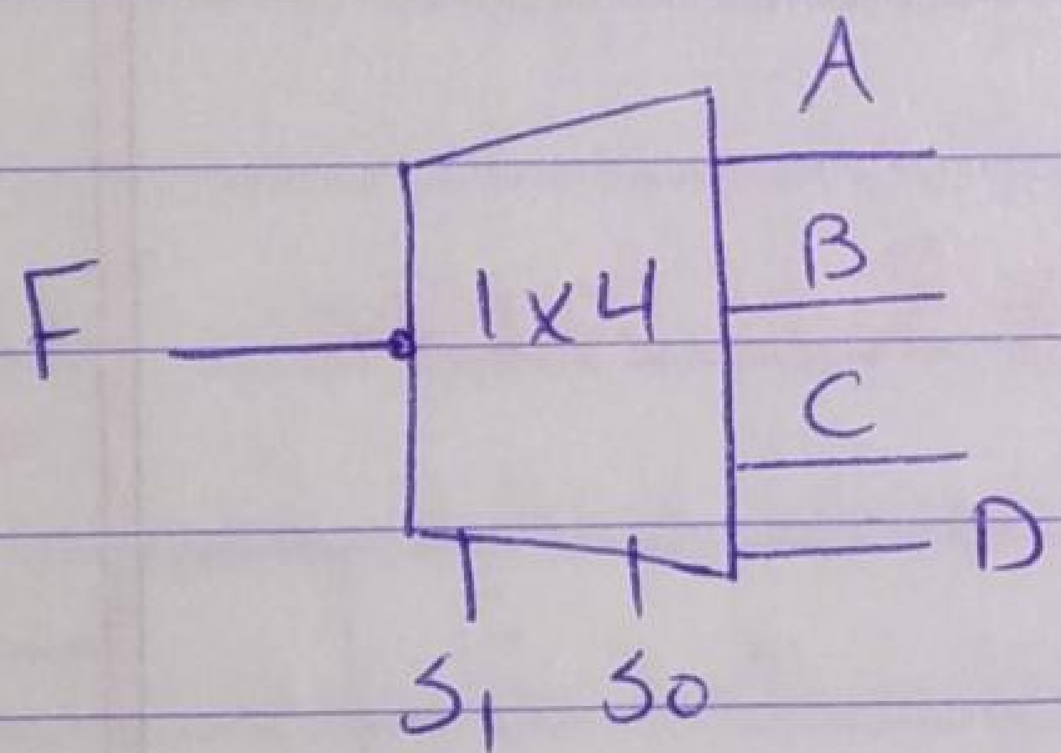


Demultiplexer (DeMux) :-

⇛ Inverse operation of multiplexer.



ex 1 x 4 DeMux.

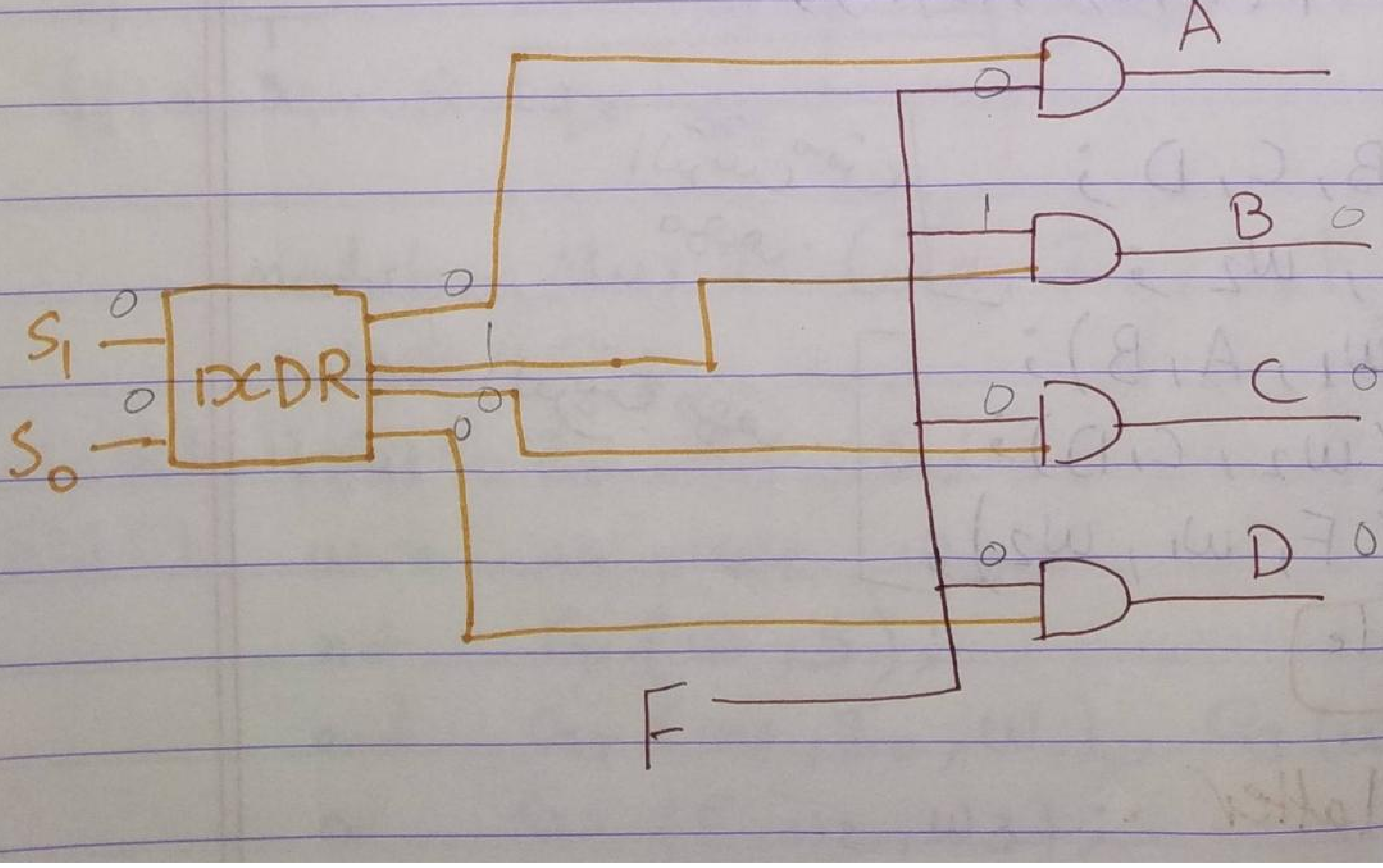


$A = F$, $S_1S_0 = 0\ 0$

$B = F$, $S_1S_0 = 0\ 1$

$C = F$, $S_1S_0 = 1\ 0$

$D = F$, $S_1S_0 = 1\ 1$



Scanned by TapScanner

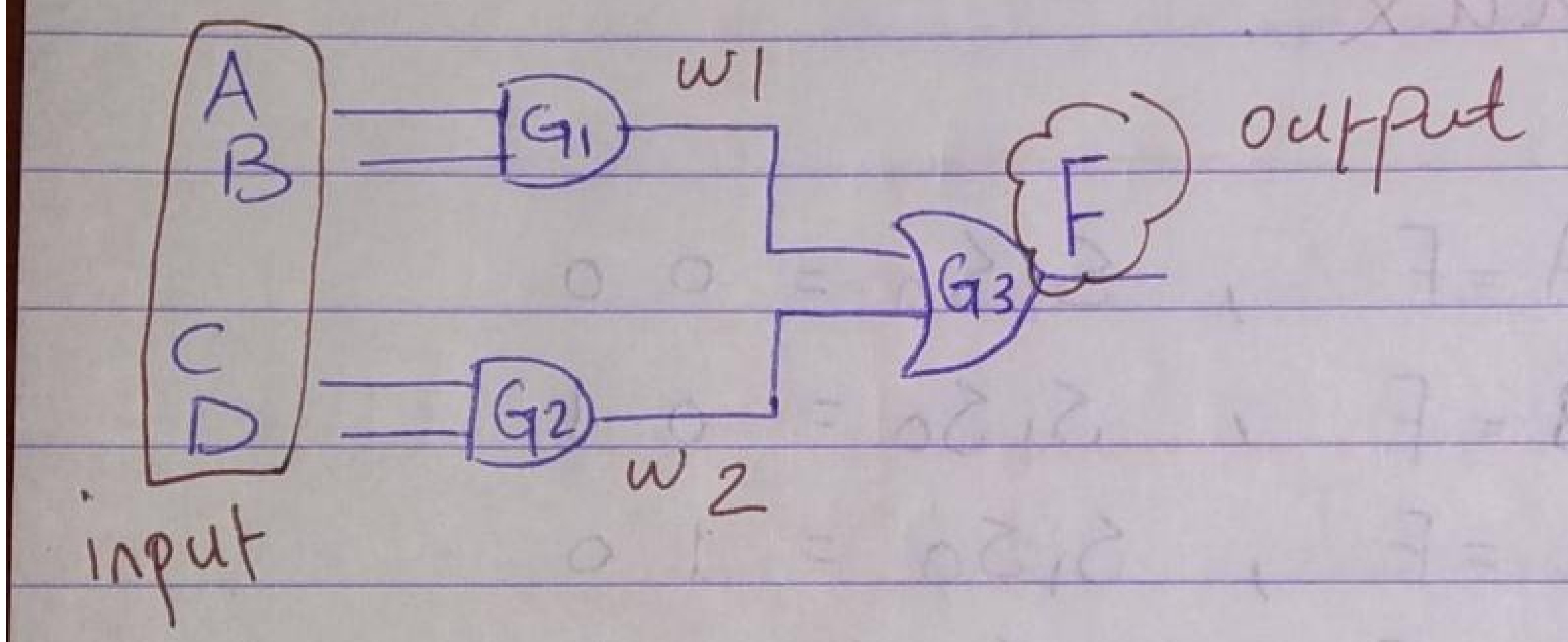
programming for Combinational circit.

HDL (Hardware Description language) :-

There are 3 ways to build the circit.

① Gate level Implementation.

ex (HdL-gate level)



```
module Ex1(A,B,C,D,F);
output F;
input A,B,C,D;
wire w1, w2;
and G1(w1,A,B);
and G2(w2,C,D);
or G3(F,w1,w2);
endmodule
```

الترتيب مش مهم.

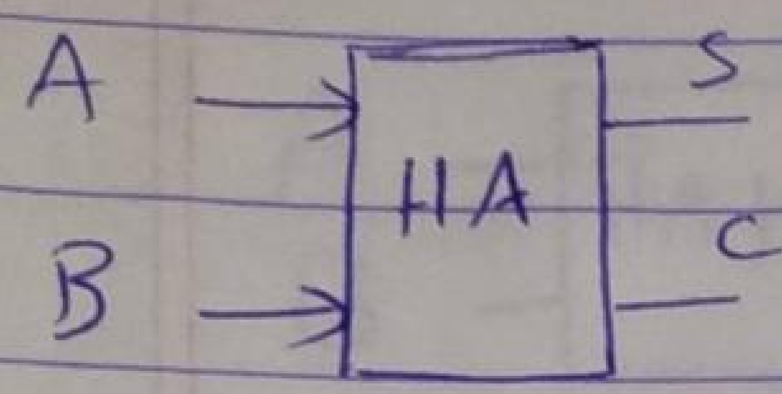
الترتيب مهم.

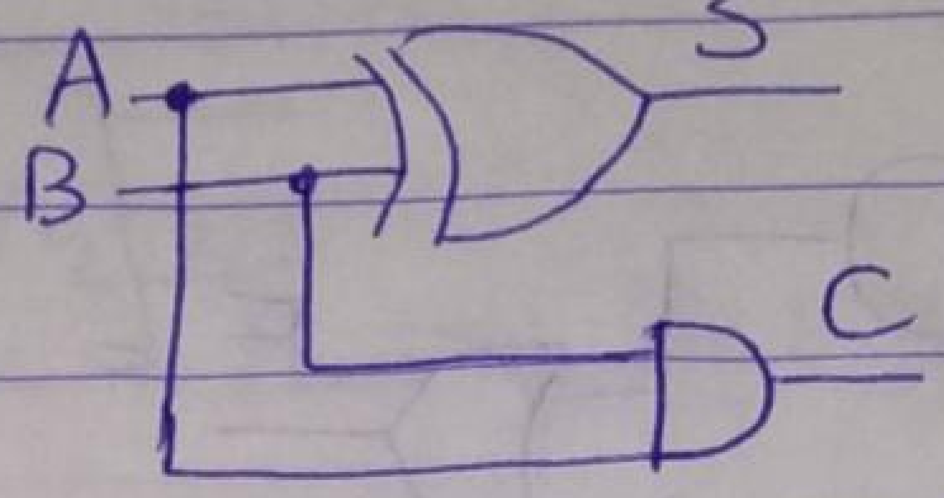
small latter.

Scanned by TapScanner

ex Implementation HA (half-adder).

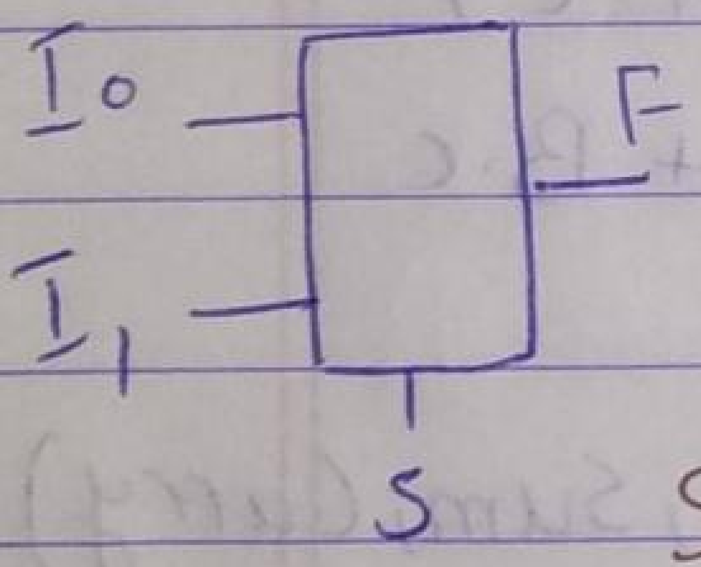
Gate level ⟹

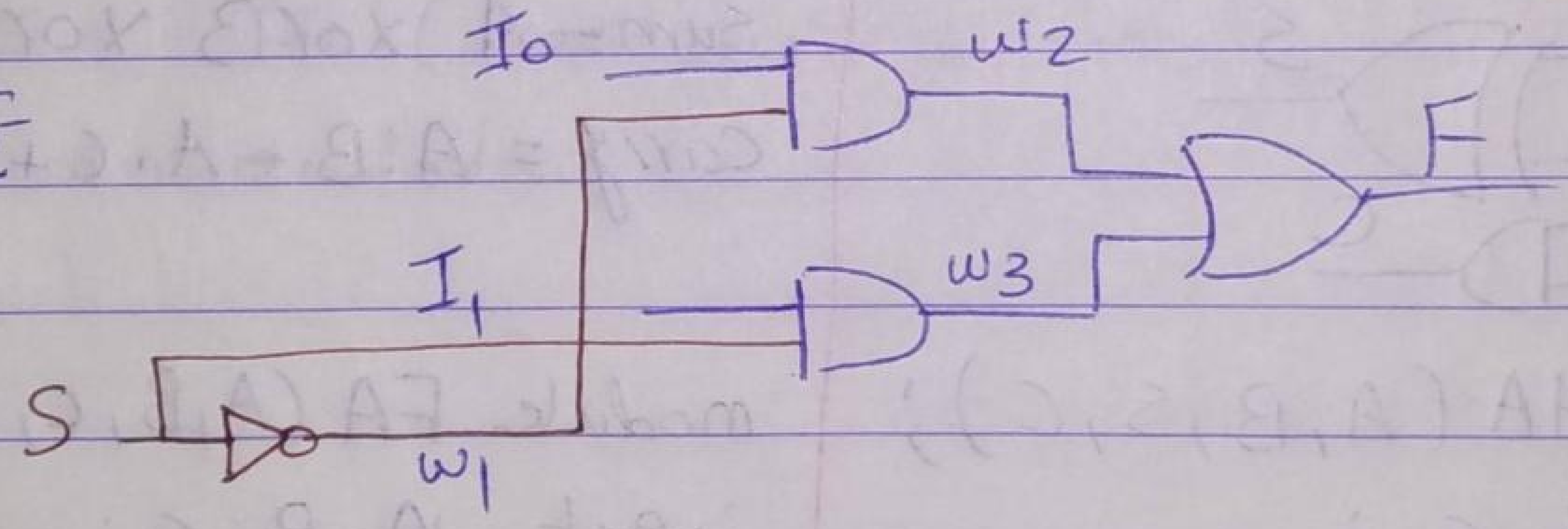




```
module Ex2 (A, B, S, C);
 output  S, C;
 input  A, B;
  xor   G1 (S, A, B);
  and   G2 (C, A, B);
    endmodule.
```

ex mux 2x1 Gate-level:





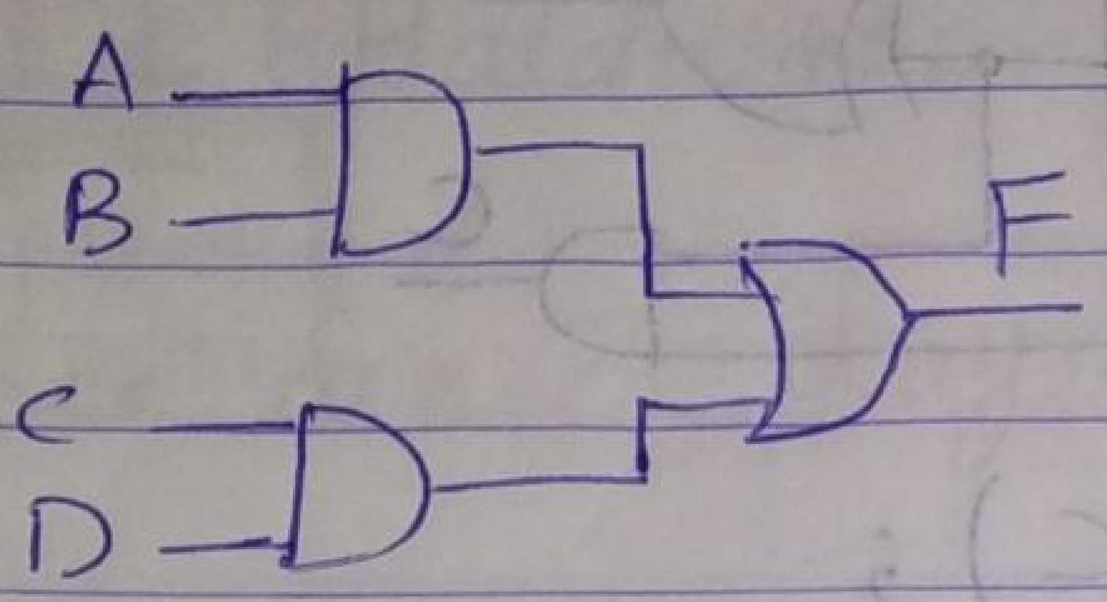
```
module Mux21 (I0, I1, S, F);
 output F;
 input I0, I1, S;
 wire  w1, w2, w3;
 not   G1 (w1, S);
 and   G2 (w2, I0, w1), G3 (w3, I1, S);
 or    G4 (F, w2, w3);
   endmodule
```

Scanned by TapScanner

(wire مافيها)

2 Data Flow implementation. (assign).

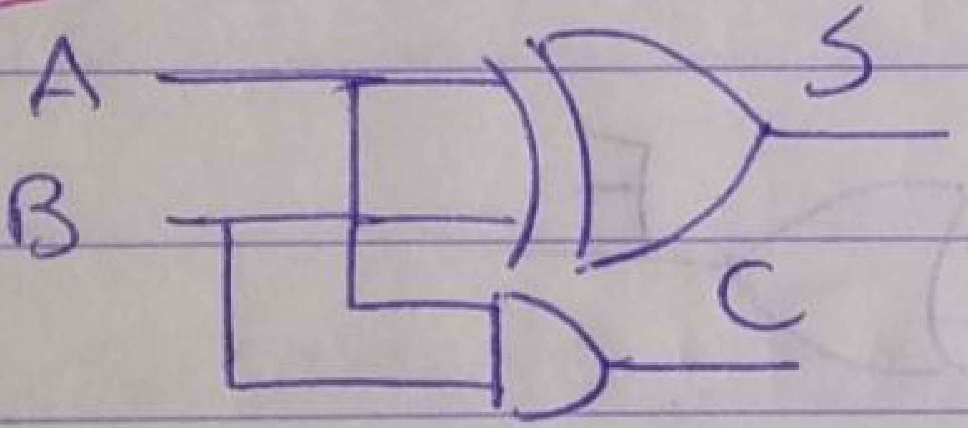
ex



F = A·B + C·D

```
module  Ex (A, B, C, D, F);
input   A, B, C, D;
output  F;
assign  F = (A && B) || (C && D);
     endmodule .
```

ex HA



```
module HA (A, B, S, C);
output  S, C;
input   A, B;
assign S = A ^ B;
assign C = A && B;
     endmodule
```

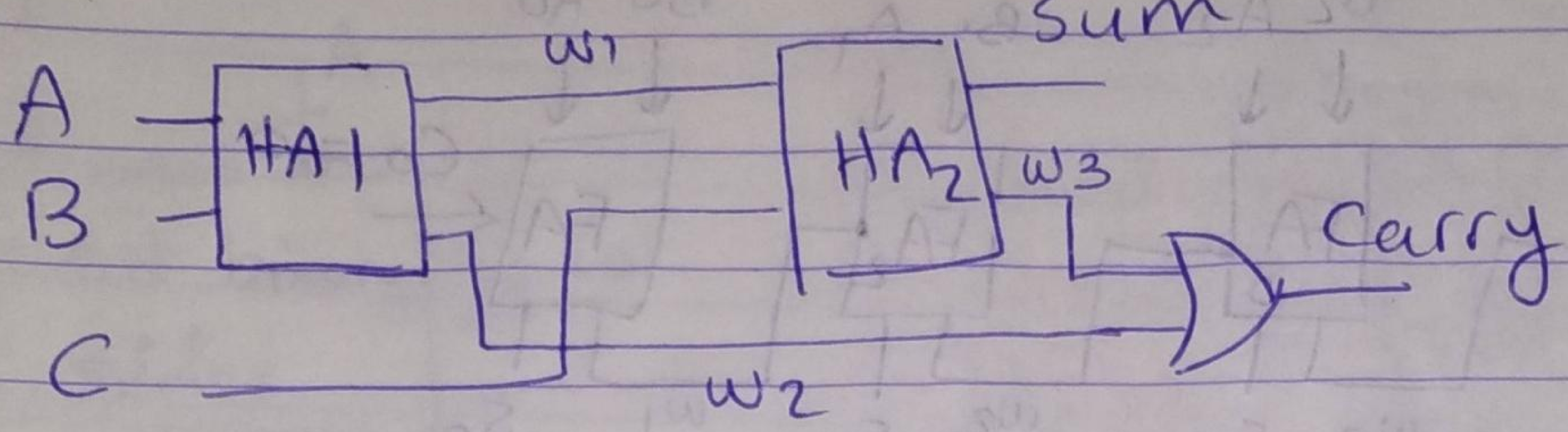
Full-adder.

sum = A xor B xor C;

carry = A.B + A.C + B.C

```
module FA (A, B, C, Sum, Carry);
input  A, B, C;
output  Sum, Carry;
assign sum  = A ^ B ^ C;
assign carry = (A&&B) || (A&&C) ||
                        (B&&C);
       endmodule
```

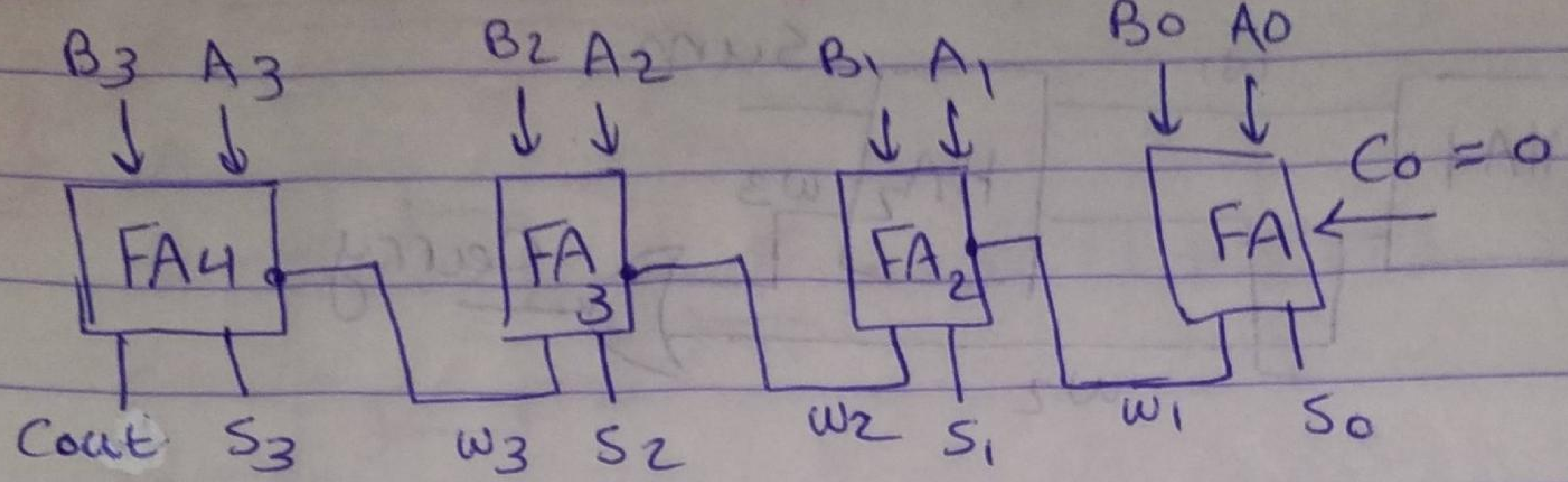
⇒ تكملة



Structure :-

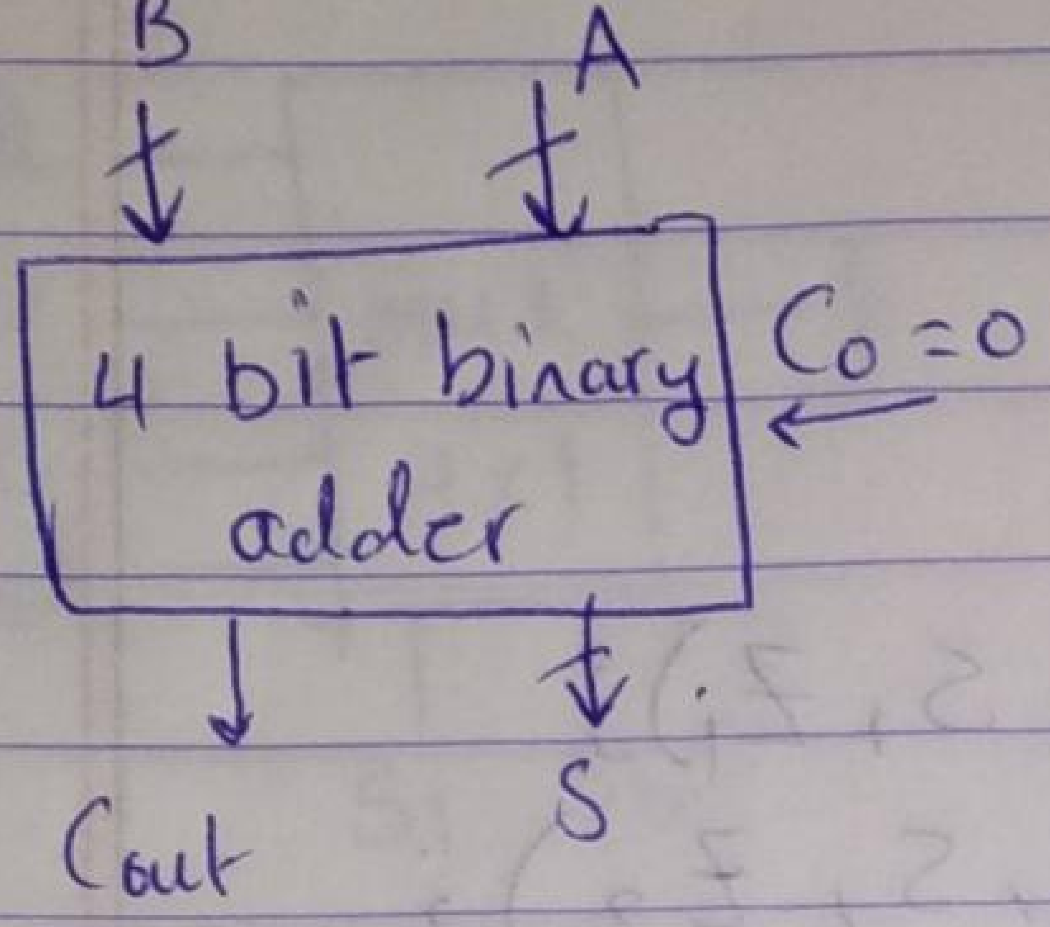
```
module FA (A, B, C, Sum, carry);
 input  A, B, C;
 output  Sum, carry;
  wire  w1, w2, w3;
   HA  HA1 (A, B, w1, w2);
   HA  HA2 (w1, C, Sum, w2);
   or  G1 (carry, w3, w2);
      endmodule
```

ex 4 bit Binary adder



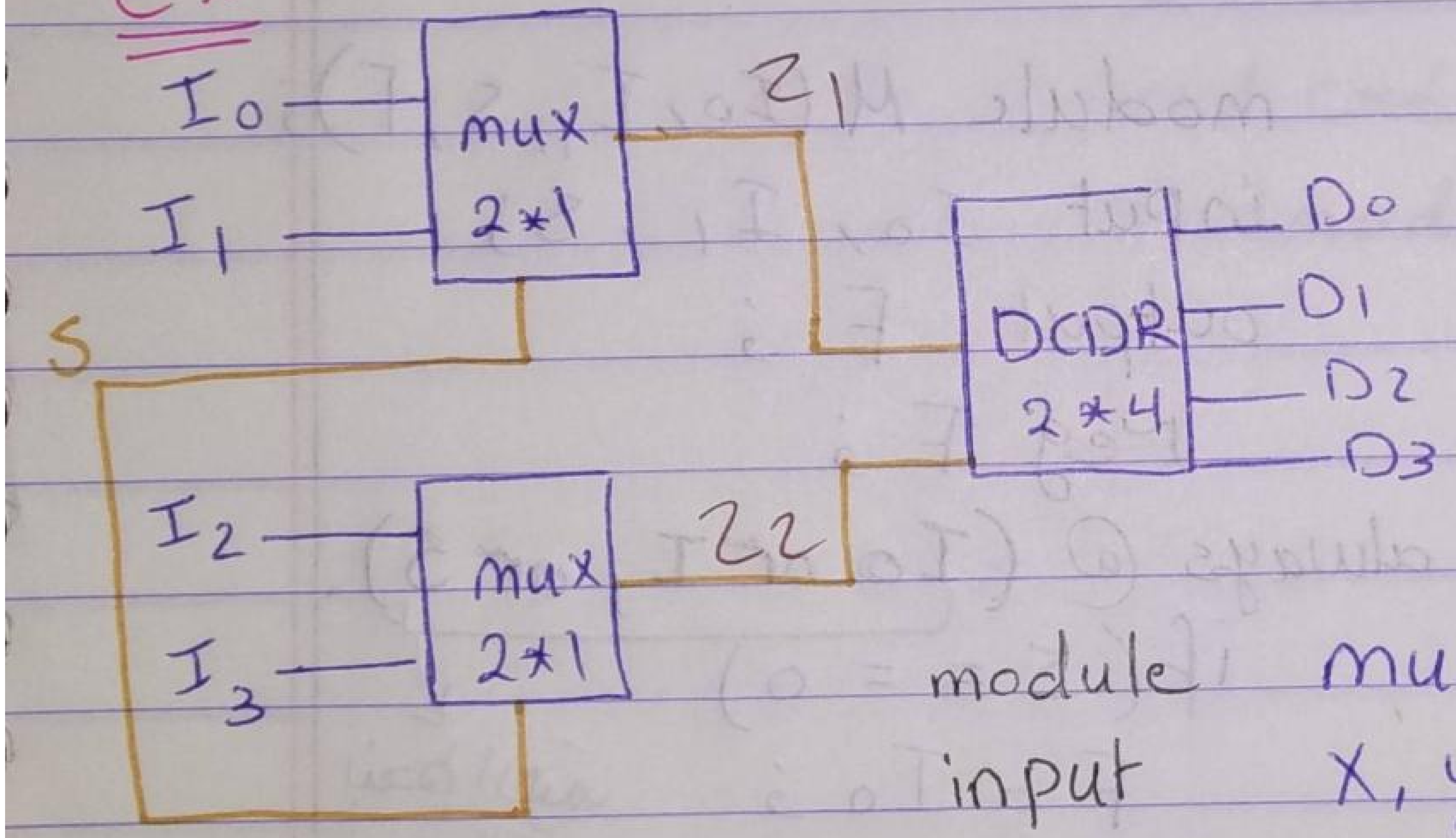
```
module 4BA(A, B, Co, S, Cout);
input  Co;
input  [3:0] A, B;
output [3:0] S;
output Cout;
wire   w1, w2, w3;
FA     FA1 (A[0], B[0], Co, S[0], w1);
FA     FA2 (A[1], B[1], w1, S[1], w2);
FA     FA3 (A[2], B[2], w2, S[2], w3);
FA     FA4 (A[3], B[3], w3, S[3], Cout);
   endmodule
```

* Data Flow (4 bit Binary adder).

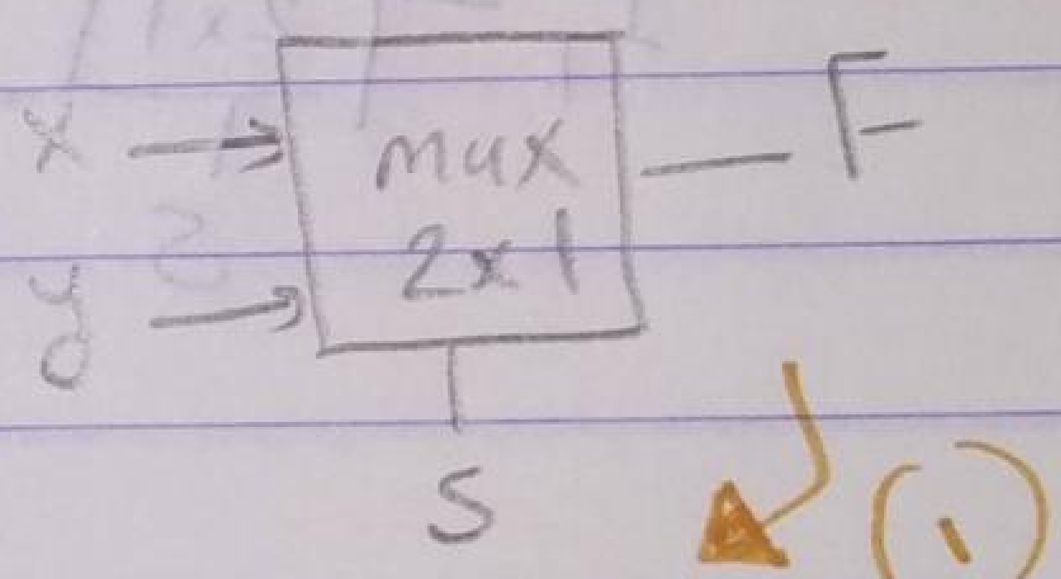


```
module BA(A, B, C0, Cout, S);
input [3:0] A, B;
input C0;
output [3:0] S;
output Cout;
assign {Cout, S} = A + B + C0;
    endmodule
```

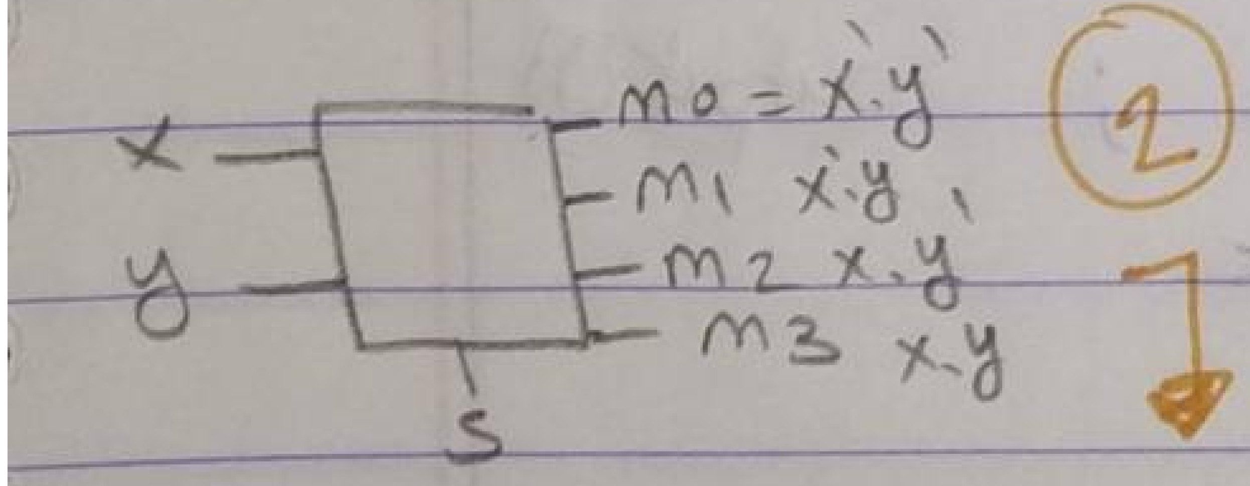
ex



ex



```
module mux21(x, y, s, F);
input x, y, s;
output F;
assign F = (x && !s) || (y && s);
    endmodule
```



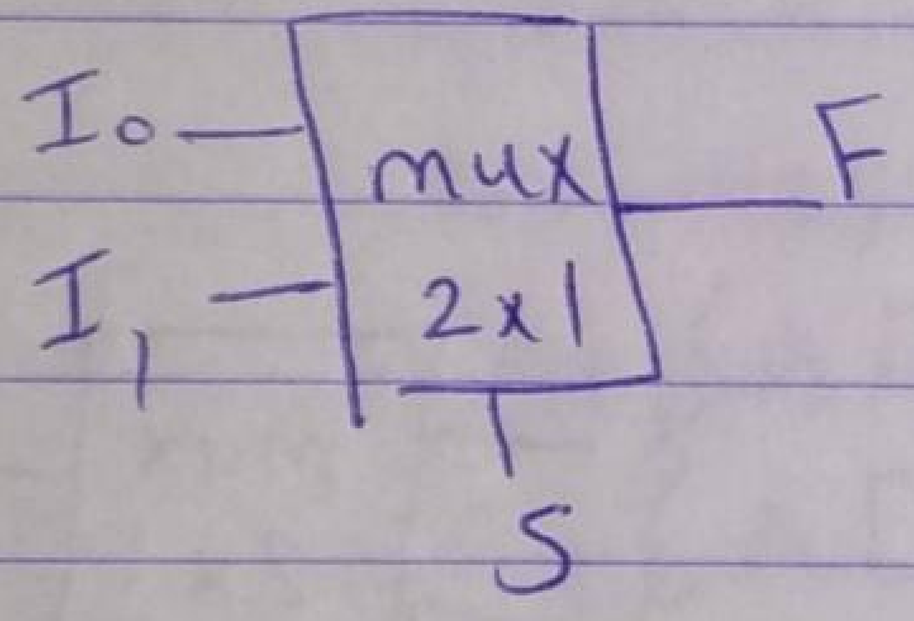
```
module DCDR (x, y, m);
input x, y;
output [0:3] m;
assign m[0] = ( ! x && !y);
       m[1] = ( ! x && y);
       m[2] = ( x && !y);
       m[3] = ( x && y); endmodule.
```

3

```
module  system (I, D, S);
input   [0:3] I;
input   S;
output  [0:3] D;
 wire  ~~wire~~ z1 , z2 ;
  mux21   M1 (I [0], I [1], S, Z1);
  mux21   M2 (I[2], I[3], S, Z2);
 endmodule
```

3 Behaviral (always).

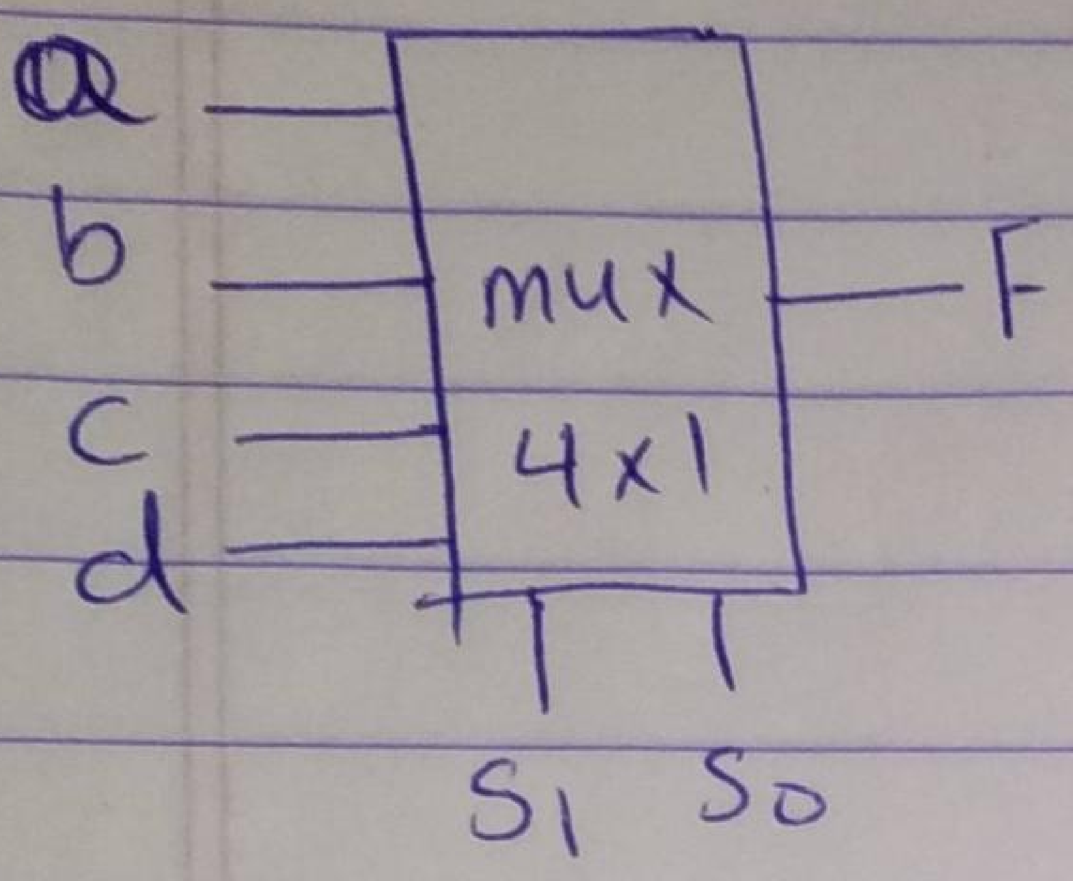
ex



```
module M (I0, I1, S, F);
input  I0, I1, S;
output  F ;
   reg F ;
always @ (I0 or I1 or S).
  if(S = = 0)
     F = I0 ;
   else
     F = I1 ;
 endmodule
```

بنحط القيمَ
الي بتأثر على
ال output

mux 4 x 1



```
module M1 (a,b,c,d,So,S1,F);
input  a,b,c,d,So,S1;
  output  F;
  reg  F;
always@(a or b or c or d or So,S1)
  Case {S1 So}
    2'b00 : F<=a;
    2'b01 : F<=b;
    2'b10 : F<=C;
    2'b11 : F<=d;
    end case
    endmodule
```

Binary