Chapter-05: Combinational Logic with MSI and LSI

*** Ex-5.1: Design an excess-3-to-BCD code converter using a 4-bit full adders MSI circuit.** (01)

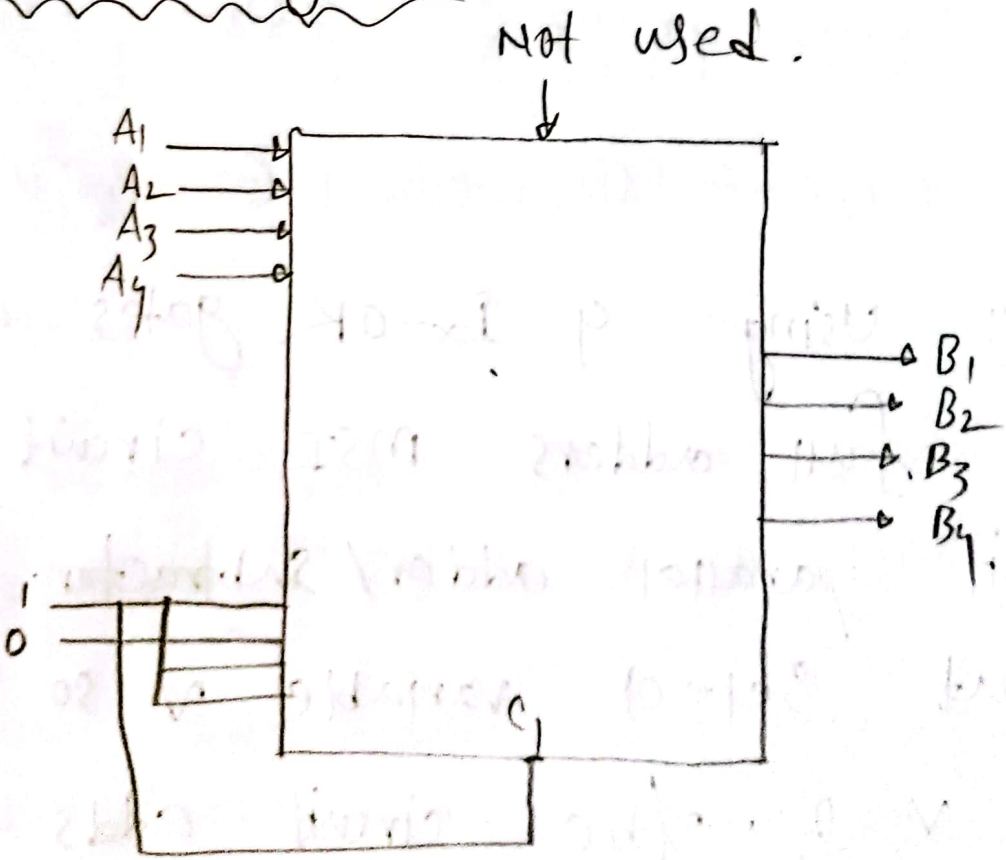
Soln: Excess-3 = BCD + 0011

$\Rightarrow$ BCD = Excess3 − 0011

2's complement of 0011 = 1101

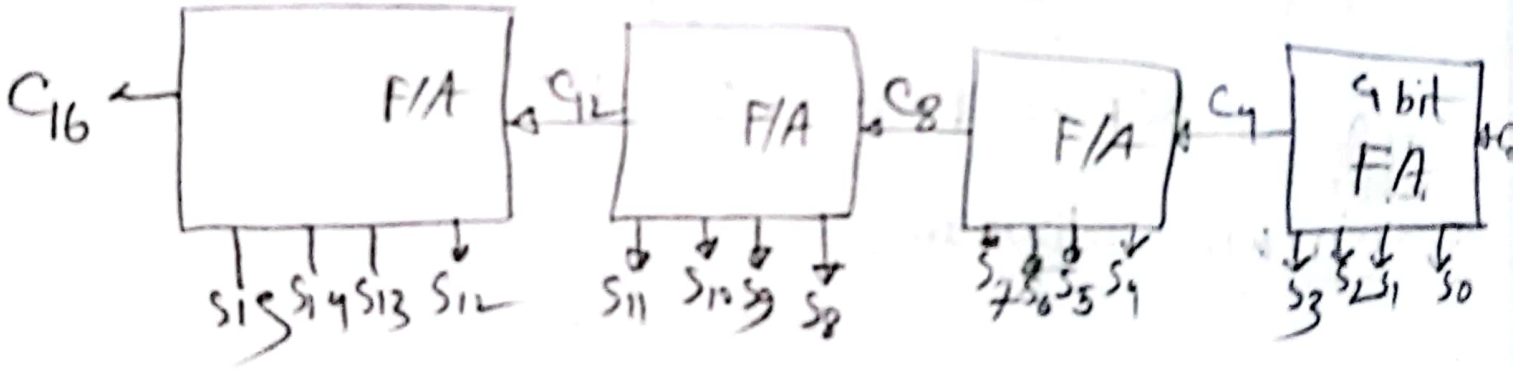
BCD = Excess-3 + 1101

$\Rightarrow$ circuit diagram:



* Ex-5.2: Using four msi circuits. Construct a binary parallel adder to add two 16-bit binary numbers. Label all carries between the MSI circuits.

Soln:



$$C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$$

* Ex-5.3: Using 4 Ex-OR gates and a 4-bit full adders MSI circuit, construct a 4-bit parallel adder/subtractor. Use an input select variable V so that when V=0, the circuit adds and

when V=1, the circuit subtracts.

Soln: we know that,

$$C_1 = G_0 + P_0 C_{in}$$

$$C_2 = G_1 + P_1 C_1$$
$$= G_1 + P_1 G_0 + P_1 P_0 C_{in}$$

$$C_3 = G_2 + P_2 C_2$$
$$= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in}$$

$$C_4 = G_3 + P_3 C_3$$
$$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$$

$$C_5 = G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1 + P_4 P_3 P_2 P_1 P_0 C_{in}.$$

* **Ex-3.5:**

(a) using the AND-OR-Invert implementation procedure described in section 3.7, show that the output carry in a full-adder circuit can be expressed as:

$C_{i+1} = G_i + P_i C_i = (G_i' P_i' + G_i C_i')$

Soln:

(04)

$$R.H.S = (G_i' P_i' + G_i' C_i')'$$

$$= (G_i' P_i')' (G_i' C_i')'$$

$$= (G_i + P_i)(G_i + C_i)$$

$$= G_i + G_i C_i + P_i G_i + P_i C_i$$

$$= G_i (1 + C_i) + P_i G_i + P_i C_i$$

$$= G_i (1 + P_i) + P_i C_i$$

$$= G_i + P_i C_i.$$

$$= A_i B_i + (A_i + B_i) C_i$$

$$= A_i B_i + A_i C_i + B_i C_i$$

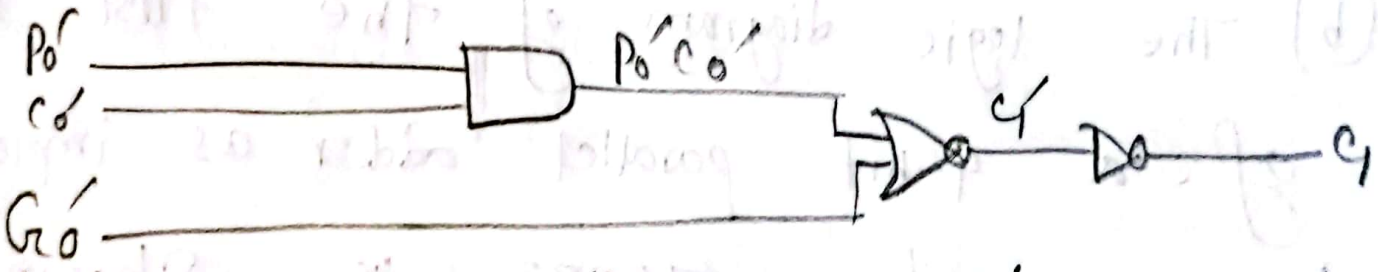
$$= C_{i+1}$$

Ahmad

(b) IC type 74182 is a look-ahead carry generator MSI circuit that generates the carries wit AND-OR-Invert gates. The MSI circuit assumes that the input terminals have the complements of the G's, the P's and C_1. Derive the Boolean functions for the look-ahead carries C_2, C_3 and C_4 in this IC.

(05)

Soln:



$C_1' = (G_0' + P_0'C_0')' = G_0(P_0'C_0)'' = G_0(P_0 + C_0)$

$C_1' = G_0P_0 + G_0P_0 + G_0C_0$

$C_4 = (G_0P_0 + G_0C_0)'$ $\quad C_4 = (G_3P_3 + G_3C_3)'$

$C_2 = (G_1P_1 + G_1C_1)'$

$C_2 = (G_2P_2 + P_2C_2)'$

* Ex-5.6: (a) Redefine the carry propagate and carry generate as follows: (06)

$P_i = A_i + B_i'$

$G_i = A_i B_i$

Show that the output carry and output sum of a full-adder becomes:

$C_{i+1} = (C_i' G_i' + P_i')'$

$= G_i + P_i C_i$

$= (P_i G_i') \oplus C_i$

(b) The logic diagram of the first stage of a 4-bit parallel adder as implemented in IC type 74283 is shown in fig. P5-6. Identify the P_i' and G_i' terminals as defined in (a) and show that the circuit implements a full-adder circuit.

(c) obtain the output carries C3 and C4 as a function of P_1', P_2', P_3', G_1', G_2', G_3' and C in AND-OR Invert form, and draw the two-level look-ahead circuit for this IC. (07)

Soln: Define,

$P_i = A_i + B_i$ and

$G_i = A_i B_i$.

Show that $C_{i+1} = G_i' (C_i' + P_i')'$ and

$S_i = (P_i G_i') \oplus C_i$

The output of a full adder is:

$S_i = A_i \oplus B_i \oplus C_i$

$C_{i+1} = A_i B_i + A_i C_i + B_i C_i$

$S_i = (P_i G_i') \oplus C_i$

$= [(A_i + B_i)(A_i B_i)'] \oplus C_i$

$= [A_i A_i' + A_i B_i' + B_i A_i' + B_i B_i'] \oplus C_i$

$= [A_i B_i' + A_i' B_i] \oplus C_i$

$= A_i \oplus B_i \oplus C_i$

$$C_{i+1} = (C_i'G_i' + P_i')'$$

$$= (C_i'G_i')'P_i$$

$$= (C_i + G_i)P_i$$

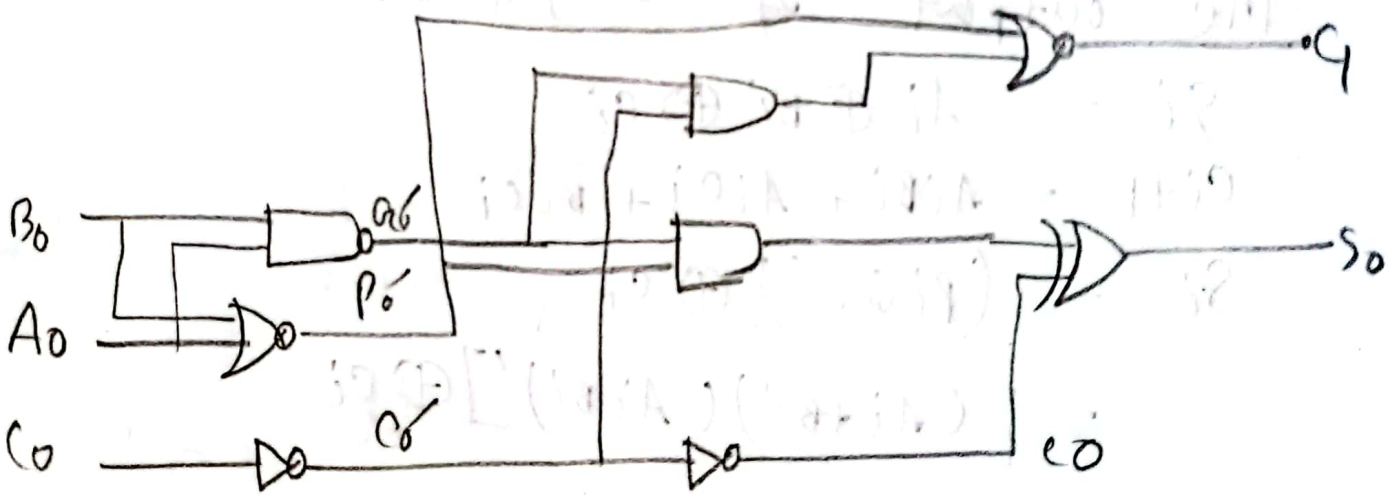
$$= (C_i + A_iB_i)(A_i + B_i)$$

$$= C_iA_i + C_iB_i + A_iB_iA_i + A_iB_iB_i$$

$$= A_iC_i + B_iC_i + A_iB_i + A_iB_i$$

$$= A_iC_i + B_iC_i + A_iB_i$$

Logic circuit:



Ex-5.7:

(a) Assume that the XOR gate has a propagation delay of 20ns and that

the AND or OR gates have a propagation delay of 10ns. what is the total propagation delay time in the 4-bit adder of fig. 5-5 ?

(09)

Soln: Total delay will be due to 3 levels as above :

⇒ P & G generator.

⇒ Carry generator.

⇒ Sum generator.

#P&G generator has a single level of Ex-OR & AND gates they will work in parallel delay1 = max(20, 10) = 20

#Carry generator has a two levels AND + OR gates they will work one after another delay2 = 10+10 = 20.

Sum generator has a single level of Ex-OR gates they will work all in parallel delay3 = 20 (10)

∴ Total delay = delay1 + delay2 + delay3

= 20 + 20 + 20

= 60ns.

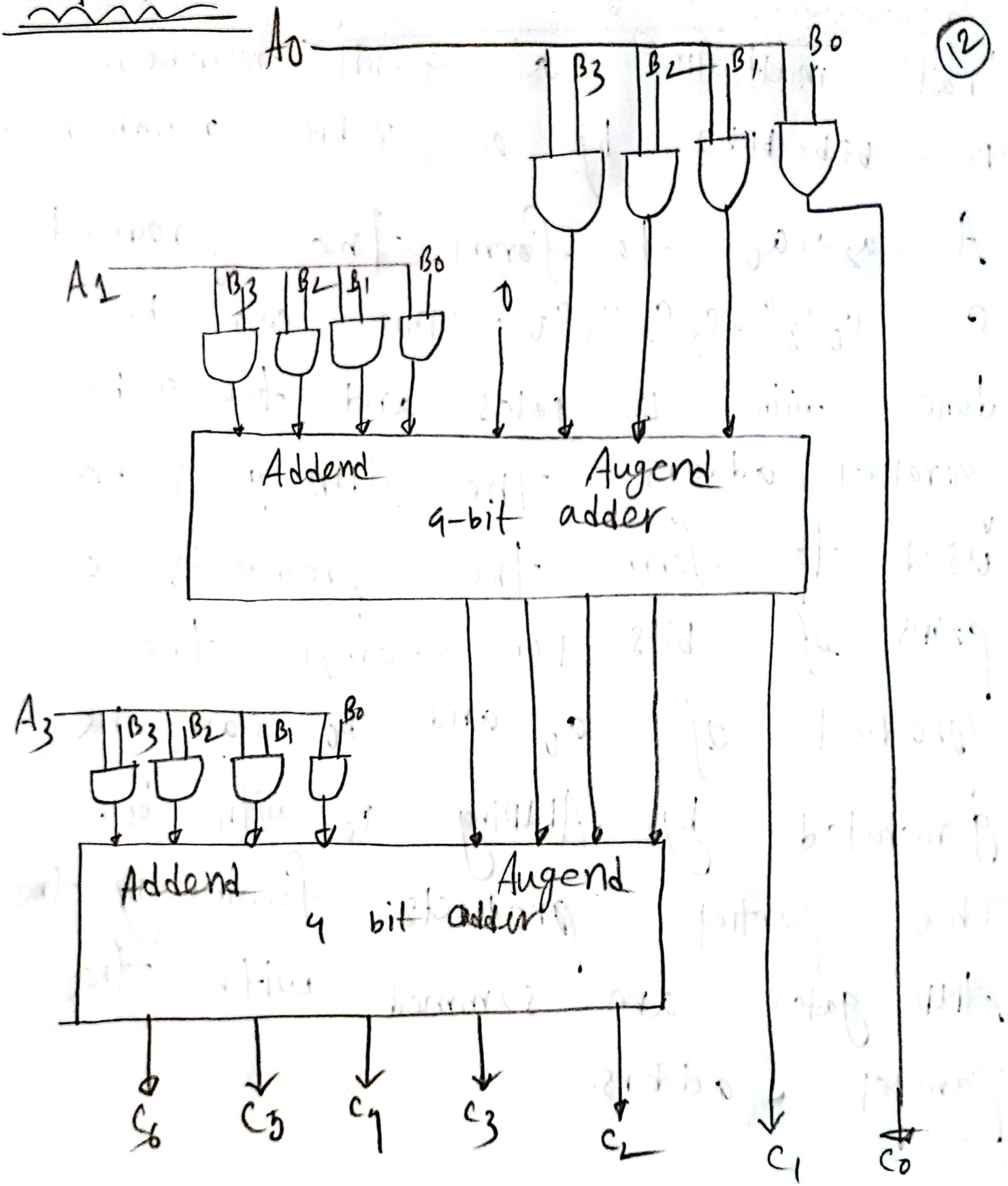
(b) Assume that C_5 is propagated in the box of fig. 5-5 at the same time as the other carries. what will be the propagation delay time of the 16-bit adder of problem 5-2?

Soln:

For 16 bit adder we have time delay = 120ns.

* Ex-5.8: Design a binary multiplier that multiplies a 4-bit number $B = b_3 b_2 b_1 b_0$ by a 3-bit number $A = a_2 a_1 a_0$ to form the product $C = c_6 c_5 c_4 c_3 c_2 c_1 c_0$. This can be done with 12 gates and two 4-bit parallel adders. The AND gates are used to form the products of pairs of bits. For example, the product of a_0 and b_0 can be generated by ANDing a_0 with b_0. The partial products formed by the AND gates are summed with the parallel adders.

Soln:


A0
B3
B2
B1
B0
A1
B3
B2
B1
B0
0
Addend
Augend
4-bit adder
A3
B3
B2
B1
B0
Addend
Augend
4 bit adder
C4
C3
C2
C1
C0

*Ex-5.9: How many don't-care inputs are there in a BCD adder? (13)

Soln: There are 16 x 16 possibilities without the carry. This is doubled when you add in the carry, so there are a total of 2 x16 x16 = 512 possible input permutations. out of this, only 2 x10 x10 = 200 are valid inputs — obviously.

Therefore, the answer is 512 - 200 = 312.

* Ex-5.10: Design a combinational circuit that generates the 9's complement of a BCD. Digit?

Soln: At first, we can draw a truth table for simplify.

BCD

9's complement

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	0	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
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

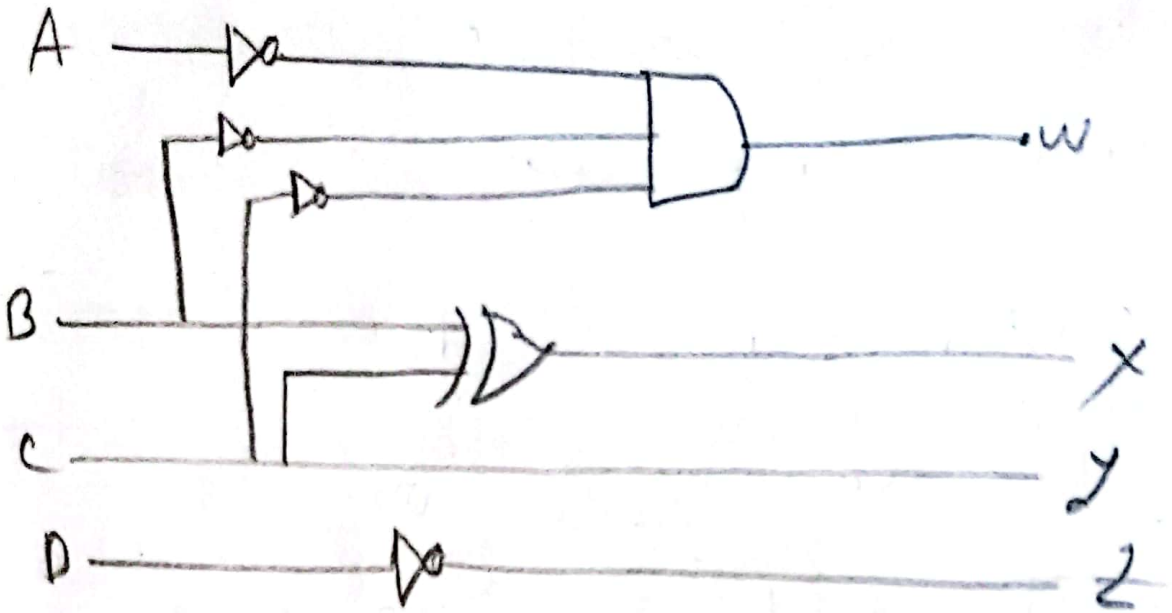
(M)

⇒ we can simplify this using k-map and find:

$w = A'B'C'$; $x = BC' + B'C = B \oplus C$; $y = C$

$z = D'$.

⇒ Circuit diagram:



*Ex-5.11: Design a decimal arithmetic unit with two selection variables, V_1 and V_0 and two BCD digits, A and B. The unit should have four arithmetic operations which depend on the value of the selection variables as shown below:

V_1	V_0	output function
0	0	A+9's complement of B
0	1	A+B
1	0	A+10's complement of B
1	1	A+1 (add 1 to A)

CamScanner

Soln:

⇒ In the first case, when $V_1 = 0$ and $V_0 = 0$, the output function in the sum of A and the 9's complement of B, which is equivalent to A-B in decimal arithmetic.

⇒ In the second case, when $V_1 = 0$ and $V_0 = 1$, the output function is the sum of A and B

⇒ In the third case, when $V_1 = 1$ and $V_0 = 0$, the output function is the sum of A and the 10's complement of B, which is equivalent to A-B-1 in decimal arithmetic.

⇒ In the fourth case, when $V_1 = 1$ and $V_0 = 1$, the output function is the sum of A and 1.

* Ex-5.12: It is necessary to design a decimal adder for two digits represented in the excess-3 code (Table-1.2) Show that the correction after adding the two digits with a 4-bit binary adder is as follows:

(a) The output carry is equal to the carry out of the binary adder.

Soln: Let, A and B be the two digits to be added. since the operands are represented in excess-3, the actual computations performed by the binary adder is $(A+3)+(B+3) = (A+B)+6$ instead of $A+B$.

If the carry is equal to 1, it means that $(A+B)+6 > 15$, or equivalently

$(A+B) \geq 9$. The required output carry is therefore the same as the carry from the binary adder. (14)

(b) if output carry = 1, add 0011

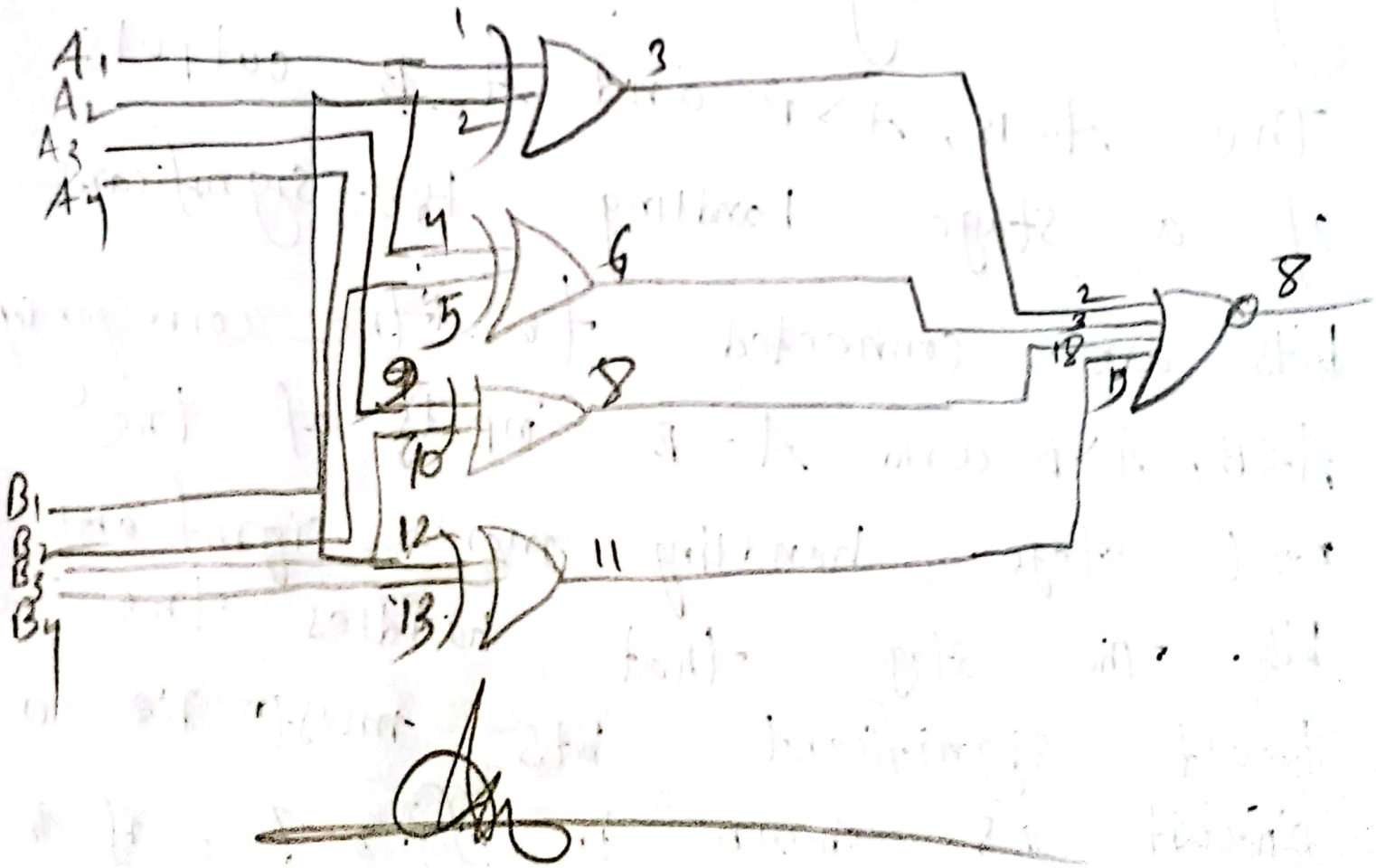
Soln: If the output carry is equal to 1, the 4-bit sum in binary is equal to $A+B+6-16 = A+B-10$ instead. Hence 0011 should be added to the 4-bit output.

(c) if output carry = 0, add 1101.

Soln: if the output carry is 0 the 4-bit sum in binary is equal to $A+B+6$, but the required output is $A+B+3$ instead. The correct sum can be produced by subtracting 3 from the output, or adding 1101

in binary, which is the 2's complement of 3, to the output.

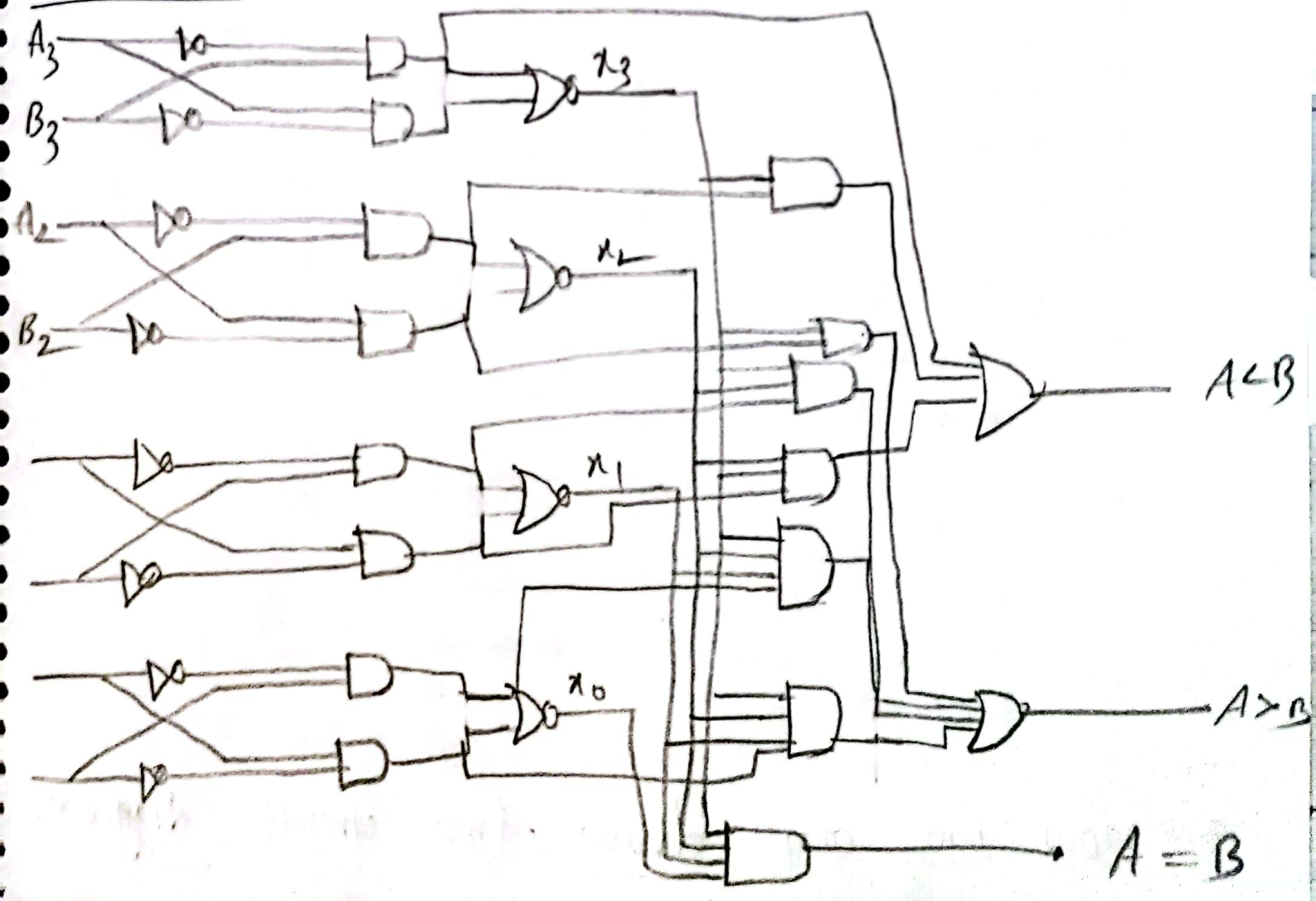
* Ex-5.13: Design a circuit that compares two 4-bit numbers, A and B, to check if they are equal. The circuit has one output x, so that x=1 if A=B and x=0 if A≠B.



* Ex-5.14: The 74L85 IC is a 4-bit magnitude comparator similar to that in fig-5.7, except that it has three more inputs and internal circuits that perform the equivalent logic as shown in Fig. P5-14. With these ICs, numbers of greater length may be compared by connecting comparators in cascade. The $A<B$, $A>B$ and $A=B$ outputs of a stage handling less-significant bits are connected to the corresponding $A<B$, $A>B$ and $A=B$ inputs of the next stage handling more significant bits. The stage that handles the least significant bits must be a circuit as shown in fig. 5-7. if the 74L85 IC is used, a 1 must be

applied to the $A=B$ input and a 0 to the $A<B$ and $A>B$ inputs in the IC that handles the four least (21) significant bits. Using one circuit as in fig. 5-7 and one 74L85 IC, obtain a circuit to compare two 8-bit numbers. Justify the circuit operation.

Soln:



★ E.X- 5.15: modify the BCD-to-Decimal decoder of Fig. 5-10 to give an output of all 0's when any invalid input combination occurs.

Soln: Truth table:

Input BCD	output (Decimal)
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	0
1011	0
1100	0
1101	0
1110	0
1111	0

⇒ now we can draw the circuit diagram.

* Ex-5.16: Design a BCD-to-excess-3 converter with a BCD-to-decimal decoder and four OR gates. (23)

Soln:

No	BCD 8 A	4 B	2 C	1 D	Excess-3 5 W	4 X	2 Y	1 Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

$\Rightarrow$ using K-map we find:

$W = \Sigma(5, 6, 7, 8, 9)$

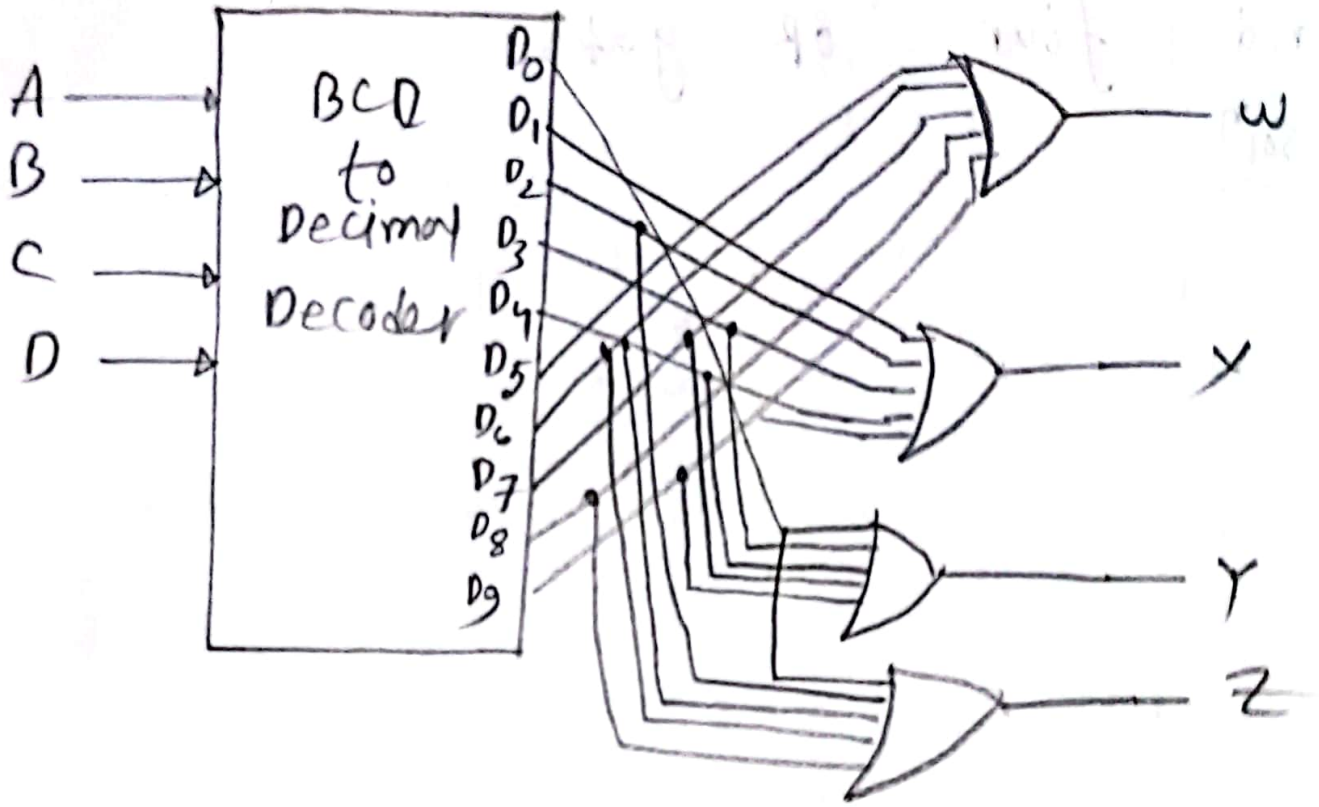
$X = \Sigma(1, 2, 3, 4, 9)$

$Y = \Sigma(0, 3, 4, 7, 9)$

$Z = \Sigma(0, 2, 4, 6, 8)$

CamScanner

⇒ circuit diagram:



* Ex-5.17: A combinational circuit is defined by the following three functions

$F1 = x'y' + xyz'$

$F2 = x' + y$

$F3 = xy + x'y'$

Design the circuit with a decoder and external gates.

Soln:

$\Rightarrow F_1 = x'y' + xyz'$

$= x'y'(z+z') + xyz'$

$= x'y'z + x'y'z' + xyz'$

$= m_1 + m_0 + m_6$

$F_1 = \Sigma(0, 1, 6)$

$\Rightarrow F_2 = x' + y$

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

$= x'y + x'y' + yz + yz'$

$= x'y(z+z') + x'y'(z+z') + yz(x+x') + yz'(x+x')$

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

$+ xy'z' + x'y'z'$

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

$F_2 = \Sigma(0, 1, 2, 3, 6, 7)$

$\Rightarrow F_3 = xy + x'y'$

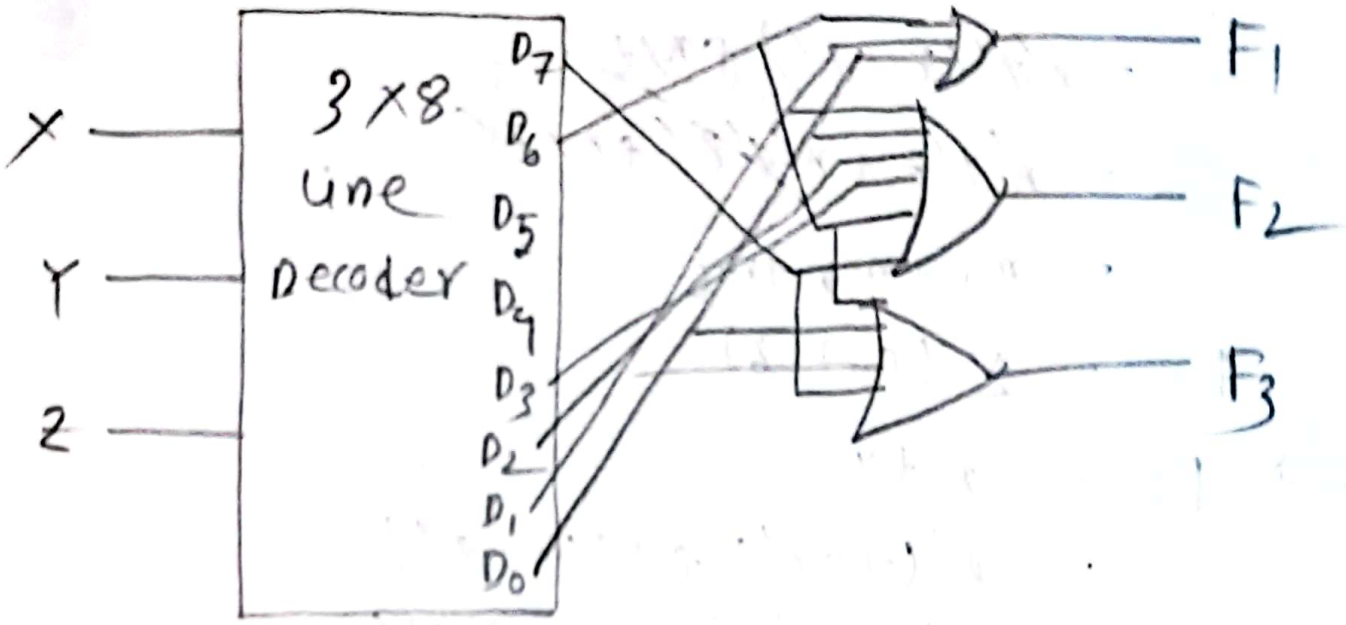
$= xy(z+z') + x'y'(z+z')$

$= xyz + xyz' + x'y'z + x'y'z'$

$= m_7 + m_6 + m_1 + m_0$

$F_3 = \Sigma(0, 1, 6, 7)$

circuit diagram:



* E.X – 5.18: A combinational circuit is defined by the following two functions

$F_1(x,y) = \Sigma(0,3)$

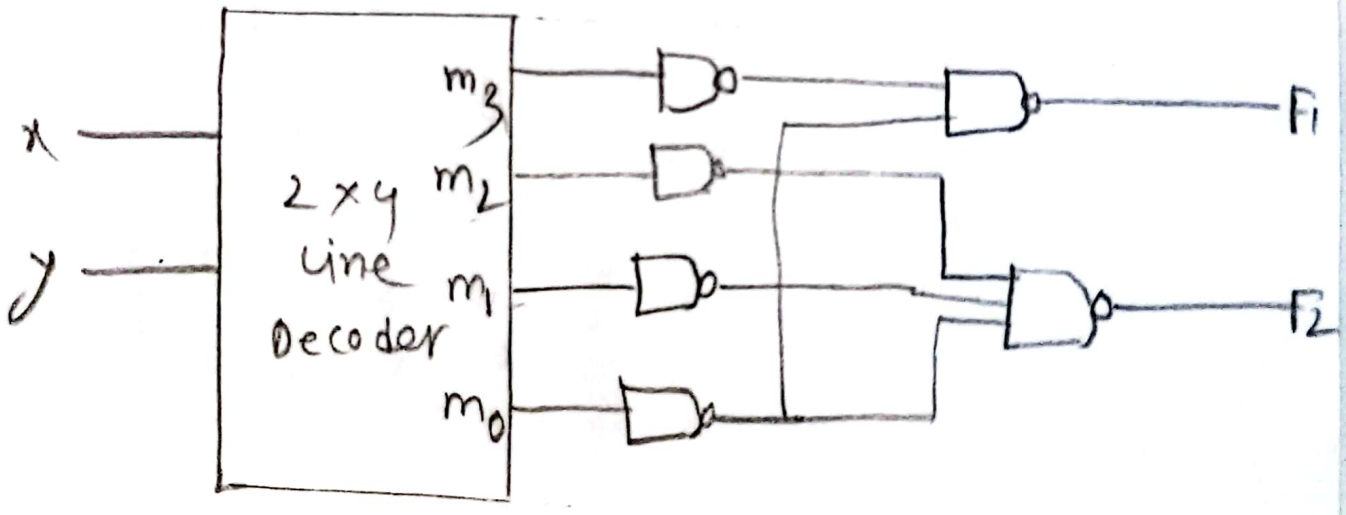
$F_2(x,y) = \Sigma(1,2,3)$

Implement the combinational circuit by means of the decoder shown in fig. 5-12 and external NAND gates.

Soln:

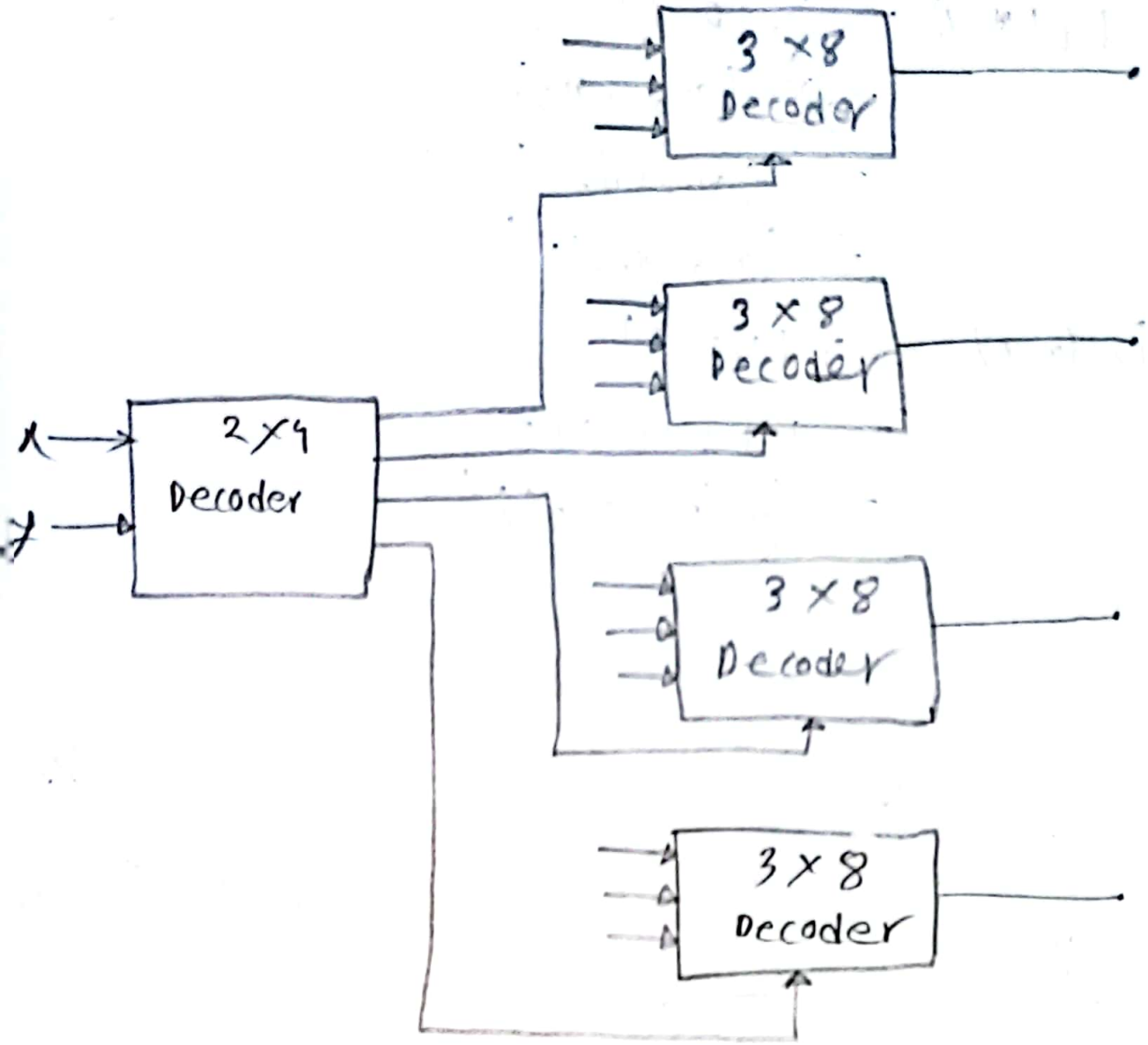
$$\Rightarrow F_1(x,y) = \Sigma(0,3)$$
$$= \Sigma(m_0 + m_3)$$
$$= (m_0 + m_3)'$$
$$= (m_0' + m_3')'$$

$$\Rightarrow F_2(x,y) = \Sigma(1,2,3)$$
$$= m_1 + m_2 + m_3$$
$$= (m_1' m_2' m_3')'$$



* Ex- 5.19: Construct a 5 x 32 decoder with four 3 x 8 decoder / demultiplexers and a 2 x 4 decoder. use a block diagram construction as in fig - 5.14.

Soln:



* Ex-5.20: Draw the logic diagram of a 2-line-to-4-line decoder/demultiplexer using NOR gates only.

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

Here,

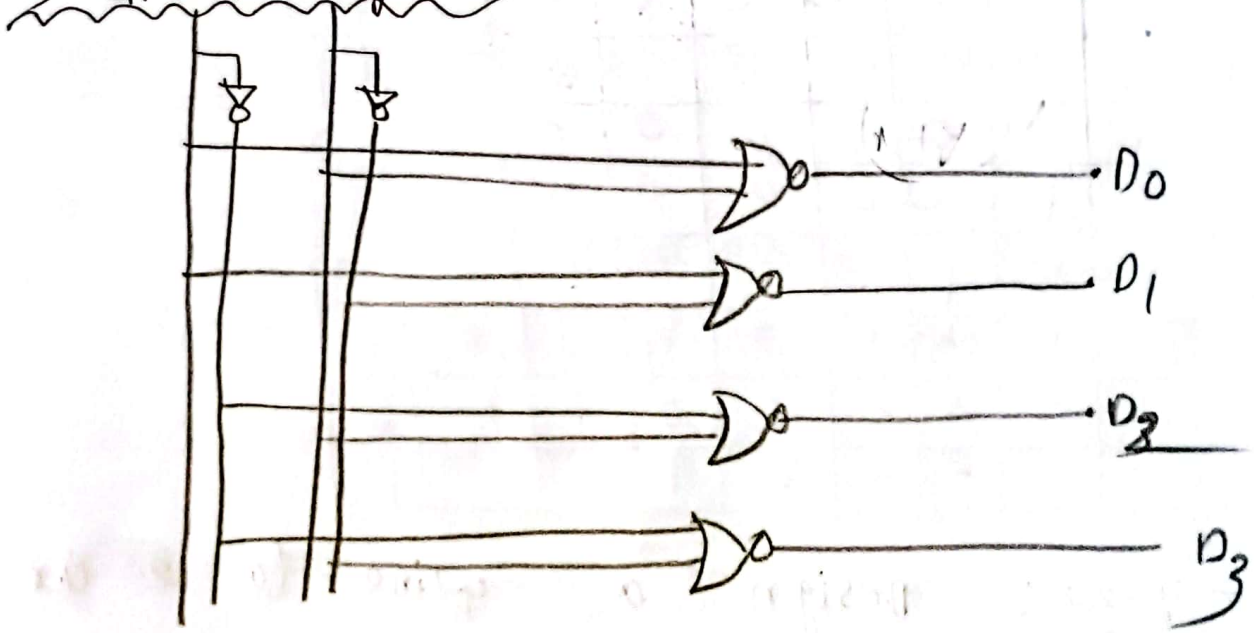
$D_0 = x'y' = (x'y')'' = (x+y)'$

$D_1 = x'y = (x'y')' = (x+y')'$

$D_2 = xy' = (x+y)'$

$D_3 = xy = (x'+y')'$

⇒ Circuit Diagram:



* Ex - 5.21: Specify the truth table of an octal-to-Binary priority encoder. provide an output to indicate if at least one of the inputs is a 1. The table can be listed with 9 rows and some of the inputs will have don't care values.

Inputs								outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	Y_2	Y_1	Y_0
1	0	0	0	0	0	0	0	0	0	0
x	1	0	0	0	0	0	0	0	0	1
x	x	1	0	0	0	0	0	0	1	0
x	x	x	1	0	0	0	0	0	1	1
x	x	x	x	1	0	0	0	1	0	0
x	x	x	x	x	1	0	0	1	0	1
x	x	x	x	x	x	1	0	1	1	0
x	x	x	x	x	x	x	1	1	1	1

* Ex - 5.22: Design a 4-line to 2-line priority encoder. Include an output E to indicate that at least one input is a 1.

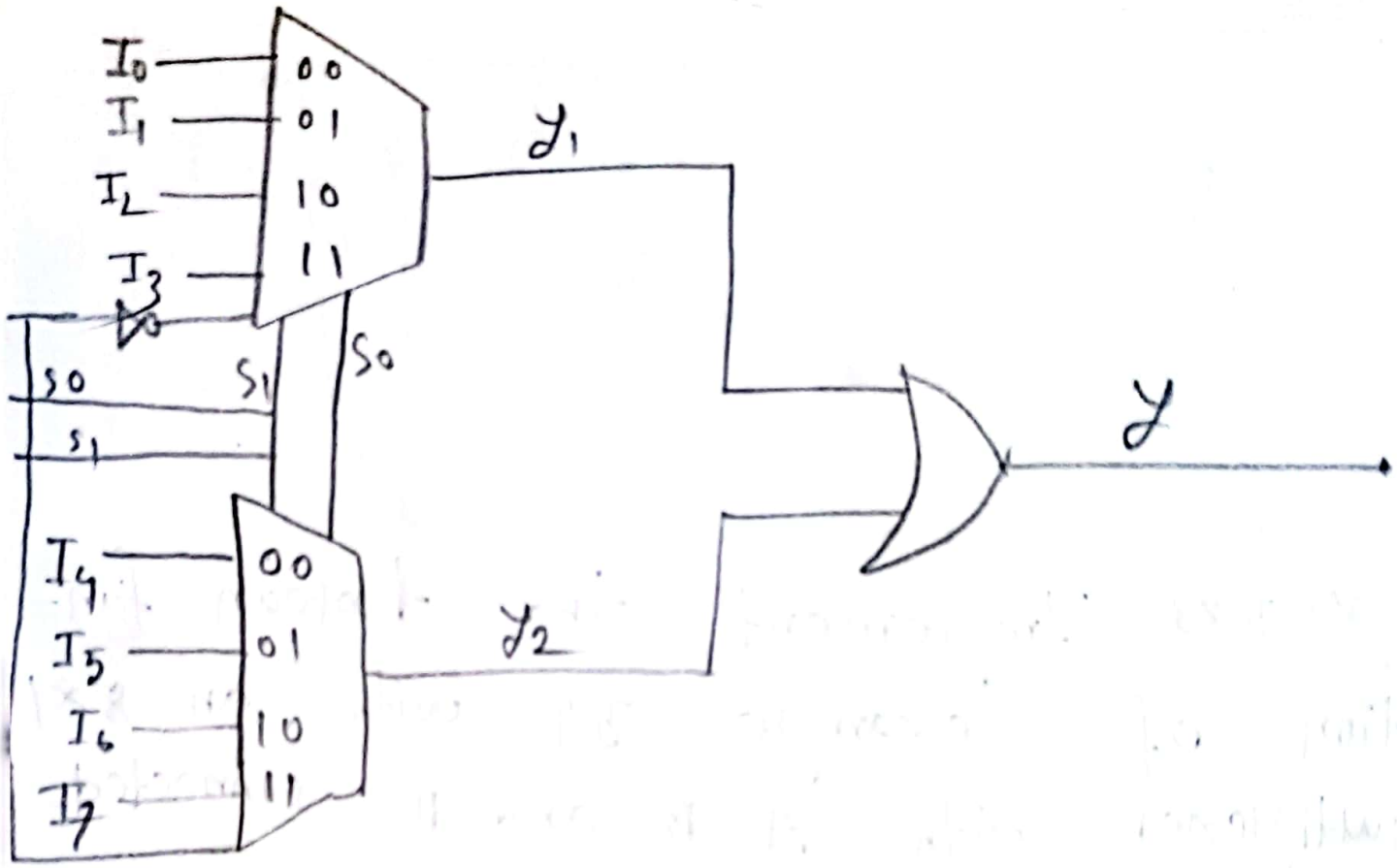
Soln:

Inputs				outputs		
D_0	D_1	D_2	D_3	Y_1	Y_0	E
0	0	0	0	x	x	0
1	0	0	0	0	0	1
0	1	0	0	0	1	1
0	0	1	0	1	0	1
0	0	0	1	1	1	1

* E.X-5.23: Implement the boolean function of example 5.4 with an 8×1 multiplexer with A, B and D connected to selection lines S_2, S_1, S_0 respectively.

Soln: I_0 through $I_7 = C', 1, C', 0, C', C', 0, C$

* E.X-5.24: Implement the combinational circuit specified in problem 5.17 with a dual 4-line to 1-line multiplexers, an OR gate, and Inverter.



E	S_1	S_0	Y
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

CamScanner

* Ex-5.25: The 32×6 ROM together with the 2^0 line as shown in fig. P5-25 converts a 6-bit binary number to its corresponding 2-digit BCD number. For example, binary 100001 converts to BCD 01|0001 (decimal-33). Specify the truth table for the ROM.

Soln:

I_5	I_4	I_3	I_2	I_1	2^0	O_6	O_5	O_4	O_3	O_2	O_1	O_0	decimal
0	0	0	0	0	0	0	0	0	0	0	0	0	00
0	0	0	0	0	1	0	0	0	0	0	0	1	01
0	0	0	0	1	0	0	0	0	0	0	1	0	02
0	0	0	0	1	1	0	0	0	0	0	1	1	03
0	0	0	1	0	0	0	0	0	0	1	0	0	04
0	0	0	1	0	1	0	0	0	0	1	0	1	05
0	0	0	1	1	0	0	0	0	0	1	1	0	06
0	0	0	1	1	1	0	0	0	0	1	1	1	07
0	0	1	0	0	0	0	0	0	1	0	0	0	08
0	0	1	0	0	1	0	0	0	1	0	0	1	09
0	0	1	0	1	0	0	0	1	0	0	0	0	10
0	0	1	0	1	1	0	0	1	0	0	0	1	11
0	0	1	1	0	0	0	0	1	0	0	1	1	12
0	0	1	1	0	1	0	0	1	0	0	1	1	13

up to 63. (3M)

* Ex-5.26: prove that a 32x8 ROM can be used to implement a circuit that generates the binary square of an input 5-bit number with B0 = A0 and B1 = 0 as in Fig. 5.24(a). Draw a block diagram of the circuit and list the first four and the last four entries of the ROM truth table.

Soln: The truth table for the ROM would be as follows:

A_4	A_3	A_2	A_1	A_0	B_1	B_0	output
0	0	0	0	0	0	0	000000
0	0	0	0	1	0	1	000001
0	0	0	1	0	0	0	000100
0	0	0	1	1	0	1	000101
--	--	--	--	--			
1	1	1	0	1	0	1	111101
1	1	1	1	0	0	0	111000
1	1	1	1	1	0	1	111111

The first four and the last four entries of the ROM truth table are shown above.

* Ex-5.26: What size ROM would it take to implement:

(a) A BCD adder/subtractor with a control input to select between the addition and subtraction.

Soln: To implement a BCD adder/subtractor with a control input to select between addition and subtraction,

we can use a 4-bit BCD Adder and a 4-bit BCD subtractor, and a 2-to-1 multiplexer to select between their outputs based on the control input. each of the adder, and subtractor circuits can be implemented using a 16 x 4 ROM. Therefore the total size of the ROM required would be: (36)

$2 \times 16 \times 4 + 1 \times 2\text{-to-}1 = 64 + 2 = 66$.

So, a 66 x 4 ROM would be needed to implement a BCD adder/subtractor with a control input to select between addition and subtraction.

(b) A binary multiplier that multiplies two 4-bit numbers:

Soln: To implement a binary multiplier that multiplies two 4-bit numbers, we would need a ROM with a size of 4x4 = 16 rows and 8 columns. The input bits of the two numbers would be connected to the address lines of the ROM and the outputs bits of the product would be read from the data lines of the ROM. (37)
Therefore, a 16 x 8 ROM would be needed to implement a binary multiplier that multiplies two 4-bit numbers.

(c) Dual 4-line-to-1-line with common selection inputs.

Soln: To implement dual 4-line to 1-line multiplexers with common selection inputs, we would need two 4x2 ROM's and a 2-to-1 multiplexer to select between their outputs. Each 4x2 ROM can be implemented using a 16x1 ROM and the 2-to-1 multiplexer can be implemented using a 2x1 ROM. Therefore the total size of ROM required would be:

2 x 16x1 ROM + 1x2x1 ROM + 1 x 2-to-1 multiplexer = 32+2+2 = 36.

* Ex-5.28: Each output inverts in the PLA of fig. 5.26 is replaced by an XOR gate. Each XOR gate has two inputs. One output is connected

to the output of the OR gate, and the other input is connected through links to a signal equivalent to either '0' or '1'. Show how to select the true/complement output in this configuration. (32)

Soln: The input signals are connected to the XOR gate inputs and the output of the XOR gate is connected to the mux input. The select input of the mux is connected to the signal that corresponds to logic 1, and the other input of the mux is connected to the complement output of the XOR gate. Finally the output of the mux is connected

to the output buffer, which can invert or pass through the signal as required.

* Ex-5.29: Derive the PLA program table for a combinational circuit that squares a 3-bit number. Minimize the number of product terms. (10)

Soln:

⇒ Truth table:

S.N	input XYZ	outputs D_5	D_4	D_3	D_2	D_1	D_0
0	000	0	0	0	0	0	0
1	001	0	0	0	0	0	1
2	010	0	0	0	1	0	0
3	011	0	0	1	0	0	0
4	100	0	1	0	0	0	0
5	101	0	1	0	0	0	1
6	110	1	0	0	1	0	0
7	111	1	1	0	0	0	1

CamScanner

⇒ Derive the logic equation from above truth table:

$D_0 = \Sigma(1,3,5,7)$

$D_1 = 0$

$D_2 = \Sigma(2,6)$

$D_3 = \Sigma(3)$

$D_4 = \Sigma(4,5,7)$

$D_5 = \Sigma(6,7)$

(41)

⇒ After minimizing, we find:

$D_0 = z$, $D_1 = 0$.

$D_2 = x'yz' + xyz' = yz'$

$D_3 = x'yz + xy'z$

$D_4 = xy'z' + x'yz + xyz'$

$D_5 = xyz + xyz'$

* Ex-5.30: List the PLA program table for the BCD-to-excess-3 code converter defined in section 4.5.

Soln:

Here,

$W = A + BC + BD$

$W' = A'B' + A'C'D'$

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

$x' = B'C'D' + BC + BD$

$y = CD + C'D'$

$y' = C'D + CD'$

$z = D'$

$z' = D$.

	product	ABCD	outputs F_1, F_2, F_3, F_4
A	1	1 - - -	1 - - -
BC	2	- 1 1 -	1 1 - -
BD	3	- 1 - 1	1 1 - -
B'C'D'	4	- 0 0 0	- 1 - -
CD	5	- - 1 1	- - 1 -
C'D'	6	- - 0 0	- - 1 -
D'	7	- - - 0	- - - 1

* Ex-5.31: Design a combinational circuit using a ROM which accepts a 2-bit number and generate a binary number equal to the cube of the input binary number.

Soln:

⇒ Truth table:

Input	output
0 0	0 0 0
0 1	0 0 1
1 0	1 0 0
1 1	1 1 1

⇒ now, we can draw the circuit diagram.

* Ex-5.32: Using 3×8 decoder and two OR gates design a full subtractor.

⇒ Truth table:

Inputs			outputs	
A	B	Bin	Bout	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

⇒ Logic diagram: