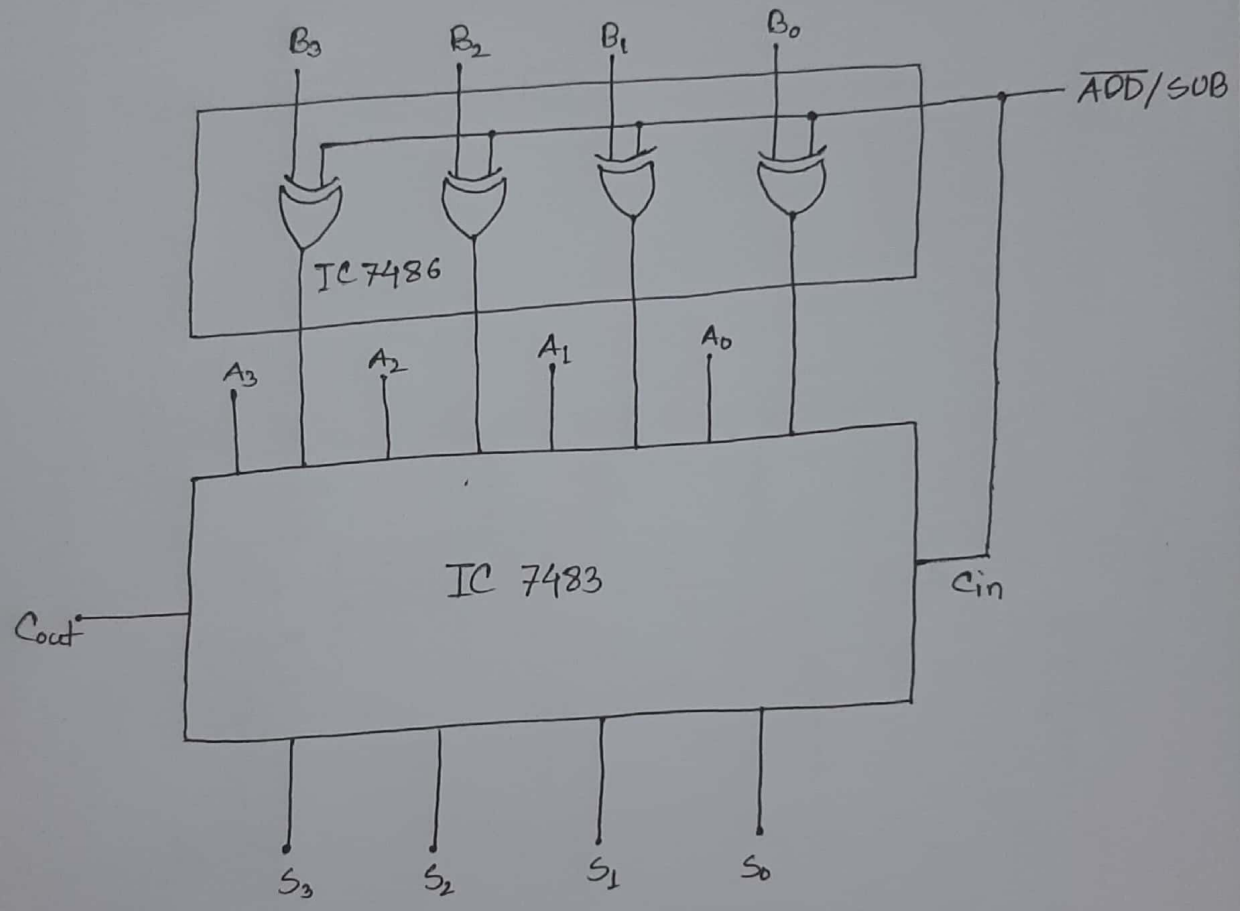
5.1: Design an excess-3-to BCD code converter using a 4-bit full adders MSI circuit.

Answer: The design of an excess-3-to BCD code converter using a 4-bit full-adders MSI circuit is given below:



Here X-OR gates are used as controlled inverters and will be used for 1's complement of B.

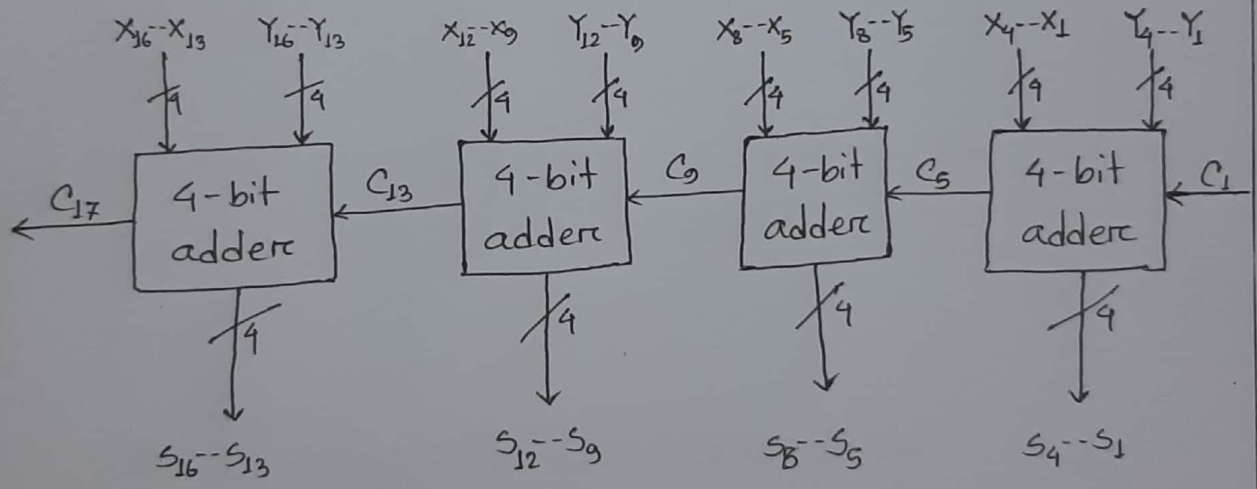
Here, $A_3A_2A_1A_0 - (3)_{10}$ = $A_3A_2A_1A_0$ + 1's complement of 3_{10} + 1

So, $B_3B_2B_1B_0 = (3)_{10} = (0011)_2$

And $C_{in} = 1$.

5.2: Using four MSI circuits, construct a binary parallel adder to add two 16-bit binary numbers. Label all the carries between the MSI circuits.

Answer: Larger parallel adders can be bew built from smaller ones. To add two 16-bit binary numbers we ne ed a 16-bit parallel binary adder. It can be constructed from 4-bit parallel adders:



5.3: Using 4 exclusive-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.

Answer:

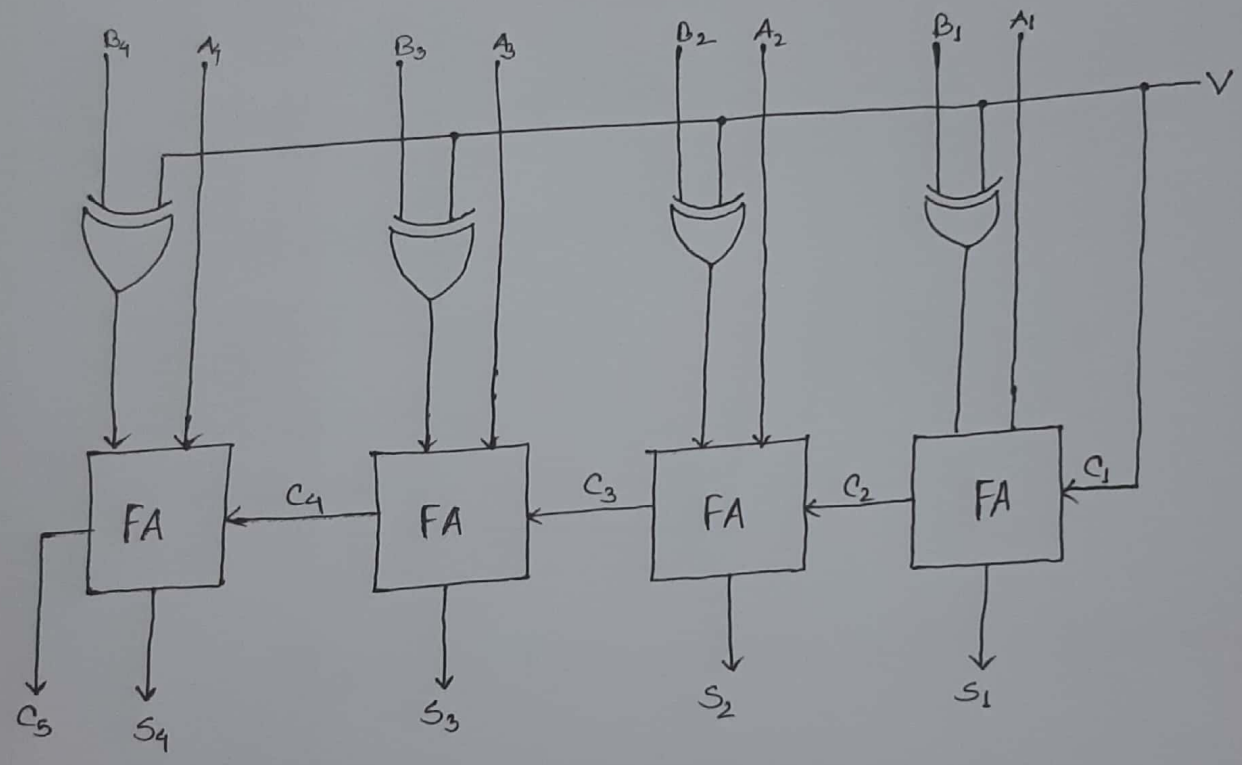


fig: 4-bit adder-subtractor

The adder-subtractor circuit of the figure has the following values for mode input V and data inputs A and B.

5.4. Derive the two-level equation for the output carry C_5 shown in the look-ahead carry generator of fig. 5-5.

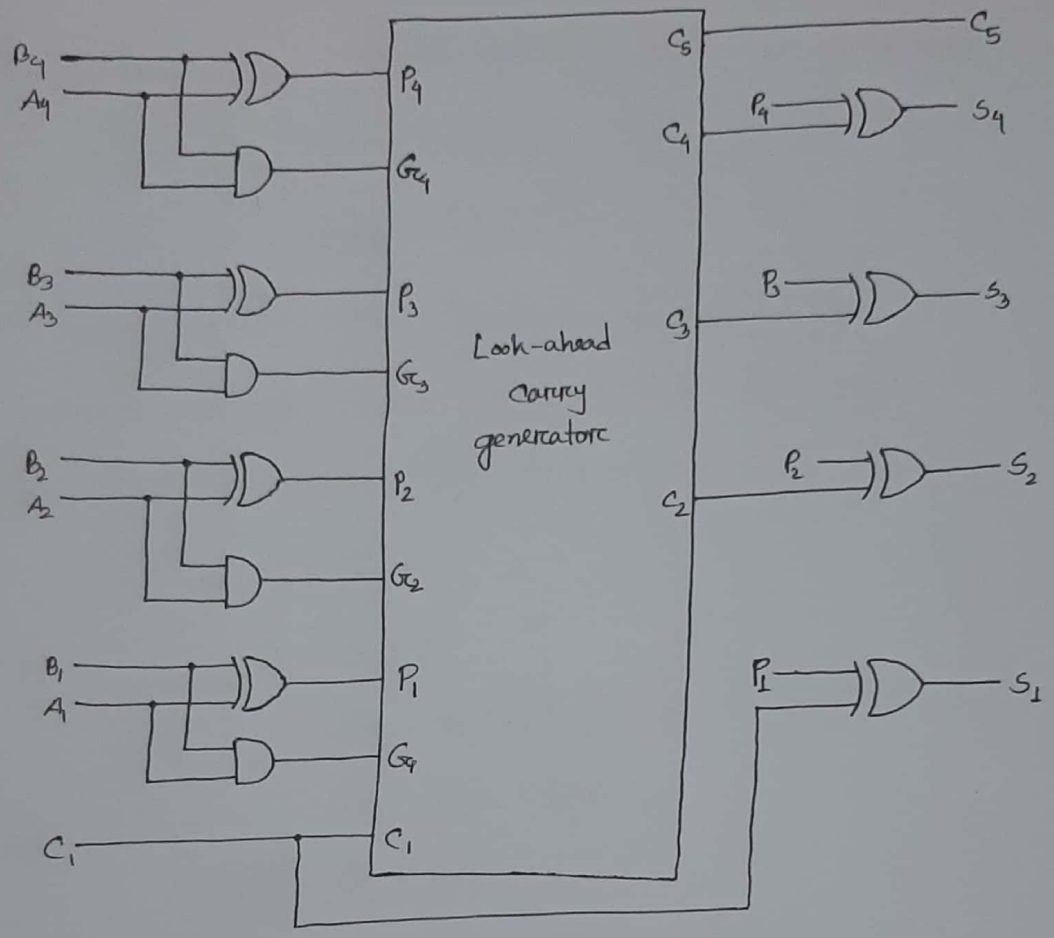


Figure 5.5 : 4-bit full adders with look ahead carry

Answer: Considering the circuit of the full-adder-

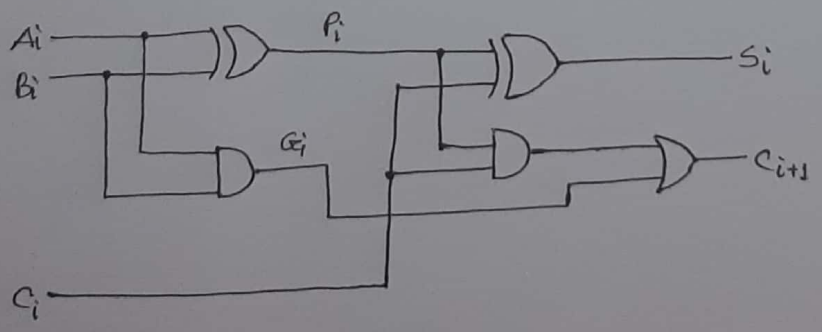


Figure - Full adder circuit

If we define two new binary variables -

$$P_i = A_i \oplus B_i$$

$$G_i = A_i B_i$$

the output sum and carry can be expressed as:

$$S_i = P_i \oplus C_i$$

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

We now write the Boolean function for the carry output of each stage and substitute for each C_i its value from the previous equations:

$$C_2 = G_1 + P_1 C_1$$

$$C_3 = G_2 + P_2 C_2$$

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

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

$$C_4 = G_3 + P_3 C_3$$

$$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_1$$

$$C_5 = G_4 + P_4 C_4$$

$$= G_4 + P_4(G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_1)$$

$$= G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1 + P_4 P_3 P_2 P_1 C_1$$

5.6: a) Redefine the carry propagate and carry generate as follows:

$$P_i = A_i + B_i$$

$$G_i = A_i B_i$$

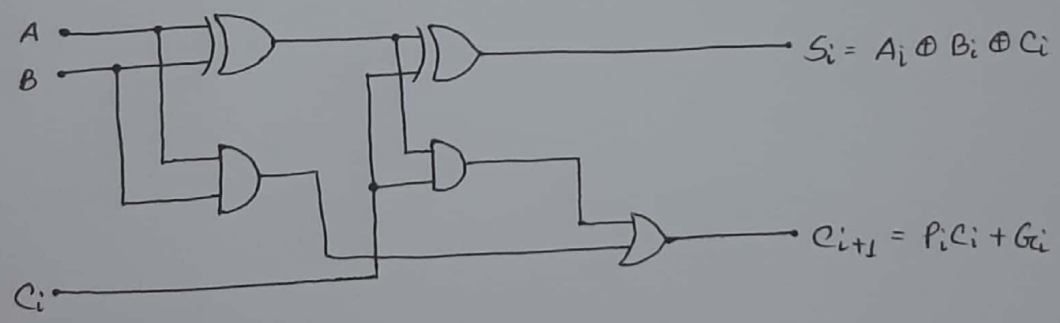
Show that the output carry and output sum of a full adder becomes:

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

$$= G_i + P_i C_i$$

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

Answer: Full-adder with P & G is shown below:



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

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

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

$$= C_i P_i + P_i G_i$$

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

$$= C_i P_i + A_i B_i$$

$$= C_i P_i + G_i = C_{i+1} = L \cdot H \cdot S$$

Again we have to prove, $S_i = (P_i G_i') \oplus C_i$

$R \cdot H \cdot S = P_i G_i' \oplus C_i$

$= (A_i + B_i) \cdot (A_i B_i)' \oplus C_i$

$= \{(A_i + B_i)(A_i' + B_i')\} \oplus C_i$

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

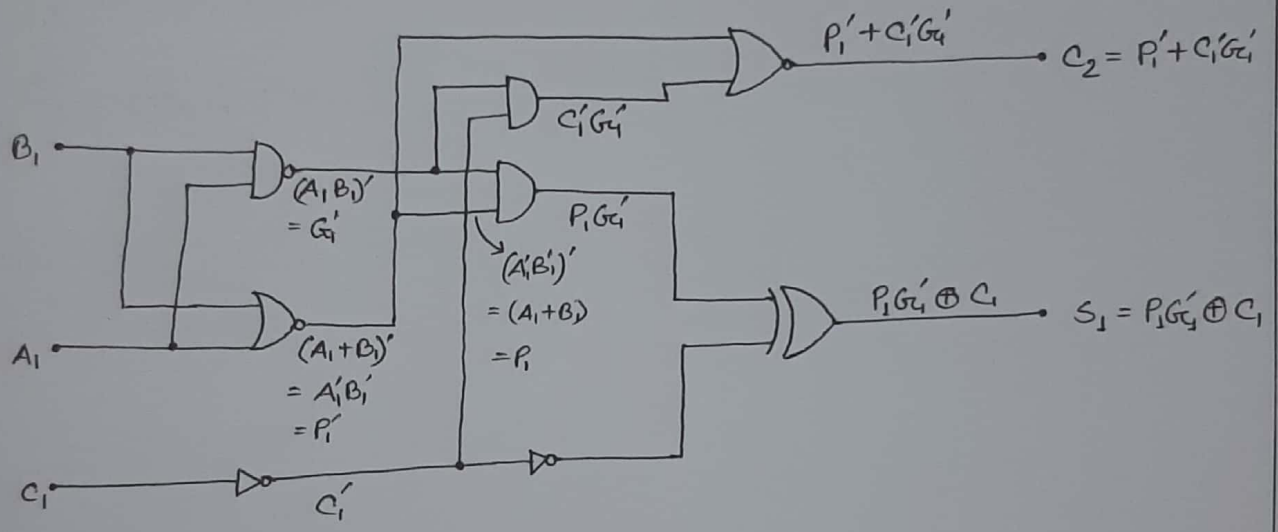
$= A_i \oplus B_i \oplus C_i$

$= S_i$

$= L \cdot H \cdot S.$

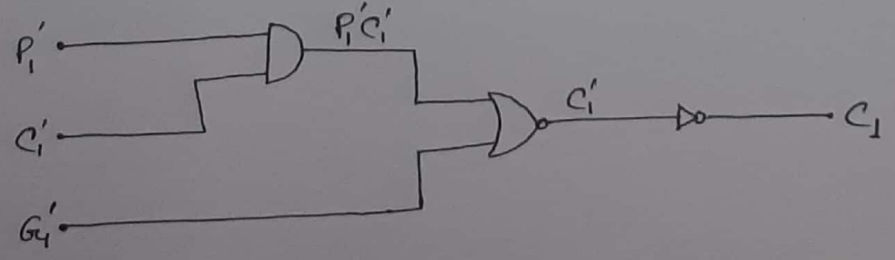
b) The logic diagram of the first stage of a 4-bit parallel adders as implemented in IC type 74283 is shown in figure P5-6. Identify the P_1' and G_1' terminals as defined in (a) show that the circuit implements a full-adder circuit.

Answer:



c) Obtain the output carries C_3 and C_4 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.

Answer:



$$C_1' = (G_{c_1}' + P_1' C_1')'$$

$$\Rightarrow C_1' = G_{c_1} \cdot (P_1 + C_1)$$

$$\Rightarrow C_1' = G_{c_1} P_1 + G_{c_1} C_1$$

$$\Rightarrow C_1 = (G_{c_1} P_1 + G_{c_1} C_1)'$$

$$C_2 = (G_{c_2} P_2 + G_{c_2} C_2)'$$

$$C_3 = (G_{c_3} P_3 + G_{c_3} C_3)'$$

$$C_4 = (G_{c_4} P_4 + G_{c_4} C_4)'$$

5.7 : (a) Assume that the exclusive-OR gate has a propagation delay of 20 ns and that the AND or OR gates have a propagation delay of 10 ns. What is the total propagation delay time in the 4-bit adder of figure 5-5?

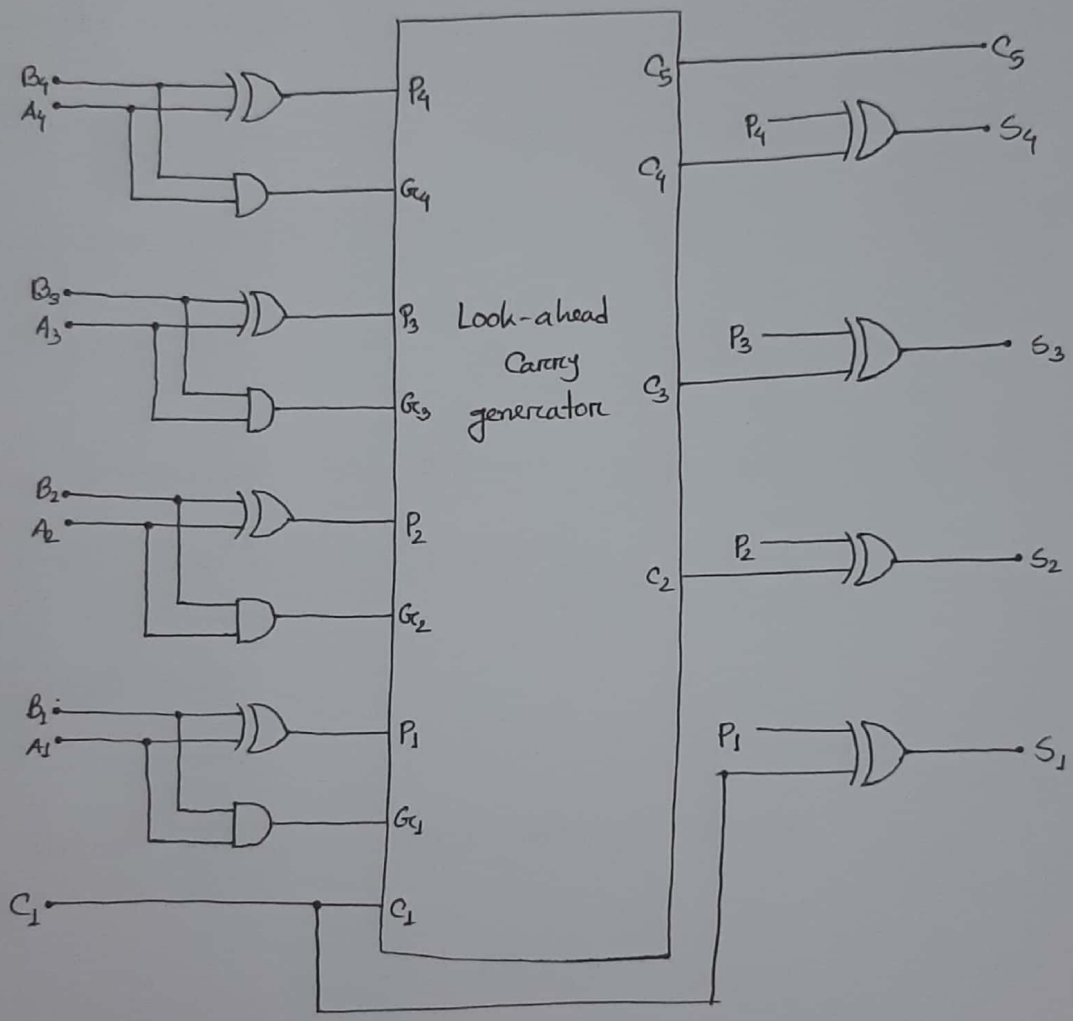


Fig. 5-5 : 4-bit-full-adders with look-ahead carry

Answer: Here total delay will be due to three levels. They are -

i. P and Gc generators

ii. Carry generators

iii. Sum generators

P and Gc generators has a single level of EX-OR and AND gates, they will work in parallel

delay1 = max (20, 10) = 20

Carry generators has two levels AND + OR gates. They will work one after another

delay2 = 10 + 10 = 20

Sum generators has a ~~sign~~ single level of EX-OR gates. They will work all in parallel.

delay3 = 20

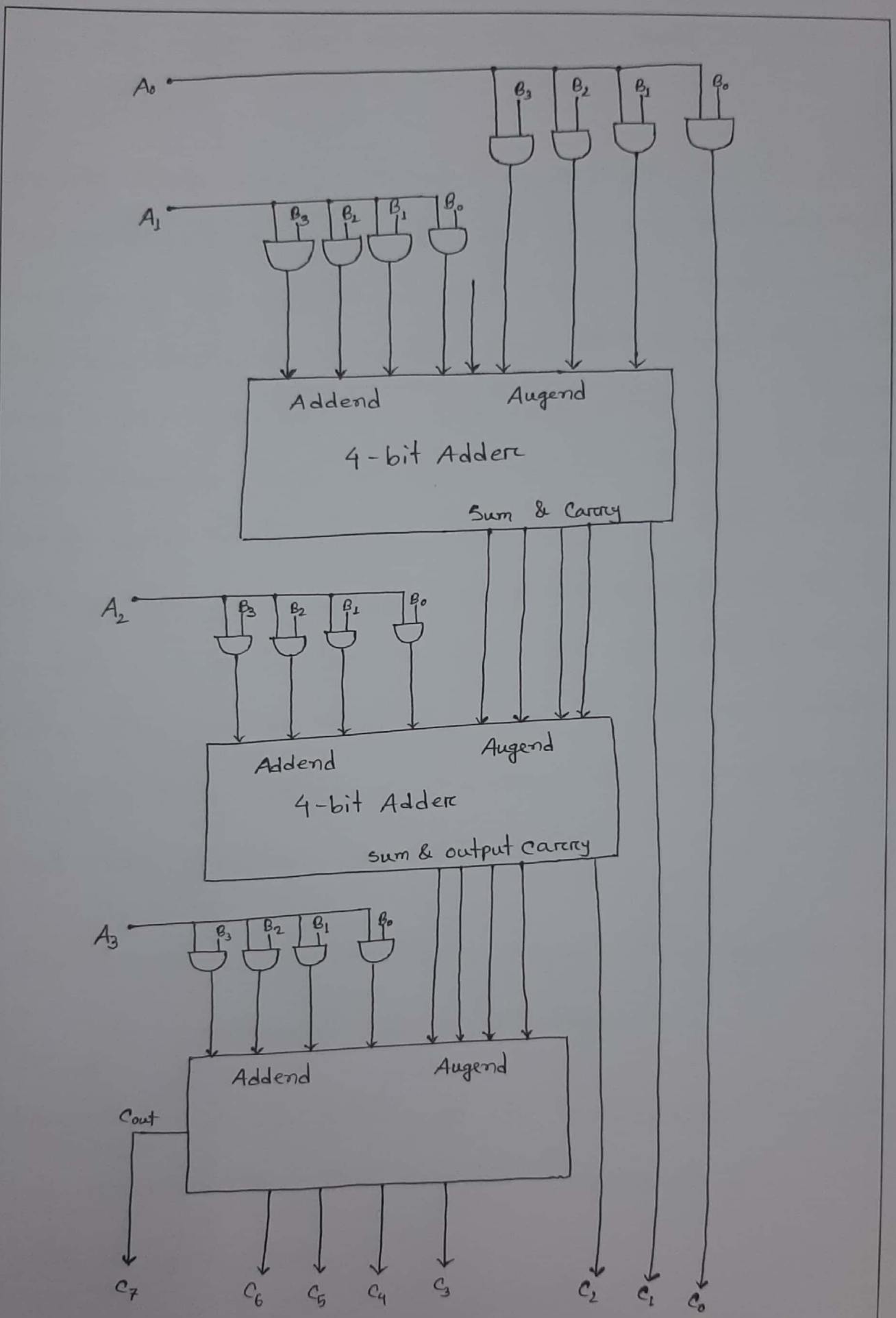
∴ Total delay = delay1 + delay2 + delay3

= 20 + 20 + 20

= 60 ns

5.8: Design a binary multiplier that multiplies a 4-bit number $B = b_3b_2b_1b_0$ by a 3-bit number $A = a_2a_1a_1$ to form the product $C = c_6c_5c_4c_3c_2c_1c_0$. 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.

Answer: The binary multiplier that multiplies a 4-bit number $B = b_3b_2b_1b_0$ by a 3-bit number $A = a_2a_1a_0$ to form the product $C = c_6c_5c_4c_3c_2c_1c_0$ is designed here:



5.9: How many don't care inputs are there in a BCD adder?

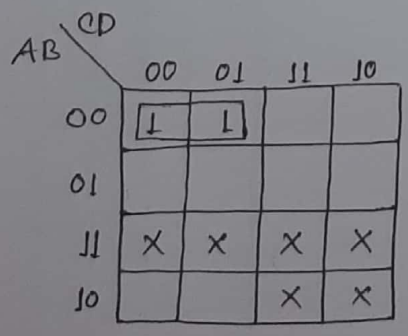
Answer: There are in total 512 combinations with a bcd adder (8 inputs for two bcd numbers and one input considering the carry in and therefore 2^9 combinations). But bcd includes numbers from 0000 through 1001 (0 to 9) and hence we have 10 valid combinations for bcd numbers. Therefore since we have two bcd numbers there are 10*10 valid combinations. Taking carry in into consideration there are 10*10*2 valid combinations (carry in is either 0 or 1). That is there are 200 valid combinations. This implies that there are 512-200 = 312 invalid or don't care conditions. Therefore the total number of don't care conditions are 312.

5.10: Design a combinational circuit that generate the 9's complement of a BCD digit.

Answer: The truth table of the combinational circuit that generate the 9's ~~complet~~ complement of a BCD digit is given below:

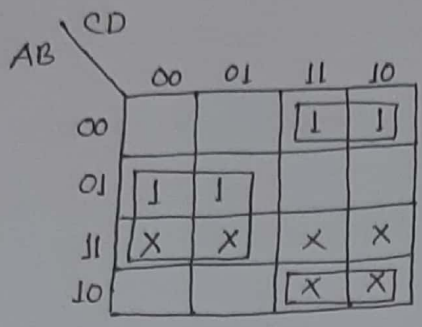
BCD				9's Complement			
A	B	C	D	w	x	y	z
0	0	0	0	1	0	0	1
0	0	0	1	1	0	0	0
0	0	1	0	0	1	1	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	0	0
0	1	1	0	0	0	1	1
0	1	1	1	0	0	1	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

K-map for 'w':



$W = A'B'C'$

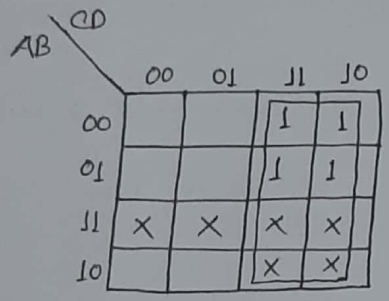
K-map for 'x':



$X = BC' + B'C$

$= B \oplus C$

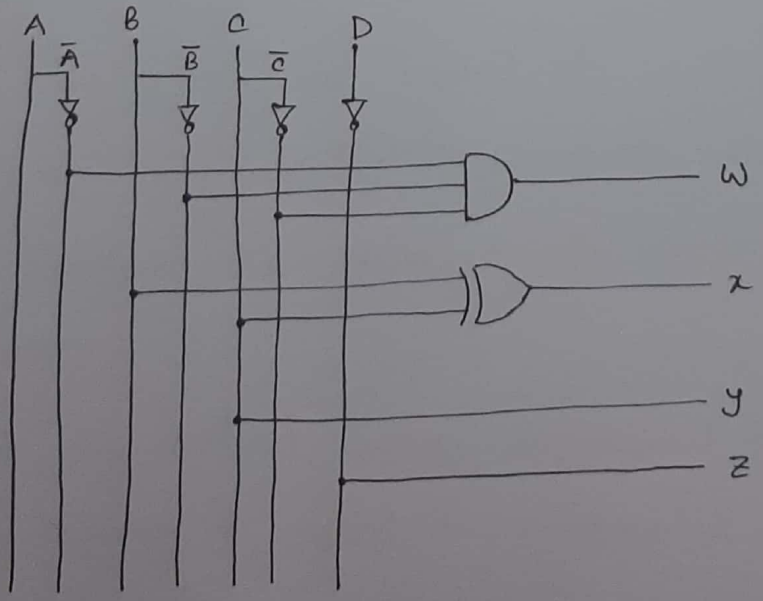
K-map for 'y':



$Y = C$

$Z = D'$

Combinational Circuit:



5.12: It is necessary to design a decimal adder for two digits d represented in the excess-3 code. Show that the correction after checking 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

b) If output carry = 1, add 0011.

c) If output carry = 0, add 1101

construct the adder with two 4-bit binary adders and an ~~int~~ inverter.

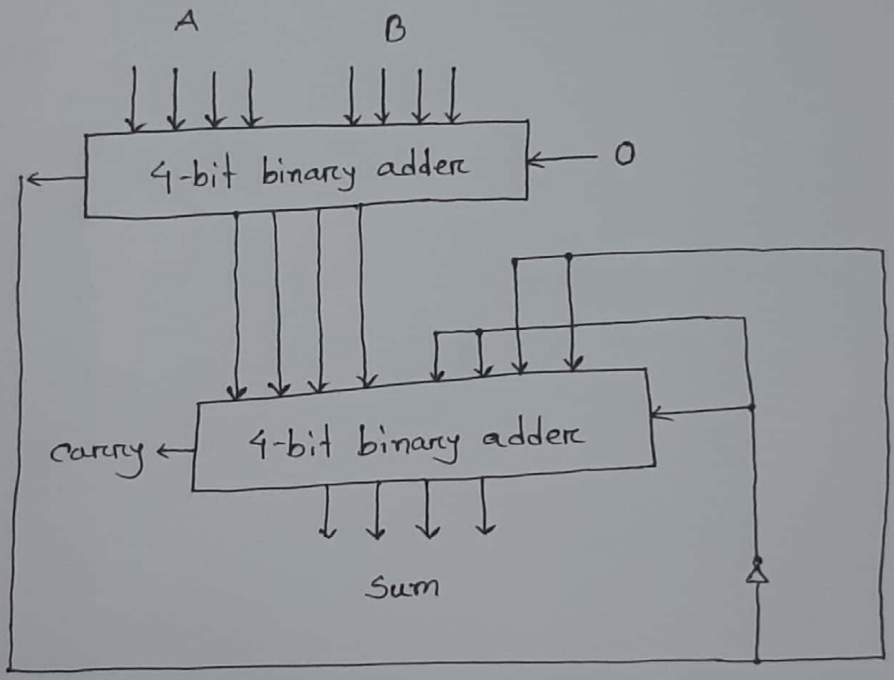
Answer: Let A and B be the two digits to be added. Since the operands are represented in excess-3, the actual computation performed by the binary adder is $(A+3)+(B+3) = (A+B)+6$ instead of $A+B$.

a) If the carry in equal to 1, it means that $(A+B)+6 > 15$, or, equivalently $(A+B) > 9$. The required output carry is therefore the same as the carry from the binary adder.

b) If the output carry is equal to 1, the 4-bit sum in binary is equal to $A+B+6-16 = A+B-10$, but the required output is $A+B+3-10$ instead. Hence 0011 (3 in

binary) should be added to the 4-bit output.

c) 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.



5.13: Design a circuit that 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 $A \neq B$.

Answer: Let the two 4-bit numbers are:

$$A = A_3 A_2 A_1 A_0$$

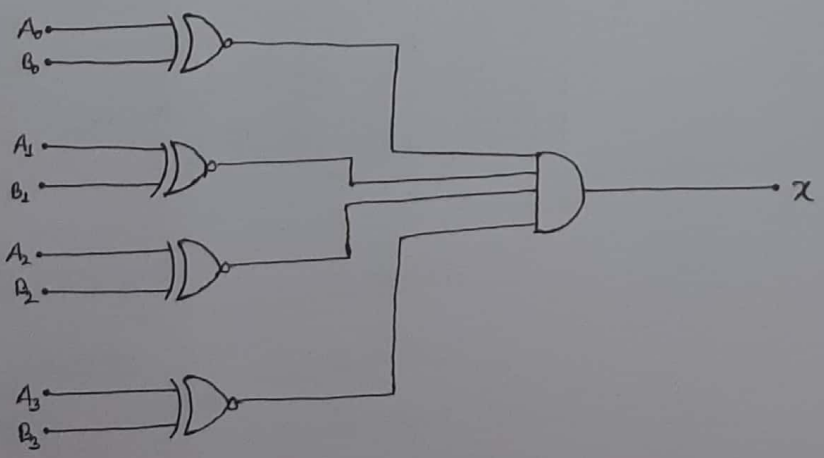
$$B = B_3 B_2 B_1 B_0$$

Truth table:

A	B	x
0	0	1
0	1	0
1	0	0
1	1	1

} X-NOR

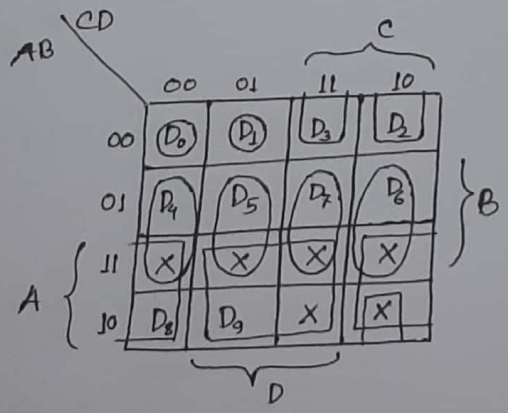
Circuit:



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.

Answer: Truth table-

A	B	C	D	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9
0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1



$D_0 = A'B'C'D'$

$D_1 = A'B'C'D$

$D_2 = B'CD'$

$D_3 = B'CD$

$D_4 = BC'D'$

$D_5 = BC'D$

$D_6 = BCD'$

$D_7 = BCD$

$D_8 = AD'$

$D_9 = AD$

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

$$F_1 = x'y' + xyz'$$
$$F_2 = x' + y$$
$$F_3 = xy + x'y'$$

Design the circuit with a adder decoder and external gates.

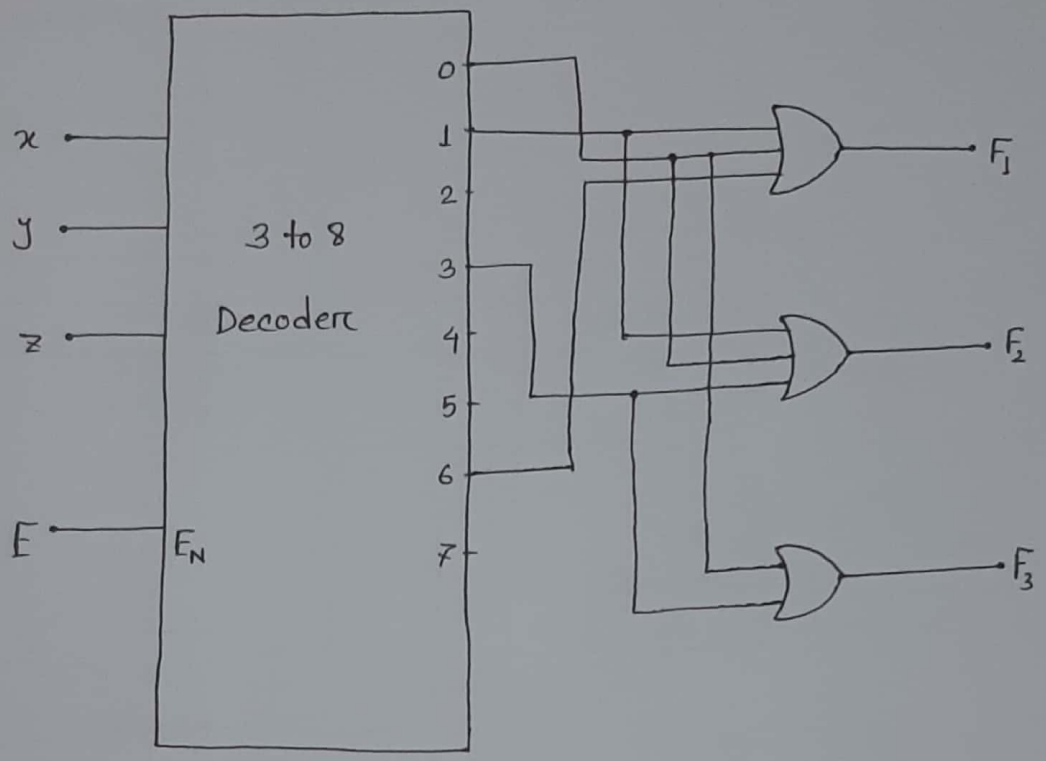
Answer: The given functions are

$$F_1 = x'y' + xyz'$$
$$= x'(z+z')y' + xyz'$$
$$= x'y'z + x'y'z' + xyz'$$
$$= \Sigma(1, 0, 6)$$

$$F_2 = x'y$$
$$= x'(y+y') + y(x+x')$$
$$= x'y + x'y' + xy + x'y$$
$$= x'y + x'y' + xy$$
$$= \Sigma(1, 0, 3)$$

$$F_3 = xy + x'y'$$
$$= \Sigma(3, 0)$$

Circuit:



5.19: Construct a 5 × 32 decoder with four 3×8 decoders demultiplexers and a 2×4 decoder. Use a block diagram construction.

Answer:

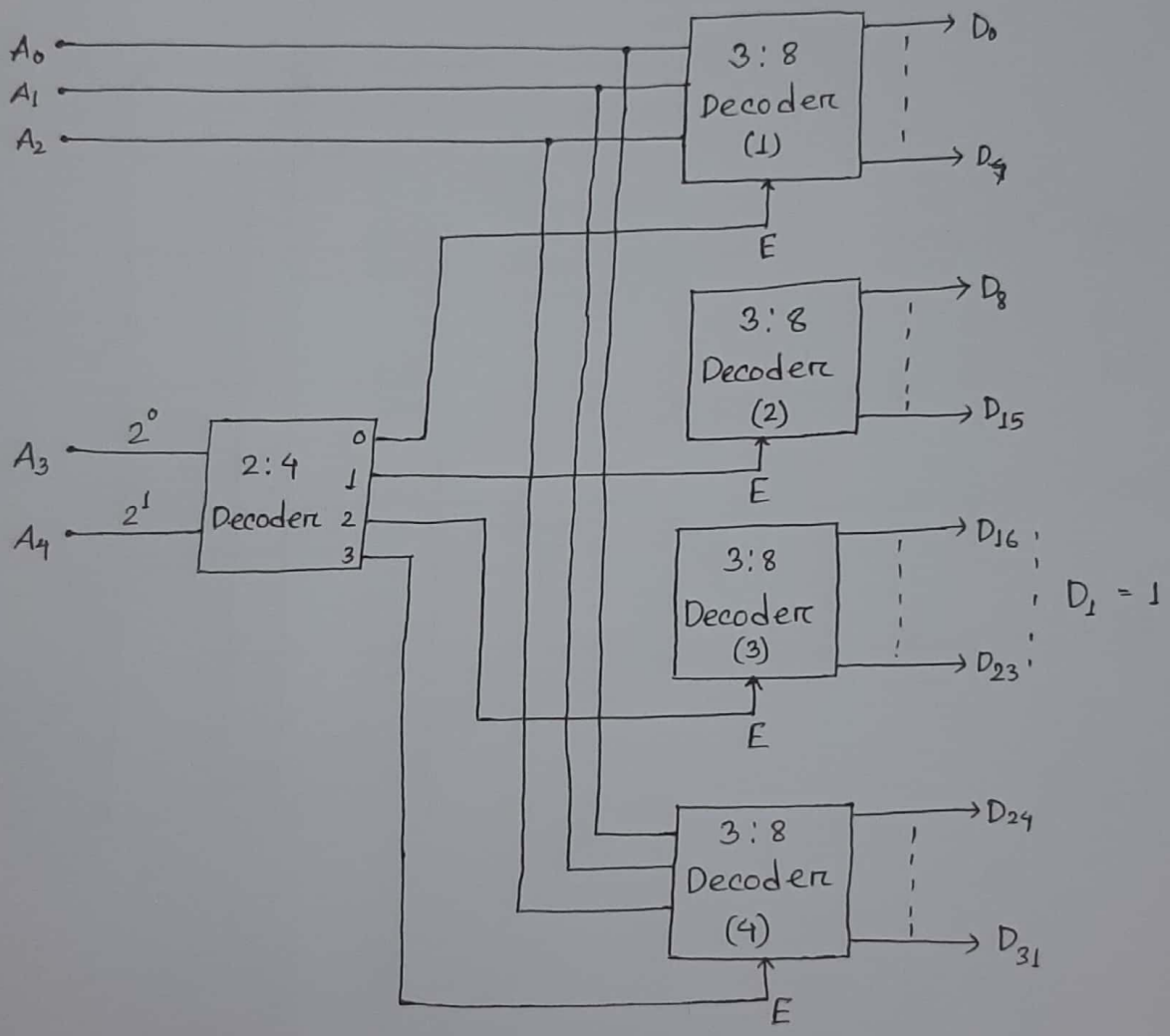
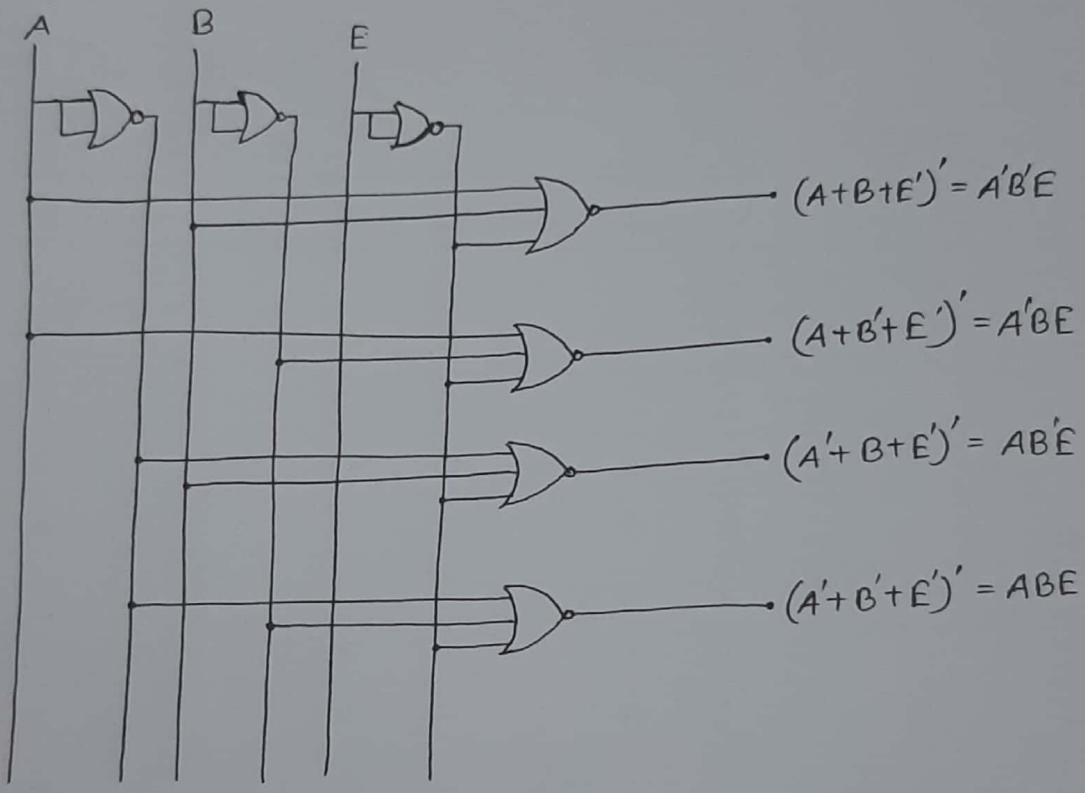


Figure: A 5×32 decoder with four 3×8 decoder and a 2×4 decoder.

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

Answer: The logic diagram of a 2-line to 4-line decoder using NOR gates only in given below:



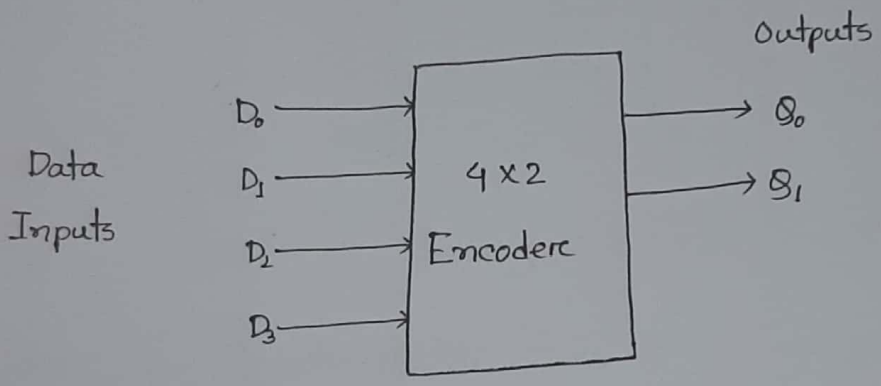
5.21: Specify the truth table of an octal-to-binary priority encoder. Provide an output to indicate if at least one of the inputs is a 1. The table can be listed with 9 rows, and some of the inputs will have don't care values.

Answer: The truth table is given below-

Inputs								Outputs			
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z	V
0	0	0	0	0	0	0	0	x	x	x	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

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.

Answer: Priority encoders take all of their data inputs one at a time and converts them to into an equivalent binary code as its output.



Inputs				Outputs	
D_3	D_2	D_1	D_0	Q_1	Q_2
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
0	0	0	0	X	X

5.25: The 32 x 6 ROM together with the 2^0 line as shown in Fig. P5-25 converts to BCD 011 0011 (decimal 33). Specify the truth table for ROM.

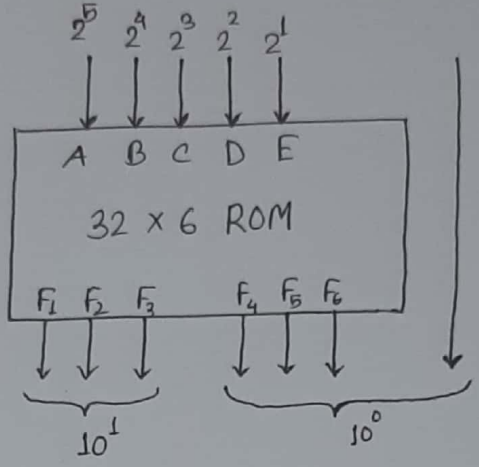


Fig. P5-25: Binary-to-decimal Converters.

Answer: The truth table for ROM in given below:

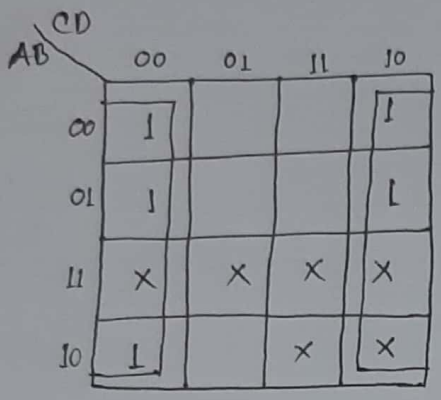
ROM Inputs					2^0	Outputs of ROM						$D_0 = 2^0$	Decimal
A	B	C	D	E		F_6	F_5	F_4	F_3	F_2	F_1		
0	0	0	0	0	0	0	0	0	0	0	0	0	00
0	0	0	0	0	1	0	0	0	0	0	0	1	01
0	0	0	0	1	0	0	0	0	0	0	1	0	02
0	0	0	0	1	1	0	0	0	0	0	1	1	03
0	0	0	1	0	0	0	0	0	0	1	0	0	04
0	0	0	1	0	1	0	0	0	0	1	0	1	05
0	0	0	1	1	0	0	0	0	0	1	1	0	06
0	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	0	12
0	0	1	1	0	1	0	0	1	0	0	1	1	13

ROM Inputs					2^0	Outputs of ROM					$D_0 = 2^0$	Decimal
A	B	C	D	E		F_6	F_5	F_4	F_3	F_2 F_1		
0	0	1	1	1	0	0	0	1	0	1 0	0	14
0	0	1	1	1	1	0	0	1	0	1 0	1	15
0	1	0	0	0	0	0	0	1	0	1 1	0	16
0	1	0	0	0	1	0	0	1	0	1 1	1	17
0	1	0	0	1	0	0	0	1	1	0 0	0	18
0	1	0	0	1	1	0	0	1	1	0 0	1	19
0	1	0	1	0	0	0	1	0	0	0 0	0	20
0	1	0	1	0	1	0	1	0	0	0 0	1	21
0	1	0	1	1	0	0	1	0	0	0 1	0	22
0	1	0	1	1	1	0	1	0	0	0 1	1	23
0	1	1	0	0	0	0	1	0	0	1 0	0	24
0	1	1	0	0	1	0	1	0	0	1 0	1	25
0	1	1	0	1	0	0	1	0	0	1 1	0	26
0	1	1	0	1	1	0	1	0	0	1 1	1	27
0	1	1	1	0	0	0	1	0	1	0 0	0	28
0	1	1	1	0	1	0	1	0	1	0 0	1	29
0	1	1	1	1	0	0	1	1	0	0 0	0	30
0	1	1	1	1	1	0	1	1	0	0 0	1	31
1	0	0	0	0	0	0	1	1	0	0 1	0	32
1	0	0	0	0	1	0	1	1	0	0 1	1	33
1	0	0	0	1	0	0	1	1	0	1 0	0	34
1	0	0	0	1	1	0	1	1	0	1 0	1	35
1	0	0	1	0	0	0	1	1	0	1 1	0	36
1	0	0	1	0	1	0	1	1	0	1 1	1	37
1	0	0	1	1	0	0	1	1	1	0 0	0	38
1	0	0	1	1	1	0	1	1	1	0 0	1	39
1	0	1	0	0	0	1	0	0	0	0 0	0	40
1	0	1	0	0	1	1	0	0	0	0 0	1	41
1	0	1	0	1	0	1	0	0	0	0 1	0	42
1	0	1	0	1	1	1	0	0	0	0 1	1	43
1	0	1	1	0	0	1	0	0	0	1 0	0	44
1	0	1	1	0	1	1	0	0	0	1 0	1	45
1	0	1	1	1	0	1	0	0	0	1 1	0	46

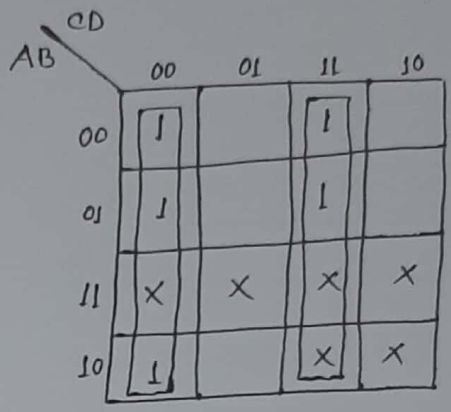
ROM Inputs					2^0	Outputs of ROM						$D_0 = 2^0$	Decimal
A	B	C	D	E		F_6	F_5	F_4	F_3	F_2	F_1		
1	0	1	1	1	1	1	0	0	0	1	1	1	47
1	1	0	0	0	0	1	0	0	1	0	0	0	48
1	1	0	0	0	1	1	0	0	1	0	0	1	49
1	1	0	0	1	0	1	0	1	0	0	0	0	50
1	1	0	0	1	1	1	0	1	0	0	0	1	51
1	1	0	1	0	0	1	0	1	0	0	1	0	52
1	1	0	1	0	1	1	0	1	0	0	1	1	53
1	1	0	1	1	0	1	0	1	0	1	0	0	54
1	1	0	1	1	1	1	0	1	0	1	0	1	55
1	1	1	0	0	0	1	0	1	0	1	1	0	56
1	1	1	0	0	1	1	0	1	0	1	1	1	57
1	1	1	0	1	0	1	0	1	1	0	0	0	58
1	1	1	0	1	1	1	0	1	1	0	0	1	59
1	1	1	1	0	0	1	1	0	0	0	0	0	60
1	1	1	1	0	1	1	1	0	0	0	0	1	61
1	1	1	1	1	0	1	1	0	0	0	1	0	62
1	1	1	1	1	1	1	1	0	0	0	1	1	63

5.30: List the PLA program table for the BCD-to-excess-3 code converter defined in Section 4-5.

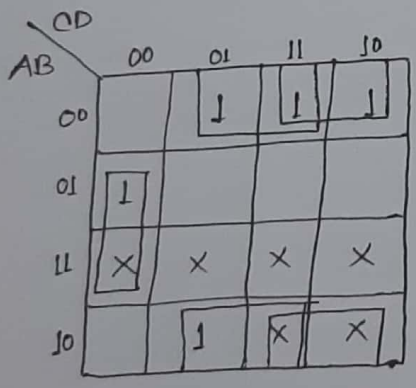
Answer: Maps for BCD-to-excess 3 code converter are given below:



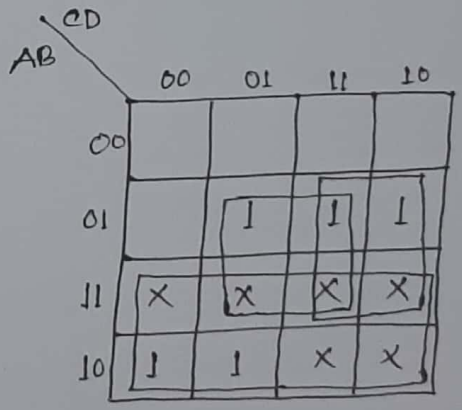
$z = D'$



$y = CD + C'D'$



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



$w = A + BC + BD$

$w = A + BC + BD$

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

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

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

$y = CD + C'D'$

$y' = C'D + CD'$

$z = D'$

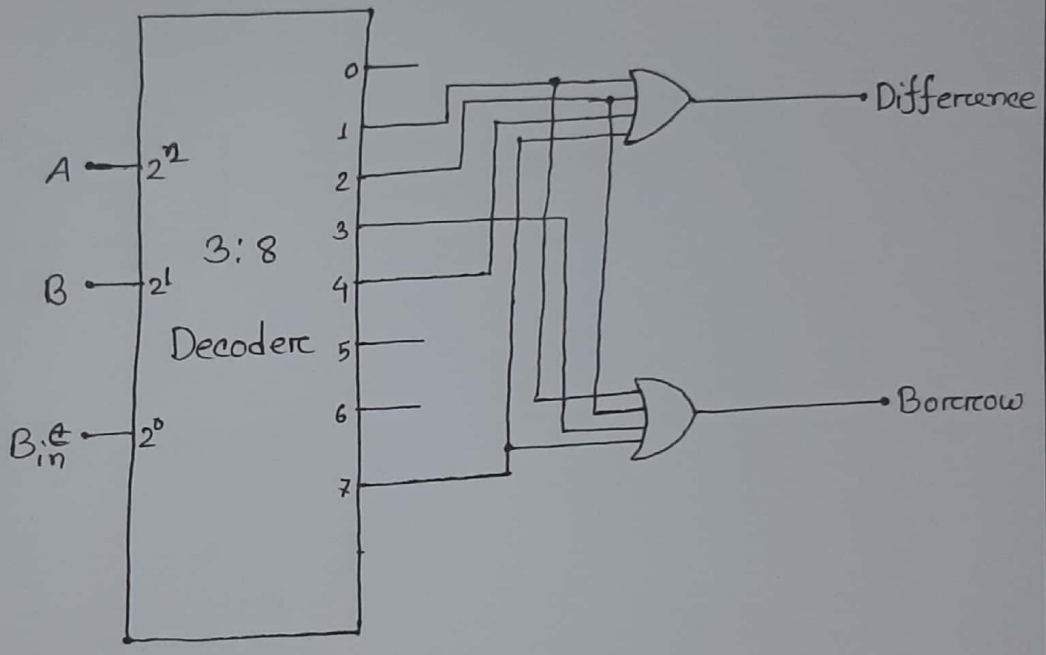
$z' = D$

Programming Table:

	Product term	Inputs				Outputs			
		A	B	C	D	$F_1(T)$	$F_2(C)$	$F_3(T)$	$F_4(T)$
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

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

Answer: A full-subtractor using 3×8 decoder and two OR gates in given below:



5.33: With 2×1 mux ~~to~~ implement XOR gate and AND gate

Answer: Truth table of XOR gate-

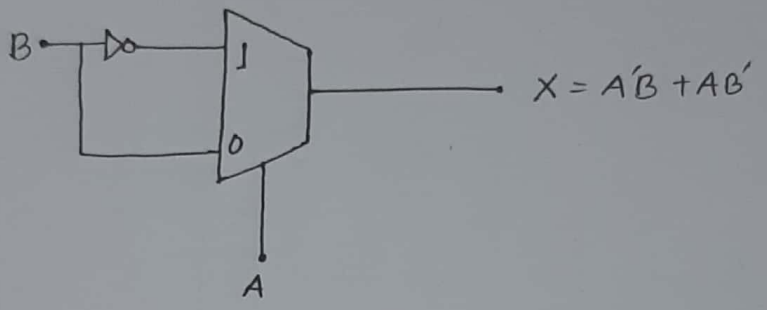
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

K-map:

A \ B	0	1
0	0	1
1	1	0

$X = AB' + A'B$

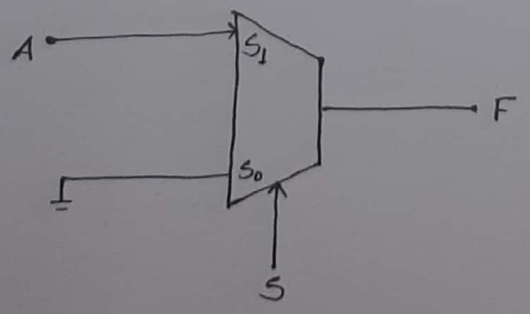
Implementation of XOR gate:



Truth table of AND gate:

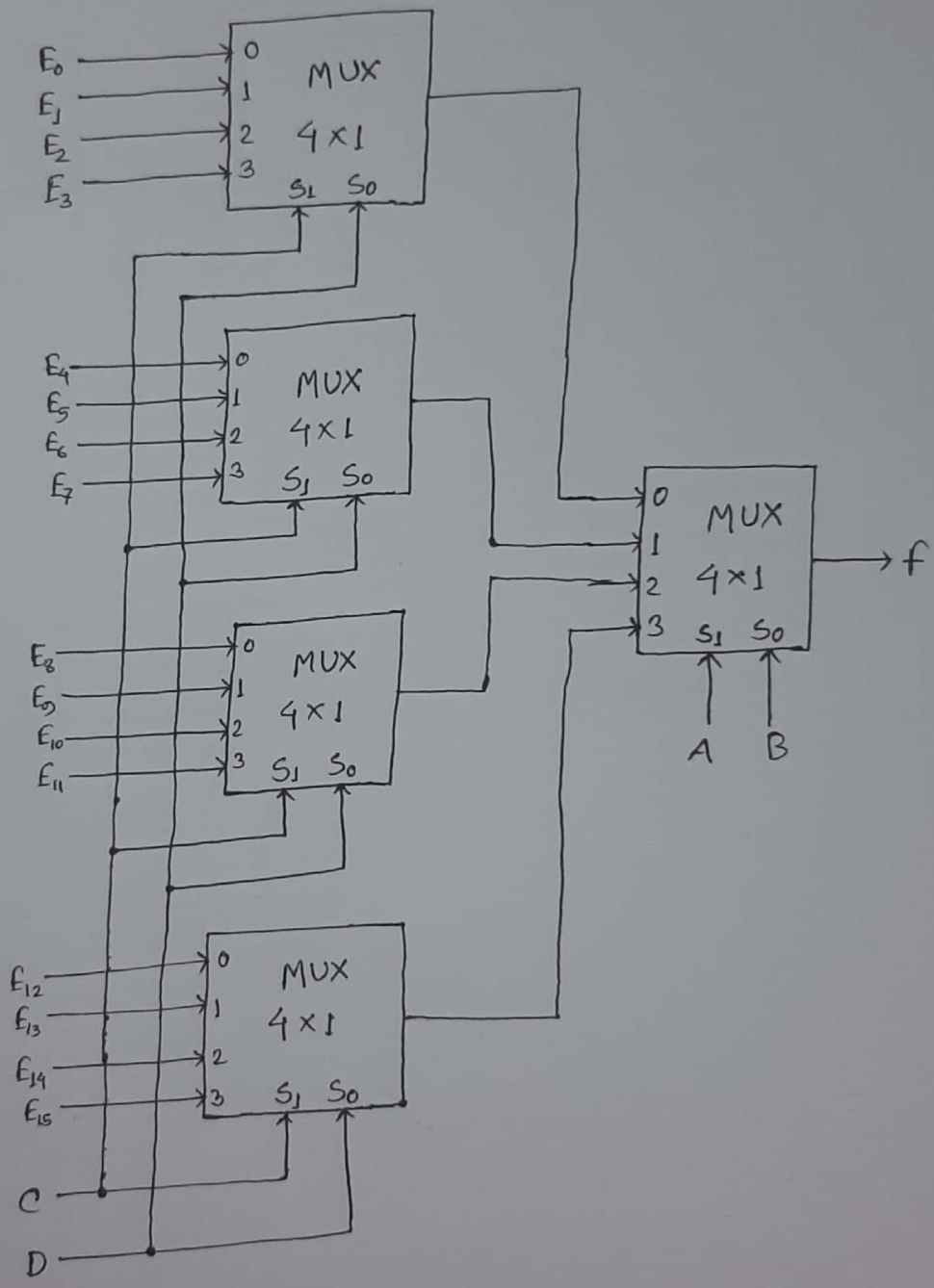
S	A	F
0	0	0
0	1	0
1	0	0
1	1	1

Implementation of AND gate:



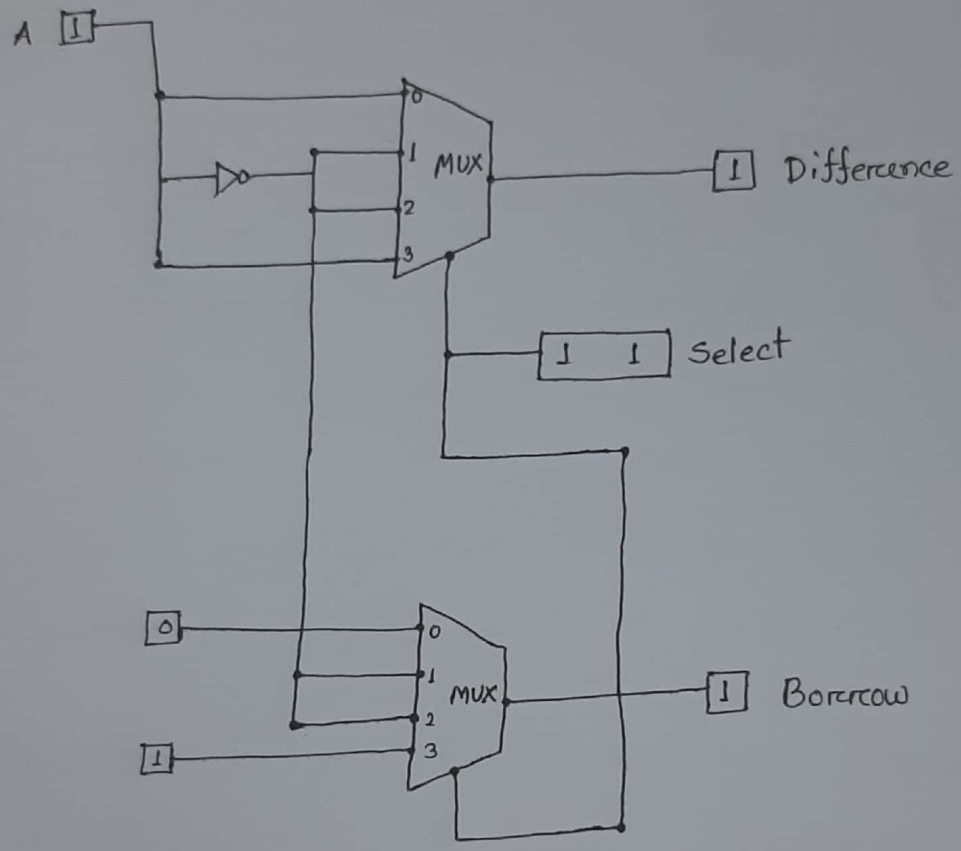
5.34 : Construct 16×1 mux using five 4×1 only.

Answer: Construction of a 16×1 multiplexer using five 4×1 only in given below:



5.35: Implement a full-subtractor circuit with multiplexer.

Answer: Implementation with of a full-subtractor circuit with multiplexer is given below-



5.37: Differentiate between

a) PLA and ROM

b) MUX, Decoder

Answer:

a) Difference between PLA and ROM:

PLA	ROM
1. Both AND and OR arrays are configurable.	1. Only the OR gates array is configurable.
2. It can take 'don't care term' into account.	2. It can't take 'don't care term' into account.
3. It doesn't have all the combinations of product terms	3. It has all the combinations of product terms.

b) Difference between Multiplexer and Decoder:

Multiplexer	Decoder
1. It accepts several inputs & allows only one data output.	1. It takes n input binary code & converts into a corresponding outputs.
2. Select line are used to select data input and allow only one of them.	2. Enable inputs are used to control the operation of the decoder.
3. It can be used in data routing & waveform generation.	3. Application of decoder is in Decimal to BCD encoder.
4. It converts the unary code into binary code.	4. Decoder converts binary code into unary code.