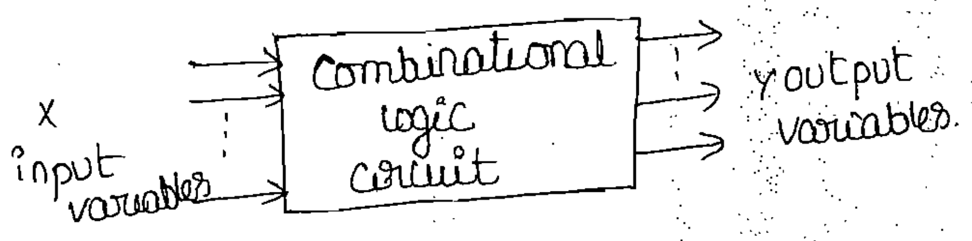
126 Combinational logic circuits design.

logic circuits for digital systems may be combinational or sequential. In combinational circuits, the output variable at any instant of time are dependent only on the present input variables. In sequential circuits the output variables at any instant of time are dependent on the present and past input variables.

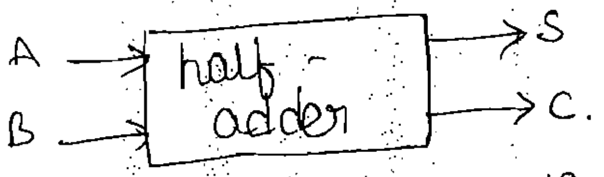


Adders :-

The most basic arithmetic operation is the addition of two binary digits. A combinational circuit that performs the addition of two bits is called a "half-adder". one that performs the addition of three bits (two bits and previous carry) is called a "full-adder".

Half - adder

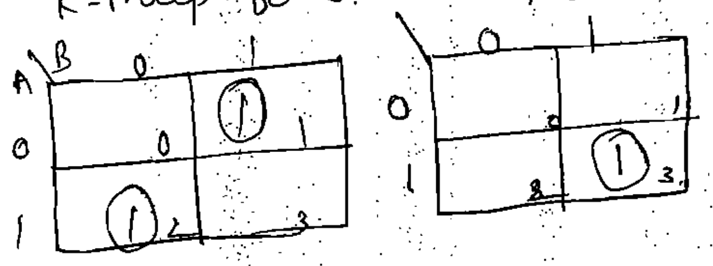
A half adder is a combinational circuit with two binary inputs. (augend and addend bits) and two outputs. sum and carry.



Inputs		output	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(a) Truth table.

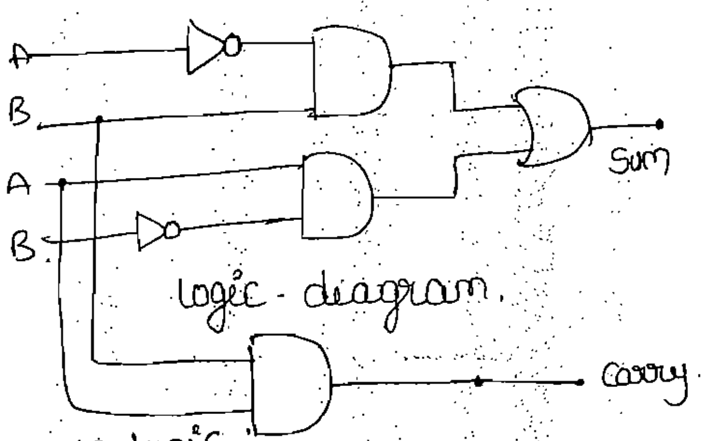
K-map for S.   K-map for C



$S = A\bar{B} + \bar{A}B$

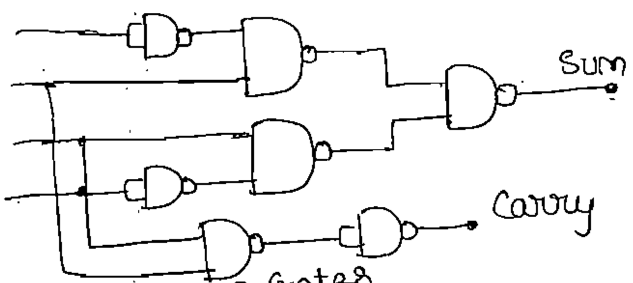
$= A \oplus B$

$C = A \cdot B$



logic diagram.

NAND Gates



~~NOR Gates~~.

NAND logic :-

$S = A\cdot B + \bar{A}B$

$S = A\cdot\bar{B} + \bar{A}B + A\cdot\bar{A} + B\cdot\bar{B}$

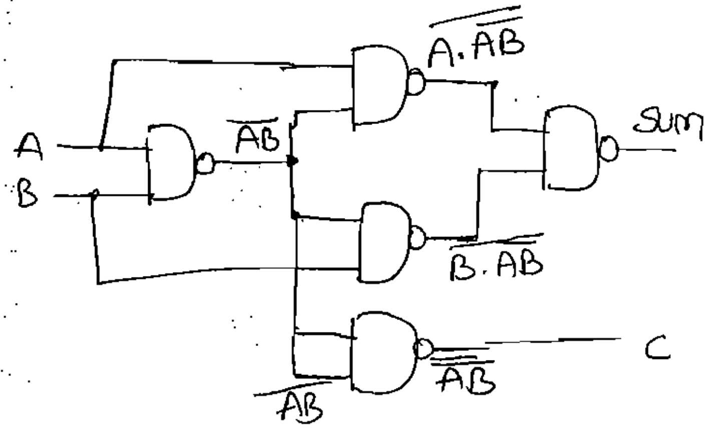
$= A(\bar{A}+\bar{B}) + B(\bar{A}+\bar{B})$

$= A\cdot\overline{AB} + B\cdot\overline{AB}$

$= \overline{\overline{A\cdot\overline{AB} + B\cdot\overline{AB}}}$

$= \overline{\overline{A\cdot\overline{AB}}\cdot\overline{B\cdot\overline{AB}}}$

$C = AB = \overline{\overline{AB}}$


A
B
$\overline{AB}$
$\overline{A\cdot\overline{AB}}$
Sum
$\overline{B\cdot\overline{AB}}$
C
$\overline{\overline{AB}}$
$\overline{AB}$

NOR logic

$S = A\cdot\bar{B} + \bar{A}\cdot B$

$= A\cdot\bar{B} + \bar{A}\cdot B + A\cdot\bar{A} + B\cdot\bar{B}$

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

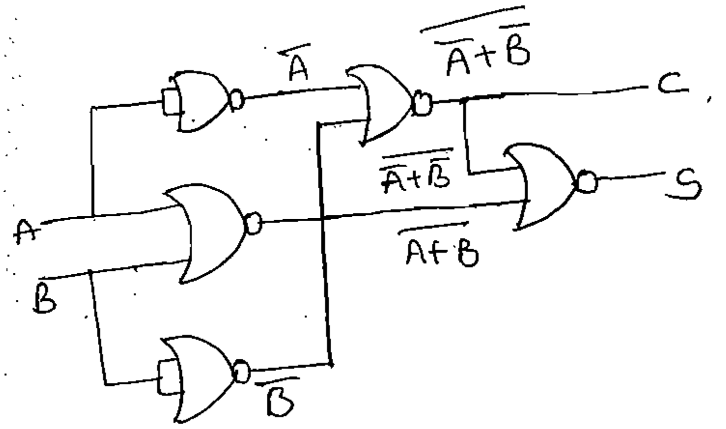
$= (A+B)(\bar{A}+\bar{B})$

$= \overline{\overline{(A+B)(\bar{A}+\bar{B})}}$

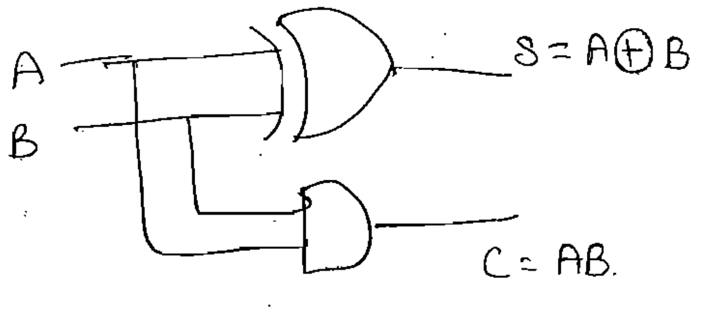
$= \overline{\overline{(A+B)} + \overline{(\bar{A}+\bar{B})}}$

$C = AB = \overline{\overline{AB}}$ = ~~AB~~.

$= \overline{\bar{A}+\bar{B}}$

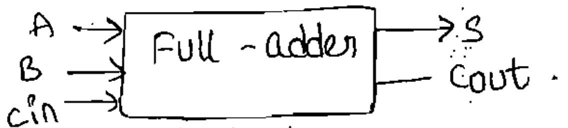


Simple logic diagram is.



Full adder :-

A full adder is a combinational circuit that adds two bits and a carry and outputs are sum and carry. The full-adder adds the bits A and B and the carry from the previous column called the carry-in C_{in}.

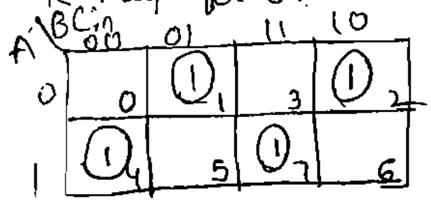


Block diagram.

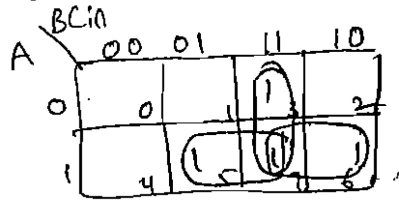
inputs			outputs	
A	B	C_{in}	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Truth table

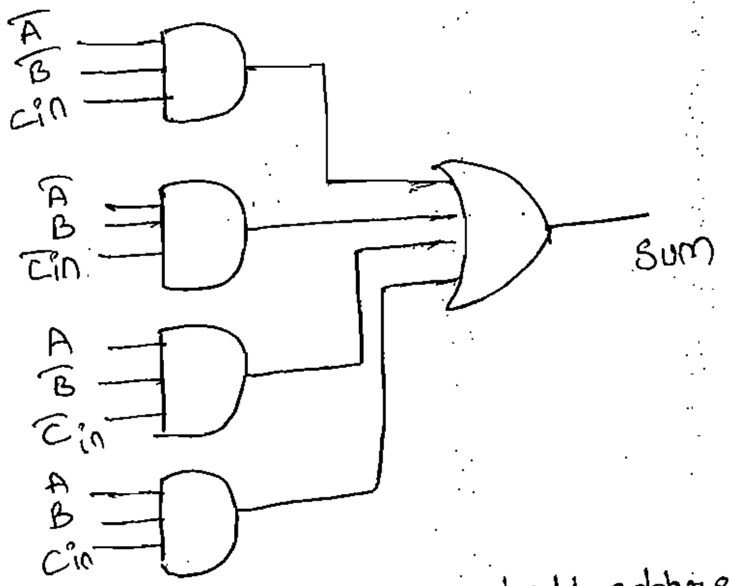
K-map for S.

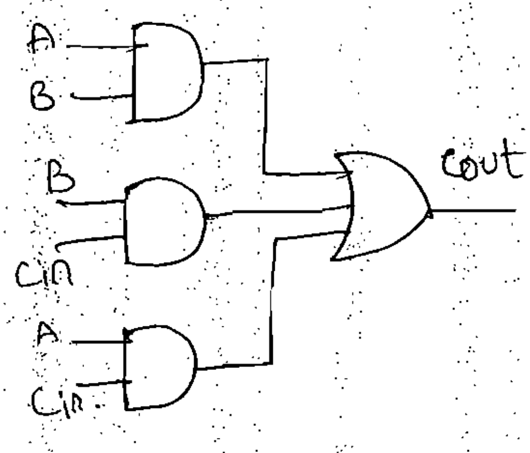


$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$.



$C_{out} = AB + BC_{in} + AC_{in}$.





Full adder (By using two half adders and one OR gate).

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

$= C_{in}(AB + \bar{A}\bar{B}) + \bar{C}_{in}(\bar{A}B + A\bar{B})$

$= \overline{A \oplus B}\, C_{in} + \bar{C}_{in}(A \oplus B)$

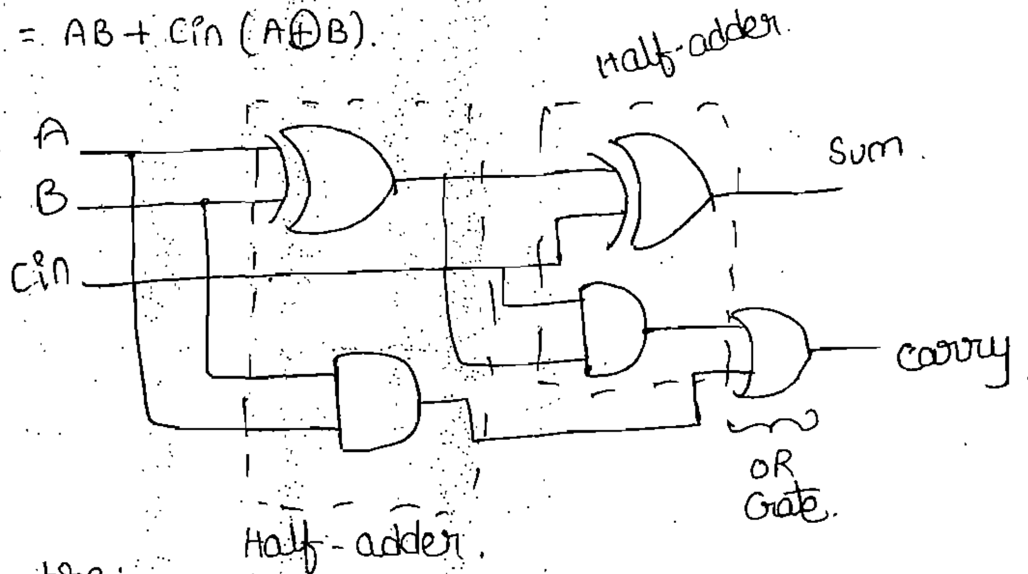
$= A \oplus B \oplus C_{in}$.

$C_{out} = \overline{A}BC_{in} + A\overline{B}C_{in} + AB\overline{C}_{in} + ABC_{in}$

$= AB(C_{in}+\overline{C}_{in}) + \overline{A}BC_{in} + A\overline{B}C_{in}$

$= AB + C_{in}(\overline{A}B + A\overline{B})$

$= AB + C_{in}(A \oplus B).$

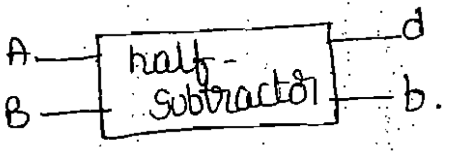


Subtractors:-

In subtraction, each subtrahend bit of the number is subtracted from its corresponding significant minuend bit to form a difference position.

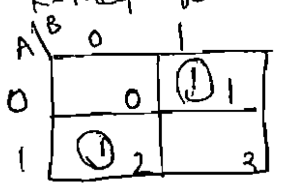
Half-subtractor:-

A half subtractor is a combinational circuit that subtracts one bit from the other and produces the difference. It also has an output to specify if a 1 has been borrowed.

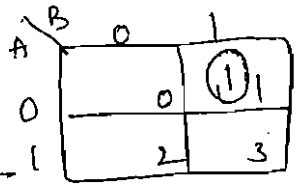


inputs		outputs	
A	B	d	b
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

K-map for d



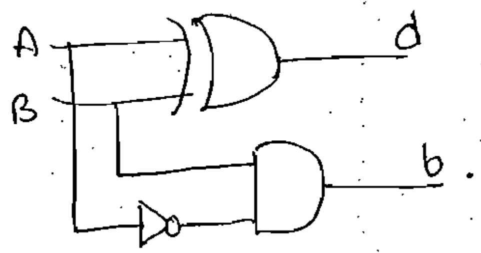
$d = A\overline{B} + \overline{A}B$

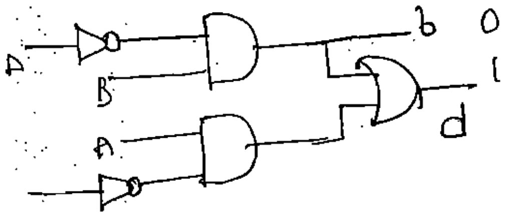


$b = \overline{A}B$

$= A\overline{B} + \overline{A}B$

$= A \oplus B$





logic diagrams of a half-subtrator.

NAND logic :-

$d = A\bar{B} + \bar{A}B$

$= A\bar{B} + \bar{A}B + A\cdot\bar{A} + B\bar{B}$

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

$= \overline{\overline{A\cdot\overline{AB} + B\cdot\overline{AB}}}$

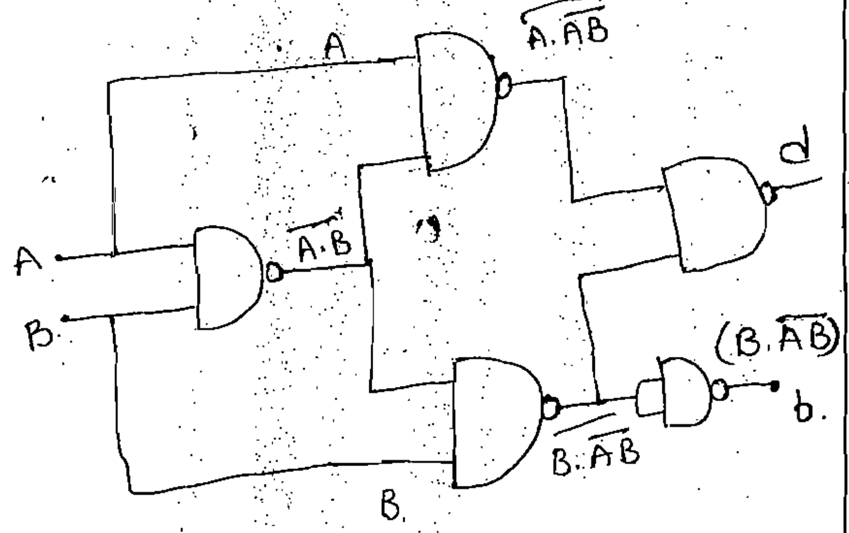
$= \overline{\overline{A\cdot\overline{AB}}\cdot\overline{B\cdot\overline{AB}}}$

$b = \bar{A}B$

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

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

$= B(\overline{AB})$.



NOR logic

$d = A\bar{B} + \bar{A}B$

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

$= \overline{\bar{B}(A+B)} + \overline{\bar{A}(A+B)}$

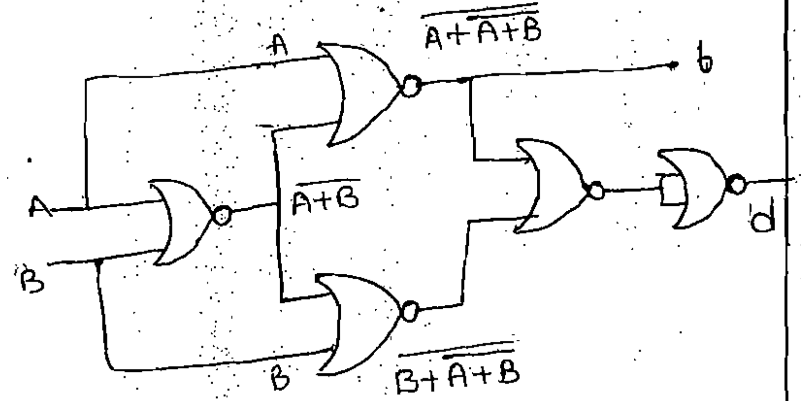
$= \overline{B+\overline{A+B}} + \overline{A+\overline{A+B}}$

$b = \bar{A}B$

$= \bar{A}B + A\cdot\bar{A}$

$= \overline{\bar{A}(A+B)}$

$= \overline{A + (\overline{A+B})}$



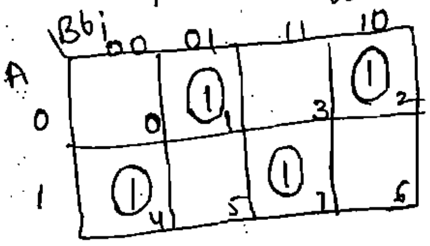
Full - subtractor :-

The half - ~~adder~~ subtrator can be used, only for LSB subtration. If there is a borrow during the subtration of the LSBs, it affects the subtraction in the next higher column. the subtrahend bit is subtracted from the minuend bit, considering the borrow from that column used for the subtraction in the preceding column.

In full subtractor the inputs are A, B, borrow in b_i, and outputs are difference bit (d) and borrow (b).

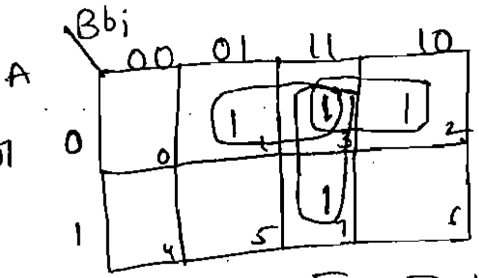
inputs			outputs	
A	B	b_i	d	b
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-map for difference

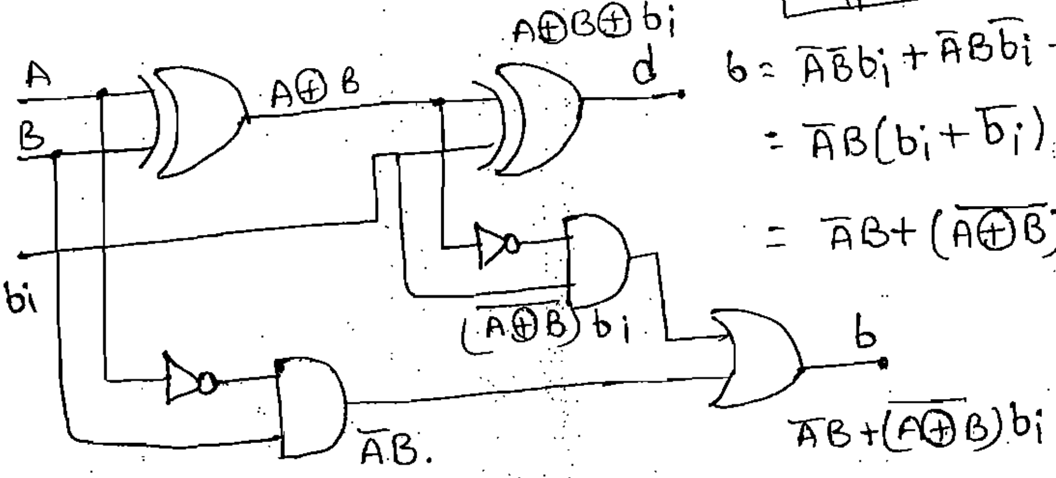


$$d = \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + A\bar{B}\bar{b}_i + ABb_i$$
$$= b_i(AB + \bar{A}\bar{B}) + \bar{b}_i(\bar{A}B + A\bar{B})$$
$$= b_i(\overline{A \oplus B}) + \bar{b}_i(A \oplus B)$$
$$= A \oplus B \oplus b_i$$

K-map for borrow.



Full-subtractor by using two half-subtractor



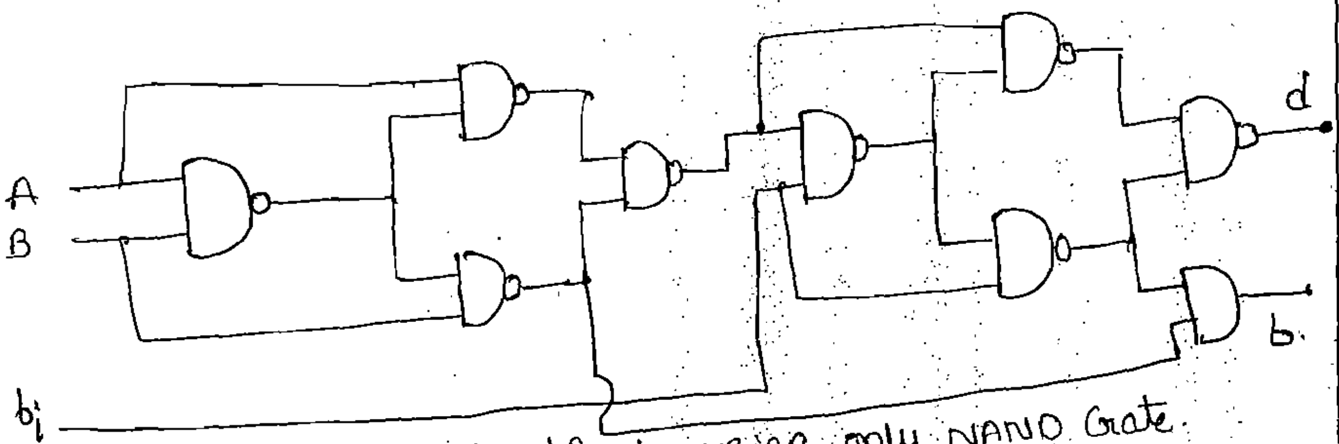
A
B
b_i
$A \oplus B$
$A \oplus B \oplus b_i$
d
$(\overline{A \oplus B})b_i$
b
$\bar{A}B$.
$\bar{A}B + (\overline{A \oplus B})b_i$

$$b = \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + \bar{A}Bb_i + ABb_i$$
$$= \bar{A}B(b_i + \bar{b}_i) + (AB + \bar{A}\bar{B})b_i$$
$$= \bar{A}B + (\overline{A \oplus B})b_i$$

NAND logic:-

$$d = A \oplus B \oplus b_i = \overline{\overline{(A \oplus B) \oplus b_i}} = \overline{\overline{(A \oplus B)\overline{(A \oplus B)b_i}} \cdot \overline{b_i\overline{(A \oplus B)b_i}}}$$

$$b = \bar{A}B + b_i(\overline{A \oplus B}) = \overline{\overline{\bar{A}B + b_i(\overline{A \oplus B})}}$$
$$= \overline{\overline{\bar{A}B} \cdot \overline{b_i(\overline{A \oplus B})}} = \overline{\overline{B(\overline{A+\bar{B}})} \cdot \overline{b_i[\overline{b_i} + (\overline{A \oplus B})]}}$$
$$= \overline{\overline{B.\overline{AB}} \cdot \overline{b_i[\overline{b_i \cdot (A \oplus B)}]}}$$



full subtractor by using only NAND Gate.

NOR logic.

$$d = \overline{\overline{A\oplus B\oplus b_i}}$$

$$= \overline{\overline{(A\oplus B)b_i + \overline{(A\oplus B)}\,\overline{b_i}}}$$

$$= \overline{\overline{[(A\oplus B) + \overline{(A\oplus B)}\,\overline{b_i}]\,[b_i + \overline{(A\oplus B)}\,\overline{b_i}]}}$$

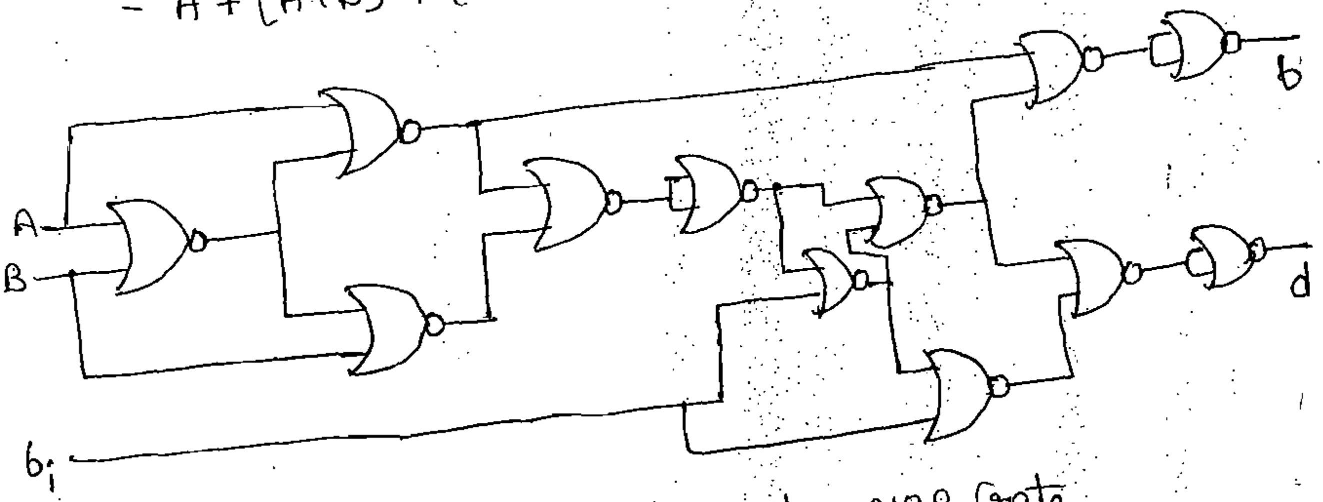
$$= \overline{\overline{(A\oplus B) + \overline{\overline{(A\oplus B)} + b_i}} + \overline{b_i + \overline{(A\oplus B) + b_i}}}$$

$$= \overline{\overline{\overline{\overline{(A\oplus B) + \overline{\overline{(A\oplus B)} + b_i}} + \overline{b_i + \overline{(A\oplus B) + b_i}}}}}$$

$$b = \overline{A}B + b_i\,\overline{(A\oplus B)}$$

$$= \overline{A}(A+B) + \overline{(A\oplus B)}\,[A\oplus B + b_i]$$

$$= \overline{\overline{\overline{A + \overline{(A+B)}} + \overline{(A\oplus B)} + \overline{(A\oplus B) + b_i}}}$$

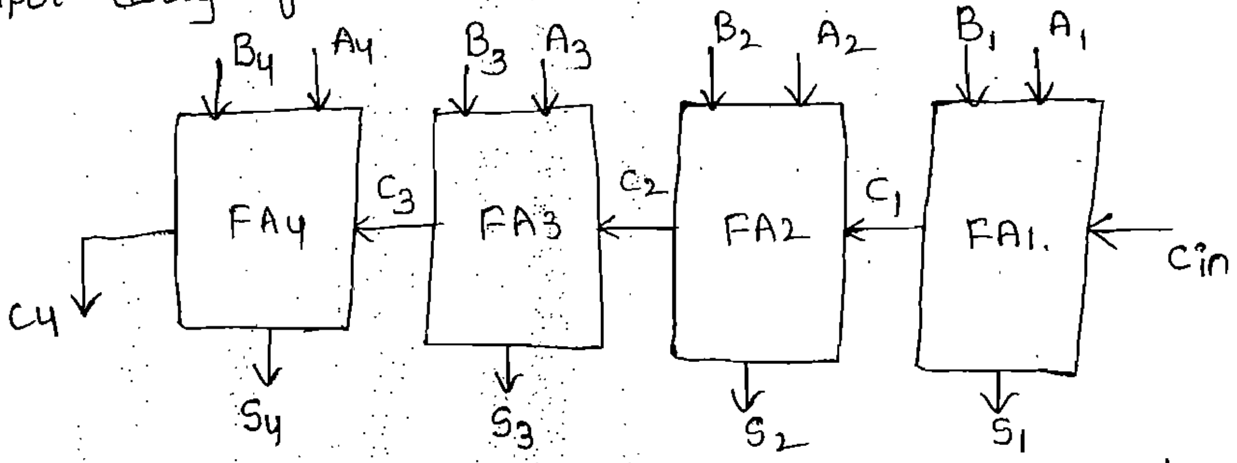


Full subtractor by using only NOR Gate

Applications of full adders :-

Binary parallel adder :-

A Binary parallel adder is a digital circuit that adds two binary numbers in parallel form and produces the arithmetic sum of those numbers in parallel form. It consists of full adders connected in a chain, with the output carry from each full-adder connected to the input carry of the next full-adder in the chain.



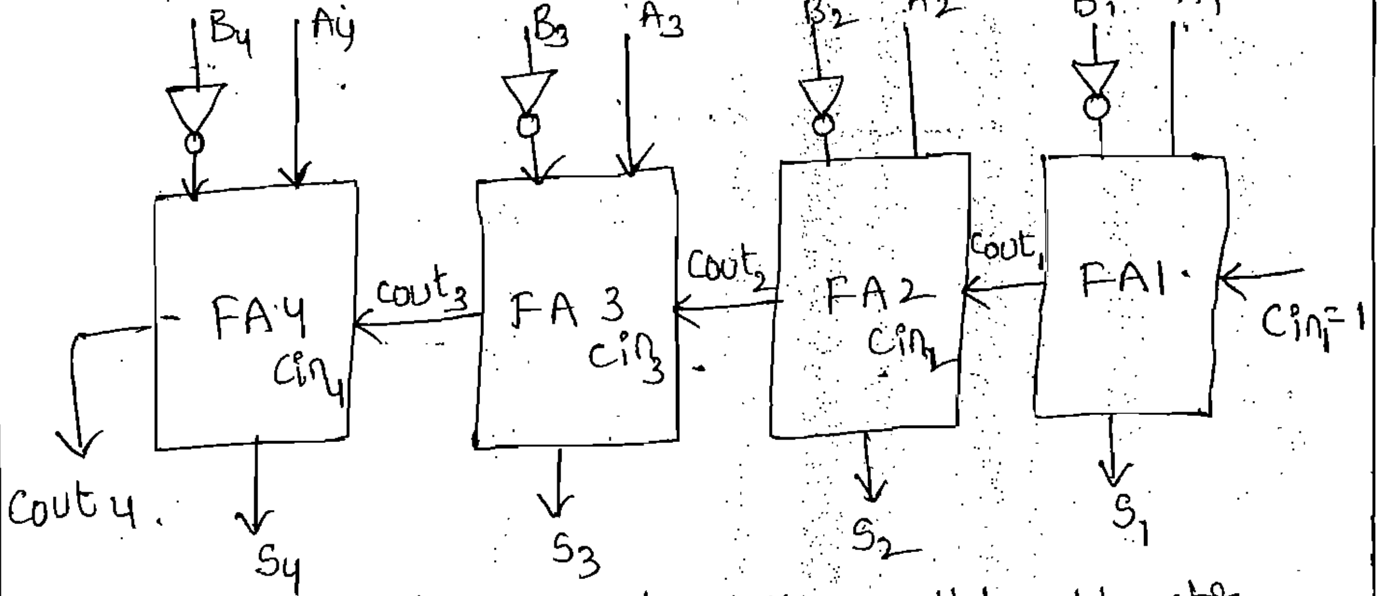
Logic diagram of 4-bit binary parallel adder.

The inter-connection of full-adder (FA) circuits to provide a 4-bit parallel adder. The augend bits of A and addend bits of B are designated by subscript numbers from right to left, with subscript 1 denoting the lower-order bit. The input carry to the adder is Cin and the output carry is C_4. The S outputs generate the required sum bits. when the 4-bit full adder circuit is enclosed within an IC package, it has four terminals for the augend bits, four terminals for the addend bits, four terminals for the sum bits and two input terminals for the input and output carries.

The parallel adder in which the carry-out of each full adder is the carry-In to the next most significant adder is called a ripple carry adder. In the parallel adder the carry-out of each stage is connected to the carry-in of the next stage. The sum and carry out bits of any stage cannot be produced, until some time after the carry-In of that stage occurs. This is due to the propagation delays in the logic circuitry, which lead to a time delay in the addition process.

4-bit parallel subtractor:-

The subtraction of binary numbers can be carried out most conveniently by means of complements. The subtraction A-B can be done by taking the 2's complement of B and adding it to A. The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits. The 1's complement can be implemented with not gate (inverters).



Logic diagram of 4-bit parallel subtractor.

Binary adder - Subtractor :-

The addition and subtraction operations are combined into one circuit with one common binary adder. This is done by including an X-OR gate with each Full adder. The M mode input controls the operation.

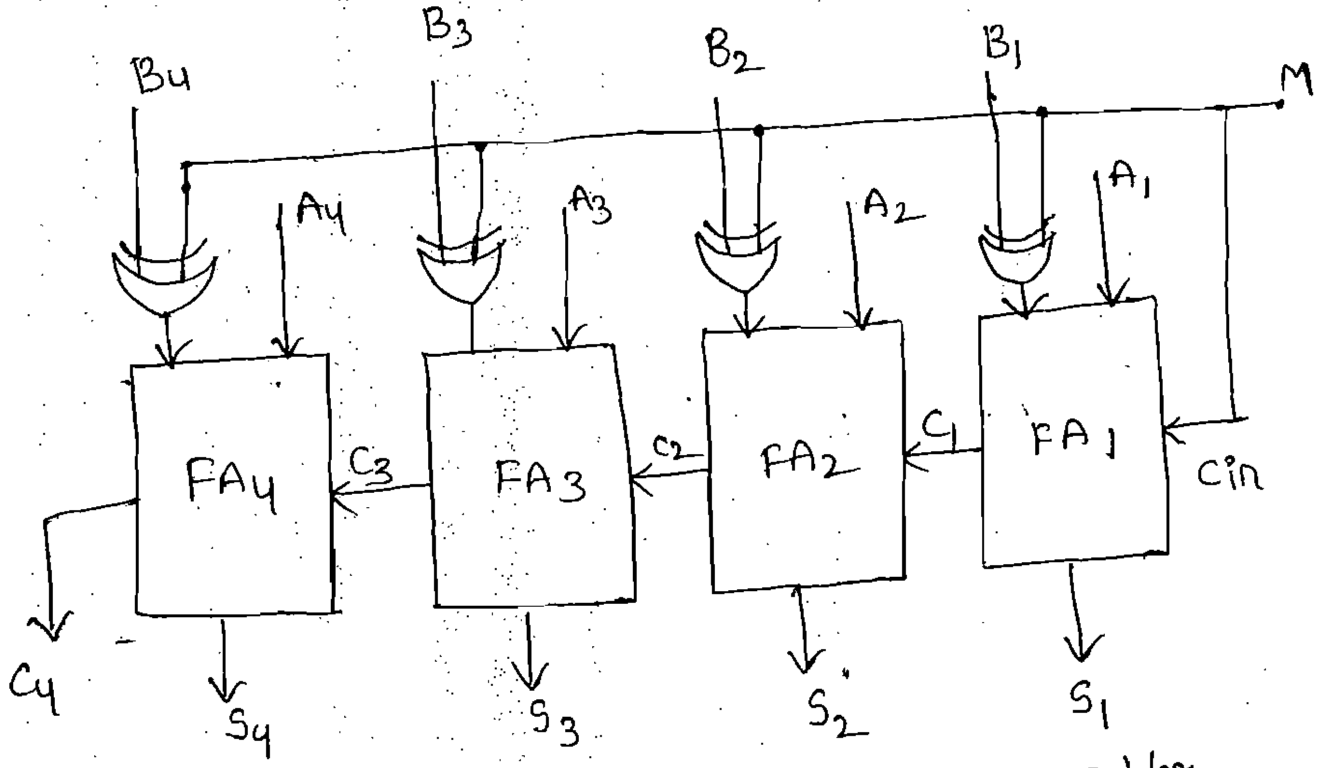
→ when M=0, the circuit is an adder.

→ when M=1, the circuit is an subtractor.

Each XOR gate receives input M and one of the inputs of B.

→ when M=0, $B \oplus 0 = B$. The full adder receives the value of B, the input carry is '0' and the circuit performs A+B.

→ when M=1, $B \oplus 1 = \overline{B}$, The full adder receives the value of $\overline{B}$, the input carry is '1' and the circuit performs A-B.

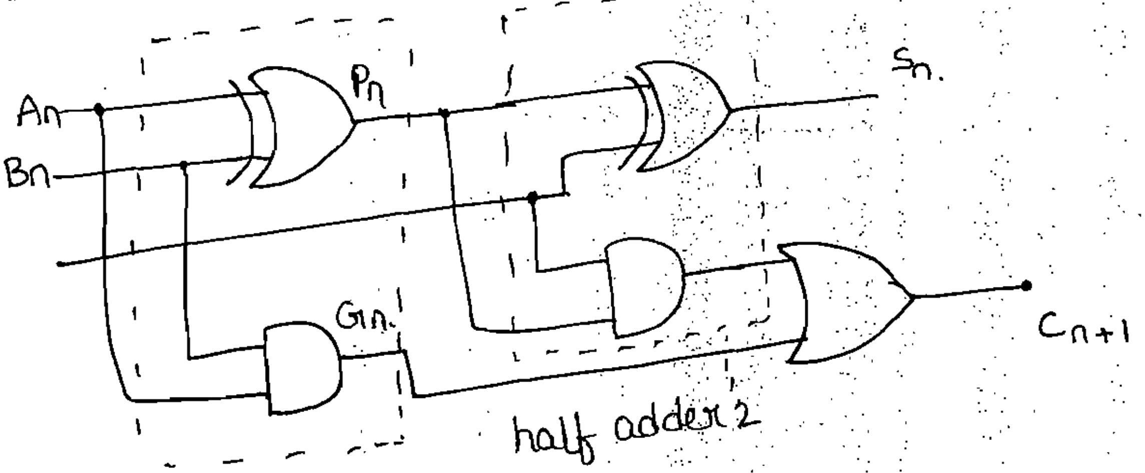


Logic diagram of a 4-bit binary - adder - Subtractor.

LOOK-A-HEAD-CARRY ADDER :-

The parallel adder, the speed with which an addition can be performed is governed by the time requires for the carries to propagate or ripple through all of the stages of the adder.

The look-ahead-carry adder speeds up the process by eliminating this ripple carry delay. It examines all the input bits simultaneously. The methode of speeding up the process is based on the two additional functions of the full adder, called the carry generate and carry propagate functions.



Half adder 1

carry generate :-

consider one full adder stage, nth stage of a parallel adder. carry is generated only if both the input bits are 1, that is, if both the bits A and B are 1s, a carry has to be generated in this stage regardless of whether the input carry Cin is a 0 or a 1. If Gn is a carry-generation function.

$G_n = A \cdot B$.

The present bit as the nth bit, then G rewrite as a

$$G_n = A_n \cdot B_n.$$

carry propagation :-

A carry is propagated if any one of the two input bits A & B are 0, a carry will never be propagated. on the other hand, if both A and B are 1, then it will not propagate the carry but will generate the carry.

If P is taken as a

$$P = A \oplus B.$$

The present bit as the nth bit, then P rewrite as a

$$P_n = A_n \oplus B_n.$$

For the final sum and carry outputs of the nth stage.

$$S_n = P_n \oplus C_n$$

$$(\because P_n = A_n \oplus B_n).$$

$$C_{on} = C_{n+1} = G_n + P_n C_n$$
$$= A_n \cdot B_n + P_n C_n$$
$$= A_n \cdot B_n + (A_n \oplus B_n) C_n.$$

Based on these, the expression for the carry-outs of various full-adders are.

$n=1,$ $C_1 = G_0 + P_0 C_0$
$$= G_0 + (A_0 \oplus B_0) C_0$$
$$= A_0 \cdot B_0 + (A_0 \oplus B_0) C_0.$$

$n=2$

$C_2 = G_1 + P_1 \cdot C_1 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$

$n=3$

$C_3 = G_2 + P_2 \cdot C_2 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 C_0$

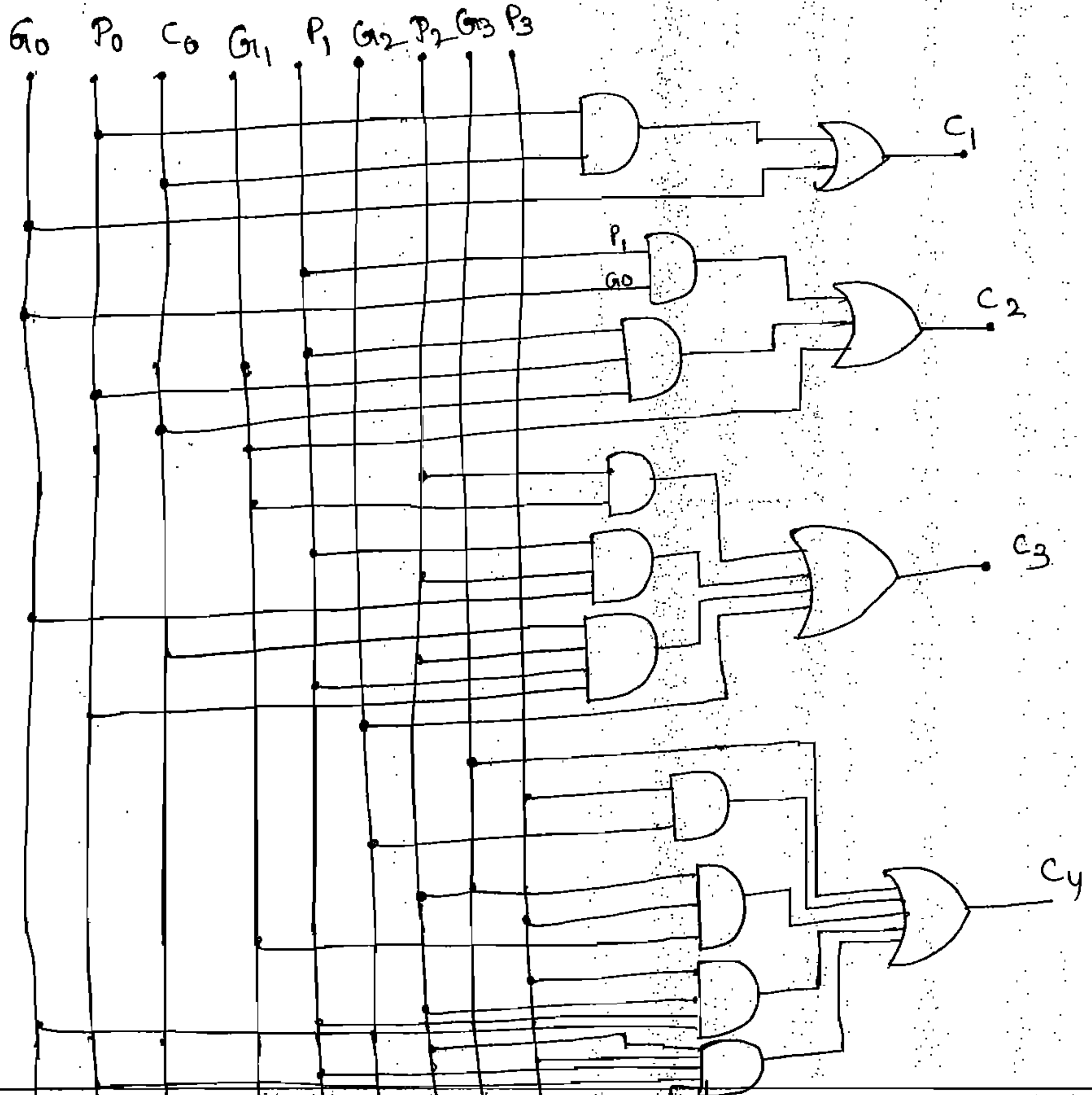
$n=4$

$C_4 = G_3 + P_3 \cdot C_3 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 P_1 G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$

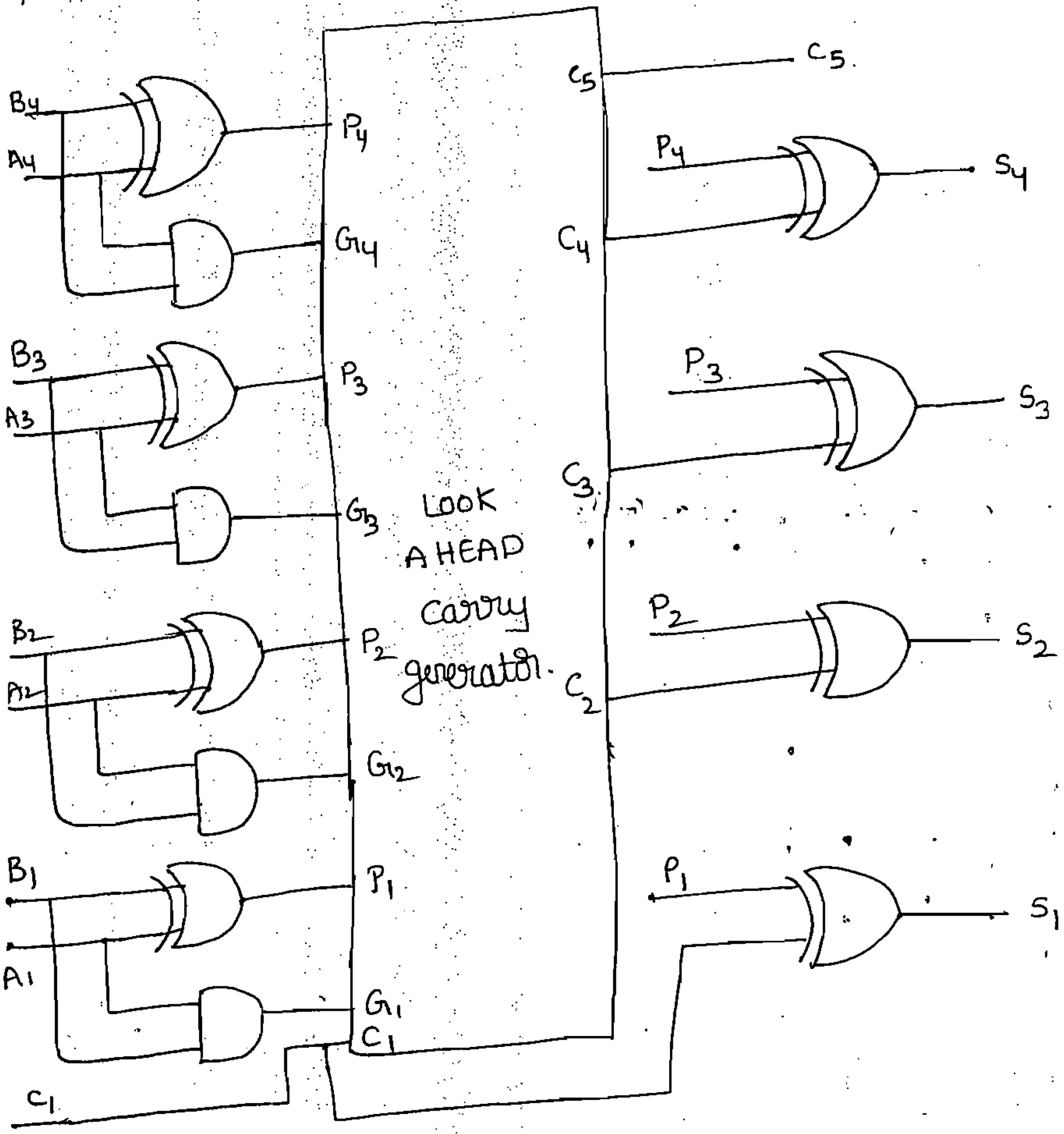
The general expression for n-stages.

$C_n = G_{n-1} + P_{n-1} \cdot C_{n-1}$

$= G_{n-1} + P_{n-1} \cdot G_{n-2} + P_{n-1} \cdot P_{n-2} \cdot G_{n-3} + \cdots + P_{n-1} \cdots P_0 \cdot C_0$

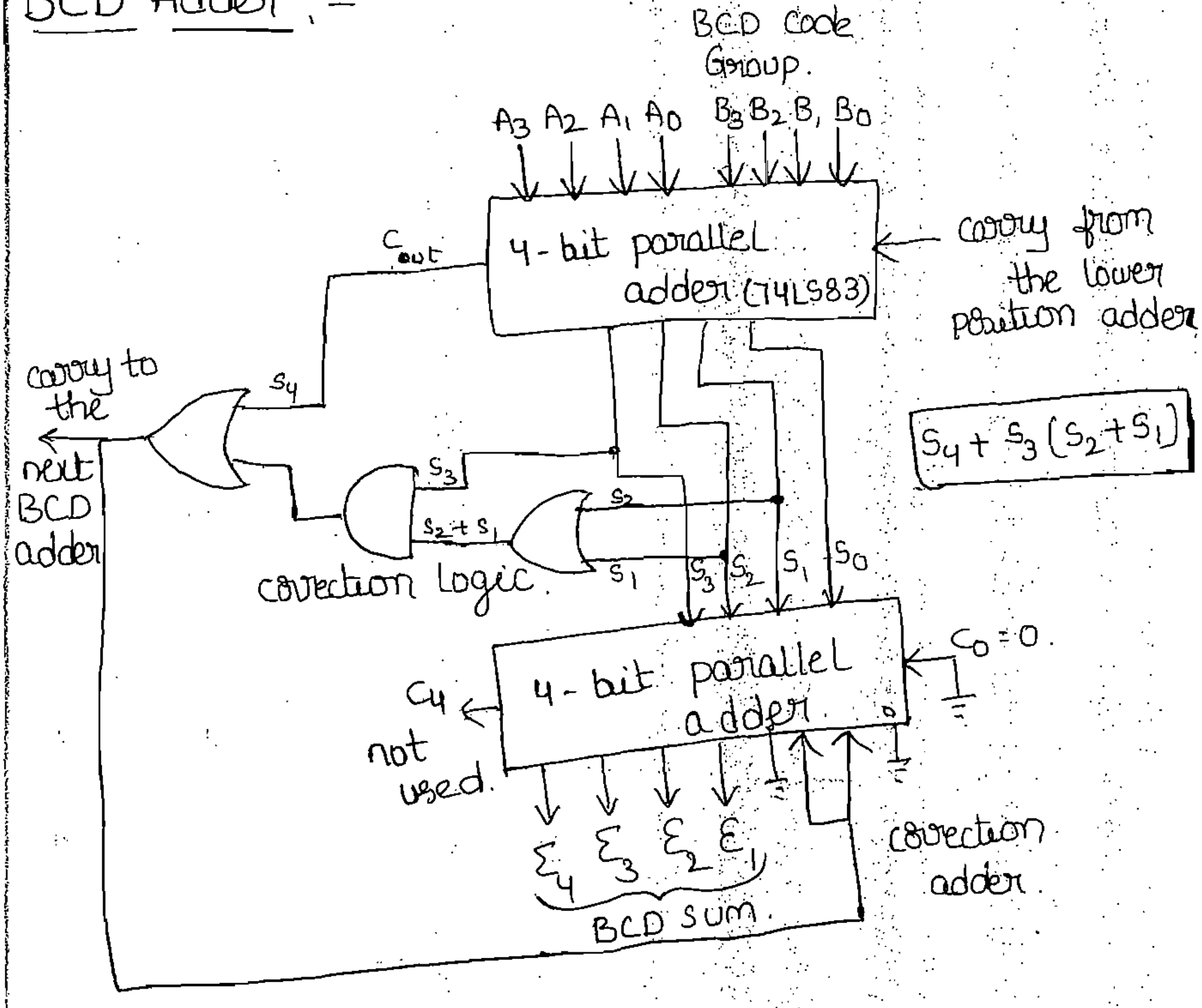


The block diagram of a 4-stage look-ahead-carry parallel adder is shown in the below figure.



Basic logic diagram of a 4-bit Look-Ahead carry adder.

BCD Adder :-



→ In BCD adder, Add the 4-bit BCD code groups for each decimal digit position using ordinary binary addition.

→ For those positions where the sum is 9 or less, the sum is in proper BCD form and no correction is needed.

→ where the sum of two digits is greater than 9, a correction of 0110 should be added to that sum, to produce the proper BCD result.

→ In above figure 4-bit parallel adder (using IC 74LS83). The two BCD groups A_3, A_2, A_1, A_0 and B_3, B_2, B_1, B_0 are applied to a 4-bit parallel adder.

The adder output will be ~~$

The circuit consists of three basic parts. The BCD code groups $A_3, A_2, A_1 A_0$ and B_3, B_2, B_1, B_0 are added together in upper 4-bit parallel adder to produce the sum $S_4 S_3 S_2 S_1 S_0$. The logic gates shown implement the expression for X. The lower 4-bit adder will add the ~~add~~ correction 0110 to the sum bits only when X = 1, producing final BCD sum output represented by $\Sigma_3 \Sigma_2 \Sigma_1 \Sigma_0$.

when x = 0, there is no carry and no correction.

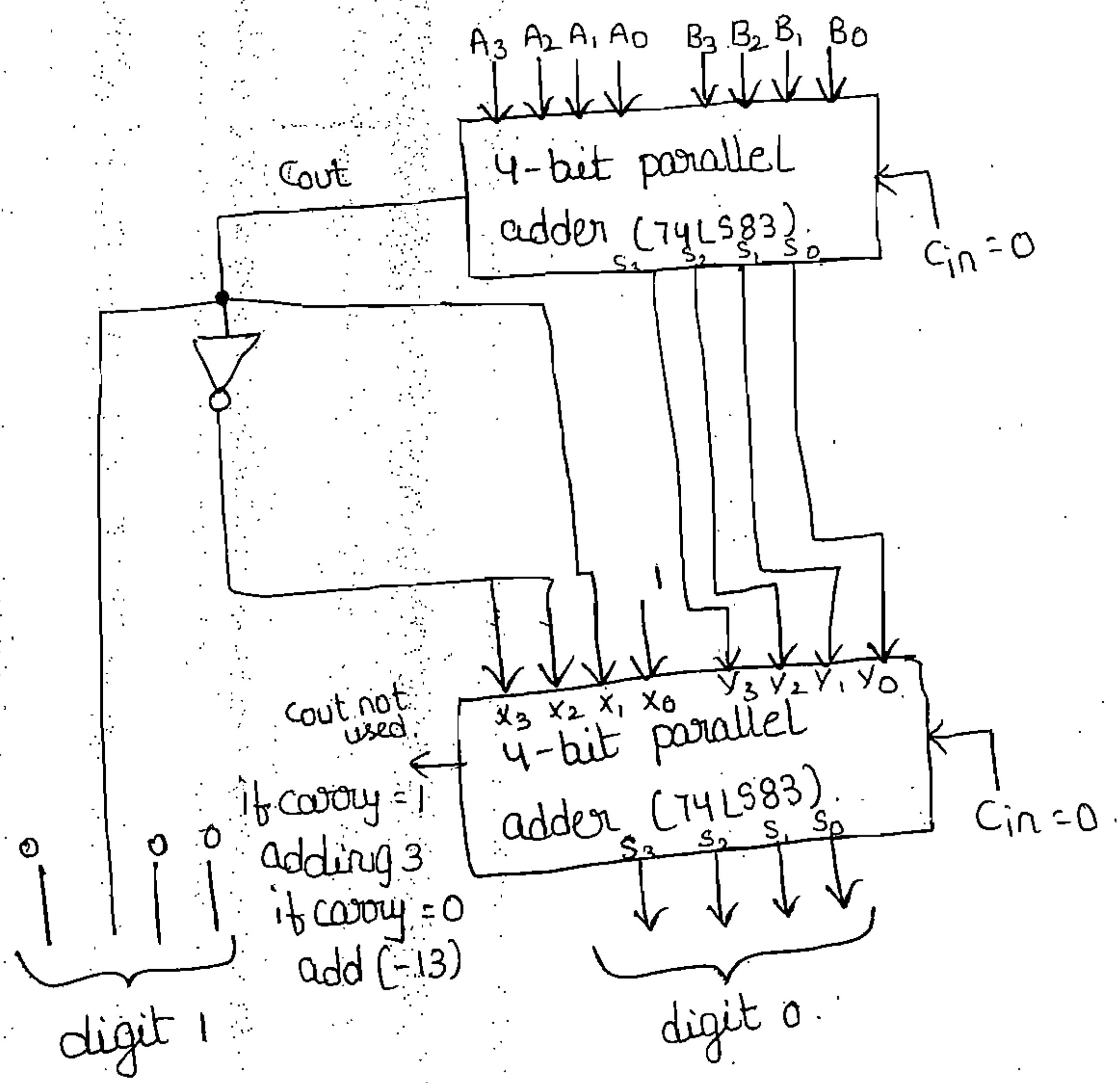
In such cases $\Sigma_3 \Sigma_2 \Sigma_1 \Sigma_0 = S_3 S_2 S_1 S_0$. Two or more BCD adders can be connected in cascade when two or more digit decimal numbers are to be added. The carry-out of the first BCD adder is connected as the carry-in of the second BCD adder.

Excess-3 Adder :-

→ In Excess-3 addition.

1. Add two XS-3 code groups.
2. If carry = 1 add 0011
   if carry = 0 subtract 0011, or add 1101 (13 in decimal).

In figure the augend $(A_3, A_2, A_1 A_0)$ and addend $(B_3 B_2 B_1 B_0)$ in XS-3 added using the 4-bit parallel adder. If the carry is a 1, then 0011 is added to the sum bits $S_3 S_2 S_1 S_0$ of the upper adder in the lower 4-bit parallel adder. If the carry is a '0' then 1101 is added to the sum bits.



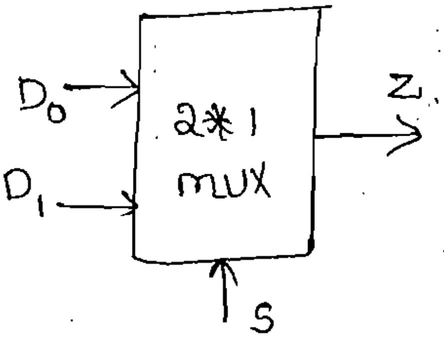
The final Answer in XS-3 form.

Multiplexers (data selectors).

multiplexing means sharing. A multiplexer or data selector is a logic circuit that accepts several data inputs and allows only one of them at a time to get through to the output. The routing of the desired data inputs to the output is controlled by SELECT inputs. Normally there are 2^n input lines and n select lines and one output.

Basic 2-input multiplexer :-

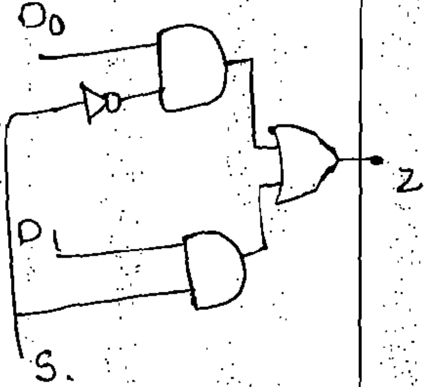
In 2-input multiplexer have 2-inputs they are D_0 and D_1. and one select line S. and output is Z.



Block diagram.

S	Z
0	D_0
1	D_1

Truth table.



$Z = \bar{S}D_0 + SD_1$

Logic diagram.

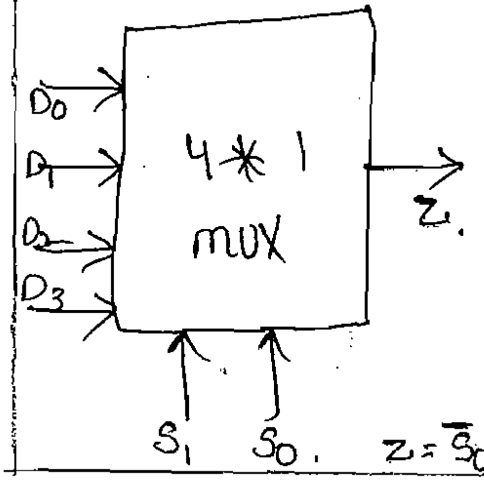
The logic levels applied to the S inputs determines which AND gate is enabled. So that its data input passes through the OR gate to the output.

when S=0, AND gate 1 is enabled and AND gate 2 is disabled, S_0, $Z = D_0$

S=1, AND gate 2 is enabled and AND gate 1 is disabled, S_0, $Z = D1$.

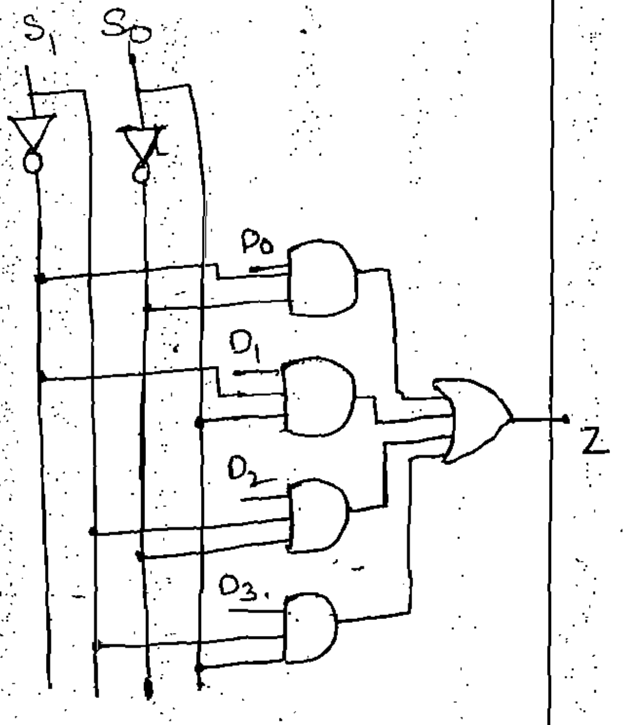
Basic 4-input multiplexer :-

Block diagram



Truth table

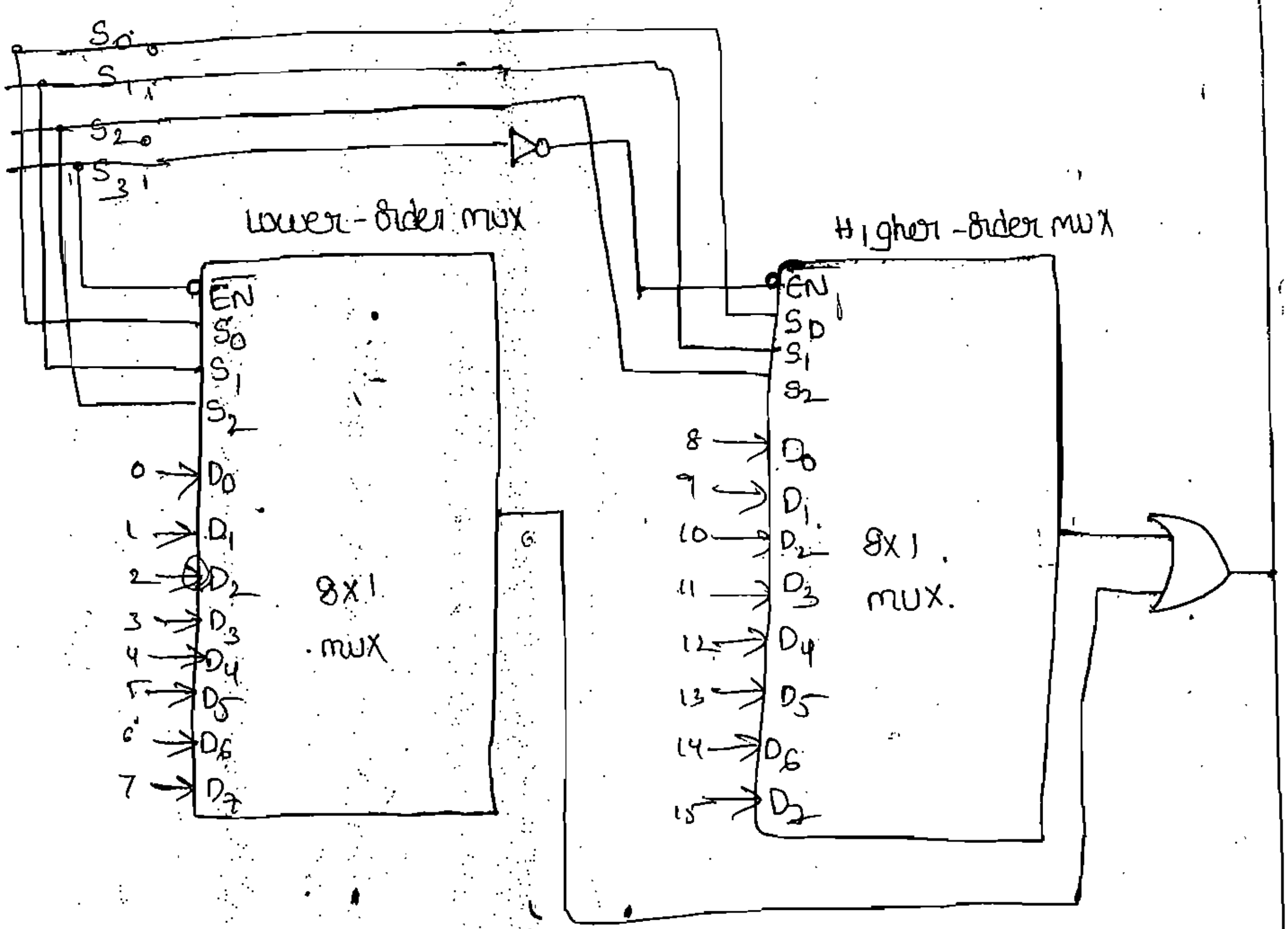
S_1	S_0	Z
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



$Z = \bar{S}_0\bar{S}_1D_0 + S_0\bar{S}_1D_1 + \bar{S}_0S_1D_2 + S_0S_1D_3$

The 16-input multiplexers from Two 8-input multiplexers:-

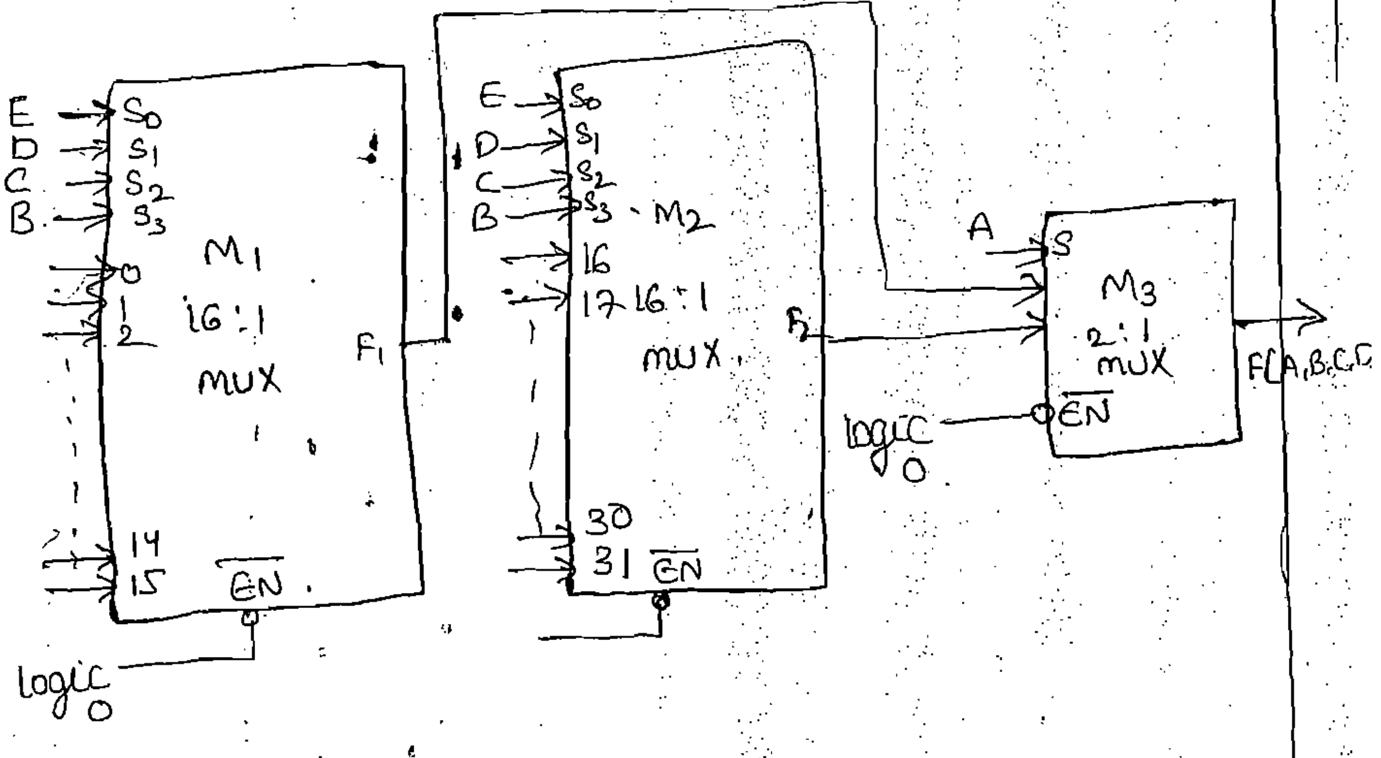
To use two 8-inputs multiplexers to get a 16-inputs multiplexers one OR gate and one inverter are also required. The four select inputs S_3, S_2, S_1 and S_0 and will select one of the 16 inputs to pass through to X. The S_3 input determines which multiplexer is enabled. when $S_3 = 0$, the left multiplexer is enabled and S_2, S_1 and S_0 inputs determine which of its data inputs will apper at its output and pass through the OR gate to X. when $S_3 = 1$, the right multiplexers is enabled and S_2, S_1 and S_0 inputs select one of its data inputs for passage to output X.



Logic diagram for cascading of two 8x1 mux to get 16x

Design of a 32*1 mux using Two 16*1 muxs and one 2*1 mux :-

To obtain a 32*1 mux using two 16*1 muxes and one 2*1 mux. A 32*1 mux has 32 data inputs. So it requires five data select lines. since a 16*1 mux has only four data select lines, the inputs B,C,D,E are connected to the data select lines of the both 16*1 muxes and the most significant input A is connected to the single data select line of the 2*1 mux. For the values of BCDE = 0000 to 1111, inputs 0 to 15 will appear at the input terminals 0 of the 2:1 mux through the output F_1 of the First 16*1 mux and inputs 16 to 31 will apper at the input terminal 1 of the 2*1 mux through the output F_2 of the second 16*1 mux. For A = 0, output F = F_1, for A = 1, output F = F_2.

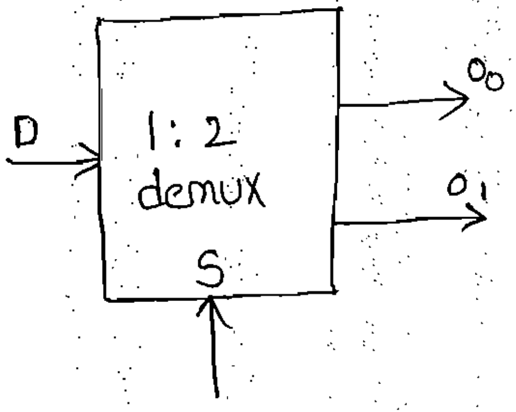


DEMULTIPLEXERS :-

A demultiplexer performs the reverse operation; it takes a single input and distributes it over several outputs. so a demultiplexer is also called as a "data Distributors" since it transmits the same data to different destinations. A demultiplexer is a 1-to-N device.

1-line to 2-line demultiplexer :-

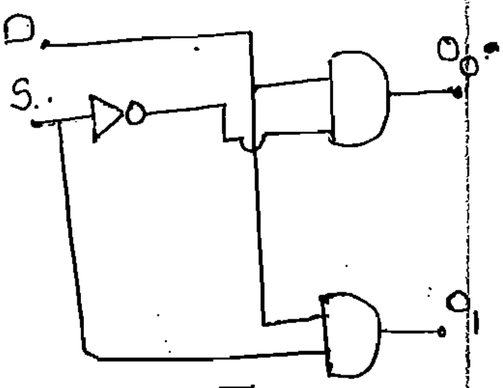
The input data line goes to all of the AND gates. The select line S enable only one gate at a time, and the data appearing on the input line will pass through the selected gate to the associated output lines.



Block diagram.

S	O_0	O_D
0	0	D
1	D	0

Truth table.



$O_0 = D\overline{S}$

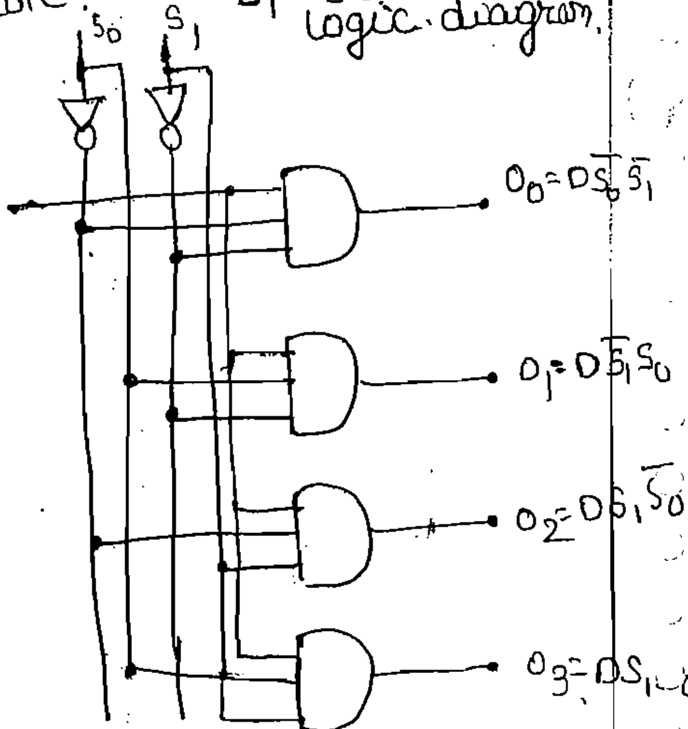
$O_1 = DS$

Logic diagram.

1-line to 4-line demultiplexer :-

S_1	S_0	D_3	O_2	O_1	O_0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0

Truth table.



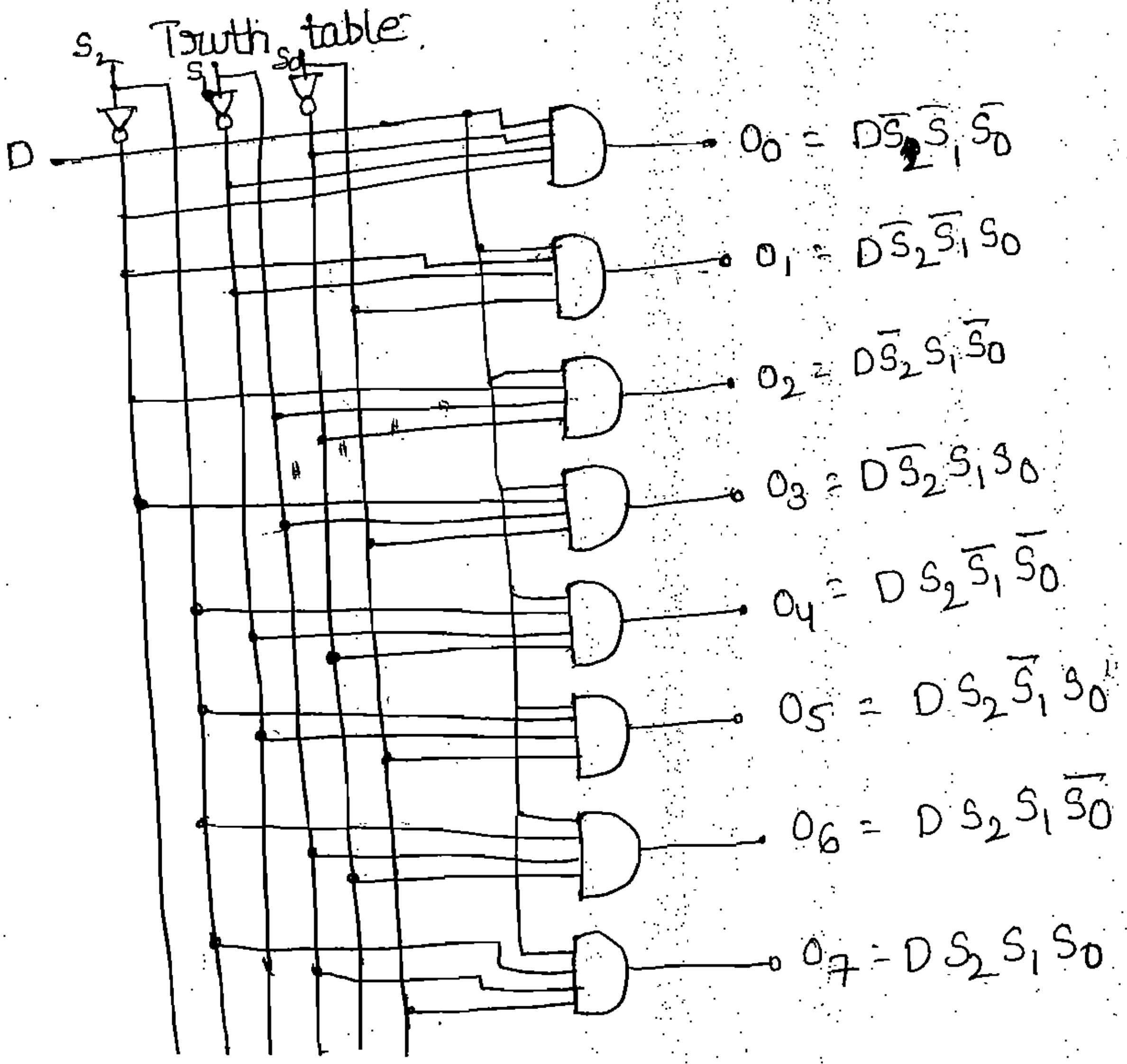
S_0
S_1
$O_0 = D\overline{S_0}\,\overline{S_1}$
$O_1 = D\overline{S_1}S_0$
$O_2 = DS_1\overline{S_0}$
$O_3 = DS_1S_0$

Logic diagram.

1-line to 8-line demultiplexer:-

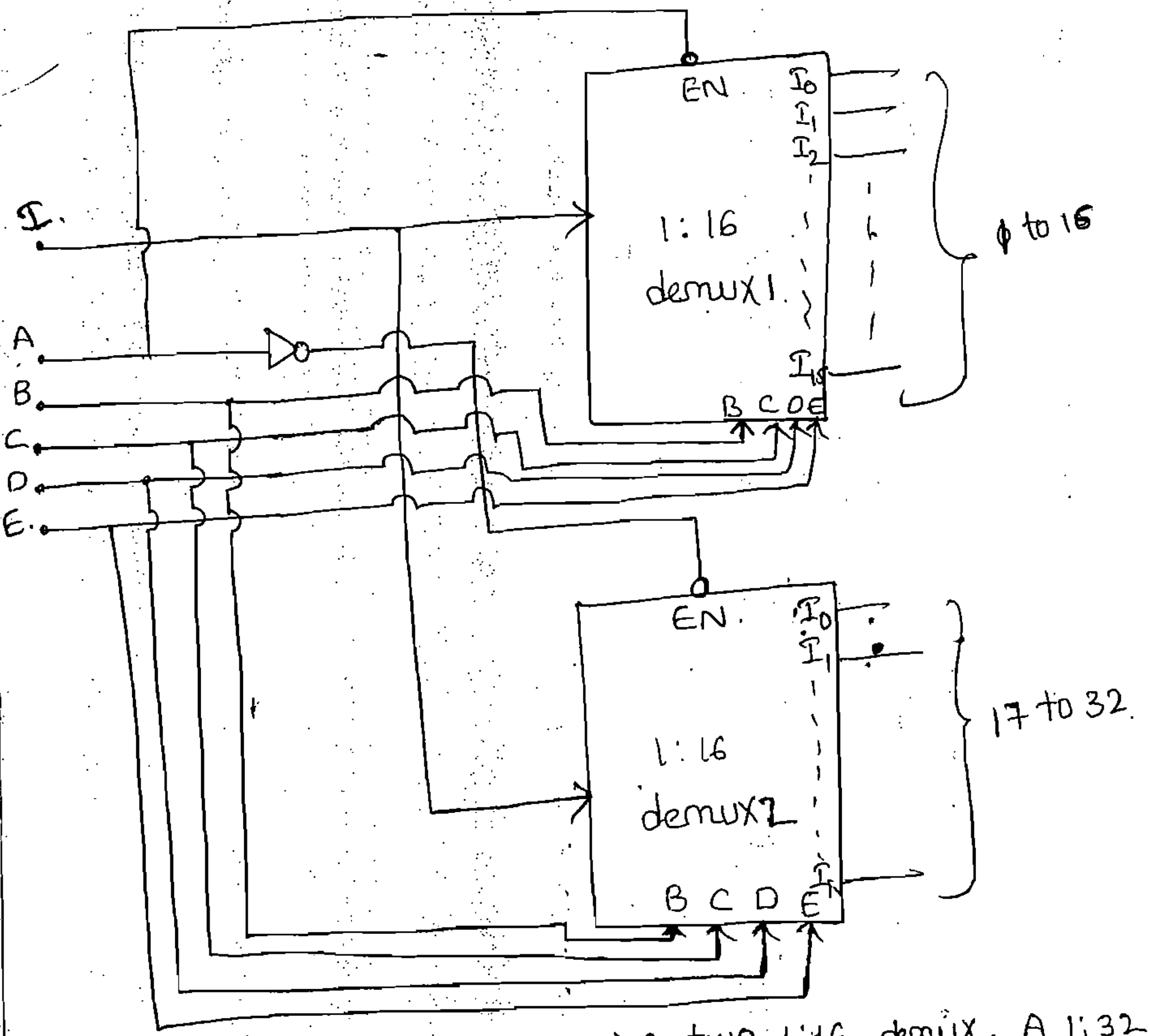
S_2	S_1	S_0	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	D
0	0	1	0	0	0	0	0	0	D	0
0	1	0	0	0	0	0	0	D	0	0
0	1	1	0	0	0	0	D	0	0	0
1	0	0	0	0	0	D	0	0	0	0
1	0	1	0	0	D	0	0	0	0	0
1	1	0	0	D	0	0	0	0	0	0
1	1	1	D	0	0	0	0	0	0	0

Truth table:



S_2
S_1
S_0
D
$O_0 = D\bar{S_2}\bar{S_1}\bar{S_0}$
$O_1 = D\bar{S_2}\bar{S_1}S_0$
$O_2 = D\bar{S_2}S_1\bar{S_0}$
$O_3 = D\bar{S_2}S_1S_0$
$O_4 = DS_2\bar{S_1}\bar{S_0}$
$O_5 = DS_2\bar{S_1}S_0$
$O_6 = DS_2S_1\bar{S_0}$
$O_7 = DS_2S_1S_0$

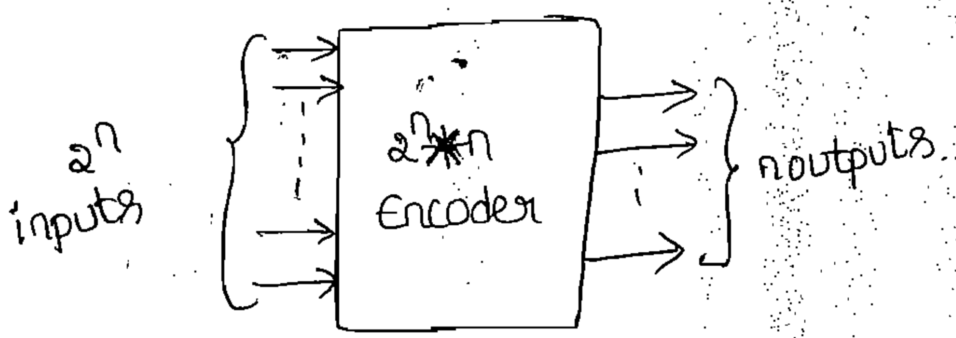
design of 1:32 demux using Two 1:16 demux.



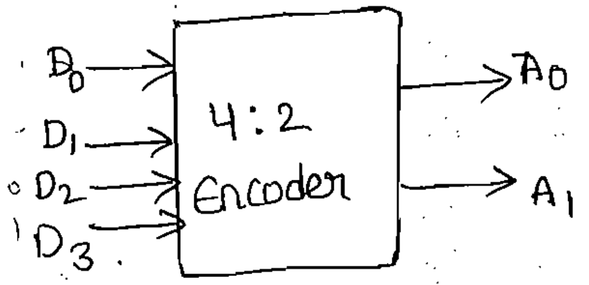
To obtain 1:32 demux using two 1:16 demux. A 1:32 demux has 32 data outputs. So it requires five data select lines. Since 1:16 demux has only four select inputs. the inputs B, C, D, E are connected to the data select lines of both the 1:16 demuxes and the most significant input A is connected to the single data select line of the both 1:16 demuxs $\overline{EN}$ input. 1 to 16 will apper in the first demux when (A=0). 17 to 32 will apper in the second demux when A=1.

ENCoders :-

An encoder is a device whose inputs are decimal digits and/or alphabetic characters and whose outputs are the coded representation of those inputs, An encoder is a device which converts familiar numbers or symbols into coded format. The Encoder has 2^n inputs and n outputs.



4bit - Encoder :-



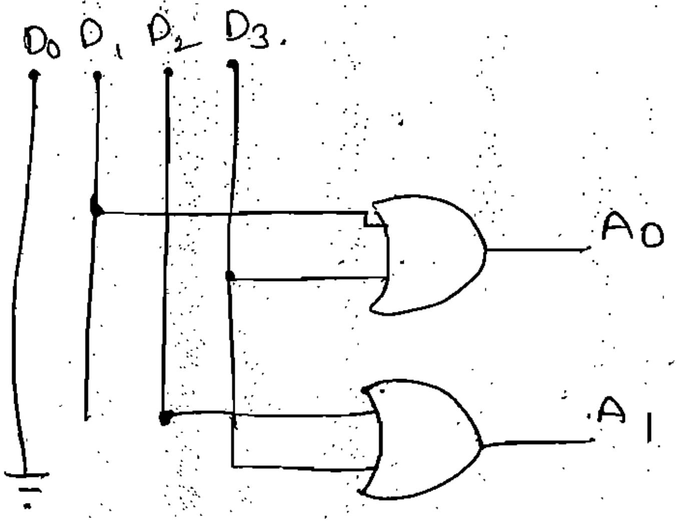
Block diagram.

inputs	outputs A_1	A_0
D_0	0	0
D_1	0	1
D_2	1	0
D_3	1	1

Truth table.

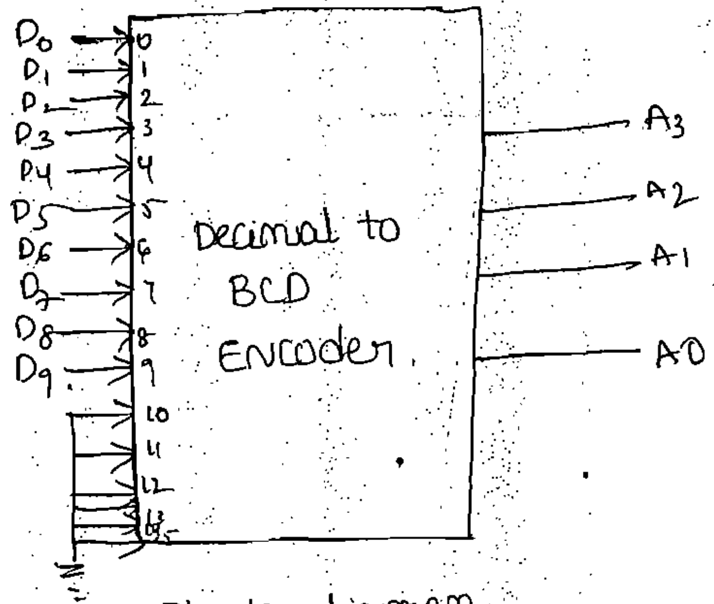
$$A_1 = D_2 + D_3$$

$$A_0 = D_1 + D_3 .$$



Decimal to BCD Encoder :-

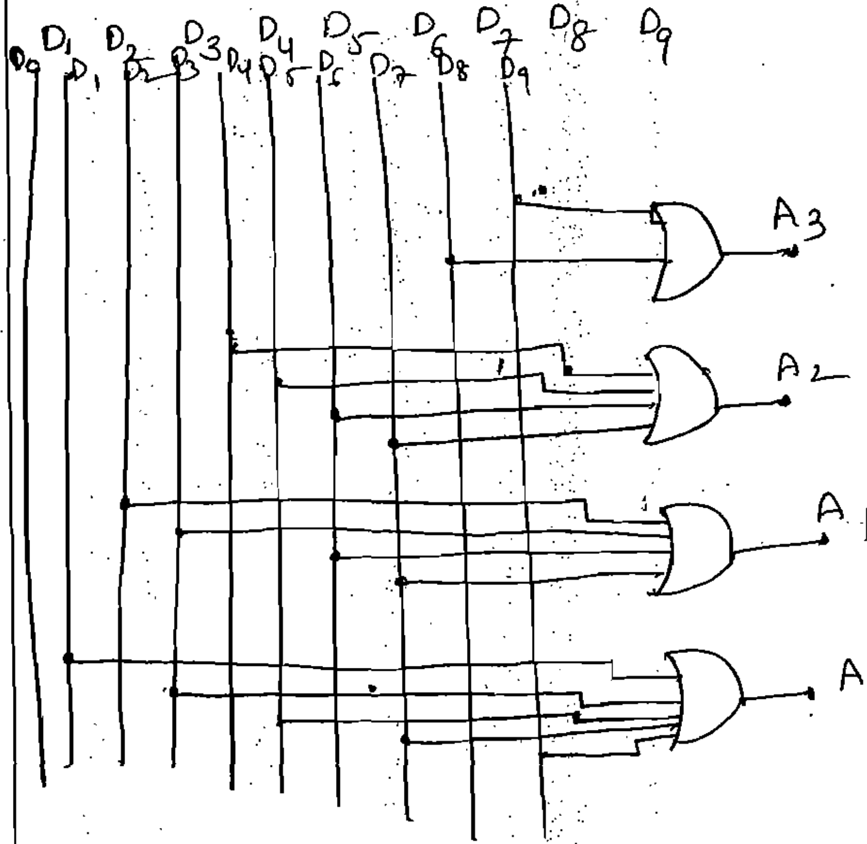
In this type of encoder has 10 inputs - one for each decimal digit, and 4 outputs corresponding to the BCD code.



Block diagram.

inputs.		Binary output			
Decimal		A_3	A_2	A_1	A_0
D_0	0	0	0	0	0
D_1	1	0	0	0	1
D_2	2	0	0	1	0
D_3	3	0	0	1	1
D_4	4	0	1	0	0
D_5	5	0	1	0	1
D_6	6	0	1	1	0
D_7	7	0	1	1	1
D_8	8	1	0	0	0
D_9	9	1	0	0	1

Truth table.



$A_0 = D_1 + D_3 + D_5 + D_7 + D_9$

$A_1 = D_2 + D_3 + D_6 + D_7$

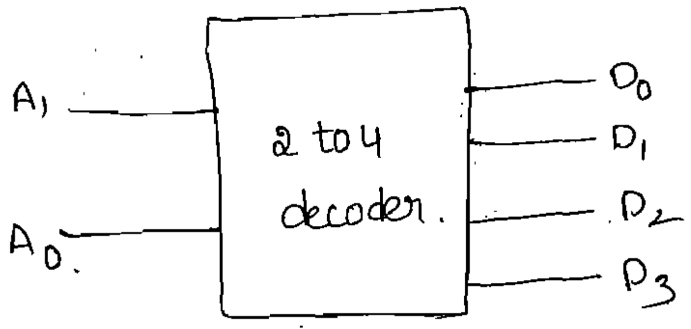
$A_2 = D_4 + D_5 + D_6 + D_7$

$A_3 = D_8 + D_9$.

Decoders :-

A decoder is a logic circuit that converts an N-bit binary input code into 2^n output lines such that only one output line is activated for each one of the possible combinations of inputs. For each of these input combinations, only one of the output will be activated (high), all the other outputs will remain inactive (low). Some decoders are designed to produce active low output, while all the other outputs remain high.

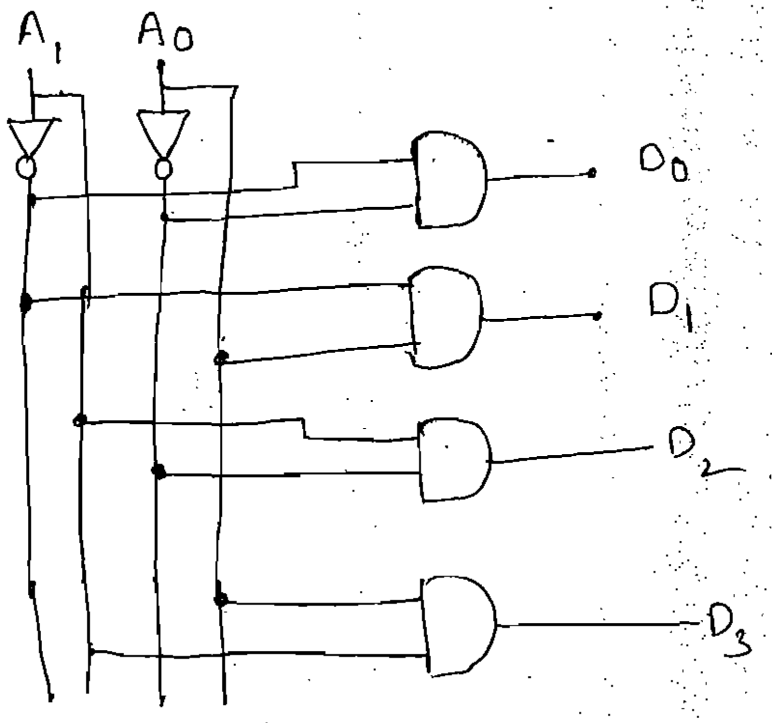
2-4 line decoder :-



Block diagram.

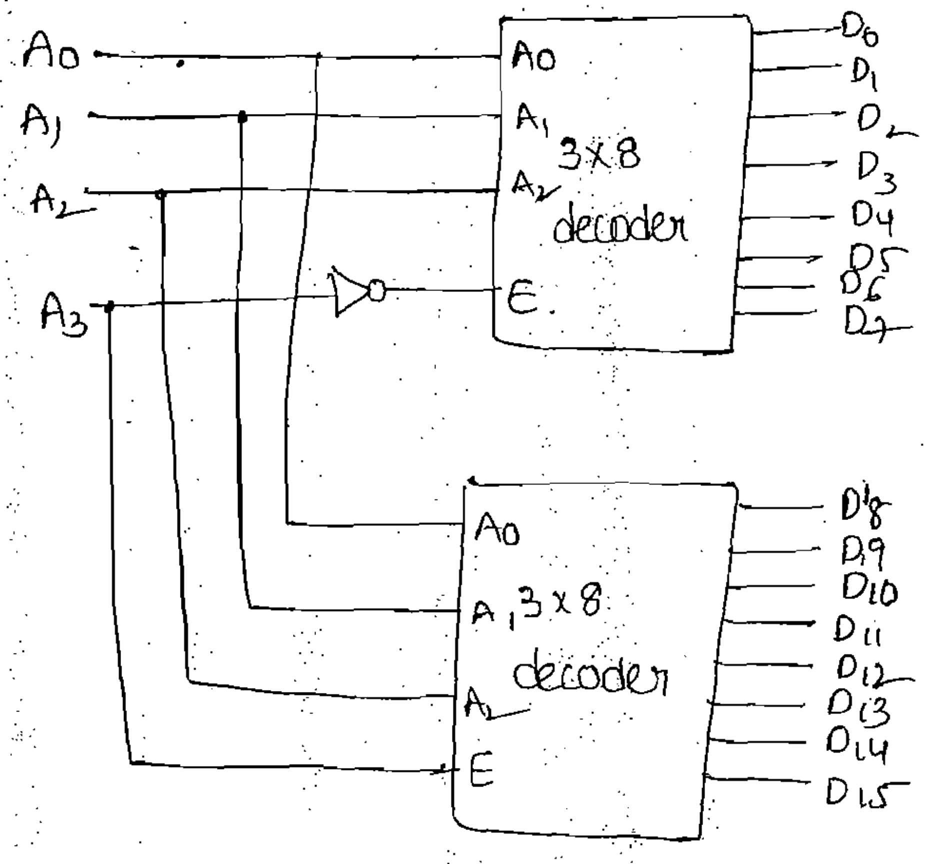
A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Truth table.



4-to-16 decoder from two 3-to-8 decoders :-

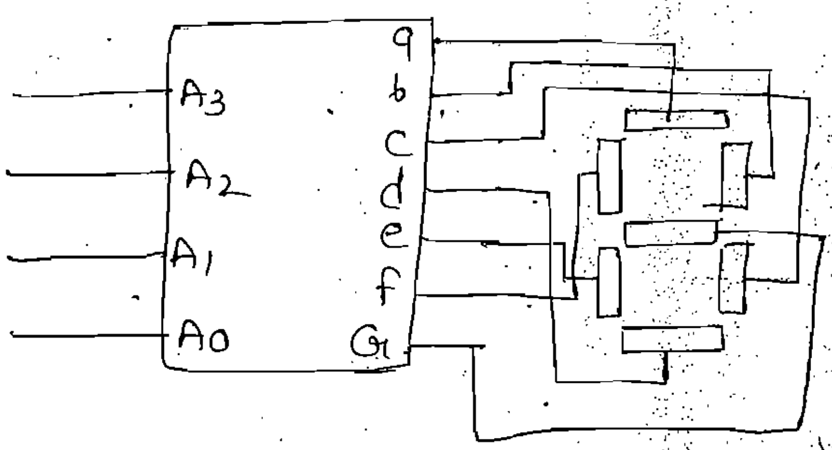
Decoders with enable inputs can be connected together to form a larger decoder. To obtain a 4-to-16 decoder it requires two 3-to-8 decoders. The most significant input bit A_3 is connected through an inverter to $\overline{E}$ on the upper decoder and directly to E on the lower decoder. Thus A_3 is low, the upper decoder is enabled and the lower decoder is disabled. The bottom decoder outputs all 0's, and top 8 outputs generates minterms. when A_3 is high, the lower decoder is enabled and the upper decoder is disabled. The bottom decoder outputs generates minterms 1000 to 1111 while the outputs of the top decoder are all 0's.

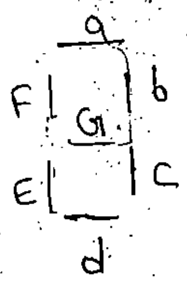


logic diagram

Seven segment Decoders :-

This type of decoders accepts the BCD code and provides outputs to energize seven segment display devices in order to produce a decimal read out. Each segment is made up of a material that emits light when current is passed through it. The most commonly used materials include LEDS, incandescent filaments and LCDS.





Block diagram.

Decimal digit	BCD input				Seven segment code						
	A_3	A_2	A_1	A_0	a	b	c	d	e	f	G
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	0	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	1	1	1

$a = \Sigma m(0,2,3,5,6,7,8,9) + d(10,11,12,13,14,15)$.

$b = \Sigma m(0,1,2,3,4,7,8,9) + d(10,11,12,13,14,15)$

$c = \Sigma m(0,1,3,4,5,6,7,8,9) + d(10,11,12,13,14,15)$.

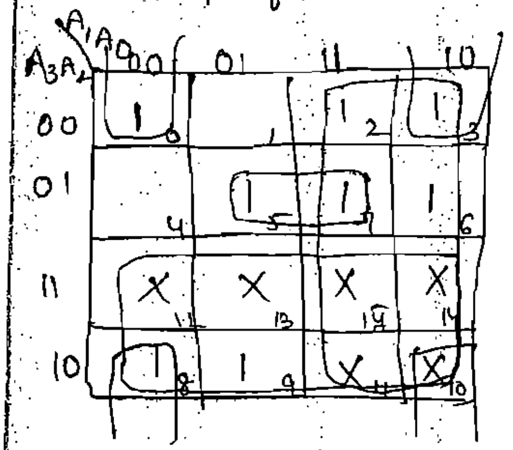
$d = \Sigma m(0,2,3,5,6,8,9) + d(10,11,12,13,14,15)$.

$e = \Sigma m(0,2,6,8) + d(10,11,12,13,14,15)$.

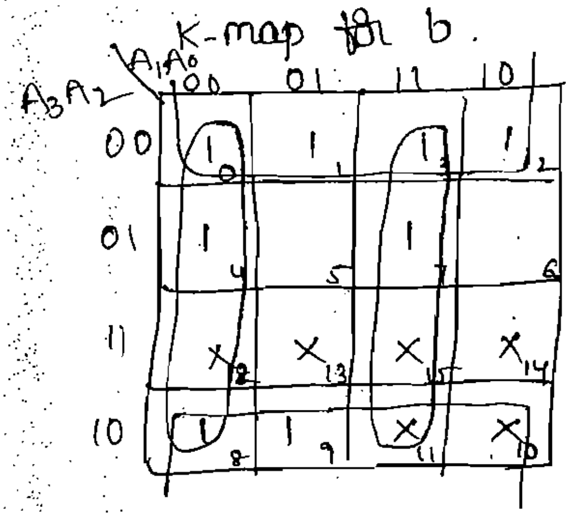
$f = \Sigma m(0,4,5,6,8,9) + d(10,11,12,13,14,15)$.

$g = \Sigma m(2,3,4,5,6,8,9) + d(10,11,12,13,14,15)$.

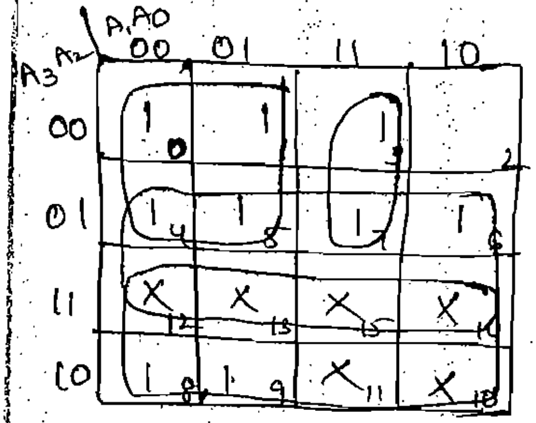
K-map for a.



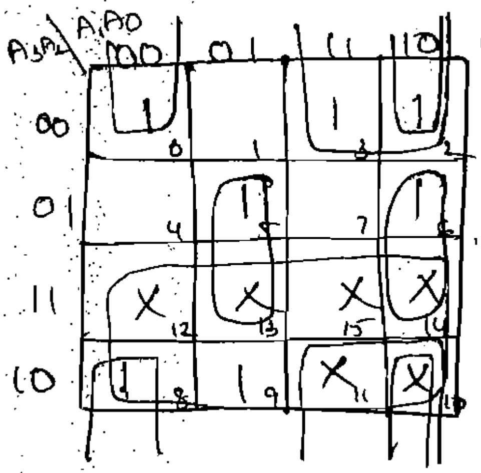
K-map for b.



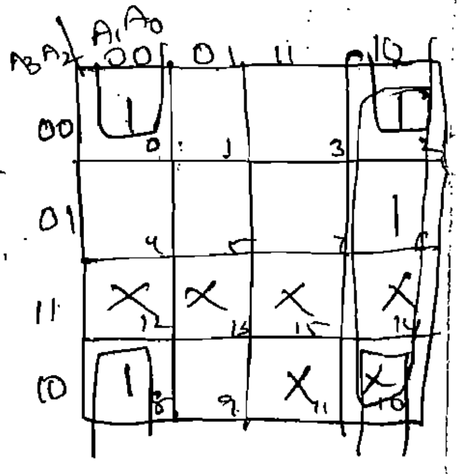
$a = A_1 + A_3 + \overline{A_2}\,\overline{A_0} + \overline{A_3}A_2A_0$ $b = \overline{A_2} + A_1\overline{A_0} + A_1A_0$



K-map for c



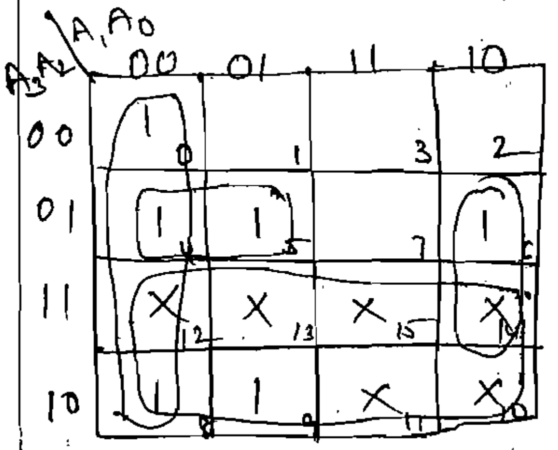
K-map for d



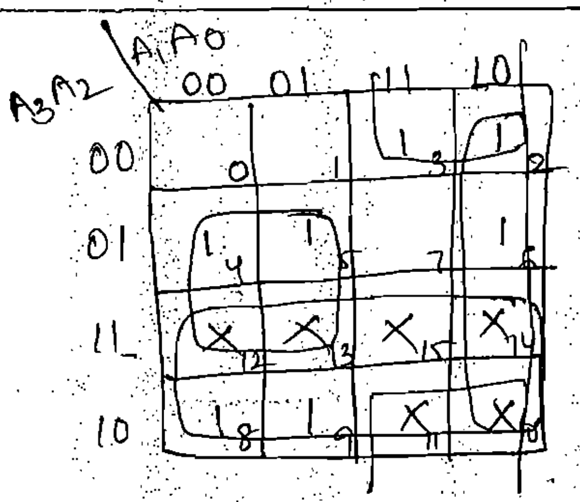
K-map for e

$C = A_2 + A_3 + \overline{A_3}\,\overline{A_1} + \overline{A_3}A_1A_0$ $d = A_3 + \overline{A_2}A_1 + A_2\overline{A_1}A_0 + A_2A_1\overline{A_0} + \overline{A_2}\,\overline{A_0}$

$e = A_1\overline{A_0} + \overline{A_2}\,\overline{A_0}$



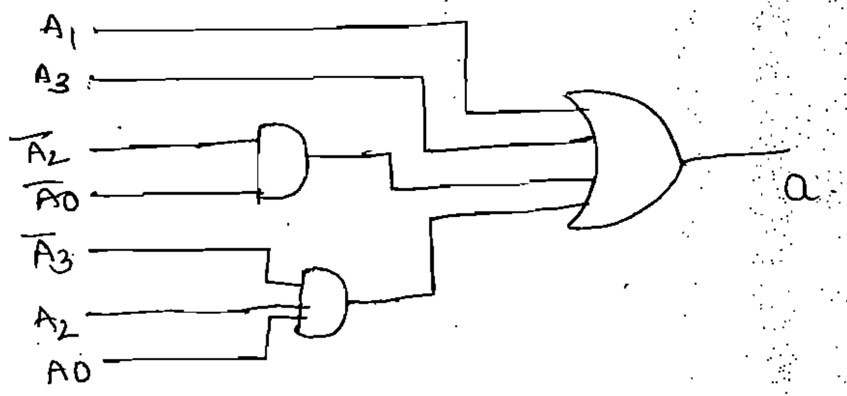
K-map for f.

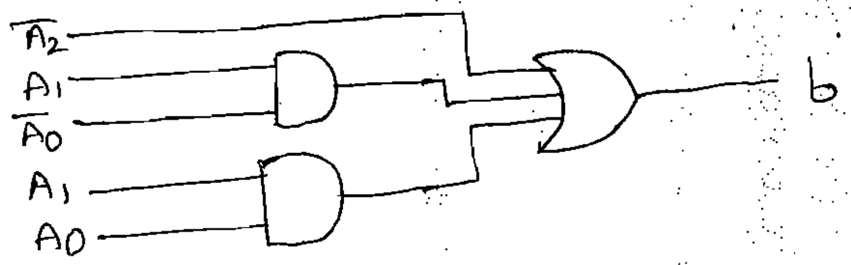


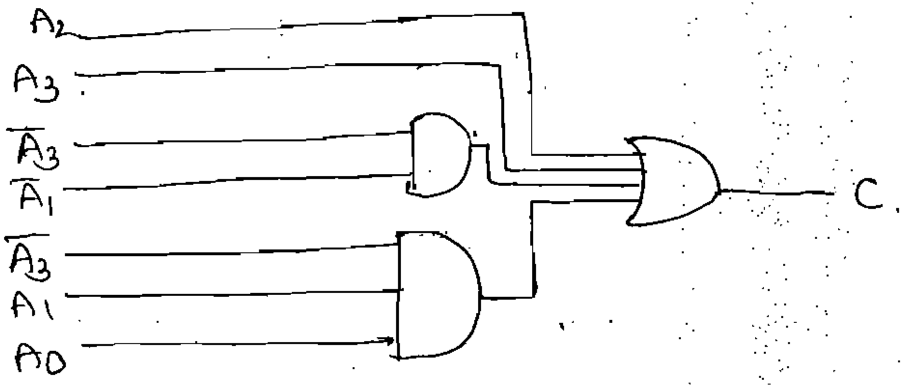
K-map for G.

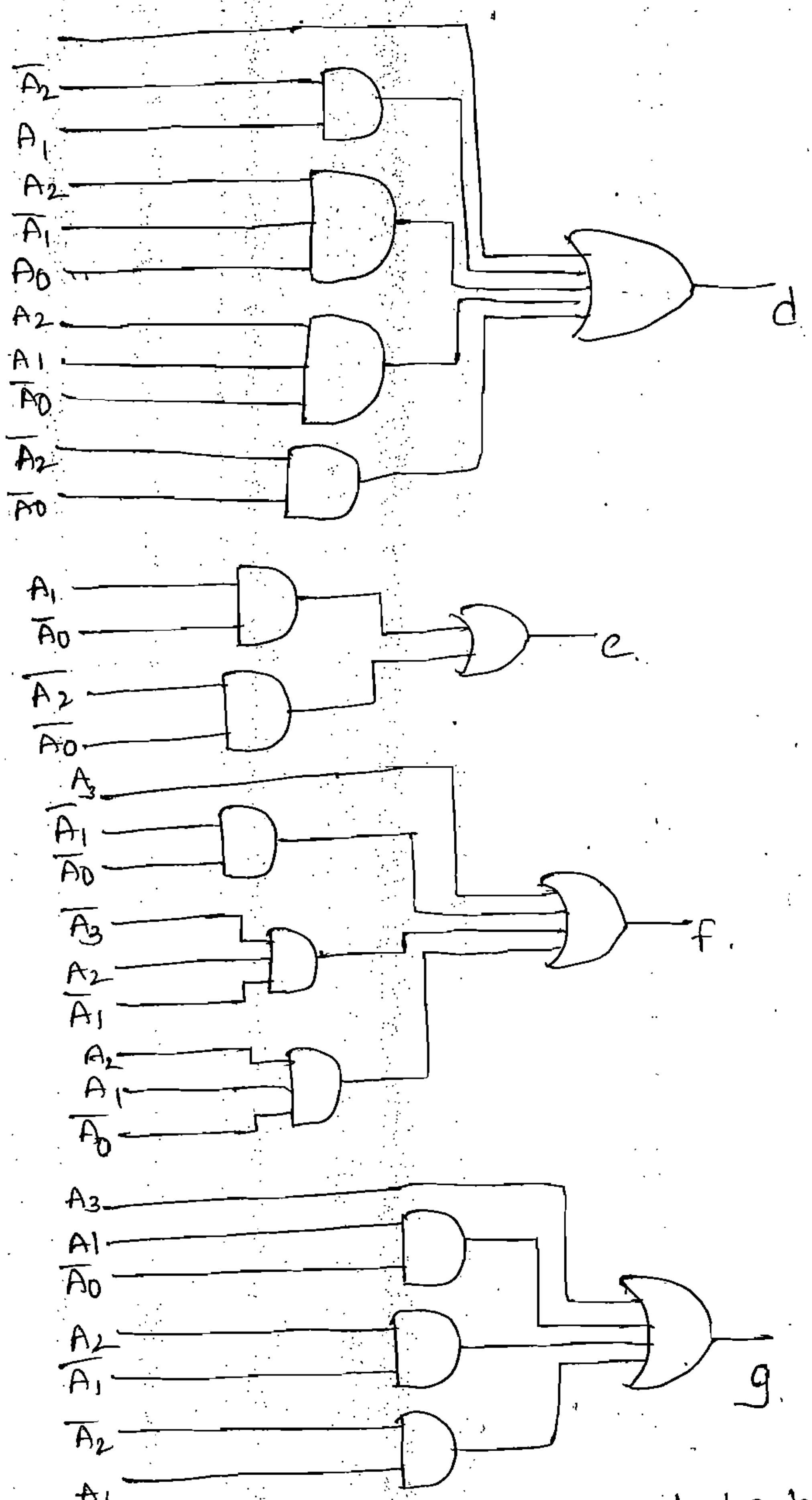
$$f = \bar{A}_1\bar{A}_0 + A_3 + \bar{A}_3A_2\bar{A}_1 + A_2A_1\bar{A}_0$$

$$G = A_3 + A_1\bar{A}_0 + A_2\bar{A}_1 + \bar{A}_2A_1$$





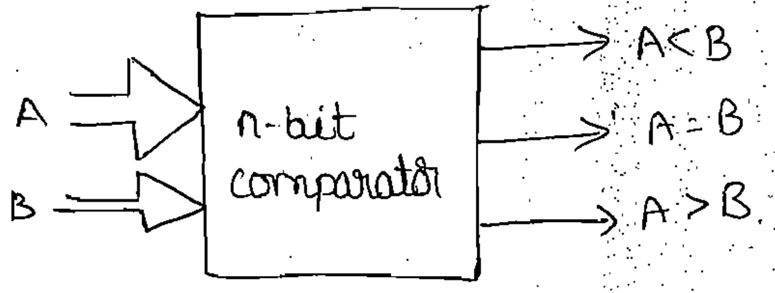




diagrams for seven segment display.

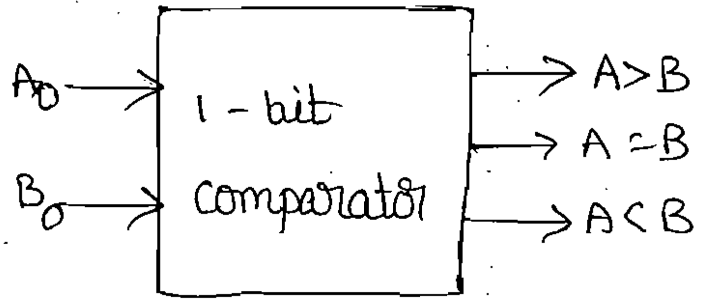
Comparator :-

The comparator is a combinational logic circuit It compares the magnitude of two n-bit numbers and provides the relative result as the output The block diagram of an n-bit digital comparator has 2 inputs and three outputs. A and B are the n-bit inputs. The comparator outputs are A>B, A=B and A<B. Depending upon the result of comparision, one of these outputs will be high.



1-bit comparator :-

The one bit comparator is a combinational logic circuit with two inputs A and B and three outputs namely A<B, A=B, A<B.



Block diagram

Inputs		outputs		
A_0	B_0	A<B	A=B	A>B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

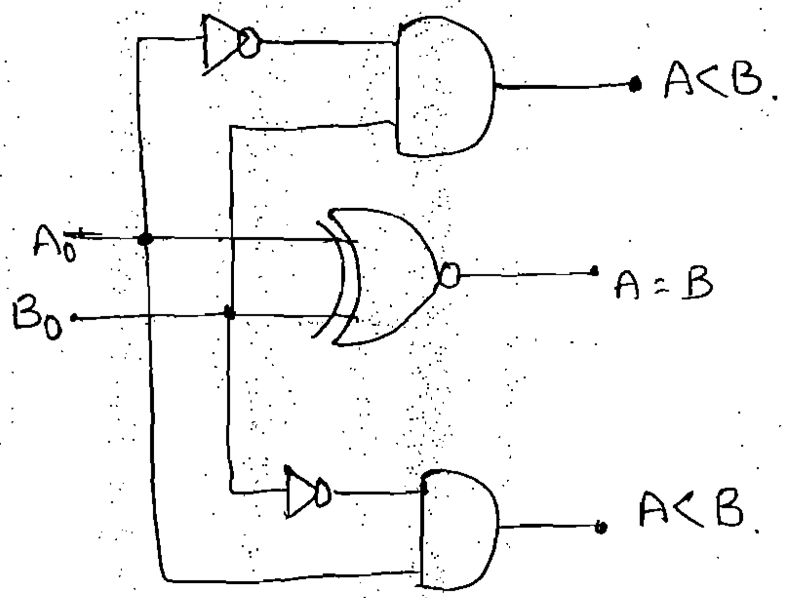
Truth table

In truth table

$A=B$: $\overline{A_0}\,\overline{B_0} + A_0 B_0 = A_0 \odot B_0$

$A<B$: $\overline{A_0} B_0$

$A>B$: $A_0 \overline{B_0}$



2-bit comparator :-

The logic for a 2-bit comparator. let the two 2-bit numbers be $A = A_1A_0$ and $B = B_1B_0$.

→ if $A_1=1$ and $B_1=0$, then $A>B$ or

→ if A_1 and B_1 are equal and $A_0=1$ and $B_0=0$ then $A>B$.

$A>B$: $A_1\overline{B_1} + (A_1 \odot B_1) A_0 \overline{B_0}$

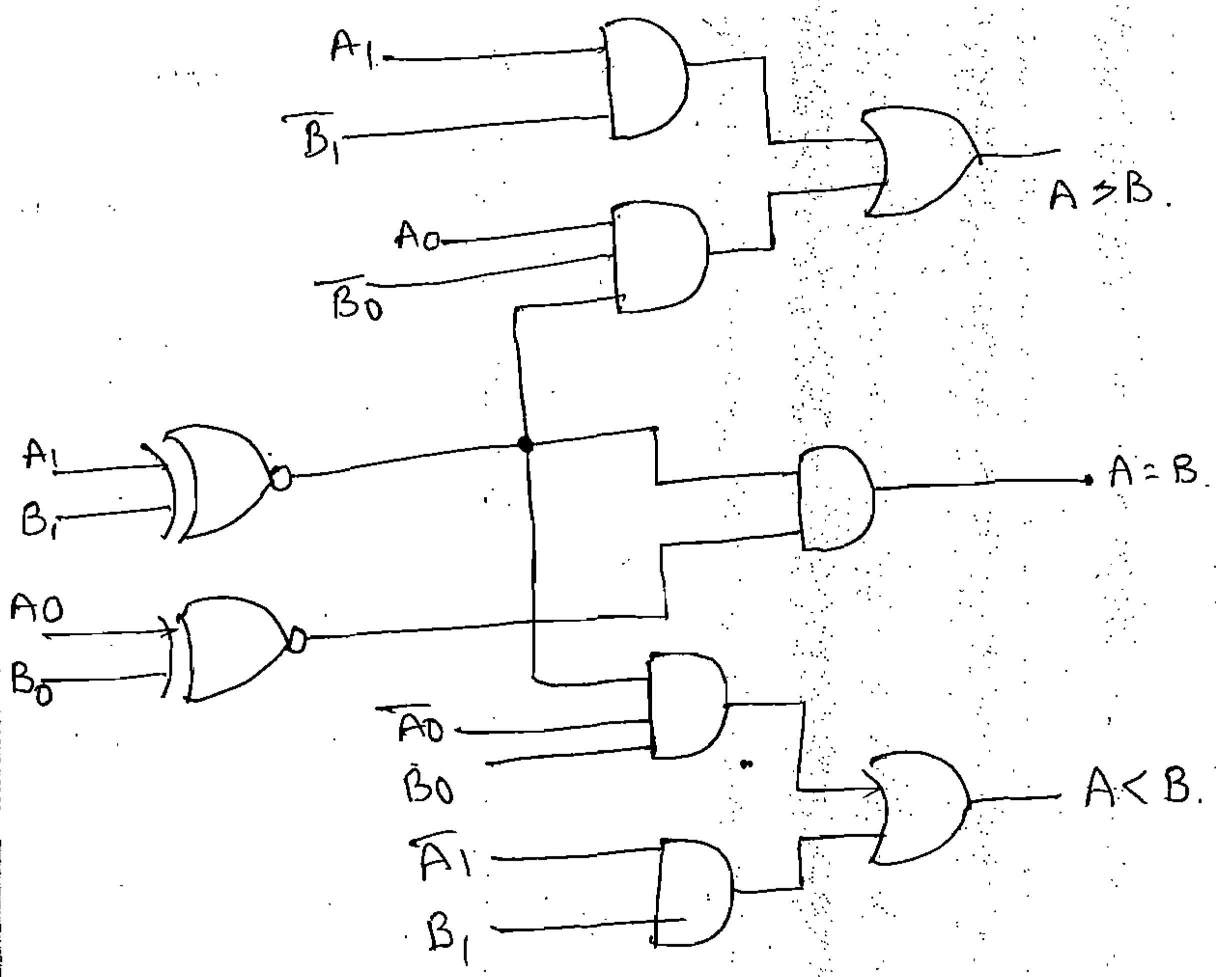
→ if $B_1=1$ and $A_1=0$ then $A<B$ or

→ if B_1 and A_1 are equal and $B_0=1$ and $A_0=0$ then $A<B$

$A<B$: $\overline{A_1}B_1 + (A_1 \odot B_1) \overline{A_0} B_0$

→ if A_1 and B_1 are equal and if A_0 and B_0 are equal then $A = B$

$A = B$: $(A_1 \odot B_1)(A_0 \odot B_0)$



4-bit comparator :-

The logic for a 4-bit comparator. let the four bit numbers will be $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$.

→ if $A_3 = 1$ and $B_3 = 0$, then $A > B$ or

→ if A_3 and B_3 are equal, and if $A_2 = 1$ and $B_2 = 0$, or

→ if A_3 and B_3 are equal, A_2 and B_2 are equal, and if A_1 ~~or~~ $= 1$ and $B_1 = 0$, or

→ if A_3 and B_3 are equal, and if A_2 and B_2 are equal, and if A_1 and B_1 are equal, and if $A_0 = 1$ and $B_0 = 0$.

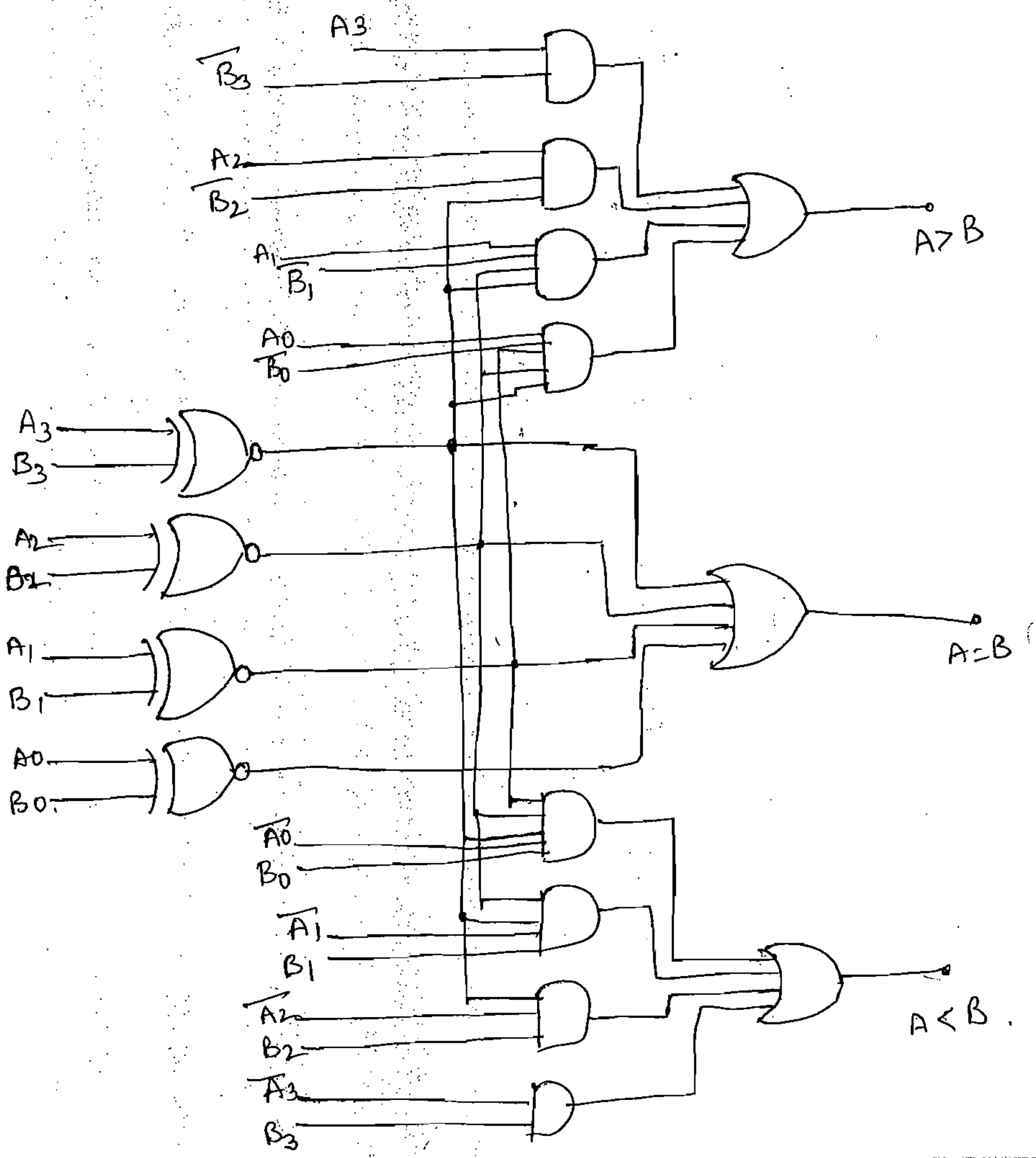
$$(A>B) = A_3\overline{B_3} + (A_3 \odot B_3)\, A_2\overline{B_2} + (A_3 \odot B_3)(A_2 \odot B_2)\, A_1\overline{B_1}$$

$$+ (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)\, A_0\overline{B_0}$$

$$(A<B) = \overline{A_3}B_3 + (A_3 \odot B_3)\,\overline{A_2}B_2 + (A_3 \odot B_3)(A_2 \odot B_2)\,\overline{A_1}B_1$$

$$+ (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)\,\overline{A_0}B_0$$

$$A=B : (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0).$$



logic diagram for 4-bit comparator.

priority Encoders :-

It is possible that two or more inputs are active at a time. To overcome this, priority encoders are used. A priority encoder is a logic circuit that responds to just one input in accordance with some priority system, among all those that may be simultaneously high. The most common priority system is based on the relative magnitudes of the inputs.

In some practical applications, priority encoders may have several inputs that are routinely high at the same time, and the principal function of the encoder in those cases is to select the input with the highest priority.

4-input priority encoder :-

In 4-input priority encoder in addition to the outputs A and B, the circuit has a third output designed by V. This is a valid bit indicator that is set to 1 when one or more inputs are equal to 1. If all inputs are 0, there is no vaild input and V is equal to 0. The other two outputs are not inspected when V equals 0 and are specified as don't care conditions.

Truth table

D_0	D_1	D_2	D_3	A	B	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	

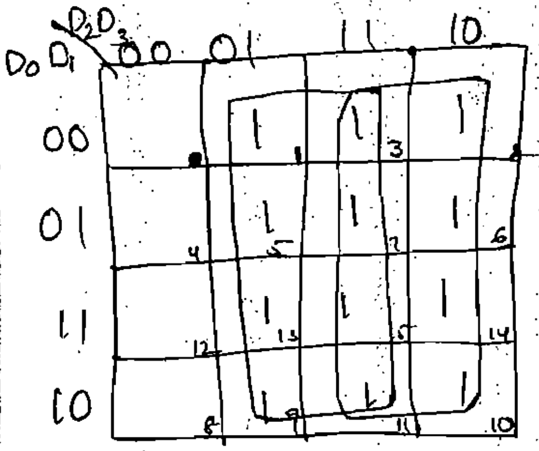
$V = D_0 + D_1 + D_2 + D_3$.

According to the truth table. the outputs A and B are.

$A = \Sigma m(1,2,3,5,6,7,9,10,11,13,14,15)$

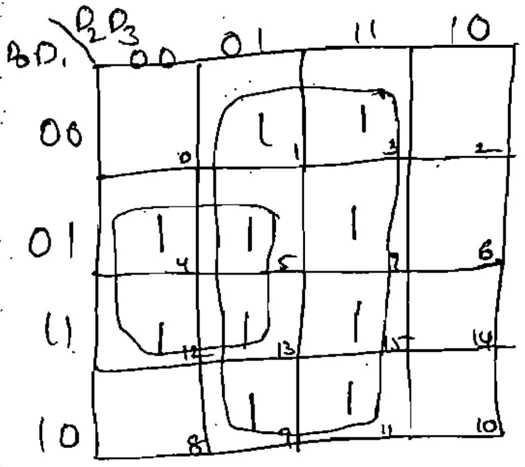
$B = \Sigma m(1,3,4,5,7,9,11,12,13,15)$

K-map for A

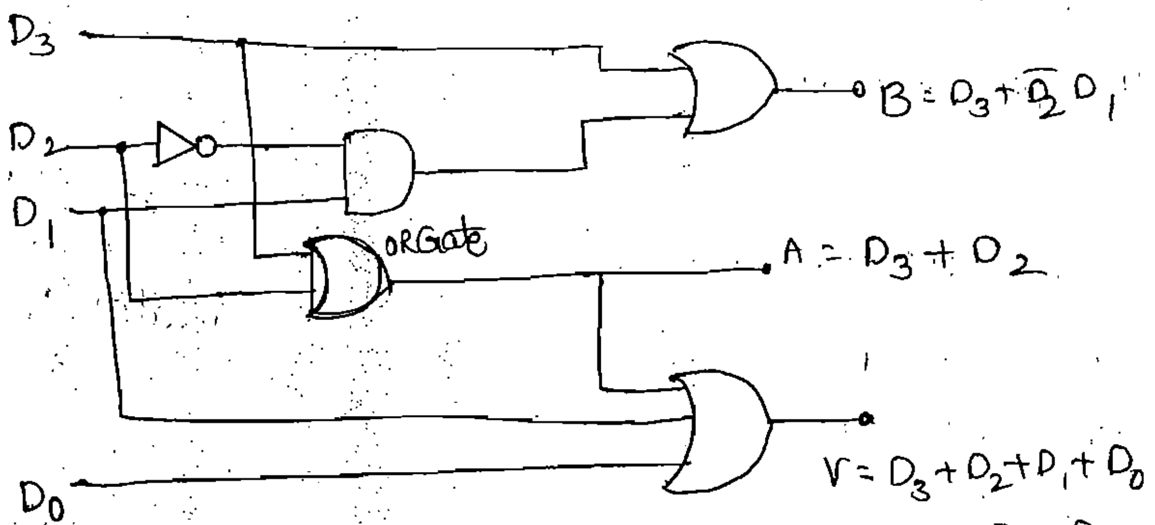


$A = D_3 + D_2$

K-map for B.



$B = D_3 + \overline{D_2} D_1$



logic diagram for
4-bit priority Encoder.

PROGRAMMABLE LOGIC DEVICES

→ logic designers have a wide range of standard IC'S available to them with numerous logic functions and logic circuit arrangements on a chip. In addition, these ICs are available from many manufactures and at a reasonably low cost.

→ PLD is an IC that contains large number of gates, flip-flops and registers that are interconnected on chip. This IC is said to be programmable because the specific function IC is determined by Interconnecting required contacts.

Basically, there are three types of programmable device which are.

→ Read only memory. (ROM)

→ programmable logic array (PLA)

→ programmiable Array logic (PAL)

READ ONLY memory :-

→ The read only memory is a type of Semiconductor memory that is designed to hold data that is either permanent or will not change frequently.

→ During operation, no new data can be written into a ROM, but data can be read from ROM. the process of Entering data is called programming or buring-in the ROM.

→ Some ROM'S cannot have their data changes once they have been programmed others can be erased and reprogrammed as often as desired.

Types of ROM'S :-

1. Masked memory ROM.
2. programmable Read only memory (PROM)
3. Erasable programmable Read only memory (EPROM)
4. Electrically Erasable programmable Read only memory (EEPROM)

Masked memory (Rom) :-

→ cannot be reprogrammed
→ Nonvolatile, retain data even when power is turn off.
→ cheaper than programmable devices.
→ useful for fixed programme instructions.

PROM

→ programmed by blowing built-in fuses.
→ can not be reprogrammed
→ Non volatile.
→ useful for small volume data storing
→ user programmable

EPROM :-

→ Erasable, programmable ROM

→ programmed by storing charge on insulated gates.

→ Erasable with ultraviolet light

→ non - volatile.

EEPROM :-

→ programmed by storing charges on insulated gates.

→ Non volatile

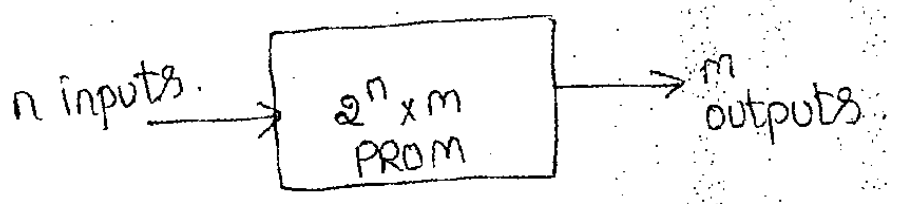
Programmable ROM :-

→ It includes both the decoder and the OR Gates with a single IC package. The following figures shows the block diagram and logic construction using 16X2 ROM.

→ It consists of n input lines and m output lines.

→ Each bit combination of the input variables is called an address.

→ Each bit combination that comes out of the output lines is called a word.



Block diagram.

An integrated circuit with programmable gates divided into an AND array and an OR array provide an AND-OR sum of products implementati

EPROM :-

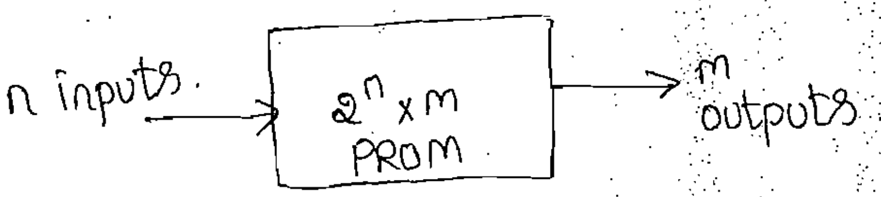
→ Erasable, programmable ROM
→ programmed by storing charge on insulated gates.
→ Erasable with ultraviolet light
→ non - volatile.

EEPROM :-

→ programmed by storing charges on insulated gates.
→ Non volatile

Programmable ROM :-

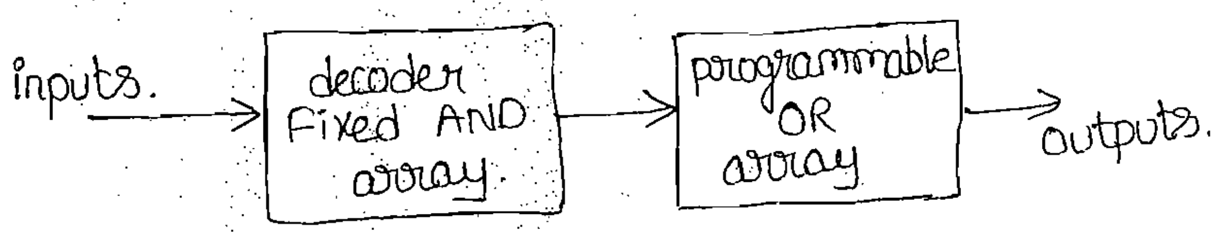
→ It includes both the decoder and the OR Gates with a single IC package. The following figures shows the block diagram and logic construction using 16X2 ROM.
→ It consists of n input lines and m output lines.
→ Each bit combination of the input variables is called an address.
→ Each bit combination that comes out of the output lines is called a word.

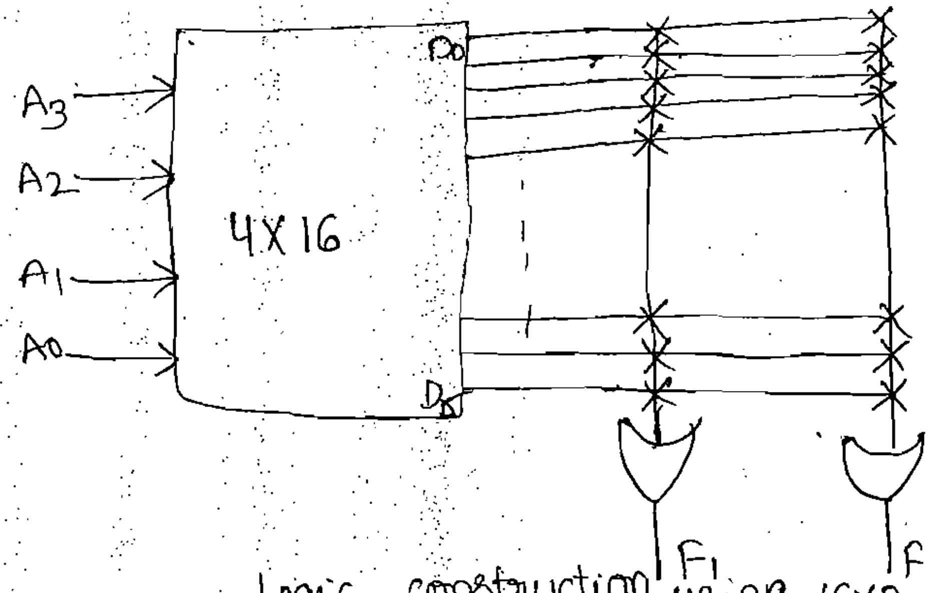


Block diagram.

An integrated circuit with programmable gates divided into an AND array and an OR array to provide an AND-OR sum of products implementation.

The programmable read-only memory (PROM) has a fixed AND array constructed as a decoder and a programmable OR array. The AND gates are programmed to provide the product terms for the Boolean functions, which are logically summed in each OR Gate.





Logic construction using 16X2 ROM.

→ Implement full-adder using PROM.

The number of inputs variables of a full adder are 3. The possible number of combinations are 8. So we need 3X8 decoder. The number of outputs of full adder are 2. They are sum and carry.

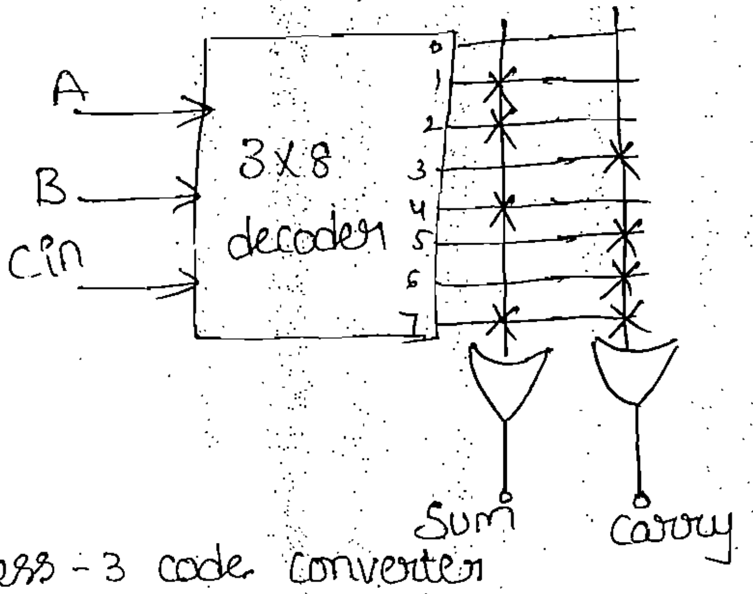
Truth table

A	B	Cin	S	C
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

output expression for sum & carry is $Sum = \Sigma m(1,2,4,7)$

$carry = \Sigma m(3,5,6,7)$

<u>logic diagram</u>



→ Design BCD to Excess-3 code converter using ~~P~~ PROM.

Step:- 1 – Truth table.

BCD				EXCESS-3			
B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
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

Step 2 :-

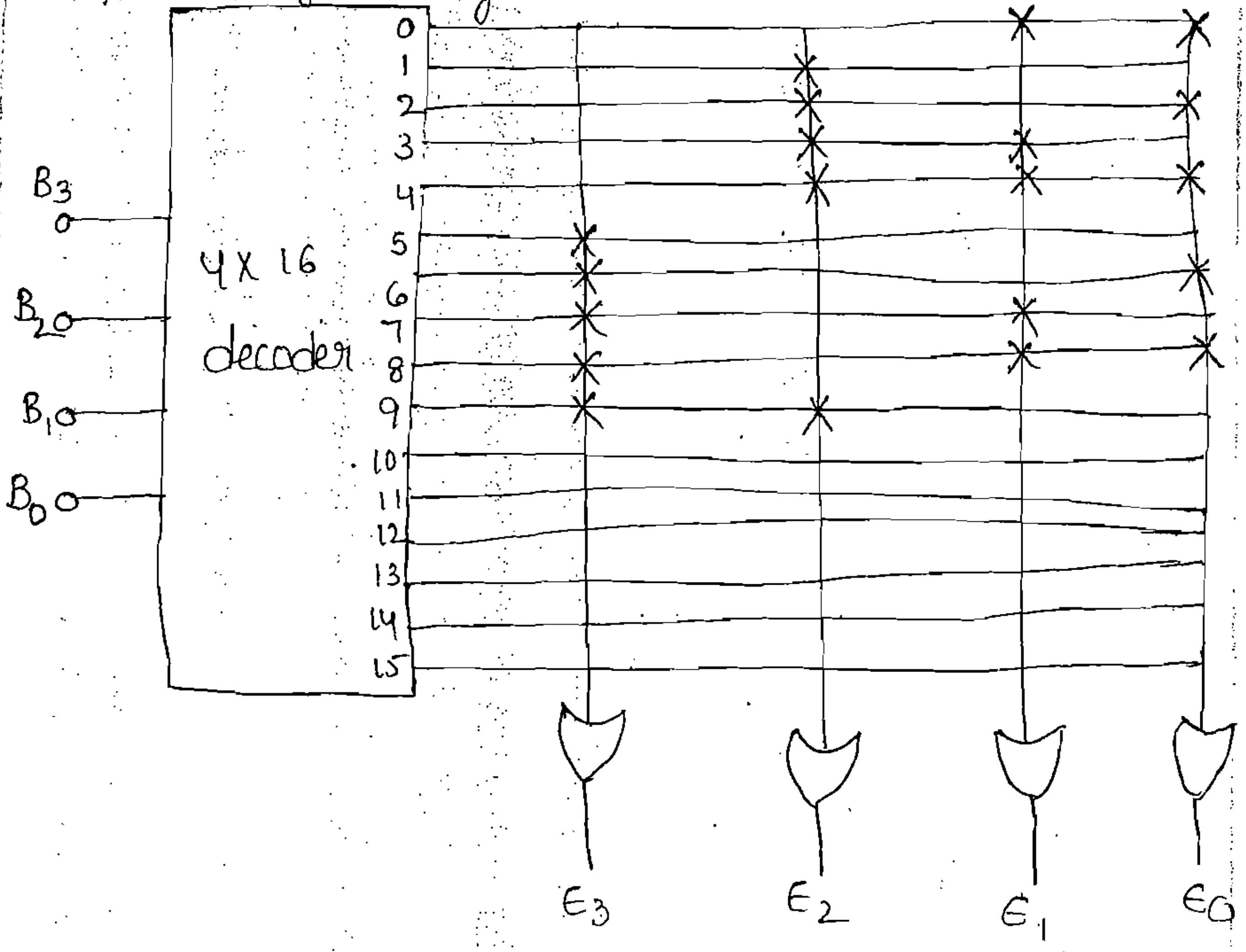
$E_0(B_3 B_2 B_1 B_0) = \Sigma m(0,2,4,6,8)$

$E_1(B_3 B_2 B_1 B_0) = \Sigma m(0,3,4,7,8)$

$E_2(B_3 B_2 B_1 B_0) = \Sigma m(1,2,3,4,9)$

$E_3(B_3 B_2 B_1 B_0) = \Sigma m(5,6,7,8,9)$

Step 3 :- logic diagram.



→ Implement the following Boolean expression using PROM.

$f(A,B,C) = \bar{A}B + C + BC$.

first given expression converted into a standard SOP form.

$f(A,B,C) = \bar{A}B + C + BC$

$= \overline{A}B(C+\overline{C}) + C(A+\overline{A})(B+\overline{B}) + BC(A+\overline{A})$

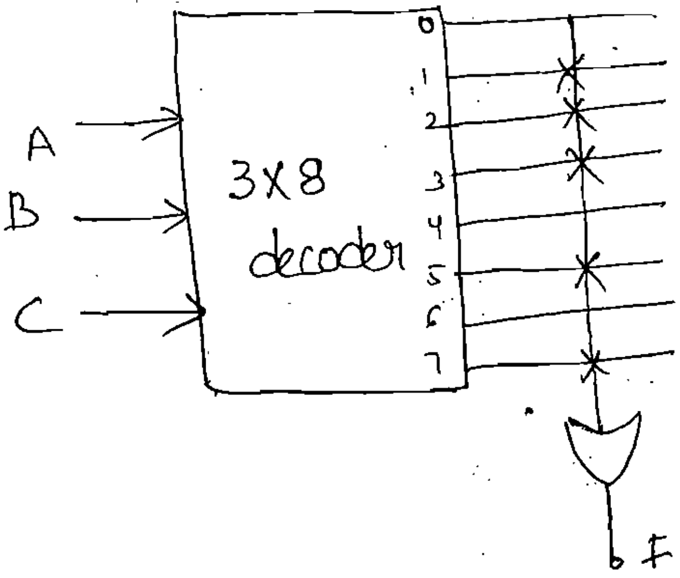
$= \underbrace{\overline{A}BC}_{1} + \overline{A}B\overline{C} + \underbrace{ABC}_{2} + A\overline{B}C + \overline{A}BC + \overline{A}\,\overline{B}C + \underbrace{ABC}_{2} + \underbrace{\overline{A}BC}_{1}$

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

011, 010, 111, 101, 011, 001

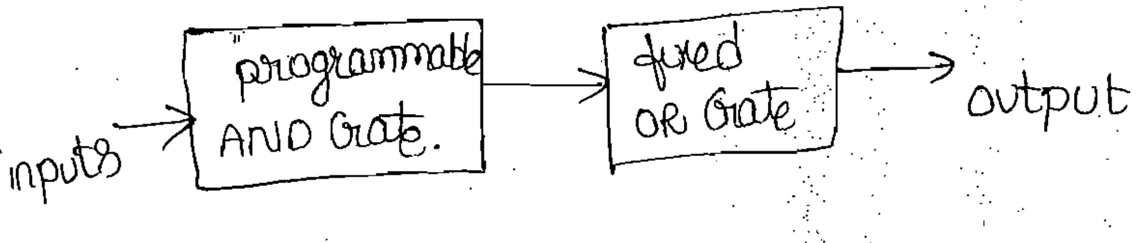
$f(A,B,C) = \Sigma m(1,2,3,5,7)$

Logic diagram :-



PAL :- programmable Array logic :-

The programmable Array logic is a programmable device with a fixed OR array and a programmable AND array because, only the AND gates are programmmble.

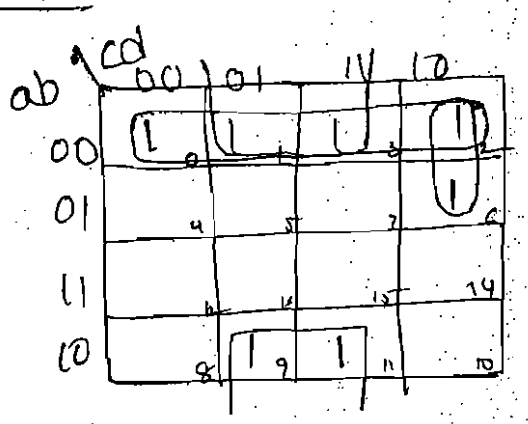


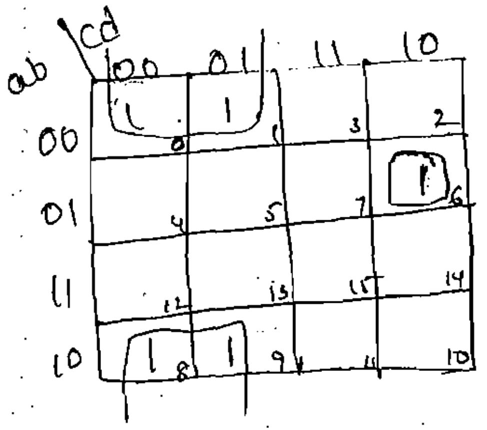
→ Implement the following functions using PAL.

$F_1(a,b,c,d) = \Sigma m(0,1,2,3,6,9,11)$

$F_2(a,b,c,d) = \Sigma m(0,1,6,8,9)$

Step 1 :- K-map simplification for F_1 and F_2



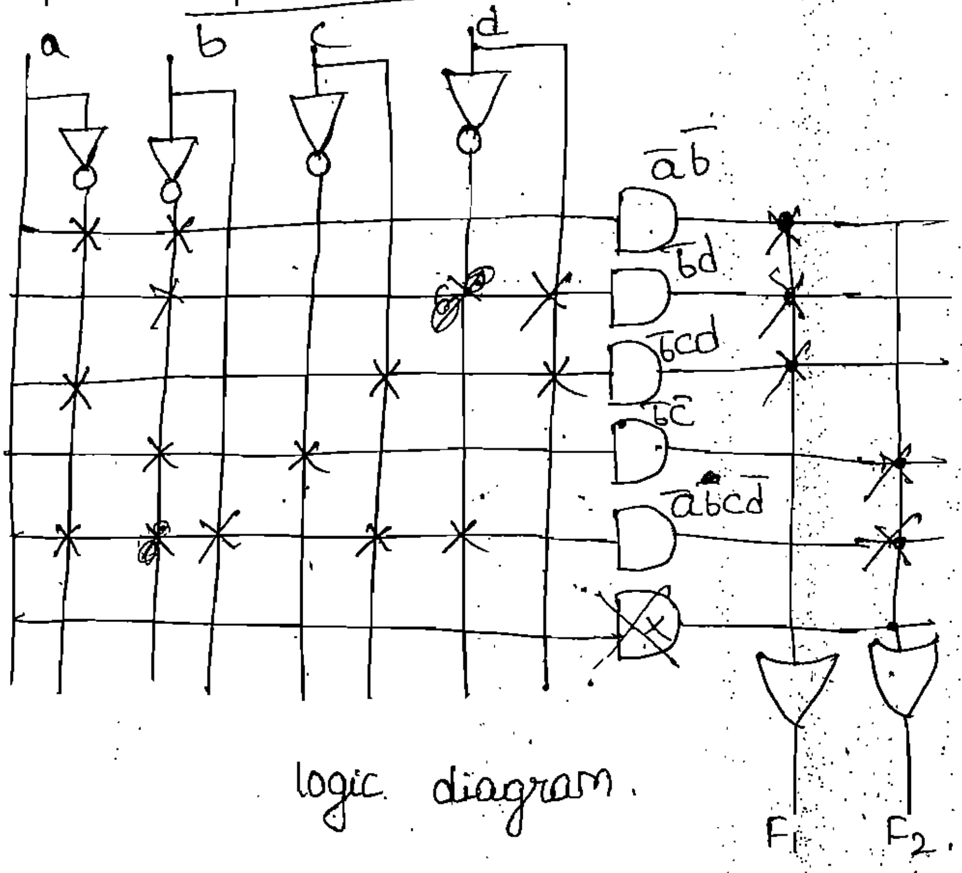


$\bar{a}\bar{b}+\bar{a}cd+\bar{b}d$ $\qquad$ $\bar{b}\bar{c}+\bar{a}bc\bar{d}$

Step 2 :- PAL programming table.

product terms	AND gate inputs a	b	c	d	outputs
1	0	0	–	–	$F_1 = \bar{a}\bar{b}+\bar{a}cd + \bar{b}d$
2	0	–	1	1	
3	–	0	–	1	
4	–	0	0	–	$F_2 = \bar{b}\bar{c} + \bar{a}bc\bar{d}$
5	0	1	1	0	
6	–	–	–	–	

step 3 :- implementation :-



logic diagram.

→ Implement 4-bit BCD to XS-3 code conversion using PAL.

Step :- 1

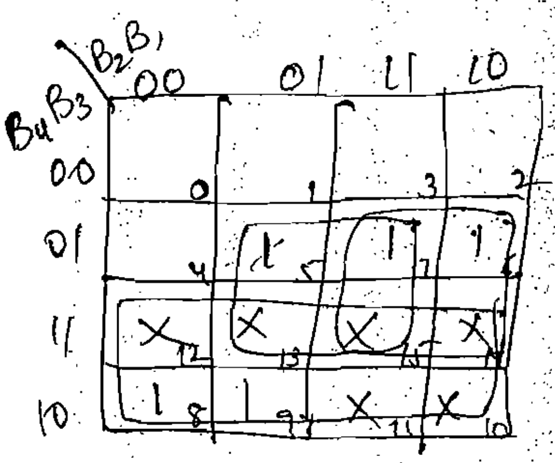
B_4	B_3	B_2	B_1	X_4	X_3	X_2	X_1
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

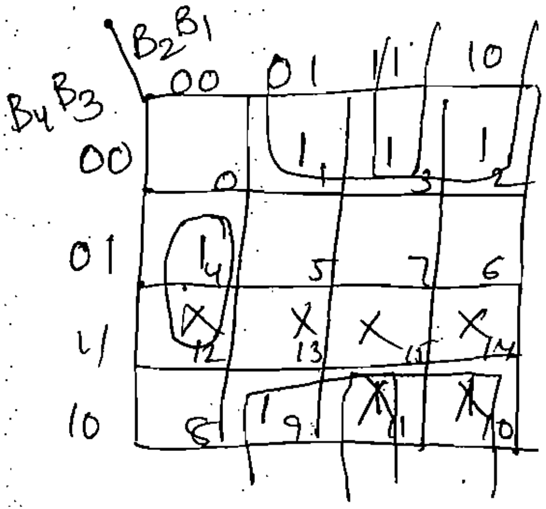
$X_4 = \Sigma m(5,6,7,8,9) + d(10,11,12,13,14,15)$

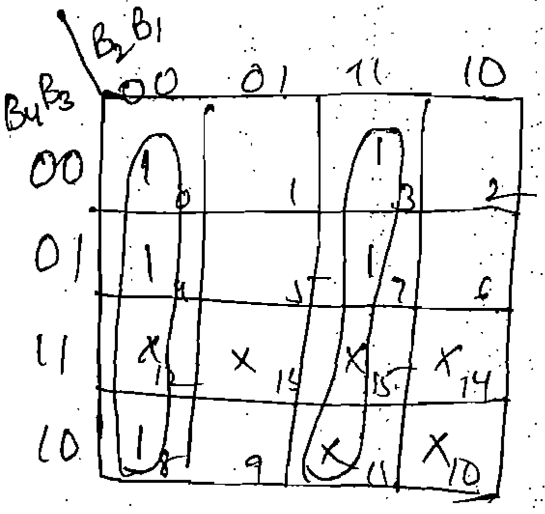
$X_3 = \Sigma m(1,2,3,4,9) + d(10,11,12,13,14,15)$

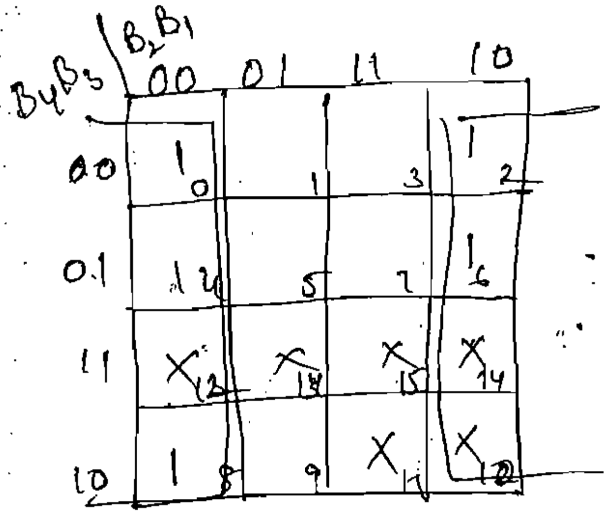
$X_2 = \Sigma m(0,3,4,7,8) + d(10,11,12,13,14,15)$

$X_1 = \Sigma m(0,2,4,6,8) + d(10,11,12,13,14,15)$









$X_4 = B_4 + B_3B_2 + B_3B_1$

$X_3 = B_3\overline{B_2}\,\overline{B_1} + \overline{B_3}B_1 + \overline{B_3}B_2$

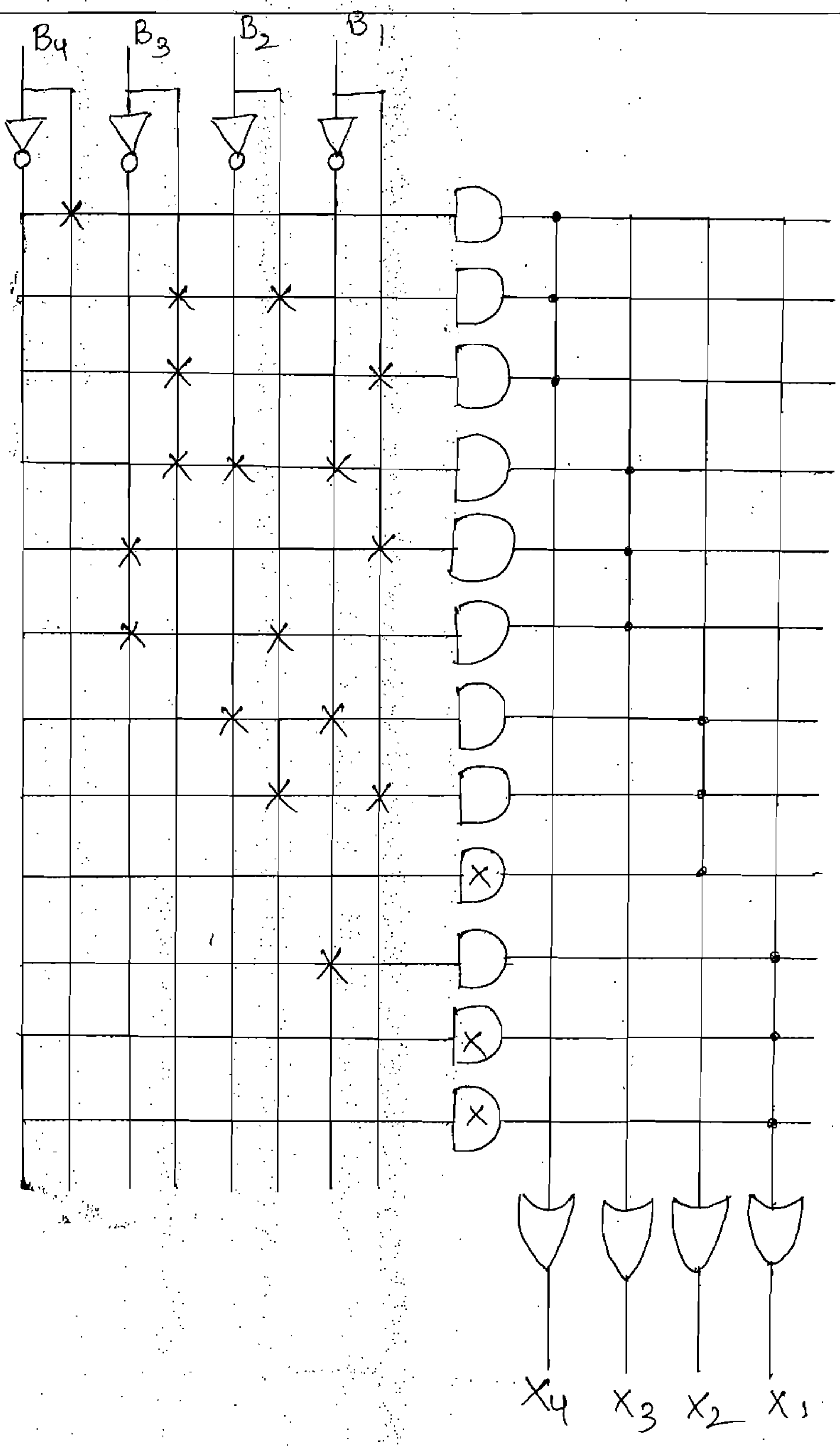
$X_2 = \overline{B_2}\,\overline{B_1} + B_2B_1$

$X_1 = \overline{B_1}$

step 2 :- programming table.

product terms.	AND Gate inputs. B4	B3	B2	B1	outputs.
1	1	–	–	–	$X_4 = B_4 + B_3B_2 + B_3B_1$
2	–	1	1	–	
3	–	1	–	1	
4	–	1	0	0	$X_3 = B_3\overline{B_2}\,\overline{B_1} + \overline{B_3}B_1 + \overline{B_3}B_2$
5	–	0	–	1	
6	–	0	1	–	
7	–	–	0	0	$X_2 = \overline{B_2}\,\overline{B_1} + B_2B_1$
8	–	–	1	1	
9	–	–	–	–	
10	–	–	–	0	$X_1 = \overline{B_1}$
11	–	–	–	–	
12	–	–	–	–	

Step 3 :- logic diagram.



logic diagram.

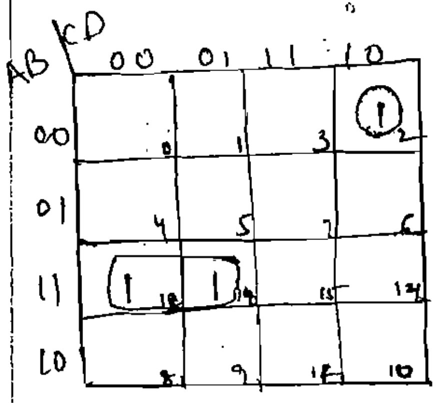
Implement the following boolean functions using PAL with four inputs and 3-wide AND-OR structure. Also write the PAL programming table.

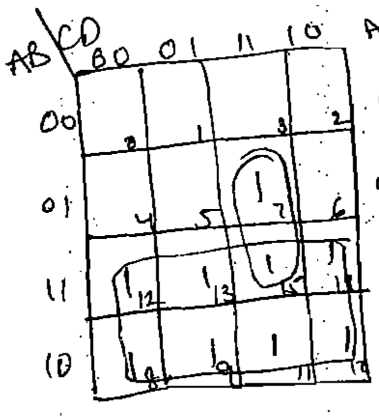
$F_1(A,B,C,D) = \Sigma m(2,12,13)$

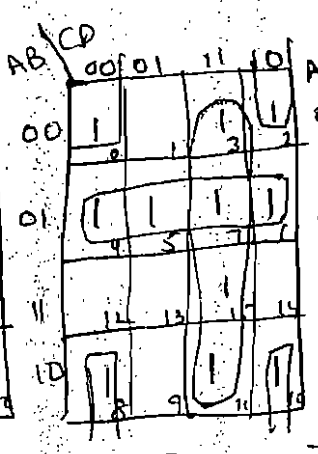
$F_2(A,B,C,D) = \Sigma m(7,8,9,10,11,12,13,14,15)$

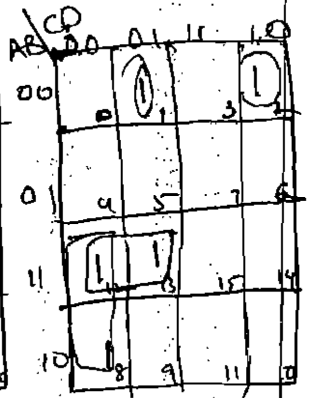
$F_3(A,B,C,D) = \Sigma m(0,2,3,4,5,6,7,8,10,11,15)$

$F_4(A,B,C,D) = \Sigma m(1,2,8,12,13)$









$F_1 = AB\overline{C} + \overline{A}\overline{B}C\overline{D}$ $F_2 = A + BCD$ $F_3 = \overline{A}B + CD + \overline{B}\overline{D}$

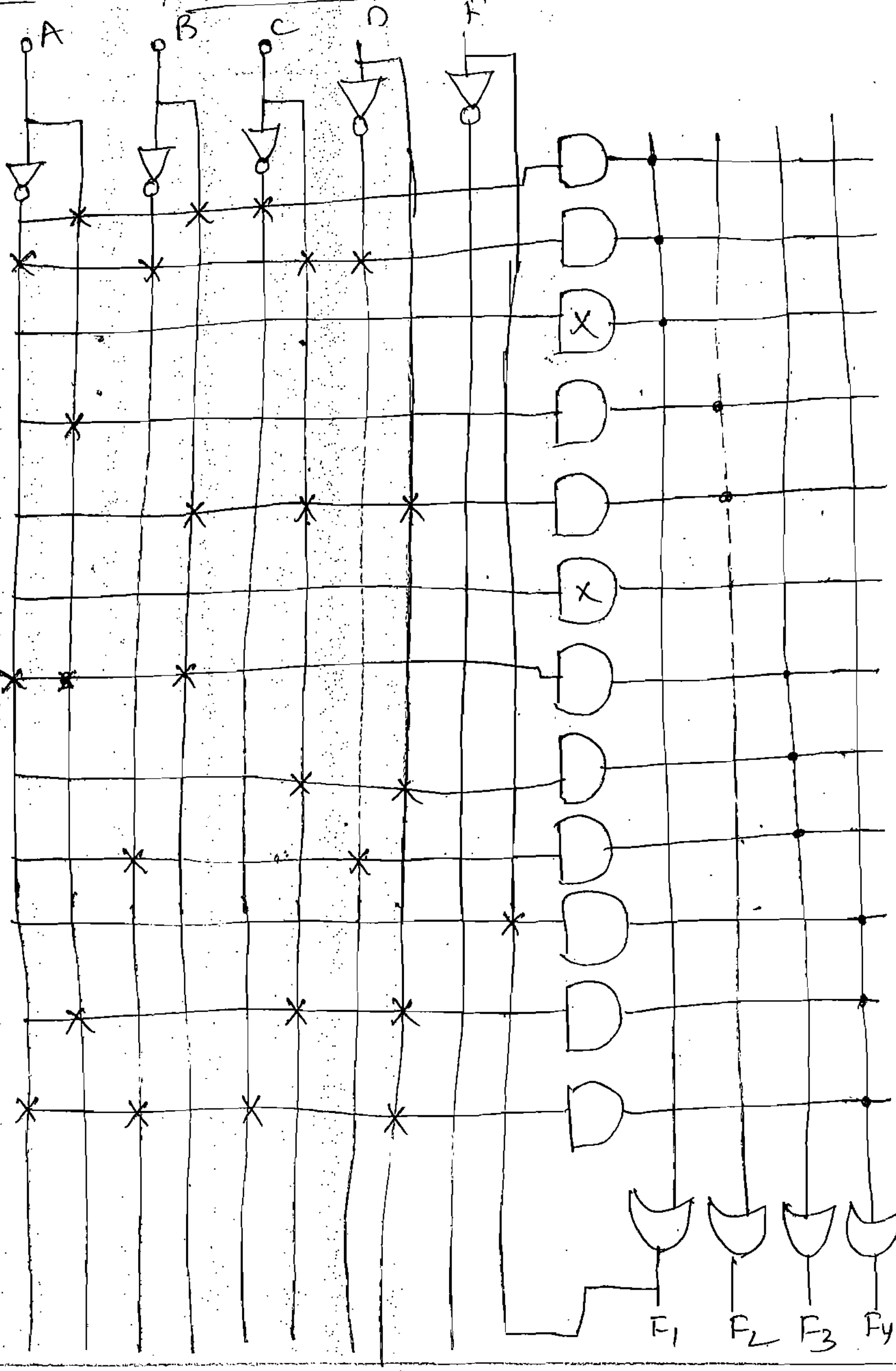
$F_4 = \underbrace{AB\overline{C}} + A\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}C\overline{D}$

$= F_1 + A\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D$

Step 2 :- programming table

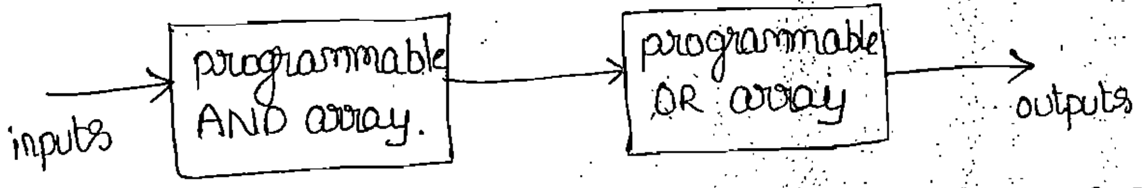
product term	AND inputs A	B	C	D	F_1	outputs
1	1	1	0	–	–	$F_1 = AB\overline{C} + \overline{A}\overline{B}C\overline{D}$
2	0	0	1	0	–	
3	–	–	–	–	–	
4	1	–	–	–	–	$F_2 = A + BCD$
5	–	1	1	1	–	
6	–	–	–	–	–	
7	0	1	–	–	–	$F_3 = \overline{A}B + CD + \overline{B}\overline{D}$
8	–	–	1	1	–	
9	–	0	–	0	–	
10	–	–	–	–	1	$F_4 = F_1 + A\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D$
11	1	–	0	0	–	
12	0	0	0	1	–	

Step 3 :- Implementation :-



PLA :- programmable logic Array

In programmable logic array where both the AND and OR arrays can be programmed. The product terms in the AND array may be shared by any OR gate to provide the required sum of products.

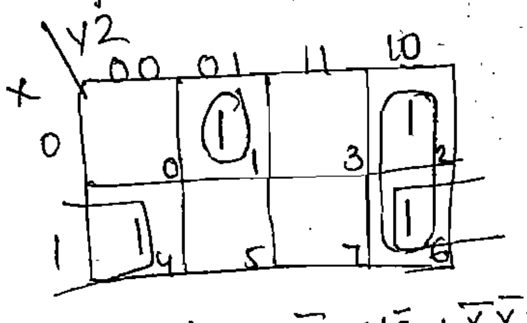
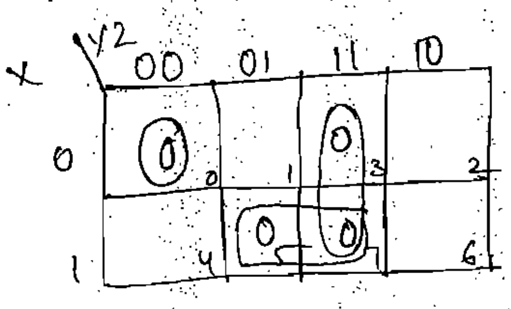


→ implement the given boolean functions by using PLA.

$A(X,Y,Z) = \Sigma m(1,2,4,6)$

$B(X,Y,Z) = \Sigma m(1,2,3,5,7)$

Step 1 :- The K-maps for the functions A, B, their minimization, and the minimal expressions for both the true and complement of those in sum of products.

$A(T) = X\bar{Z} + Y\bar{Z} + \bar{X}\bar{Y}Z.$

~~A(C)~~ $\bar{A} = XZ + YZ + \bar{X}\bar{Y}\bar{Z}$

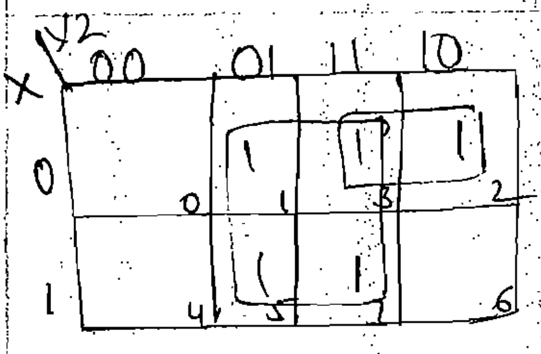
$A(C) = \overline{XZ + YZ + \bar{X}\bar{Y}\bar{Z}}$

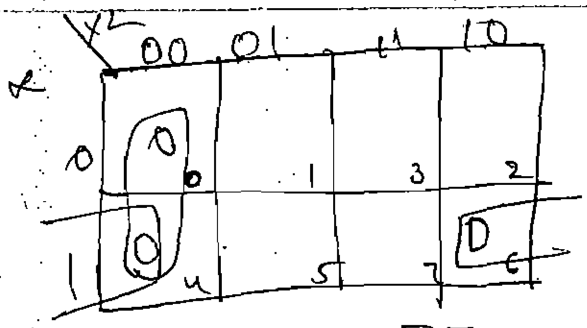
K-map for A.

Simply $A(C) = \overline{(\bar{X}+\bar{Z}) \cdot (\bar{Y}+\bar{Z}) \cdot (X+Y+Z)}$

$= \overline{(\bar{X}+\bar{Z})} + \overline{(\bar{Y}+\bar{Z})} + \overline{(X+Y+Z)}$

$= XZ + YZ + \bar{X}\bar{Y}\bar{Z}$





$B(T) = \overline{X}Y + Z.$

$\overline{B} = X\overline{Z} + \overline{Y}\,\overline{Z}$

$B(C) = \overline{X\overline{Z} + \overline{Y}\overline{Z}}$

Simply $B(C) = \overline{(Y+Z)(\overline{X}+Z)}$

$= \overline{(Y+Z)} + (\overline{\overline{X}+Z})$

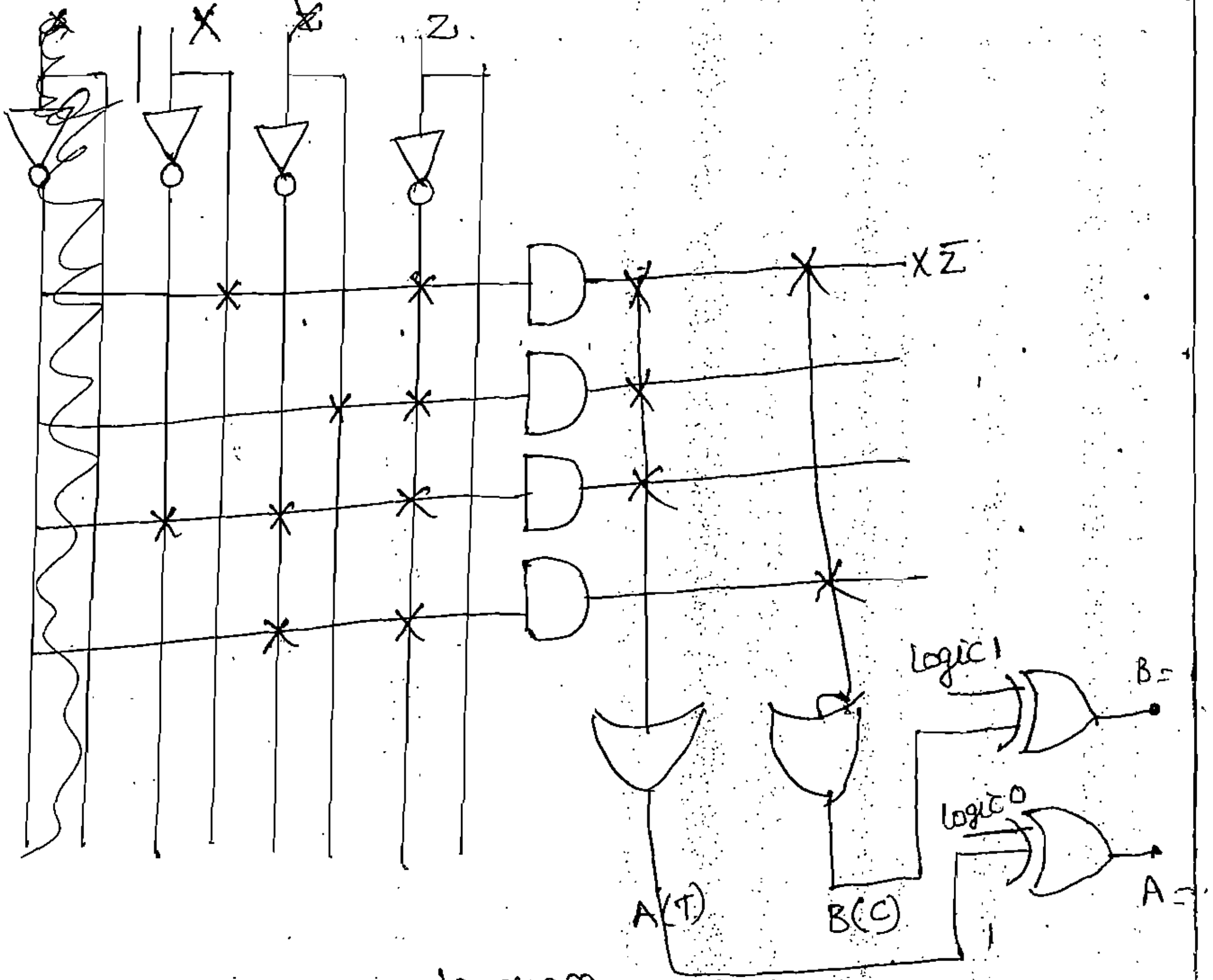
$= \overline{Y}\,\overline{Z} + X\overline{Z}$

✓ $A(T) = X\overline{Z} + Y\overline{Z} + \overline{X}\,\overline{Y}Z$ $\quad B(T) = \overline{X}Y + Z$

$A(C) = XZ + YZ + \overline{XYZ}$ $\quad B(C) = \overline{Y}\,\overline{Z} + X\overline{Z}$ ✓

Step 2 :- programming table.

product term	inputs X	Y	Z	outputs A(T)	A(C)	B(T)	B(C)
$X\overline{Z}$	1	–	0	1	–	–	1
$Y\overline{Z}$	–	1	0	1	–	–	–
$\overline{X}\,\overline{Y}Z$	0	0	1	1	–	–	–
XZ	1	–	1	–	1	–	–
YZ		1	1	–	1	–	–
$\overline{X}\,\overline{Y}\,\overline{Z}$	0	0	0	–	1	–	–
$\overline{X}Y$	0	1	–	–	–	1	–
Z	–	–	1	–	–	1	1
$\overline{Y}\,\overline{Z}$	–	0	0	–	–	–	1
$X\overline{Z}$	1	–	0	–	–	–	1



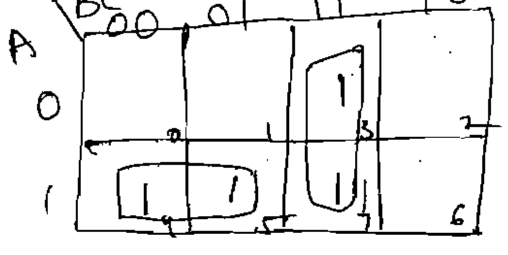
logic diagram.

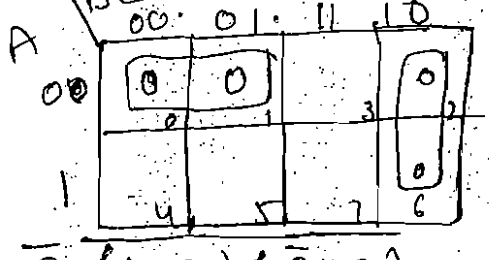
→ Implement the following boolean functions F_1 & F_2 of a combinational logic circuit using PLA.

$F_1(A,B,C) = \Sigma m(3,4,5,7)$

$F_2(A,B,C) = \Sigma m(1,4,6)$.

Step 1:- K-map for F_1 (True form) complement form





$F_1 = A\bar{B} + BC$

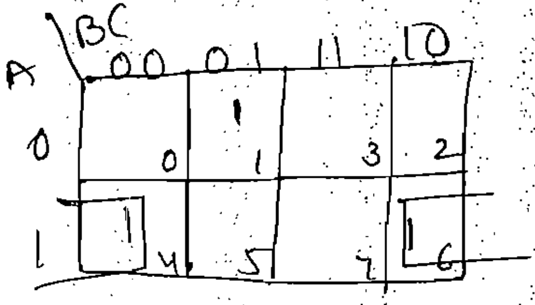
$\bar{F_1} = (A+B)(\bar{B}+C)$

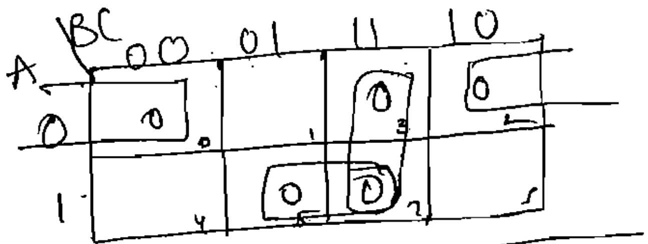
$= \overline{(A+B)} + \overline{(\bar{B}+C)}$

$F_1(T) = A\bar{B} + BC$ $\quad$ $F_1(C) = (\bar{A}.\bar{B}) + \bar{B}.\bar{C}$

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

K-map for F_2 (true form) $\qquad$ K-map for F_2(complement form)





$F_2 = \bar{A}\bar{B}C + A\bar{C}$

$\bar{F_2} = \overline{(A+C)\cdot(\bar{B}+\bar{C})\,(\bar{A}+\bar{C})}$

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

$= \bar{A}.\bar{C} + BC + AC.$

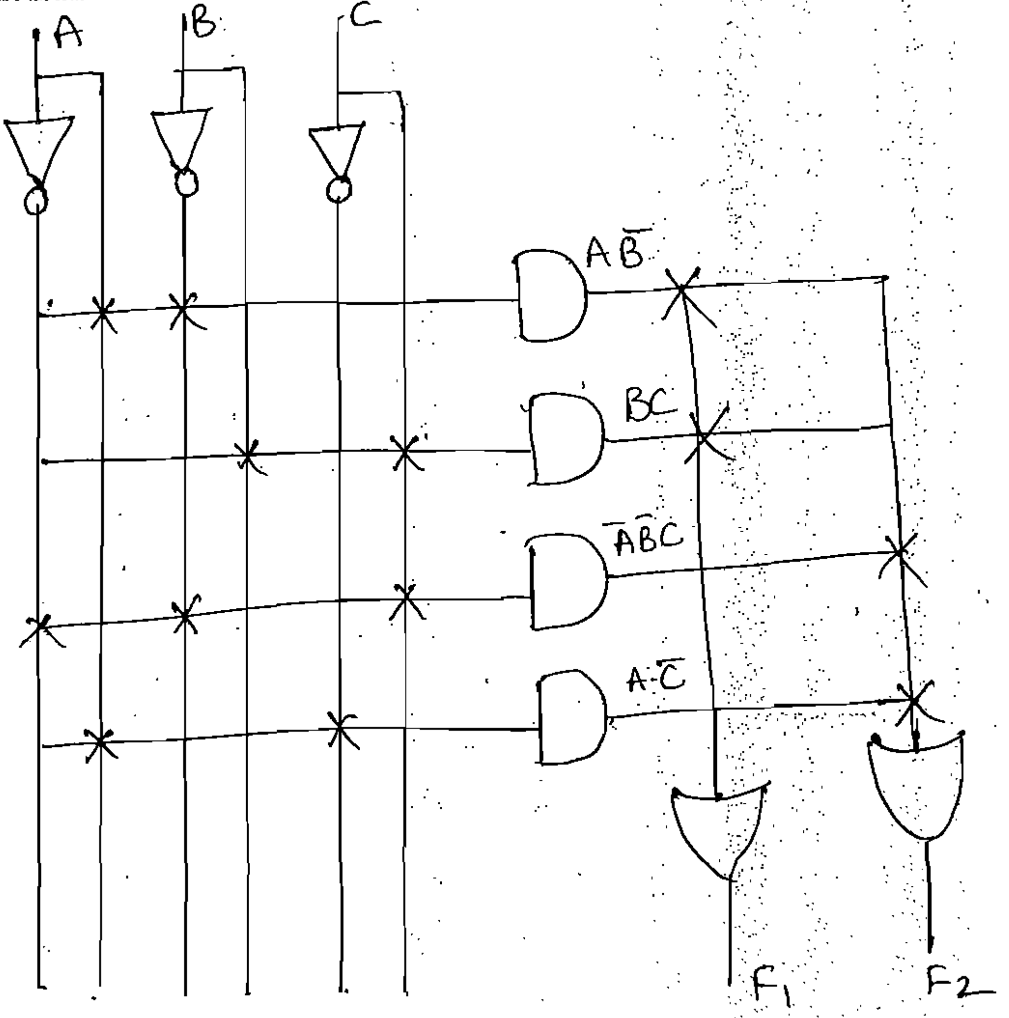
$F_1(T) = A\bar{B} + BC$ $\qquad$ $F_1(C) = \bar{A}\bar{B} + B\bar{C}$.

$F_2(T) = \bar{A}\bar{B}C + A\bar{C}$ $\qquad$ $F_2(C) = \bar{A}\bar{C} + BC + AC$

When we take $F_1(T)$ & $F_2(T)$ get 4 product terms and also having same number of product terms while taking $F_1(T)$ & $F_2(C)$, $F_1(C)$ & $F_2(T)$. So we have to consider $F_1(T)$ & $F_2(T)$.

Step 2 :- programming table

product terms	inputs			outputs	
	A	B	C	$F_1(T)$	$F_2(C)$
$A\bar{B}$	1	0	–	1	–
BC	–	1	1	1	1
$\bar{A}\bar{B}\bar{C}$	0	–	0	1	1
$A\bar{C}$	1	–	1	–	1



logic diagram.

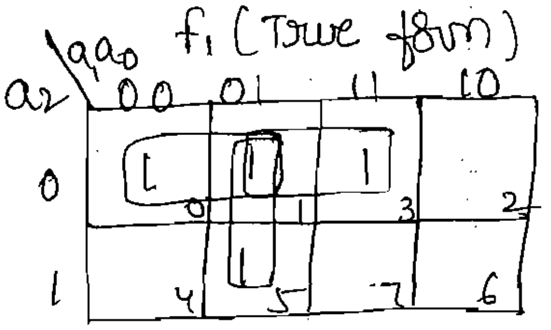
→ Implement the following multi boolean function using (3x4x2) PLA.

$F_1(a_2, a_1, a_0) = \Sigma m(0,1,3,5)$

$F_2(a_2, a_1, a_0) = \Sigma m(3,5,7)$

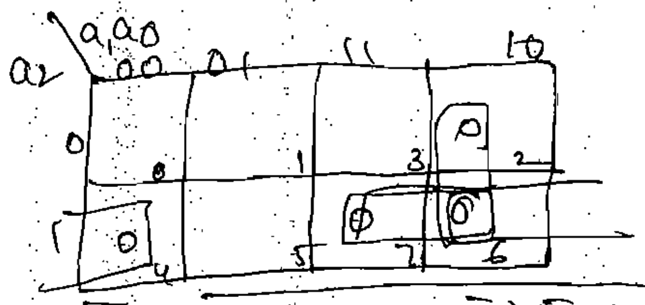
step 1:- K-maps.

f_1 (True form)



$f_1(T) = \bar{a}_1 a_0 + \bar{a}_2 \bar{a}_1 + \bar{a}_2 a_0$

f_1 (complement form)

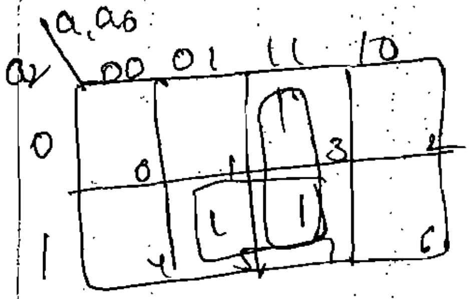


$\bar{f}_1 = (\bar{a}_2 + a_0)(\bar{a}_2 + \bar{a}_1)(\bar{a}_1 + a_0)$

$f_1(C) = \overline{\overline{a}_2 \cdot a_0 + \overline{a}_2 \cdot \overline{a}_1 + \overline{a}_1 \cdot a_0}$

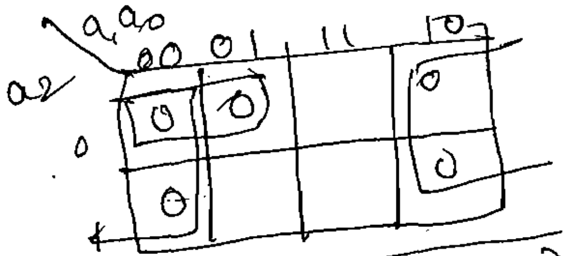
$= a_2 \cdot \overline{a}_0 + a_2 \cdot a_1 + a_1 \cdot \overline{a}_0$

F_2(True form)



$f_2(T) = a_2 a_0 + a_1 a_0$

F_2 (complement form..



$\overline{F_2} = (\overline{a_0}) \cdot (a_2 + a_1)$

$F_2(C) = \overline{a}_0 + \overline{a}_2 \cdot \overline{a}_1$

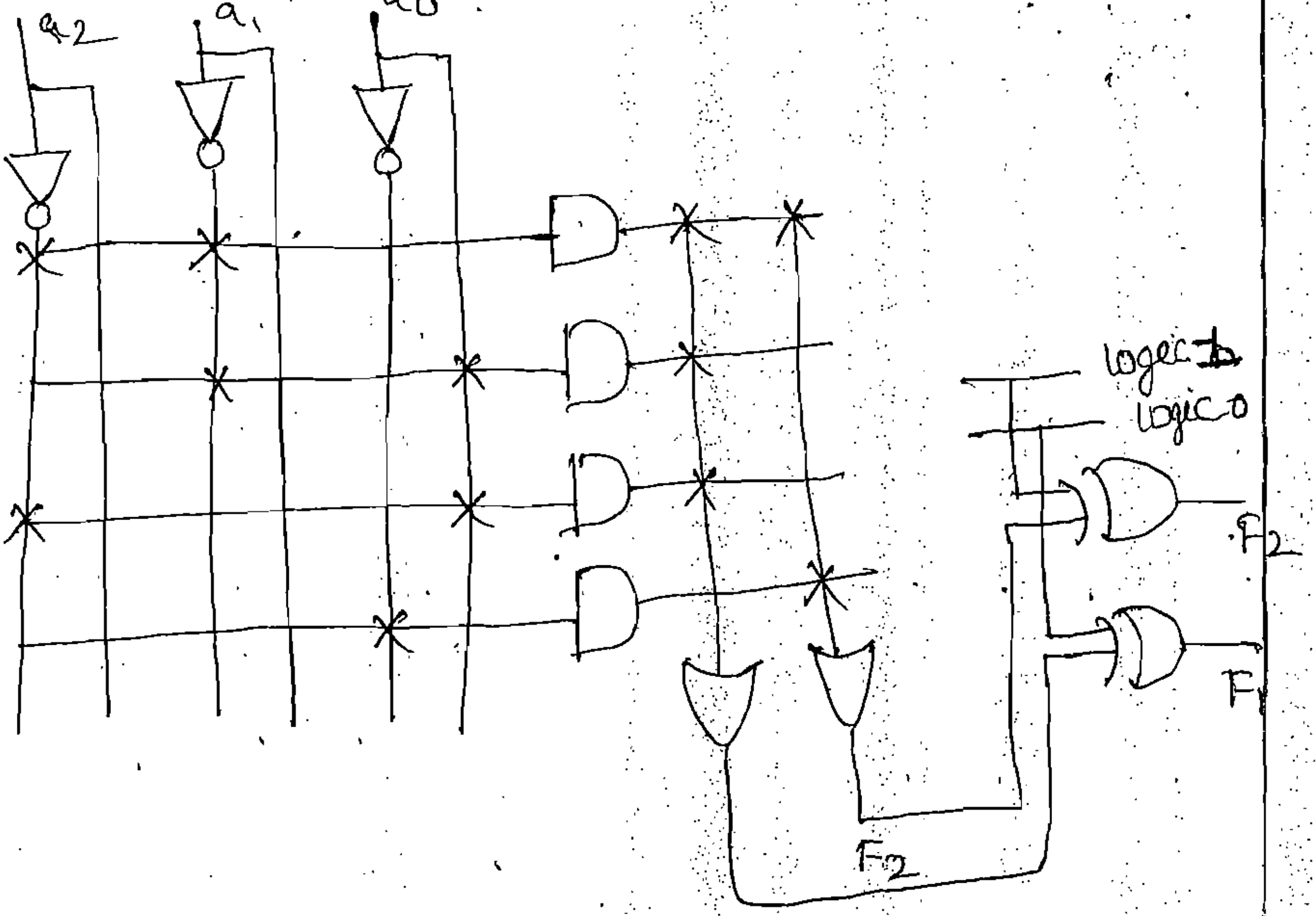
$F_1(T) = \overline{a}_1 a_0 + \overline{a}_2 \overline{a}_1 + \overline{a}_2 a_0$ $\quad$ $F_1(C) = a_2 \cdot \overline{a}_0 + a_2 \cdot a_1 + a_1 \cdot \overline{a}_0$

$F_2(T) = a_2 a_0 + a_1 a_0$ $\quad$ $F_2(C) = \overline{a}_0 + \overline{a}_2 \cdot \overline{a}_1$

Step 2 :- PLA programming table.

product terms	Inputs			outputs	
	a_2	a_1	a_0	$F_1(T)$	$F_2(C)$
$\overline{a}_1 a_0$	—	0	1	1	—
$\overline{a}_2 \overline{a}_1$	0	0	—	1	1
$\overline{a}_2 a_0$	0	—	1	1	—
$\overline{a}_0$	—	—	0	—	1

Step 3 :- logic diagram :-



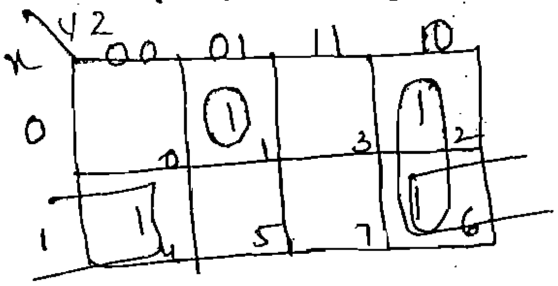
→ Implement the following using PLA

$A(x,y,z) = \Sigma m(1,2,4,6)$, $B(x,y,z) = \Sigma m(0,1,6,7)$

$C(x,y,z) = \Sigma m(2,6)$.

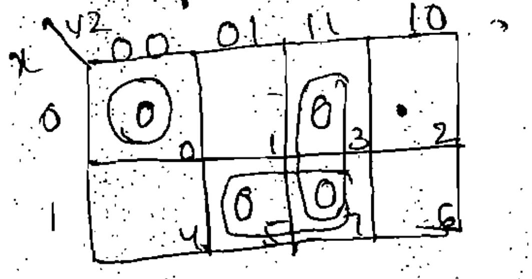
Step 1 :- K-map.

K-map for A (True form).



$A(T) = x\bar{z} + y\bar{z} + \bar{x}\bar{y}z$

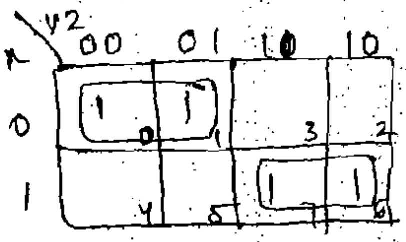
K-map for A (complement form).



$A(C) = \overline{(x+y+z)\cdot(\bar{y}+\bar{z})(\bar{x}+\bar{z})}$

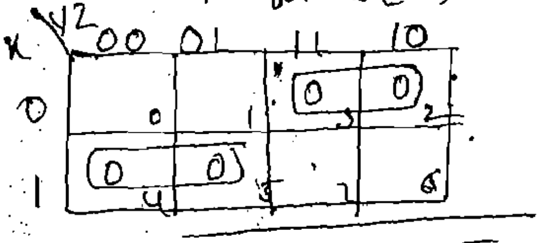
$= \bar{x}\bar{y}\bar{z} + yz + xz.$

K-map for B(T)



$B(T) = \bar{x}\bar{y} + xy$.

K-map for B(C)

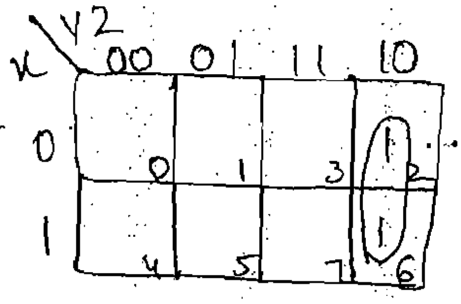


$B(C) = \overline{(\bar{x}+y)(x+\bar{y})}$

$= \bar{\bar{x}}\cdot\bar{y} + \bar{x}\bar{\bar{y}}$

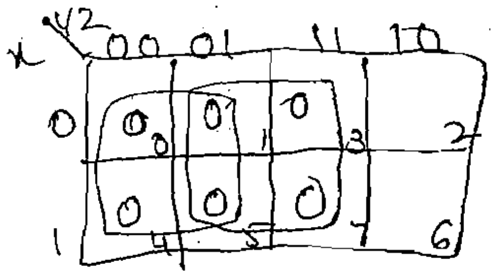
$= x\bar{y} + \bar{x}y$.

K-map for C(T)



$C(T) = y\bar{z}$

K-map for C(C)



$C(C) =$ ~~$\bar{y}\cdot z$~~ $\overline{(y\cdot\bar{z})}$

$= \bar{y} + \bar{\bar{z}}$

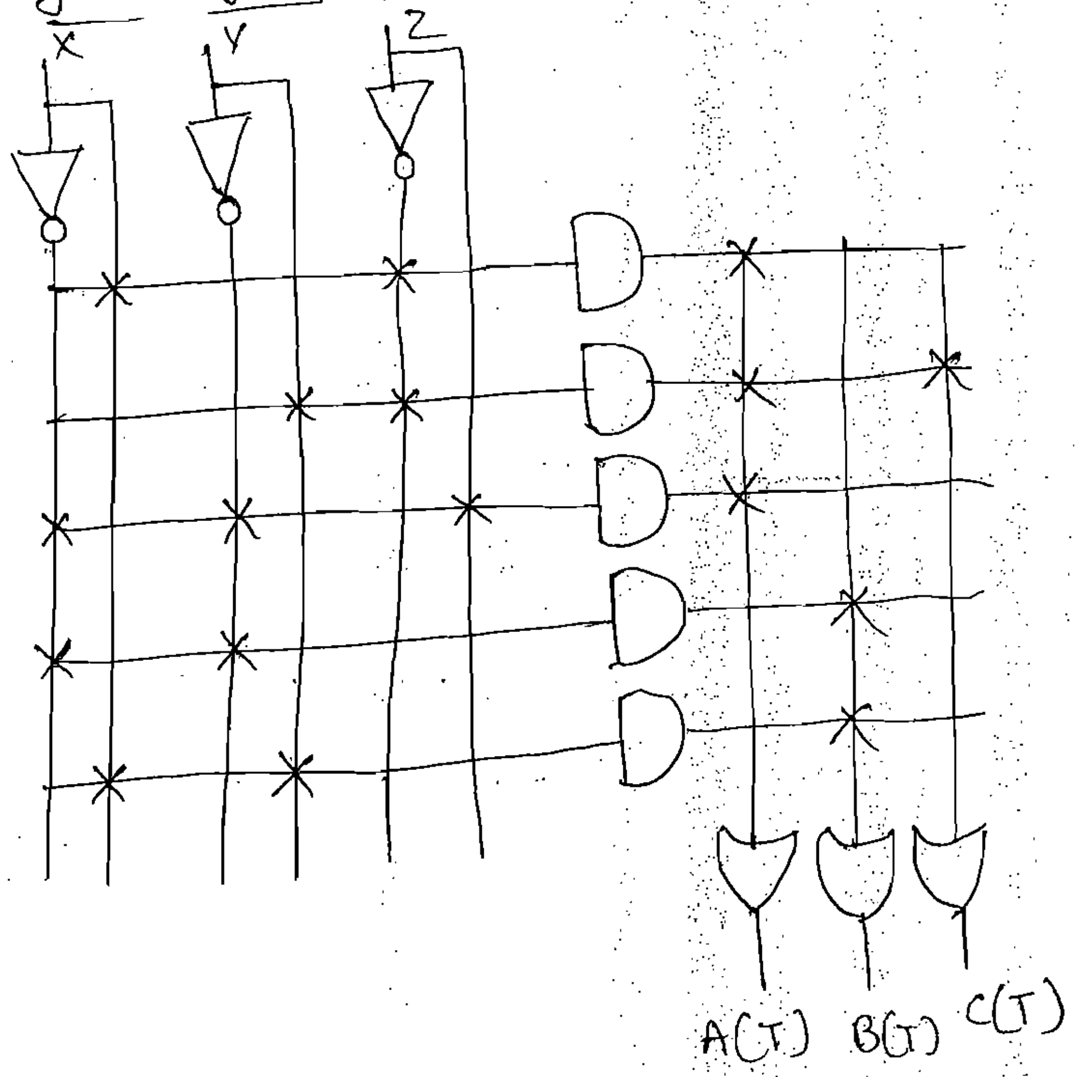
$= \bar{y} + z$.

Step 2 :- programming table.

product term	inputs			outputs		
	x	y	z.	A(T)	B(T)	C(T)
$x\bar{z}$	1	–	0	1	–	–
$y\bar{z}$	–	1	0	1	–	1
$\bar{x}\bar{y}z$	0	0	1	1	–	–
$\bar{x}\bar{y}$	0	0	–	–	1	–
xy	1	1	–	–	1	–

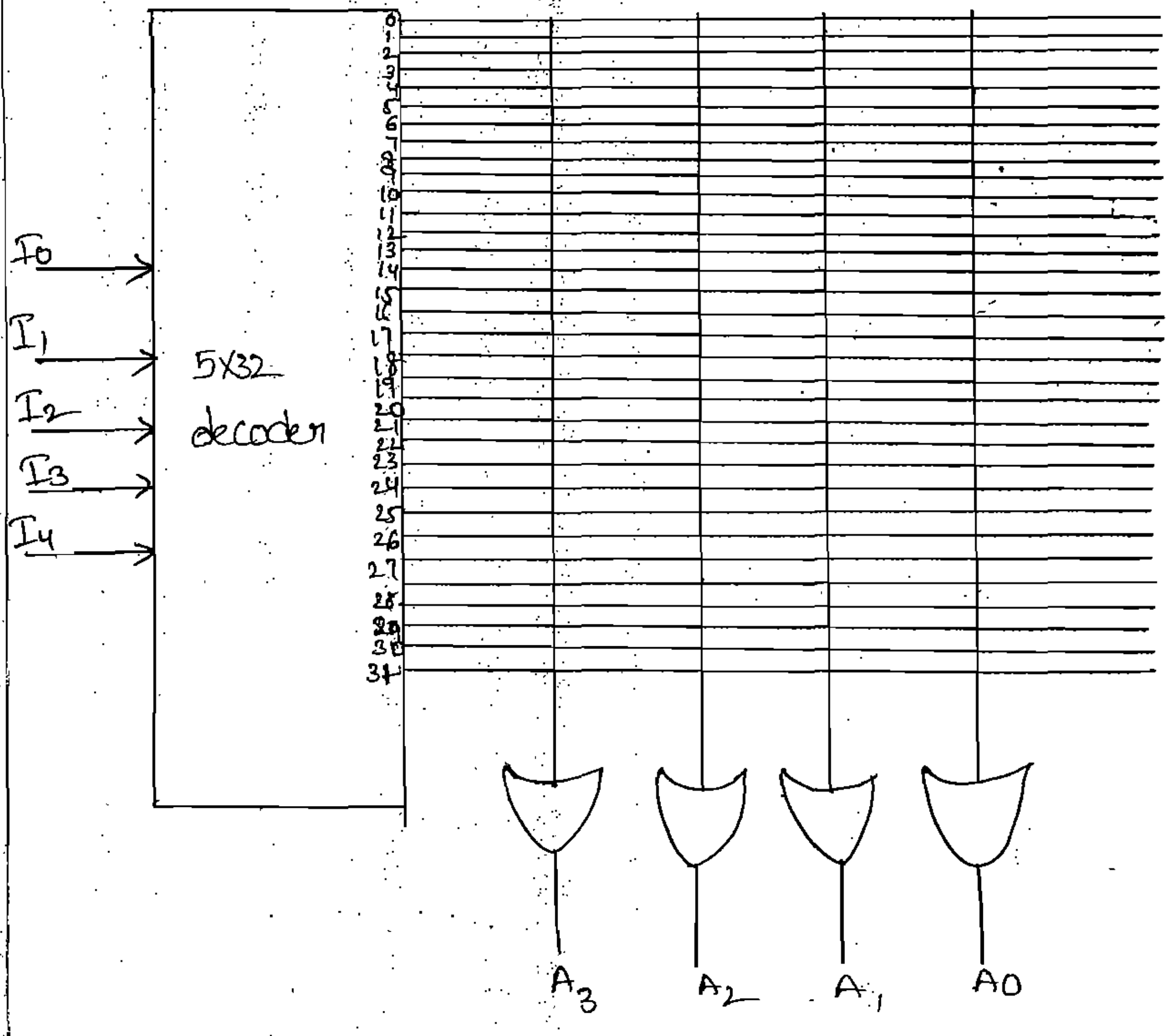
Step - 3

Logic diagram :-



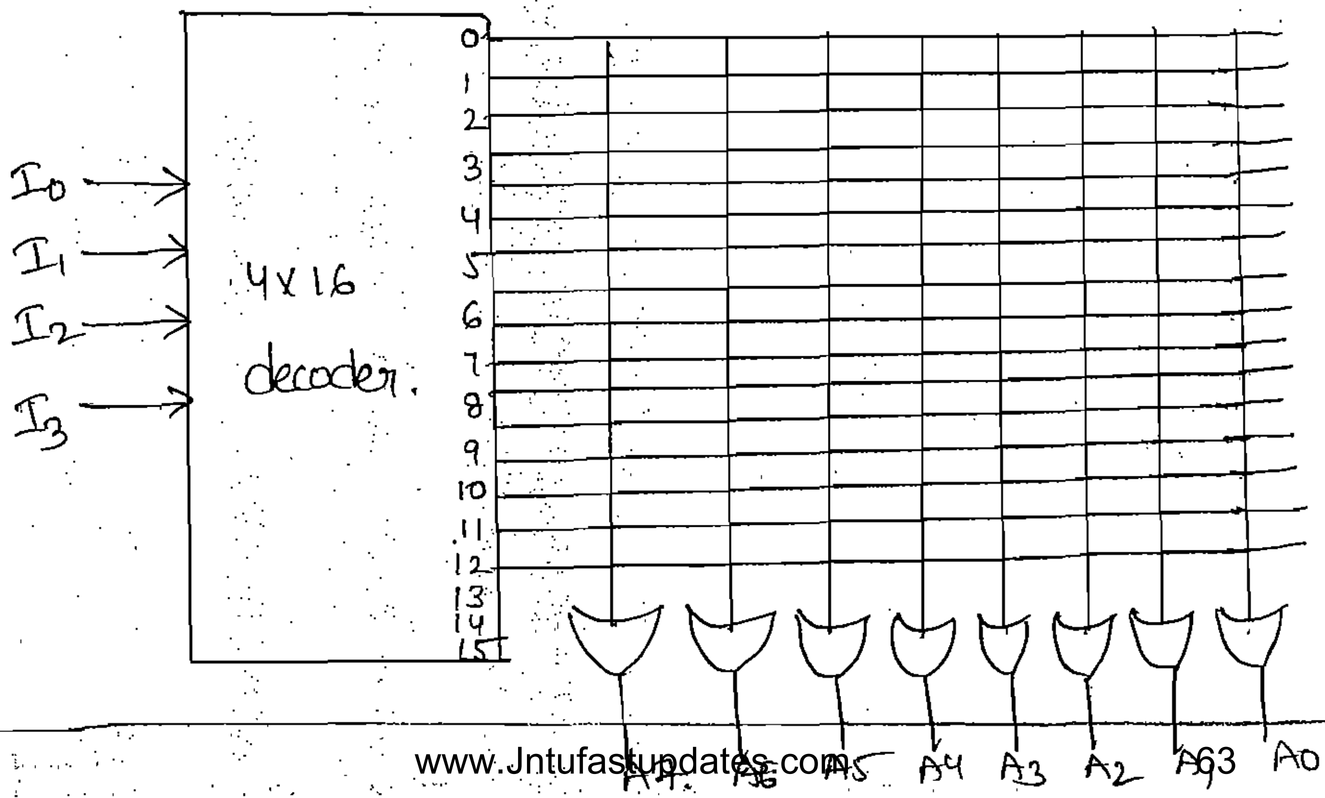
→ Give the logic implementation of a 32x4 bit ROM using a decoder of a suitable size.

A 32x4 bit ROM is to be implemented. it consists of 32 words of four bits each. There must be five input lines that form the binary numbers from 0 through 31 for the address. The five inputs are decoded into 32 distinct outputs by means of a 5x32 decoder. Each output of the decoder represent a memory address. The 32 outputs of the decoder are connected to each of the four OR gates.



32 X 4 bit ROM.

→ 16 X 8 ROM.



Comparison between PROM, PLA and PAL.

PROM	PLA	PAL
1. AND array is fixed and OR array is program-mable.	1. Both AND and OR arrays are programmable.	1. OR array is fixed and AND array is programmable.
2. cheaper and simple to use	2. costliest and more complex than PAL and PROM.	2. Cheaper and simpler.
3. All minterms are decoded.	3. AND array can be programmed to get desired minterms.	3. AND array can be programmed to get desired minterms.
4. only Boolean functions in standard sop form can be implemented using PROM.	4. Any Boolean function in sop form can be impleme nted using ~~PROM~~.	4. Any Boolean function in sop form can be implement ed using PAL.

→ Implement a Binary to BCD code converter by using PAL

→ Iam taking 3-bit Binary P,

Binary			BCD code			
B_3	B_2	B_1	C_4	C_3	C_2	C_1
0	0	0	0	0	0	0
0	0	1	0	0	0	1
0	1	0	0	0	1	0
0	1	1	0	0	1	1
1	0	0	0	1	0	0
1	0	1	0	1	0	1
1	1	0	0	1	1	0
1	1	1	0	1	1	1

→ Iam taking 4-binary.

$B_4\ B_3\ B_2\ B_1$	$C_8\ C_7\ C_6\ C_5\ \ C_4\ C_3\ C_2\ C_1$
0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 1	0 0 0 0 0 0 0 1
0 0 1 0	0 0 0 0 0 0 1 0
0 0 1 1	0 0 0 0 0 0 1 1
0 1 0 0	0 0 0 0 0 1 0 0
0 1 0 1	0 0 0 0 0 1 0 1
0 1 1 0	0 0 0 0 0 1 1 0
0 1 1 1	0 0 0 0 0 1 1 1
1 0 0 0	0 0 0 0 1 0 0 0
1 0 0 1	0 0 0 0 1 0 0 1
1 0 1 0	0 0 0 1 0 0 0 0
1 0 1 1	0 0 0 1 0 0 0 1
1 1 0 0	0 0 0 1 0 0 1 0
1 1 0 1	0 0 0 1 0 0 1 1
1 1 1 0	0 0 0 1 0 1 0 0
1 1 1 1	0 0 0 1 0 1 0 1

$C_5 = \Sigma m(10,11,12,13,14,15)$.

$C_4 = \Sigma m(8,9)$

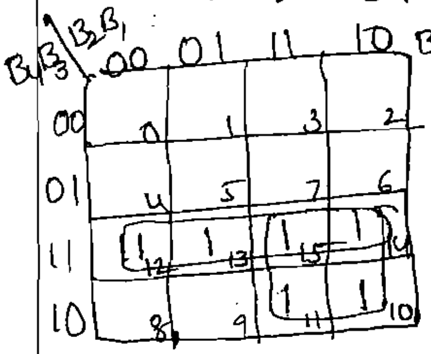
$C_3 = \Sigma m(4,5,6,7,14,15)$

$C_2 = \Sigma m(2,3,6,7,12,13)$

$C_1 = \Sigma m(1,3,5,7,9,11,13,15)$.

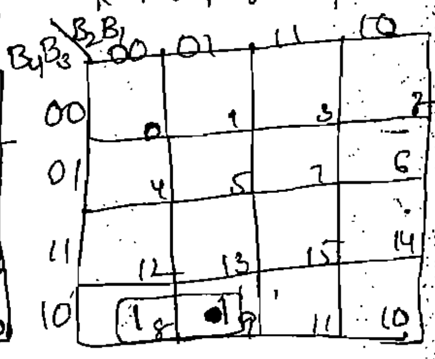
Step 1:- K-map.

K-map for C_5.



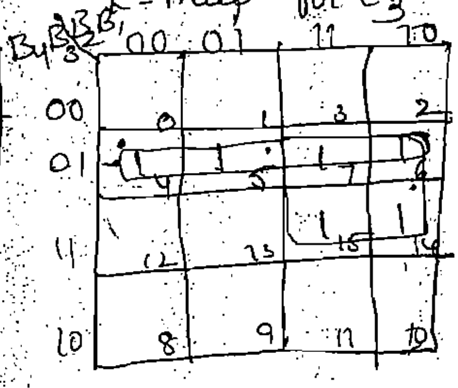
$B_4B_3 + B_4B_2$.

K-map for C_4



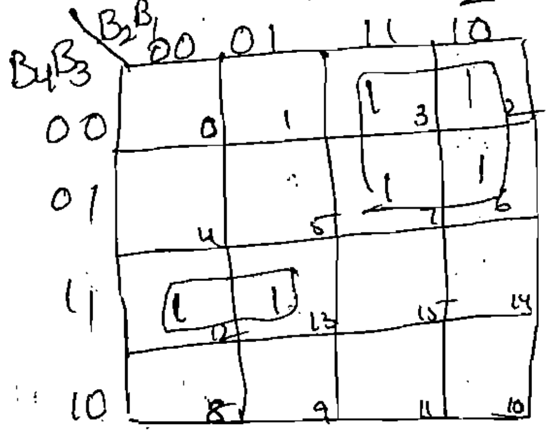
$B_4\overline{B_3}\,\overline{B_2}$

K-map for C_3



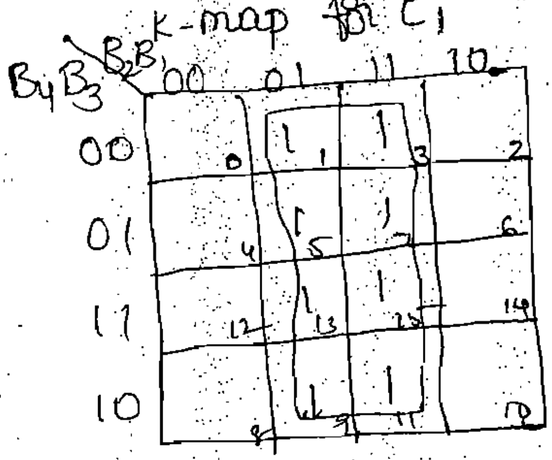
$\overline{B_4}B_3 + B_3B_2$

K-map for C_2



$B_4B_3\overline{B_2} + \overline{B_4}B_2$

K-map for C_1

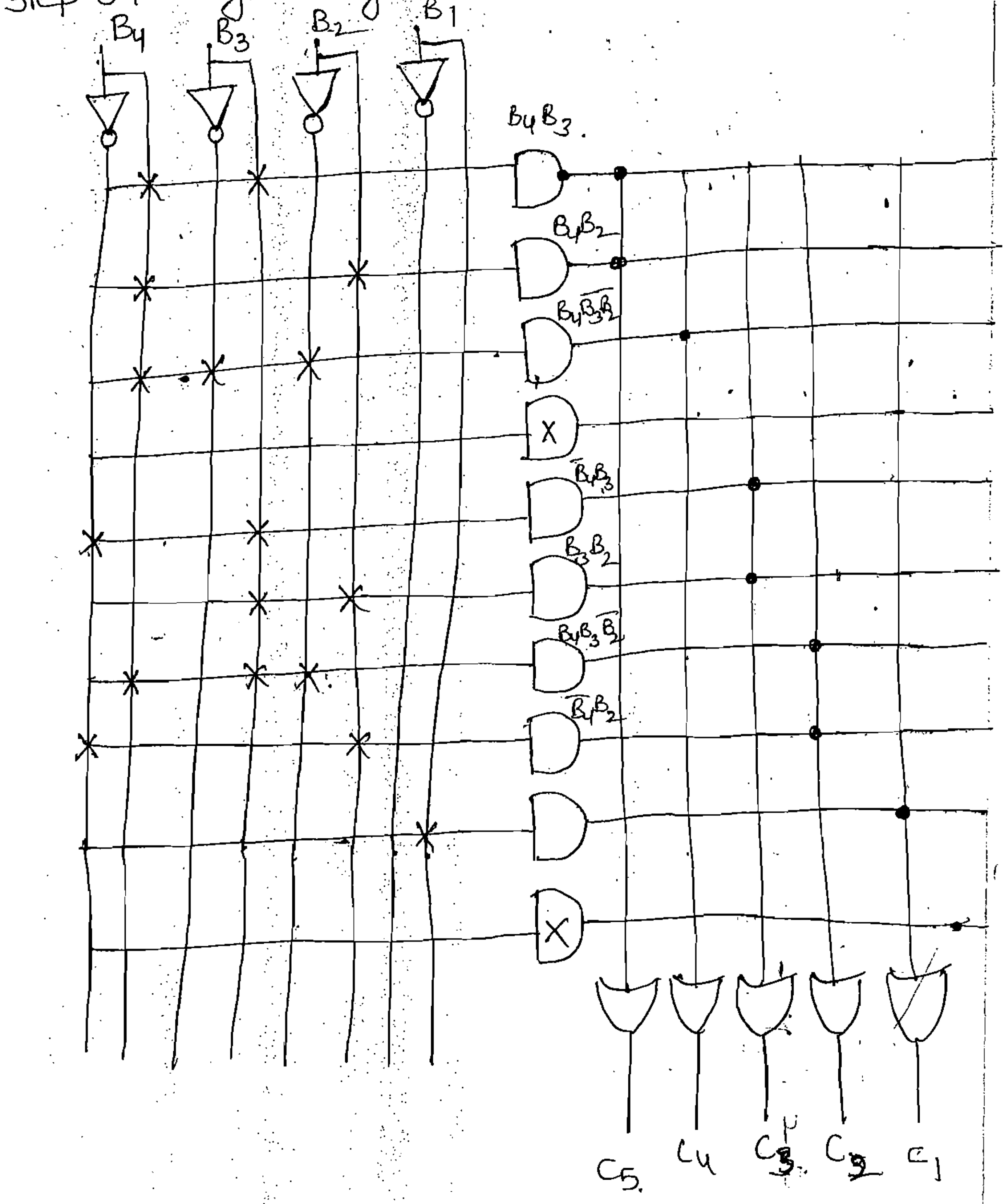


B_1

step 2 :- programming table

product terms	inputs B_4	B_3	B_2	B_1	output
1	1	1	–	–	$C_5 = B_4B_3 +$
2	1	–	1	–	B_4B_2
3	1	0	0	–	$C_4 = B_4\overline{B_3}\,\overline{B_2}$
4	–	–	–	–	
5	0	1	–	–	$C_3 = \overline{B_4}B_3 +$
6	–	1	1	–	B_3B_2
7	1	1	0	–	$C_2 = B_4B_3\overline{B_2} +$
8	0	–	1	–	$\overline{B_4}B_2$
9	–	–	–	1	$C_1 = B_1$
10	–	–	–	–	

Step 3 :- logic diagram



B_4
B_3
B_2
B_1
B_4B_3
B_4B_2
$B_4\overline{B_3}\overline{B_2}$
X
$\overline{B_4}B_3$
B_3B_2
$B_4B_3\overline{B_2}$
$\overline{B_4}B_2$
X
C_5
C_4
C_3
C_2
C_1