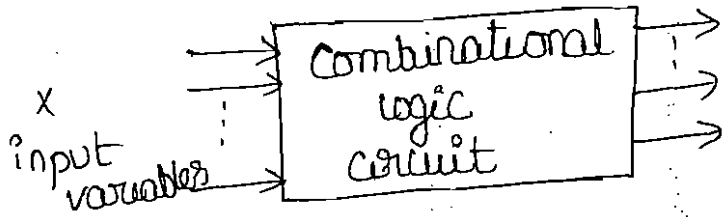
12.6 Combinational logic circuits design.

Logic circuits for digital systems may be combinational or sequential. In combinational circuits, the output variables 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.

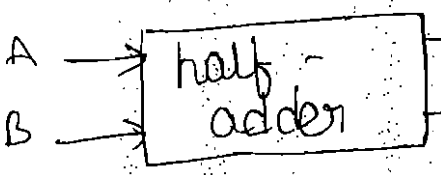


Adders :-

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

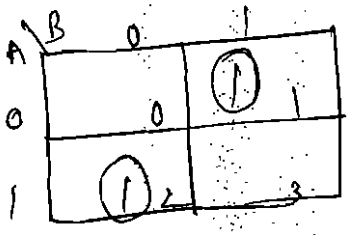
A half adder is a combinational circuit with two binary inputs. (augend and addend bits) and two outputs. Sum and carry.



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.

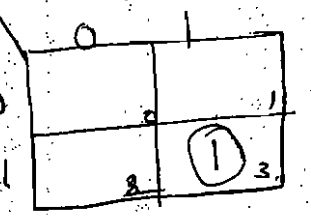
K-map for S.



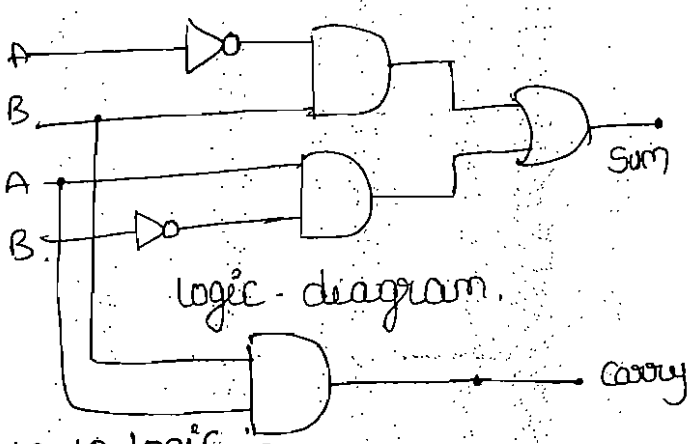
$S = A\bar{B} + \bar{A}B$

$= A \oplus B$

K-map for C

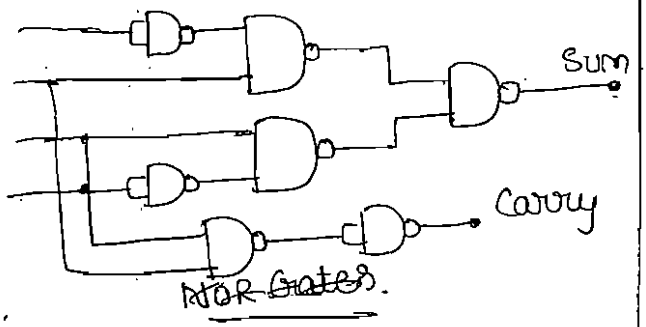


$C = A \cdot B$



Logic - diagram.

NAND Gates.



NOR Gates.

NAND logic :-

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

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

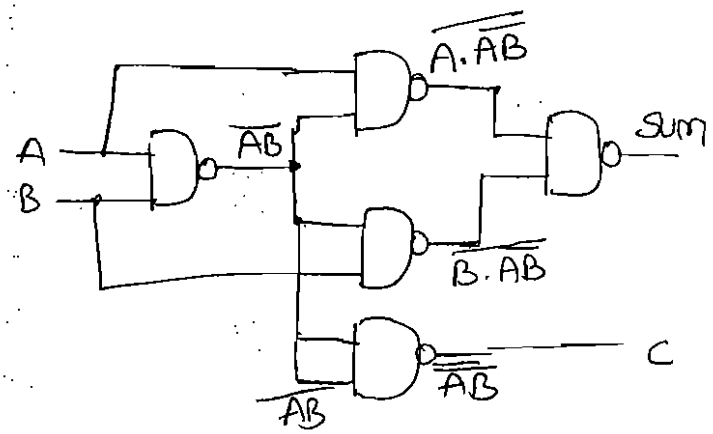
$= A(\bar{A}+\bar{B}) + B(\bar{A}+\bar{B})$

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

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

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

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



NOR logic

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

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

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

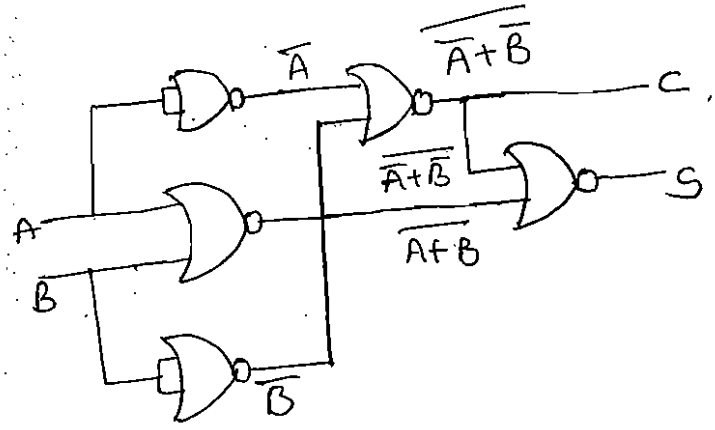
$= (A+B)(\bar{A}+\bar{B})$

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

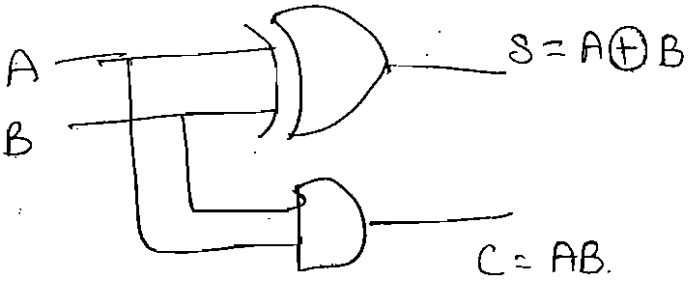
$= \overline{(\bar{A}+B)} + \overline{(\bar{A}+\bar{B})}$

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

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

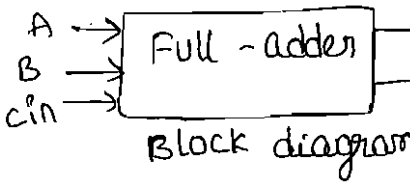


Simple logic diagram is.



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}.

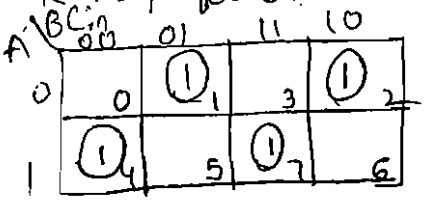


Block diagram.

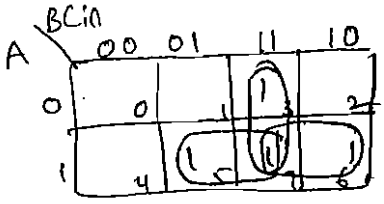
inputs			outputs	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Truth table

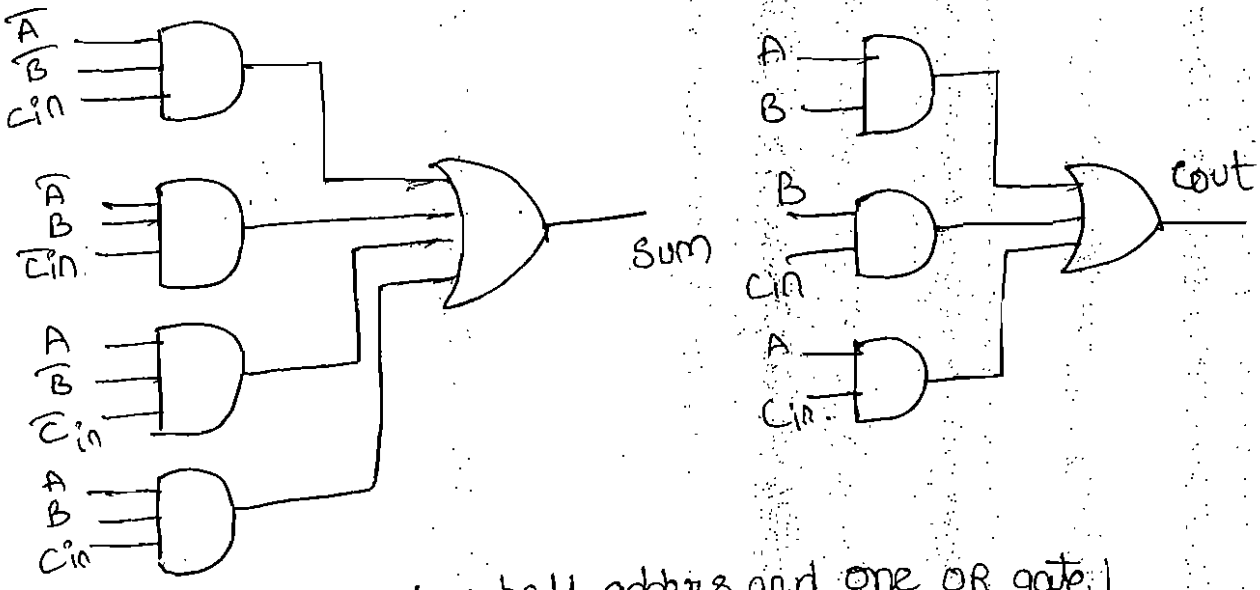
K-map for S.



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



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



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

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

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

$= \overline{A \oplus B}\, C_{in} + \overline{C_{in}}(A \oplus B)'$

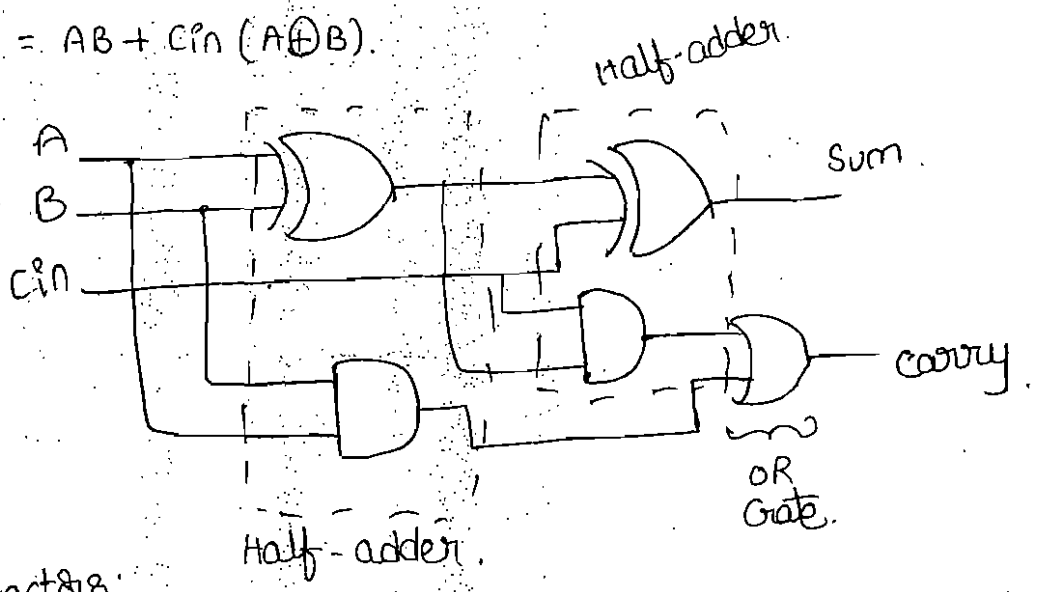
$= A \oplus B \oplus C_{in}$.

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

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

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

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

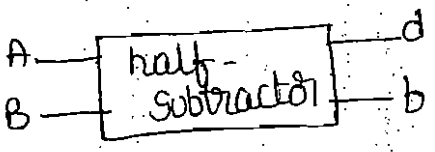


Subtractors:-

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

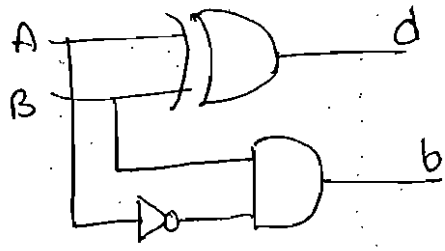
K-map for d

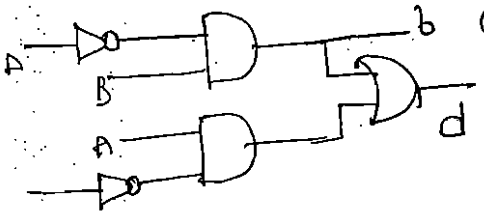
A \ B	0	1
0	0	(1) 1
1	(1) 2	3

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

$= A \oplus B$

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





A \ B	0	1
0	0	(1) 1
1	2	3

$b = \bar{A}B$

Logic diagrams of a half-subtractor.

NAND logic:-

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

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

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

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

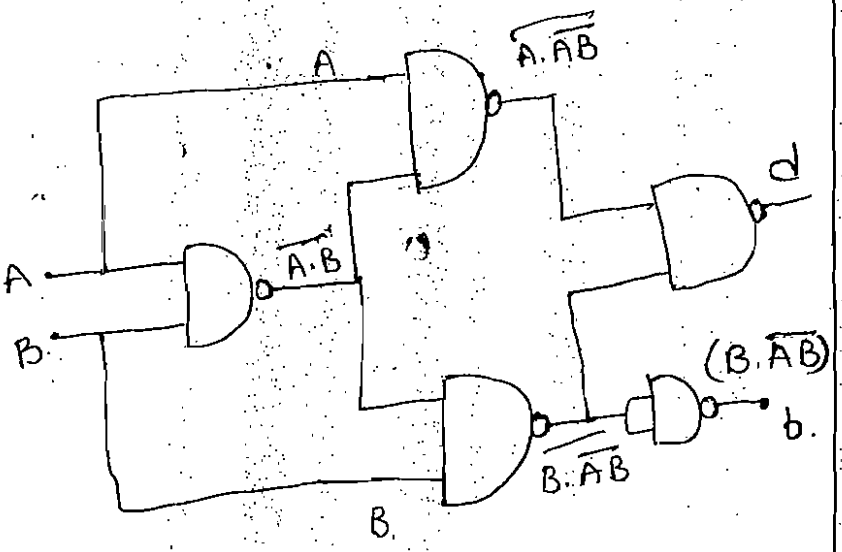
$= A.\overline{\bar{A}B}.\ B.\overline{\bar{A}B}$

$b = \bar{A}B$

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

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

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



NOR logic

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

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

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

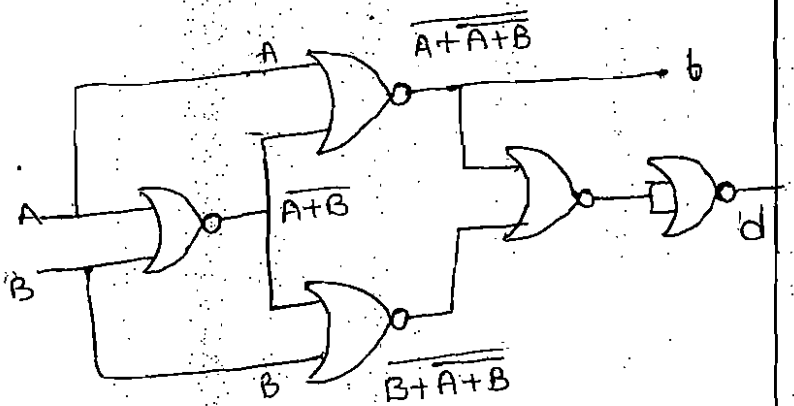
$= \overline{B+\bar{A}+B} + \overline{A+\bar{A}+B}$

$b = \bar{A}B$

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

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

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



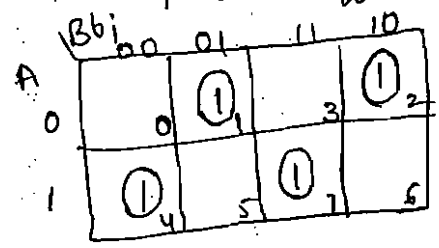
Full - Subtractor:-

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



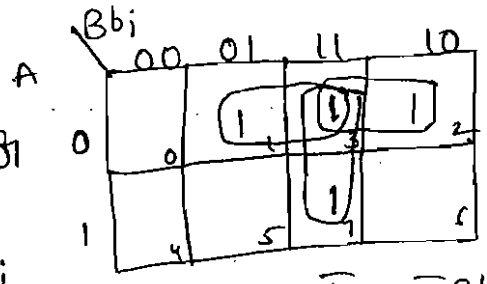
$$d = \bar{A}\bar{B}b_i + \bar{A}B\bar{b_i} + A\bar{B}\bar{b_i} + AB b_i$$

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

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

$$= A \oplus B \oplus b_i$$

K-map for borrow.

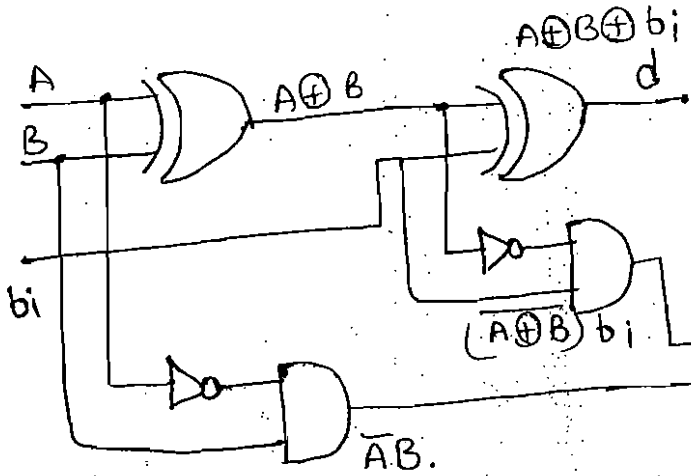


$$b = \bar{A}\bar{B}b_i + \bar{A}B\bar{b_i} + \bar{A}Bb_i + AB b_i$$

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

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

Full-Subtractor by using two half-Subtractor



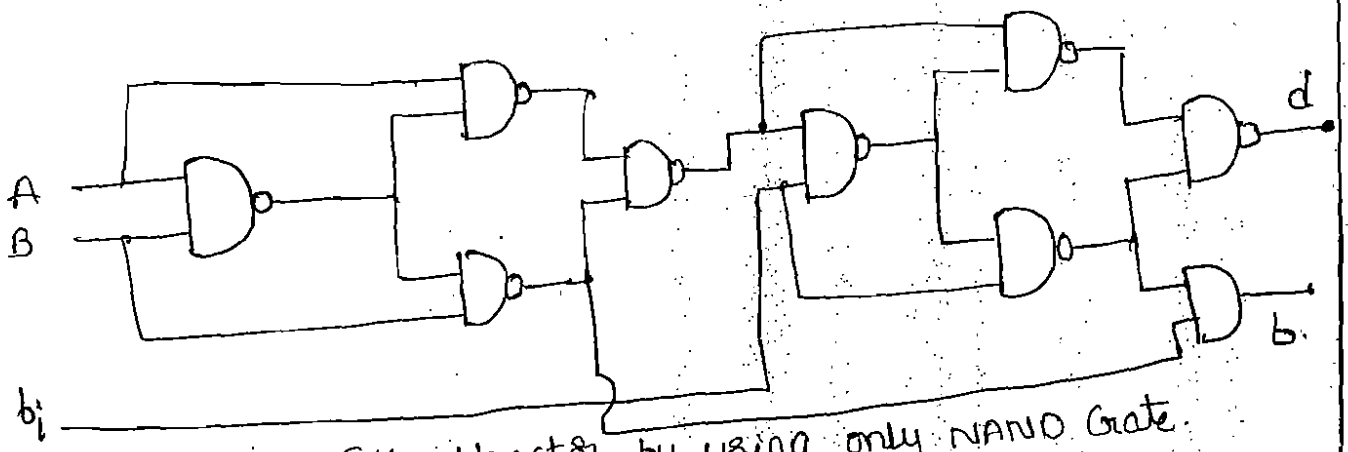
NAND logic:-

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

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

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

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



Full subtractor by using only NAND Gate.

NOR Logic.

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

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

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

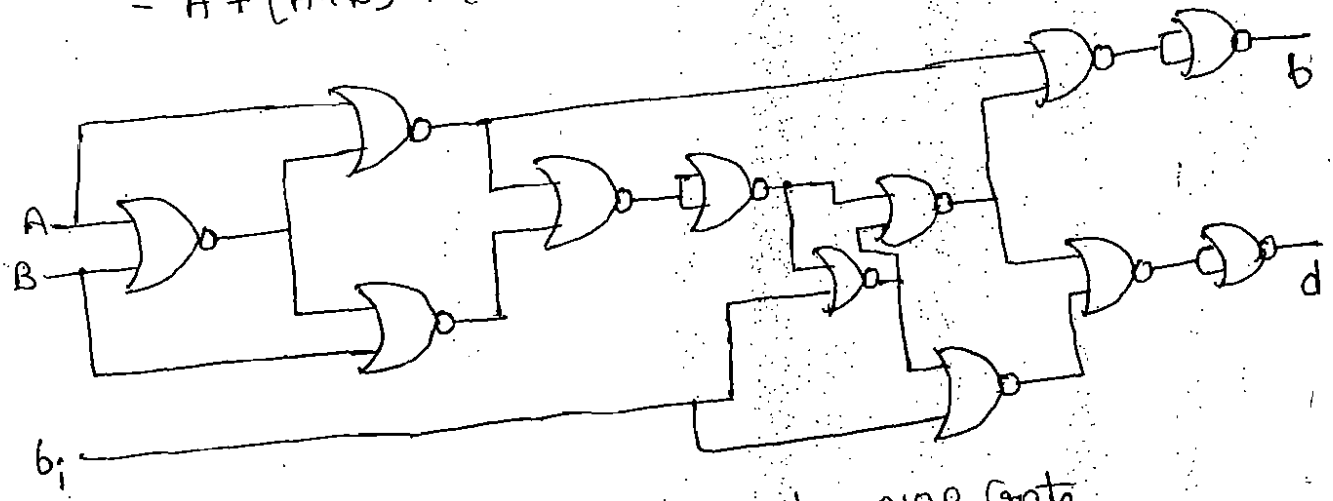
$$= \overline{(A \oplus B) + (\overline{A \oplus B}) + b_i} + \overline{b_i + (A \oplus B) + b_i}$$

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

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

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

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

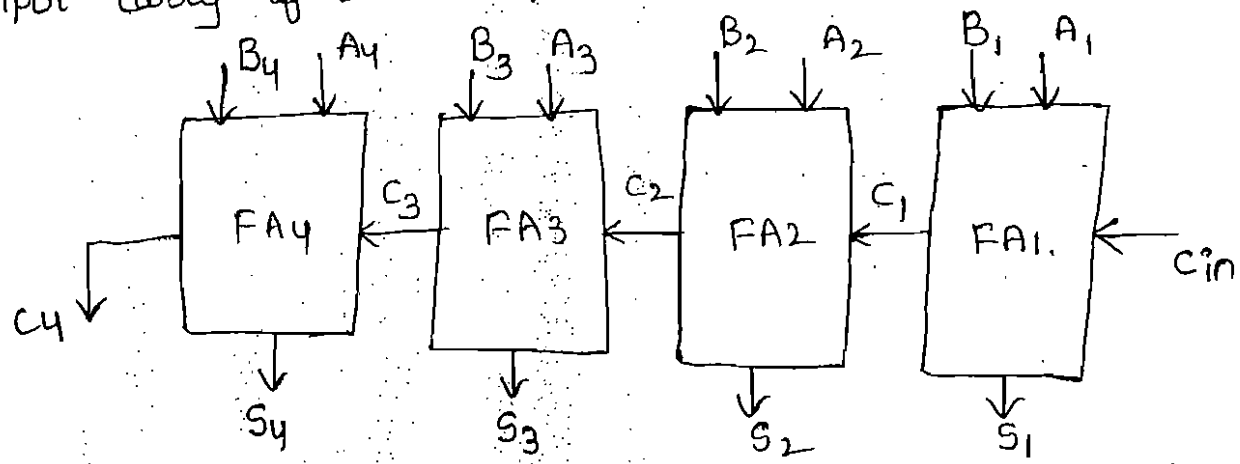


Full subtractor by using only NOR Gate

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.



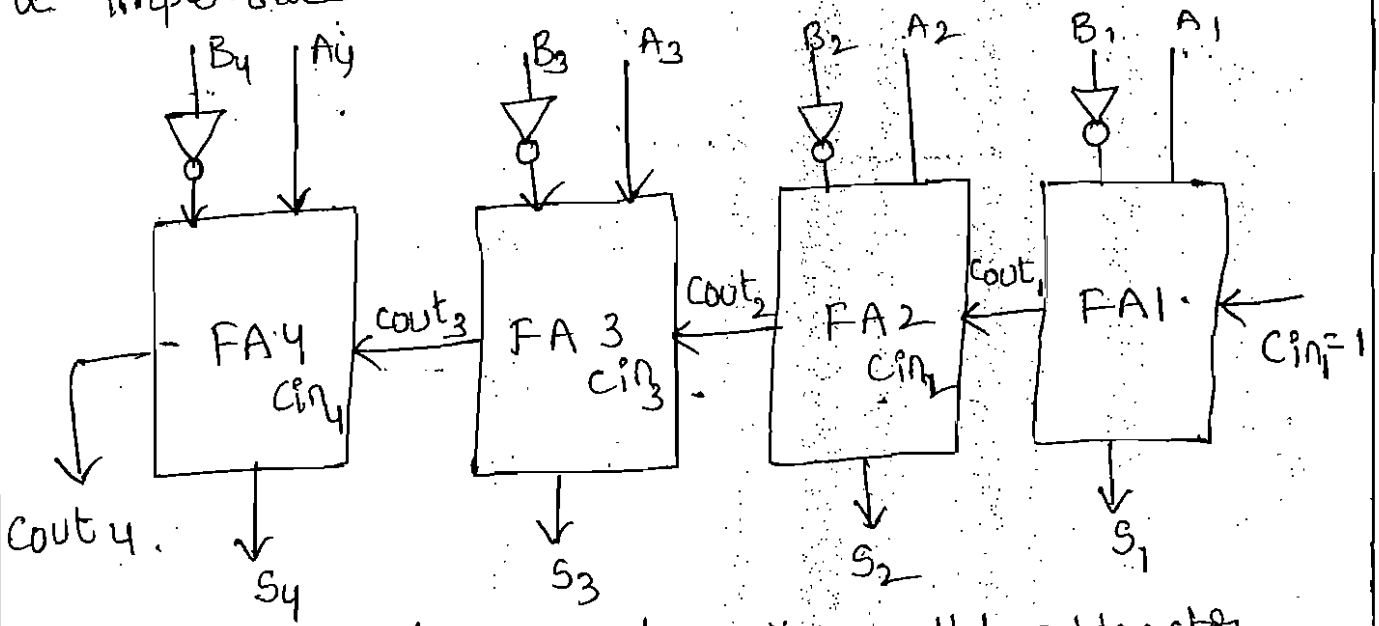
Logic diagram of 4-bit binary parallel adder.

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.

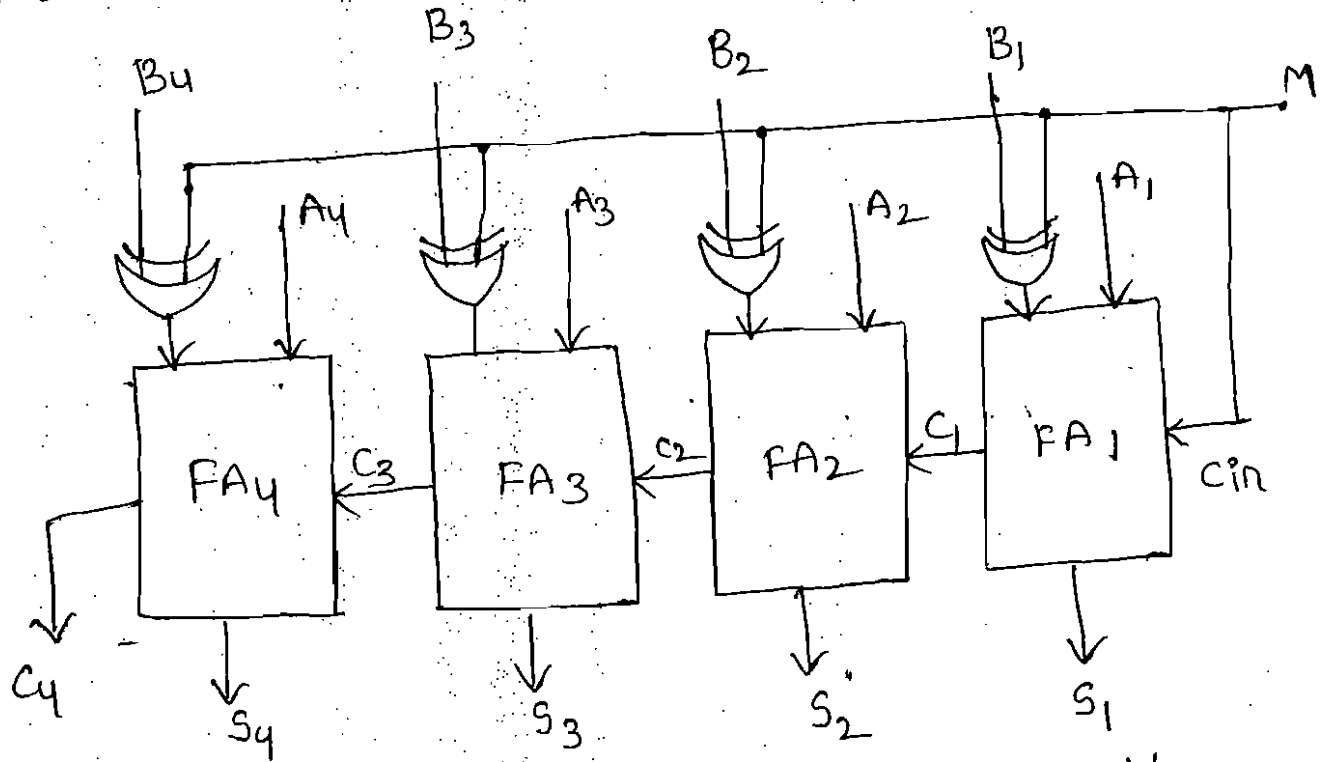
→ when M = 0, the circuit is an adder.

→ 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.

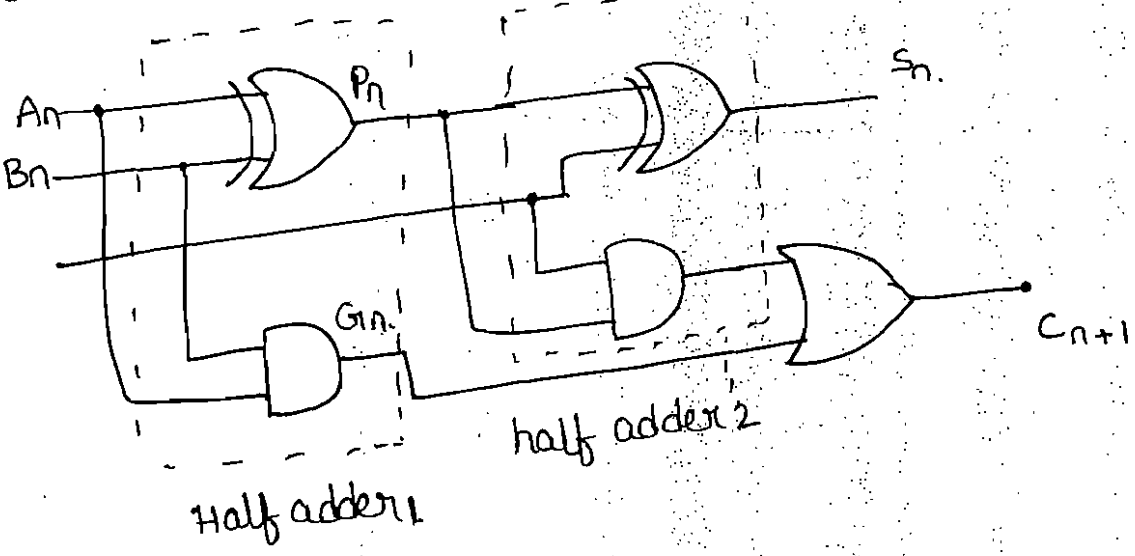


Logic diagram of a 4-bit binary - adder - subtractor.

LOOK-A-HEAD-CARRY ADDER :-

The parallel adder, the speed with which an addition can be performed is governed by the time required 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.



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 is, a carry has to be generated in this stage regardless of whether the input carry Cin is a 0 or a 1. If Gi is a carry-generation function.

$$G_i = A \cdot B.$$

The present bit as the n^{th} bit, then G_i 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 n^{th} bit, then P rewrite as a

$$P_n = A_n \oplus B_n.$$

For the final sum and carry outputs of the n^{th} 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.

$n = 1$, $C_1 = G_0 + P_0 C_0$

$$= G_0 + (A_0 \oplus B_0) C_0$$
$$= A_0 \cdot B_0 + (A_0 \oplus B_0) C_0.$$

$n = 2$

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

$n = 3$

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

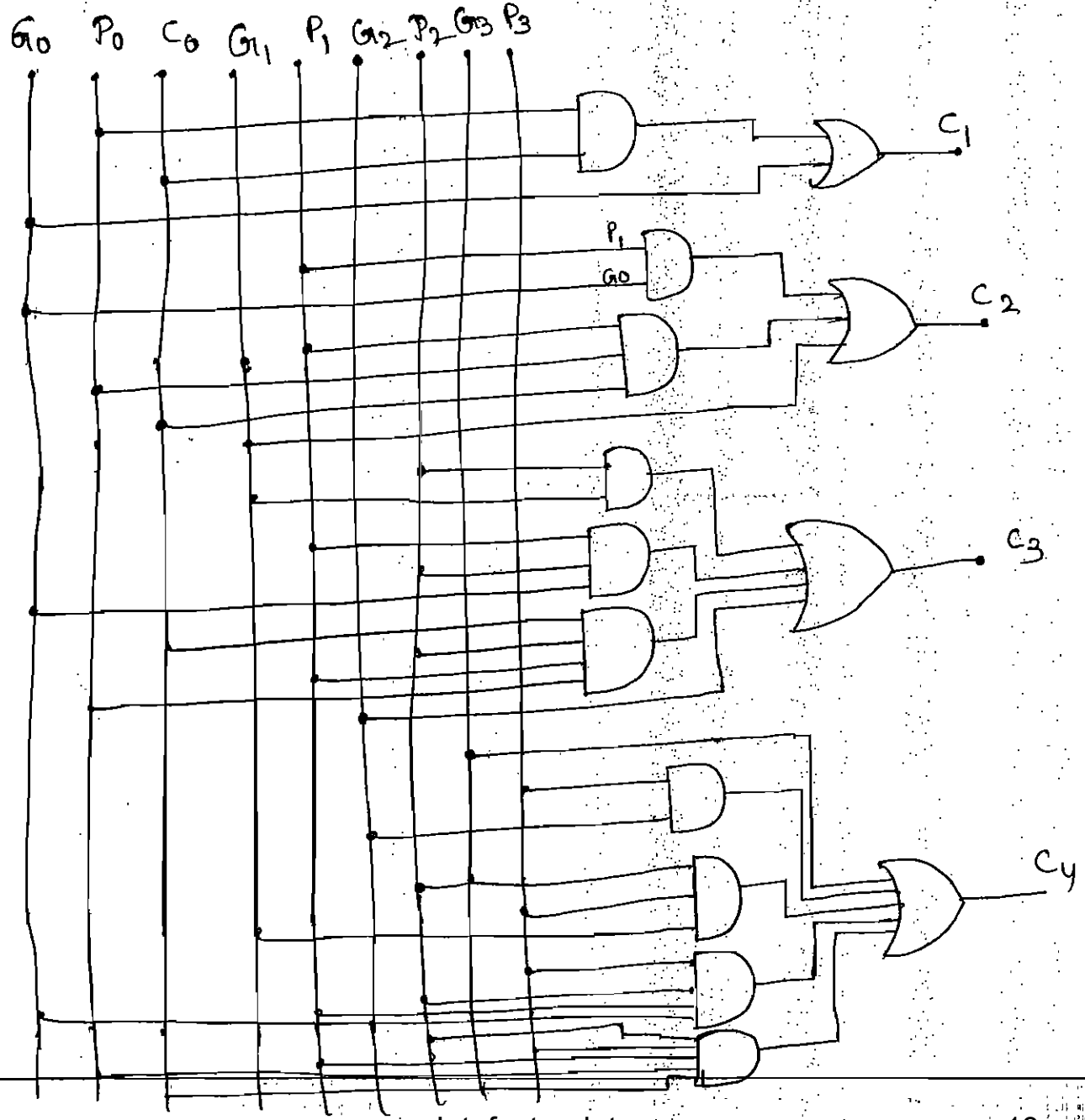
$n = 4$

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

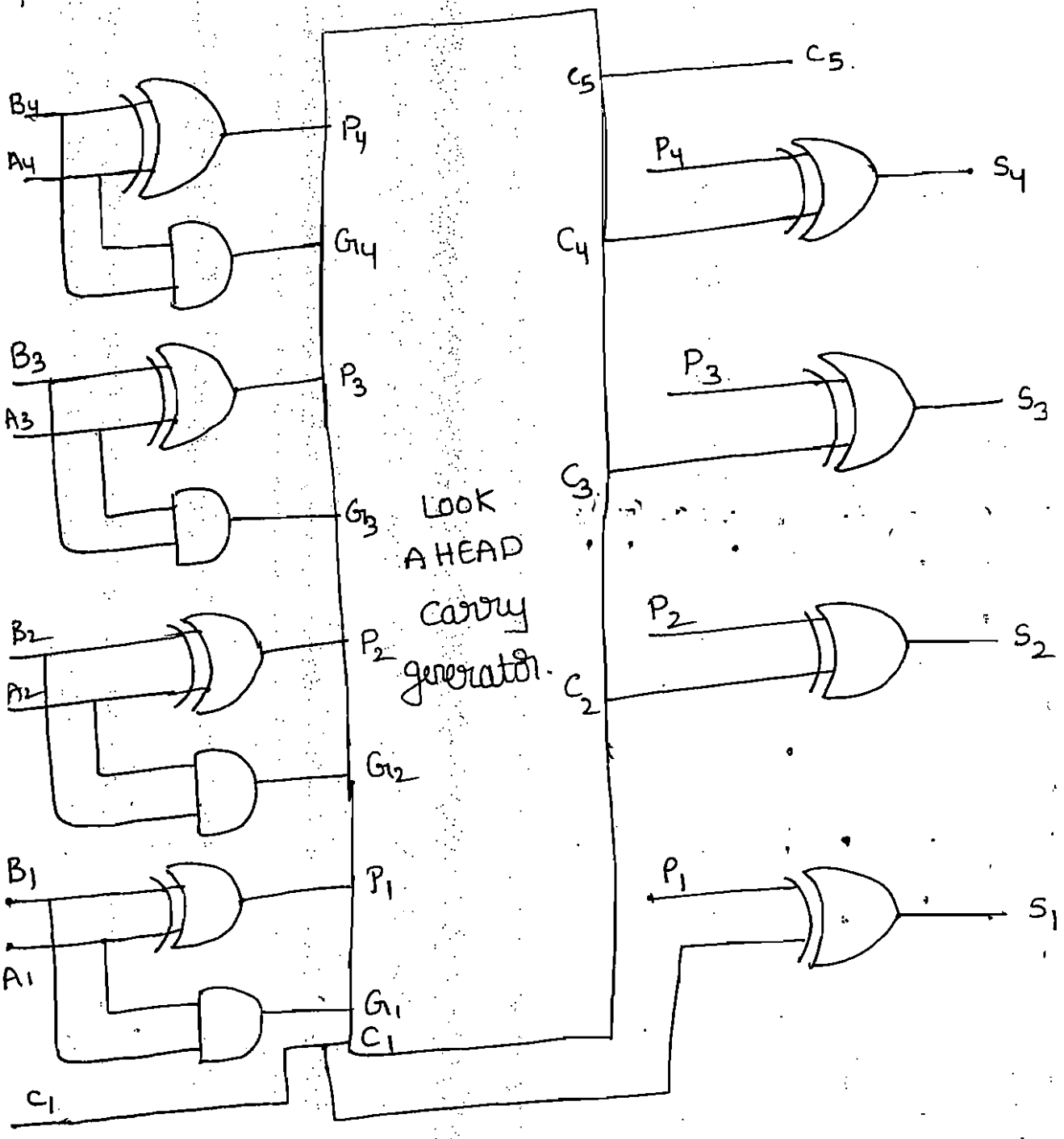
The general expression for n-stages.

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

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

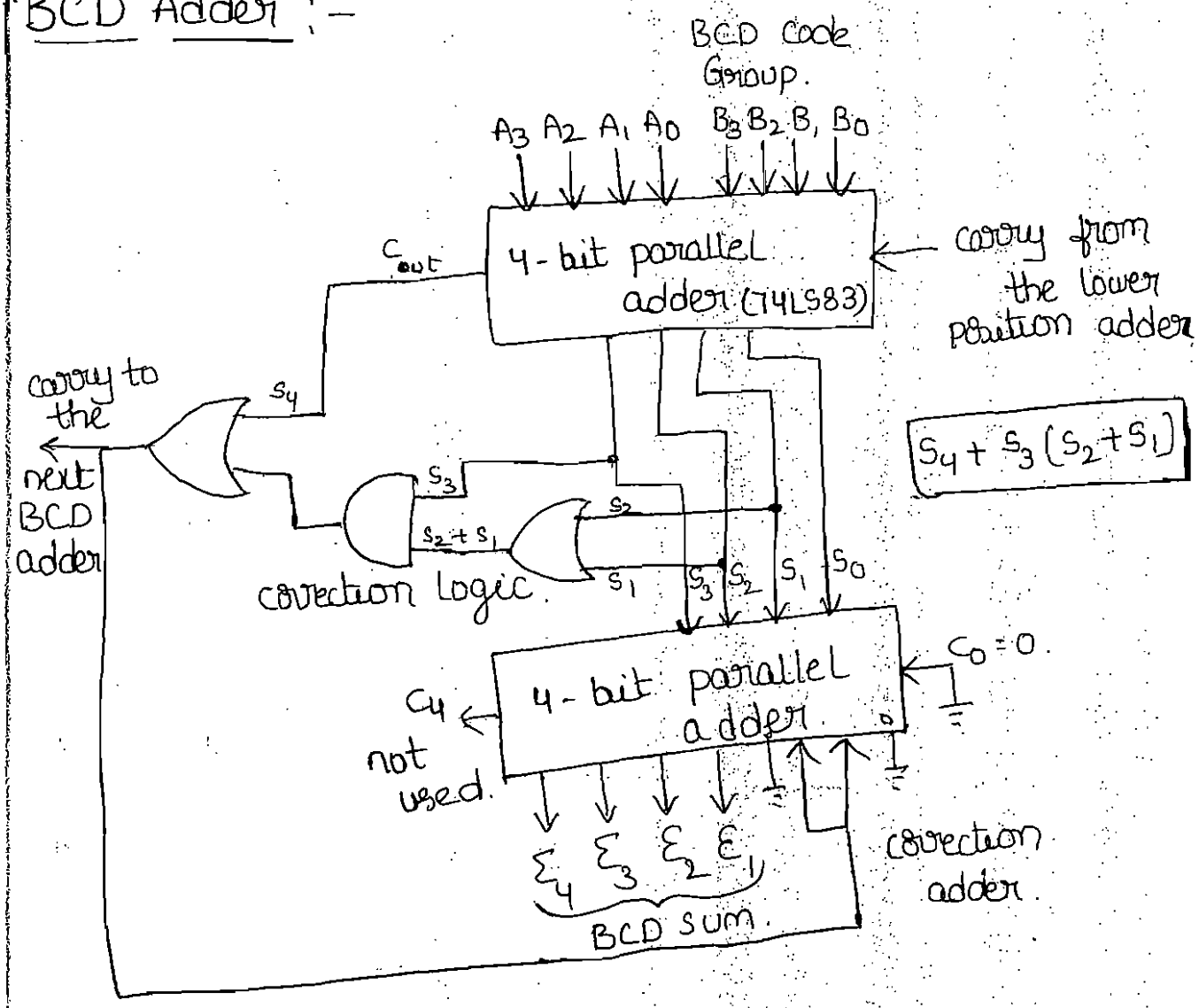


The block diagram of a 4-stage look-ahead-carry parallel adder is shown in the below figure.



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 C_4, S_3, S_2, S_1, S_0. where C_4 is taken as a S_4.

→ when both the inputs are 1001. The Sum output S_4, S_3, S_2, S_1, S_0 can range from 00000 to 10010.

→ The circuitry for a BCD adder must include the logic needed to detect whenever the sum is greater than 01001.

S_4	S_3	S_2	S_1	S_0	Decimal number
0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
1	0	0	0	0	16
1	0	0	0	1	17
1	0	0	1	0	18

→ In above Table shows the cases for greater than 1001.

The Sum will be high → wherever $S_4 = 1$,

→ wherever $S_3 = 1$ and either S_2 & S_1 & both are 1.

Then $X = S_4 + S_3(S_2 + S_1)$.

wherever $X = 1$, it is necessary to add the 0110 to the Sum bits.

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 $\varepsilon_3 \varepsilon_2 \varepsilon_1 \varepsilon_0$.

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

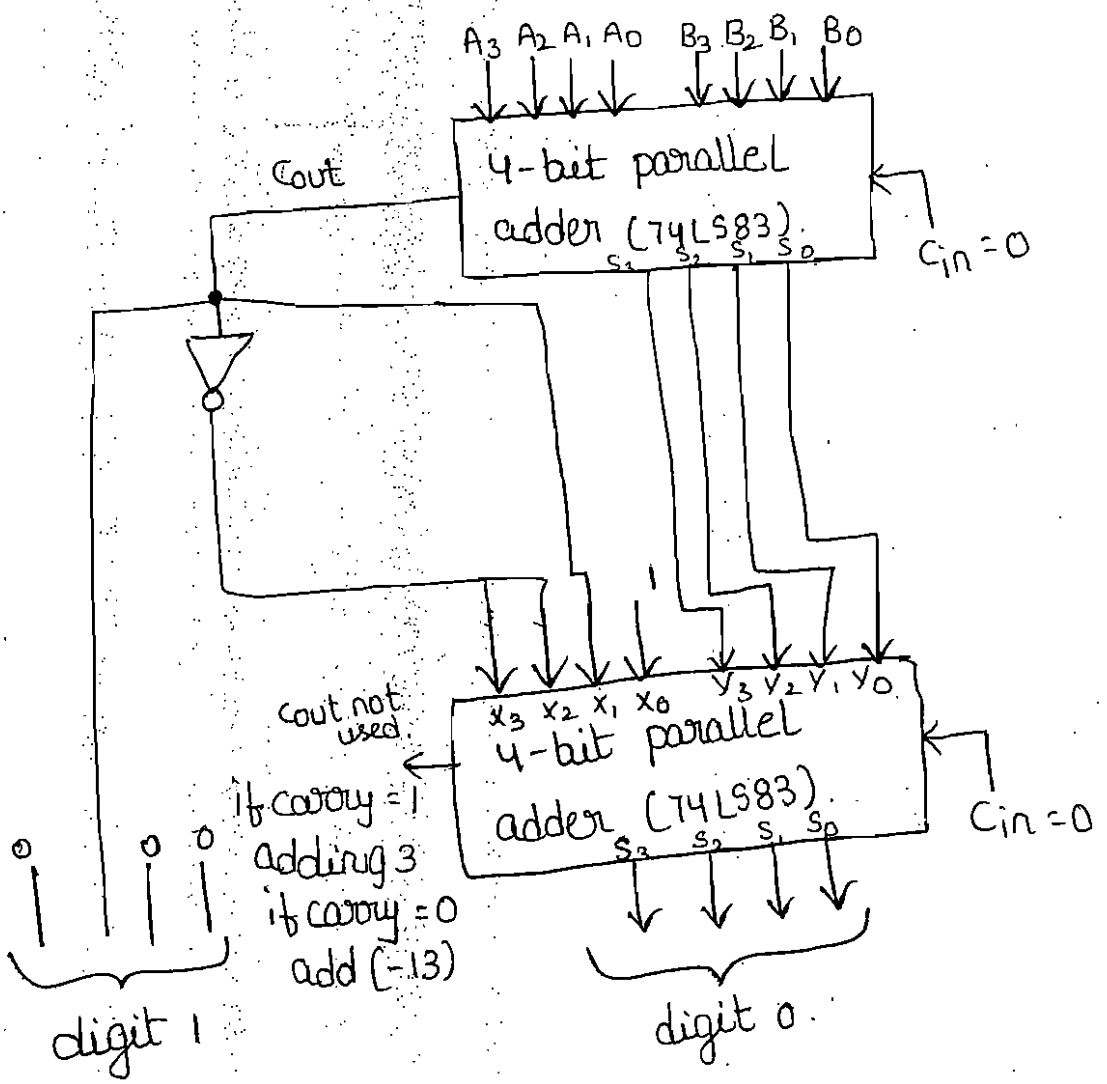
In such cases $\varepsilon_3 \varepsilon_2 \varepsilon_1 \varepsilon_0 = S_3 S_2 S_1 S_0$. Two or more BCD adders can be connected in cascade when two or more digit decimal numbers are to be added. The carry-out of the first BCD adder is connected as the carry-in of the second BCD adder.

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.



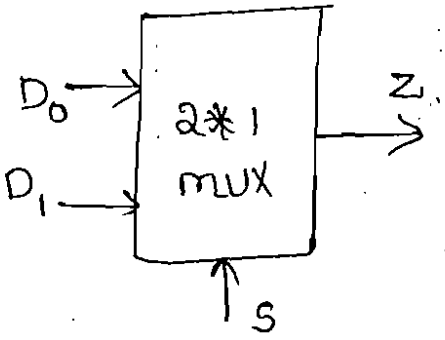
The final Answer in XS-3 form.

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 :-

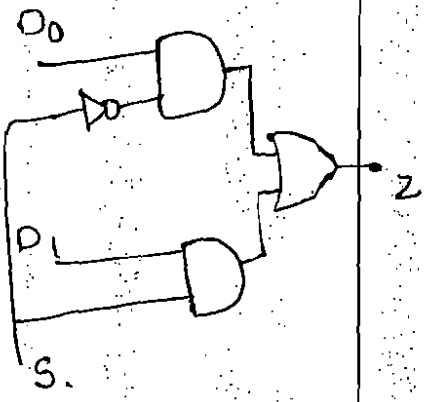
In 2-input multiplexer have 2-inputs they are D_0 and D_1. and one select line S. and output is Z.



Block diagram.

S	Z
0	D_0
1	D_1

Truth table.



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

Logic diagram.

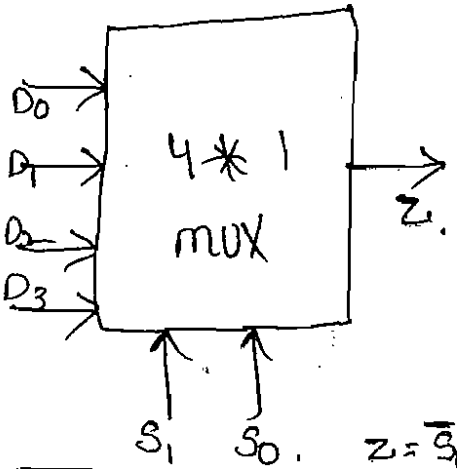
The logic levels applied to the S inputs determines which AND gate is enabled. So that its data input passes through the OR gate to the output.

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.

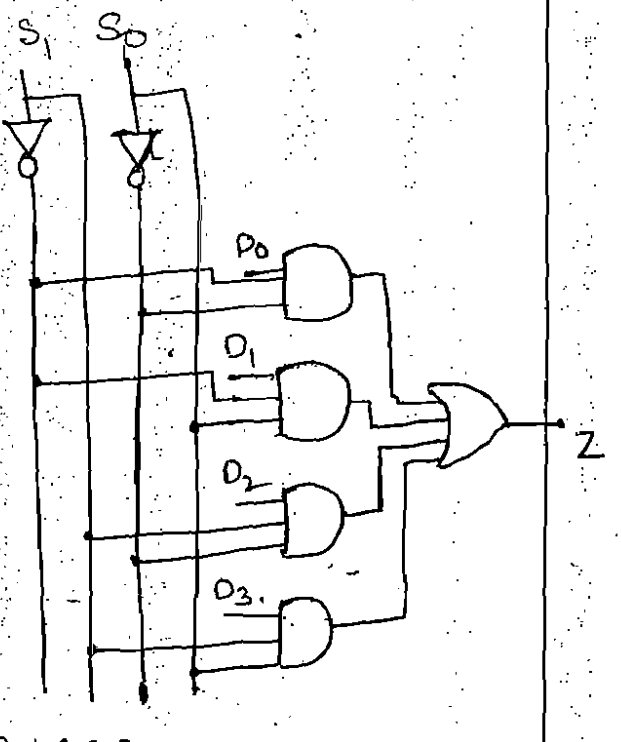
Basic 4-input multiplexer :-

Block diagram



Truth table

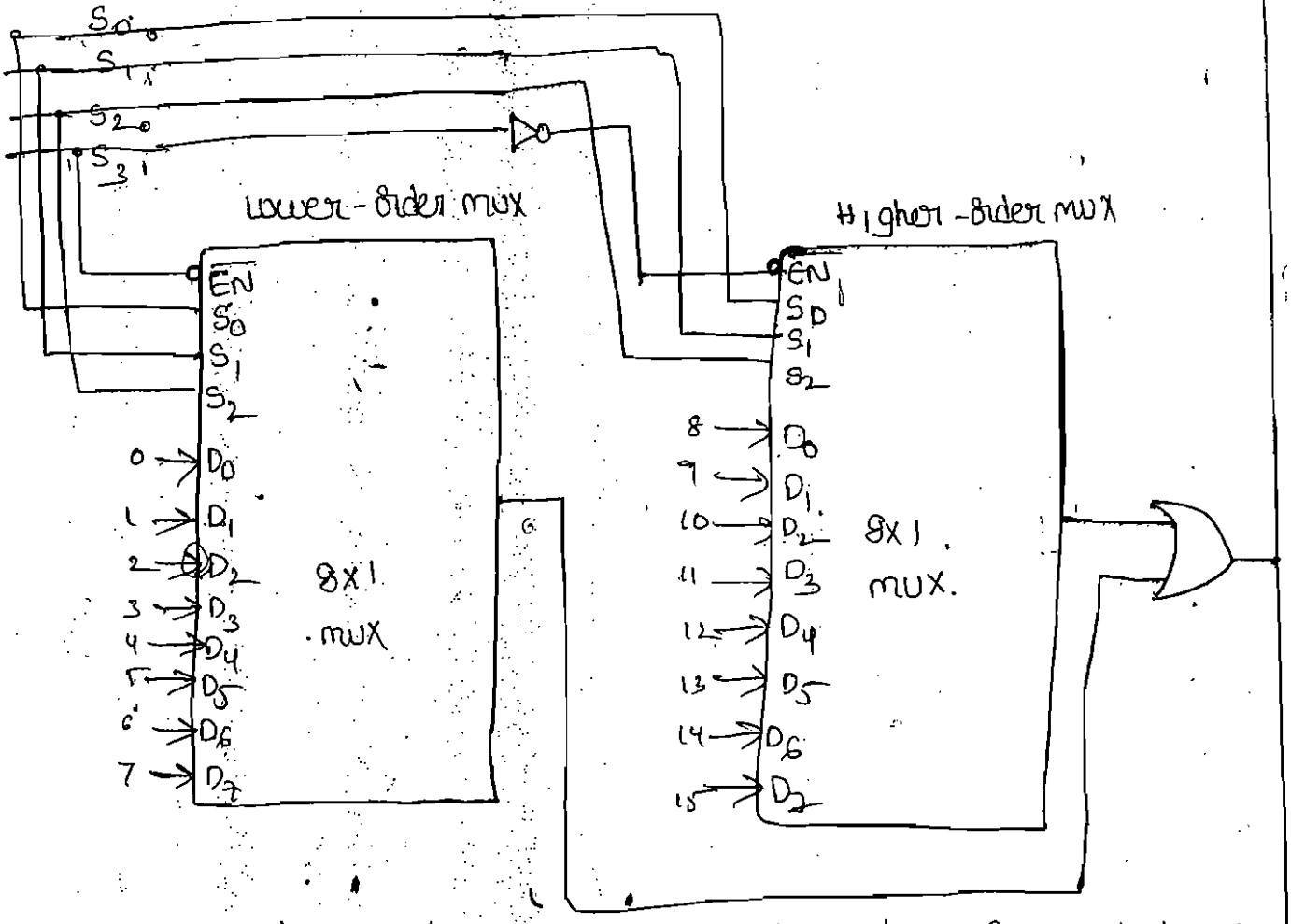
S_1	S_0	Z
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



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

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

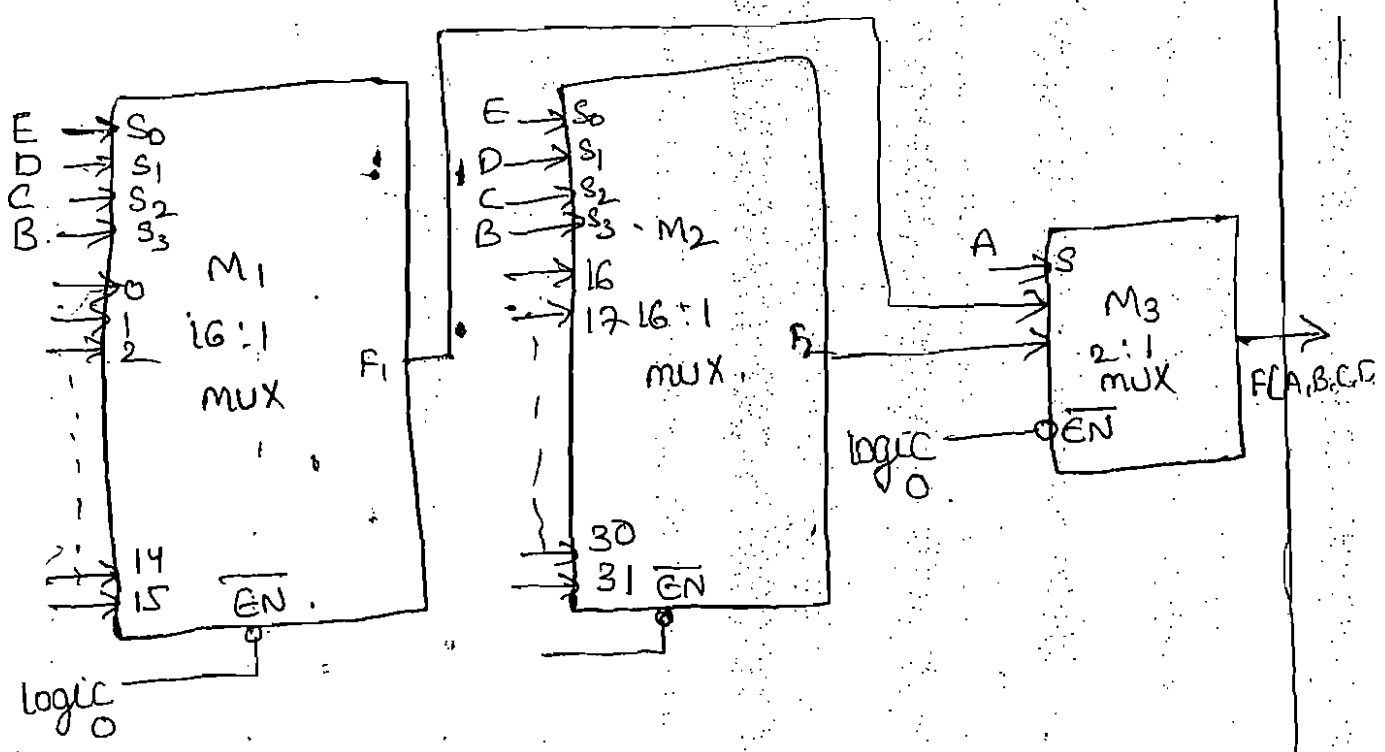
To use two 8-inputs multiplexers to get a 16-inputs multiplexers one OR gate and one inverter are also required. The four select inputs S_3, S_2, S_1 and S_0 and will select one of the 16 inputs to pass through to X. The S_3 input determines which multiplexer is enabled. When $S_3 = 0$, the left multiplexer 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 16x1

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 $F = F_2$.

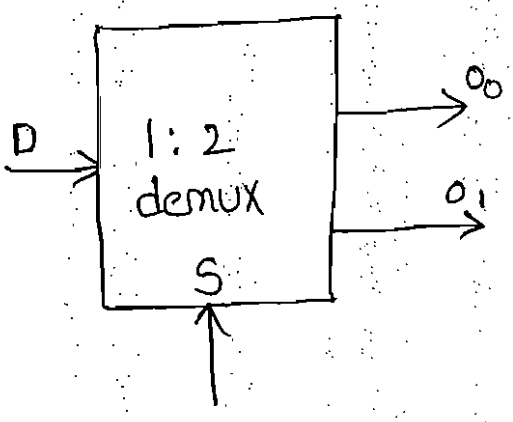


DEMULTIPLEXERS :-

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 :-

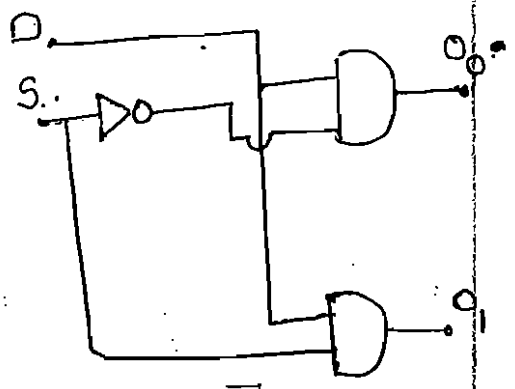
The input data line goes to all of the AND gates. The select line S enable only one gate at a time, and the data appearing on the input line will pass through the selected gate to the associated output lines.



Block diagram

S	O_0	O_1
0	0	D
1	D	0

Truth table



$O_0 = D\bar{S}$

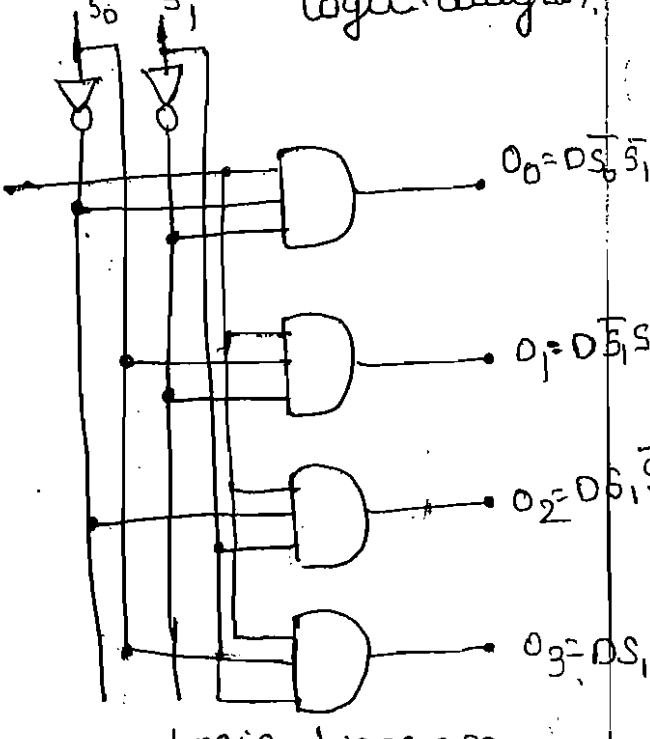
$D_1 = DS$

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.

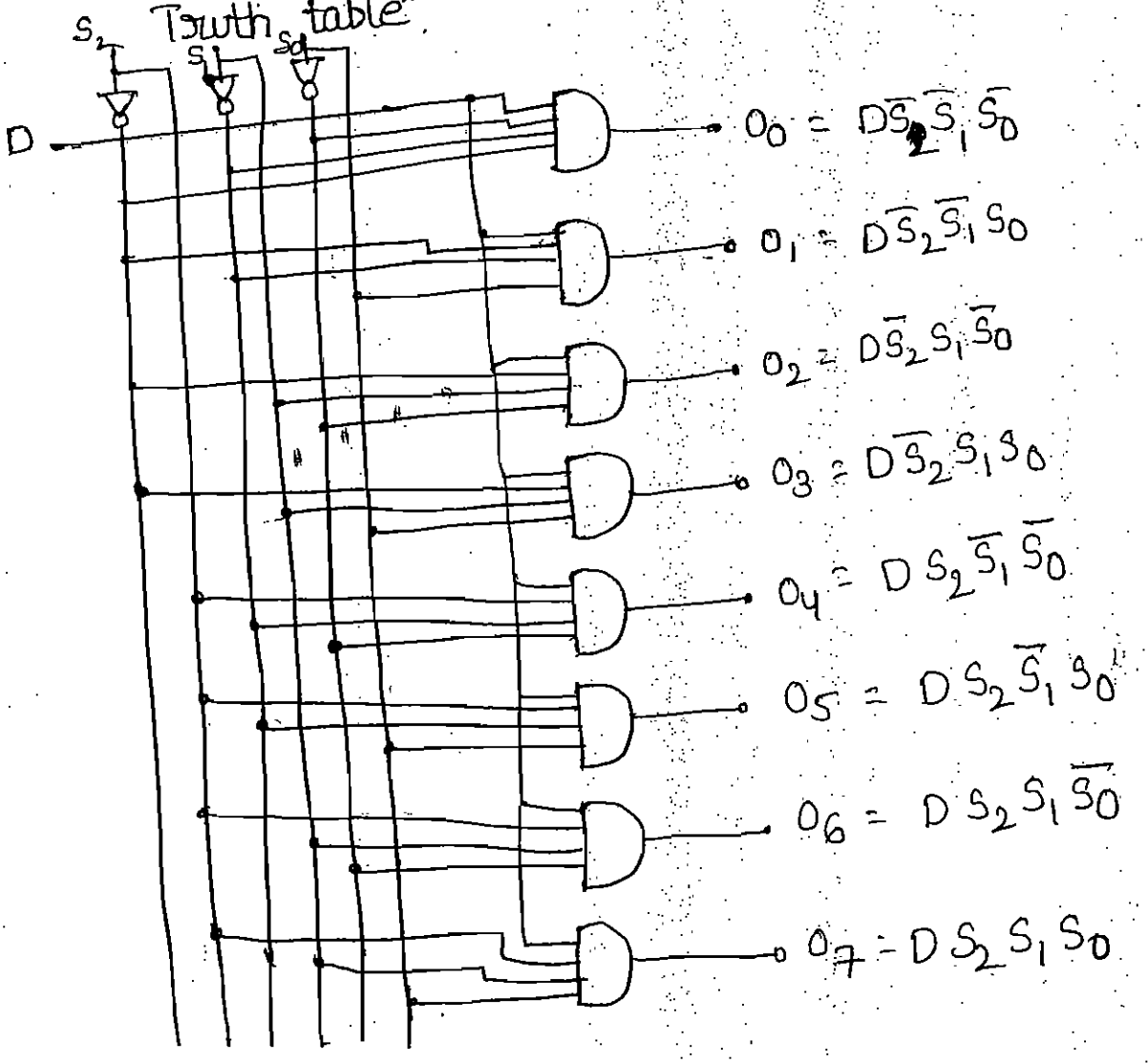


Logic diagram.

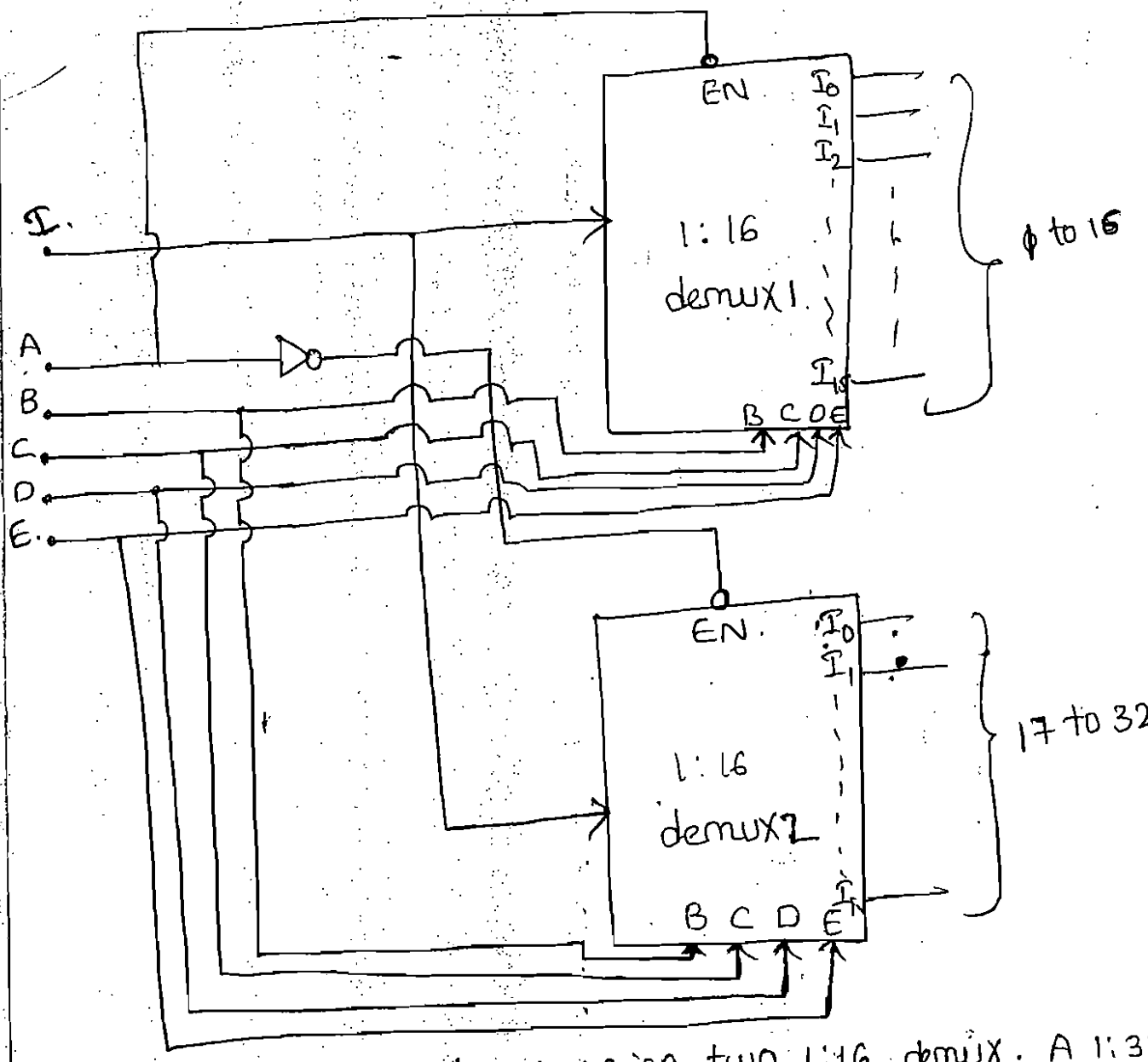
1-line to 8-line demultiplexer:-

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.



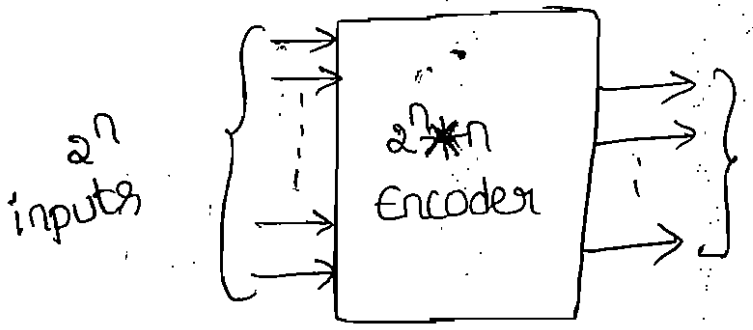
design of 1:32 demux using two 1:16 demux.



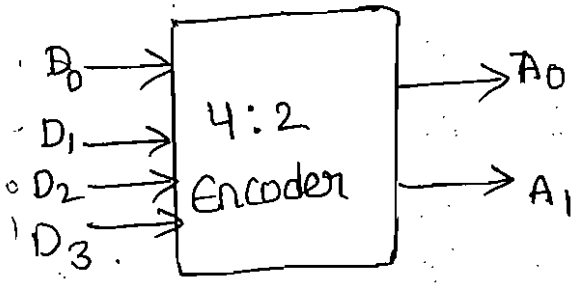
To obtain 1:32 demux using two 1:16 demux. A 1:32 demux has 32 data outputs. So it requires five data select lines. Since 1:16 demux has only four select inputs. the inputs B, C, D, E are connected to the data select lines of both the 1:16 demuxes and the most significant input A is connected to the 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 :-



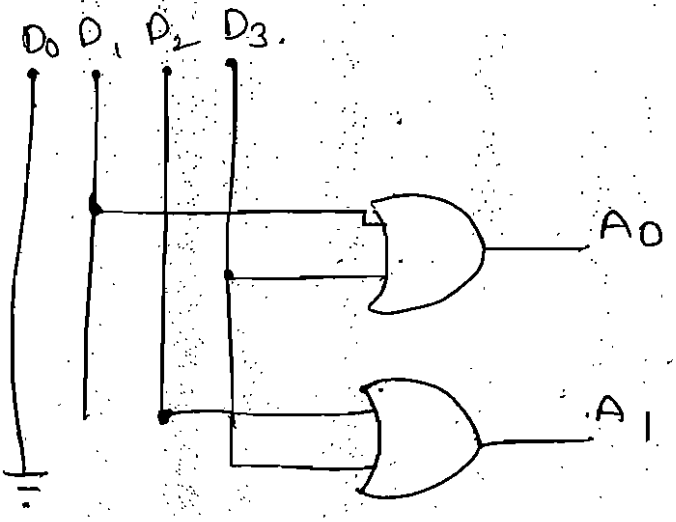
Block diagram.

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

Truth table.

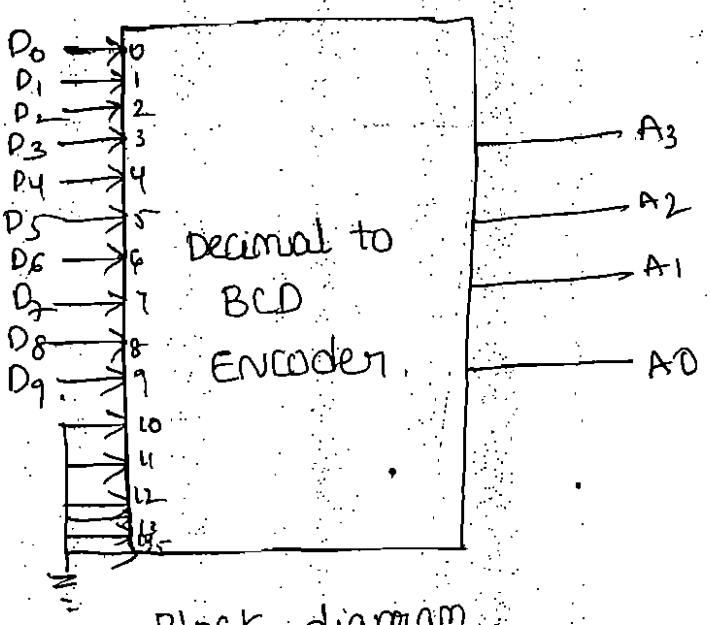
$A_1 = D_2 + D_3$

$A_0 = D_1 + D_3$.



Decimal to BCD Encoder :-

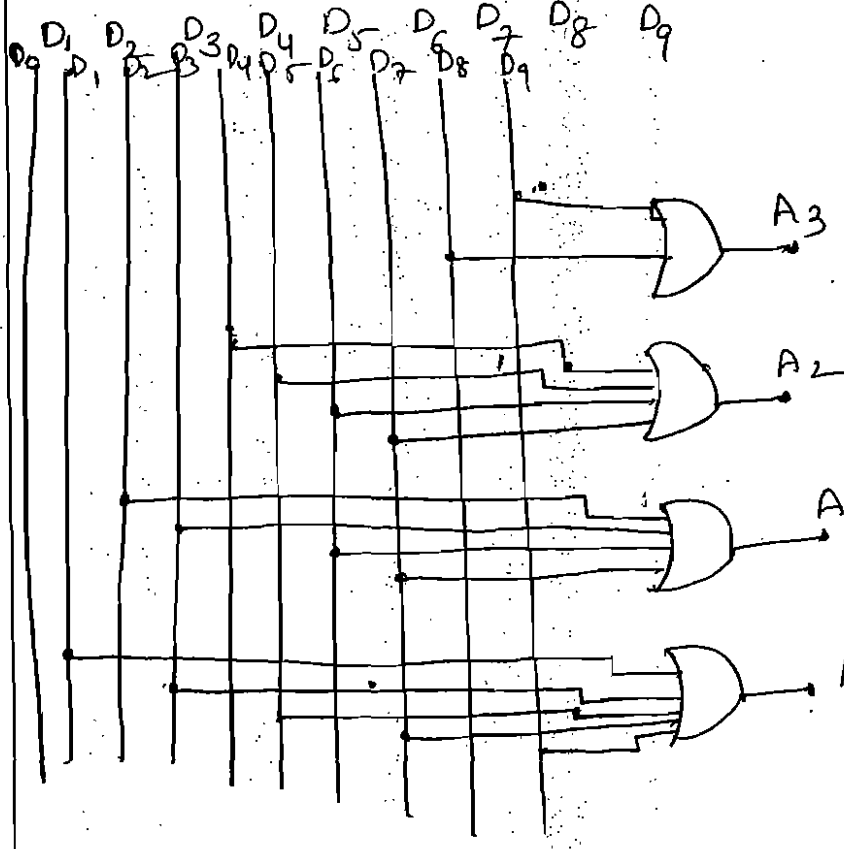
In this type of encoder has 10 inputs - one for each decimal digit, and 4 outputs corresponding to the BCD code.



Block diagram.

inputs. Decimal		Binary output 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.



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

$A_1 = D_2 + D_3 + D_6 + D_7$

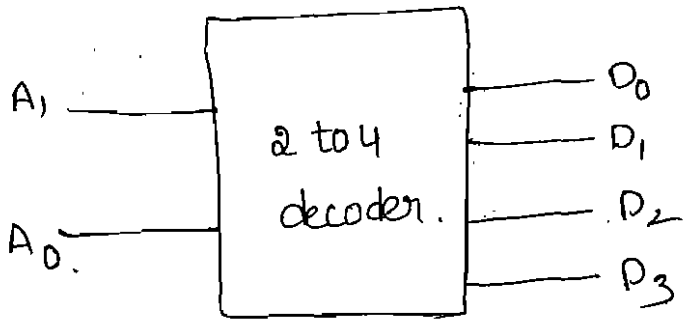
$A_2 = D_4 + D_5 + D_6 + D_7$

$A_3 = D_8 + D_9$.

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.

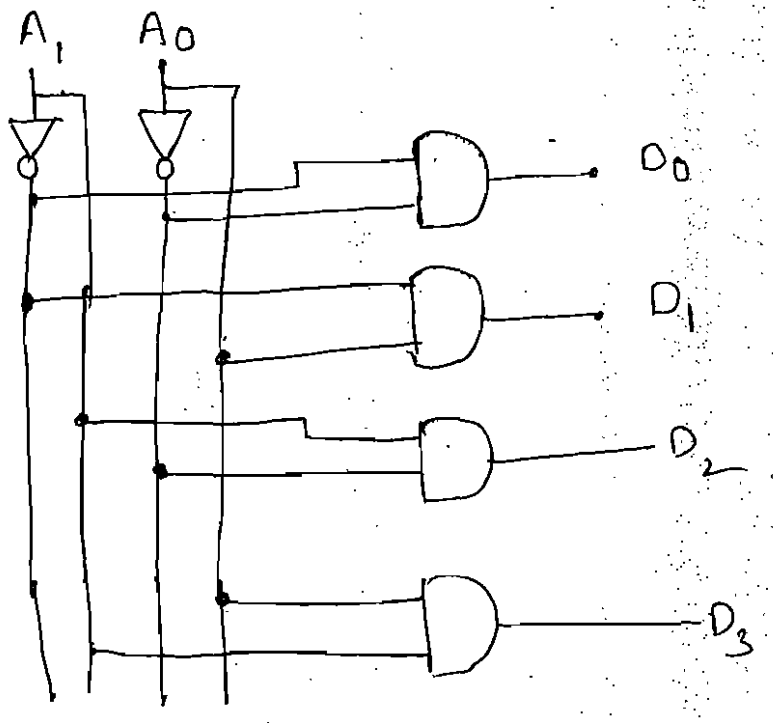
2-4 line decoder :-



Block diagram.

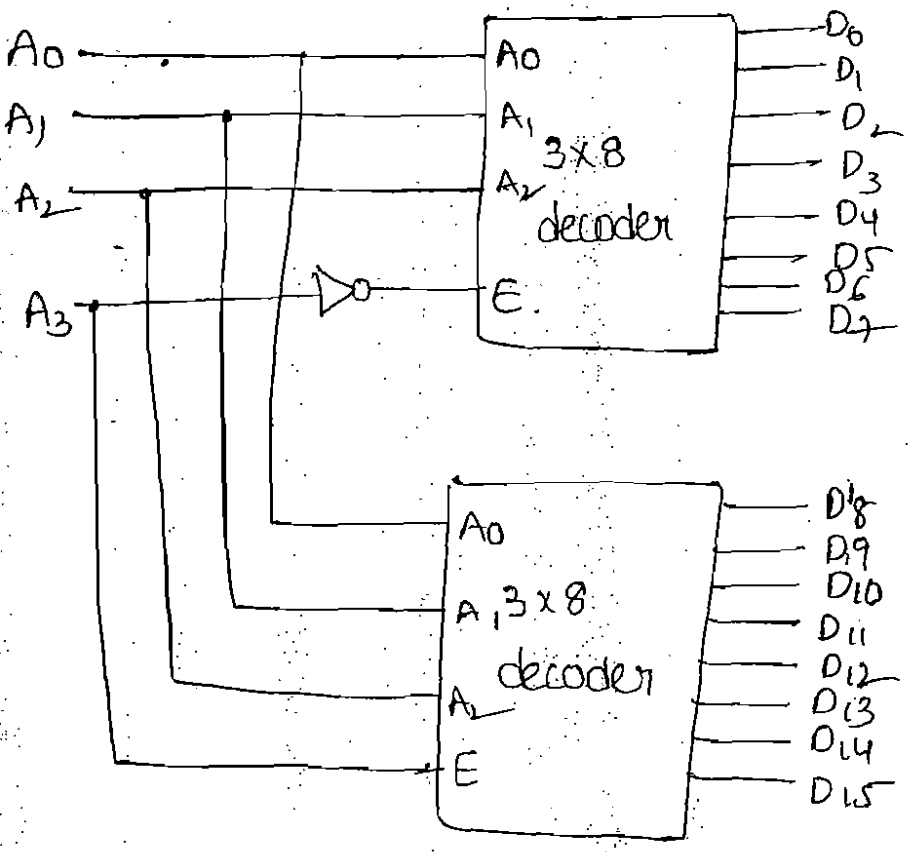
A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Truth table.



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

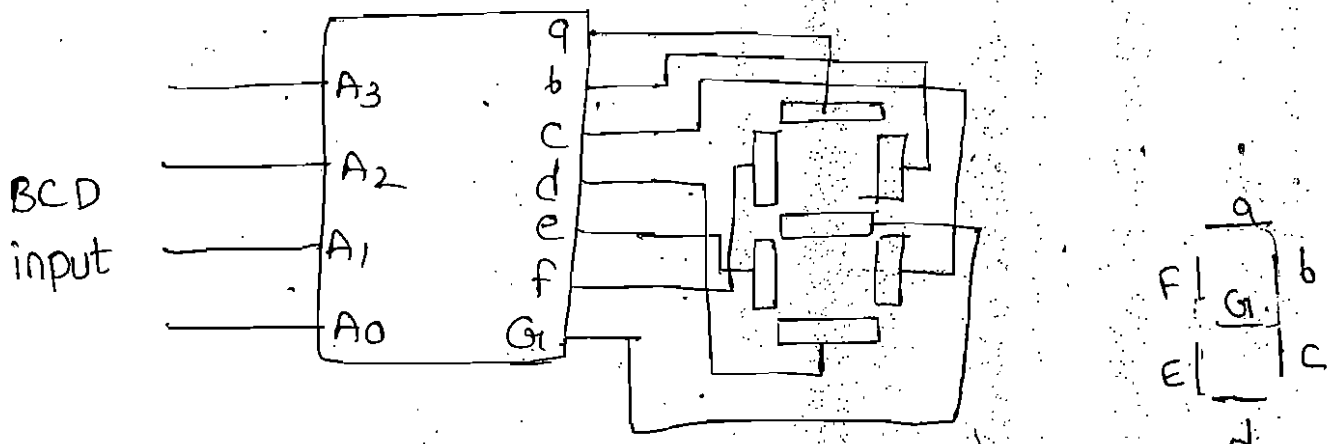
Decoders with enable inputs can be connected together to form a larger decoder. To obtain a 4-to-16 decoder it requires two 3-to-8 decoders. The most significant input bit A_3 is connected through an inverter to $\bar{E}$ on the upper decoder and directly to E on the lower decoder. Thus A_3 is low, the 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 A_3 A_2 A_1 A_0	Seven segment code 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

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

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

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

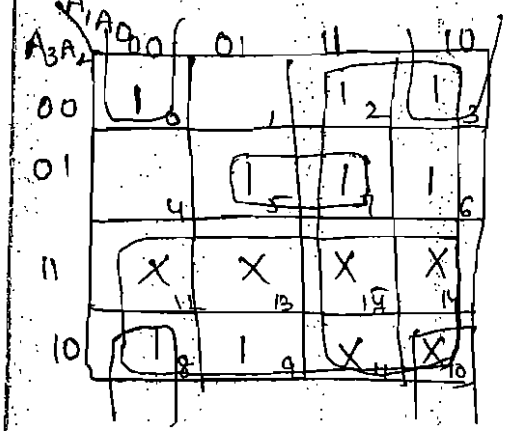
$d = \Sigma m(0,2,3,5,6,8,9) + d(10,11,12,13,14,15)$.

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

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

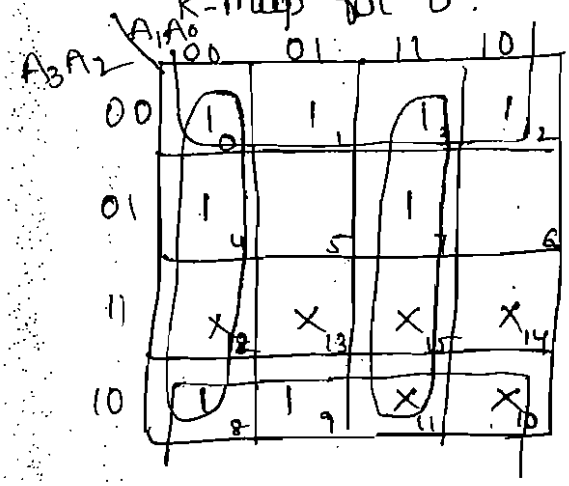
$g = \Sigma m(2,3,4,5,6,8,9) + d(10,11,12,13,14,15)$.

K-map for a.

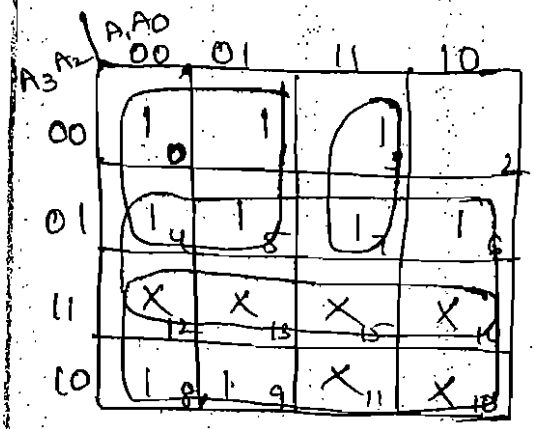


$a = A_1 + A_3 + \overline{A_2}\,\overline{A_0} + \overline{A_3}A_2A_0$

K-map for b.

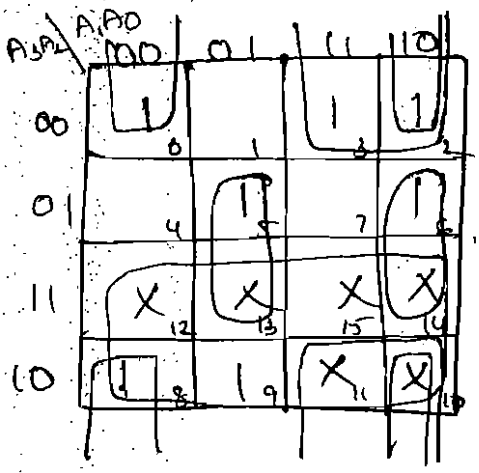


$b = \overline{A_2} + A_1\overline{A_0} + A_1A_0$



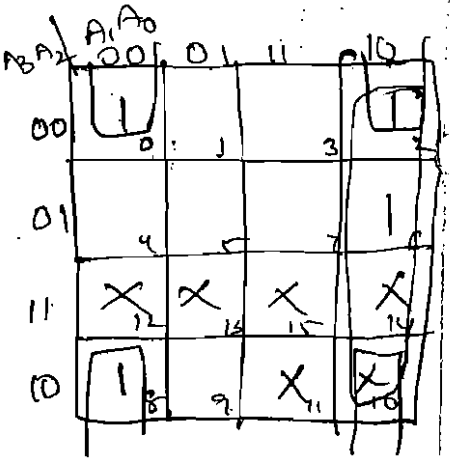
K-map for c

$c = A_2 + A_3 + \overline{A_3}\,\overline{A_1} + \overline{A_3}A_1A_0$



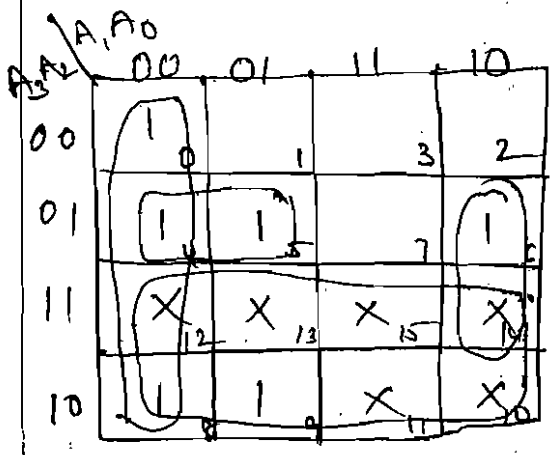
K-map for d

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

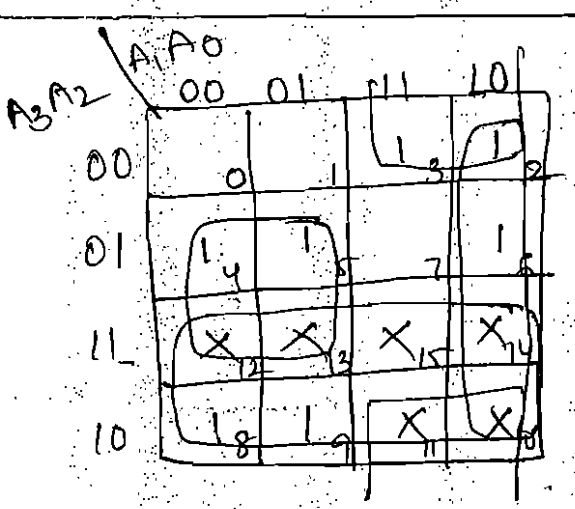


K-map for e

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



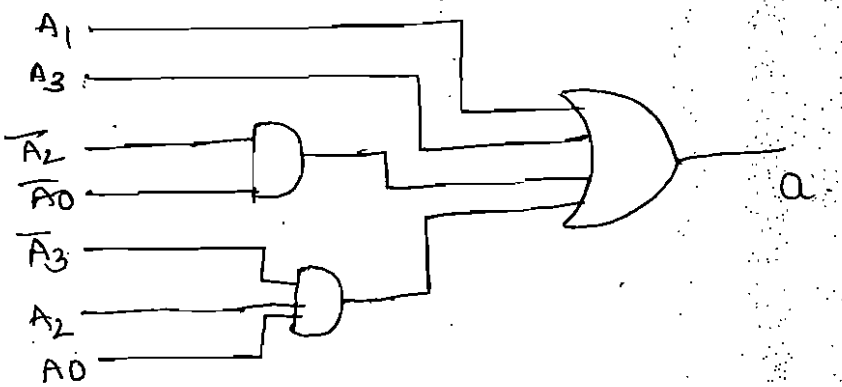
K-map for f.

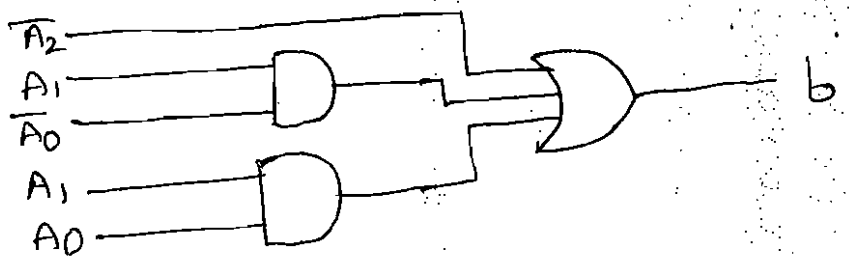


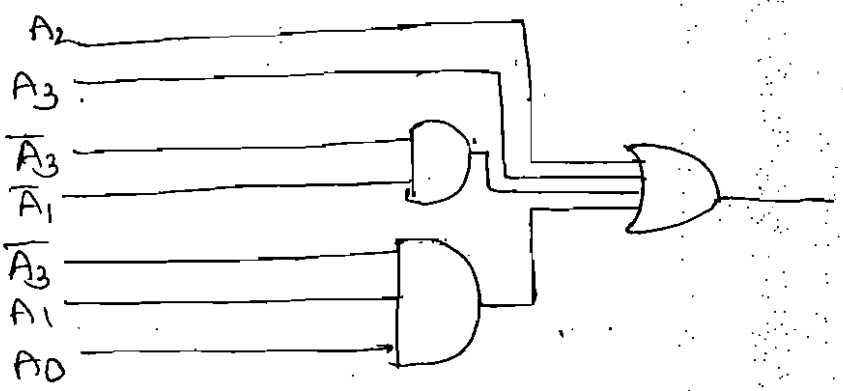
K-map for G1.

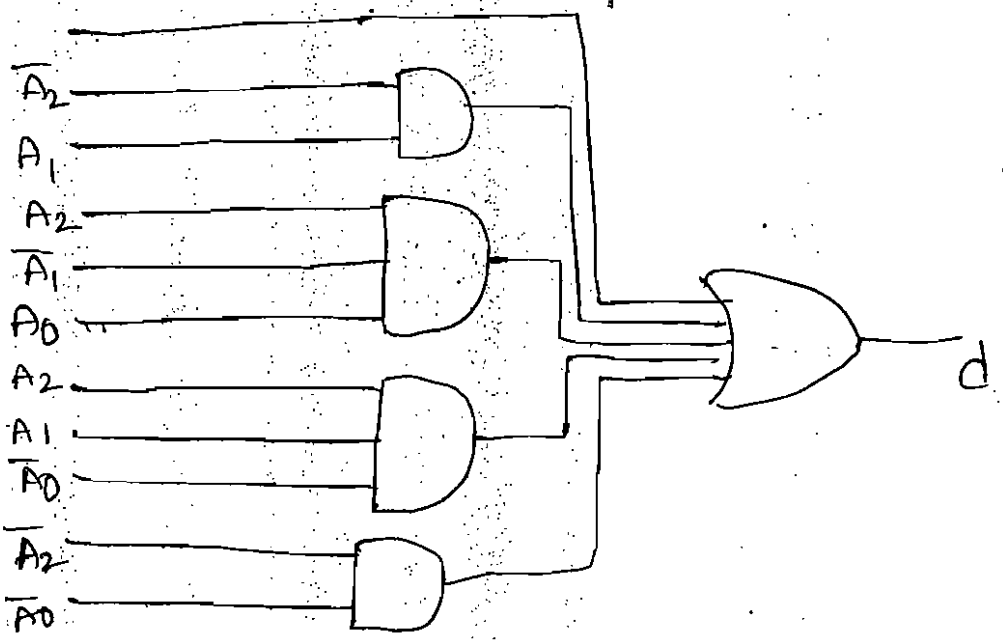
$f = \bar{A}_1\bar{A}_0 + A_3 + \bar{A}_3A_2\bar{A}_1 + A_2A_1\bar{A}_0$

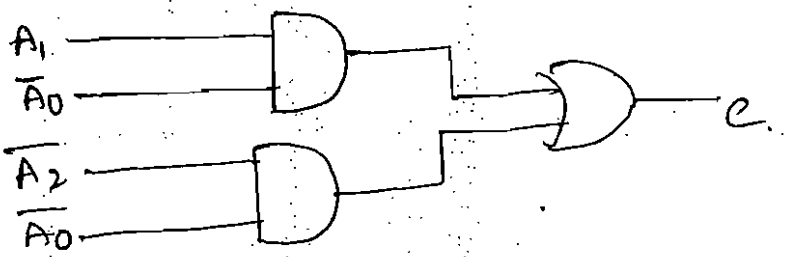
$G_1 = A_3 + A_1\bar{A}_0 + A_2\bar{A}_1 + \bar{A}_2A_1$

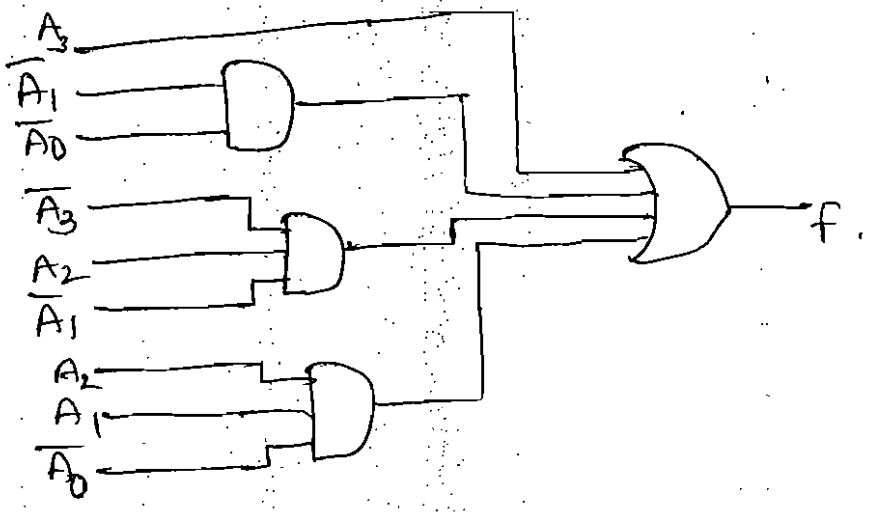


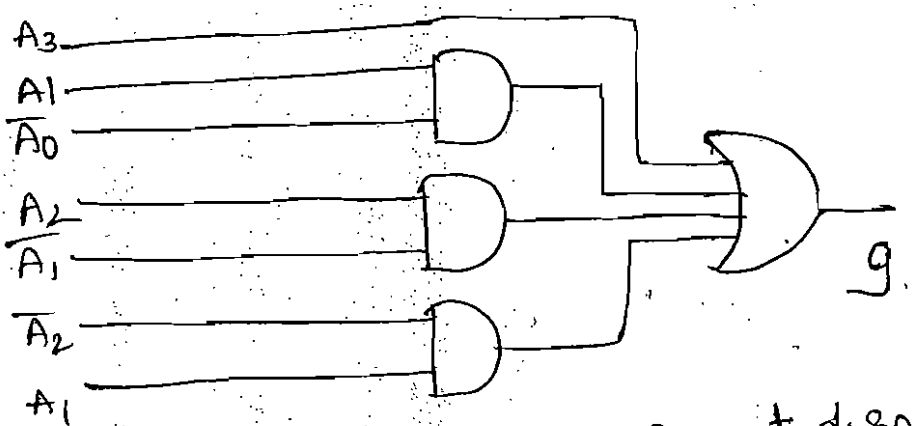








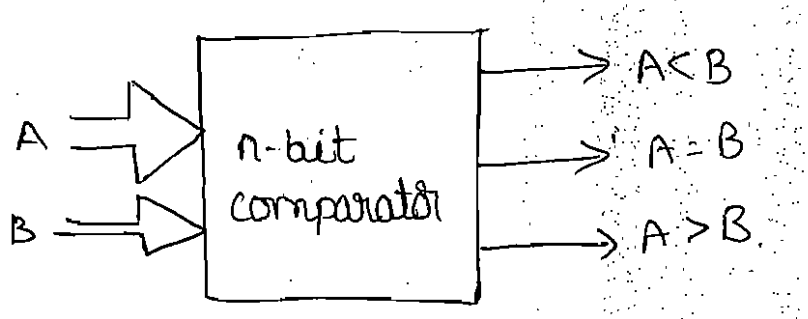




diagrams for seven segment display.

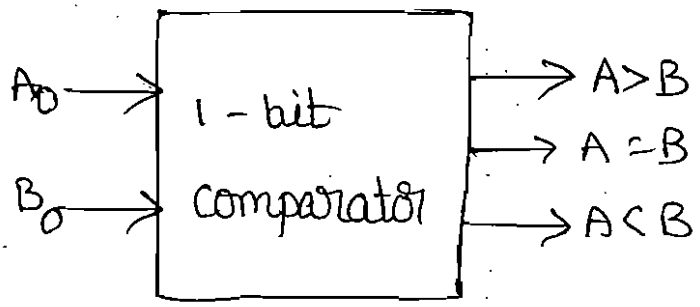
Comparator :-

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 comparsion, 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

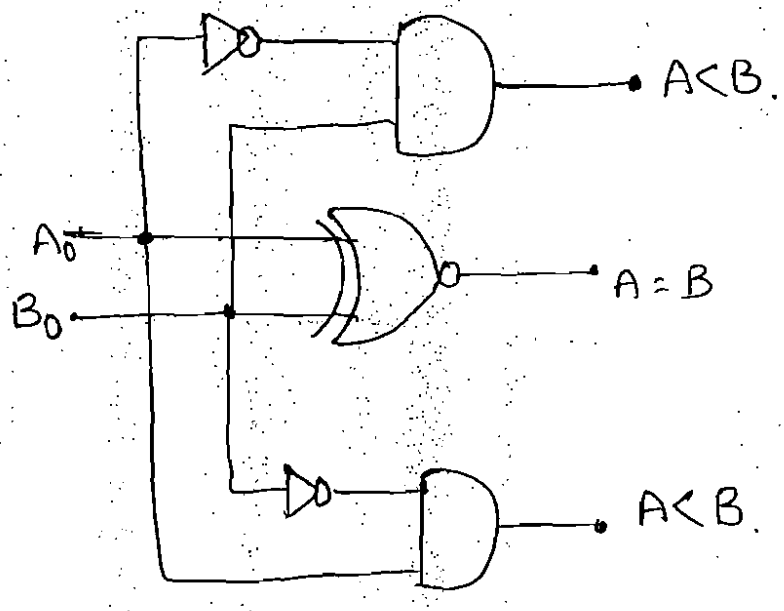
Truth table

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 $A = A_1 A_0$ and $B = B_1 B_0$.

→ 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 $A > B$.

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

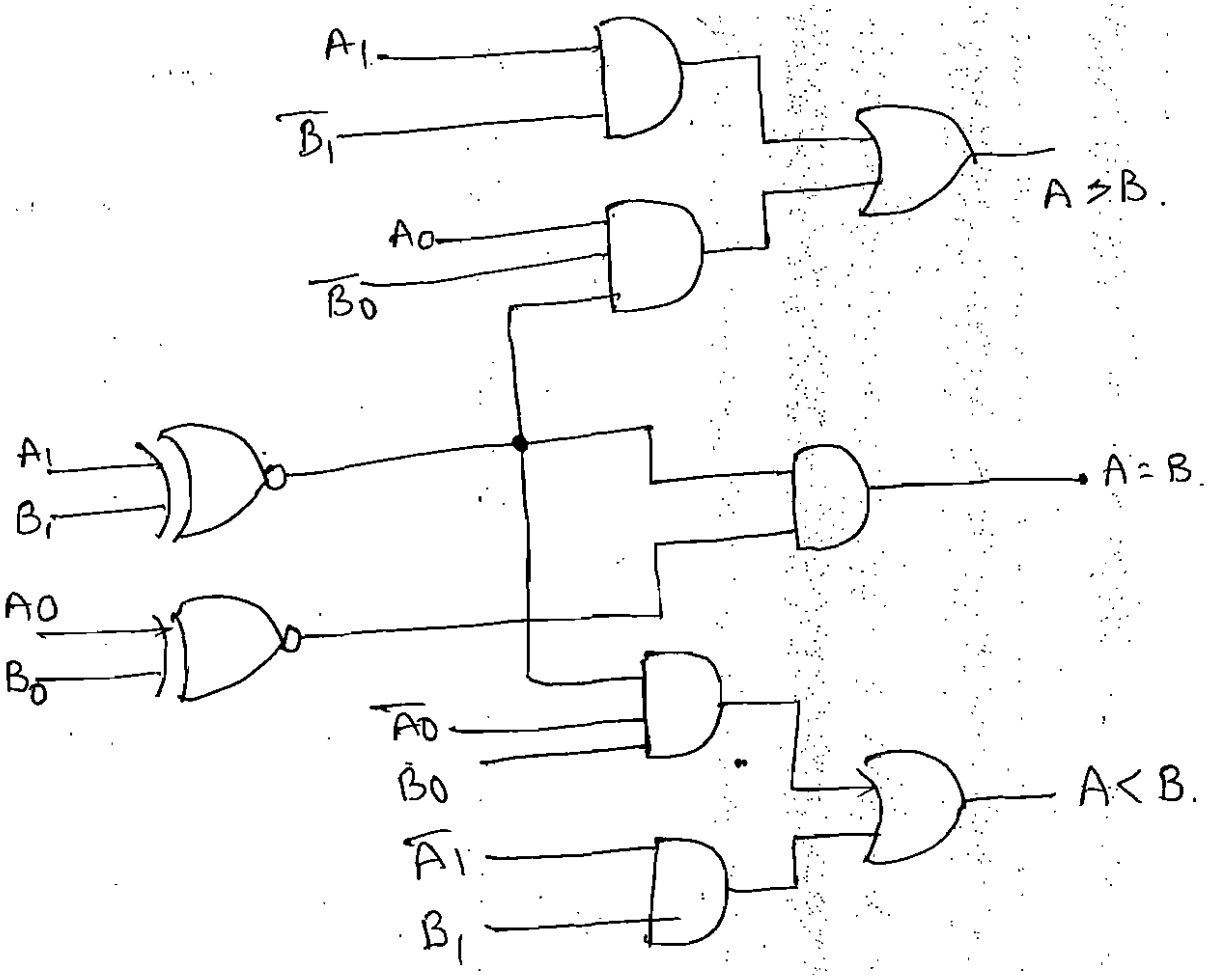
→ if $B_1 = 1$ and $A_1 = 0$ then $A < B$ or

→ 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 = 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 $B_0 = 0$.

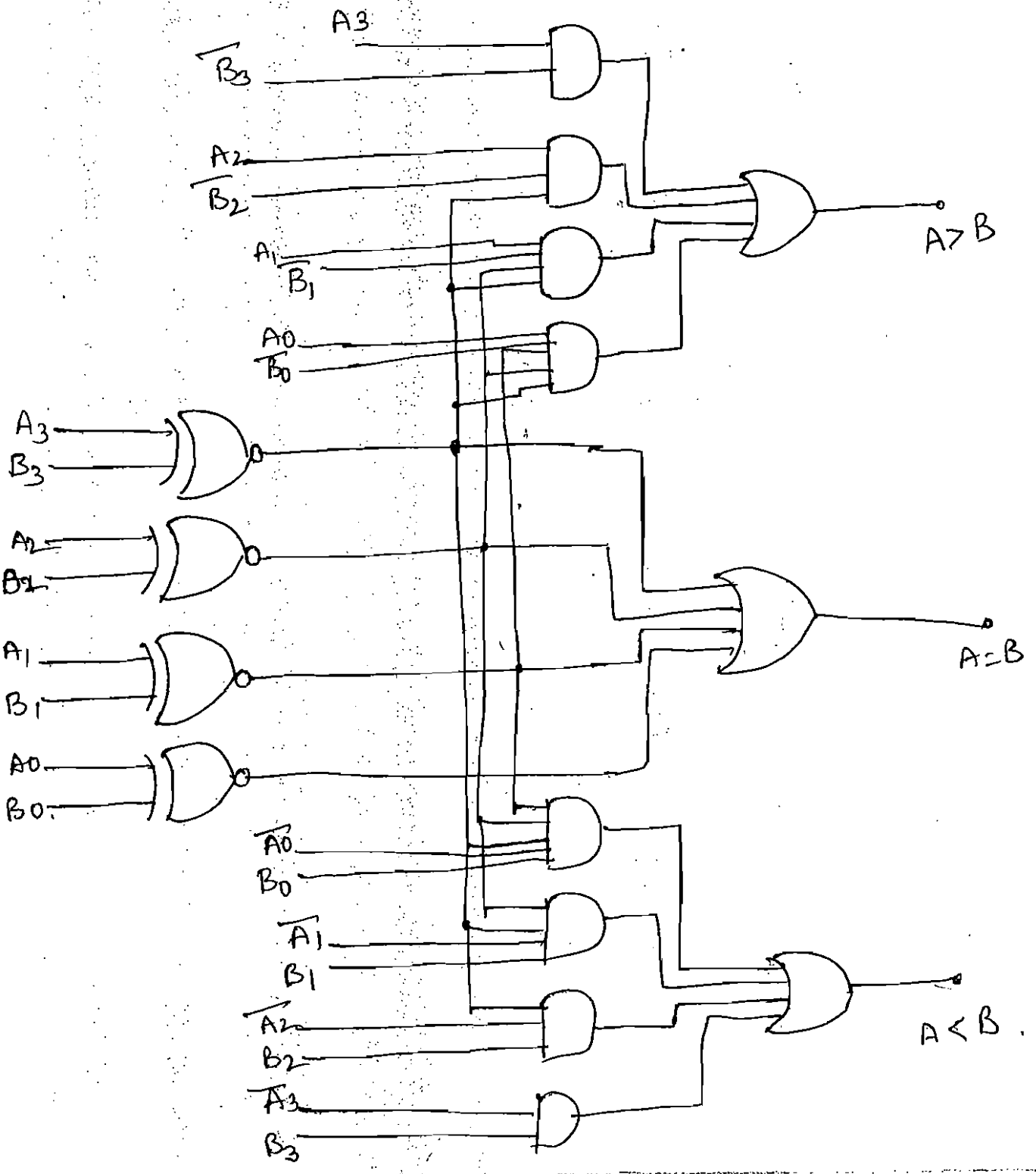
$(A>B) = A_3\overline{B_3} + (A_3 \odot B_3) A_2\overline{B_2} + (A_3 \odot B_3)(A_2 \odot B_2) A_1\overline{B_1}$

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

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

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

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



Logic diagram of 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 valid 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	1

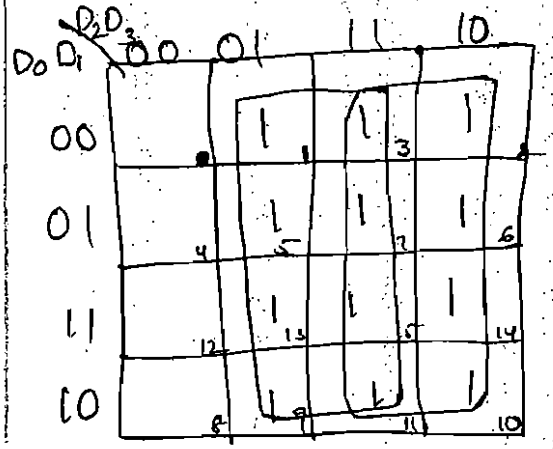
$V = D_0 + D_1 + D_2 + D_3$.

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

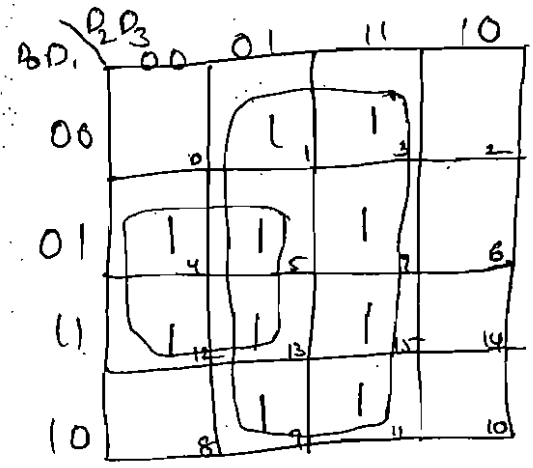
$$A = \Sigma m(1,2,3,5,6,7,9,10,11,13,14,15)$$

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

K-map for A

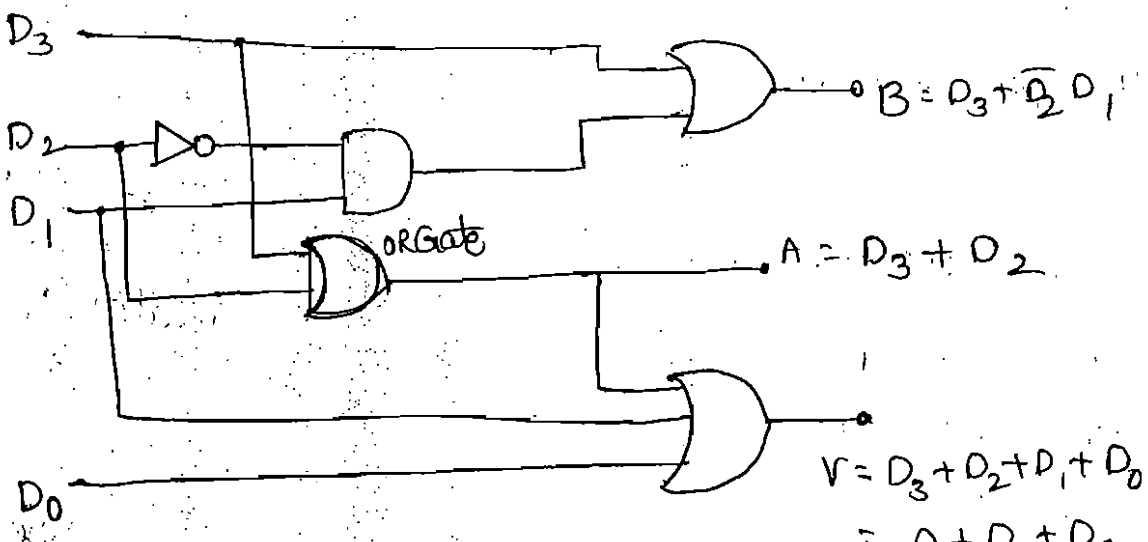


K-map for B.



$A = D_3 + D_2$

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



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 intercomected on chip. This IC is said to be programmable because the specific function IC is determined by Intercomecting 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 burung-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 :-

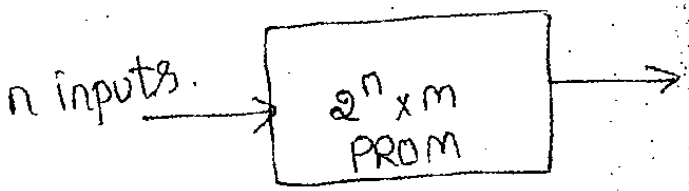
→ Erasable, programmable ROM
→ programmed by storing charge on insulated gates.
→ Erasable with ultraviolet light
→ non-volatile.

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 :-

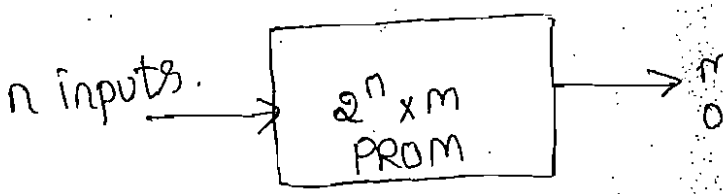
→ Erasable, programmable ROM
→ programmed by storing charge on insulated gates.
→ Erasable with ultraviolet light
→ non-volatile.

EEPROM :-

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

Programmable ROM :-

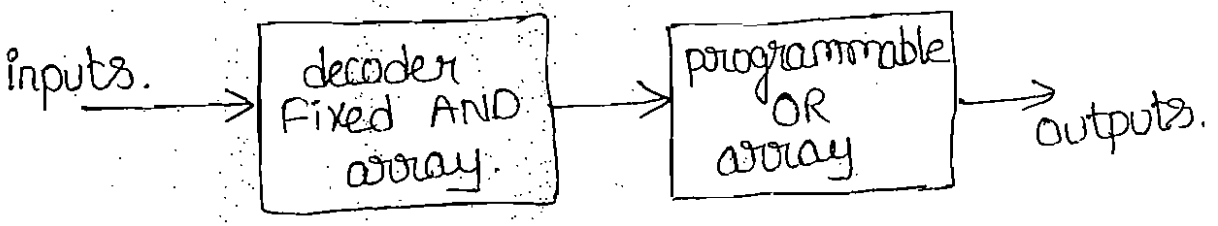
→ It includes both the decoder and the OR Gates with a single IC package. The following figures shows the block diagram and logic construction using 16X2 ROM.
→ It consists of n input lines and m output lines.
→ Each bit combination of the input variables is called an address.
→ Each bit combination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

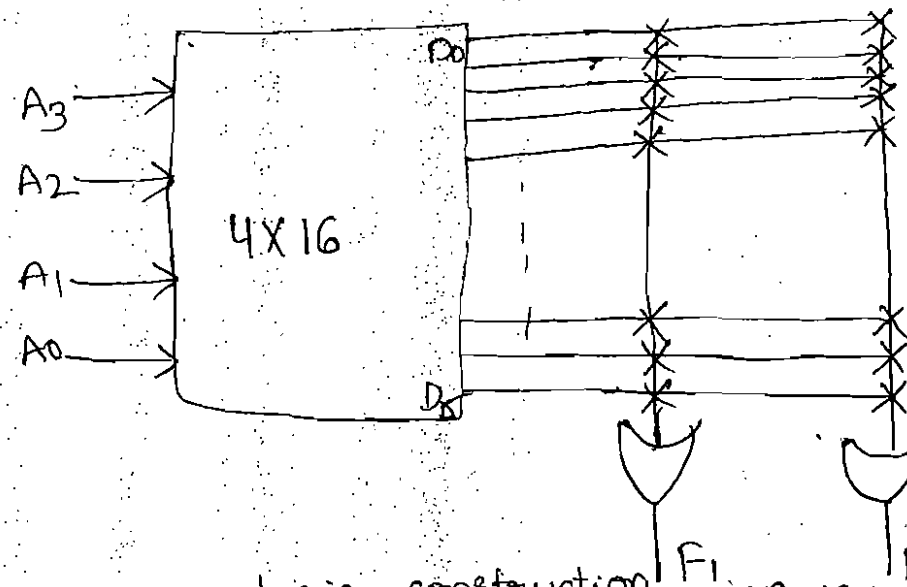


Block diagram.

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.

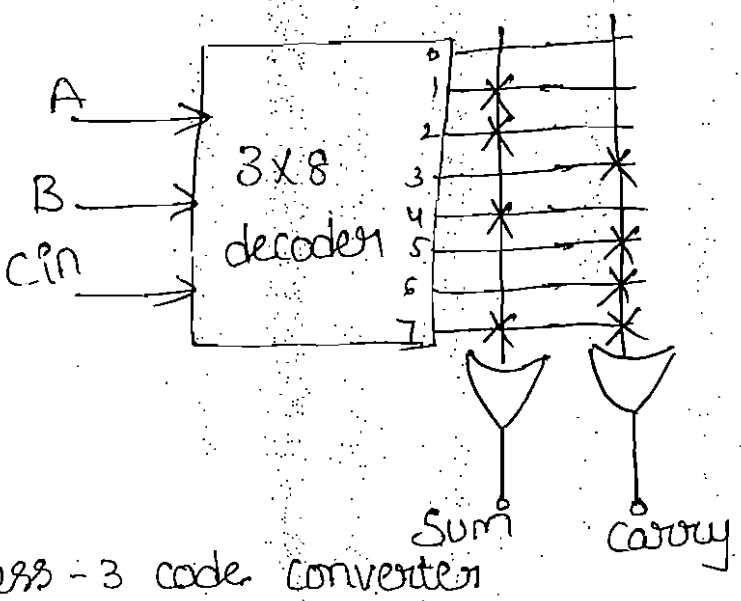
Truth table

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

Logic diagram



→ Design BCD to Excess-3 code converter using 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 :-

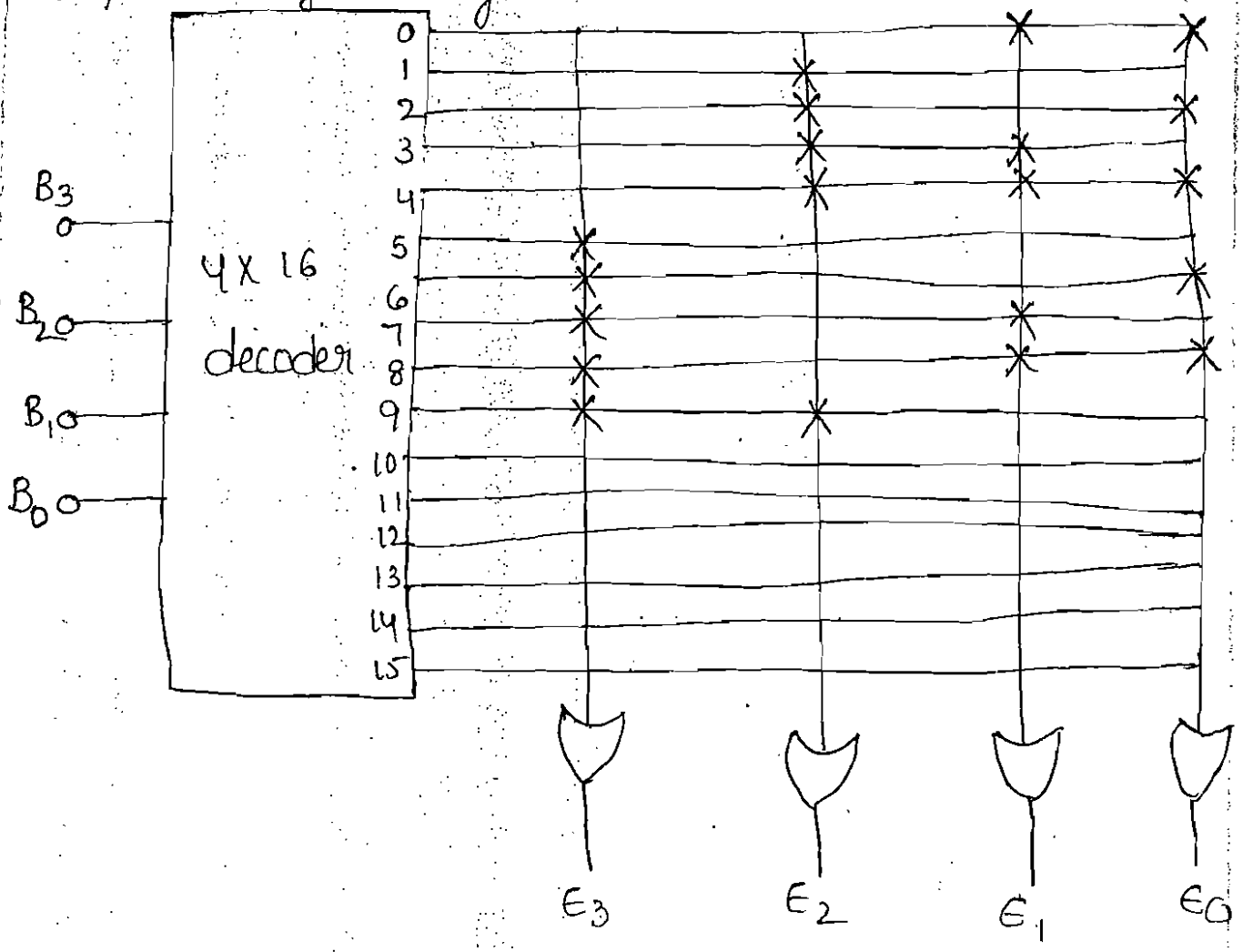
$E_0(B_3\ B_2\ B_1\ B_0) = \Sigma m(0,2,4,6,8)$

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

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

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

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.

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

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

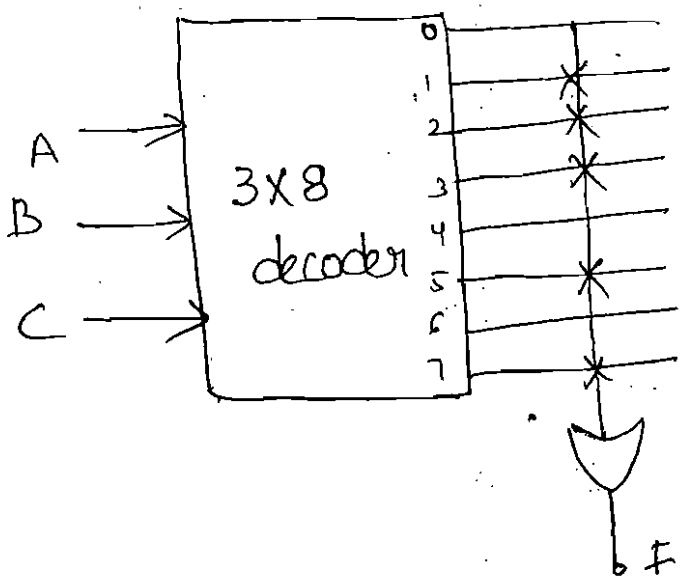
$$= \underset{1}{\bar{A}BC} + \bar{A}B\bar{C} + \underset{2}{ABC} + A\bar{B}C + \bar{A}BC + \bar{A}\bar{B}C + \underset{2}{AB\bar{C}} + \underset{1}{\bar{A}BC}$$

$$= \bar{A}BC + \bar{A}B\bar{C} + ABC + A\bar{B}C + \bar{A}BC + \bar{A}\bar{B}C$$

$011, 010, 111, 101, 011, 001$

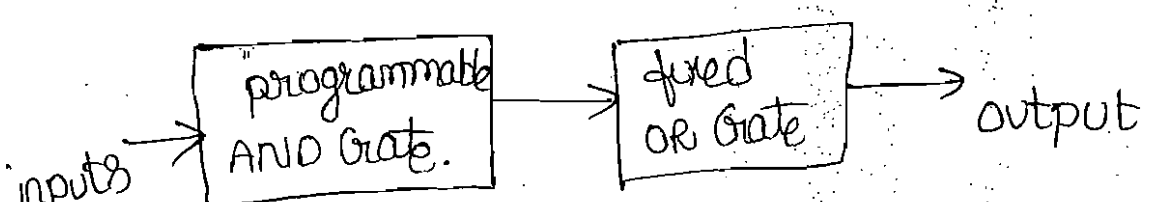
$f(A,B,C) = \Sigma m(1,2,3,5,7)$

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 programmble.

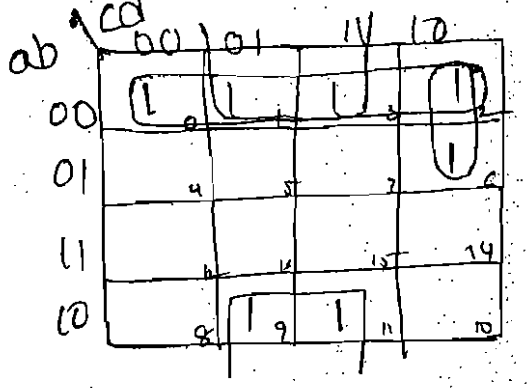


→ Implement the following functions using PAL.

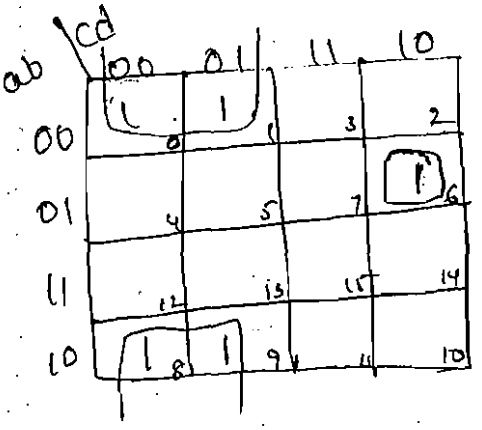
$F_1(a,b,c,d) = \Sigma m(0,1,2,3,6,9,11)$

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

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



$\bar{a}\bar{b} + \bar{a}c\bar{d} + \bar{b}d$

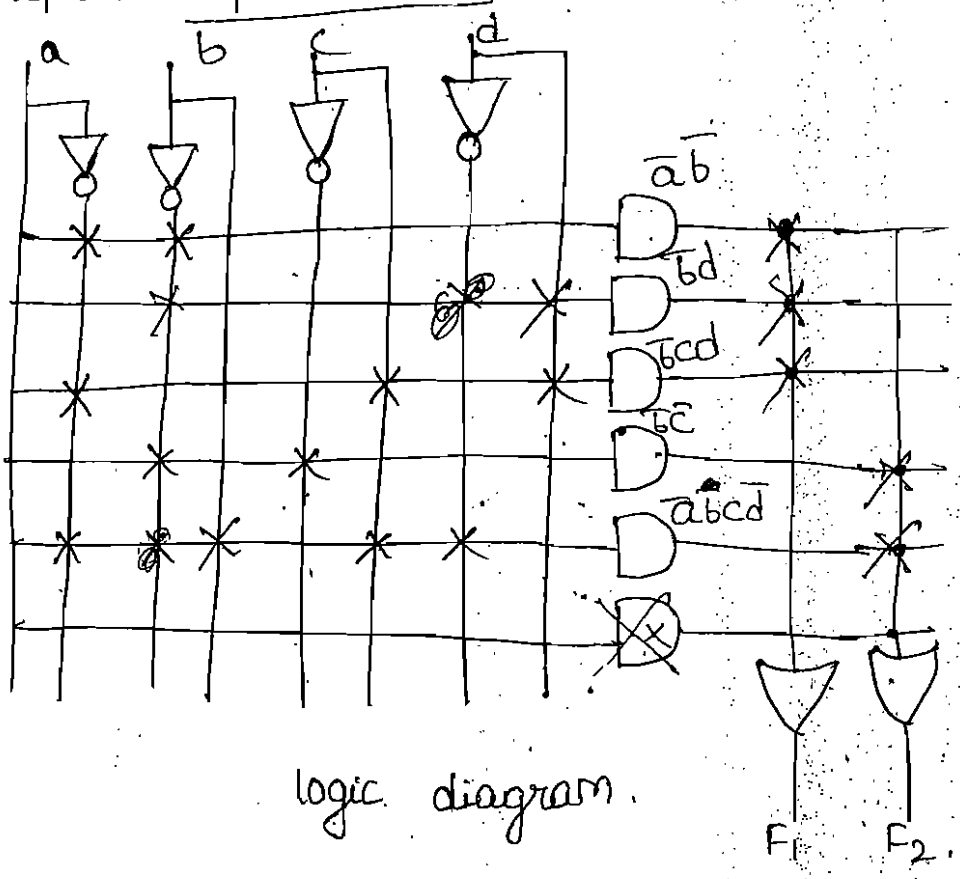


$\bar{b}\bar{c} + \bar{a}bc\bar{d}$

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

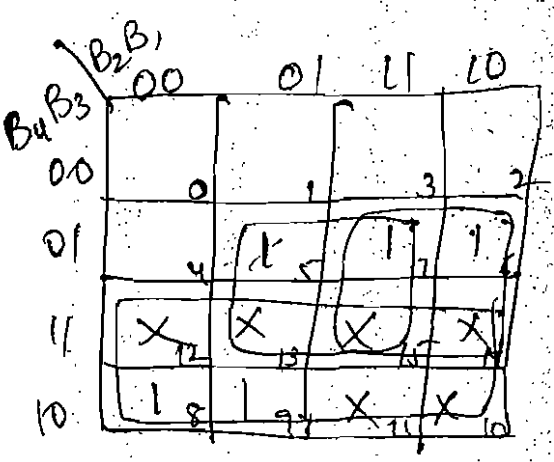
B_4	B_3	B_2	B_1	X_4	X_3	X_2	X_1
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

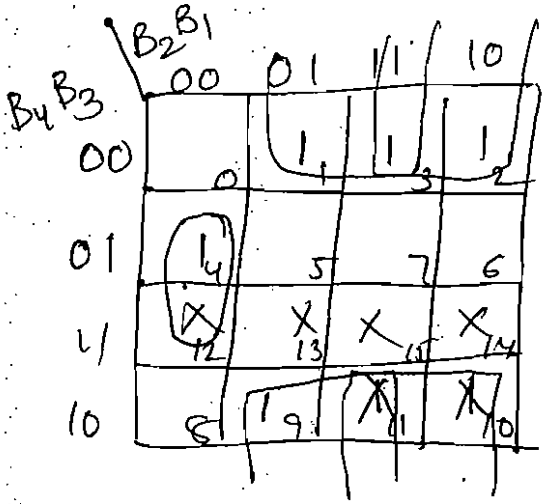
$X_4 = \Sigma m(5,6,7,8,9) + d(10,11,12,13,14,15)$

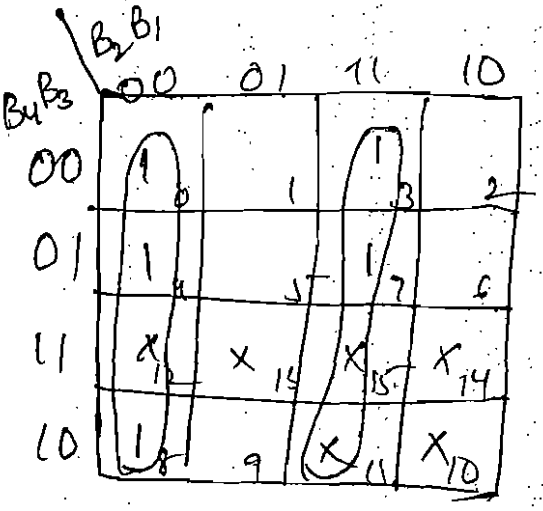
$X_3 = \Sigma m(1,2,3,4,9) + d(10,11,12,13,14,15)$

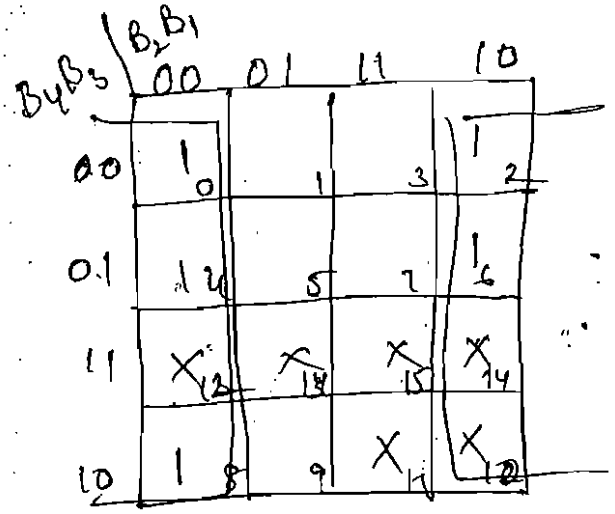
$X_2 = \Sigma m(0,3,4,7,8) + d(10,11,12,13,14,15)$

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









$X_4 = B_4 + B_3B_2 + B_3B_1$

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

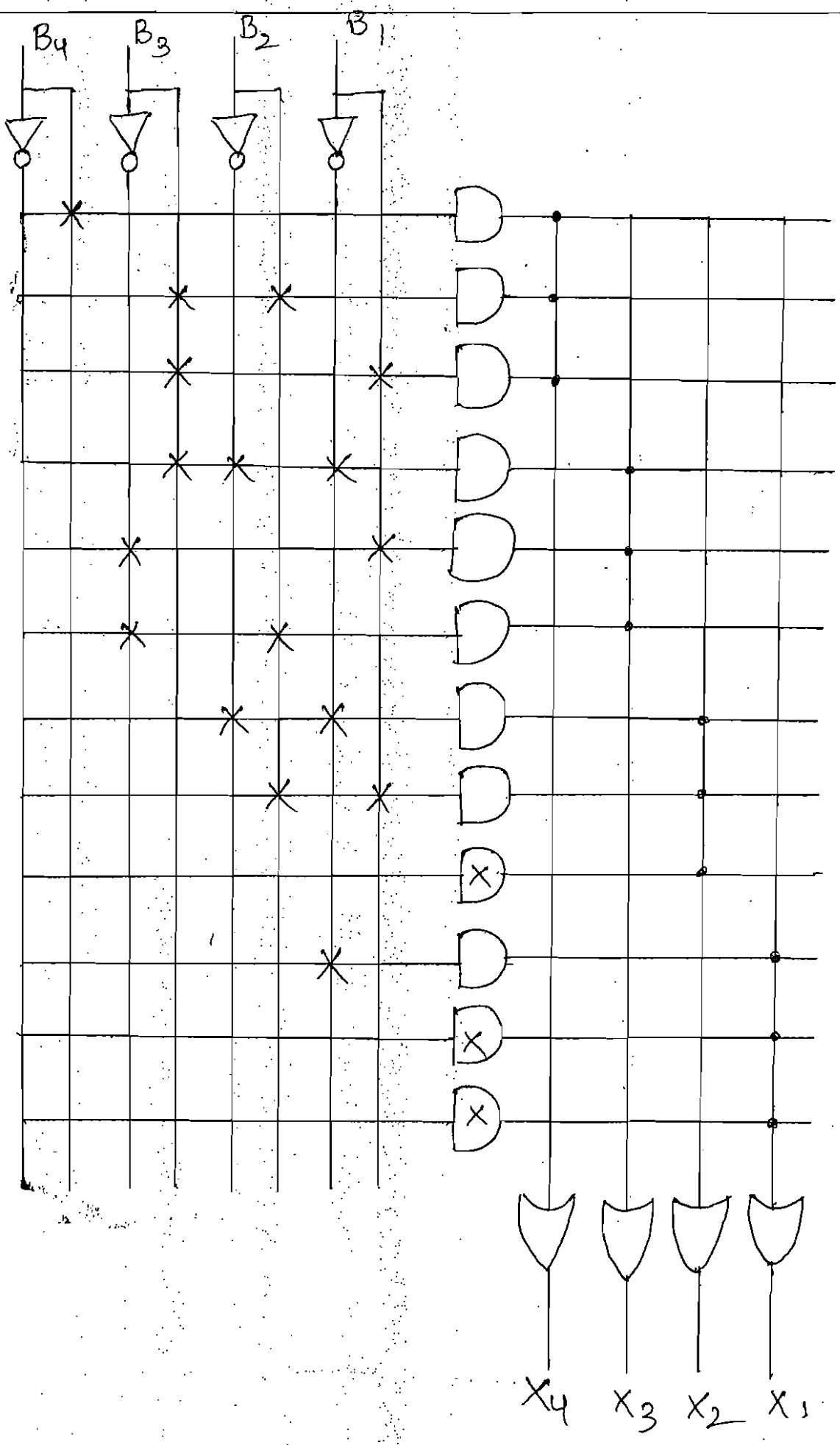
$X_2 = \overline{B_2\overline{B_1} + B_2B_1}$

$X_1 = \overline{B_1}$

Step 2 :- programming table.

product terms.	AND Gate inputs. B_4	B_3	B_2	B_1	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.

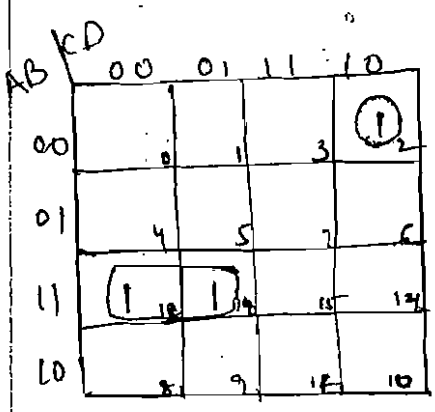
Implement the following boolean functions using PAL with four inputs and 3-wide AND-OR structure. Also write the PAL programming table.

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

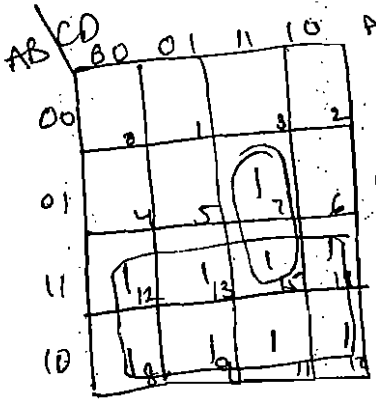
$F_2(A,B,C,D) = \Sigma m(7,8,9,10,11,12,13,14,15)$

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

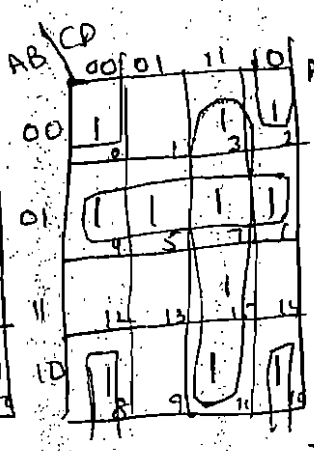
$F_4(A,B,C,D) = \Sigma m(1,2,8,12,13)$



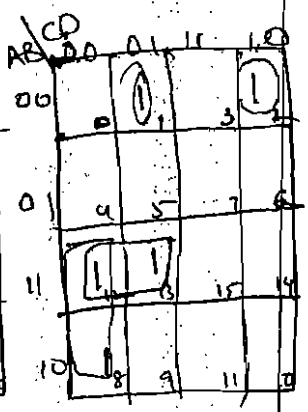
$F_1 = AB\bar{C} + \bar{A}\bar{B}C\bar{D}$



$F_2 = A + BCD$



$F_3 = \bar{A}B + C\bar{D} + \bar{B}\bar{D}$



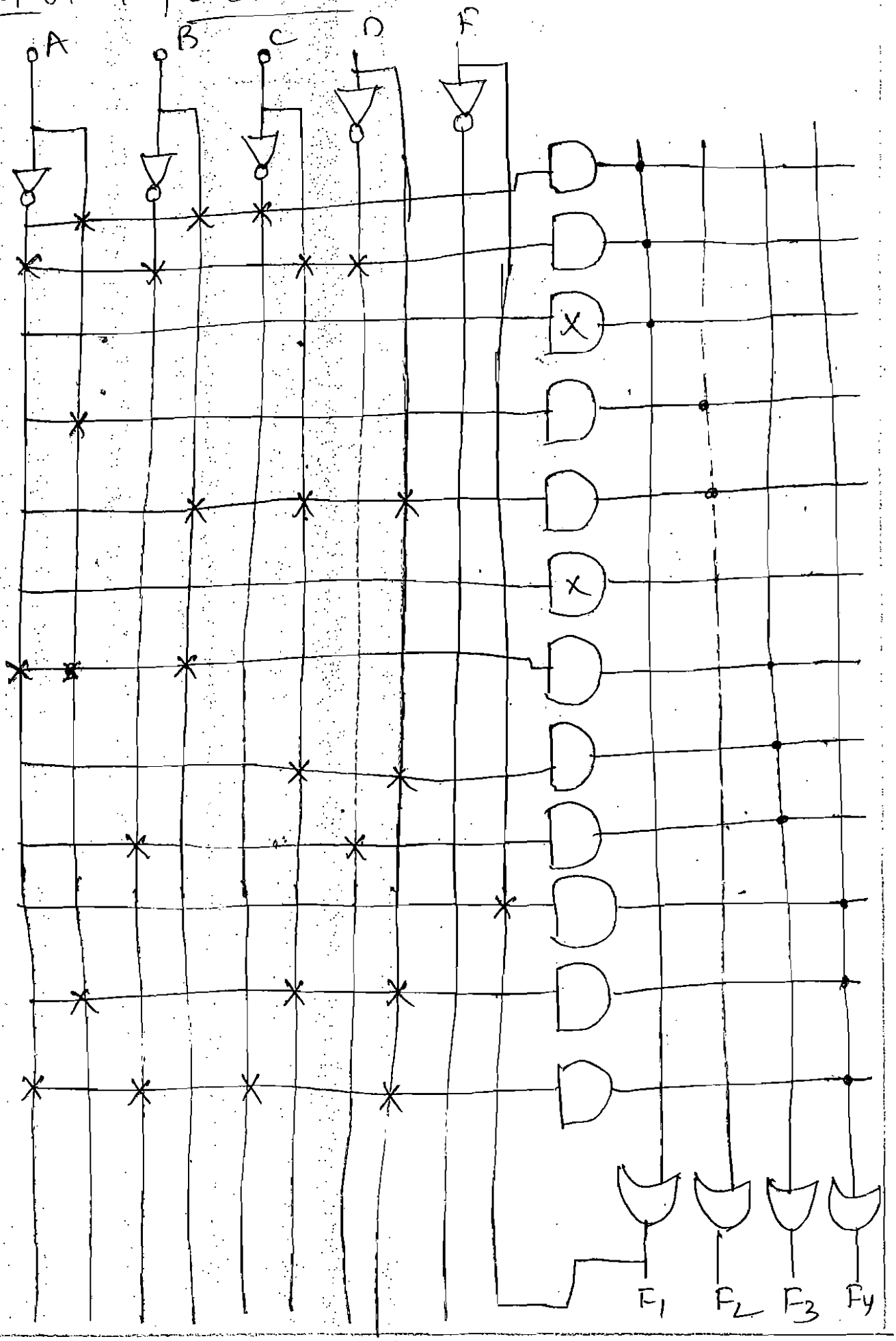
$F_4 = AB\bar{C} + A\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}\bar{B}C\bar{D}$

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

Step 2:- programming table

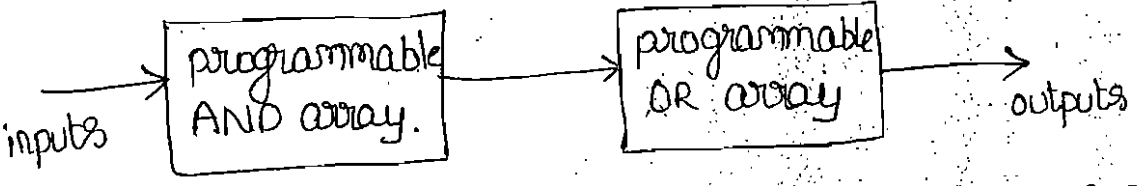
product term	AND inputs A B C D	F_1	outputs
1	1 1 0 –		$F_1 = AB\bar{C} + \bar{A}\bar{B}C\bar{D}$
2	0 0 1 0		
3	– – – –		– – –
4	1 – – –		$F_2 = A + BCD$
5	– 1 1 1		
6	– – – –		– – –
7	0 1 – –		$F_3 = \bar{A}B + C\bar{D} + \bar{B}\bar{D}$
8	– – 1 1		
9	– 0 – 0		– – –
10	– – – –	1	
11	1 – 0 0		$F_4 =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

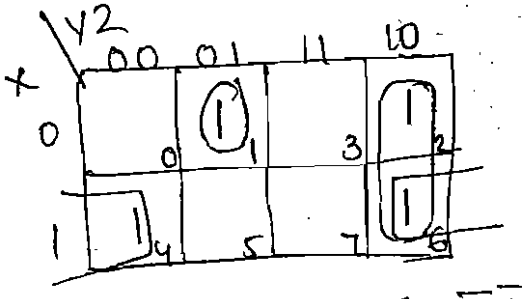


→ Implement the given boolean functions by using PLA.

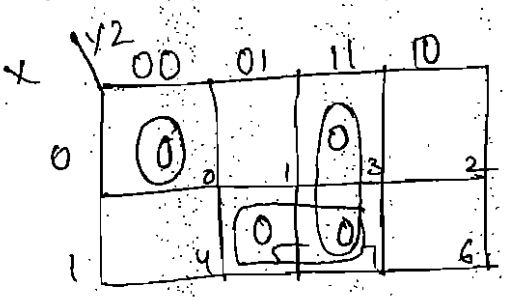
$A(X,Y,Z) = \Sigma m(1,2,4,6)$

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

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.



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



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

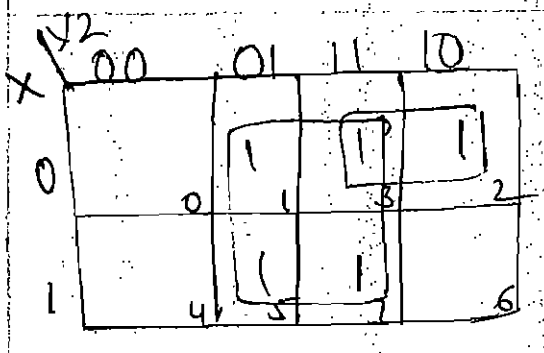
$A(C) = \overline{XZ + YZ + \bar{X}\bar{Y}\bar{Z}}$

K-map for A.

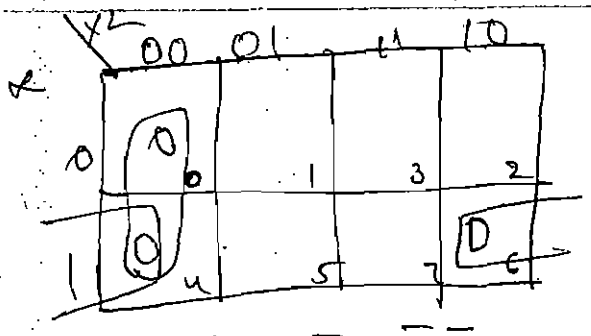
Simply $A(C) = \overline{(\bar{X}+\bar{Z})\cdot(\bar{Y}+\bar{Z})\cdot(X+Y+Z)}$

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

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



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



$\bar{B} = X\bar{Z} + \bar{Y}\bar{Z}$

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

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

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

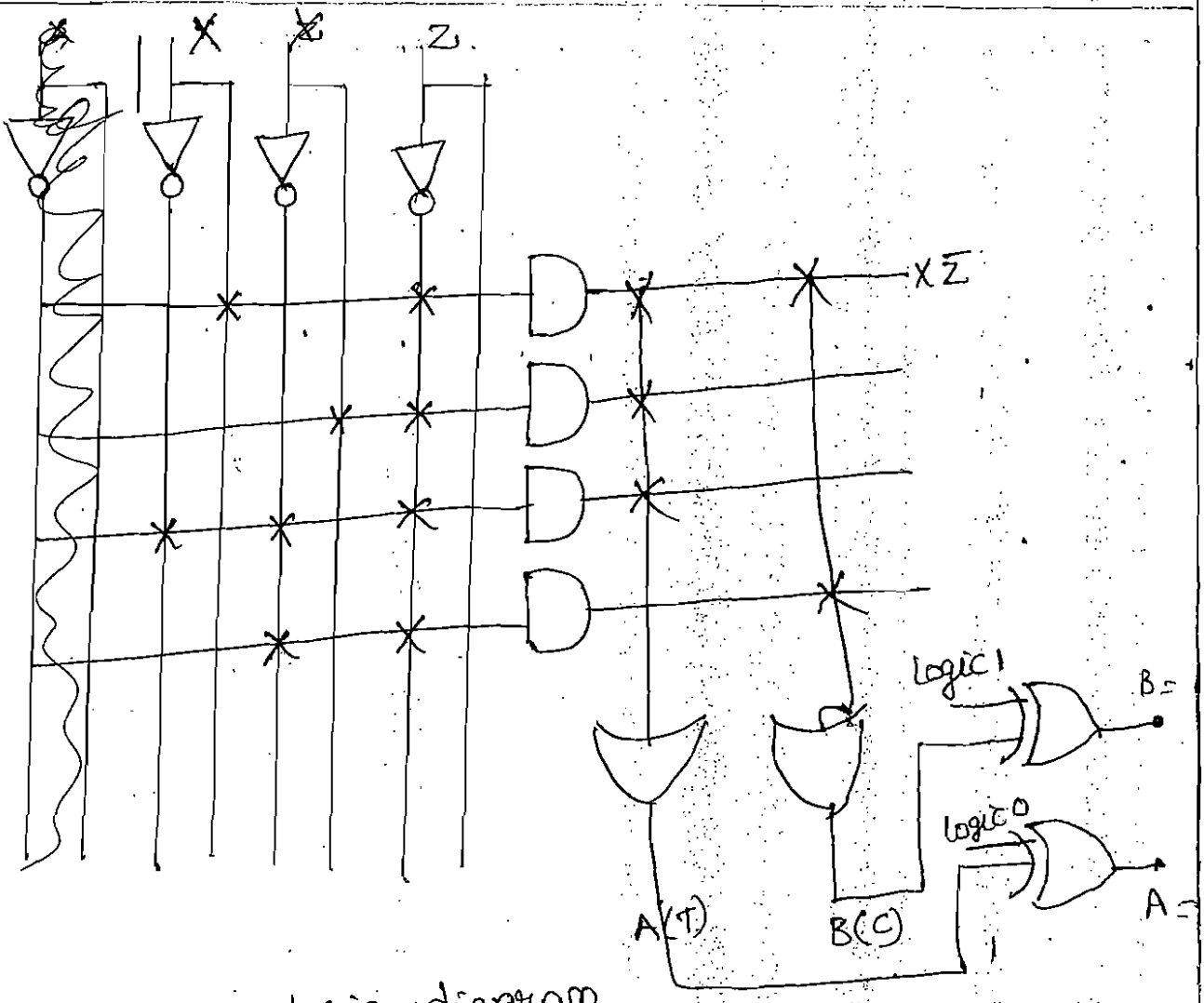
$= \bar{Y}\bar{Z} + X\bar{Z}$

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

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

Step 2 :- programming table.

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



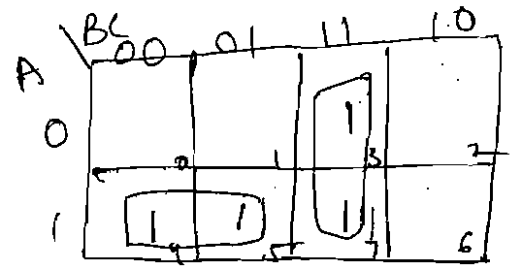
Logic diagram.

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

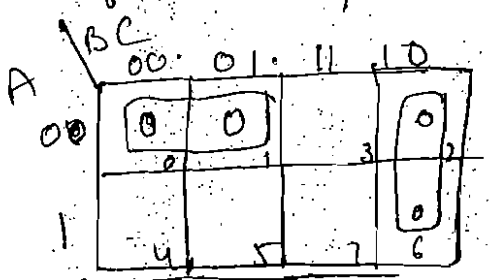
$F_1(A,B,C) = \Sigma m(3,4,5,7)$

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

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



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



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

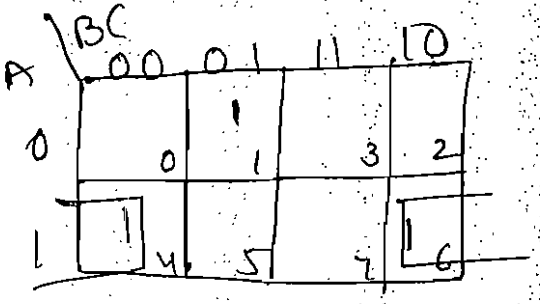
$= \overline{(A+B)} + \overline{(\bar{B}+C)}$

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

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

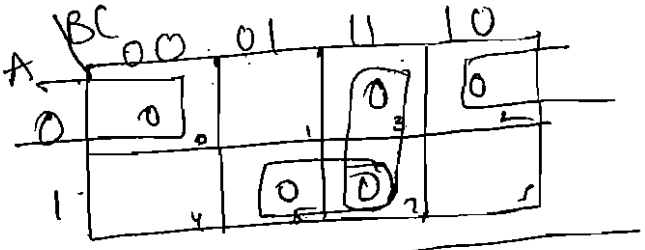
$= \bar{A}.\bar{B} + B.\bar{C}$

K-map for F_2 (true form)



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

K-map for F_2 (complement form)



$\bar{F}_2 = (\bar{A}+C).(\bar{B}+\bar{C})(\bar{A}+\bar{C})$

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

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

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

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

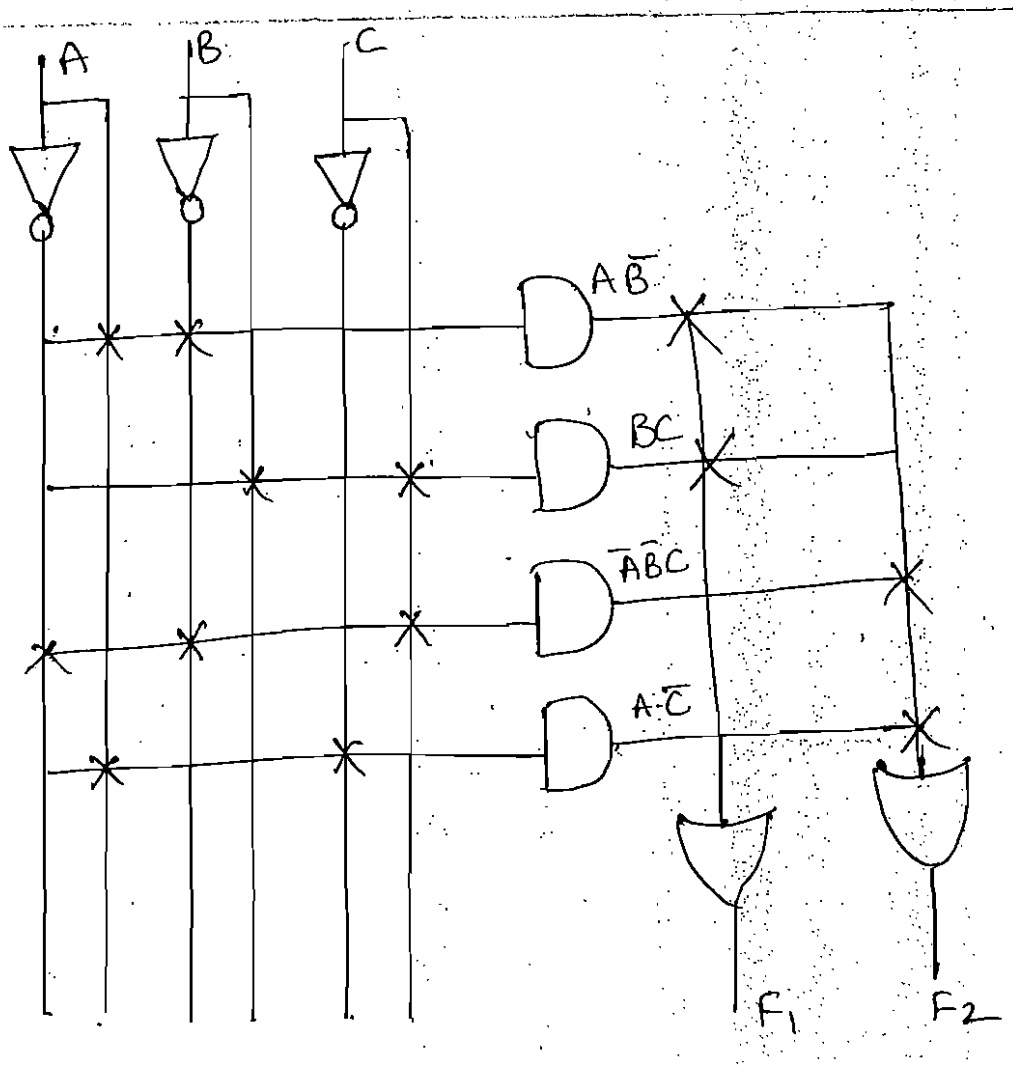
$F_1(C) = \bar{A}\bar{B} + B\bar{C}$

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

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 A	B	C	outputs $F_1(T)$	$F_2(C)$
$A\bar{B}$	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.

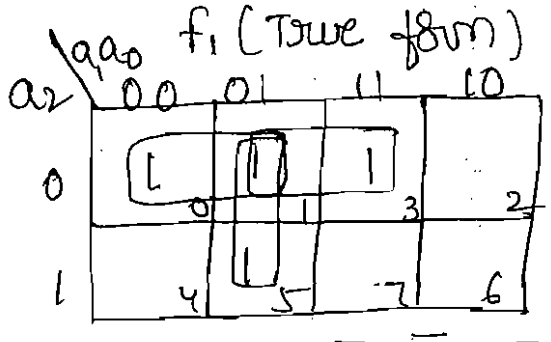
→ Implement the following multi boolean function using (3X4X2) PLA.

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

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

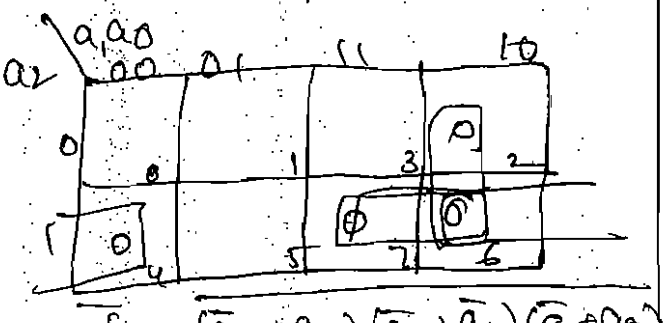
step 1:- K-maps.

f_1 (True form)



$f_1(T) = \bar{a}_1a_0 + \bar{a}_2\bar{a}_1 + \bar{a}_2a_0$

f_1 (complement form)

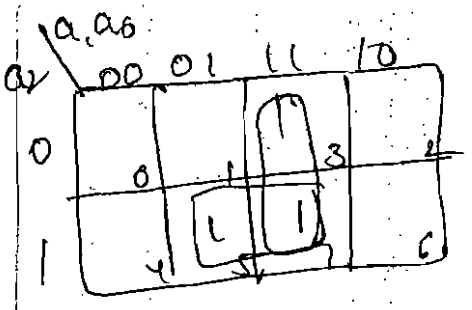


$\bar{f}_1 = (\bar{a}_2 + a_0)(\bar{a}_2 + \bar{a}_1)(\bar{a}_1 + a_0)$

$f_1(C) = \overline{\overline{a}_2} \cdot a_0 + \overline{\overline{a}_2} \cdot \overline{\overline{a}_1} + \overline{\overline{a}_1} \cdot \overline{a}_0$

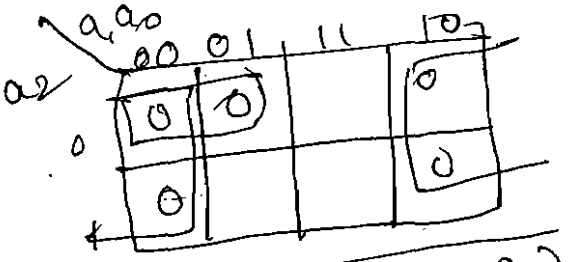
$= a_2 \cdot \overline{a}_0 + a_2 \cdot a_1 + a_1 \cdot \overline{a}_0$

f_2(True form)



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

F_2 (complement form..



$\overline{F}_2 = (\overline{a}_0) \cdot (a_2 + a_1)$

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

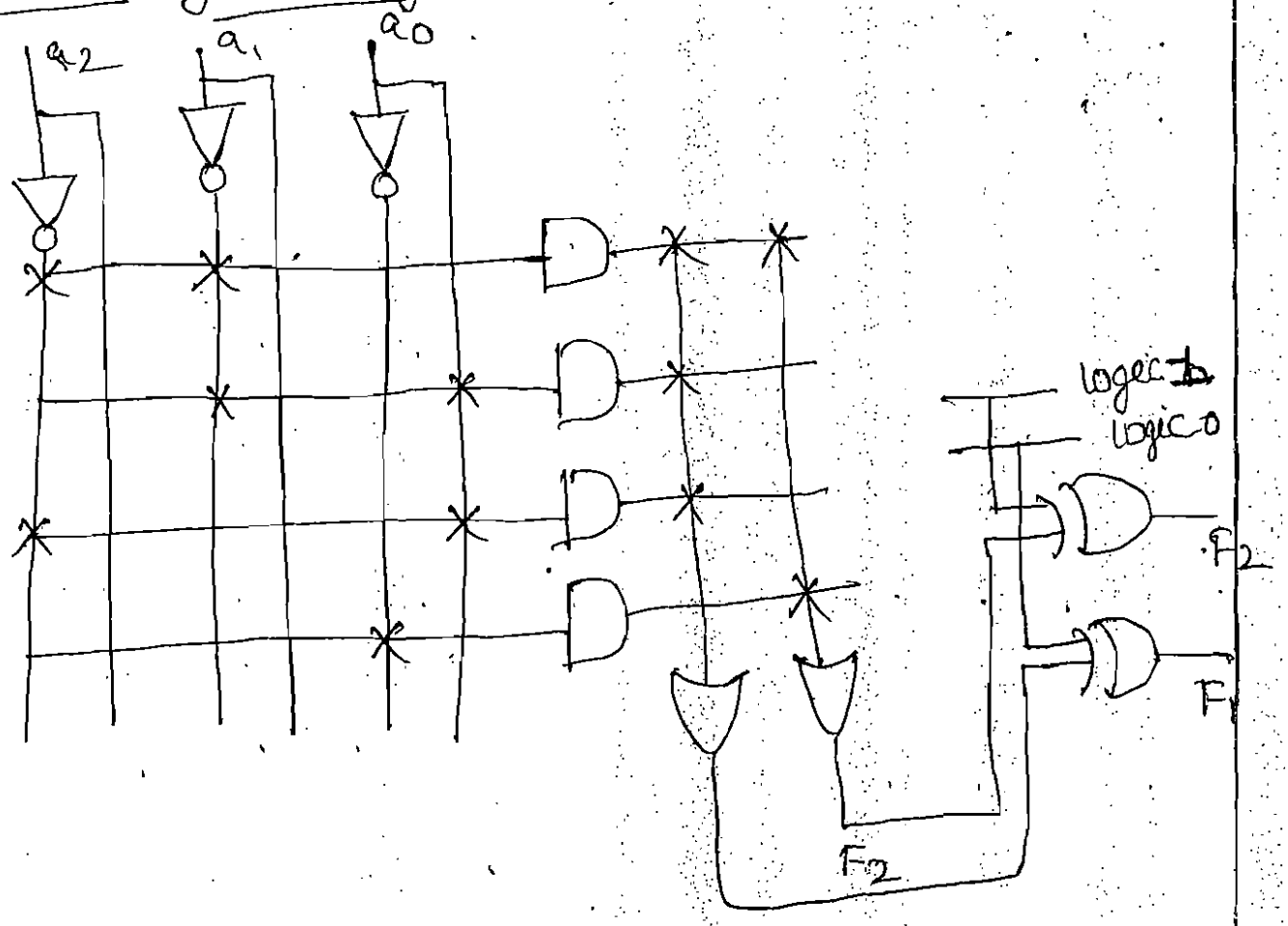
$F_1(T) = \overline{a}_1 a_0 + \overline{a}_2 \overline{a}_1 + \overline{a}_2 a_0$ $F_1(C) = a_2 \cdot \overline{a}_0 + a_2 \cdot a_1 + a_1 \cdot \overline{a}_0$

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

step 2:- PLA programming table.

product terms	Inputs a_2	a_1	a_0	outputs $F_1(T)$	$F_2(C)$
$\overline{a}_1 a_0$	—	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 :-



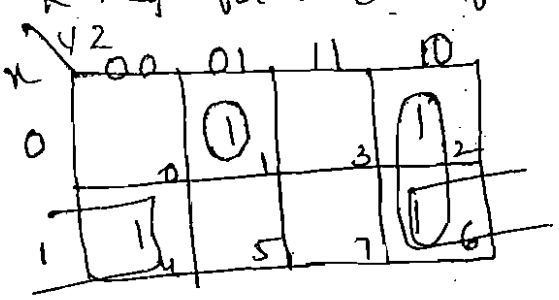
→ Implement the following using PLA

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

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

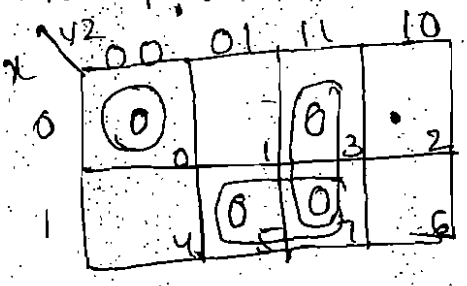
Step 1 :- K-map.

K-map for A (True form).



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

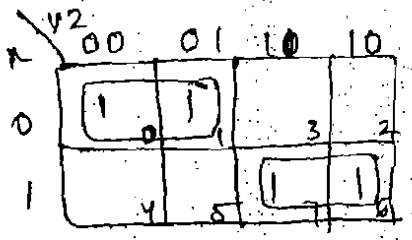
K-map for A (complement form).



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

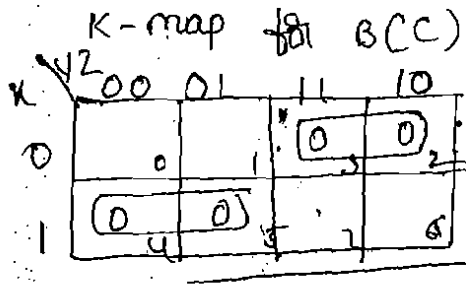
$= \bar{x}\bar{y}\bar{z} + yz + xz.$

K-map for B(T)



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

K-map for B(C)

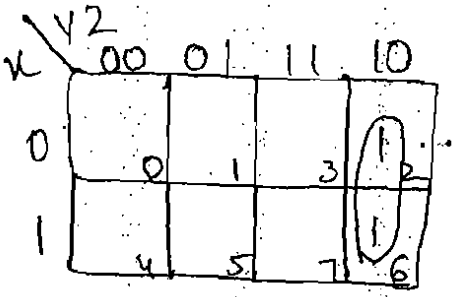


$B(C) = \overline{(\bar{x}+y)(x+\bar{y})}$

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

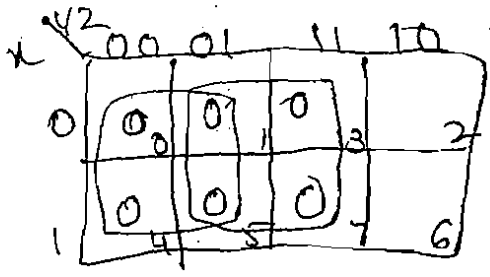
$= x\bar{y} + \bar{x}y$.

K-map for C(T)



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

K-map for C(C)



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

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

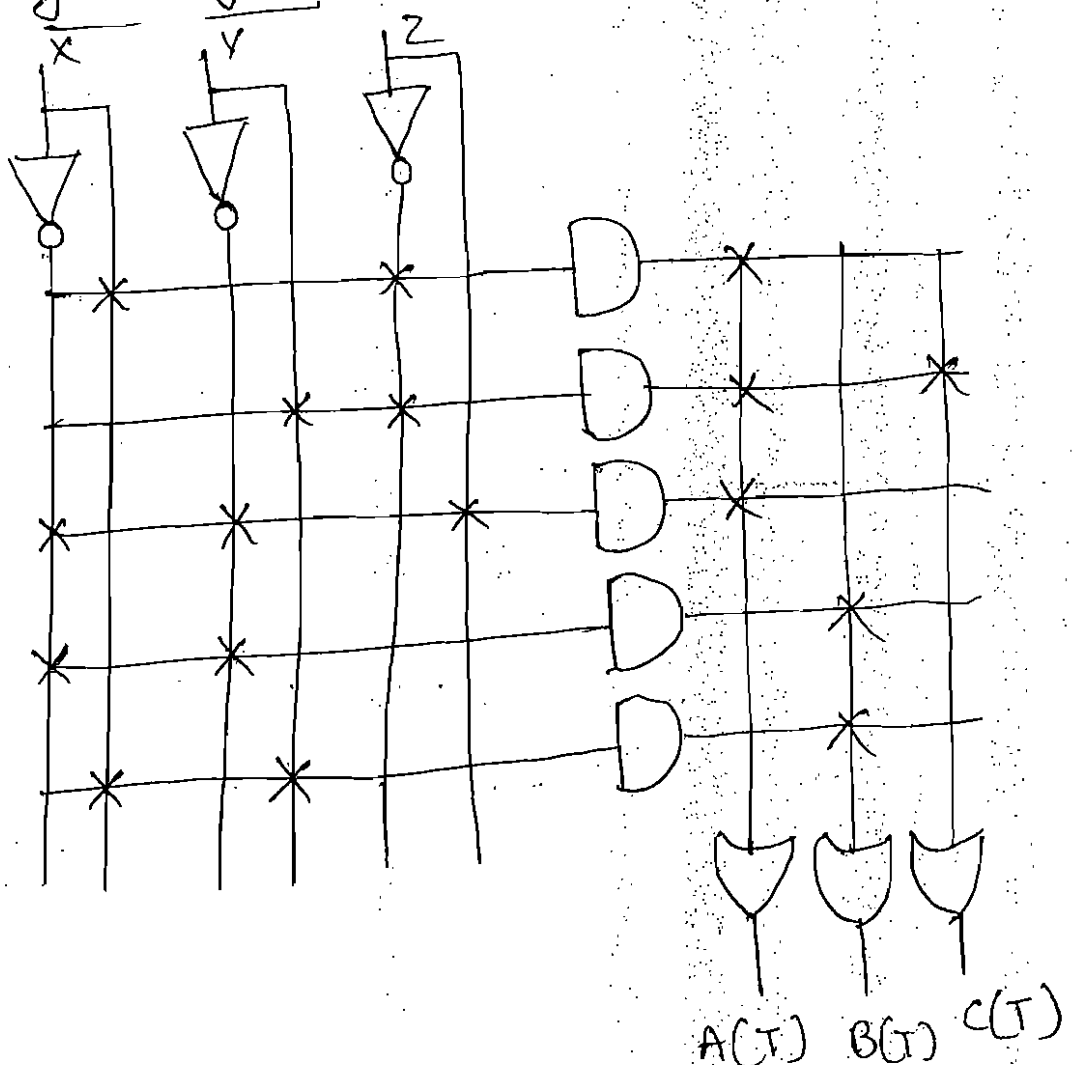
$= \bar{y} + z$.

Step 2 :- programming table.

product term	inputs x	y	z.	outputs 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	x	–	1	–
xy	1	1	–	–	1	–

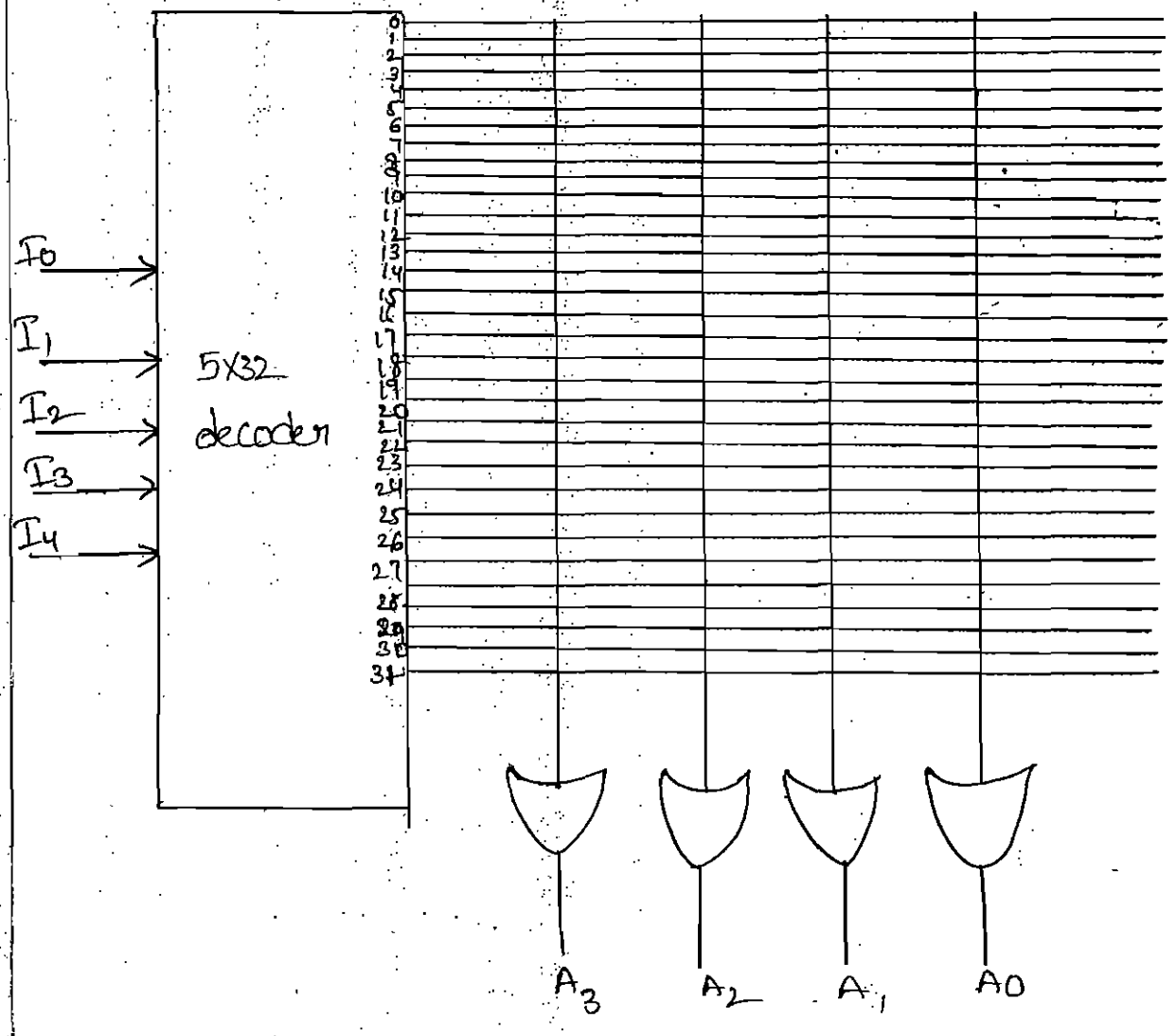
Step - 3

Logic diagram :-



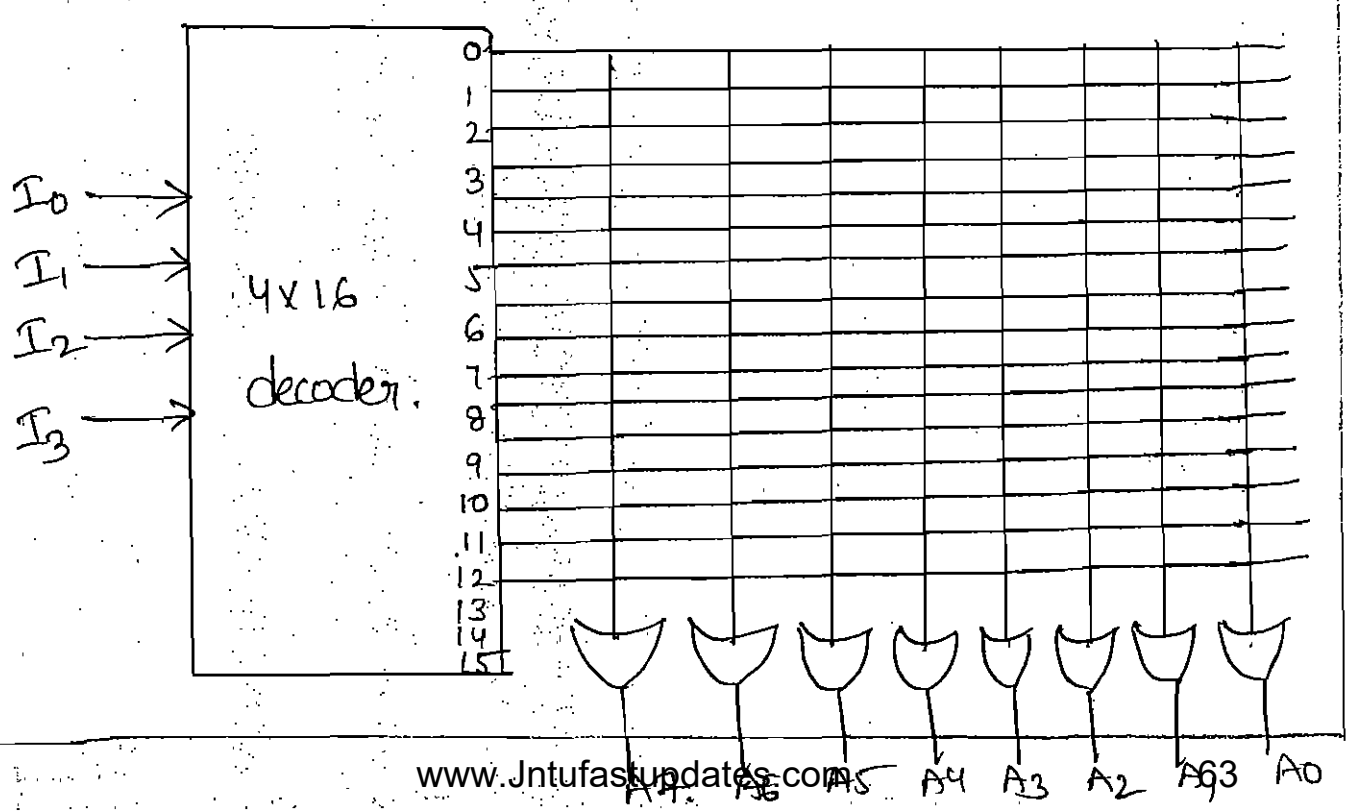
1) Give the logic implementation of a 32x4 bit ROM using a decoder of a suitable size.

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 programmable.	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 implemented using PROM.	4. Any Boolean function in SOP form can be implemented using PAL.

→ Implement a Binary to BCD code converter by using PAL

→ Iam taking 3-bit Binary P,

Binary B_3 B_2 B_1	BCD code 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

→ Ram 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 X_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

$C_5 = Em(10,11,12,13,14,15)$.

$C_4 = Em(8,9)$

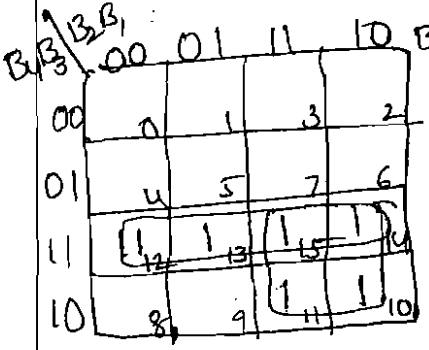
$C_3 = Em(4,5,6,7,14,15)$

$C_2 = Em(2,3,6,7,12,13)$

$C_1 = Em(1,3,5,7,9,11,13,15)$.

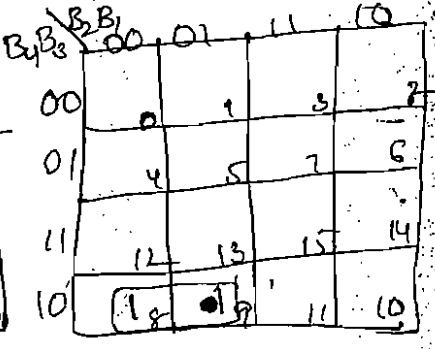
Step 1:- K-map.

K-map for C_5.



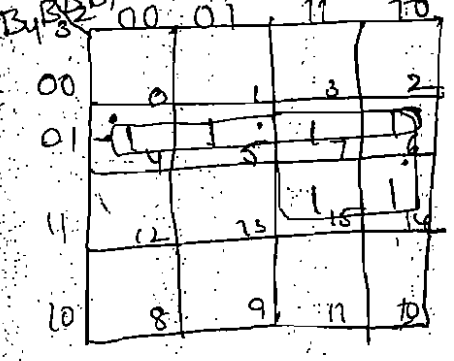
$B_4B_3 + B_4B_2$.

K-map for C_4



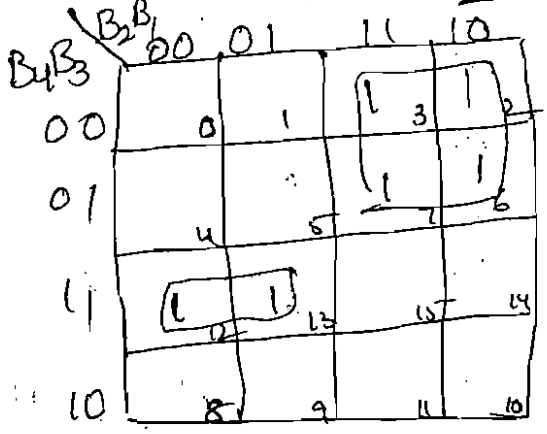
$B_4\overline{B_3}\,\overline{B_2}$

K-map for C_3



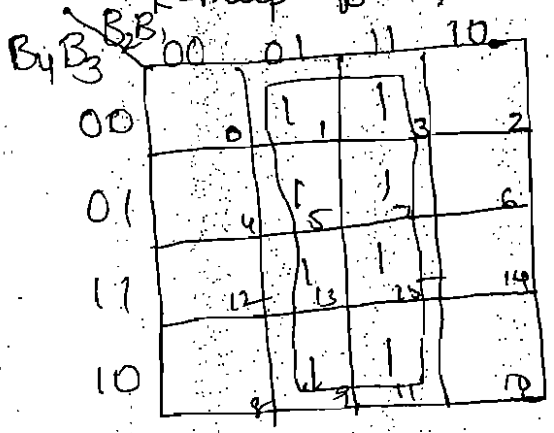
$\overline{B_4}B_3 + B_3B_2$

K-map for C_2



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

K-map for C_1

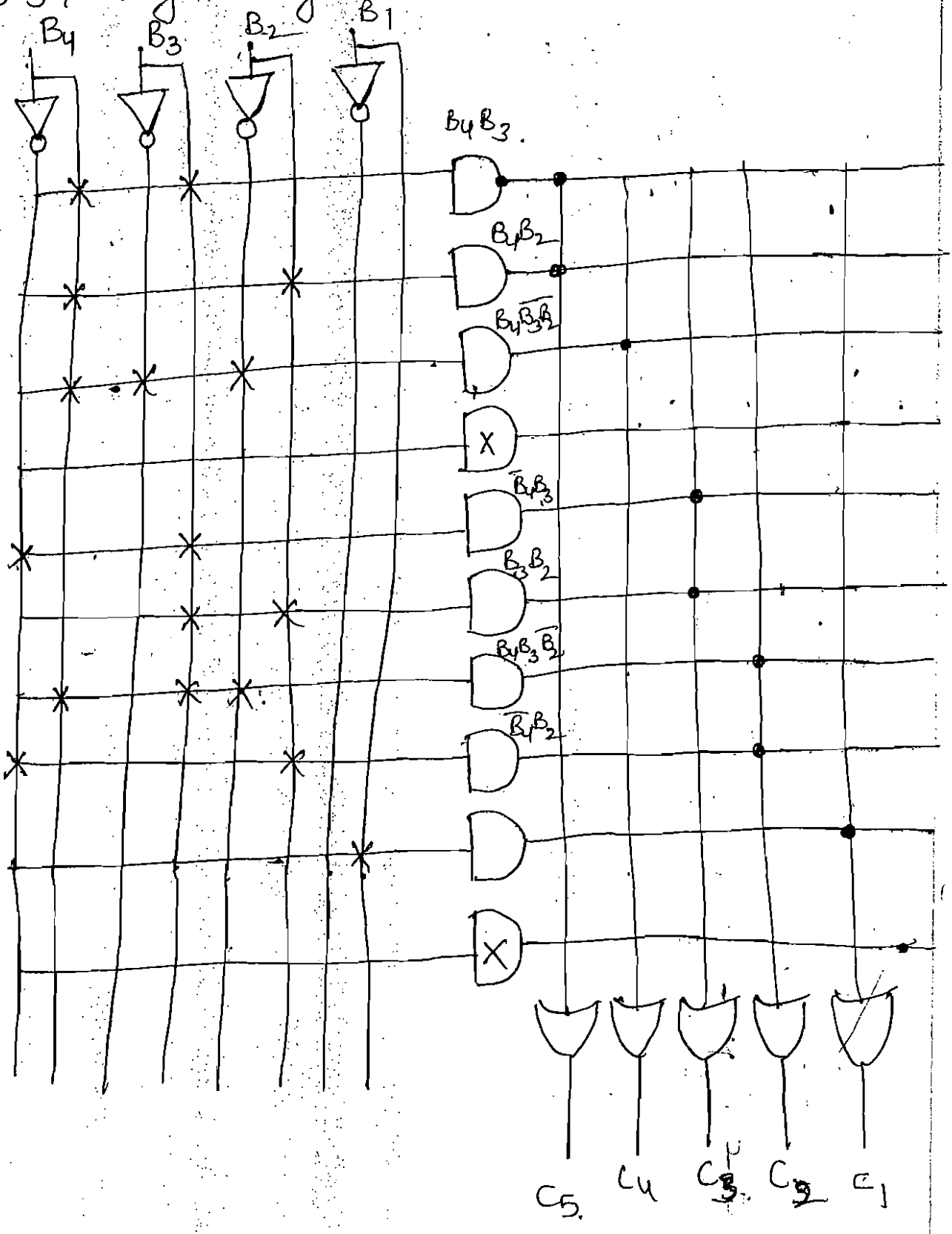


B_1

step 2:- programming table

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

Step 3 :- logic diagram.



MODULE-III:

Combinational Logic Circuits

Combinational Logic Design

Logic circuits for digital systems may be combinational or sequential. The output of a combinational circuit depends on its present inputs only .Combinational circuit processing operation fully specified logically by a set of Boolean functions .A combinational circuit consists of input variables, logic gates and output variables.Both input and output data are represented by signals, i.e., they exists in two possible values. One is logic –1 and the other logic 0.

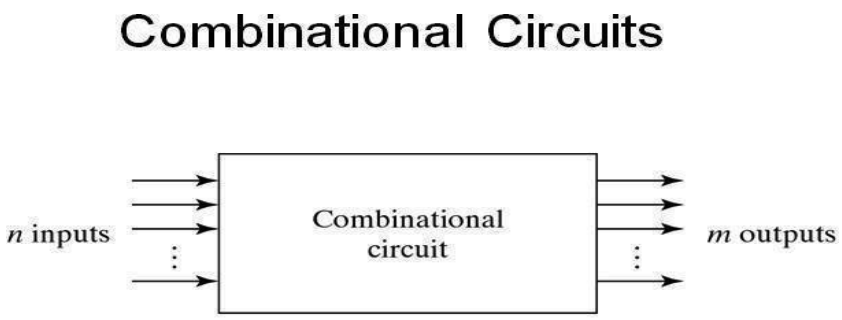


Fig. Block Diagram of Combinational Circuit

For n input variables,there are 2^n possible combinations of binary input variables .For each possible input Combination ,there is one and only one possible output combination.A combinational circuit can be described by m Boolean functions one for each output variables.Usually the input s comes from flip-flops and outputs goto flip-flops.

Design Procedure:

1.The problem is stated
2. The number of available input variables and required output variables is determined. 3.The input and output variables are assigned letter symbols.
4.The truth table that defines the required relationship between inputs and outputs is derived.
5.The simplified Boolean function for each output is obtained.
6.The logic diagram is drawn.

Adders:

Digital computers perform variety of information processing tasks,the one is arithmetic operations.And the most basic arithmetic operation is the addition of two binary digits.i.e, 4 basic possible operations are:

0+0=0,0+1=1,1+0=1,1+1=10

The first three operations produce a sum whose length is one digit, but when augends and addend bits are equal to 1,the binary sum consists of two digits.The higher significant bit of this result is called a carry.A combinational circuit that performs the addition of two bits is called a half-adder. One that performs the addition of 3 bits (two significant bits & previous carry) is called a full adder.& 2 half adder can employ as a full-adder.

The Half Adder: A Half Adder is a combinational circuit with two binary inputs (augends and addend bits and two binary outputs (sum and carry bits.) It adds the two inputs (A and B) and produces the sum (S) and the carry (C) bits. It is an arithmetic operation of addition of two single bit words.

Inputs		Outputs	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(a) Truth table

(b) Block diagram

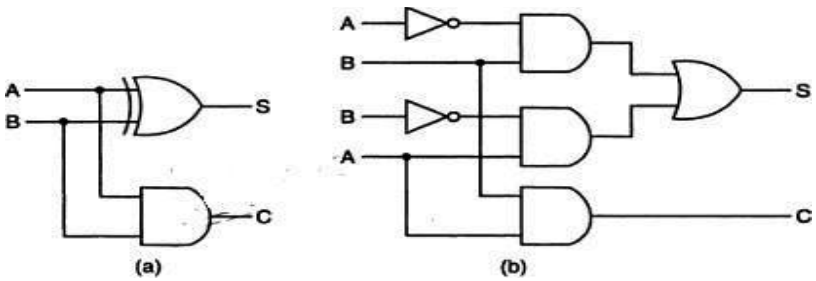
The Sum(S) bit and the carry (C) bit, according to the rules of binary addition, the sum (S) is the X-OR of A and B (It represents the LSB of the sum). Therefore,

S=A + B= $A \oplus B$

The carry (C) is the AND of A and B (it is 0 unless both the inputs are 1).Therefore,

C=AB

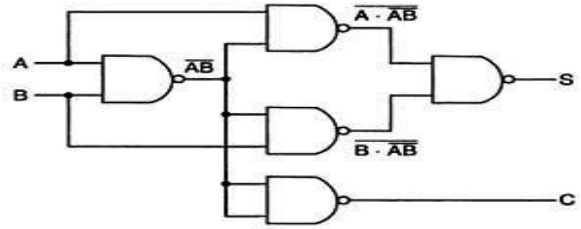
A half-adder can be realized by using one X-OR gate and one AND gate a



Logic diagrams of half-adder

NAND LOGIC:

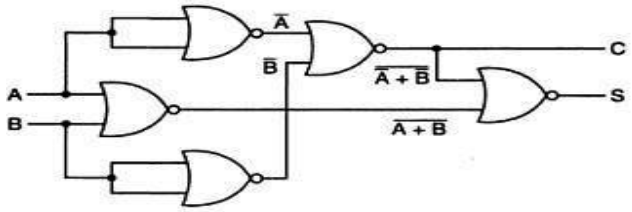
$$S = A\bar{B} + \bar{A}B = A\bar{B} + A\bar{A} + \bar{A}B + B\bar{B}$$
$$= A(\bar{A} + \bar{B}) + B(\bar{A} + \bar{B})$$
$$= A \cdot \overline{A\bar{B}} + B \cdot \overline{A\bar{B}}$$
$$= \overline{A \cdot \overline{A\bar{B}} \cdot B \cdot \overline{A\bar{B}}}$$
$$C = AB = \overline{\overline{AB}}$$



Logic diagram of a half-adder using only 2-input NAND gates.

NOR Logic:

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



Logic diagram of a half-adder using only 2-input NOR gates.

The Full Adder:

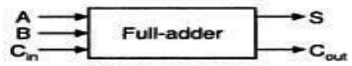
A Full-adder is a combinational circuit that adds two bits and a carry and outputs a sum bit and a carry bit. To add two binary numbers, each having two or more bits, the LSBs can be added by using a half-adder. The carry resulted from the addition of the LSBs is carried over to the next significant column and added to the two bits in that column. So, in the second and higher columns, the two data bits of that column and the carry bit generated from the addition in the previous column need to be added.

The full-adder adds the bits A and B and the carry from the previous column called the carry-in C_{in} and outputs the sum bit S and the carry bit called the carry-out C_{out} . The variable S gives the value of the least significant bit of the sum. The variable C_{out} gives the output carry.The

eight rows under the input variables designate all possible combinations of 1s and 0s that these variables may have. The 1s and 0s for the output variables are determined from the arithmetic sum of the input bits. When all the bits are 0s , the output is 0. The S output is equal to 1 when only 1 input is equal to 1 or when all the inputs are equal to 1. The C_{out} has a carry of 1 if two or three inputs are equal to 1.

Inputs			Sum	Carry
A	B	C_{in}	S	C_{out}
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

(a) Truth table



(b) Block diagram

Full-adder.

From the truth table, a circuit that will produce the correct sum and carry bits in response to every possible combination of A,B and C_{in} is described by

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

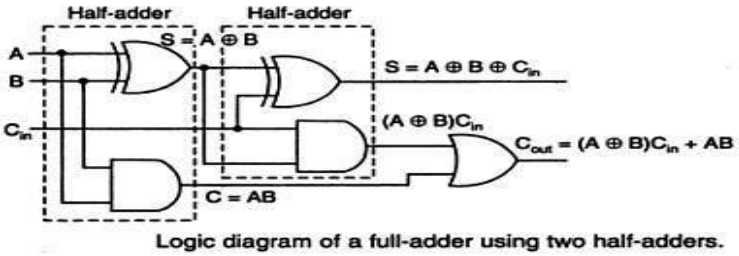
$$C_{out} = ABC_{in} + A\bar{B}C_{in} + \bar{A}BC_{in} + AB\bar{C}_{in}$$

and

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

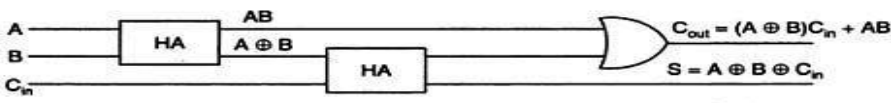
$$C_{out} = AC_{in} + BC_{in} + AB$$

The sum term of the full-adder is the X-OR of A,B, and C_{in}, i.e, the sum bit the modulo sum of the data bits in that column and the carry from the previous column. The logic diagram of the full-adder using two X-OR gates and two AND gates (i.e, Two half adders) and one OR gate is



Logic diagram of a full-adder using two half-adders.

The block diagram of a full-adder using two half-adders is



Block diagram of a full-adder using two half-adders.

Even though a full-adder can be constructed using two half-adders, the disadvantage is that the bits must propagate through several gates in accession, which makes the total propagation delay greater than that of the full-adder circuit using AOI logic.

The Full-adder neither can also be realized using universal logic, i.e., either only NAND gates or only NOR gates as

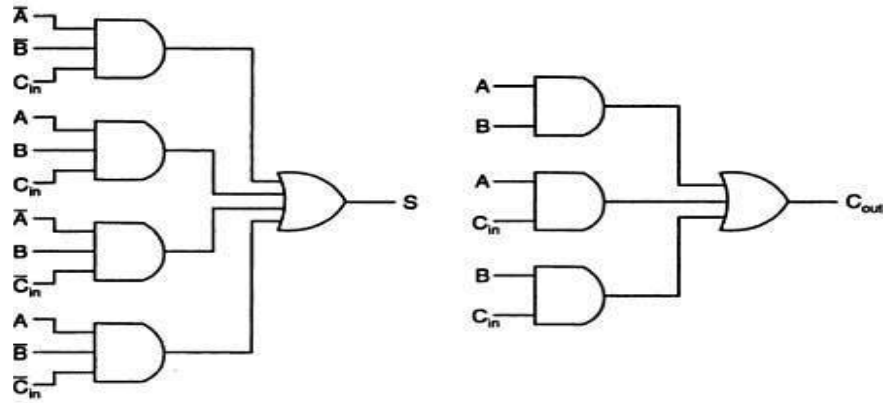
$$A \oplus B = \overline{\overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}}}$$

Then

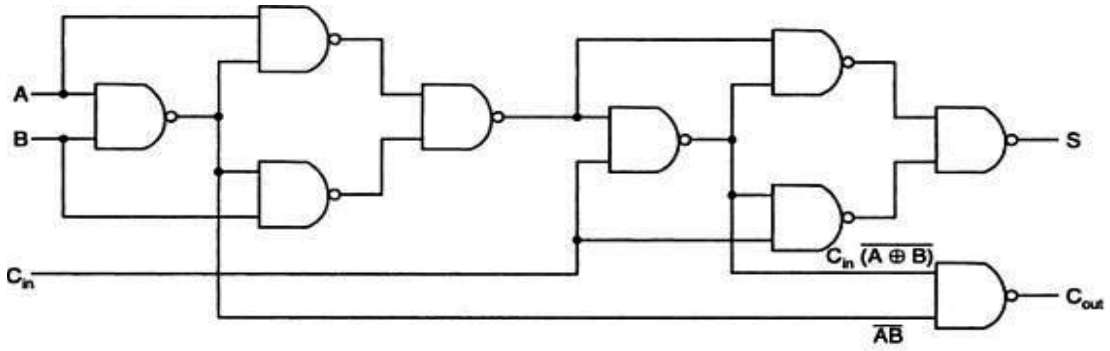
$$S = A \oplus B \oplus C_{in} = \overline{\overline{(A \oplus B) \cdot \overline{(A \oplus B)C_{in}}} \cdot \overline{C_{in} \cdot \overline{(A \oplus B)C_{in}}}}$$

NAND Logic:

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



Sum and carry bits of a full-adder using AOI logic.



Logic diagram of a full-adder using only 2-input NAND gates.

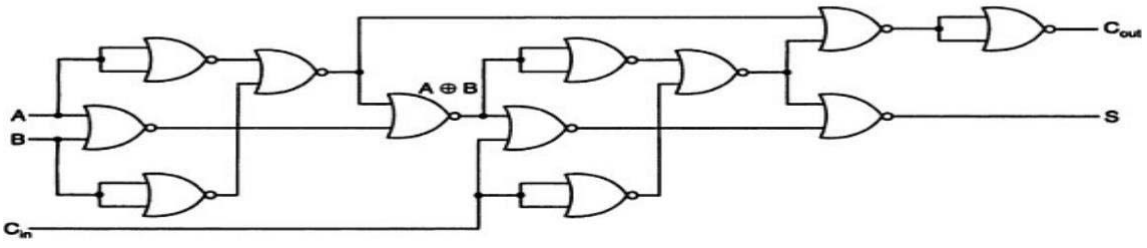
NOR Logic:

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

Then

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

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



Logic diagram of a full-adder using only 2-input NOR gates.

Subtractors:

The subtraction of two binary numbers may be accomplished by taking the complement of the subtrahend and adding it to the minuend. By this, the subtraction operation becomes an addition operation and instead of having a separate circuit for subtraction, the adder itself can be used to perform subtraction. This results in reduction of hardware. In subtraction, each subtrahend bit of the number is subtracted from its corresponding significant minuend bit to form a difference bit. If the minuend bit is smaller than the subtrahend bit, a 1 is borrowed from the next significant position., that has been borrowed must be conveyed to the next higher pair of bits by means of a signal coming out (output) of a given stage and going into (input) the next higher stage.

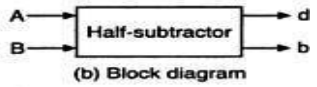
The Half-Subtractor:

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. . It is used to subtract the LSB of the subtrahend from the LSB of the minuend when one binary number is subtracted from the other.

A Half-subtractor is a combinational circuit with two inputs A and B and two outputs d and b. d indicates the difference and b is the output signal generated that informs the next stage that a 1 has been borrowed. When a bit B is subtracted from another bit A, a difference bit (d) and a borrow bit (b) result according to the rules given as

Inputs		Outputs	
A	B	d	b
0	0	0	0
1	0	1	0
1	1	0	0
0	1	1	1

(a) Truth table



(b) Block diagram

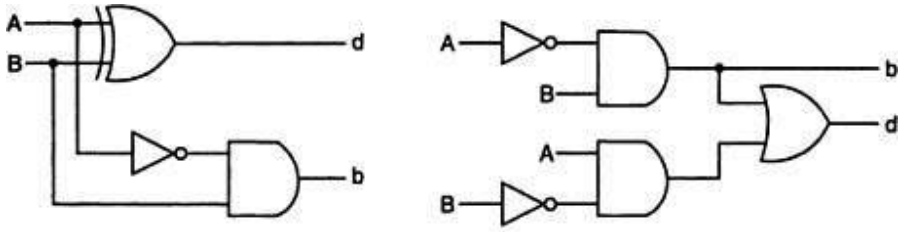
Half-subtractor.

The output borrow b is a 0 as long as A≥B. It is a 1 for A=0 and B=1. The d output is the result of the arithmetic operation 2b+A-B.

A circuit that produces the correct difference and borrow bits in response to every possible combination of the two 1-bit numbers is , therefore ,

d=A + B= $A \oplus B$ and b= B

That is, the difference bit is obtained by X-OR ing the two inputs, and the borrow bit is obtained by ANDing the complement of the minuend with the subtrahend.Note that logic for this exactly the same as the logic for output S in the half-adder.



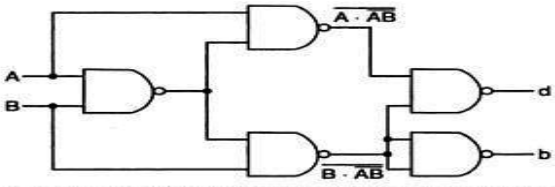
Logic diagrams of a half-subtractor.

A half-substractor can also be realized using universal logic either using only NAND gates or using NOR gates as:

NAND Logic:

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

$$b = \overline{A}B = B(\overline{A} + \overline{B}) = B(\overline{AB}) = \overline{B \cdot \overline{AB}}$$



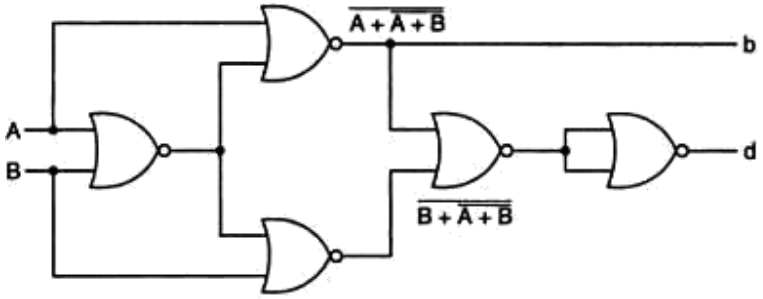
Logic diagram of a half-subtractor using only 2-input NAND gates.

NOR Logic:

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

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

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



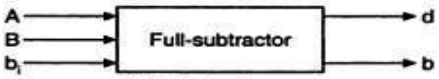
Logic diagram of a half-subtractor using only 2-input NOR gates.

The Full-Subtractor:

The half-subtractor can be only for LSB subtraction. IF there is a borrow during the subtraction 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. Such a subtraction is performed by a full-subtractor. It subtracts one bit (B) from another bit (A) , when already there is a borrow b_i from this column for the subtraction in the preceding column, and outputs the difference bit (d) and the borrow bit(b) required from the next d and b. The two outputs present the difference and output borrow. The 1s and 0s for the output variables are determined from the subtraction of A-B-b_i.

Inputs			Difference	Borrow
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

(a) Truth table



(b) Block diagram

Full-subtractor.

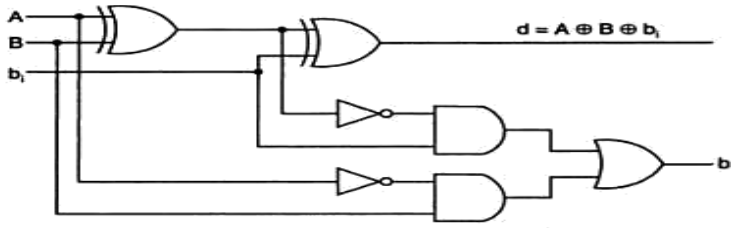
From the truth table, a circuit that will produce the correct difference and borrow bits in response to every possiblecombinations of A,B and b_i is

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

and

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

A full-subtractor can be realized using X-OR gates and AOI gates as



Logic diagram of a full-subtractor.

The full subtractor can also be realized using universal logic either using only NAND gates or using NOR gates as:

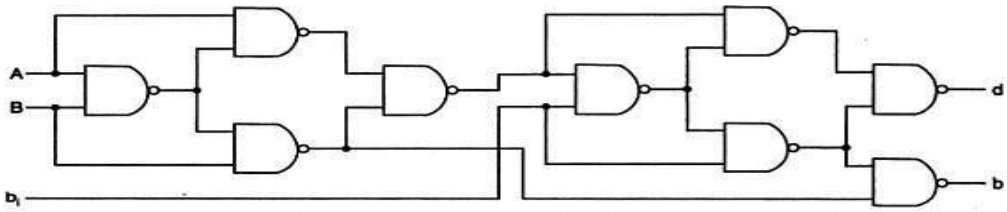
NAND Logic:

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

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

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

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



Logic diagram of a full-subtractor using only 2-input NAND gates.

NOR Logic:

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

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

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

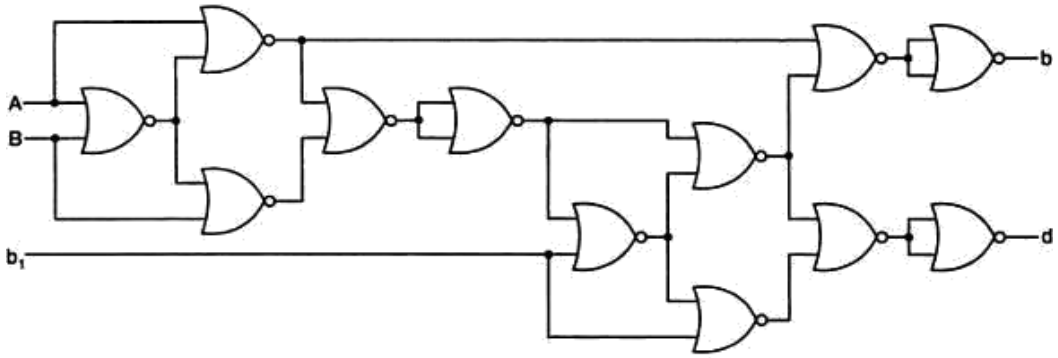
$$= \overline{(A \oplus B) + (\overline{A \oplus B}) + b_i} + \overline{b_i + (\overline{A \oplus B}) + b_i}$$

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

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

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

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

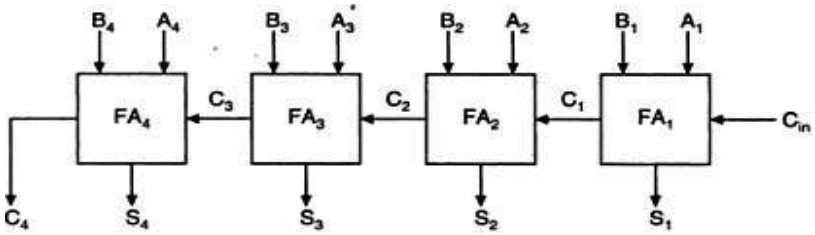


Logic diagram of a full subtractor using only 2-input NOR gates.

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.

The interconnection of four full-adder (FA) circuits to provide a 4-bit parallel adder. The augends 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 carries are connected in a chain through the full-adders. The input carry to the adder is C_{in} and the output carry is C_4. The S output generates the required sum bits. When the 4-bit full-adder circuit is enclosed within an IC package, it has four terminals for the augends bits, four terminals for the addend bits, four terminals for the sum bits, and two terminals for the input and output carries. AN n-bit parallel adder requires n-full adders. It can be constructed from 4-bit, 2-bit and 1-bit full adder ICs by cascading several packages. The output carry from one package must be connected to the input carry of the one with the next higher –order bits. The 4-bit full adder is a typical example of an MSI function.



Logic diagram of a 4-bit binary parallel adder.

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 sometime after the carry-in of that stage occurs. This is due to the propagation delays in the logic circuitry,

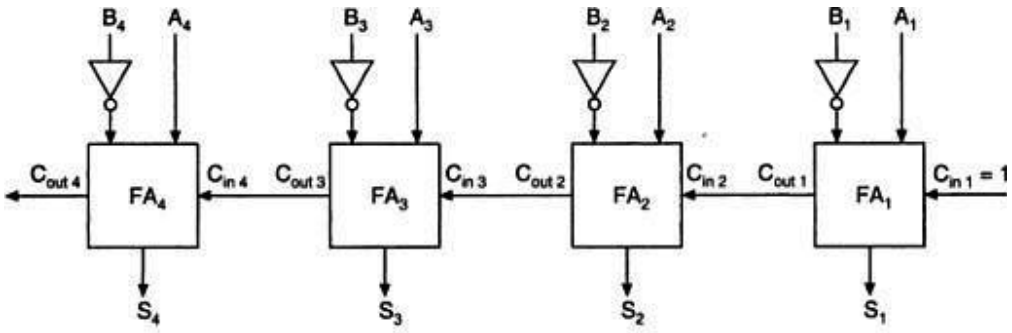
which lead to a time delay in the addition process. The carry propagation delay for each full-adder is the time between the application of the carry-in and the occurrence of the carry-out.

The 4-bit parallel adder, the sum (S_1) and carry-out (C_1) bits given by FA_1 are not valid, until after the propagation delay of FA_1. Similarly, the sum S_2 and carry-out (C_2) bits given by FA_2 are not valid until after the cumulative propagation delay of two full adders (FA_1 and FA_2) , and so on. At each stage ,the sum bit is not valid until after the carry bits in all the preceding stages are valid. Carry bits must propagate or ripple through all stages before the most significant sum bit is valid. Thus, the total sum (the parallel output) is not valid until after the cumulative delay of all the adders.

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.. The greater the number of bits that a ripple carry adder must add, the greater the time required for it to perform a valid addition. If two numbers are added such that no carries occur between stages, then the add time is simply the propagation time through a single full-adder.

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 inverters as

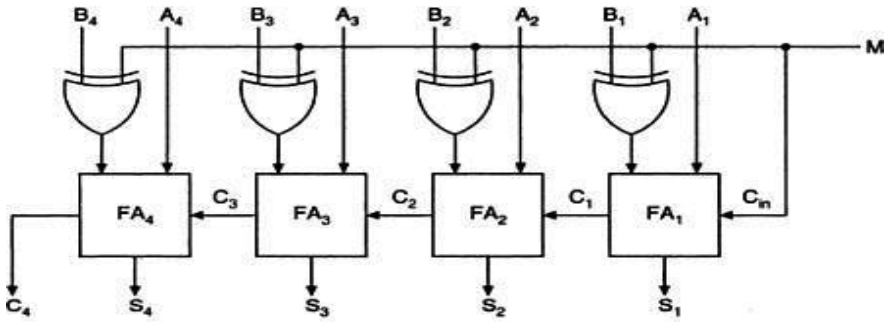


Logic diagram of a 4-bit parallel subtractor.

Binary-Adder Subtractor:

A 4-bit 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 mode input M controls the operation. When M=0, the circuit is an adder, and when M=1, the circuit becomes a subtractor. Each X-OR 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 $B \oplus 1 = B'$ and C_1=1. The B inputs are complemented and a 1 is through the input carry. The circuit performs the operation A plus the 2's complement of B.



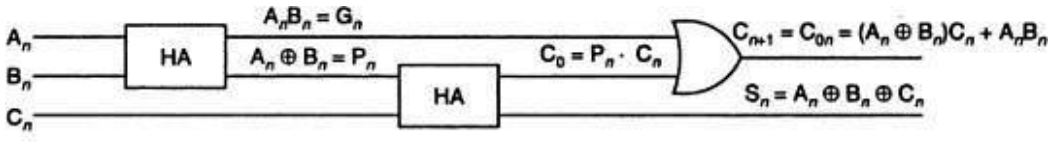
Logic diagram of a 4-bit binary adder-subtractor.

The Look-Ahead –Carry Adder:

In parallel-adder,the speed with which an addition can be performed is governed by the time required 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 and also generates the carry-in bits for all the stages simultaneously.

The method of speeding up the addition process is based on the two additional functions of the full-adder, called the carry generate and carry propagate functions.

Consider one full adder stage; say the nth stage of a parallel adder as shown in fig. we know that is made by two half adders and that the half adder contains an X-OR gate to produce the sum and an AND gate to produce the carry. If both the bits A_n and B_n are 1s, a carry has to be generated in this stage regardless of whether the input carry C_{in} is a 0 or a 1. This is called generated carry, expressed as $G_n = A_n.B_n$ which has to appear at the output through the OR gate as shown in fig.



A full adder (*n*th stage of a parallel adder).

Thereis another possibility of producing a carry out. X-OR gate inside the half-adder

at the input produces an intermediary sum bit- call it P_n –which is expressed as . Next P_n and C_n are added using the X-OR gate inside the second half adder to produce the final

sum bit and $S_n = P_n \oplus C_n$ where $P_n = A_n \oplus B_n$ **and output carry** $C_0 = P_n.C_n = (A_n \oplus B_n)C_n$ **which becomes carry for the (n+1) th stage.**

Consider the case of both P_n and C_n being 1. The input carry C_n has to be propagated to the output only if P_n is 1. If P_n is 0, even if C_n is 1, the and gate in the second half-adder will inhibit C_n . the carry out of the nth stage is 1 when either $G_n=1$ or $P_n.C_n=1$ or both G_n and $P_n.C_n$ are equal to 1.

For the final sum and carry outputs of the nth stage, we get the following Boolean expressions.

$$S_n = P_n \oplus C_n \text{ where } P_n = A_n \oplus B_n$$
$$C_{on} = C_{n+1} = G_n + P_n C_n \text{ where } G_n = A_n \cdot B_n$$

Observe the recursive nature of the expression for the output carry at the nth stage which becomes the input carry for the (n+1)st stage .it is possible to express the output carry of a higher significant stage is the carry-out of the previous stage.

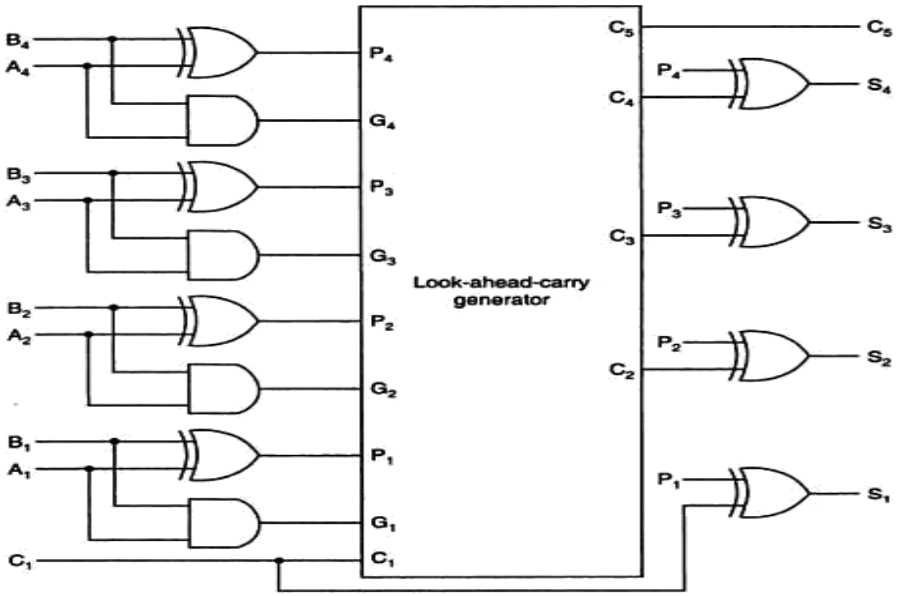
Based on these , the expression for the carry-outs of various full adders are as follows,

$$C_1 = G_0 + P_0 \cdot C_0$$
$$C_2 = G_1 + P_1 \cdot C_1 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$$
$$C_3 = G_2 + P_2 \cdot C_2 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0$$
$$C_4 = G_3 + P_3 \cdot C_3 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

The general expression for n stages designated as 0 through $(n-1)$ would be

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

Observe that the final output carry is expressed as a function of the input variables in SOP form. Which is two level AND-OR or equivalent NAND-NAND form. Observe that the full look-ahead scheme requires the use of OR gate with (n+1) inputs and AND gates with number of inputs varying from 2 to (n+1).



Logic diagram of a 4-bit look-ahead-carry adder.

2's complement Addition and Subtraction using Parallel Adders:

Most modern computers use the 2's complement system to represent negative numbers and to perform subtraction operations of signed numbers can be performed using only the addition operation ,if we use the 2's complement form to represent negative numbers.

The circuit shown can perform both addition and subtraction in the 2's complement. This adder/subtractor circuit is controlled by the control signal ADD/SUB'. When the ADD/SUB' level is HIGH, the circuit performs the addition of the numbers stored in registers A and B. When the ADD/Sub' level is LOW, the circuit subtract the number in register B from the number in register A. The operation is:

When ADD/SUB' is a 1:

1. **AND gates 1,3,5 and 7 are enabled , allowing B_0,B_1,$B_2$and B_3 to pass to the OR gates 9,10,11,12 . AND gates 2,4,6 and 8 are disabled , blocking B_0',B_1',B_2', and B_3' from reaching the OR gates 9,10,11 and 12.**
2. **The two levels B_0 to B_3 pass through the OR gates to the 4-bit parallel adder, to be added to the bits A_0 to A_3. The sum appears at the output S_0 to S_3**
3. **Add/SUB' =1 causes no carry into the adder.**

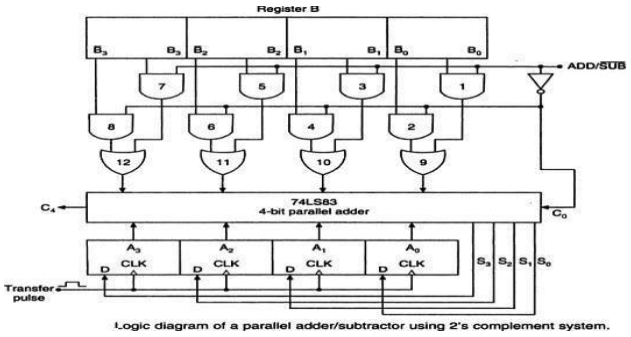
When ADD/SUB' is a 0:

1. **AND gates 1,3,5 and 7 are disabled , allowing B_0,B_1,$B_2$and B_3 from reaching the OR gates 9,10,11,12 . AND gates 2,4,6 and 8 are enabled , blocking B_0',B_1',B_2', and B_3' from reaching the OR gates.**

2. **The two levels B_0‘ to B_3‘ pass through the OR gates to the 4-bit parallel adder, to be added to the bits A_0 to A_3.The C_0 is now 1.thus the number in register B is converted to its 2‘s complement form.**

3. **The difference appears at the output S_0 to S_3.**

Adders/Subtractors used for adding and subtracting signed binary numbers. In computers , the output is transferred into the register A (accumulator) so that the result of the addition or subtraction always end up stored in the register A This is accomplished by applying a transfer pulse to the CLK inputs of register A.



Logic diagram of a parallel adder/subtractor using 2's complement system.

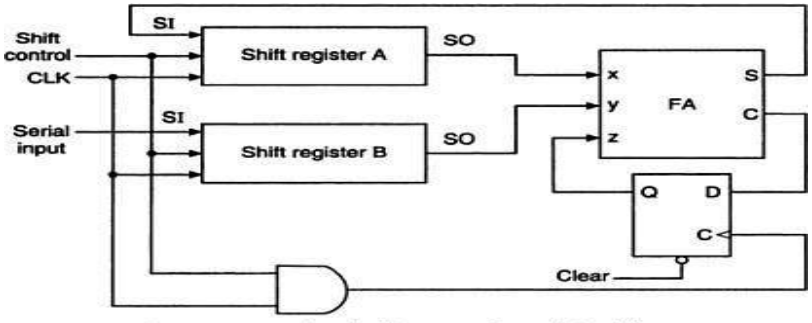
Serial Adder:

A serial adder is used to add binary numbers in serial form. The two binary numbers to be added serially are stored in two shift registers A and B. Bits are added one pair at a time through a single full adder (FA) circuit as shown. The carry out of the full-adder is transferred to a D flip-flop. The output of this flip-flop is then used as the carry input for the next pair of significant bits. The sum bit from the S output of the full-adder could be transferred to a third shift register. By shifting the sum into A while the bits of A are shifted out, it is possible to use one register for storing both augend and the sum bits. The serial input register B can be used to transfer a new binary number while the addend bits are shifted out during the addition.

The operation of the serial adder is:

Initially register A holds the augend, register B holds the addend and the carry flip-flop is cleared to 0. The outputs (SO) of A and B provide a pair of significant bits for the full-adder at x and y. The shift control enables both registers and carry flip-flop , so, at the clock pulse both registers are shifted once to the right, the sum bit from S enters the left most flip-flop of A , and the output carry is transferred into flip-flop Q . The shift control enables the registers for a number of clock pulses equal to the number of bits of the registers. For each succeeding clock pulse a new sum bit is transferred to A, a new carry is transferred to Q, and both registers are shifted once to the right. This process continues until the shift control is disabled. Thus the addition is accomplished by passing each pair of bits together with the previous carry through a single full adder circuit and transferring the sum, one bit at a time, into register A.

Initially, register A and the carry flip-flop are cleared to 0 and then the first number is added from B. While B is shifted through the full adder, a second number is transferred to it through its serial input. The second number is then added to the content of register A while a third number is transferred serially into register B. This can be repeated to form the addition of two, three, or more numbers and accumulate their sum in register A.



Logic diagram of a serial adder.

Difference between Serial and Parallel Adders:

The parallel adder registers with parallel load, whereas the serial adder uses shift registers. The number of full adder circuits in the parallel adder is equal to the number of bits in the binary numbers, whereas the serial adder requires only one full adder circuit and a carry flip-flop. Excluding the registers, the parallel adder is a combinational circuit, whereas the serial adder is a sequential circuit. The sequential circuit in the serial adder consists of a full-adder and a flip-flop that stores the output carry.

BCD Adder:

The BCD addition process:

1. **Add the 4-bit BCD code groups for each decimal digit position using ordinary binary addition.**
2. **For those positions where the sum is 9 or less, the sum is in proper BCD form and no correction is needed.**
3. **When 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. This will produce a carry to be added to the next decimal position.**

A BCD adder circuit must be able to operate in accordance with the above steps. In other words, the circuit must be able to do the following:

1. **Add two 4-bit BCD code groups, using straight binaryaddition.**

2. **Determine, if the sum of this addition is greater than 1101 (decimal 9); if it is , add 0110 (decimal 6) to this sum and generate a carry to the next decimalposition.**

The first requirement is easily met by using a 4- bit binary parallel adder such as the 74LS83 IC .For example , if the two BCD code groups $A_3A_2A_1A_0$and $B_3B_2B_1B_0$ are applied to a 4-bit parallel adder, the adder will output $S_4S_3S_2S_1S_0$, where S_4 is actually C_4 , the carry –out of the MSB bits.

The sum outputs $S_4S_3S_2S_1S_0$ can range anywhere from 00000 to 100109when both the BCD code groups are 1001=9). The circuitry for a BCD adder must include the logic needed to detect whenever the sum is greater than 01001, so that the correction can be added in. Those cases , where the sum is greater than 1001 are listed as:

S_4	S_3	S_2	S_1	S_0	Decimal number
0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
1	0	0	0	0	16
1	0	0	0	1	17
1	0	0	1	0	18

Let us define a logic output X that will go HIGH only when the sum is greater than 01001 (i.e, for the cases in table). If examine these cases ,see that X will be HIGH for either of the following conditions:

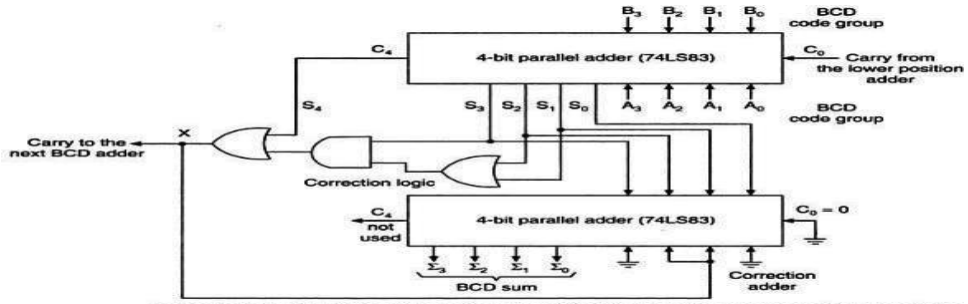
1. **Whenever S_4 =1(sum greater than 15)**
2. **Whenever S_3 =1 and either S_2 or S_1 or both are 1 (sum**

10 to 15) This condition can be expressed as

$X=S_4+S_3(S_2+S_1)$

Whenever X=1, it is necessary to add the correction factor 0110 to the sum bits, and to generate a carry. The circuit consists of three basic parts. The two BCD code groups $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ are added together in the upper 4-bit adder, to produce the sum $S_4S_3S_2S_1S_0$. The logic gates shown implement the expression for X. The lower 4-bit adder will add the correction 0110 to the sum bits, only when X=1, producing the final BCD sum output represented by $\sum_3\sum_2\sum_1\sum_0$. The X is also the carry-out that is produced when the sum is greater than 01001. When X=0, there is no carry and no addition of 0110. In such cases, $\sum_3\sum_2\sum_1\sum_0= S_3S_2S_1S_0$.

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, the carry-out of the second BCD adder is connected as the carry-in of the third BCD adder and so on.



Logic diagram of a BCD adder using two 4-bit adders and a correction-detector circuit.

EXCESS-3(XS-3) ADDER:

To perform Excess-3 additions,

1. **Add two xs-3 code groups**
2. **If carry=1, add 0011(3) to the sum of those two code groups**

If carry =0, subtract 0011(3) i.e., add 1101 (13 in decimal) to the sum of those two code groups.

Ex: Add 9 and 5

	1100	**9 in Xs-3**
	+1000	**5 in xs-3**
1	**0100**	**there is a carry**
+0011	**0011**	**add 3 to each group**
0100	**0111**	**14 in xs-3**
(1)	**(4)**	

EX:

(b)	0 1 1 1	4 in XS-3
	+0 1 1 0	3 in XS-3
	1 1 0 1	no carry
	+1 1 0 1	Subtract 3 (i.e. add 13)
Ignore carry	1 1 0 1 0	7 in XS-3
	(7)	

Implementation of xs-3 adder using 4-bit binary adders is shown. The augend ($A_3A_2A_1A_0$) and addend ($B_3B_2B_1B_0$) in xs-3 are added using the 4-bit parallel adder. If the carry is a 1, then 0011(3) is added to the sum bits $S_3S_2S_1S_0$ of the upper adder in the lower 4-bit parallel

adder. If the carry is a 0, then 1101(3) is added to the sum bits (This is equivalent to subtracting 0011(3) from the sum bits. The correct sum in xs-3 is obtained

Excess-3 (XS-3) Subtractor:
To perform Excess-3 subtraction,

1. **Complement the subtrahend**
2. **Add the complemented subtrahend to the minuend.**
3. **If carry =1, result is positive. Add 3 and end around carry to the result . If carry=0, the result is negative. Subtract 3, i.e, and take the 1's complement of the result.**

Ex: Perform 9-4

	1100	**9 in xs-3**
	+1000	**Complement of 4 n Xs-3**
	- - - - - - - - - -	
(1)	**0100**	**There is a carry**
	+0011	**Add 0011(3)**
	- - - - - - - - - -	
0111		
	1	**End around carry**
	- - - - - - - - - -	
1000		**5 in xs-3**

The minuend and the 1's complement of the subtrahend in xs-3 are added in the upper 4-bit parallel adder. If the carry-out from the upper adder is a 0, then 1101 is added to the sum bits of the upper adder in the lower adder and the sum bits of the lower adder are complemented to get the result. If the carry-out from the upper adder is a 1, then 3=0011 is added to the sum bits of the lower adder and the sum bits of the lower adder give the result.

Binary Multipliers:

In binary multiplication by the paper and pencil method, is modified somewhat in digital machines because a binary adder can add only two binary numbers at a time. In a binary multiplier, instead of adding all the partial products at the end, they are added two at a time and their sum accumulated in a register (the accumulator register). In addition, when the multiplier bit is a 0,0s are not written down and added because it does not affect the final result. Instead, the multiplicand is shifted left by one bit.

The multiplication of 1110 by 1001 using this process is

Multiplicand 1110		
Multiplier	**1001**	
	1110	**The LSB of the multiplier is a 1; write down the multiplicand; shift the multiplicand one position to the left (11100)**
	1110	**The second multiplier bit is a 0; write down the previous result 1110; shift the multiplicand to the left again (1 1 1 0 0 0)**

0000 **The fourth multiplier bit is a 1 write down the new multiplicand add it to the first partial product to obtain the final product.**

1111110

This multiplication process can be performed by the serial multiplier circuit , which multiplies two 4-bit numbers to produce an 8-bit product. The circuit consists of following elements

X register: A 4-bit shift register that stores the multiplier --- it will shift right on the falling edge of the clock. Note that 0s are shifted in from the left.

B register: An 8-bit register that stores the multiplicand; it will shift left on the falling edge of the clock. Note that 0s are shifted in from the right.

A register: An 8-bit register, i.e, the accumulator that accumulates the partial products.

Adder:An 8-bit parallel adder that produces the sum of A and B registers. The adder outputs S_7 through S_0 are connected to the D inputs of the accumulator so that the sum can be transferred to the accumulator only when a clock pulse gets through the AND gate.

The circuit operation can be described by going through each step in the multiplication of 1110 by 1001. The complete process requires 4 clock cycles.

1. Before the first clock pulse: Prior to the occurrence of the first clock pulse, the register A is loaded with 00000000, the register B with the multiplicand 00001110, and the register X with the multiplier 1001. Assume that each of these registers is loaded using its asynchronous inputs(i.e., PRESET and CLEAR). The output of the adder will be the sum of A and B,i.e., 00001110.

2. First Clock pulse:Since the LSB of the multiplier (X_0) is a 1, the first clock pulse gets through the AND gate and its positive going transition transfers the sum outputs into the accumulator. The subsequent negative going transition causes the X and B registers to shift right and left, respectively. This produces a new sum of A and B.

3. Second Clock Pulse: The second bit of the original multiplier is now in X_0 . Since this bit is a 0, the second clock pulse is inhibited from reaching the accumulator. Thus, the sum outputs are not transferred into the accumulator and the number in the accumulator does not change. The negative going transition of the clock pulse will again shift the X and B registers. Again a new sum is produced.

4. Third Clock Pulse:The third bit of the original multiplier is now in X_0;since this bit is a 0, the third clock pulse is inhibited from reaching the accumulator. Thus, the sum outputs are not transferred into the accumulator and the number in the accumulator does not change. The negative going transition of the clock pulse will again shift the X and B registers. Again a new sum is produced.

5.Fourth Clock Pulse: The last bit of the original multiplier is now in X_0 , and since it is a 1, the positive going transition of the fourth pulse transfers the sum into the accumulator. The accumulator now holds the final product. The negative going transition of the clock pulse shifts X and B again. Note that, X is now 0000, since all the multiplier bits have been shifted out.

Code converters:

The availability of a large variety of codes for the same discrete elements of information results in the use of different codes by different digital systems. It is sometimes necessary to use the output of one system as the input to another. A conversion circuit must be inserted between the two systems if each uses different codes for the same information. Thus a

code converter is a logic circuit whose inputs are bit patterns representing numbers (or character) in one cod and whose outputs are the corresponding representation in a different code. Code converters are usually multiple output circuits.

To convert from binary code A to binary code B, the input lines must supply the bit combination of elements as specified by code A and the output lines must generate the corresponding bit combination of code B. A combinational circuit performs this transformation by means of logic gates.

For example, a binary –to-gray code converter has four binary input lines B_4, B_3,B_2,B_1 and four gray code output lines G_4,G_3,G_2,G_1. When the input is 0010, for instance, the output should be 0011 and so forth. To design a code converter, we use a code table treating it as a truth table to express each output as a Boolean algebraic function of all the inputs.

In this example, of binary –to-gray code conversion, we can treat the binary to the gray code table as four truth tables to derive expressions for G_4, G3, G2, and G1. Each of these four expressions would, in general, contain all the four input variables B_4, B3,B_2,and B1. Thus,this code converter is actually equivalent to four logic circuits, one for each of the truth tables.

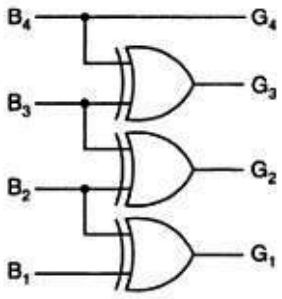
The logic expression derived for the code converter can be simplified using the usual techniques, including ‗don't cares' if present. Even if the input is an unweighted code, the same cell numbering method which we used earlier can be used, but the cell numbers -- must correspond to the input combinations as if they were an 8-4-2-1 weighted code. s

Design of a 4-bit binary to gray code converter:

$$G_4 = \Sigma m(8, 9, 10, 11, 12, 13, 14, 15) \qquad G_4 = B_4$$
$$G_3 = \Sigma m(4, 5, 6, 7, 8, 9, 10, 11) \qquad G_3 = \bar{B}_4 B_3 + B_4 \bar{B}_3 = B_4 \oplus B_3$$
$$G_2 = \Sigma m(2, 3, 4, 5, 10, 11, 12, 13) \qquad G_2 = \bar{B}_3 B_2 + B_3 \bar{B}_2 = B_3 \oplus B_2$$
$$G_1 = \Sigma m(1, 2, 5, 6, 9, 10, 13, 14) \qquad G_1 = \bar{B}_2 B_1 + B_2 \bar{B}_1 = B_2 \oplus B_1$$

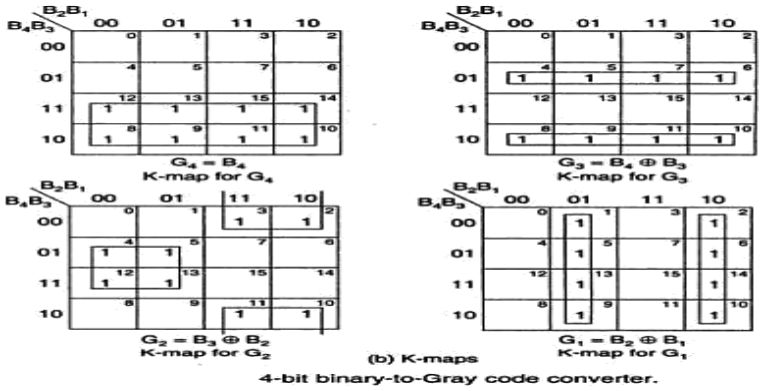
4-bit binary				4-bit Gray			
B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

(a) Conversion table



(c) Logic diagram

4-bit binary-to-Gray code converter



4-bit binary-to-Gray code converter.

Design of a 4-bit gray to Binary code converter:

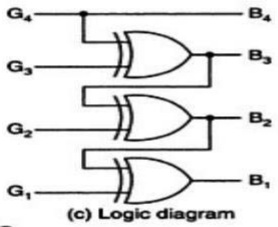
$$B_4 = \Sigma\, m(12, 13, 15, 14, 10, 11, 9, 8) = \Sigma\, m(8, 9, 10, 11, 12, 13, 14, 15)$$
$$B_3 = \Sigma\, m(6, 7, 5, 4, 10, 11, 9, 8) = \Sigma\, m(4, 5, 6, 7, 8, 9, 10, 11)$$
$$B_2 = \Sigma\, m(3, 2, 5, 4, 15, 14, 9, 8) = \Sigma\, m(2, 3, 4, 5, 8, 9, 14, 15)$$
$$B_1 = \Sigma\, m(1, 2, 7, 4, 13, 14, 11, 8) = \Sigma\, m(1, 2, 4, 7, 8, 11, 13, 14)$$

$$B_4 = G_4$$
$$B_3 = \bar{G}_4 G_3 + G_4 \bar{G}_3 = G_4 \oplus G_3$$
$$B_2 = \bar{G}_4 G_3 \bar{G}_2 + \bar{G}_4 \bar{G}_3 G_2 + G_4 \bar{G}_3 \bar{G}_2 + G_4 G_3 G_2$$
$$= \bar{G}_4 (G_3 \oplus G_2) + G_4 (\overline{G_3 \oplus G_2}) = G_4 \oplus G_3 \oplus G_2 = B_3 \oplus G_2$$
$$B_1 = \bar{G}_4 \bar{G}_3 \bar{G}_2 G_1 + \bar{G}_4 \bar{G}_3 G_2 \bar{G}_1 + \bar{G}_4 G_3 G_2 G_1 + \bar{G}_4 G_3 \bar{G}_2 \bar{G}_1 + G_4 G_3 \bar{G}_2 G_1$$
$$+ G_4 G_3 G_2 \bar{G}_1 + G_4 \bar{G}_3 G_2 G_1 + G_4 \bar{G}_3 \bar{G}_2 \bar{G}_1$$

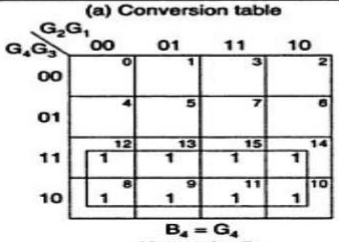
$$= \bar{G}_4 \bar{G}_3 (G_2 \oplus G_1) + G_4 G_3 (G_2 \oplus G_1) + \bar{G}_4 G_3 (\overline{G_2 \oplus G_1}) + G_4 \bar{G}_3 (\overline{G_2 \oplus G_1})$$
$$= (G_2 \oplus G_1)(\overline{G_4 \oplus G_3}) + (\overline{G_2 \oplus G_1})(G_4 \oplus G_3)$$
$$= G_4 \oplus G_3 \oplus G_2 \oplus G_1$$

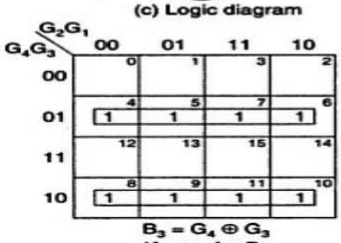
4-bit Gray				4-bit binary			
G_4	G_3	G_2	G_1	B_4	B_3	B_2	B_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

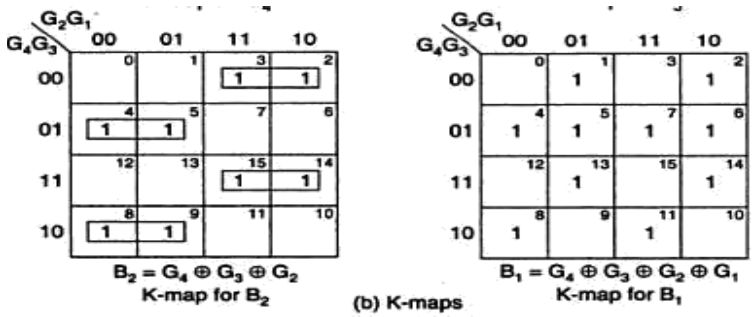
(a) Conversion table



(c) Logic diagram







(b) K-maps

4-bit Gray-to-binary code converter.

Design of a 4-bit BCD to XS-3 code converter:

8421 code				XS-3 code			
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

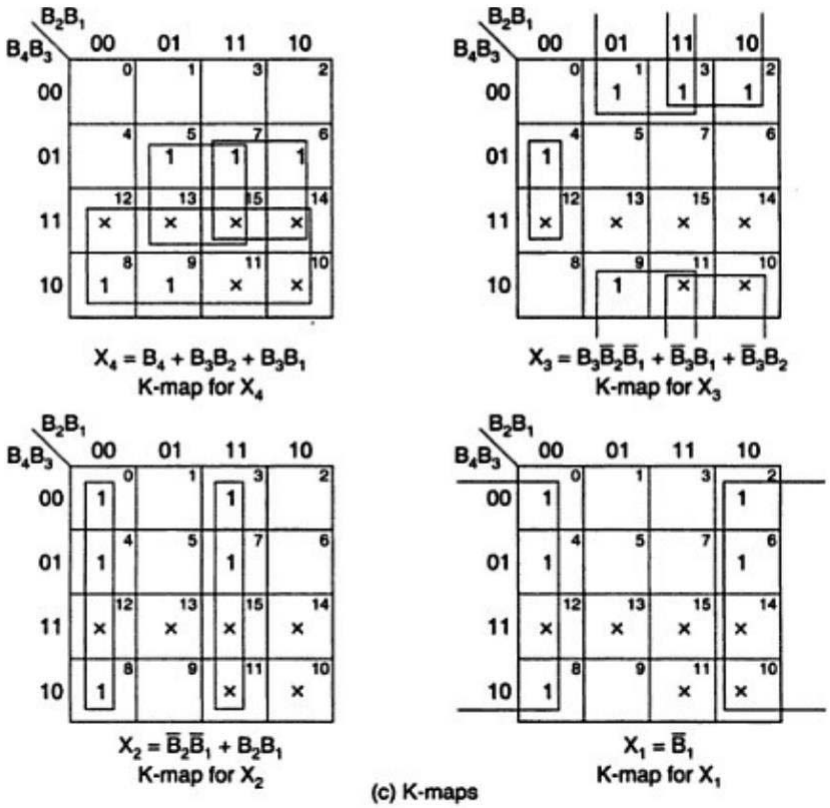
(a) Conversion table

$X_4 = \Sigma\, m(5, 6, 7, 8, 9) + d(10, 11, 12, 13, 14, 15)$
$X_3 = \Sigma\, m(1, 2, 3, 4, 9) + d(10, 11, 12, 13, 14, 15)$
$X_2 = \Sigma\, m(0, 3, 4, 7, 8) + d(10, 11, 12, 13, 14, 15)$
$X_1 = \Sigma\, m(0, 2, 4, 6, 8) + d(10, 11, 12, 13, 14, 15)$

The minimal expressions are
$X_4 = B_4 + B_3B_2 + B_3B_1$
$X_3 = B_3\bar{B}_2\bar{B}_1 + \bar{B}_3B_1 + \bar{B}_3B_2$
$X_2 = \bar{B}_2\bar{B}_1 + B_2B_1$
$X_1 = \bar{B}_1$

(b) Minimal expressions

4-bit BCD-to-XS-3 code converter



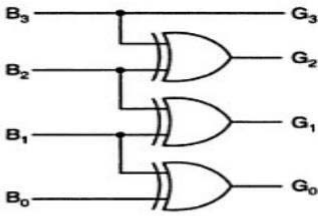
(c) K-maps

4-bit BCD-to-XS-3 code converter.

Design of a BCD to gray code converter:

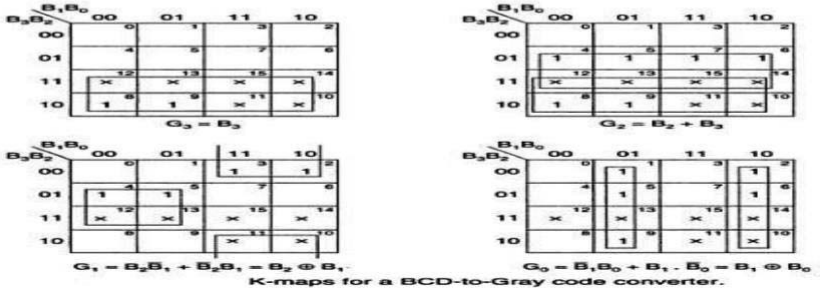
BCD code				Gray code			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1

(a) BCD-to-Gray code conversion table



(b) Logic diagram

BCD-to-Gray code converter.

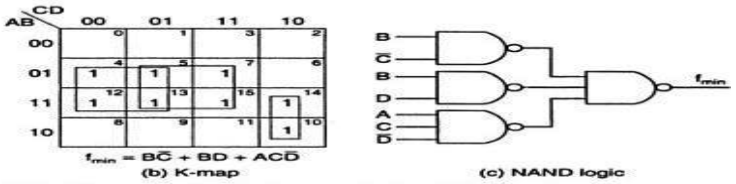


K-maps for a BCD-to-Gray code converter.

Design of a SOP circuit to Detect the Decimal numbers 5 through 12 in a 4-bit gray code Input:

Decimal number	4-bit Gray code A	B	C	D	Output f
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	1	0
3	0	0	1	0	0
4	0	1	1	0	0
5	0	1	1	1	1
6	0	1	0	1	1
7	0	1	0	0	1
8	1	1	0	0	1
9	1	1	0	1	1
10	1	1	1	1	1
11	1	1	1	0	1
12	1	0	1	0	1
13	1	0	1	1	0
14	1	0	0	1	0
15	1	0	0	0	0

(a) Truth table



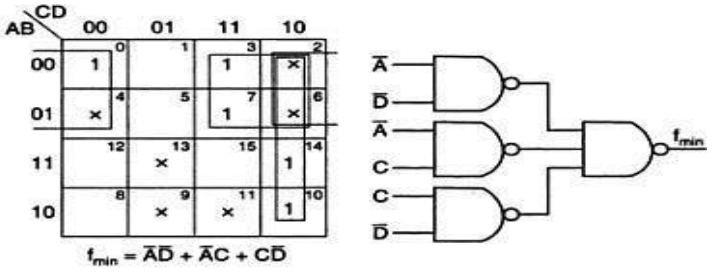
(b) K-map (c) NAND logic

Truth table, K-map and logic diagram for the SOP circuit.

Design of a SOP circuit to detect the decimal numbers 0,2,4,6,8 in a 4-bit 5211 BCD code input:

Decimal number	5211 code A	B	C	D	Output f
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	1	1
3	0	1	0	1	0
4	0	1	1	1	1
5	1	0	0	0	0
6	1	0	1	0	1
7	1	1	0	0	0
8	1	1	1	0	1
9	1	1	1	1	0

(a) Truth table



(b) K-map (c) Logic diagram

Truth table, K-map and logic diagram for the SOP circuit.

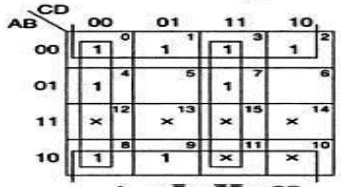
Design of a Combinational circuit to produce the 2's complement of a 4-bit binary number:

Input				Output			
A	B	C	D	E	F	G	H
0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	0
0	0	1	1	1	1	0	1
0	1	0	0	1	1	0	0
0	1	0	1	1	0	1	1
0	1	1	0	1	0	1	0
0	1	1	1	1	0	0	1
1	0	0	0	1	0	0	0
1	0	0	1	0	1	1	1
1	0	1	0	0	1	1	0
1	0	1	1	0	1	0	1
1	1	0	0	0	1	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	0	1	0
1	1	1	1	0	0	0	1

(a) Conversion table

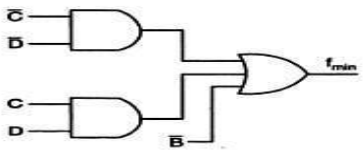
Conversion table and K-maps for the circuit

(a) Seven-segment display



$f_{min} = \bar{B} + \bar{C}\bar{D} + CD$

(b) K-map



(c) Logic diagram

Comparators:

$$\text{EQUALITY} = (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$

Block diagram of a 1-bit comparator.

1. Magnitude Comparator:

The logic for a 1-bit magnitude comparator: Let the 1-bit numbers be $A = A_0$ and $B = B_0$.
If $A_0 = 1$ and $B_0 = 0$, then $A > B$.
Therefore,

$$A > B: G = A_0\bar{B}_0$$

If $A_0 = 0$ and $B_0 = 1$, then $A < B$.
Therefore,

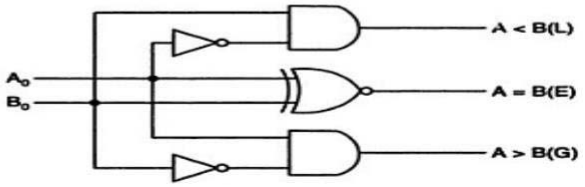
$$A < B: L = \bar{A}_0 B_0$$

If A_0 and B_0 coincide, i.e. $A_0 = B_0 = 0$ or if $A_0 = B_0 = 1$, then $A = B$.
Therefore,

$$A = B : E = A_0 \odot B_0$$

A_0	B_0	L	E	G
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

(a) Truth table



(b) Logic diagram

1-bit comparator.

1- bit Magnitude Comparator:

The logic for a 2-bit magnitude comparator: Let the two 2-bit numbers be $A = A_1 A_0$ and $B = B_1 B_0$.

1. If $A_1 = 1$ and $B_1 = 0$, then $A > B$ or
2. If A_1 and B_1 coincide and $A_0 = 1$ and $B_0 = 0$, then $A > B$. So the logic expression for $A > B$ is

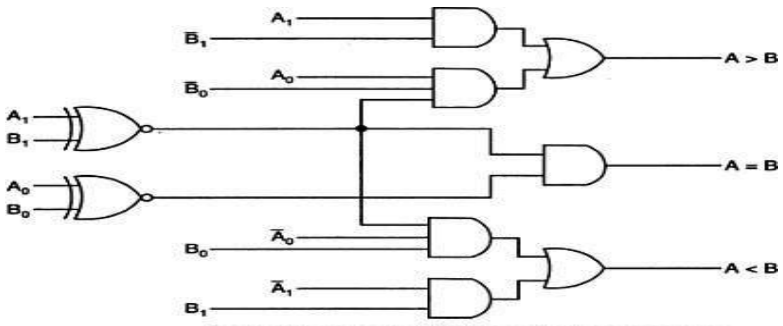
$$A > B : G = A_1\bar{B}_1 + (A_1 \odot B_1)A_0\bar{B}_0$$

1. If $A_1 = 0$ and $B_1 = 1$, then $A < B$ or
2. If A_1 and B_1 coincide and $A_0 = 0$ and $B_0 = 1$, then $A < B$. So the expression for $A < B$ is

$$A < B : L = \bar{A}_1 B_1 + (A_1 \odot B_1)\bar{A}_0 B_0$$

If A_1 and B_1 coincide and if A_0 and B_0 coincide then $A = B$. So the expression for $A = B$ is

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



Logic diagram of a 2-bit magnitude comparator.

4- Bit Magnitude Comparator:

The logic for a 4-bit magnitude comparator: Let the two 4-bit numbers be $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$.

1. If $A_3 = 1$ and $B_3 = 0$, then $A > B$. Or
2. If A_3 and B_3 coincide, and if $A_2 = 1$ and $B_2 = 0$, then $A > B$. Or
3. If A_3 and B_3 coincide, and if A_2 and B_2 coincide, and if $A_1 = 1$ and $B_1 = 0$, then $A > B$. Or
4. If A_3 and B_3 coincide, and if A_2 and B_2 coincide, and if A_1 and B_1 coincide, and if $A_0 = 1$ and $B_0 = 0$, then $A > B$.

From these statements, we see that the logic expression for $A > B$ can be written as

$$(A > B) = A_3\bar{B}_3 + (A_3 \odot B_3)A_2\bar{B}_2 + (A_3 \odot B_3)(A_2 \odot B_2)A_1\bar{B}_1 + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)A_0\bar{B}_0$$

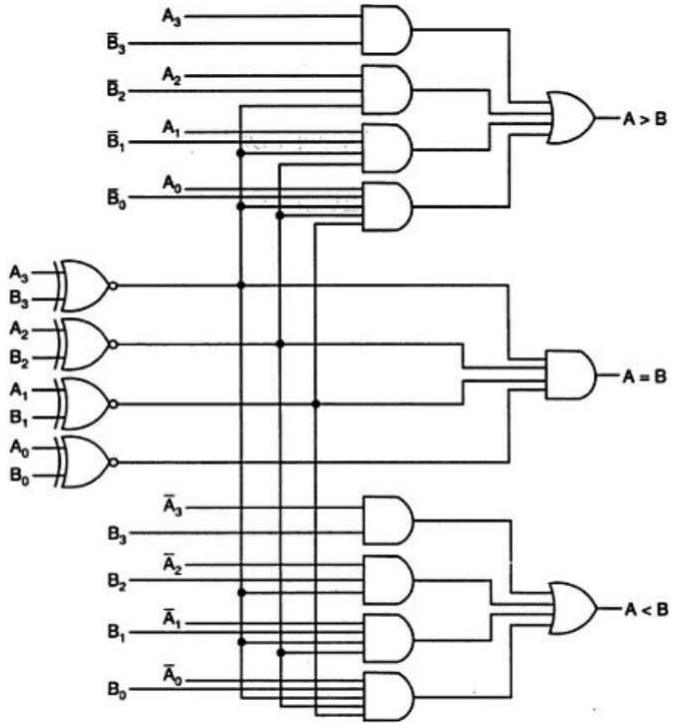
Similarly, the logic expression for $A < B$ can be written as

$$A < B = \bar{A}_3B_3 + (A_3 \odot B_3)\bar{A}_2B_2 + (A_3 \odot B_3)(A_2 \odot B_2)\bar{A}_1B_1 + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)\bar{A}_0B_0$$

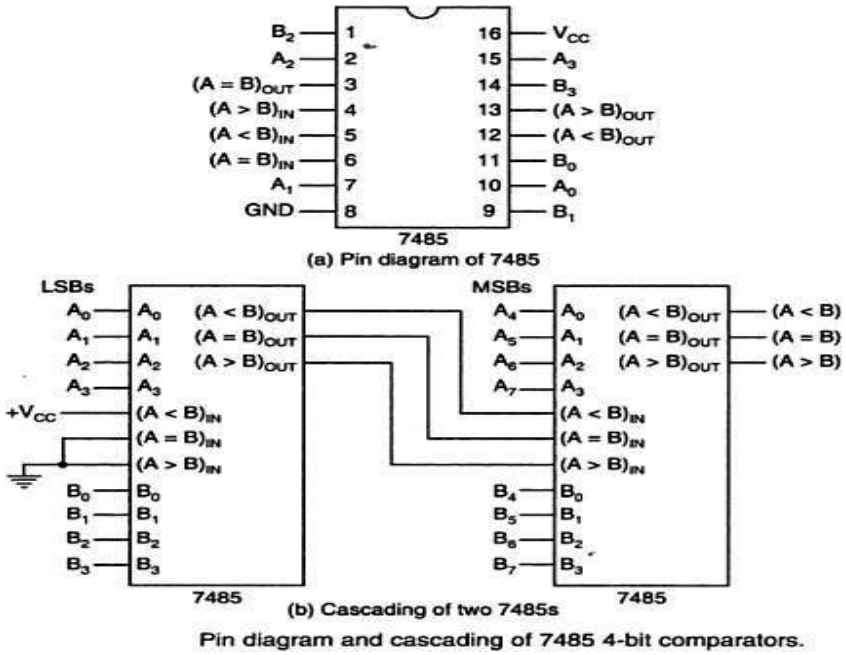
If A_3 and B_3 coincide and if A_2 and B_2 coincide and if A_1 and B_1 coincide and if A_0 and B_0 coincide, then $A = B$.

So the expression for $A = B$ can be written as

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

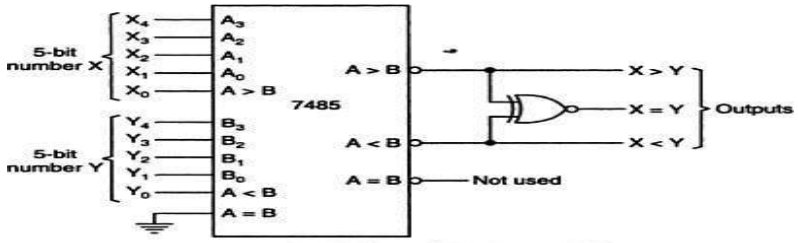


IC Comparator:

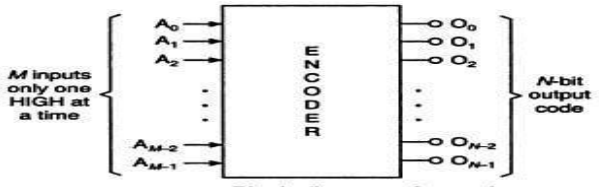


Pin diagram and cascading of 7485 4-bit comparators.

ENCODERS:



Use of 7485 as a 5-bit comparator.

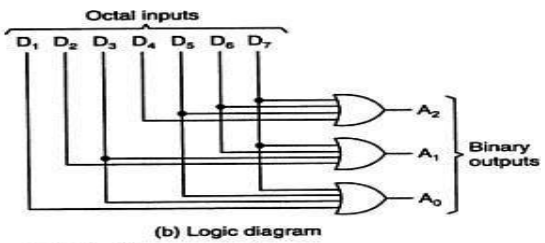


Block diagram of encoder.

Octal to Binary Encoder:

Octal digits		Binary		
		A_2	A_1	A_0
D_0	0	0	0	0
D_1	1	0	0	1
D_2	2	0	1	0
D_3	3	0	1	1
D_4	4	1	0	0
D_5	5	1	0	1
D_6	6	1	1	0
D_7	7	1	1	1

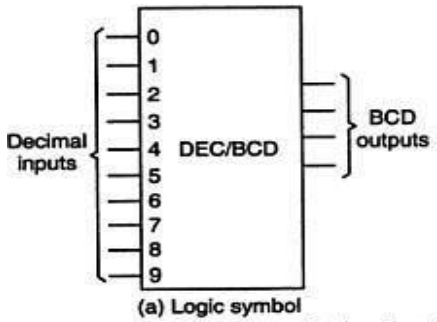
(a) Truth table



(b) Logic diagram

Octal-to-binary encoder.

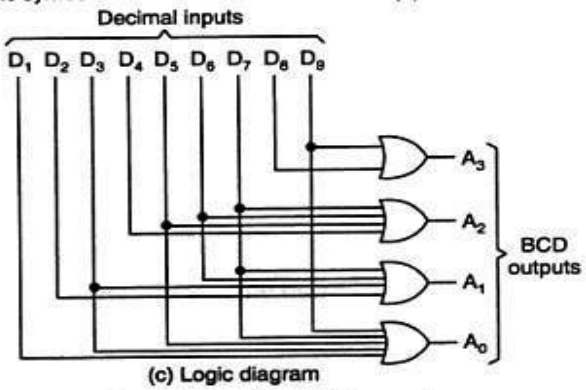
Decimal to BCD Encoder:



(a) Logic symbol

Decimal inputs		Binary			
		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

(b) Truth table



(c) Logic diagram

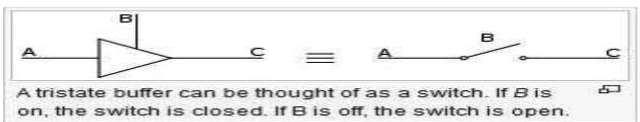
Decimal-to-BCD encoder.

Tristate bus system:

In three-state, tri-state, or 3-statelogic allows an output port to assume a high impedance state in addition to the 0 and 1 logic levels, effectively removing the output from the circuit.

This allows multiple circuits to share the same output line or lines (such as a bus which cannot listen to more than one device at a time).

Three-state outputs are implemented in many registers, bus drivers, and flip-flops in the 7400 and 4000 series as well as in other types, but also internally in many integrated circuits. Other typical uses are internal and external buses in microprocessors, computer memory, and peripherals. Many devices are controlled by an active-low input called OE (Output Enable) which dictates whether the outputs should be held in a high-impedance state or drive their respective loads (to either 0- or 1-level).



A tristate buffer can be thought of as a switch. If *B* is on, the switch is closed. If B is off, the switch is open.

INPUT		OUTPUT
A	B	C
0	1	0
1		1
X	0	Z (high impedance)