❖ Boolean Algebra

There are some important rules must be considered to understand the Boolean algebra which are:

1- $A + 1 = 1$, which does not mean addition operation, it means *OR* operation.

2- $A + 0 = A$

3- $\bar{A}$ means the inversion of A

4- $\bar{\bar{A}} = A$

5- $A + A = A$

6- $A.A = A$

7- $\bar{A}.A = 0$

8- $\bar{A} + A = 1$

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

Ex3/ Prove that $A + \bar{A}B = A + B$.

Sol: take the right side

$A(1 + B) + \bar{A}B = A + AB + \bar{A}B$ { since $(1 + B) = 1$ }

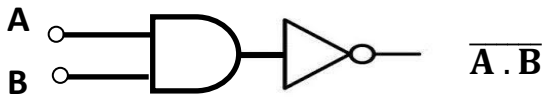
$\rightarrow A + AB + \bar{A}B = A + B(A + \bar{A})$

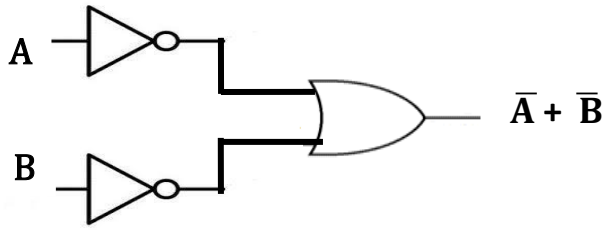
$= A + B$

❖ De- Morgan's Theorem

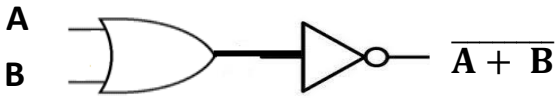
De-Morgan's Theorems are two additional simplification techniques, which used to simplify Boolean expressions. It is important to note that the more simple Boolean expression gives simple logic circuit.

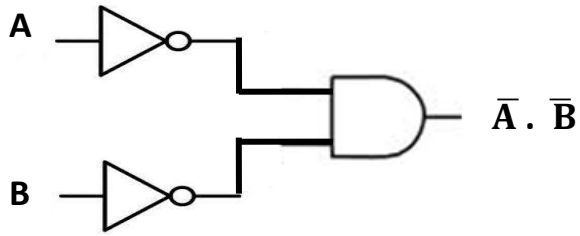
1- $\overline{(A \cdot B)} = \overline{(A)} + \overline{(B)}$



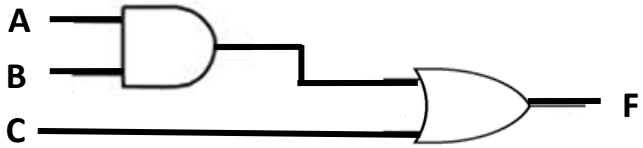


2- $\overline{A + B} = \overline{A} \cdot \overline{B}$





Ex4/ Write the Boolean expression for the following logic circuit.



Sol: F = A . B + C

Ex 5/ Simplify the following Boolean algebra

$Y = \overline{A} + AB + ACB + CA + \overline{B} + \overline{C}$

Sol:

$\overline{A} + AB + ACB + CA + \overline{B} + \overline{C} = \overline{A}(1+B) + AB + AC(1+B) + \overline{B} + \overline{C}$ (1 over $\overline{A}$, 1 over AC)

$= \overline{A} + \overline{A}B + AB + AC + \overline{B} + \overline{C} \Longrightarrow Y = \overline{A} + B(A + A) + AC + \overline{C}(1+A) + \overline{B}$ (1 over B, 1 over $\overline{C}$)

$= \overline{A} + B + AC + \overline{C} + \overline{C}A + \overline{B} \Longrightarrow Y = \overline{A} + B + A(C + \overline{C}) + \overline{C} + \overline{B}$ (1 over A)

$= \overline{A} + B + A + \overline{C} + \overline{B} \Longrightarrow Y = (\overline{A} + A) + (B + \overline{B}) + \overline{C} \Longrightarrow Y = 1 + \overline{C} = 1$

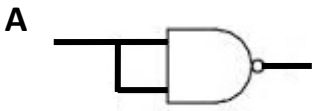
Ex6/ using *NAND* and *NOR* gates to build other logic gates.

Sol:

First using *NAND* gates:

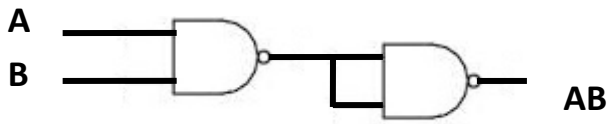
1- *NOT* gate

NOT gate can be built easily by shorting the two terminal of the *NAND* gate.



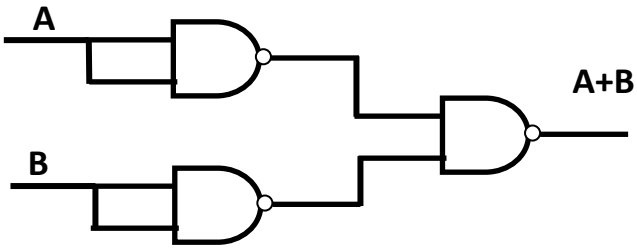
2- *AND* gate

AND gate can be obtained only by adding inverter *NAND* gate to the *NAND* gate as shown below.



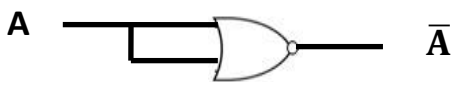
3- *OR* gate

OR gate can be represented by three *NAND* gates.

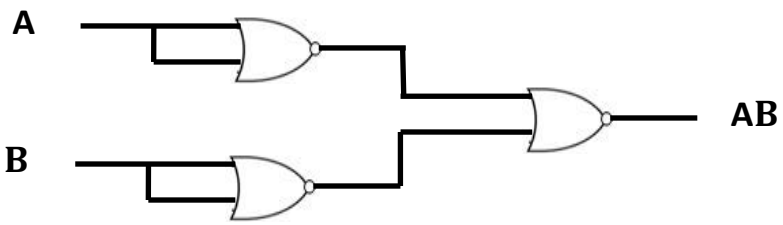


Second using *NOR* gates:

1- *NOT* gate



2- *AND* gate

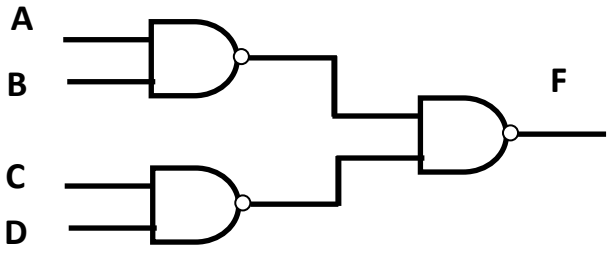


3- *OR* gate

Ex7/ Represent the following Boolean expression using only *NAND* gates. F = AB + CD

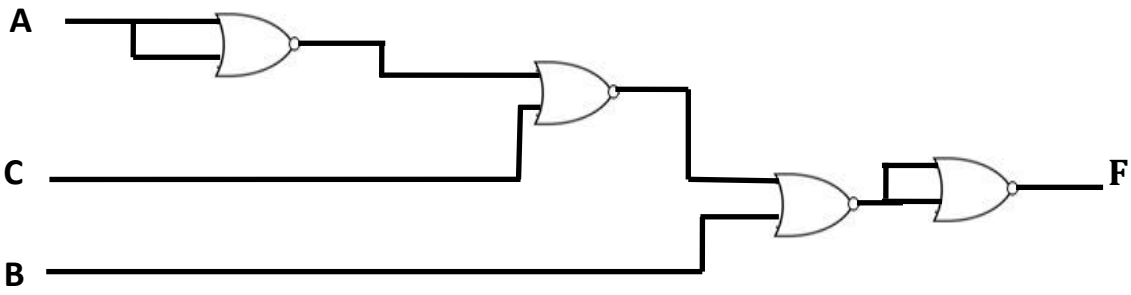
Sol:

$F = \overline{\overline{AB} + \overline{CD}} \Longrightarrow F = \overline{\overline{AB} \cdot \overline{CD}}$ the logic circuit is shown below



Ex8/draw the logic circuit to implement the following function {F = A $\bar{C}$+B} using *NOR* gates only.

Sol:



HW1: Build the following expression $F = A \oplus B$.

HW2: Implement F = ABC + AD using *NOR* gates only

HW3: Express [*NOT, AND,* and *OR*] gates using *NOR* gates only.

❖ ***Primary Multiplication***

The primary multiplication is used to express the equivalent symbols of the truth table states, for example if there are four variables then the primary multiplication of { *000, 110, 111,* and *001*} is { $\bar{A}\bar{B}\bar{C}$, $AB\bar{C}$, ABC,$\bar{A}\bar{B}C$}. This applied for two, four… etc.

❖ ***Electronic logic circuits***

The logic circuits can be divided into two categories, combinational and sequential logic circuits. In the combinational logic circuit, the output depends on the inputs to the circuit while in the sequential logic circuit; the output depends on the inputs and the past value. To design the combinational logic circuits there are two methods, sum of product (*SoP*) and product of sum (*PoS*).

❖ ***Sum of Product(SoP)***

To design any combinational logic circuit, the following steps must be applied.

1- Determine the numbers of inputs and outputs to the logic cct.
2- Draw the truth table of the logic cct.
3- Take the cases that gives ones logic (*1*) at the outputs.
4- Write the primary multiplication of the inputs that give logic (*1*) outputs, each logic (*1*) inputs represent without dash(-) while each (*0*) represent with dash.
5- Add primary multiplication to each other to obtain the canonical form (*SoP*), and then simplify the final expression.

Ex 9/ Draw the logic circuit which has the following truth table.

I/P			O/P
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Sol: To design the logic circuit, the simplified Boolean expression must be found.

$Y = \bar{A}\bar{B}C + ABC \Longrightarrow Y = C(\bar{A}\bar{B} + AB)$

$= C(A \odot B)$

The logic circuit is shown in figure (1):

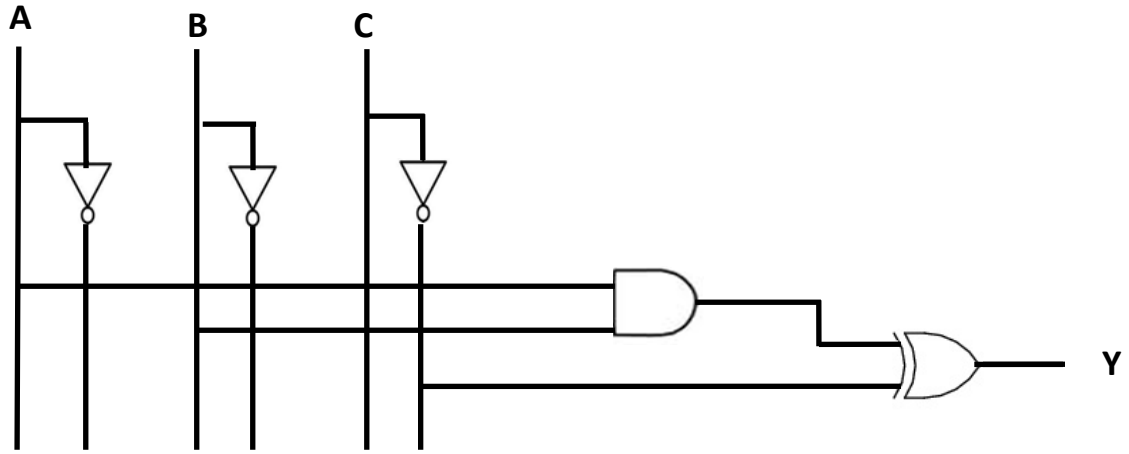


Fig 1

❖ ***Product of Sum (PoS)***

The product of sum is another way to simplify the Boolean expression of any function. In this method zeros is taken instead of ones. It must be noted that the final expressions of the outputs for (*SoP*) and (*PoS*) are the same as will be illustrated in example (*9*).

Ex10/ Find the outputs of the following truth table by using (*SoP*) and (*PoS*).

A	B	Y
0	0	1
0	1	0
1	0	1
1	1	1

Sol:

For (*SoP*)

$Y = \Sigma (0, 2, 3) = \bar{A}\,\bar{B} + A\bar{B} + AB$

$= A(\bar{B} + B) + \bar{A}\,\bar{B}$ (1)

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

$= (\bar{A} + B)$ { See Boolean Algebra point 9}

For (PoS)

$Y = \pi (0)$

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

Ex11/ Convert then simplify the following Boolean expression from (*PoS*) to (*SoP*).

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

Sol: easily by multiplying the two brackets by each other then

$Y = [A\bar{A} + AB + A\bar{C} + B\bar{A} + BB + B\bar{C} + \bar{C}\bar{A} + \bar{C}B + \bar{C}\bar{C}]$

$= B(A+\bar{A}) + \bar{C}\,(A+\bar{A}) + B(1+\bar{C}) + \bar{C}(1+B)$

$= B + \bar{C} + B + \bar{C} \Longrightarrow Y = B + \bar{C}$

Ex11/ design a logic circuit that multiply two words with (*2bits*)

Sol:

Let first word (*A*) and the second words (*B*) then the truth table will be

A_1A_2	B_1B_2	W X Y Z
0 0	0 0	0 0 0 0
0 0	0 1	0 0 0 0
0 0	1 0	0 0 0 0
0 0	1 1	0 0 0 0
0 1	0 0	0 0 0 0
0 1	0 1	0 0 0 1
0 1	1 0	0 0 1 0
0 1	1 1	0 0 1 1
1 0	0 0	0 0 0 0
1 0	0 1	0 0 1 0
1 0	1 0	0 1 0 0
1 0	1 1	0 1 1 0
1 1	0 0	0 0 0 0
1 1	0 1	0 0 1 1
1 1	1 0	0 1 1 0
1 1	1 1	1 0 0 1

The equations of the output are given below:

$W = ABCD$, $X = A\bar{B}C\bar{D} + A\bar{B}CD + ABC\bar{D}$

$Y = \bar{A}BCD + \bar{A}BC\bar{D} + A\bar{B}\bar{C}D + A\bar{B}CD + AB\bar{C}D + ABC\bar{D}$

$Z = A\bar{B}\bar{C}D + \bar{A}BCD + AB\bar{C}D + ABCD$

The final step of the design is the drawing of the logic gate circuit as shown in figure (2)

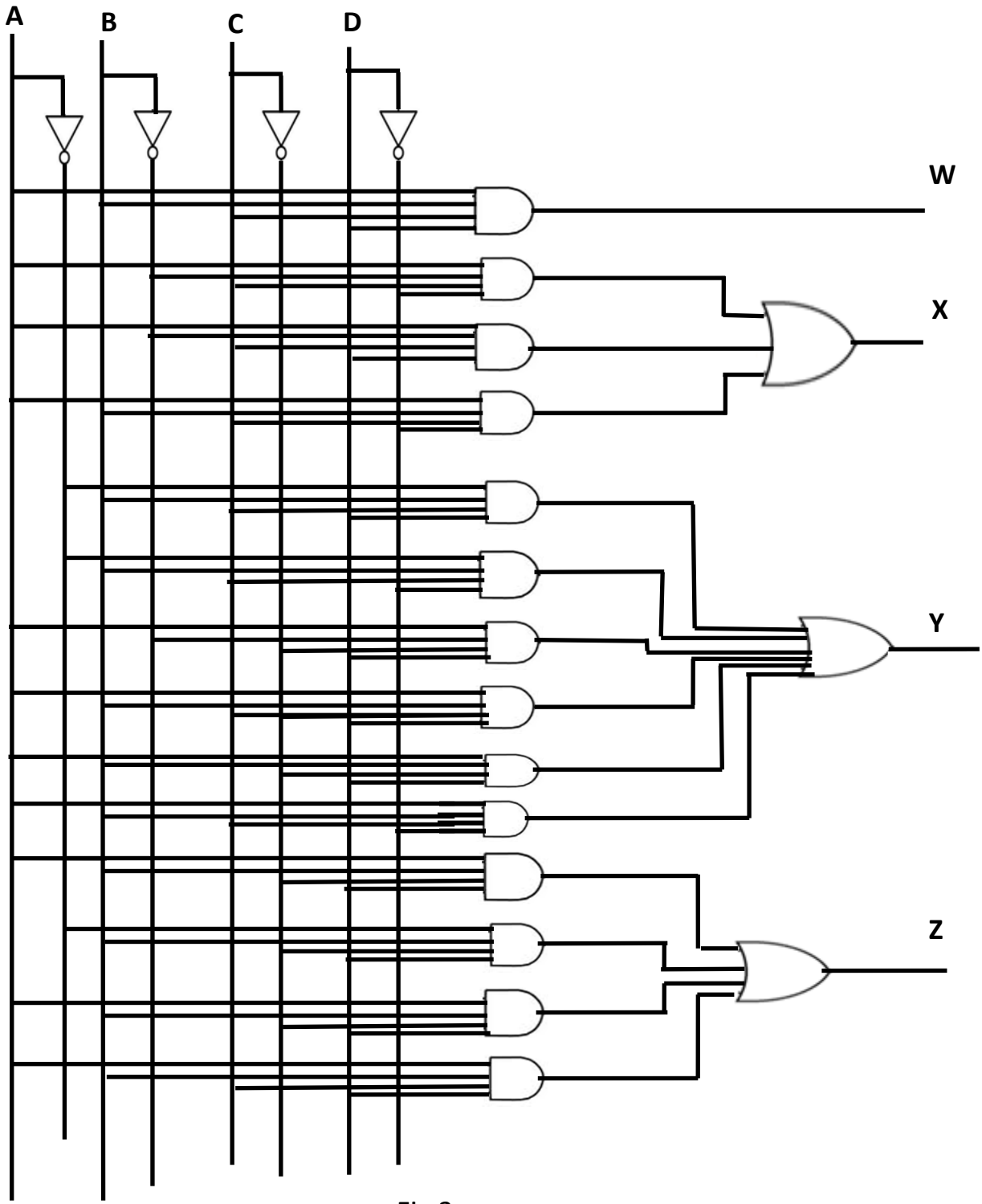


Fig 2

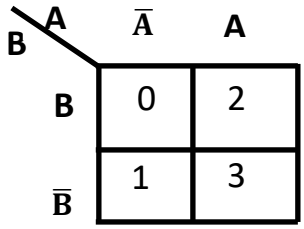
❖ Karnaugh Maps (*K- Maps*)

This is a graphical approach to finding suitable product terms for using sum of product expressions. The map is useful for problems of up to six variables and is particularly straightforward for most problems of three or four variables. Although there is no guarantee of finding a minimum solution, the methods we will develop nearly always produce a minimum. The main difference between Boolean simplification and (*K- Map*) is that (*K- Map)* gives the most simplified expression for the output of the truth table. There are many types of (*K- Maps*) depending on the numbers of inputs.

1- Two inputs

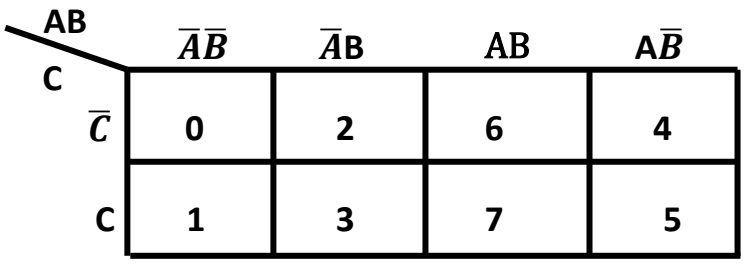
For two variables *A* and *B* the (*K- Map*) is given below for the following truth table

NOS.	A	B
0	0	0
1	0	1
2	1	0
3	1	1



2- Three input

The truth table and (*K- Map*) for three inputs is given below



NOS.	A	B	C
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

3- Four inputs

The truth table and (*K- Map*) for four inputs is given below

NOS.	A	B	C	D
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1

AB / CD	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	0	4	12	8
$\bar{C}D$	1	5	13	9
CD	2	6	15	11
$C\bar{D}$	3	7	14	10

14	1	1	1	0
15	1	1	1	1

❖ Important Notes:

1- To find the Boolean, the neighboring ones are taken with each other's.

2- Using (*K- Map*) to find the outputs gives the most simplified expression.

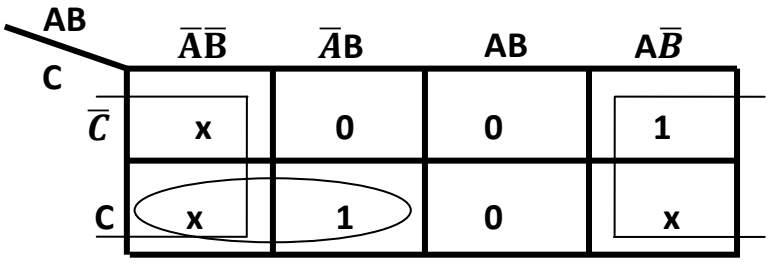
3- There are some cases cannot be obtained in real case such as the presence of (*1101*) in (*BCD*) or (<) and (>) at the same time. These cases are called don't care cases which is denoted by (*x*).

4- The value of (*x)* can be one or zero depend on the neighboring cells in the (*K- Map*).

5- *K- Map* can be used with (*PoS*) by taking the neighboring zeros instead of ones to obtain inverting of output function, then taking the compliment of result to find the final outputs.

Ex12/ for the following (*K- Maps*) find the outputs expressions.

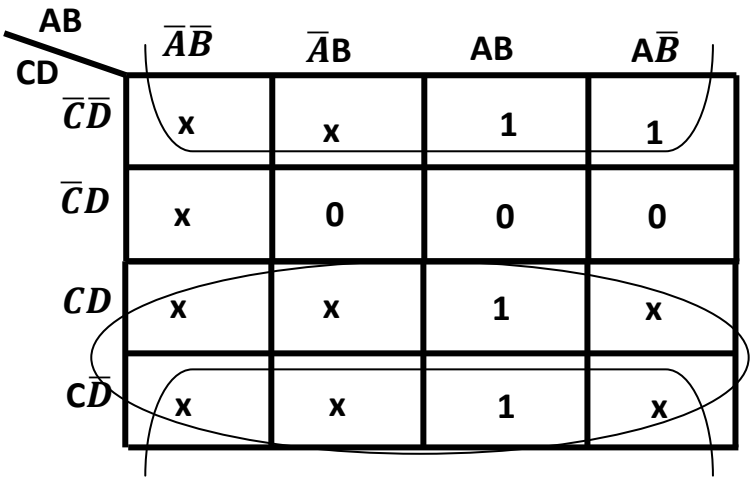
1- Three inputs



The output equation is

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

2- Four inputs



From *K- Map* it can be obtained that

$Y = C + \bar{D}$

Ex13/ design a logic circuit for the following

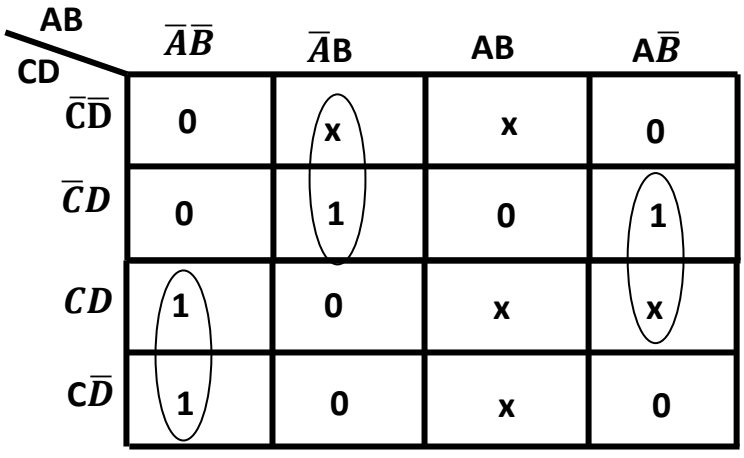
$Y = \Sigma(2, 3, 5, 9)$

Don't care = Σ (4, 12, 14, 15, 11)

Sol:

SoP is used in this equation, then the truth table and *K- Map* is given as

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	X
0	1	0	1	1



0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	X
1	1	0	0	X
1	1	0	1	0
1	1	1	0	X
1	1	1	1	X

The equation of the logic circuit is [Y = $\bar{A}\bar{B}$C + $\bar{A}$B$\bar{C}$+ A$\bar{B}$D] and the logic circuit is given in figure (3)

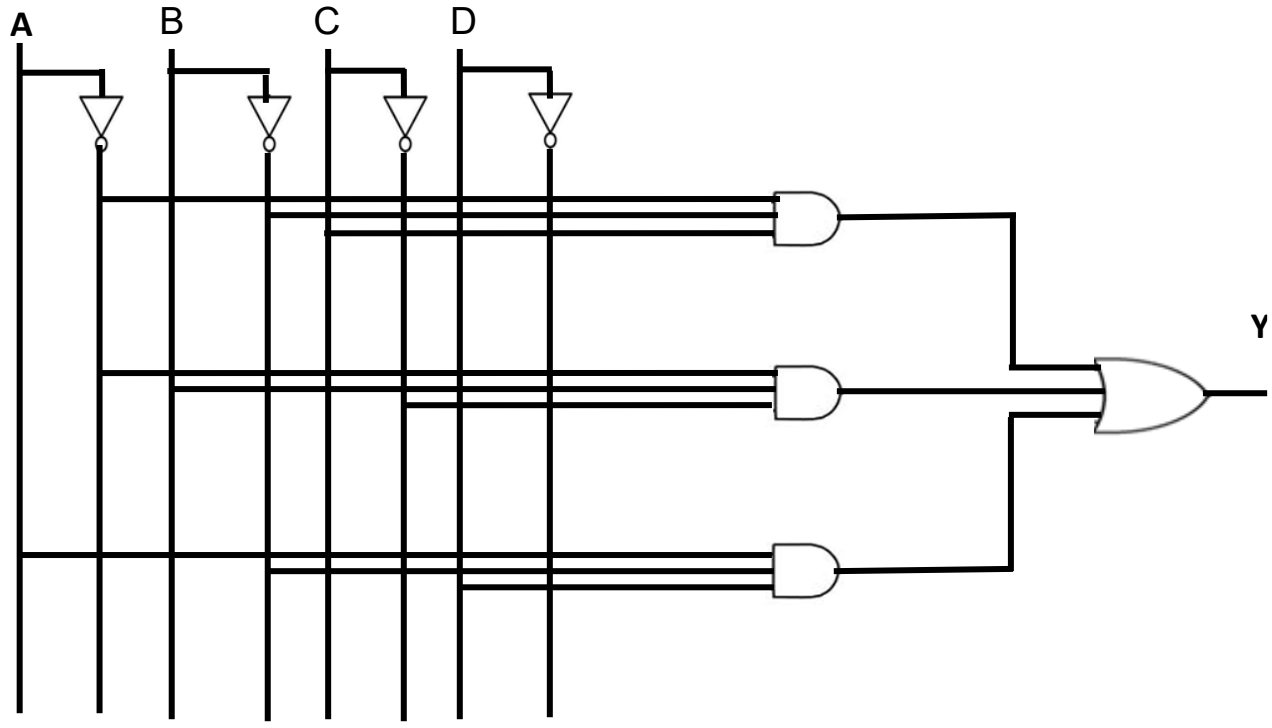


Fig 3

Ex$_{14}$/ design a logic circuit that has the following equations

$F_1 = \pi (2,3)$, $F_2 = \Sigma (2,3)$.

Sol:

By assuming that the inputs are (A and B) the equations (F_1, F_2) can be written as:

$F_1 = (A + \overline{B})(A+B)$

$= A + A\overline{B}$

$= A$

$F_2 = A\overline{B} + AB$

$= A$

This means that $F_1 = F_2$, the logic circuits that gives F_1 and F_2 are given below

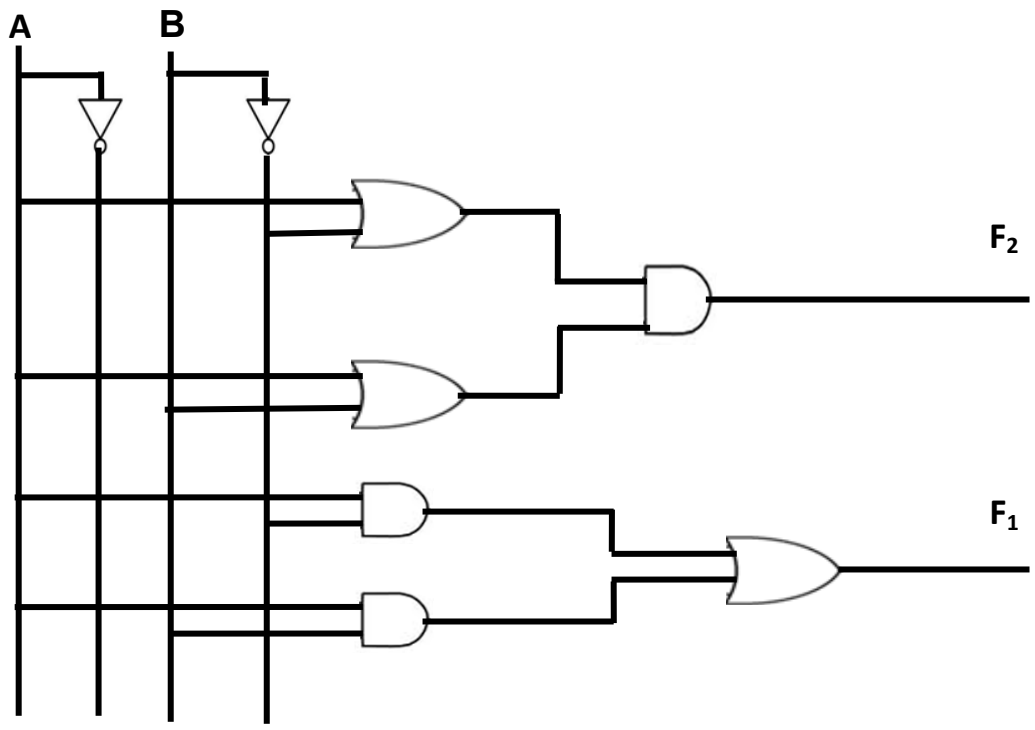


Fig 4

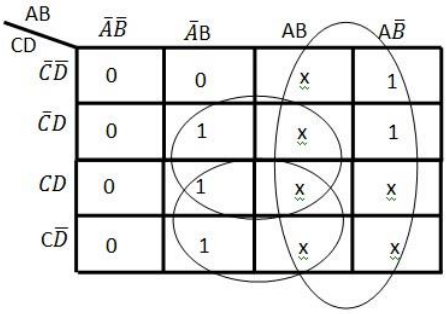
HW4: using *K- Map* to find the simplified expression for the following function, and then draw the logic circuit that implements it in both methods using PoS with Boolean algebra and *SoP* with *K- Map*

F = π [*1, 2, 4*] & don't care is [*5,7*]

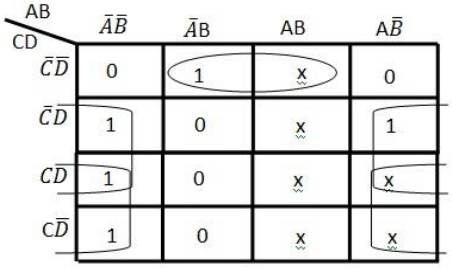
Ex 15/ design a logic circuit that converts from (*BCD*) code to (*Excess – three*) code.

Sol: The truth table has four inputs and four outputs

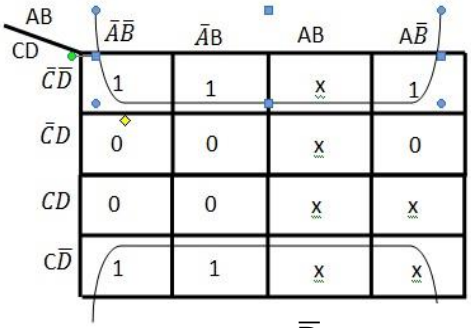
inputs				outputs			
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X



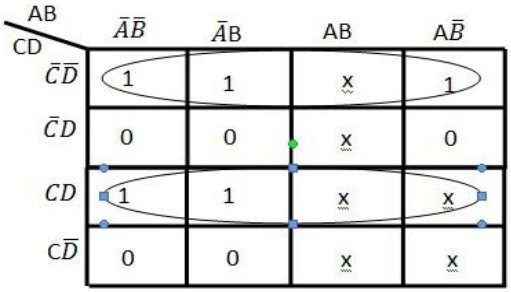
W = A + BD+BC



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



Y = $\overline{D}$



X = $\overline{C}\overline{D}$ +CD

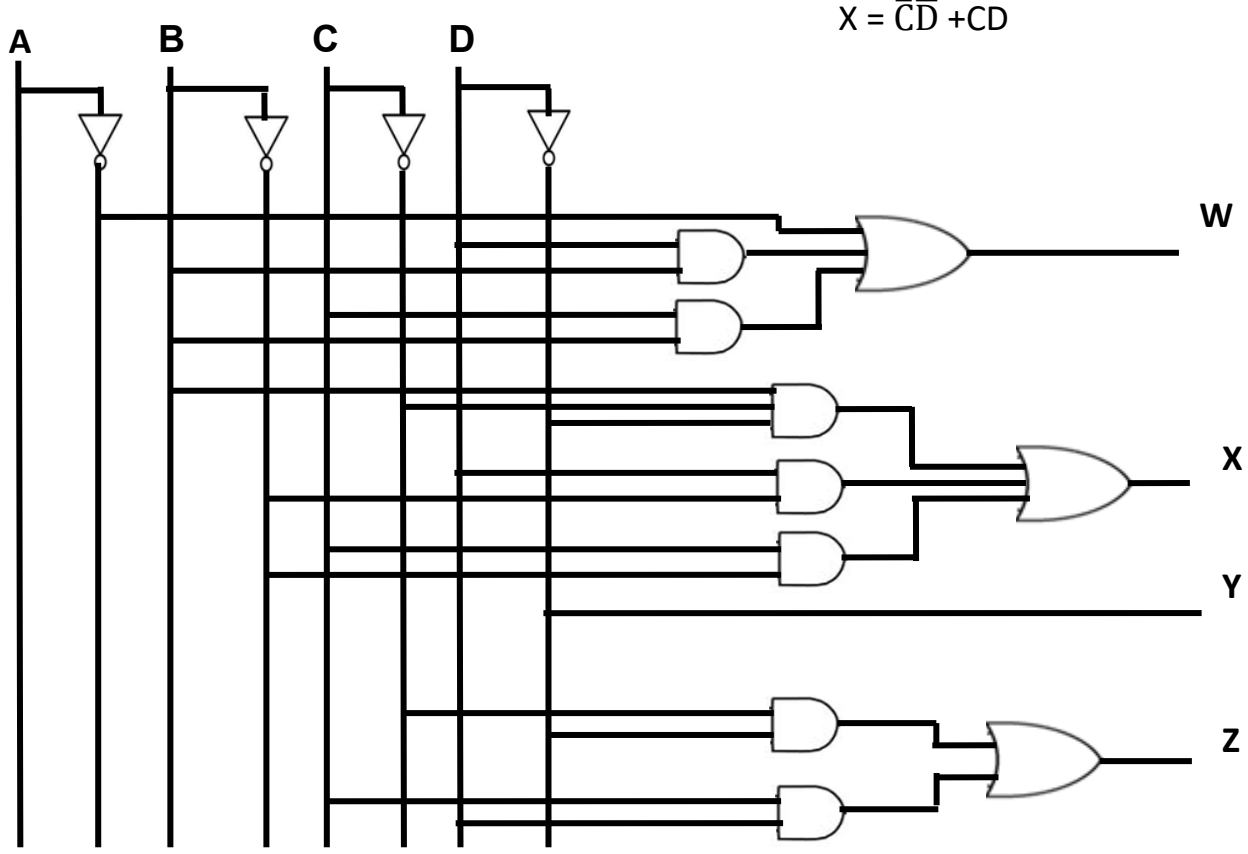


Fig5

HW5: design a logic circuit that converts (*BCD*) to (*7 – segment*)

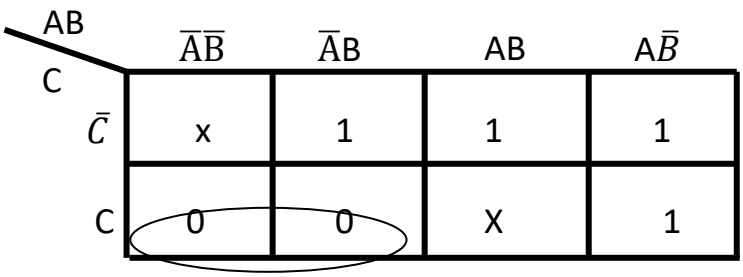
HW6: design a logic circuit that compares between two words each one of them consists two bits in *Excess – three* codes.

HW7: design a logic circuit that gives an even and odd parity for *Excess – three* codes inputs.

Ex16/ find the logic circuit of the following { $F=\Sigma(2,4,5,6)$}, and {don't care = $\prod(0,7)$ } using *K- Map PoS* minimization.

Sol: the truth table is and *K – Map* are shown below

A	B	C	Y
0	0	0	X
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	X



$\bar{F} = \bar{A}C \Longrightarrow F = \bar{\bar{F}}$

$\Longrightarrow F = \overline{\bar{A}C}$

$\Longrightarrow F = \bar{\bar{A}} + \bar{C} \Longrightarrow F = A + \bar{C}$

The logic circuit is that implement function (*F*) can be drawn as figure (6).

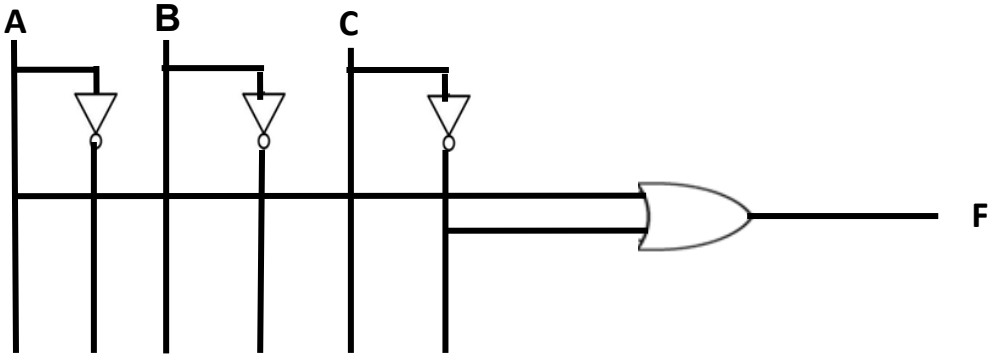
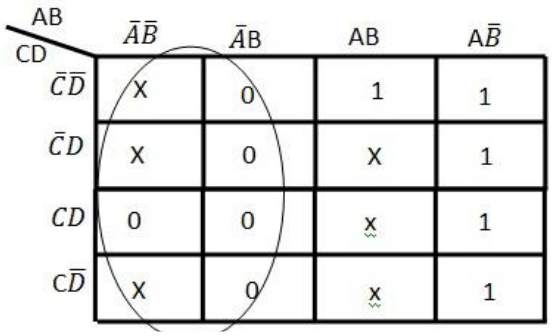


Fig6

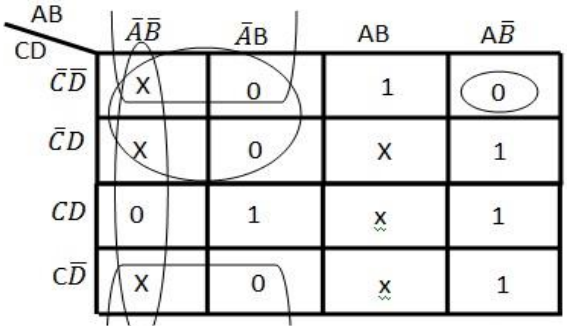
Ex17/ design a logic circuit that converts (*Excess – three*) code to (*2421*) code, using *K – Map* with *PoS* minimization.

Sol: the truth table and *K – Map* are shown below

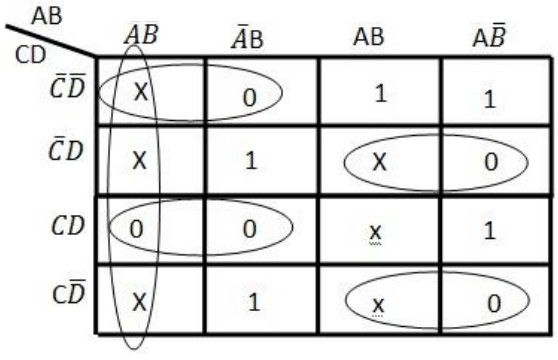
A	B	C	D	W	X	Y	Z
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	1	1	0	1
1	0	1	1	1	1	1	0
1	1	0	0	1	1	1	1



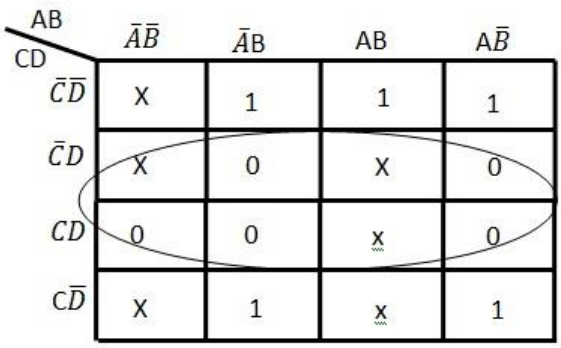
$\bar{W} = \bar{A}$



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



$\bar{Y}= \bar{A}\bar{B}+\bar{A}\bar{C}\bar{D}+\bar{A}CD+A\bar{C}D+AC\bar{D}$



$\bar{Z}=D$

The logic circuit can be drawn as in the (*SoP*) case except that not gate is putting on the final logic gate to obtain the inverting of ($\bar{A}\ \bar{B}\ \bar{C}\ \bar{D}$).

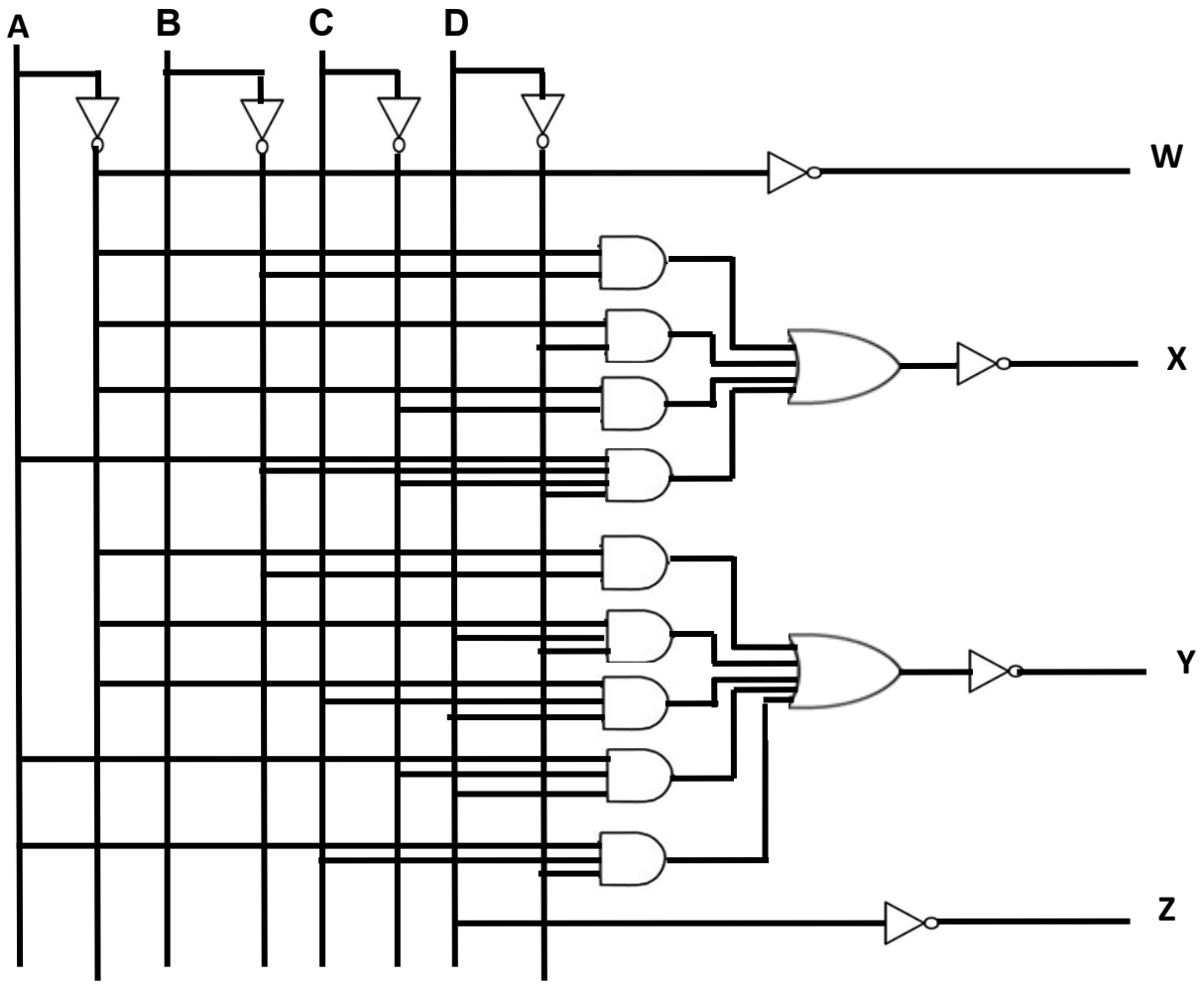


Fig7

HW6: design a logic circuit that finds the primary numbers from the following {*0* ⟶ *15*} using (*K- Map*) with *PoS* minimization.

HW7: design a logic circuit that divides numbers from (*0*) to (*15*) by two (*2*), gives (*ones*) outputs when the division operation has no carry and

(zeros) outputs when the division operation has carry using (*K- Map*) with *PoS* minimization.

HW8: design a logic circuit that compares two words each one of them has two bits using (*K- Map*) with *PoS* minimization.