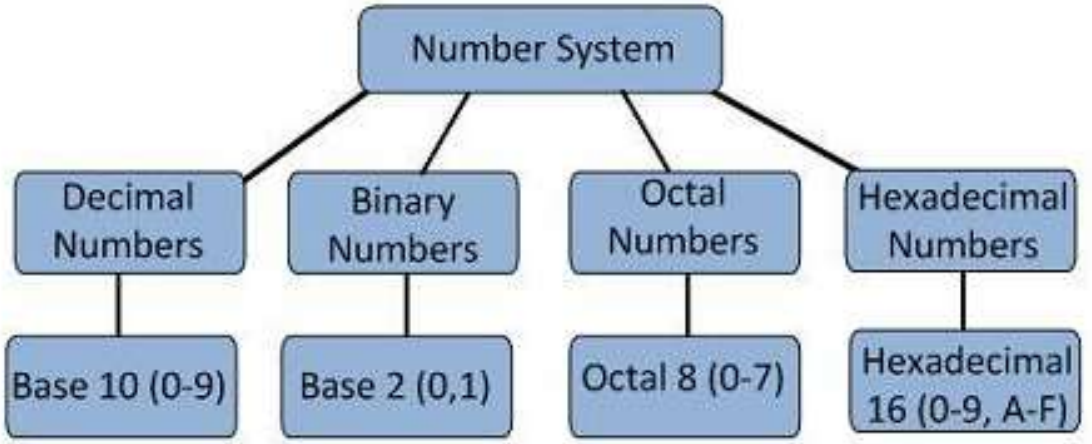
Number Systems

Unit I Session I and II

Gauri Rao

Number systems

- Represent information



- Base or radix

Numbering Systems		
System	Base	Digits
Binary	2	0 1
Octal	8	0 1 2 3 4 5 6 7
Decimal	10	0 1 2 3 4 5 6 7 8 9
Hexadecimal	16	0 1 2 3 4 5 6 7 8 9 A B C D E F

Decimal number system

- A positional number system
- Has 10 symbols or digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). Hence, its base = 10
- The maximum value of a single digit is 9 (one less than the value of the base)
- Each position of a digit represents a specific power of the base (10)
- We use this number system in our day-to-day life

$\times 10^0$)

- Example

$$1234_{10} = 1\text{x}\ 10^3 + 2\text{x}\ 10^2 + 3\text{x}10^1 + 4\text{x}10^0$$
$$= 1000 + 200 + 30 + 4$$
$$= 1234_{10}$$

Binary Number System

- A positional number system
- Has only 2 symbols or digits (0 and 1). Hence its base = 2
- The maximum value of a single digit is 1 (one less than the value of the base)
- Each position of a digit represents a specific power of the base (2)
- This number system is used in computers

Example

$$10101_2 = (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) \times (1 \times 2^0)$$

$$= 16 + 0 + 4 + 0 + 1$$

$$= 21_{10}$$

Binary to Decimal

$(10110)_2$

$$1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

$$= 16 + 0 + 4 + 2 + 0$$

$$= 22_{10}$$

Octal number system

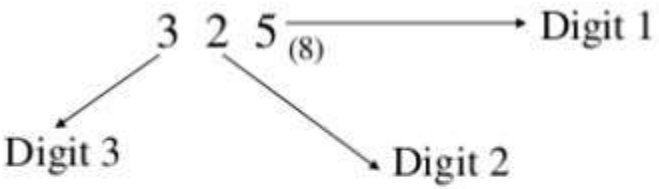
- A positional number system
- Has total 8 symbols or digits (0, 1, 2, 3, 4, 5, 6, 7). Hence, its base = 8
- The maximum value of a single digit is 7 (one less than the value of the base
- Each position of a digit represents a specific power of the base (8)

$$2057_8 = (2 \times 8^3) + (0 \times 8^2) + (5 \times 8^1) + (7 \times 8^0)$$

$$= 1024 + 0 + 40 + 7$$

$$= 1071_{10}$$

Octal to decimal



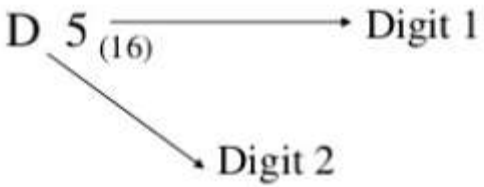
$$3 \times 8^2 + 2 \times 8^1 + 5 \times 8^0 = 3 \times 64 + 2 \times 8 + 5 \times 1$$
$$= 192 + 16 + 5$$
$$= 213$$

Hexadecimal Number System

- A positional number system
- Has total 16 symbols or digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). Hence its base = 16
- The symbols A, B, C, D, E and F represent the decimal values 10, 11, 12, 13, 14 and 15 respectively
- The maximum value of a single digit is 15 (one less than the value of the base)

$$
\begin{aligned}
1AF_{16} &= (1 \times 16^2) + (A \times 16^1) + (F \times 16^0) \\
&= 1 \times 256 + 10 \times 16 + 15 \times 1 \\
&= 256 + 160 + 15 \\
&= 431_{10}
\end{aligned}
$$

Hex to Decimal



$$D \times 16^1 + 5 \times 16^0 = 13 \times 16 + 5 \times 1$$
$$= 208 + 5$$
$$= 213$$

Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

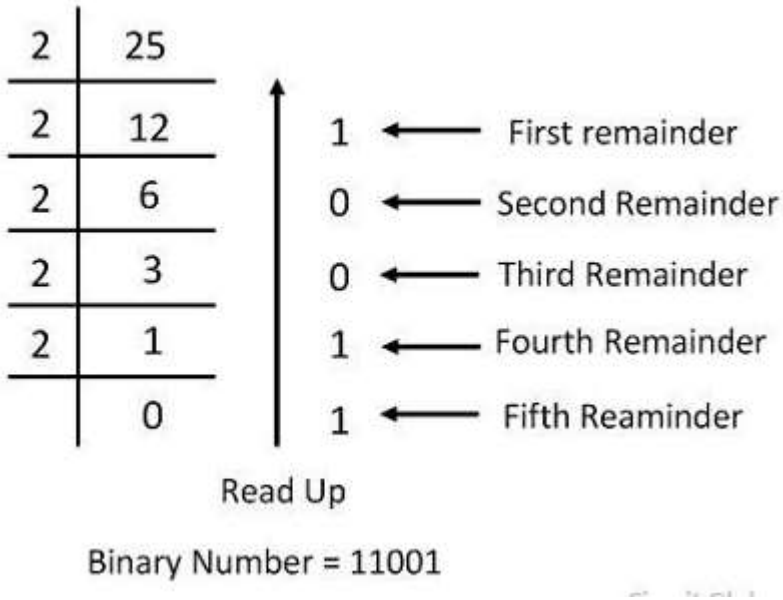
Decimal to Binary conversion

- Make a list of the binary place values up to the number being converted.
- Perform successive divisions by 2, placing the remainder of 0 or 1 in each of the positions from right to left.
- Continue until the quotient is zero.
- Example: 42_{10}

2^5	2^4	2^3	2^2	2^1	2^0
32	16	8	4	2	1
1	0	1	0	1	0

Decimal to Binary conversion

Convert decimal number 25 into its binary equivalent



Fractional part Example (0.35)

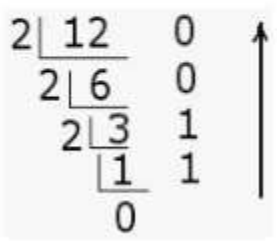
0.35 X 2	= 0.70	with a carry of	0
0.70 X 2	= 1.40	with a carry of	1
0.40 X 2	= 0.80	with a carry of	0
0.80 X 2	= 1.60	with a carry of	1
0.60 X 2	= 1.20	with a carry of	1

Thus the fractional binary number is .01011

Example

- Convert the decimal number $(12.0625)_{10}$ into binary number.

Solution:



Fractional part:

0.0625 x 2 = 0.1250 0
0.1250 x 2 = 0.2500 0
0.2500 x 2 = 0.500 0
0.500 x 2 = 1.000 1

$$(12.0625)_{10} = (1100.0001)_2$$

Decimal to Octal conversion

- Convert from **decimal to octal** by using the repeated division method used for decimal to binary conversion.
- Divide the decimal number by 8
- The first remainder is the LSB and the last is the MSB.

Example : convert 359_{10} to Decimal Value

Solve = $359_{10} = ?_8$

$$= \frac{359}{8} \longrightarrow 44 \text{ balance } 7 \longrightarrow \text{LSB}$$

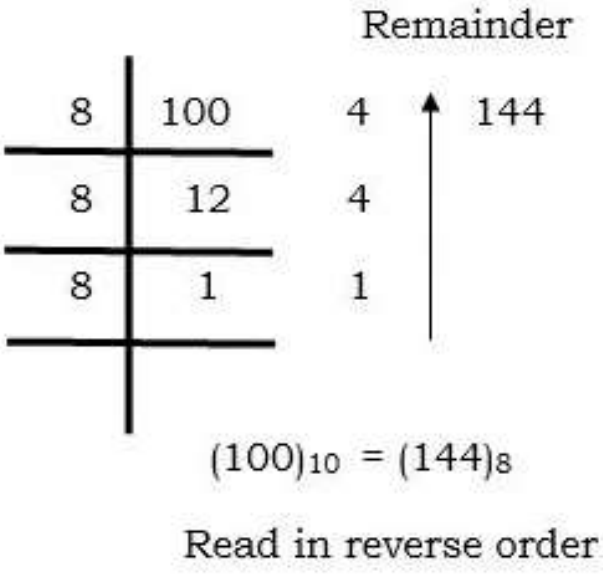
$$= \frac{44}{8} \longrightarrow 5 \text{ balance } 4$$

$$= \frac{5}{8} \longrightarrow 0 \text{ balance } 5 \longrightarrow \text{MSB}$$

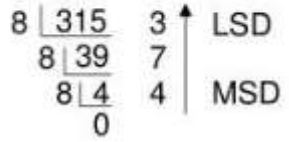
$\therefore$ Answer = 547_8

Decimal to Octal

Convert decimal number 100 into its octal equivalent.



$$(315)_{10} = (473)_8$$



Fractional part

- Convert the decimal number $(225.225)_{10}$ into octal number.

Solution:

8 | 225	
8 | 28	1
8 | 3	4
0	3

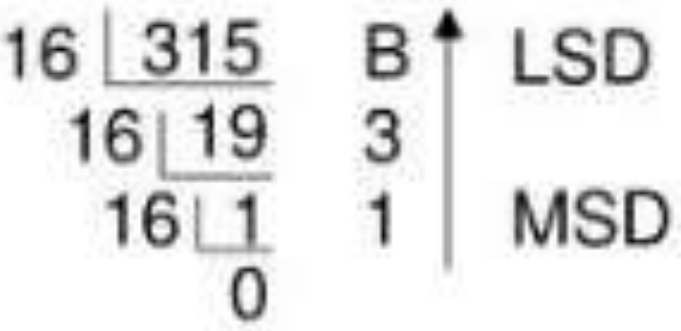
Fractional part:

0.225 x 8 = 1.800	1
0.800 x 8 = 6.400	6
0.400 x 8 = 3.200	3
0.200 x 8 = 1.600	1
0.600 x 8 = 4.800	4

$$(225.225)_{10} = (341.16314)_8$$

Decimal to Hexadecimal Conversion

- Convert decimal number 315 into its Hex equivalent.



$$(315)_{10} = (13B)_{16}$$

Binary to Hex

1. Group the digits of the binary number by **four** starting from the right.
2. Replace each group of **four** digits by an equivalent hexadecimal digit.

Convert 10110101_2 into a hexadecimal number.

$$\underbrace{1011}_{B}\ \underbrace{0101}_{5}{}_2 = B5_{16}$$

Hex to Binary

- To convert a hexadecimal to binary number, convert each hexadecimal digit to its 4 bit equivalent using the hexa number.

- Example: $(23.AB)_{16} = ()_2$

Solution: $(23.AB)_{16}$ = 2 3 . A B

0010 0011 1010 1011

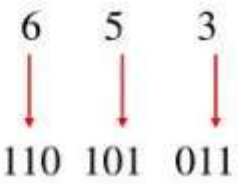
$(23.AB)_{16} = (00100011.10101011)_2$

Octal to Binary

Each octal number converts to 3 binary digits

Code
0 - 000
1 - 001
2 - 010
3 - 011
4 - 100
5 - 101
6 - 110
7 - 111

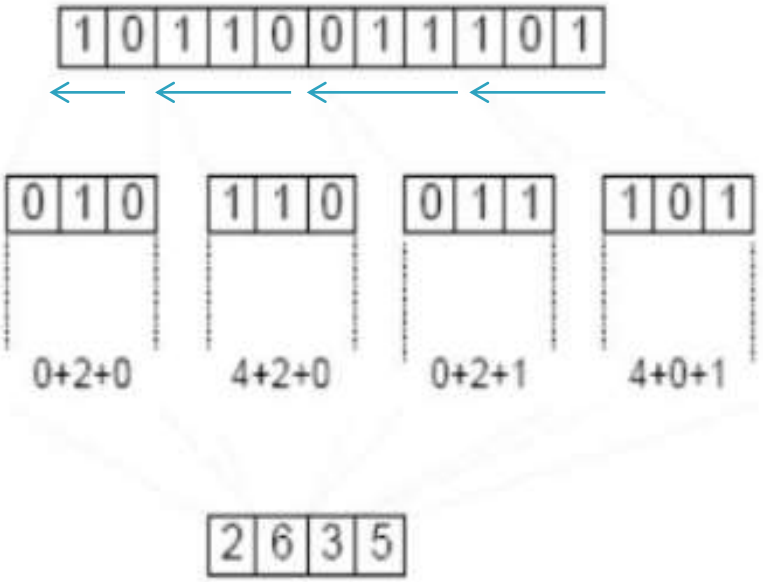
To convert 653_8 to binary, just substitute code:



Binary to Octal

- Can be converted by grouping the binary bit in group of three starting from LSB
- Octal is a base-8 system and equal to two the **power of three,** so a digit in Octal is equal to three digit in binary system.

Code
0 - 000
1 - 001
2 - 010
3 - 011
4 - 100
5 - 101
6 - 110
7 - 111



Hex to Octal

i. Hexadecimal – Decimal – Octal

ii. Hexadecimal – Binary – Octal

Convert Hexadecimal number E64B into its octal equivalent

E	6	4	B	Hex
1110	0110	0100	1011	Binary

1110 011001001011 Binary

001	110	011	001	001	011	Binary
1	6	3	1	1	3	Octal

Practice problems

- Practice conversions:

Binary	Decimal	Octal	Hex
01111101			
1110101			
1101010111			

- Practice conversions:

Decimal	Binary	Octal	Hex
72			
92			
185			

- Convert 11011111_2 into a hexadecimal number.

Check your answers

- Practice conversions:

Binary	Decimal	Octal	Hex
01111101	$125_{(10)}$	$175_{(8)}$	$7D_{(16)}$
1110101	$117_{(10)}$	$165_{(8)}$	$75_{(16)}$
1101010111	$855_{(10)}$	$1527_{(8)}$	$357_{(16)}$

- Practice conversions:

Decimal	Binary	Octal	Hex
72	$1001000_{(2)}$	$110_{(8)}$	$48_{(16)}$
92	$1011100_{(2)}$	$134_{(8)}$	$5C_{(16)}$
185	$10111001_{(2)}$	$271_{(8)}$	$B9_{(16)}$

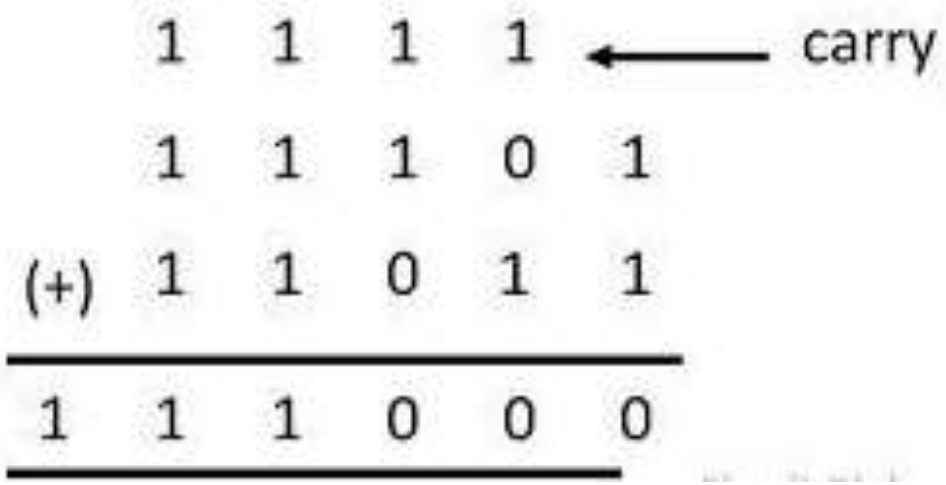
- Convert 11011111_2 into a hexadecimal number.

$$11011111_{(2)} = DF_{(16)}$$

Rules for Binary addition
1+1+1= 1 carry 1

Input A	Input B	Sum (S) A+B	Carry (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Example: Add 11101+11011



1 1 1 1 (Carry)

1 1 0 1 1 (27)

(+) 1 0 1 0 1 (21)

---

1 1 0 0 0 0 (48)

- 0+0=0
- 0+1= 1
- 1+0= 1
- 1+1 = 0 carry =1
- 1+1+1= 1 with carry =1

Example

```
   1111      carry
   10111
+ 11001
 110000
```

9+5=14 1001

+0101

1 110

Rules for Binary subtraction

Input A	Input B	Subtract (S) A-B	Borrow (B)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

0–0=0

1–0= 1

1–1= 0

0–1 = Diff 1 borrow is 1

Example : Subtract 1100–1010

```
  1 1 0 0
- 1 0 1 0
    1          Borrow
  0 0 1 0   ←
```

Solve 11011– 10110

- 38-29=09= 1001

100110

-011101

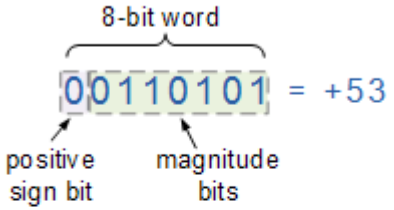
11 1 Borrow

001001

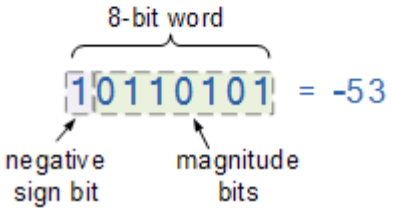
0-1= 1 with borrow 1

Sign Magnitude representation

Positive Signed Binary Numbers



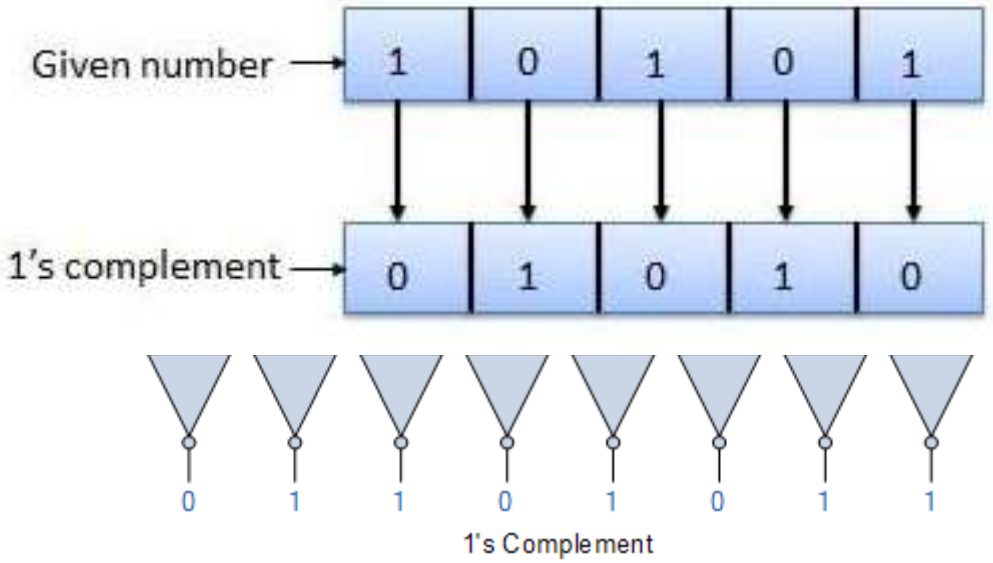
Negative Signed Binary Numbers



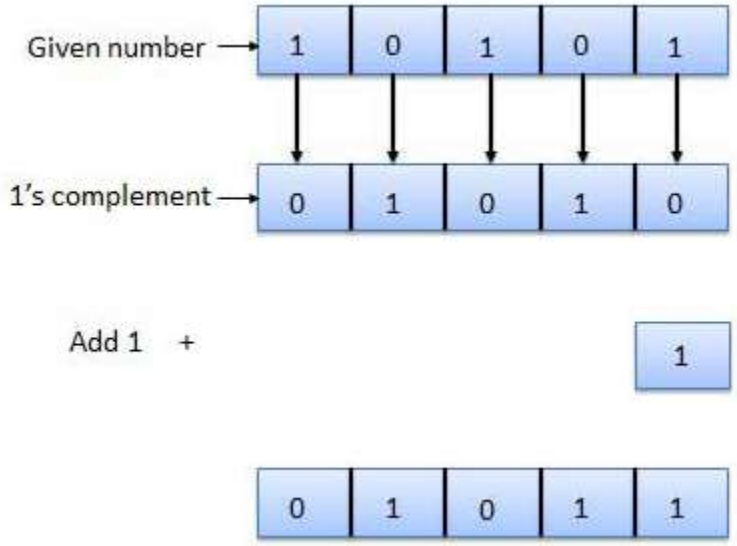
Unsigned no. 8 bits= 256= 0 to 255

Signed no. 8 bits =1 bit sign and 7 bits for magnitude = -128 to -1, 0 to +127= 256

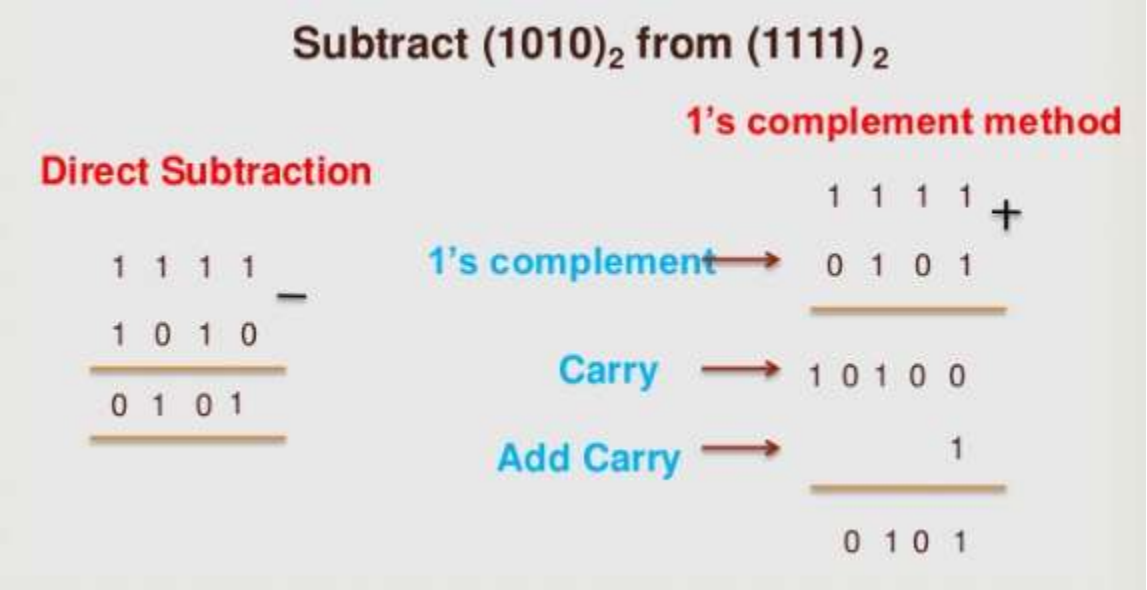
1’ complement



2's complement



Subtraction using 1's complement



RULES

A-B

1'S COMP OF B

ADD A WITH 1'S COMP OF B

IF CARRY THEN ADD IT TO RESULT

IF NO CARRY THEN TAKE 1'S COMPLEMENT OF RESULT AND APPLY - SIGN

Subtract 9–4 using 1's complement

9 = 1001 4 = 0100

1's complement of 4 is 1011

```
  1001
+ 1011
-------
1 0100
+     1
-------
  0101
```

Add carry to the answer

Final answer is 5

Subtract 4 – 9 using 1's complement

4 = 0100 and 9 = 1001

1's complement of 9 is 0110

0100

\+ 0110

0 1010

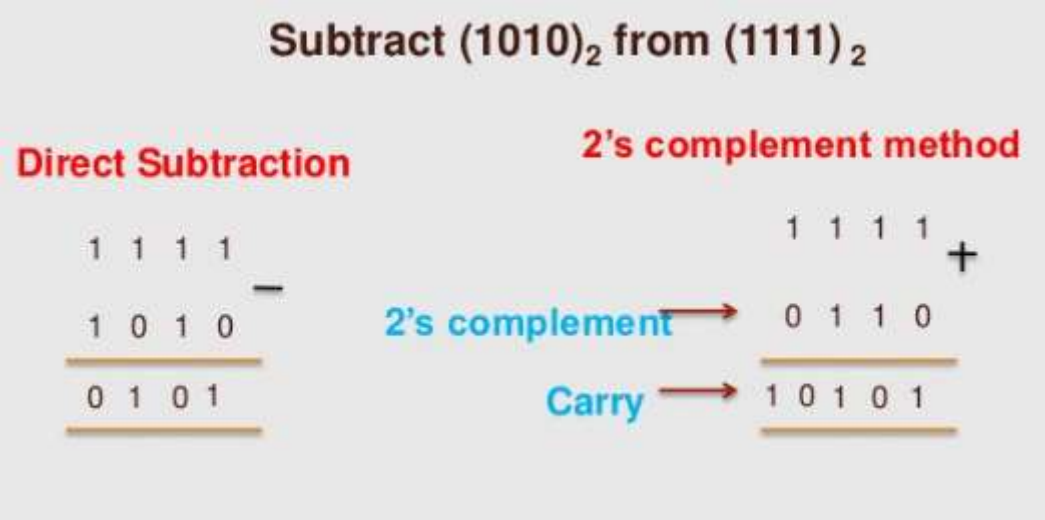
Carry is 0 so take 1's complement of answer

1010 Invert all bits

0101 = 5

apply – sign so final answer is -5

Subtraction using 2's complement



1010=10

0101= 1's complement of 10

\+ 1

0110 = 2's complement of 10

Subtract 9– 5 using 2's complement

9 = 1001 5= 0101

1's complement of 5 is 1010

2's complement of 5 is 1011

```
  1001
+ 1011
-------
1 0100      Discard carry
  0100      Final answer is 4
```

Subtract 5-9 using 2's complement

5 = 0101 and 9 = 1001

1's complement of 9 is 0110

2's complement of 9 is 0111

```
  0101
+ 0111
-------
0 1100        No carry
```

Take 2's complement of answer and apply – sign

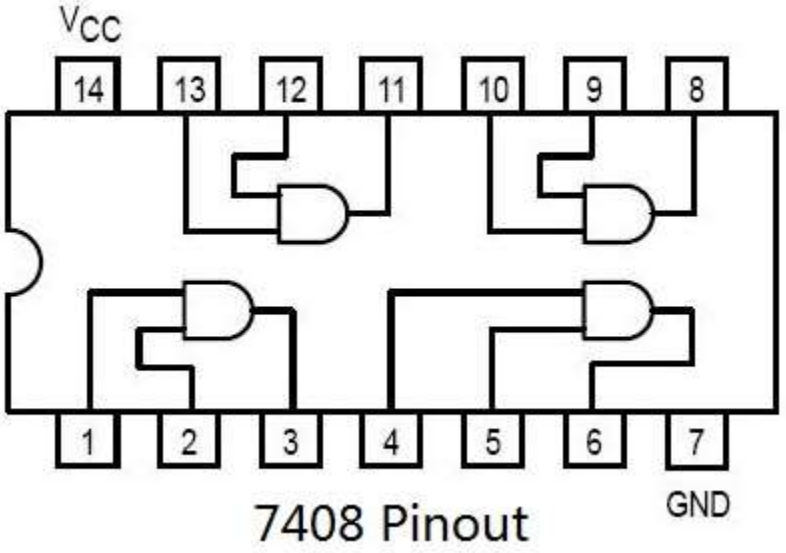
2's complement of 1100 is 0100 = -4

Decimal	Signed binary	1's complement	2's complement
+7	0 111	0 111	0 111
+6	0 110	0110	0110
+5	0 101	0101	0101
+4	0 100	0100	0100
+3	0 011	0011	0011
+2	0 010	0010	0010
+1	0 001	0001	0001
+0	0000	0000	0000
-0	0000	1111	0000
-1	1 001	1110	1111
-2	1 010	1101	1110
-3	1 011	1100	1101
-4	1 100	1011	1100
-5	1 101	1010	1011
-6	1 110	1001	1010
-7	1 111	1000	1001

GATE IC

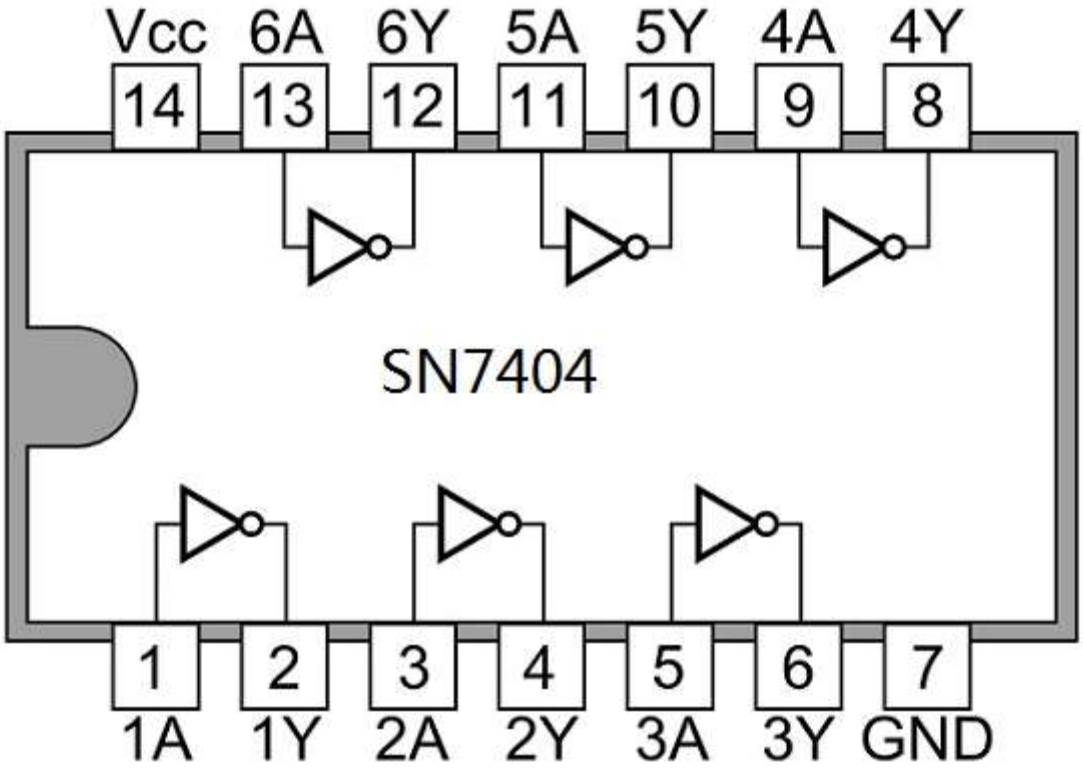
Gauri Rao

AND GATE- IC 7408

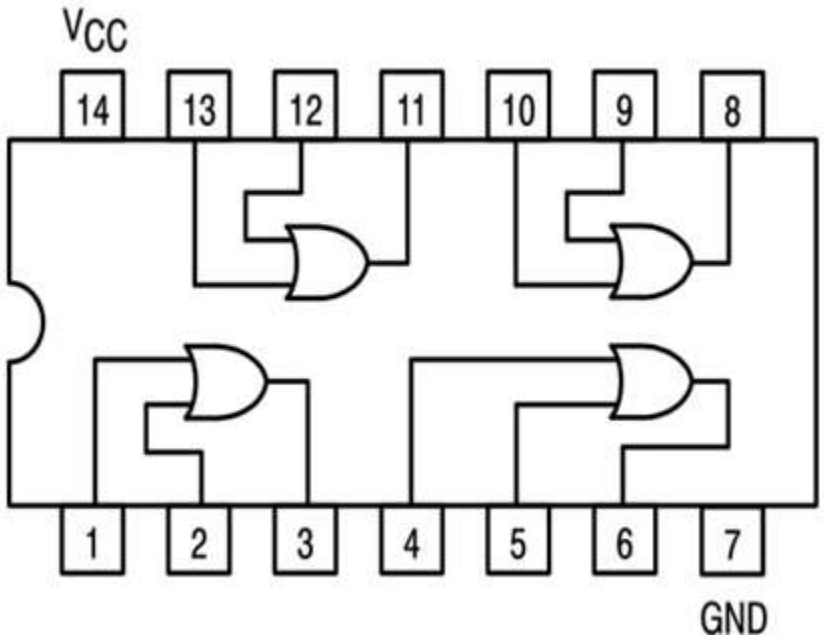


7408 Pinout

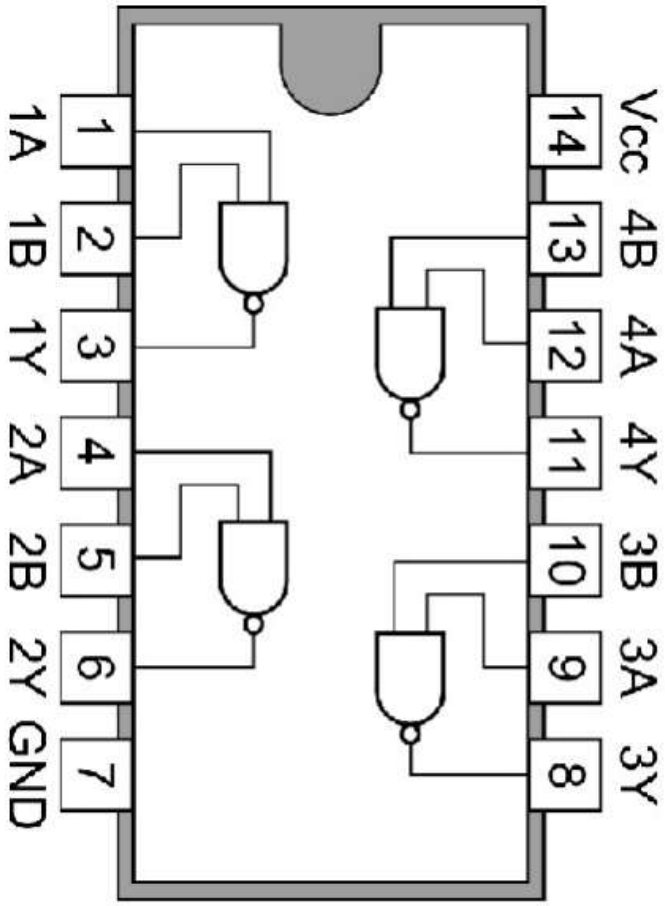
NOT GATE


Vcc
6A
6Y
5A
5Y
4A
4Y
14
13
12
11
10
9
8
SN7404
1
2
3
4
5
6
7
1A
1Y
2A
2Y
3A
3Y
GND

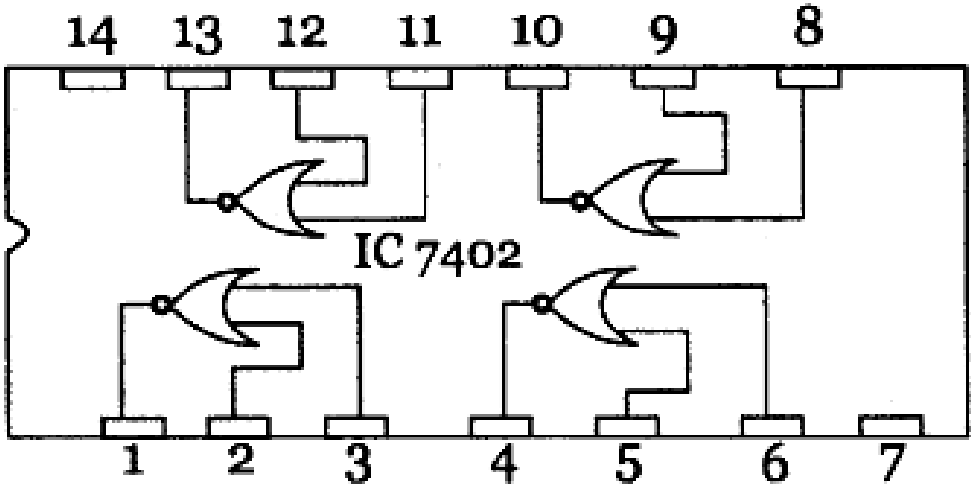
OR GATE 7432



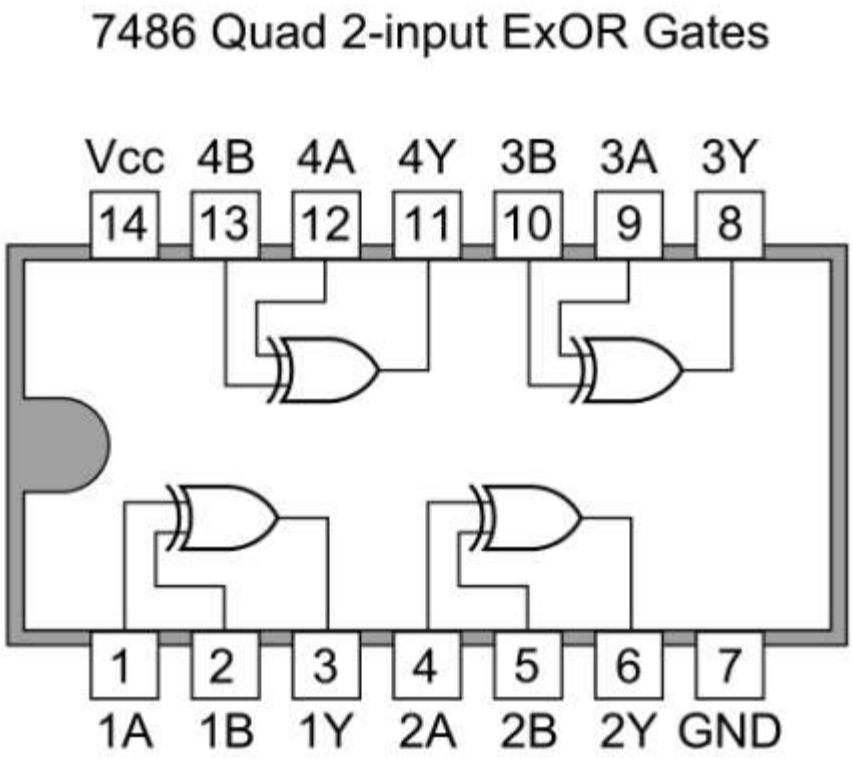
NAND GATE 7400



NOR



EXOR- 7486


7486 Quad 2-input ExOR Gates
Vcc 4B 4A 4Y 3B 3A 3Y
14 13 12 11 10 9 8
1 2 3 4 5 6 7
1A 1B 1Y 2A 2B 2Y GND

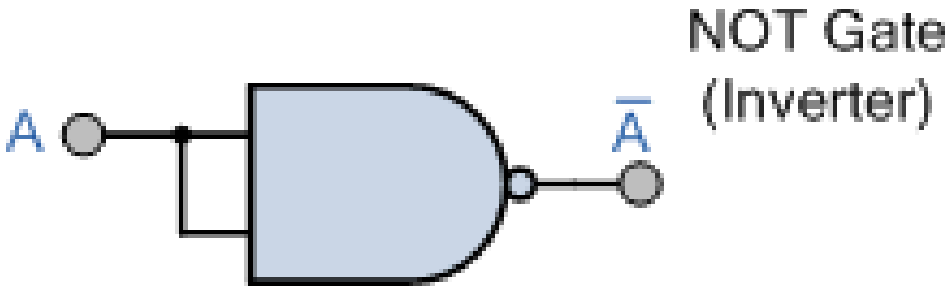
Universal Gates

Gauri Rao

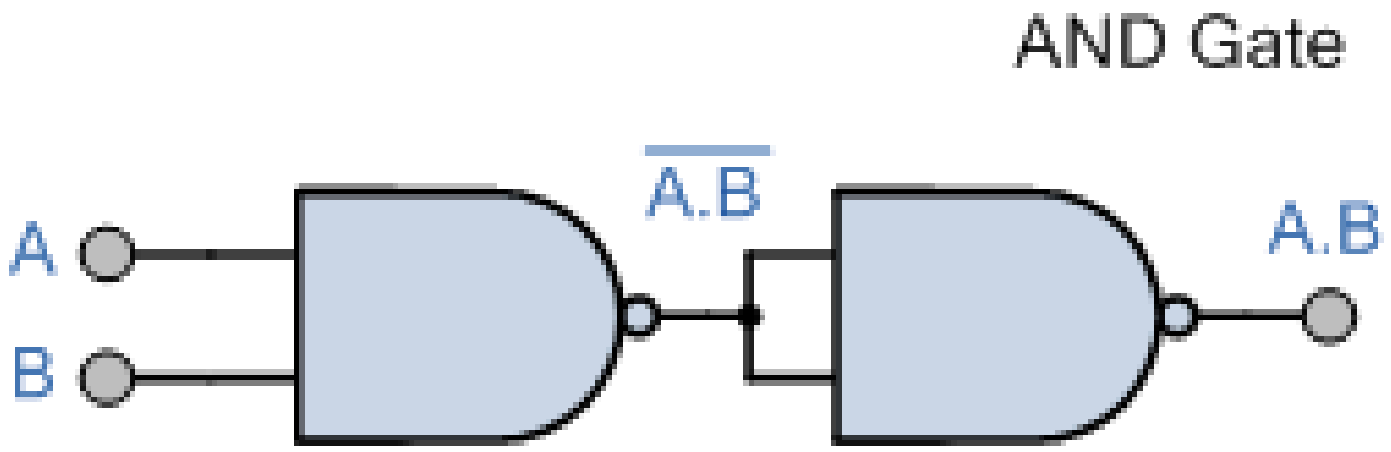
Universal gates

1. NAND
2. NOR

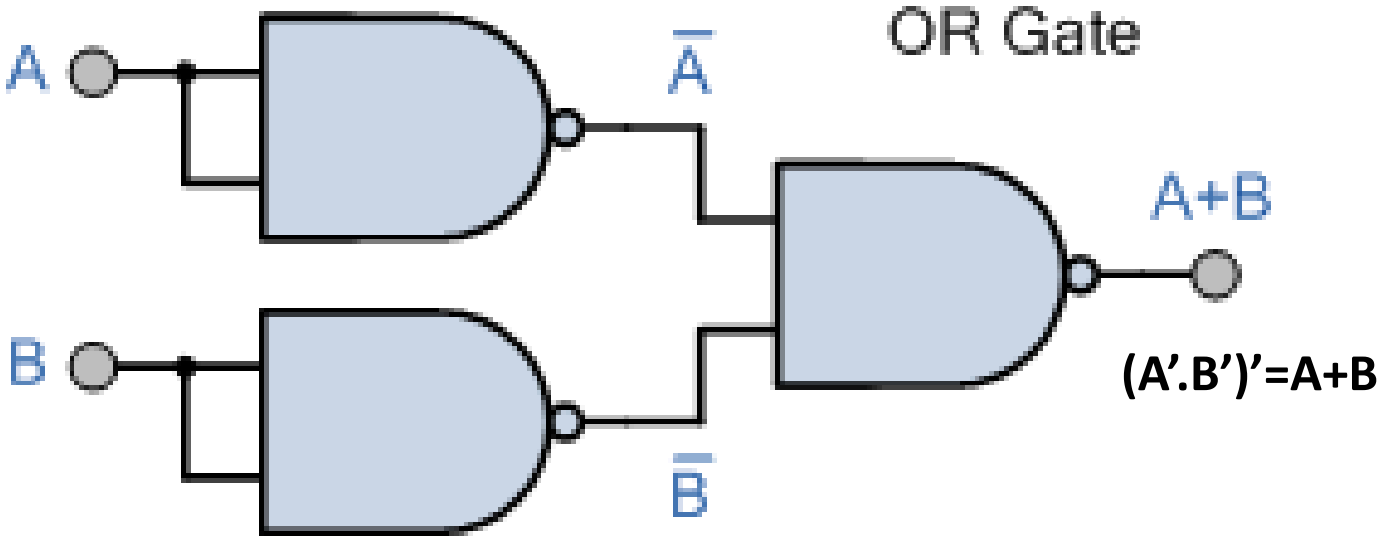
NAND as NOT



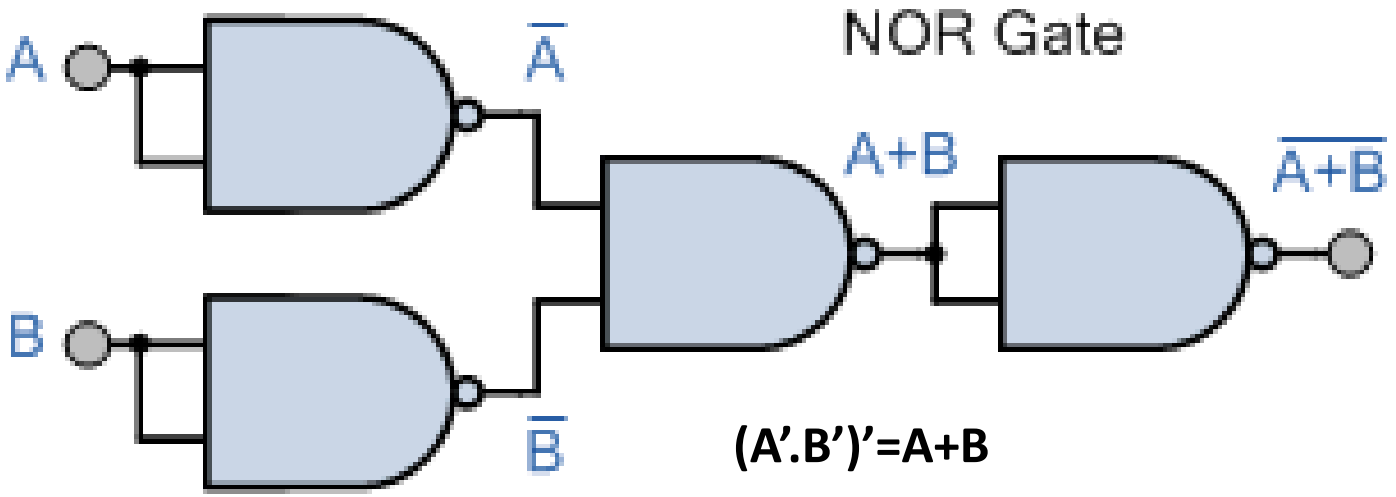
NAND as AND



NAND as OR

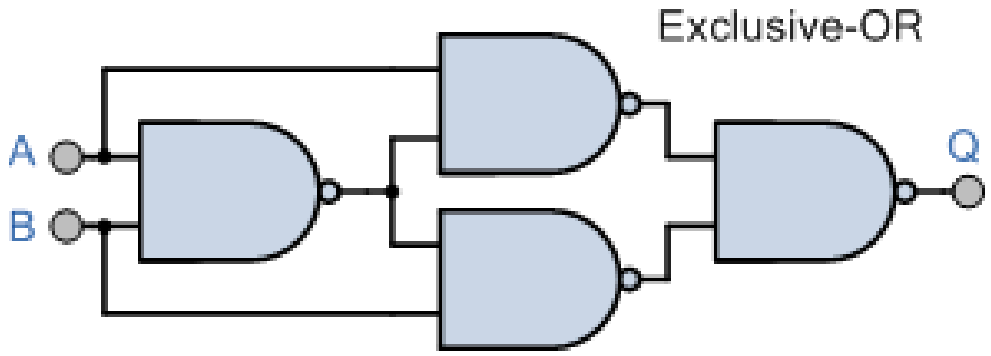


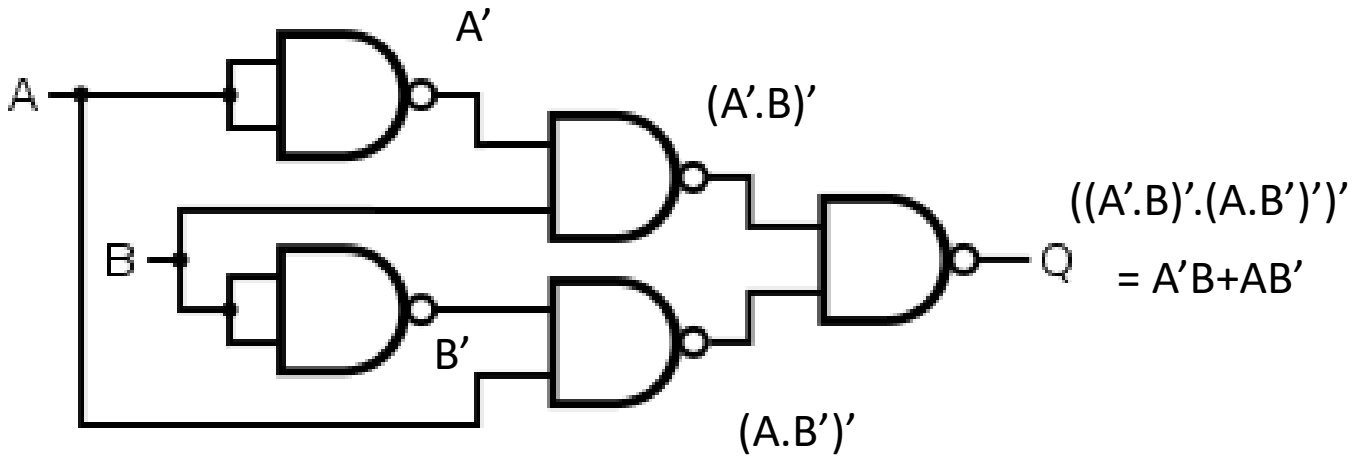
NAND as NOR



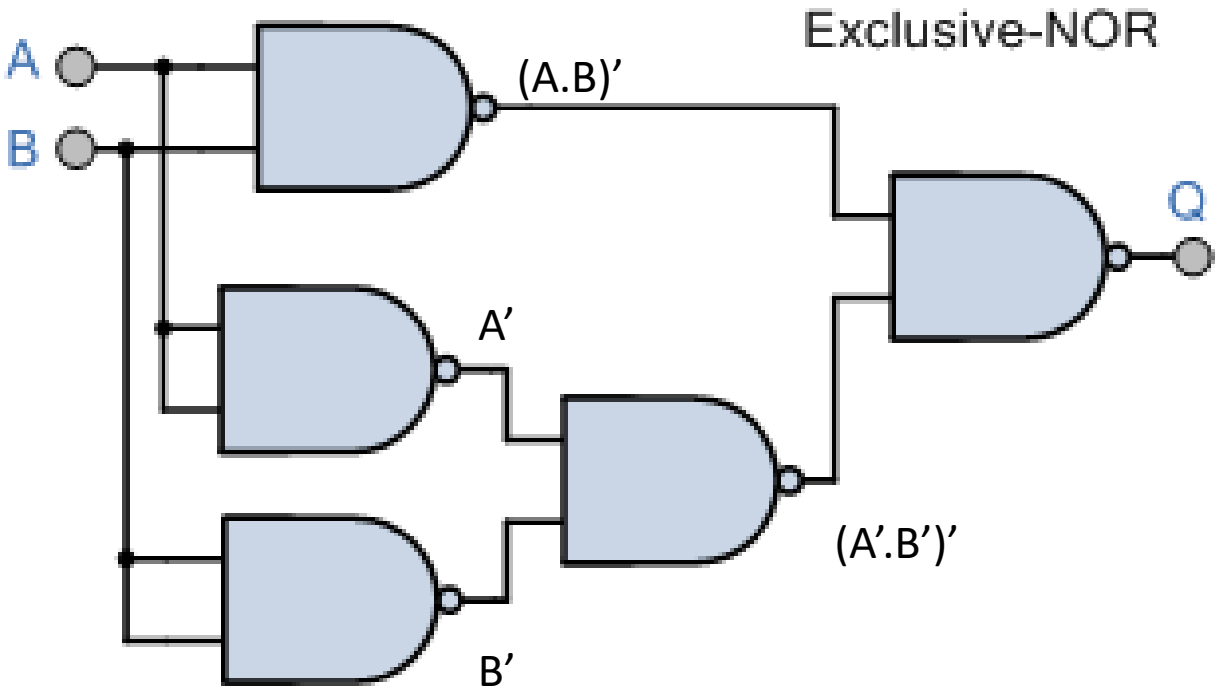
(A'.B')'=A+B

NAND AS EXOR





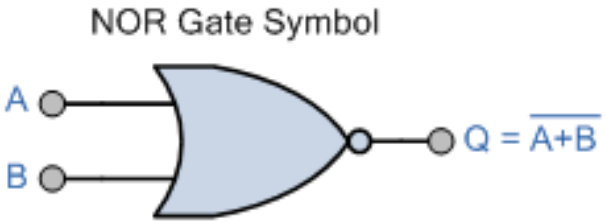
NAND AS XNOR

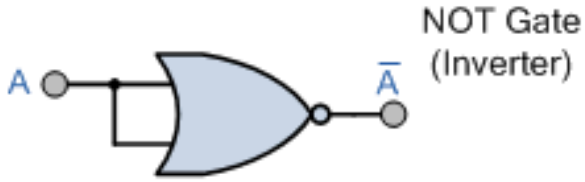


((A.B)'.(A'B')')'

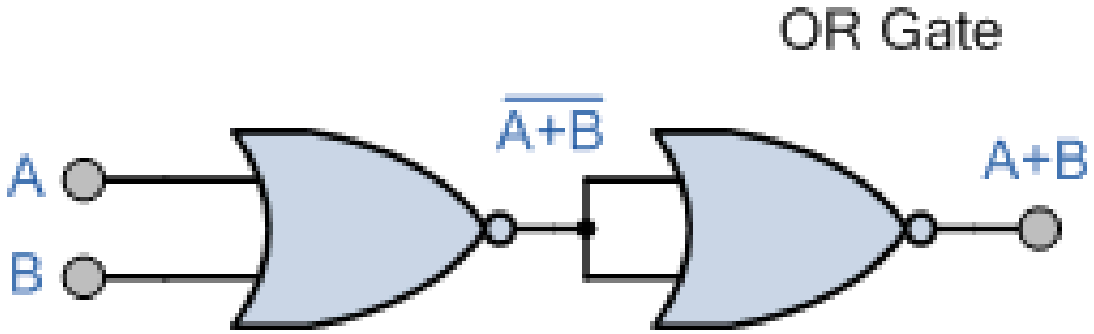
= AB+A'B'

NOR as NOT

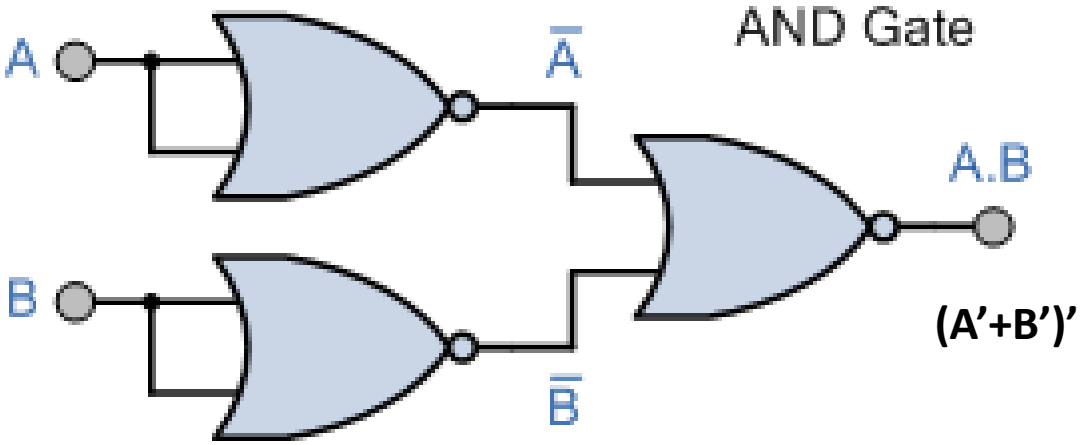

NOR Gate Symbol
A
B
Q = $\overline{A+B}$


NOT Gate
(Inverter)
A
$\overline{A}$

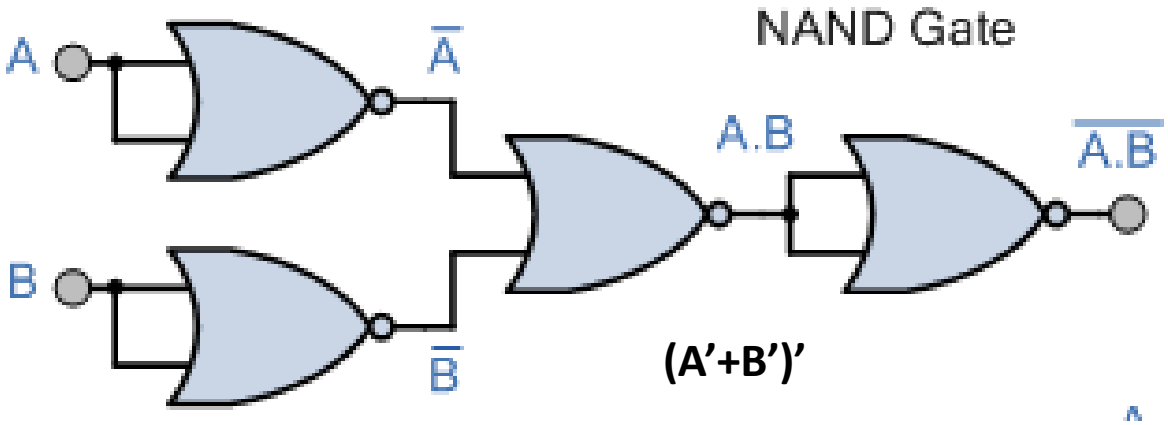
NOR as OR



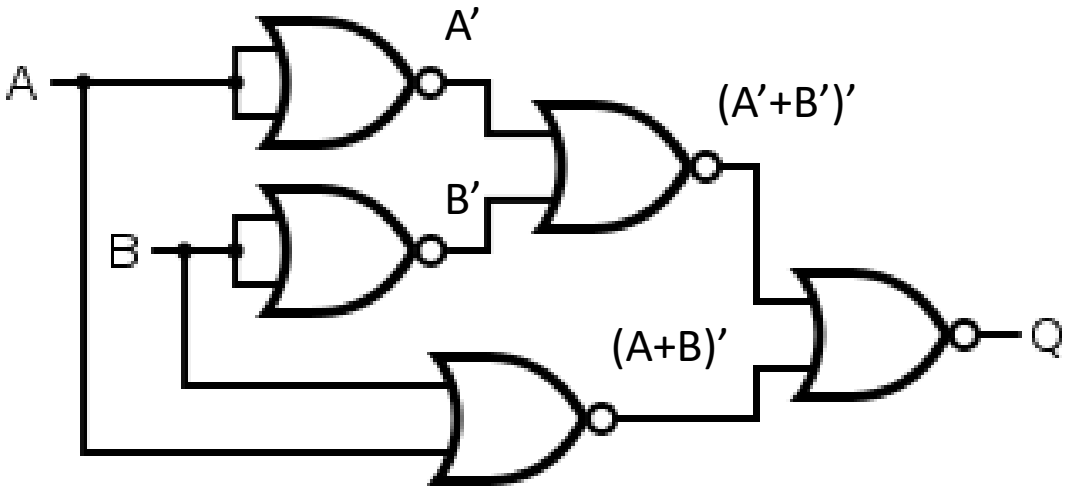
NOR as AND



NOR as NAND

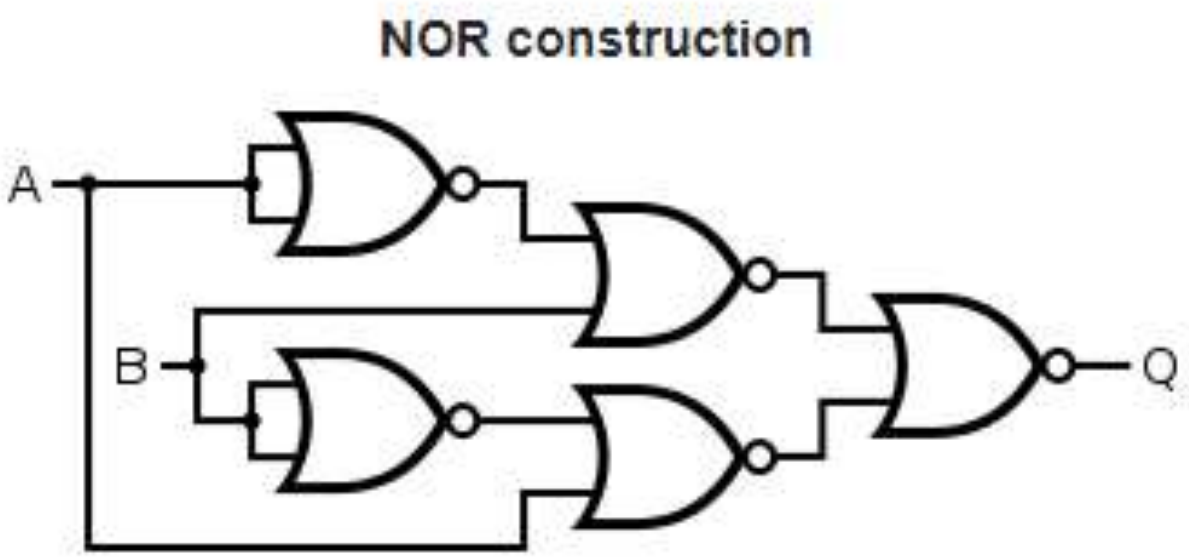


NOR AS EXOR



= [(**A** NOR **A**) NOR (**B** NOR **B**)] NOR
(**A** NOR **B**)

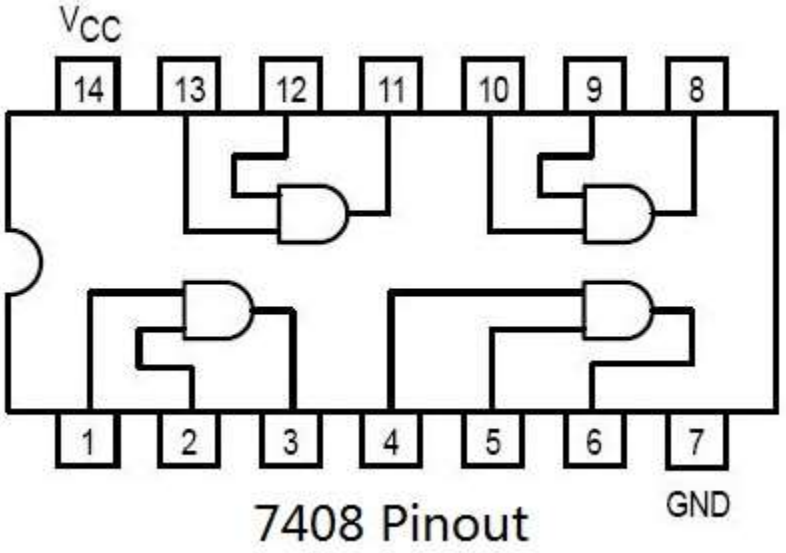
NOR AS EXNOR



GATE IC

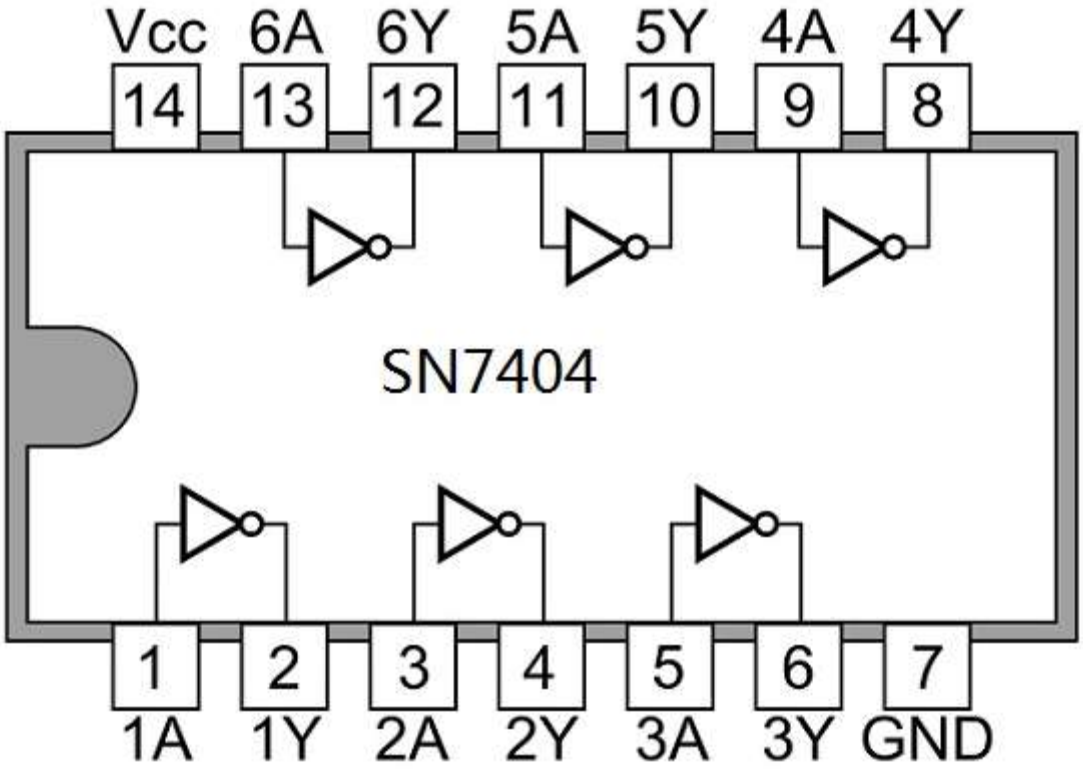
Gauri Rao

AND GATE- IC 7408

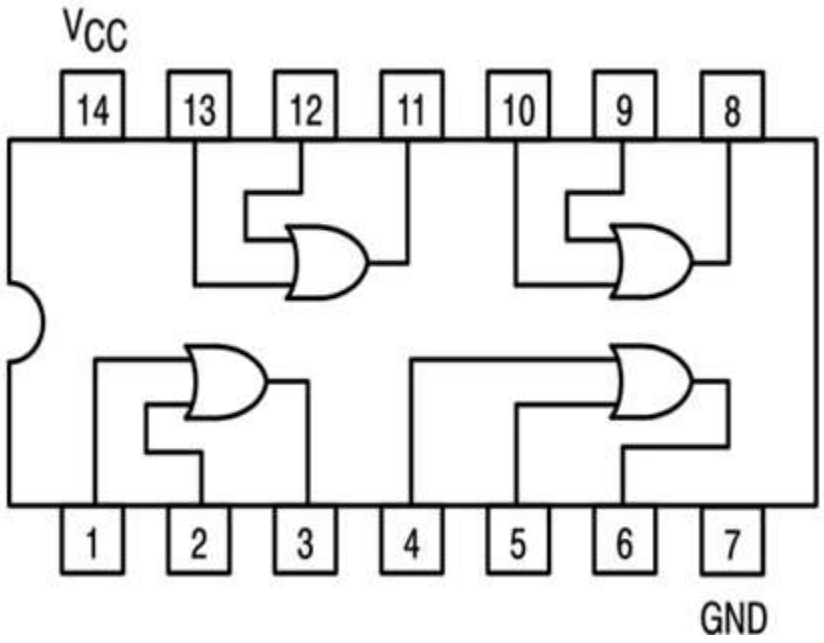


7408 Pinout

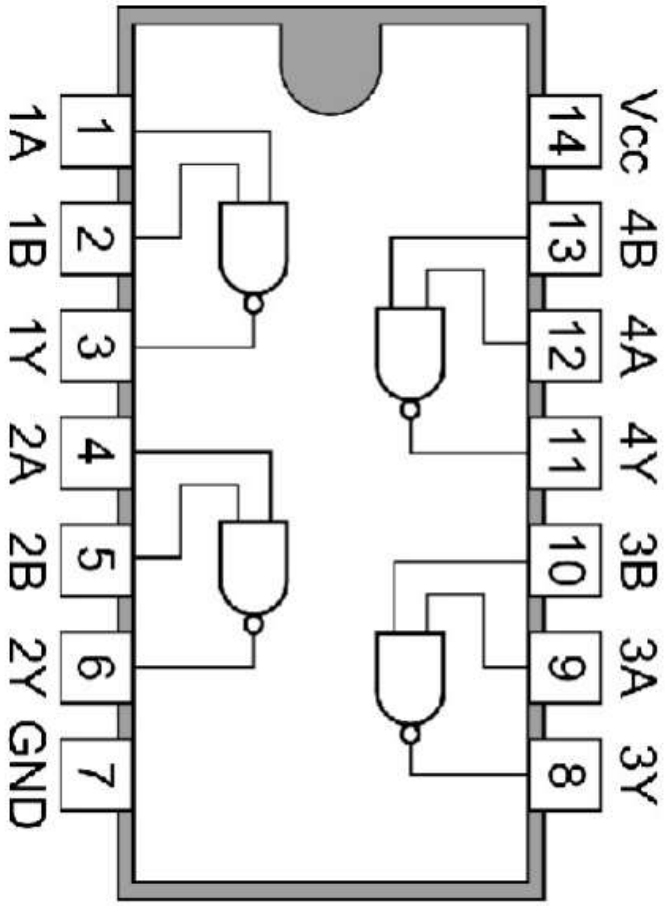
NOT GATE


Vcc
6A
6Y
5A
5Y
4A
4Y
14
13
12
11
10
9
8
SN7404
1
2
3
4
5
6
7
1A
1Y
2A
2Y
3A
3Y
GND

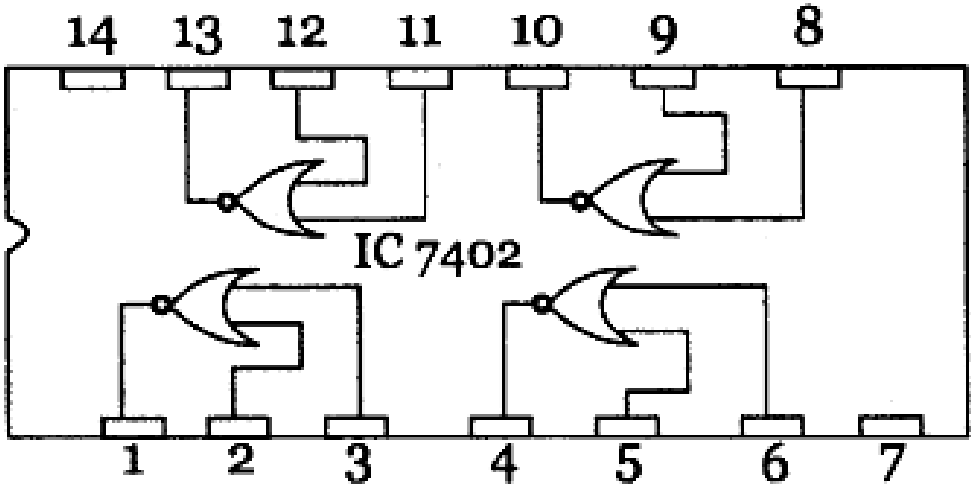
OR GATE 7432



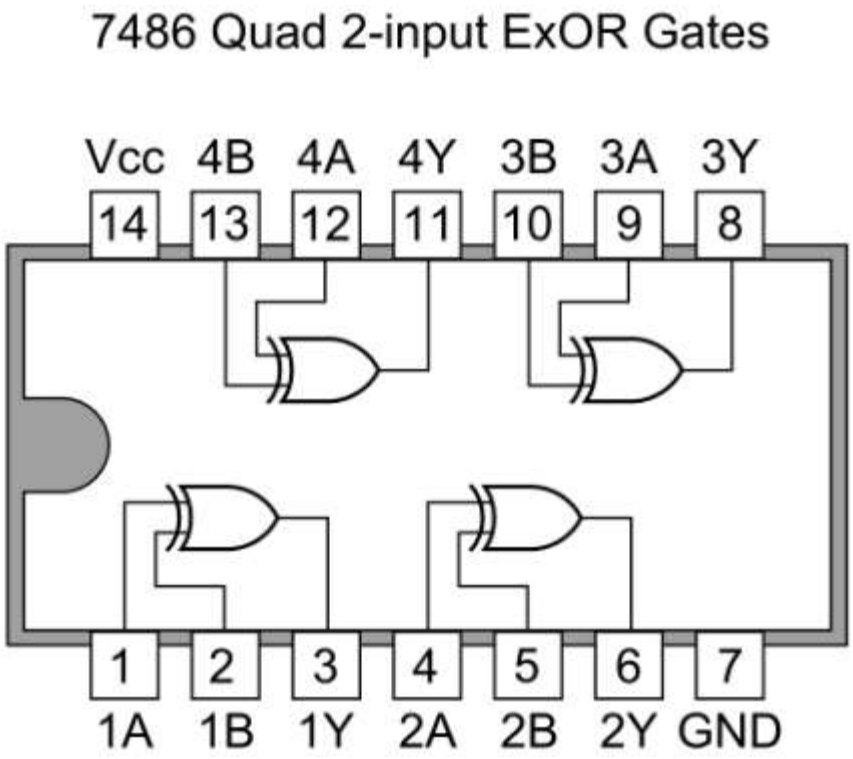
NAND GATE 7400



NOR



EXOR- 7486


7486 Quad 2-input ExOR Gates
Vcc 4B 4A 4Y 3B 3A 3Y
14 13 12 11 10 9 8
1 2 3 4 5 6 7
1A 1B 1Y 2A 2B 2Y GND

Simplification Using Boolean Laws and K map

Unit II Session I, II, III and IV

Boolean Laws

Name	AND form	OR form
Identity law	$1A = A$	$0 + A = A$
Null law	$0A = 0$	$1 + A = 1$
Idempotent law	$AA = A$	$A + A = A$
Inverse law	$A\overline{A} = 0$	$A + \overline{A} = 1$
Commutative law	$AB = BA$	$A + B = B + A$
Associative law	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributive law	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption law	$A(A + B) = A$	$A + AB = A$
De Morgan's law	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$

Prove the following laws

$$A + AB = A$$

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

$$(A + B)(A + C) = A + BC$$

Prove that $A + AB = A$

$A + AB$

↓ Factoring A out of both terms

$A(1 + B)$

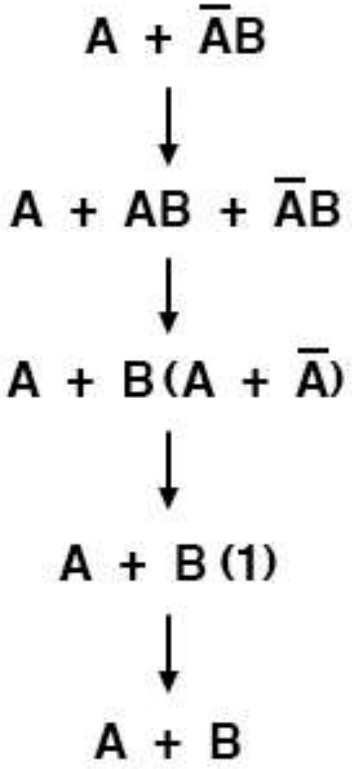
↓ Applying identity $A + 1 = 1$

$A(1)$

↓ Applying identity $1A = A$

A

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



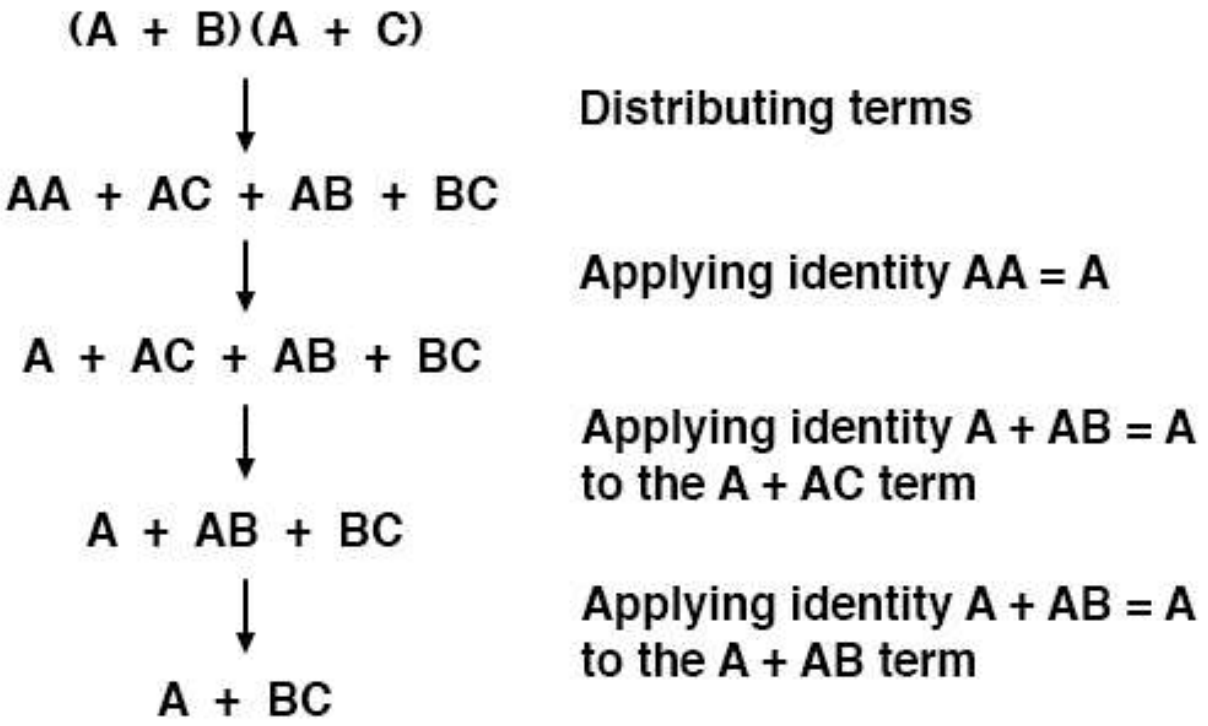
Applying the previous rule to expand A term
$A + AB = A$

Factoring B out of 2nd and 3rd terms

Applying identity $A + \bar{A} = 1$

Applying identity $1A = A$

Prove that (A+B)(A+C)=A+BC



Practice problems

1. Y= ABC +A’+AB’C

= AC(B+B’)+A’

=AC.(1)+A’

= AC+A’

=A’+C

$$F = \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + \overline{A}.\overline{C}$$

1. AB+BC+A’C= AB+A’C

LHS=AB+BC(A+A’)+A’C

=AB+ABC+A’BC+A’C

=AB(1+C)+A’C(B+1)

=AB +A’C

=RHS

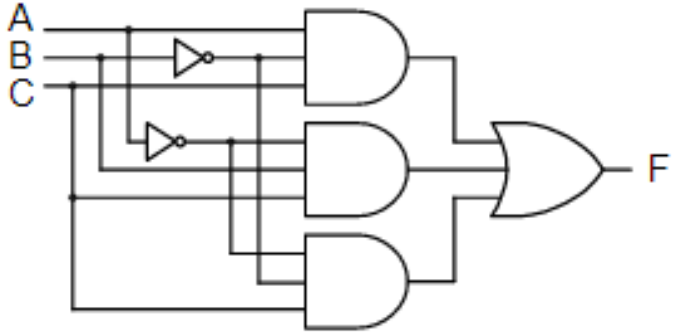
2.SIMPLIFY AND IMPLEMENT USING LOGIC GATE
Y=(AB+BC)C= ABC+BCC=ABC+BC=BC(A+1)=BC

3. AB+CD USING ONLY NOR gates

Sum Of Product (SOP) form
Y=A.B+A.C+A.B.C

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

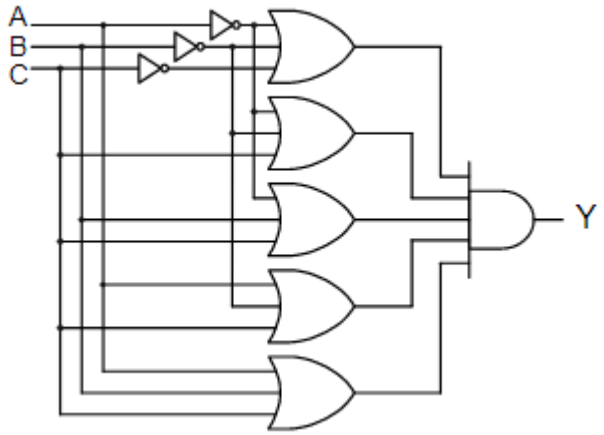
$$Y = A \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot C + \overline{A} \cdot \overline{B} \cdot C$$



POS form (A+B).(B+C).(A+B+C)

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

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



Min terms and Max terms

Variables			Min terms	Max terms
A	B	C	m_i	M_i
0	0	0	A' B' C' = m 0	A + B + C = M 0
0	0	1	A' B' C = m 1	A + B + C' = M 1
0	1	0	A' B C' = m 2	A + B' + C = M 2
0	1	1	A' B C = m 3	A + B' + C' = M 3
1	0	0	A B' C' = m 4	A' + B + C = M 4
1	0	1	A B' C = m 5	A' + B + C' = M 5
1	1	0	A B C' = m 6	A' + B' + C = M 6
1	1	1	A B C = m 7	A' + B' + C' = M 7

Write SOP expression for following truth table

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

$$F = \sum(m_1, m_2, m_3, m_5)$$

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

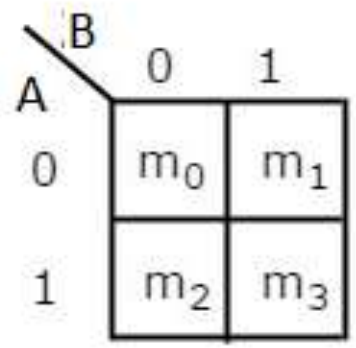
K- Map

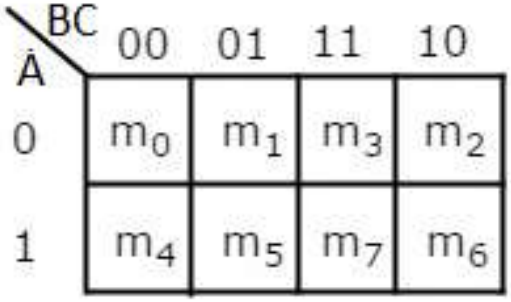
The K-map method of solving the logical expressions is referred to as the graphical technique of simplifying Boolean expressions. K-maps are also referred to as 2D truth tables as each K-map is nothing but a different format of representing the values present in a one-dimensional truth table.

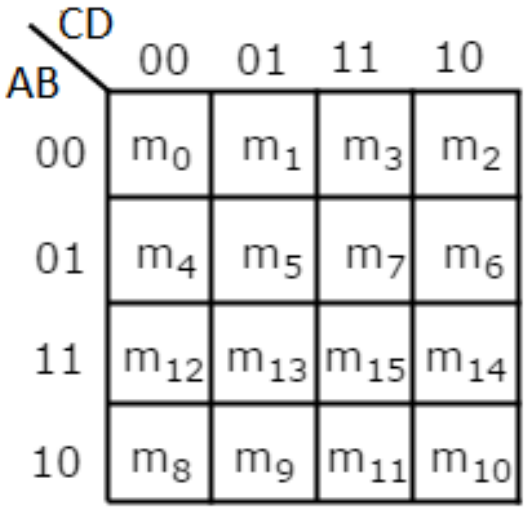
K-maps basically deal with the technique of inserting the values of the output variable in cells within a rectangle or square grid according to a definite pattern. The number of cells in the K-map is determined by the number of input variables and is mathematically expressed as two raised to the power of the number of input variables, i.e., 2^n, where the number of input variables is n.

Thus, to simplify a logical expression with two inputs, we require a K-map with 4 ($=2^2$) cells. A four-input logical expression would lead to a 16 ($=2^4$) celled-K-map, and so on.

2,3,4 VARIABLE K MAPS







2 variable k maps

A. SOP: -

A \ B	$\overline{B}$ 0	B 1
$\overline{A}$ 0	$\overline{A}.\overline{B}$	$\overline{A}.B$
A 1	$A.\overline{B}$	A.B

B. POS: -

A \ B	B 0	$\overline{B}$ 1
A 0	A+B	$A+\overline{B}$
$\overline{A}$ 1	$\overline{A}+B$	$\overline{A}+\overline{B}$

3 variable k maps

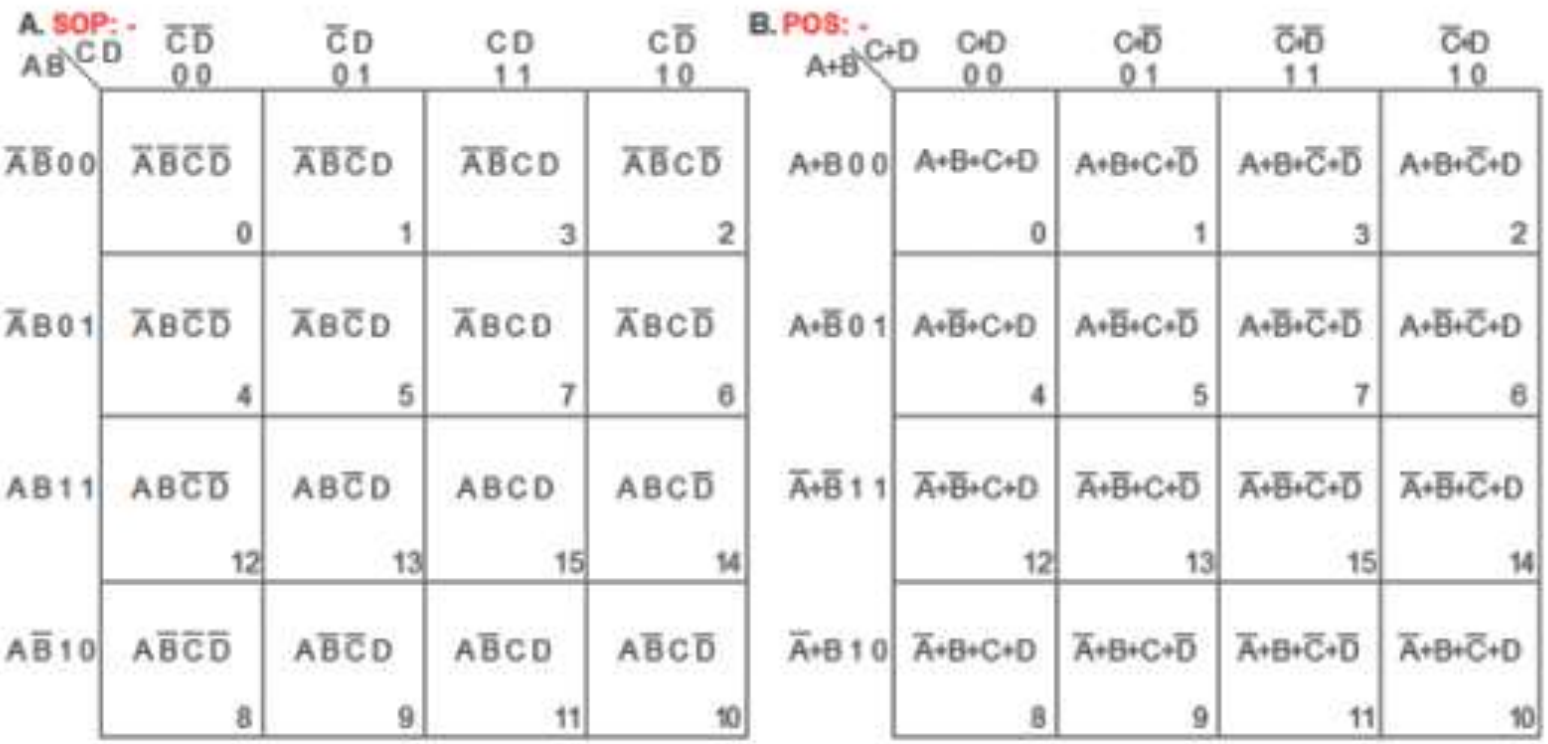
A. SOP: -

A \ BC	$\overline{B}\,\overline{C}$ 0 0	$\overline{B}\,C$ 0 1	$B\,C$ 1 1	$B\,\overline{C}$ 1 0
$\overline{A}$ 0	$\overline{A}\,\overline{B}\,\overline{C}$ 0	$\overline{A}\,\overline{B}\,C$ 1	$\overline{A}\,B\,C$ 3	$\overline{A}\,B\,\overline{C}$ 2
A 1	$A\,\overline{B}\,\overline{C}$ 4	$A\,\overline{B}\,C$ 5	$A\,B\,C$ 7	$A\,B\,\overline{C}$ 6

B. POS: -

A \ B+C	B+C 0 0	B+$\overline{C}$ 0 1	$\overline{B}$+$\overline{C}$ 1 1	$\overline{B}$+C 1 0
A 0	A+B+C 0	A+B+$\overline{C}$ 1	A+$\overline{B}$+$\overline{C}$ 3	A+$\overline{B}$+C 2
$\overline{A}$ 1	$\overline{A}$+B+C 4	$\overline{A}$+B+$\overline{C}$ 5	$\overline{A}$+$\overline{B}$+$\overline{C}$ 7	$\overline{A}$+$\overline{B}$+C 6

4 VARIABLE K MAP



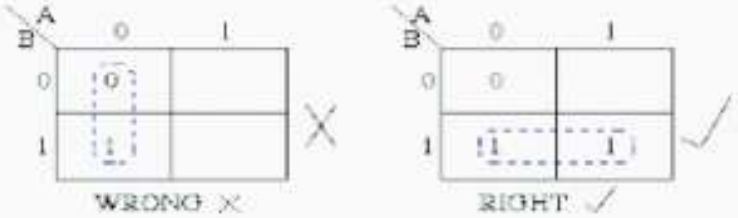
Rules for k map

- No zeros allowed.
- No diagonals.
- Only power of 2 number of cells in each group.
- Groups should be as large as possible.
- Every one must be in at least one group.
- Overlapping allowed.
- Wrap around allowed.
- Fewest number of groups possible.

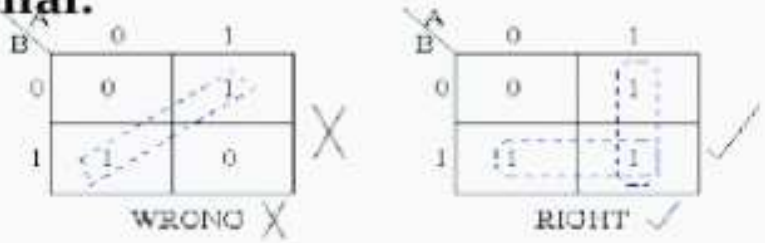
Rules of K map

The Karnaugh map uses the following rules for the simplification of expressions by grouping together adjacent cells containing *ones*.

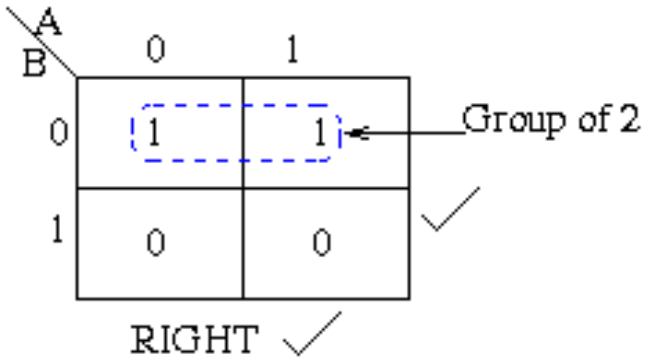
1.Groups may not include any cell containing a zero.

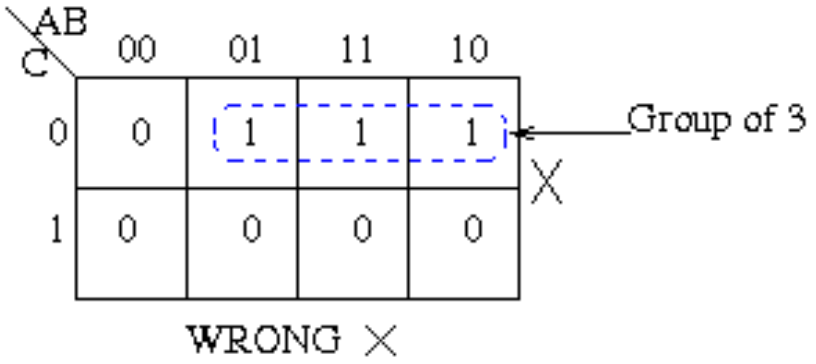


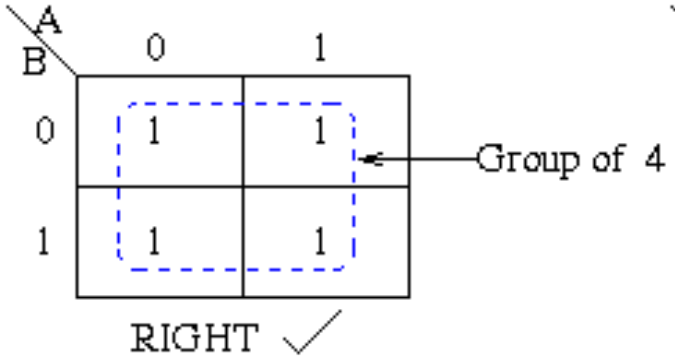
2.Groups may be horizontal or vertical, but not diagonal.

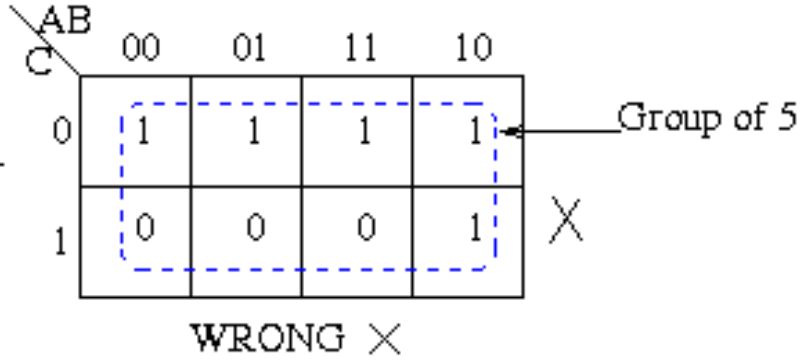


- **Groups must contain 1, 2, 4, 8, or in general 2^n cells.**
  That is if n = 1, a group will contain two 1's since $2^1 = 2$.
  If n = 2, a group will contain four 1's since $2^2 = 4$.



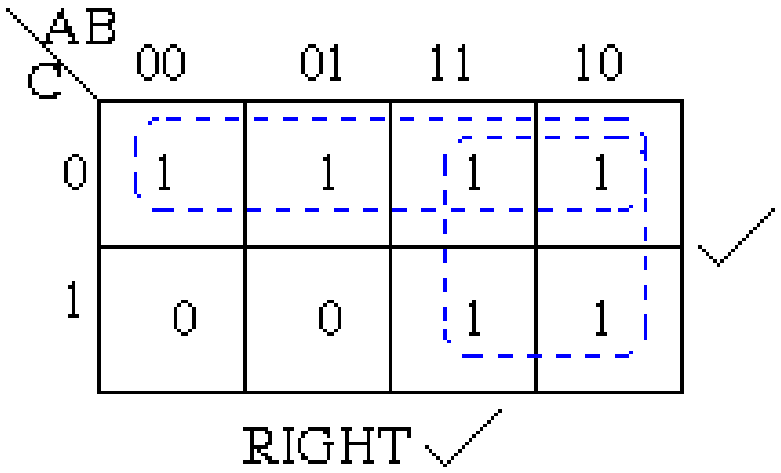




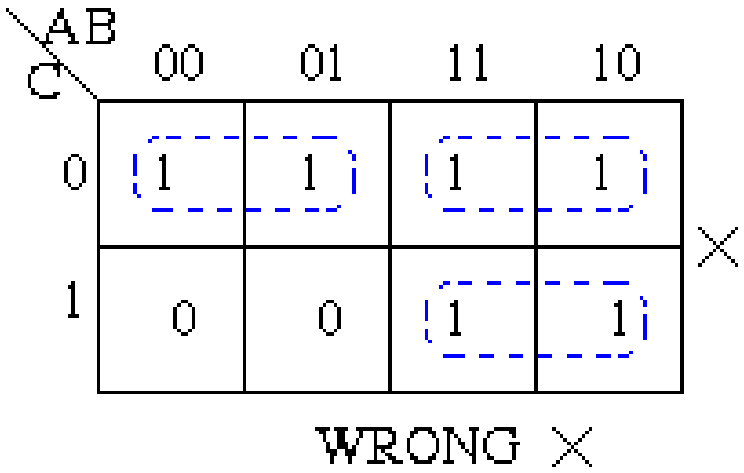


Form minimum number of groups

Each group should be as large as possible.



RIGHT ✓

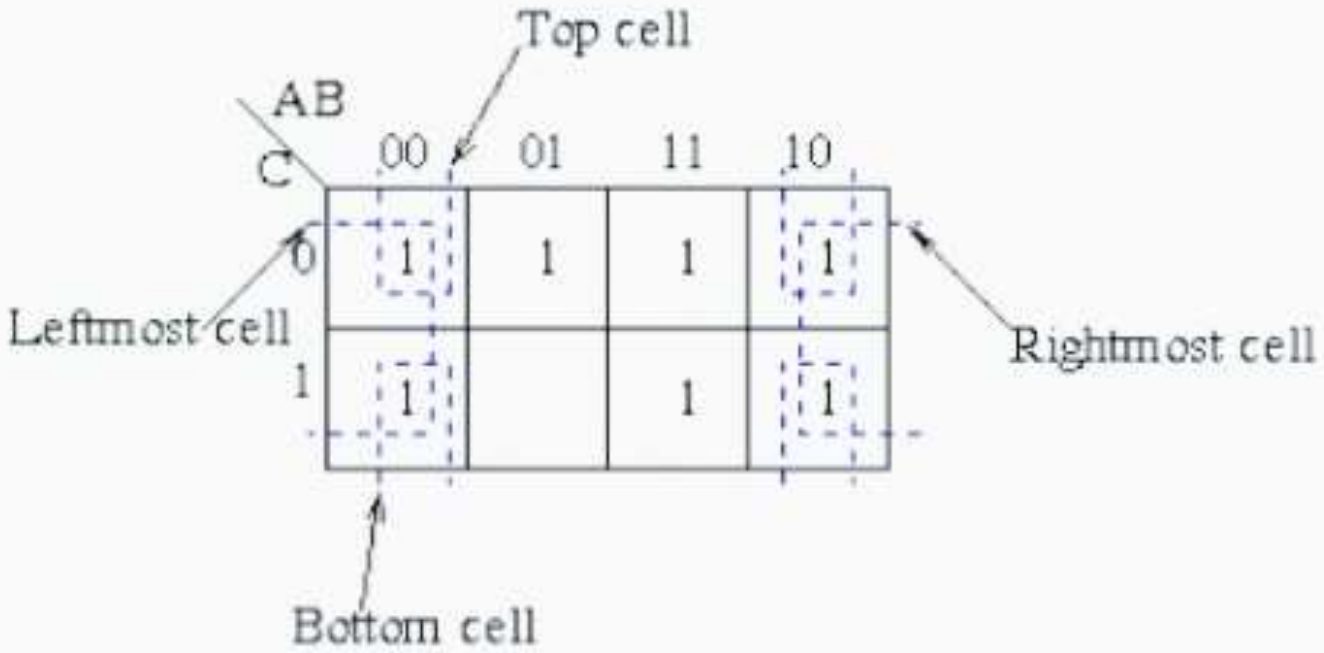


WRONG ✗

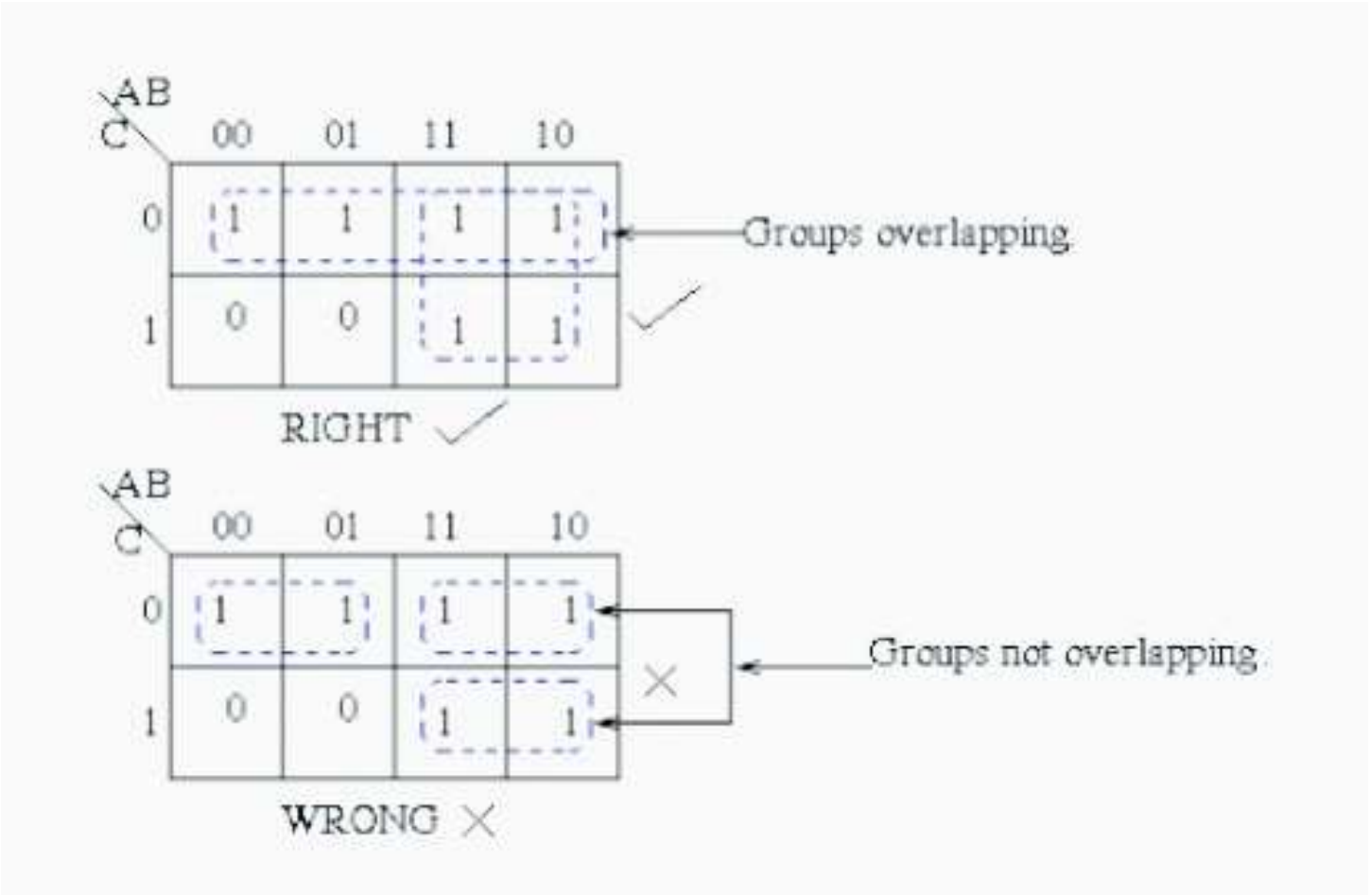
(Note that no Boolean laws broken, but not sufficiently minimal)

Wrap around

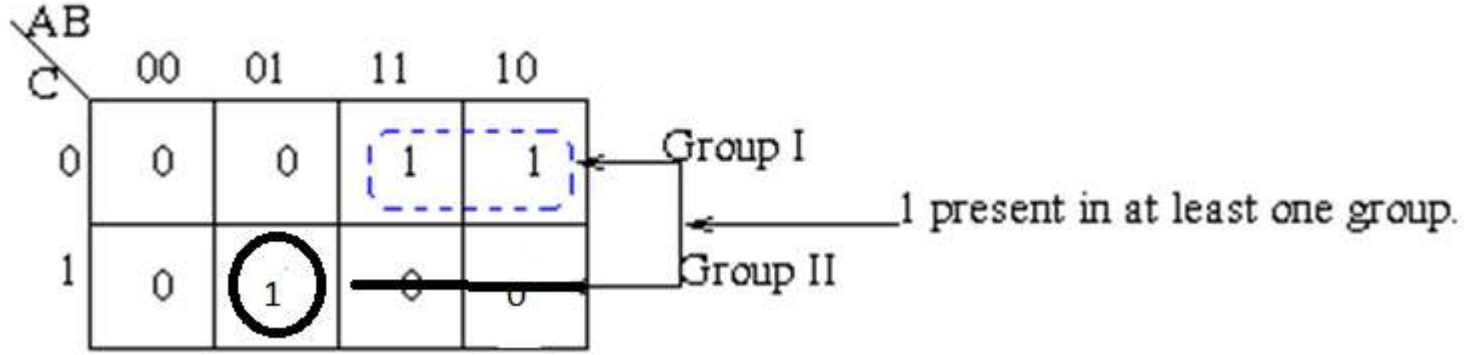
The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.



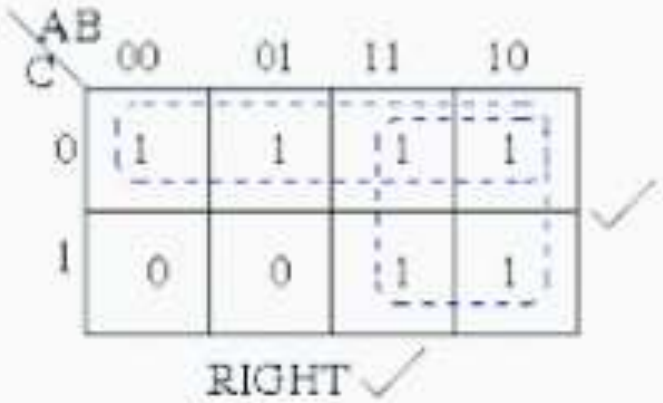
Overlapping of groups



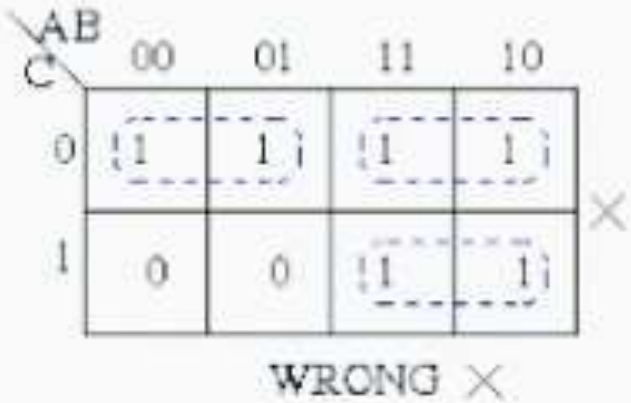
Each cell containing a *one* must be in at least one group.



Each group should be as large as possible.



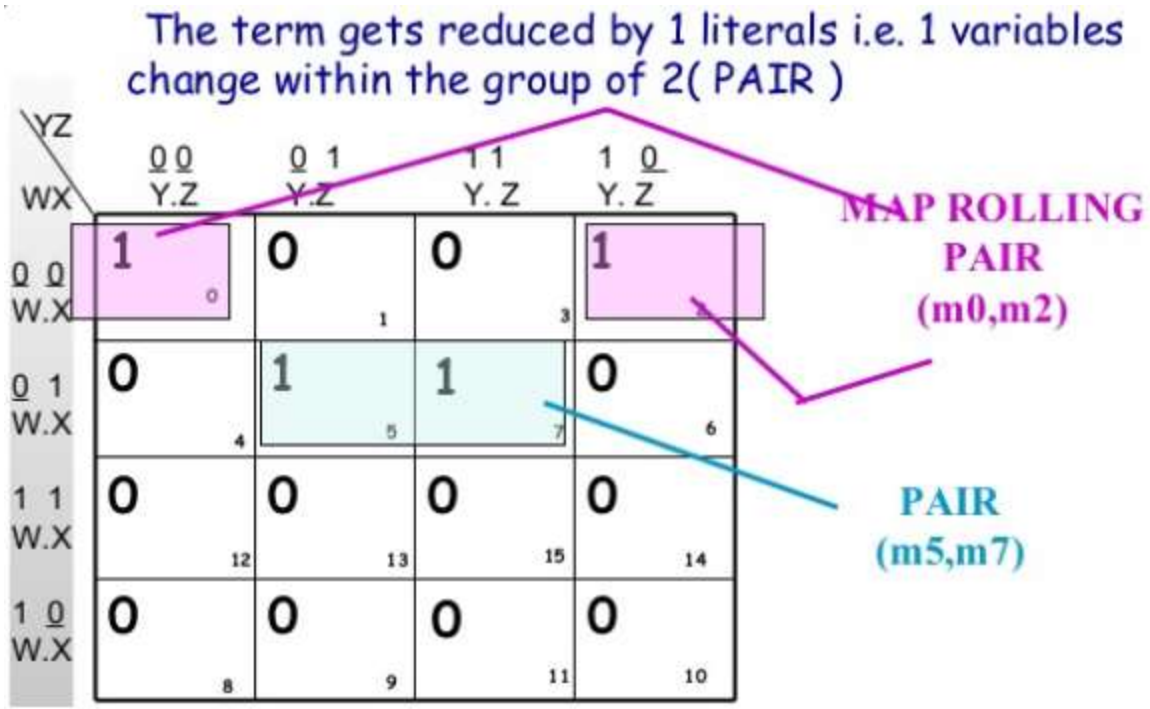
RIGHT ✓



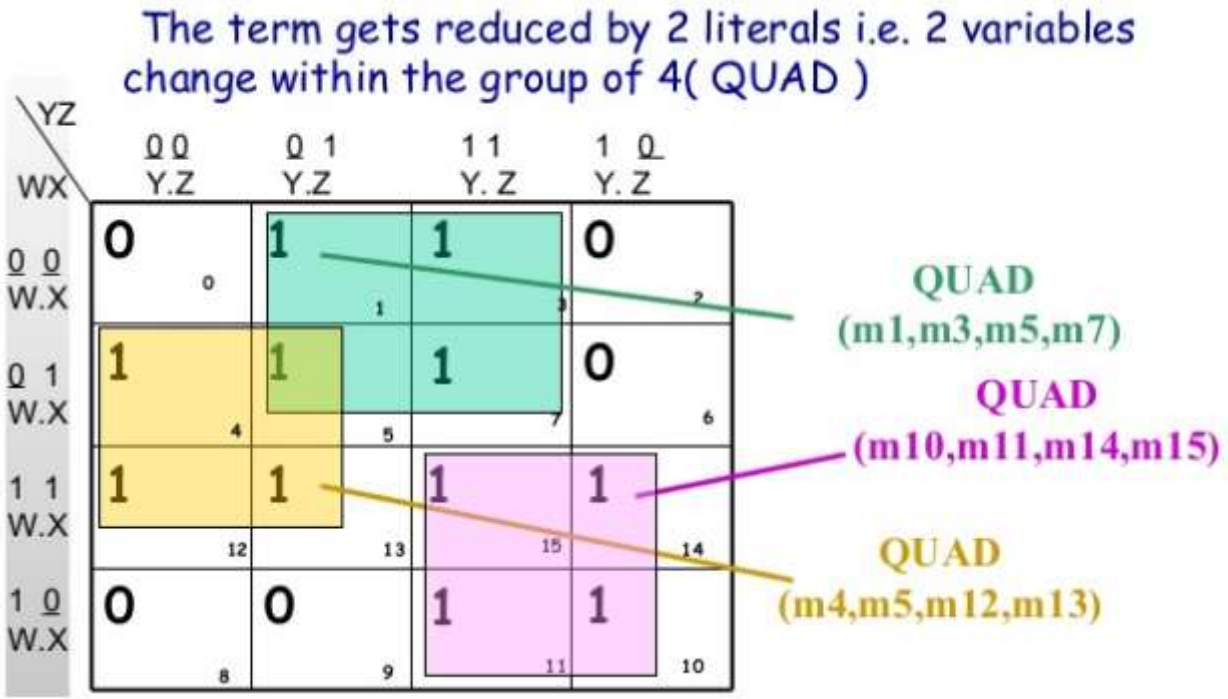
WRONG ×

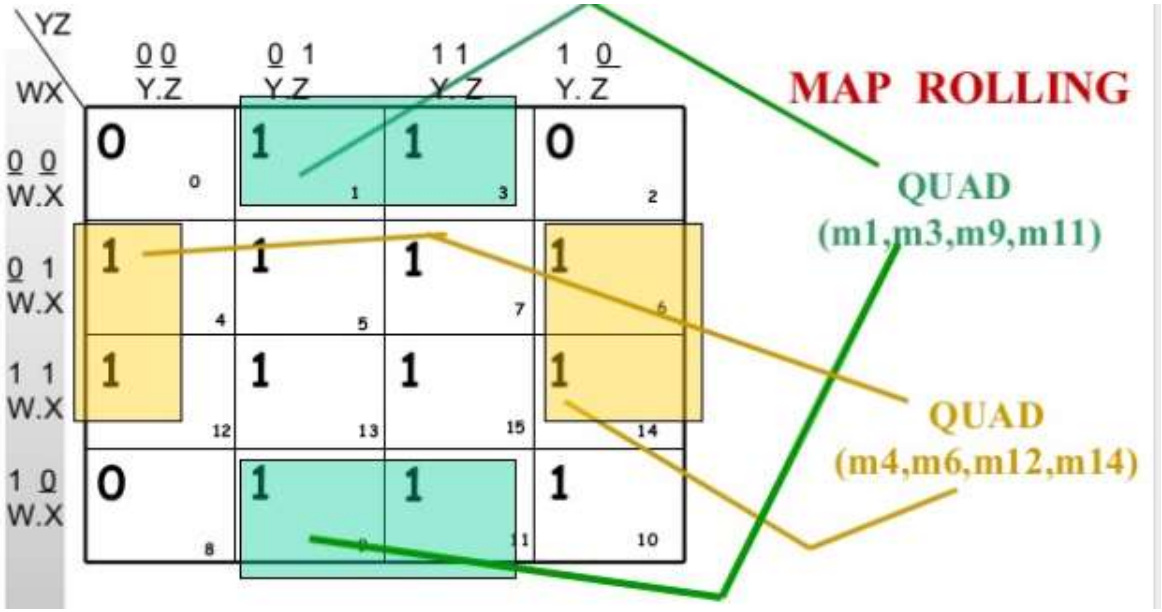
(Note that no Boolean laws broken, but not sufficiently minimal)

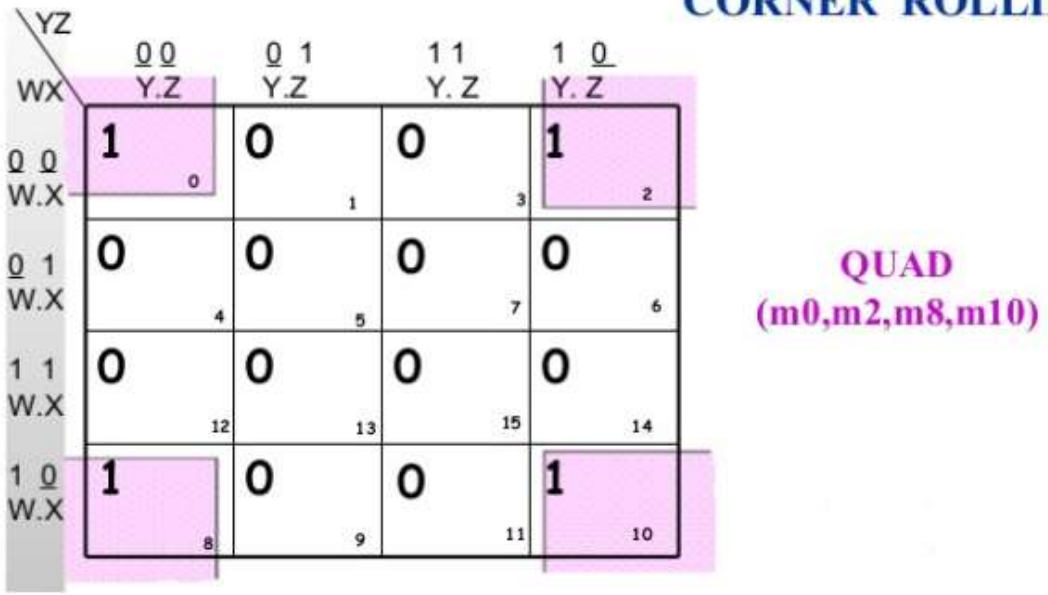
PAIR


The term gets reduced by 1 literals i.e. 1 variables change within the group of 2(PAIR)
YZ
WX
0 0 Y.Z
0 1 Y.Z
1 1 Y. Z
1 0 Y. Z
0 0 W.X
0 1 W.X
1 1 W.X
1 0 W.X
1
0
0
1
0
1
1
0
0
0
0
0
0
0
0
0
0
1
2
3
4
5
7
6
12
13
15
14
8
9
11
10
MAP ROLLING PAIR (m0,m2)
PAIR (m5,m7)

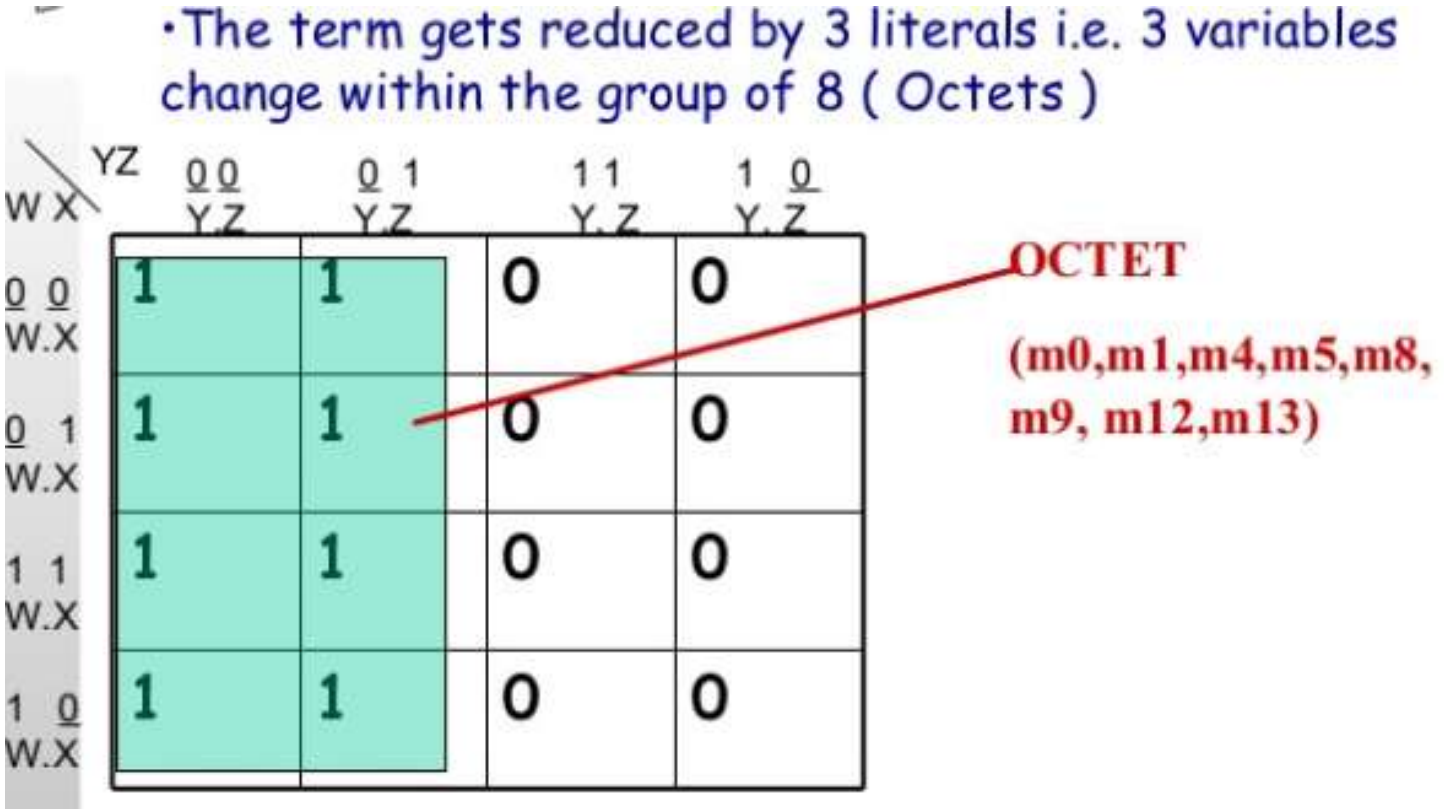
QUAD


The term gets reduced by 2 literals i.e. 2 variables change within the group of 4(QUAD)
YZ
WX
0 0 Y.Z
0 1 Y.Z
1 1 Y. Z
1 0 Y. Z
0 0 W.X
0 1 W.X
1 1 W.X
1 0 W.X
QUAD (m1,m3,m5,m7)
QUAD (m10,m11,m14,m15)
QUAD (m4,m5,m12,m13)

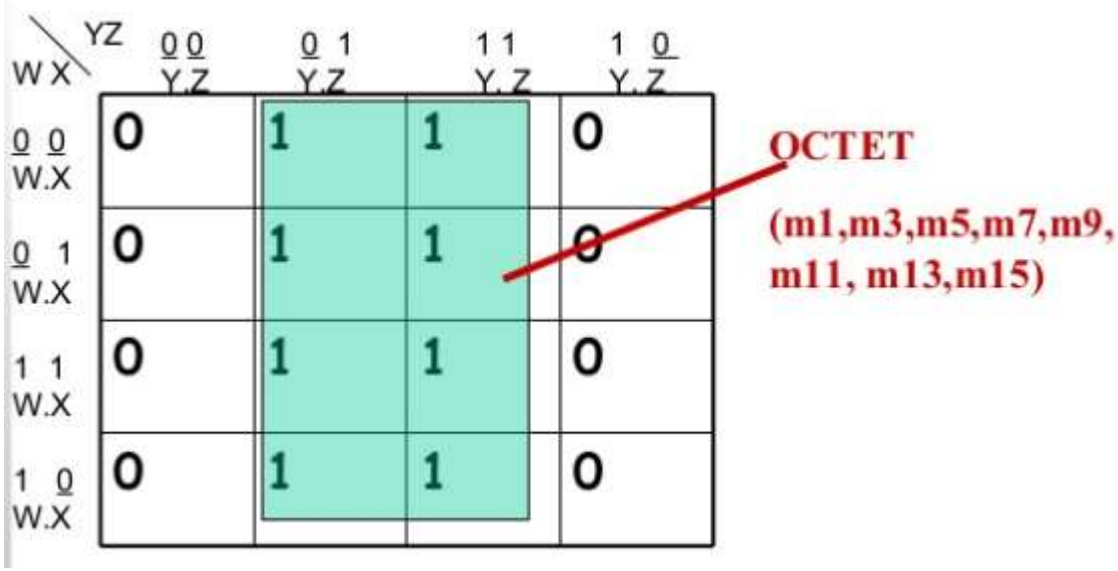

YZ
WX
00 Y.Z
01 Y.Z
11 Y.Z
10 Y.Z
00 W.X
01 W.X
11 W.X
10 W.X
MAP ROLLING
QUAD
(m1,m3,m9,m11)
QUAD
(m4,m6,m12,m14)

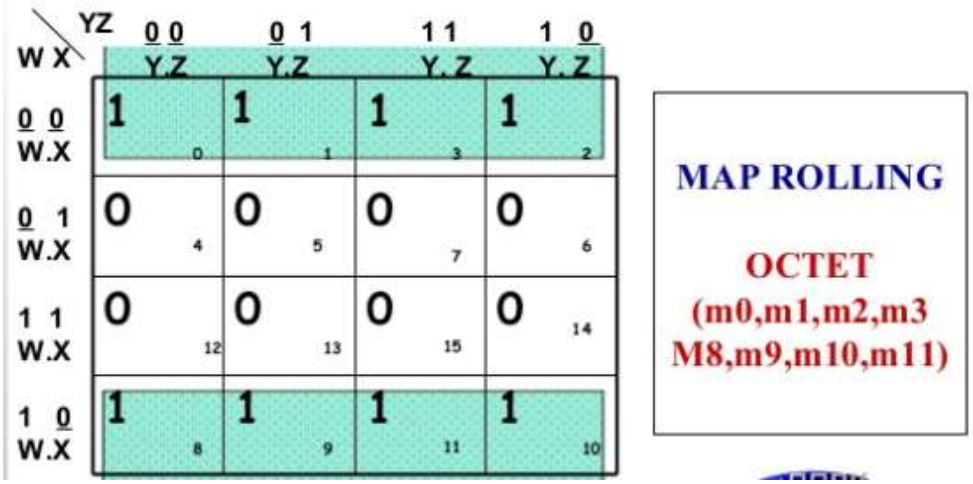

CORNER ROLLING
YZ
WX
00 Y.Z
01 Y.Z
11 Y.Z
10 Y.Z
00 W.X
01 W.X
11 W.X
10 W.X
QUAD
(m0,m2,m8,m10)

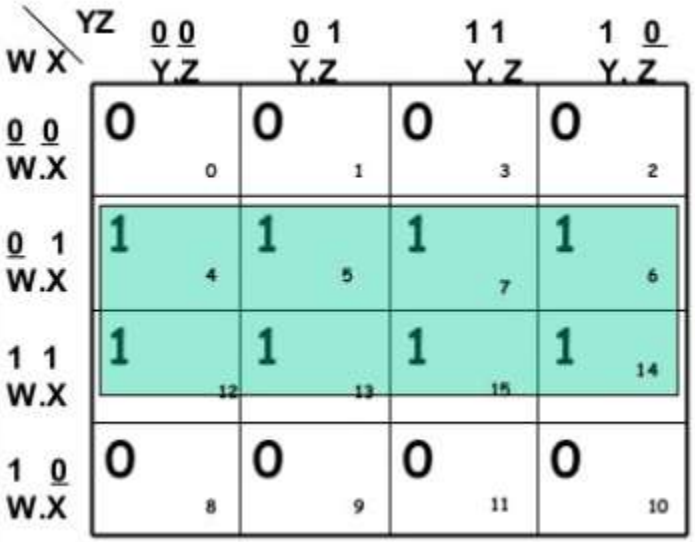
OCTATE


•The term gets reduced by 3 literals i.e. 3 variables change within the group of 8 (Octets)
YZ
W X
0 0
Y.Z
0 1
Y.Z
1 1
Y.Z
1 0
Y.Z
0 0
W.X
0 1
W.X
1 1
W.X
1 0
W.X
1 1 0 0
1 1 0 0
1 1 0 0
1 1 0 0
OCTET
(m0,m1,m4,m5,m8, m9, m12,m13)

0CTATE

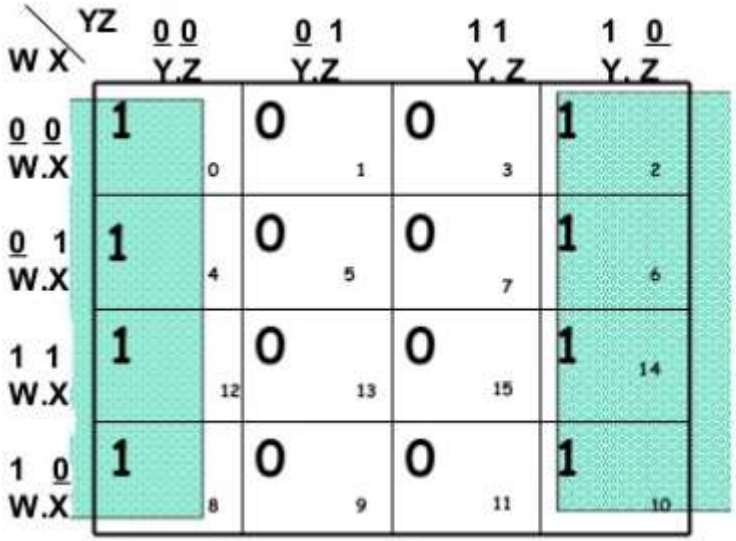

YZ
W X
0 0 Y.Z
0 1 Y.Z
1 1 Y.Z
1 0 Y.Z
0 0 W.X
0 1 W.X
1 1 W.X
1 0 W.X
OCTET
(m1,m3,m5,m7,m9, m11, m13,m15)


YZ
W X
0 0 Y.Z
0 1 Y.Z
1 1 Y.Z
1 0 Y.Z
0 0 W.X
0 1 W.X
1 1 W.X
1 0 W.X
MAP ROLLING
OCTET
(m0,m1,m2,m3
M8,m9,m10,m11)



OCTET

(m4,m5,m6,m7,m12, m13, m14,m15)

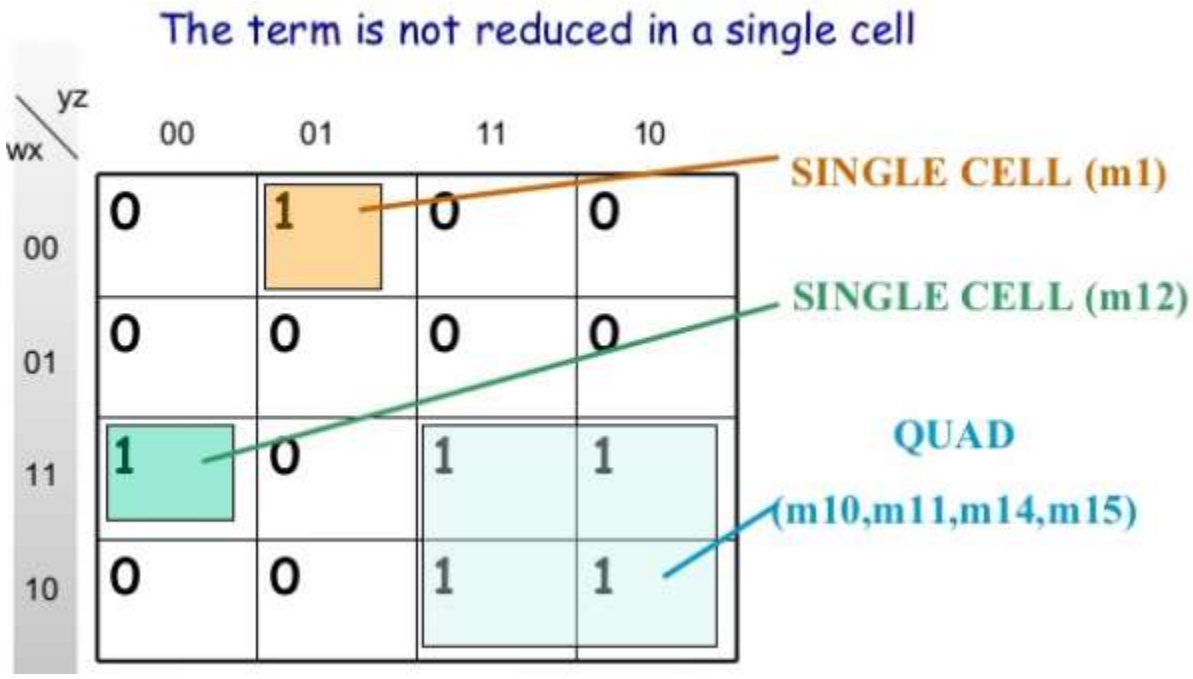


MAP ROLLING

OCTET

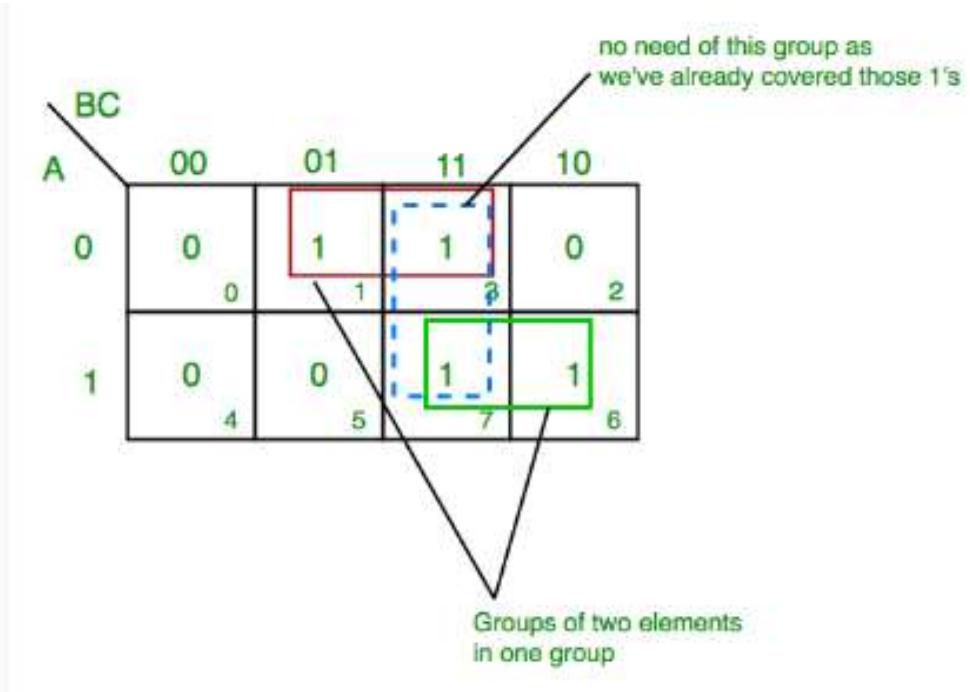
(m0,m2,m4,m6, m8, m10, m12,m14)

Single cell

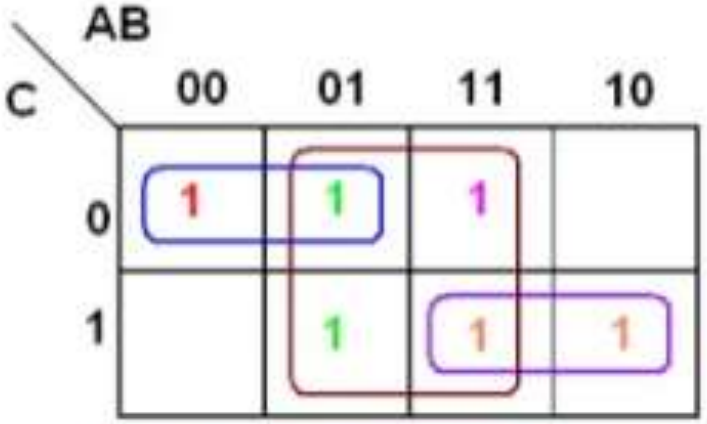

The term is not reduced in a single cell
yz
wx
00
01
11
10
SINGLE CELL (m1)
SINGLE CELL (m12)
QUAD
(m10,m11,m14,m15)

Simplify using k map

F(A,B,C)=∑ (1,3,6,7)



(A'C+AB)



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

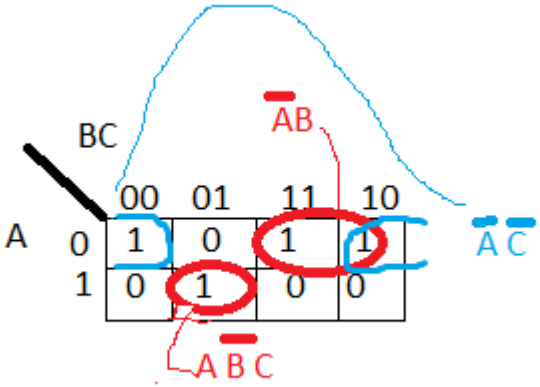
$\bar{A}.\bar{C} + B + A.C$

Simplify using k map

$f(a,b,c)= \sum m\ (m_0,m_2,m_3,m_5)$

A	BC: B'C' (00)	B'C (01)	BC (11)	BC' (10)
A' (0)	m_0	m_1	m_3	m_2
A (1)	m_4	m_5	m_7	m_6

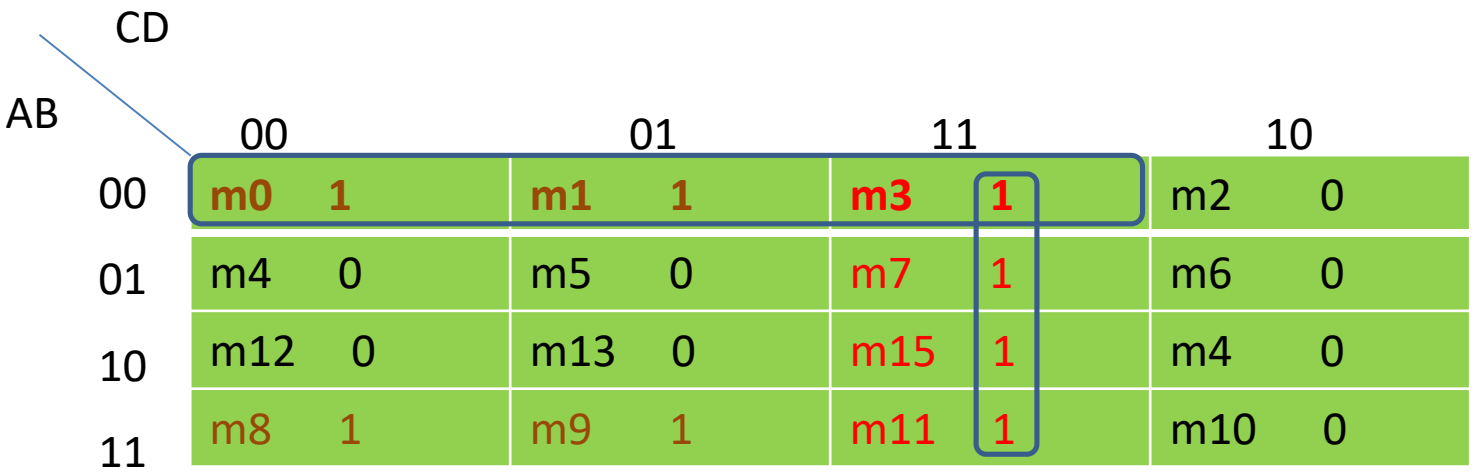
A	BC: 00	01	11	10
0	1	0	1	1
1	0	1	0	0



$= A'B+A'C'+AB'C$

Simplify using k map

- ∑m(0,1,3,7,8,9,11,15)



Y = CD+ B'C'

Simplify using k map

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

C \ AB	00	01	11	10
0	1	1	1	0
1	0	1	1	1

$\overline{A}.\overline{B}.\overline{C} + \overline{A}.B + A.B.\overline{C} + A.C$

Exercise

Simplify using k map and implement using logic gates

$F(A,B,C,D)=\sum m(0,1,2,3,5,7,8,9,11,14,15)$

Time: 10 minutes

Don’t care

One of the very significant and useful concept in simplifying the output expression using K-Map is the concept of “Don’t Cares”. The “Don’t Care” conditions allow us to replace the empty cell of a K-Map to form a grouping of the variables which is larger than that of forming groups without don’t cares. While forming groups of cells, we can consider a “Don’t Care” cell as 1 or 0 or we can also ignore that cell. Therefore, “Don’t Care” condition can help us to form a larger group of cells.

A Don’t Care cell can be represented by a cross(X) in K-Maps representing an invalid combination. For example, in Excess-3 code system, the states 0000, 0001, 0010 are invalid or unspecified. These states are called don’t cares.

Significance of "Don't Care" Conditions:

Don't Care conditions has the following significance in designing of the digital circuits:

1. **Simplification of the output:**
   These conditions denotes inputs that are invalid for a given digital circuit. Thus, they can used to further simplify the boolean output expression of a digital circuit.

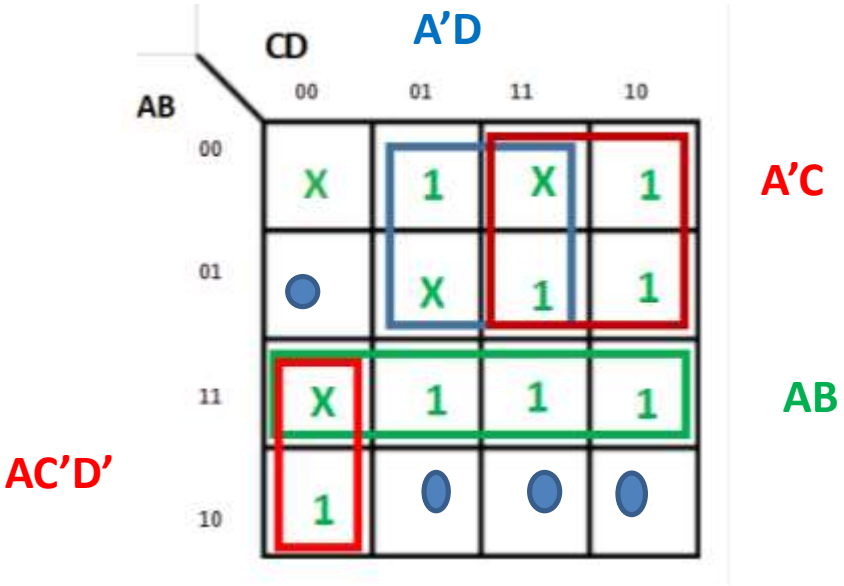
2. **Reduction in number of gates required:**
   Simplification of the expression reduces the number of gates to be used for implementing the given expression. Therefore, don't cares make the digital circuit design more economical.

3. **Reduced Power Consumption:**
   While grouping the terms long with don't cares reduces switching of the states. This decreases the memory space that is required to represent a given digital circuit which in turn results in less power consumption.

4. **Represent Invalid States in Code Converters:**
   These are used in code converters. For example- In design of 4-bit BCD-to-XS-3 code converter, the input combinations 1010, 1011, 1100, 1101, 1110, and 1111 are don't cares.

Minimise the following function in SOP minimal form using K-Maps:

$F(A, B, C, D) = m(1, 2, 6, 7, 8, 13, 14, 15) + d(0, 3, 5, 12)$

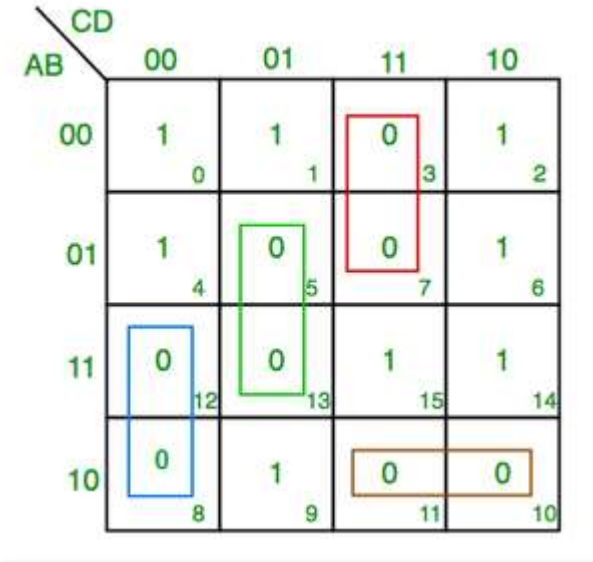


$$f = AC'D' + A'D + A'C + AB$$

POS

f(A,B,C,D)=π M(3,5,7,8,10,11,12,13)

f(A,B,C,D)=π M(3,5,7,8,10,11,12,13)



From **green** group we find terms

C' D B

Taking their complement and summing them

(C+D'+B')

From **red** group we find terms

C D A'

Taking their complement and summing them

(C'+D'+A)

From **blue** group we find terms

A C' D'

Taking their complement and summing them

(A'+C+D)

From **brown** group we find terms

A B' C

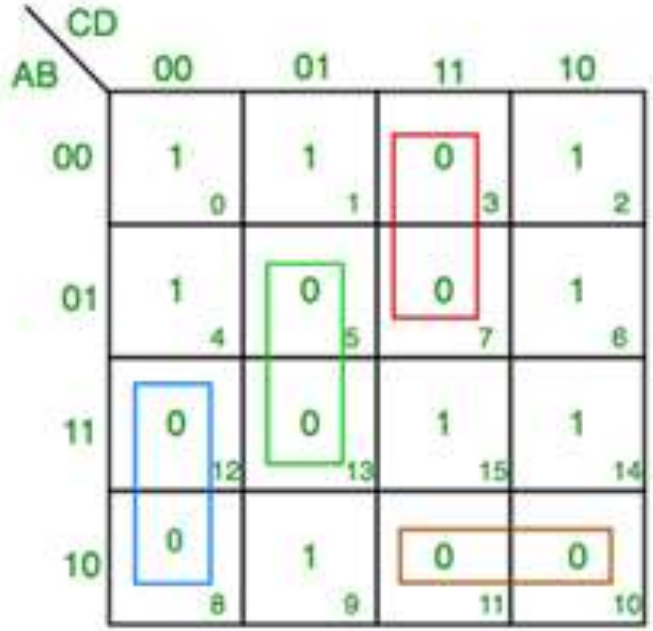
Taking their complement and summing them

(A'+B+C')

(C+D'+B').(C'+D'+A).(A'+C+D).(A'+B+C')

POS: -

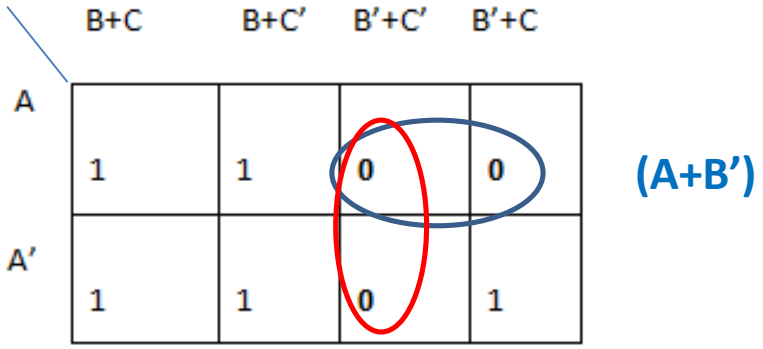
A+B \ C+D	C+D (0 0)	C+$\overline{D}$ (0 1)	$\overline{C}$+$\overline{D}$ (1 1)	$\overline{C}$+D (1 0)
A+B (0 0)	A+B+C+D 0	A+B+C+$\overline{D}$ 1	A+B+$\overline{C}$+$\overline{D}$ 3	A+B+$\overline{C}$+D 2
A+$\overline{B}$ (0 1)	A+$\overline{B}$+C+D 4	A+$\overline{B}$+C+$\overline{D}$ 5	A+$\overline{B}$+$\overline{C}$+$\overline{D}$ 7	A+$\overline{B}$+$\overline{C}$+D 6
$\overline{A}$+$\overline{B}$ (1 1)	$\overline{A}$+$\overline{B}$+C+D 12	$\overline{A}$+$\overline{B}$+C+$\overline{D}$ 13	$\overline{A}$+$\overline{B}$+$\overline{C}$+$\overline{D}$ 15	$\overline{A}$+$\overline{B}$+$\overline{C}$+D 14
$\overline{A}$+B (1 0)	$\overline{A}$+B+C+D 8	$\overline{A}$+B+C+$\overline{D}$ 9	$\overline{A}$+B+$\overline{C}$+$\overline{D}$ 11	$\overline{A}$+B+$\overline{C}$+D 10



(C+D'+B').(C'+D'+A).(A'+C+D).(A'+B+C')

Simplify following POS expression

$F(A,B,C)=\pi\ M(2,3,7)$



Y= (B'+C'). (A+B')

M= 0,4,5,7,10,11,14,15

0	1	1	1
0	0	0	1
1	1	0	0
1	1	0	0


THANK
YOU

Quine Mc Cluskey Method

Unit II

Gauri Rao

Quine-McClukey method

Quine-McClukey method is a tabular method based on the concept of prime implicants. We know that **prime implicant** is a product or sum term, which can't be further reduced by combining with any other product or sum terms of the given Boolean function.

PI- Group of minterms which can not be combined with any other minterm or groups

EPI- Prime implicant in which one or more minterms are unique. It contains at least 1 minterm which is not contained in any other prime implicant.

Procedure

Step 1 – Arrange the given min terms in an **ascending order** and make the groups based on the number of ones present in their binary representations. So, there will be **at most 'n+1' groups** if there are 'n' Boolean variables in a Boolean function or 'n' bits in the binary equivalent of min terms.

Step 2 – Compare the min terms present in **successive groups**. If there is a change in only one-bit position, then take the pair of those two min terms. Place this symbol '_' in the differed bit position and keep the remaining bits as it is.

Step 3 – Repeat step2 with newly formed terms till we get all **prime implicants**.

Step 4 – Formulate the **prime implicant table**. It consists of set of rows and columns. Prime implicants can be placed in row wise and min terms can be placed in column wise. Place '1' in the cells corresponding to the min terms that are covered in each prime implicant.

Step 5 – Find the essential prime implicants by observing each column. If the min term is covered only by one prime implicant, then it is **essential prime implicant**. Those essential prime implicants will be part of the simplified Boolean function.

Step 6 – Reduce the prime implicant table by removing the row of each essential prime implicant and the columns corresponding to the min terms that are covered in that essential prime implicant. Repeat step 5 for Reduced prime implicant table. Stop this process when all min terms of given Boolean function are over.

Example:- Simplify the following Boolean function f(W,X,Y,Z)=∑m(2,6,8,9,10,11,14,15) using Quine-McClukey tabular method.

The given Boolean function is in **sum of min terms** form. It is having 4 variables W, X, Y & Z. The given min terms are 2, 6, 8, 9, 10, 11, 14 and 15. The ascending order of these min terms based on the number of ones present in their binary equivalent is 2, 8, 6, 9, 10, 11, 14 and 15. The following table shows these **min terms and their equivalent binary** representations.

Group Name	Min terms	W	X	Y	Z
GA1	2	0	0	1	0
	8	1	0	0	0
GA2	6	0	1	1	0
	9	1	0	0	1
	10	1	0	1	0
GA3	11	1	0	1	1
	14	1	1	1	0
GA4	15	1	1	1	1

The given min terms are arranged into 4 groups based on the number of ones present in their binary equivalents. The following table shows the possible **merging of min terms** from adjacent groups.

Group Name	Min terms	W	X	Y	Z
GB1	2,6	0	-	1	0
	2,10	-	0	1	0
	8,9	1	0	0	-
	8,10	1	0	-	0
GB2	6,14	-	1	1	0
	9,11	1	0	-	1
	10,11	1	0	1	-
	10,14	1	-	1	0
GB3	11,15	1	-	1	1
	14,15	1	1	1	-

The min terms, which are differed in only one-bit position from adjacent groups are merged. That differed bit is represented with this symbol, '-'. In this case, there are three groups and each group contains combinations of two min terms. The following table shows the possible **merging of min term pairs** from adjacent groups.

Group Name	Min terms	W	X	Y	Z
GB1	2,6,10,14	-	-	1	0
	2,10,6,14	-	-	1	0
	8,9,10,11	1	0	-	-
	8,10,9,11	1	0	-	-
GB2	10,11,14,15	1	-	1	-
	10,14,11,15	1	-	1	-

The successive groups of min term pairs, which are differed in only one-bit position are merged. That differed bit is represented with this symbol, '-'. In this case, there are two groups and each group contains combinations of four min terms. Here, these combinations of 4 min terms are available in two rows. So, we can remove the repeated rows. The reduced table after removing the redundant rows is shown below.

Group Name	Min terms	W	X	Y	Z
GC1	2,6,10,14	-	-	1	0
	8,9,10,11	1	0	-	-
GC2	10,11,14,15	1	-	1	-

Min terms / Prime Implicants	2	6	8	9	10	11	14	15
YZ'	1	1			1		1	
WX'			1	1	1	1		
WY					1	1	1	1

Min terms / Prime Implicants	2	6	8	9	10	11	14	15
YZ’	1	1			1		1	
WX’			1	1	1	1		
WY					1	1	1	1

In this example problem, we got three prime implicants and all the three are essential. Therefore, the **simplified Boolean function** is

$$\mathbf{f}(W, X, Y, Z) = \mathbf{YZ' + WX' + WY.}$$

Practice problem

Simplify following expression using Quine Mc Cluskey method

$y(A,B,C,D) = \sum m\,(0,1,3,7,8,9,11,15)$

Groups

1. 0
2. 1,8
3. 3,9
4. 7,11
5. 15

Group	Minterm	A	B	C	D
1	m0	0	0	0	0
2	m1	0	0	0	1
	m8	1	0	0	0
3	m3	0	0	1	1
	m9	1	0	0	1
4	m7	0	1	1	1
	m11	1	0	1	1
5	m15	1	1	1	1

GROUP	MIN TERM	A	B	C	D
0	m0-m1	0	0	0	-
	mo-m8	-	0	0	0
1	m1-m3	0	0	-	1
	m1-m9	-	0	0	1
	m8-m9	1	0	0	-
2	m3-m7	0	-	1	1
	m3-m11	-	0	1	1
	m9-m11	1	0	-	1
3	m7-m15	-	1	1	1
	m11-m15	1	-	1	1

GROUP	Min	A	B	C	D
0	m0-m1-m8-m9	-	0	0	-
	m0-m8-m1-m-m9	-	0	0	-
1	m1-m9-m3-m11	-	0	-	1
	m1-m3-m9-m11	-	0	-	1
2	m3-m7-m11-m15	-	-	1	1
	m3-m11-m7-m15	-	-	1	1

PI		0	1	3	7	8	9	11	15	

Unit III

Session I
Combinational circuits

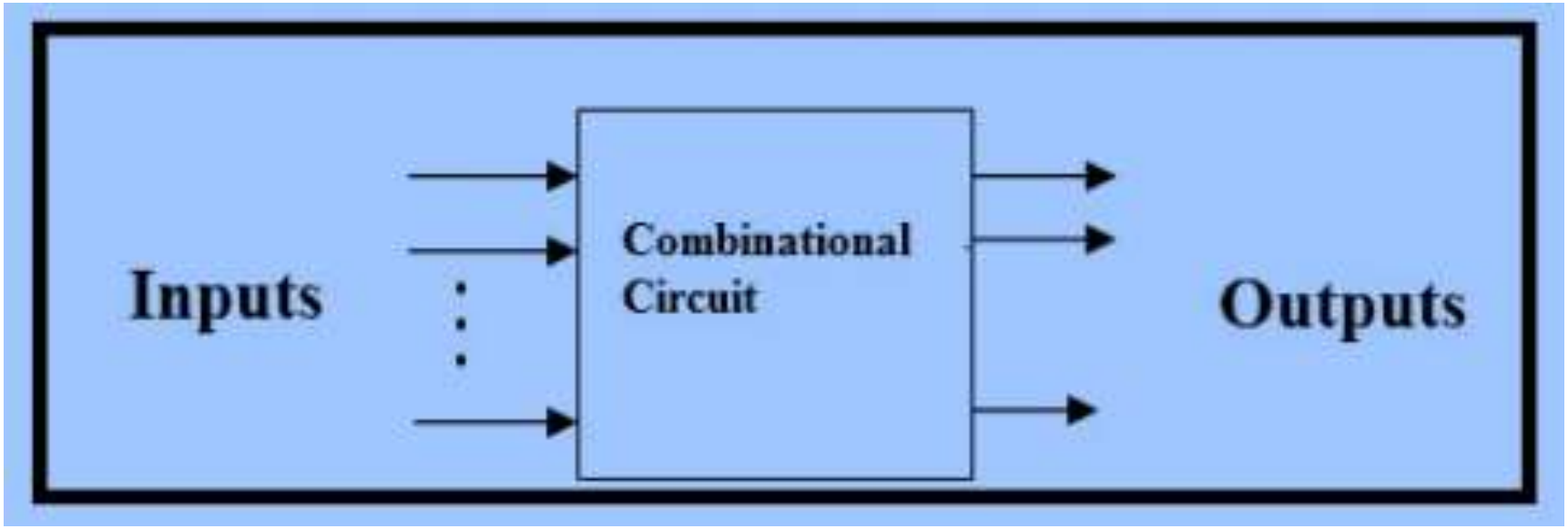
Combinational Circuits

- The output of combinational circuit at any instant of time, depends only on the levels present at input terminals.
- The combinational circuit do not use any memory. The previous state of input does not have any effect on the present state of the circuit.
- A combinational circuit can have an n number of inputs and m number of outputs

Combinational Logic Circuit

- **:**A combinational logic circuit consists of logic gates whose output is determined by the combination of current inputs.
- It consists of input variables, logic gate and output variables.
- No feedback is required.
- No memory is required.
- Examples of Combinational Circuits: Adders, Code converters,Multiplexer, Decoder etc.

Block diagram

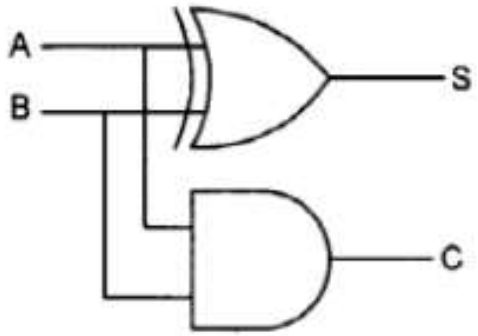


Half Adder

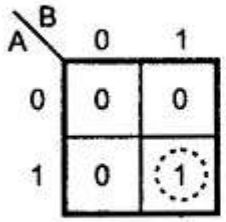
Half adder is a combinational logic circuit with two inputs and two outputs. The half adder circuit is designed to add two single bit binary number A and B. It is the basic building block for addition of two **single** bit numbers. This circuit has two outputs **carry** and **sum**.

Design Half Adder

Inputs		Outputs	
A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

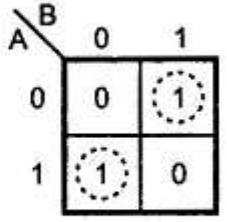


For Carry

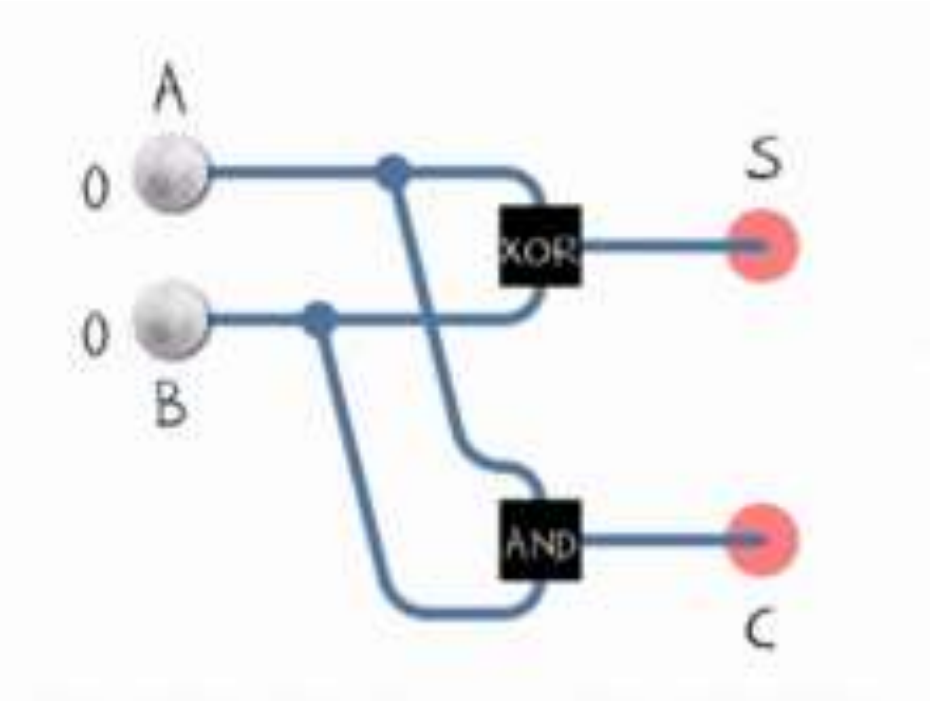


Carry = AB

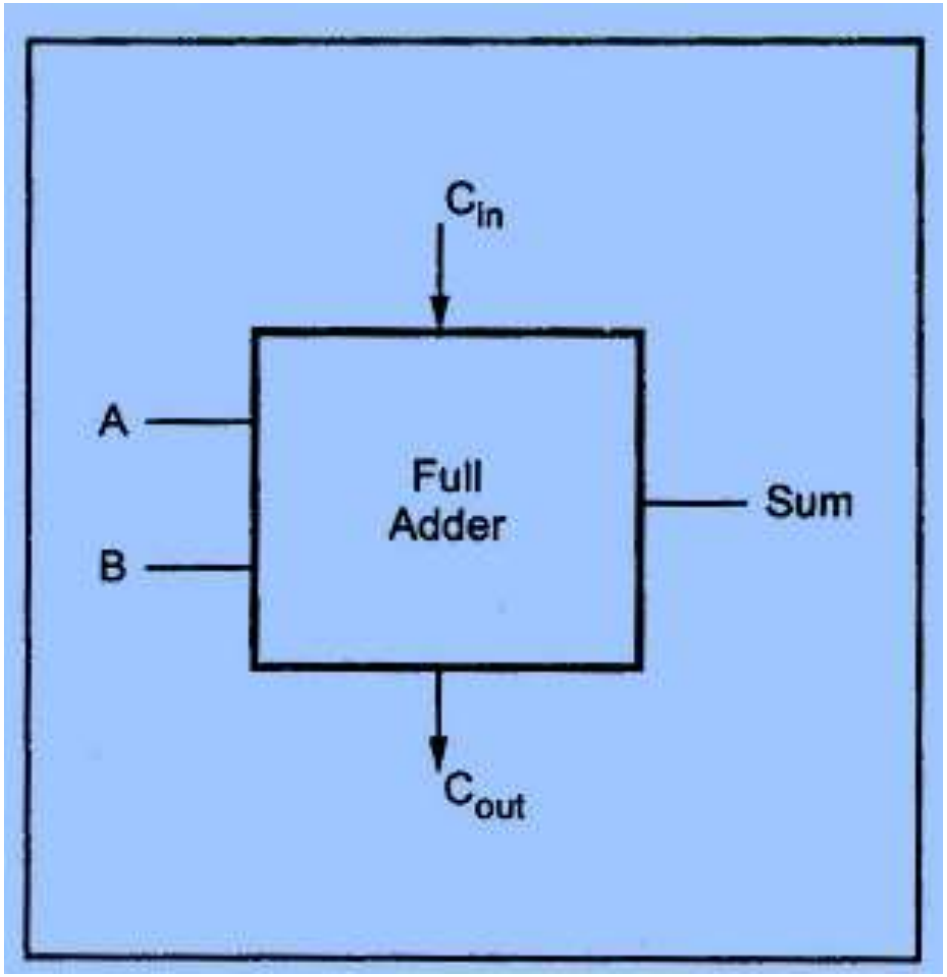
For Sum



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


A
0
0
B
XOR
AND
S
C

Full Adder

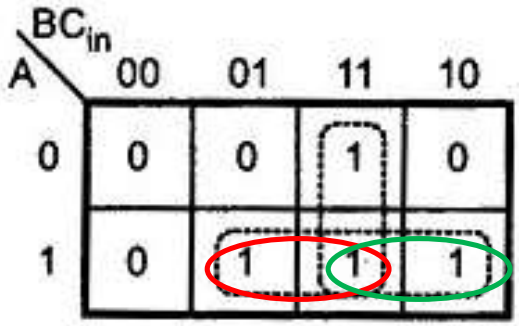


Truth Table

Inputs			Outputs	
A	B	C_{in}	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

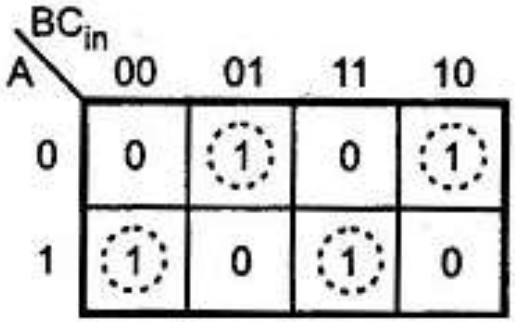
K map

For Carry (C_{out})



$C_{out} = AB + A\,C_{in} + B\,C_{in}$

For Sum



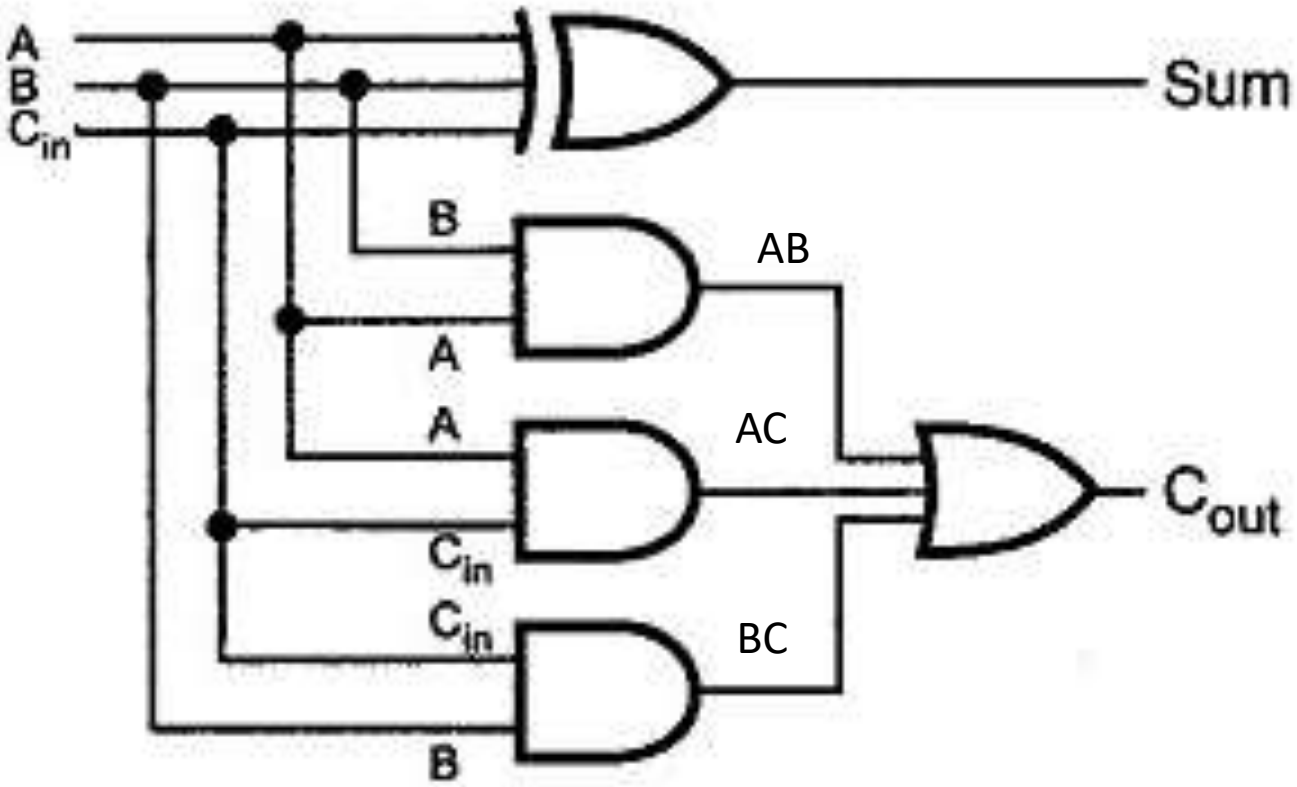
$Sum = \overline{A}\,\overline{B}C_{in} + \overline{A}B\overline{C}_{in} + A\overline{B}\,\overline{C}_{in} + ABC_{in}$

Sum
Cin(y')+Cin'(y)=Cin xor Y=Cin xor A xor B

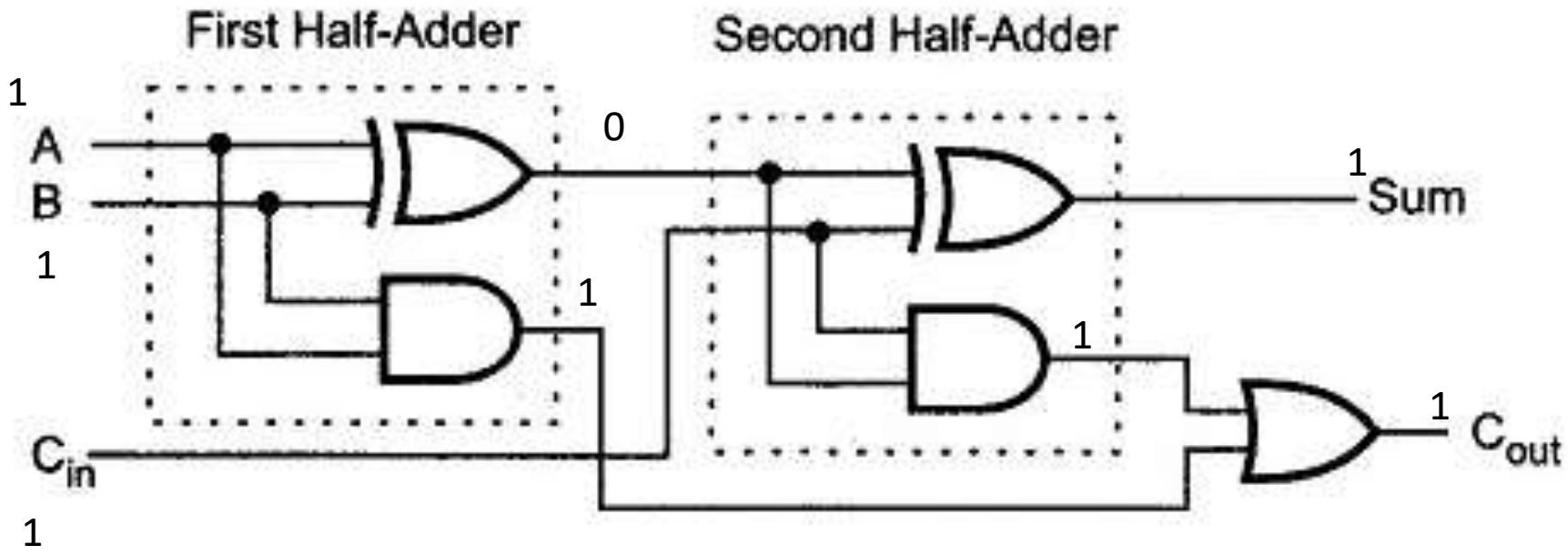
The **Boolean Expression** for sum can be further simplified as follows :

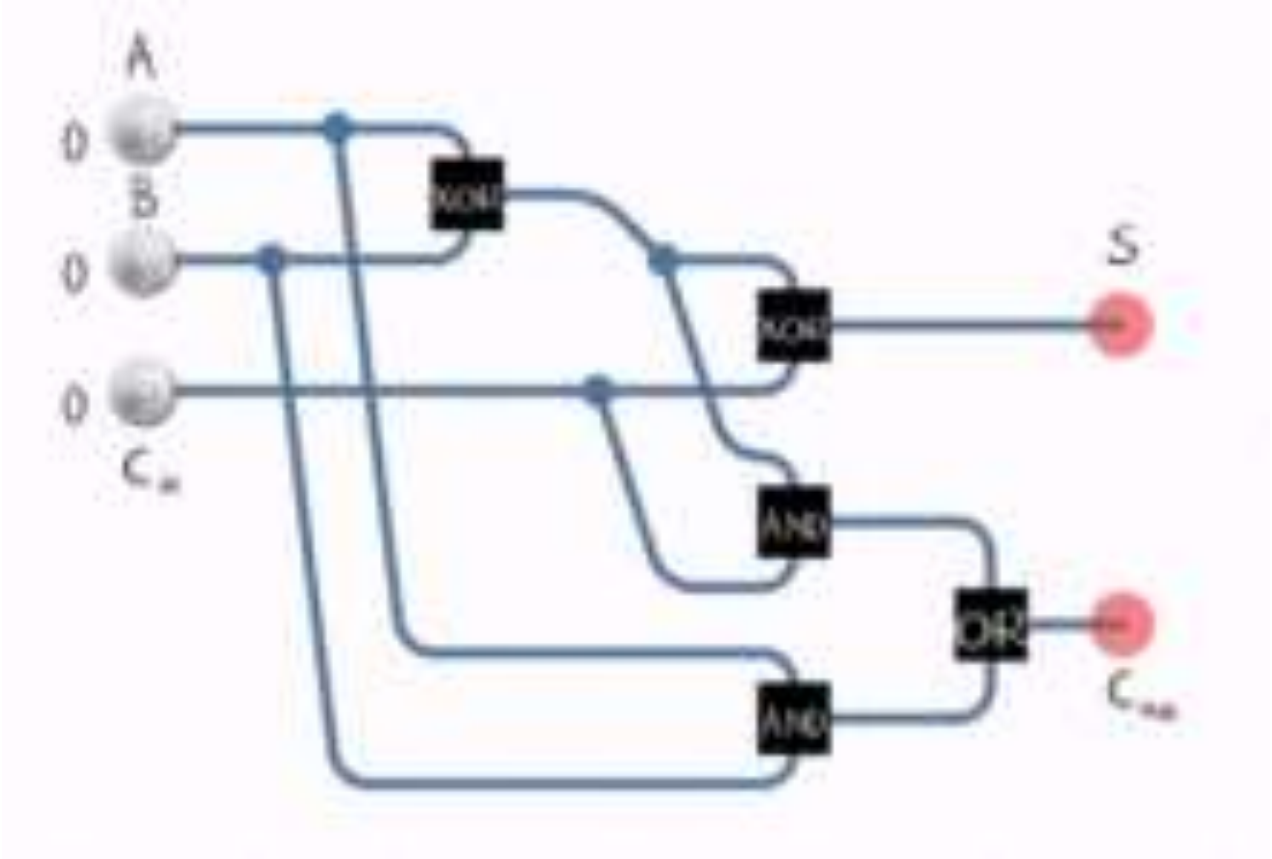
$$\begin{aligned}
\text{Sum} &= \overline{A}\ \overline{B}\ C_{in} + \overline{A}\ B\ \overline{C}_{in} + A\ \overline{B}\ \overline{C}_{in} + A\ B\ C_{in} \\
&= C_{in}\ (\overline{A}\ \overline{B} + AB) + \overline{C}_{in}\ (\overline{A}\ B + A\ \overline{B}) \\
&= C_{in}\ (A \odot B) + \overline{C}_{in}\ (A \oplus B) \\
&= C_{in}\ (\overline{A \oplus B}) + \overline{C}_{in}\ (A \oplus B) \\
&= C_{in} \oplus (A \oplus B)
\end{aligned}$$

Full adder Circuit Diagram

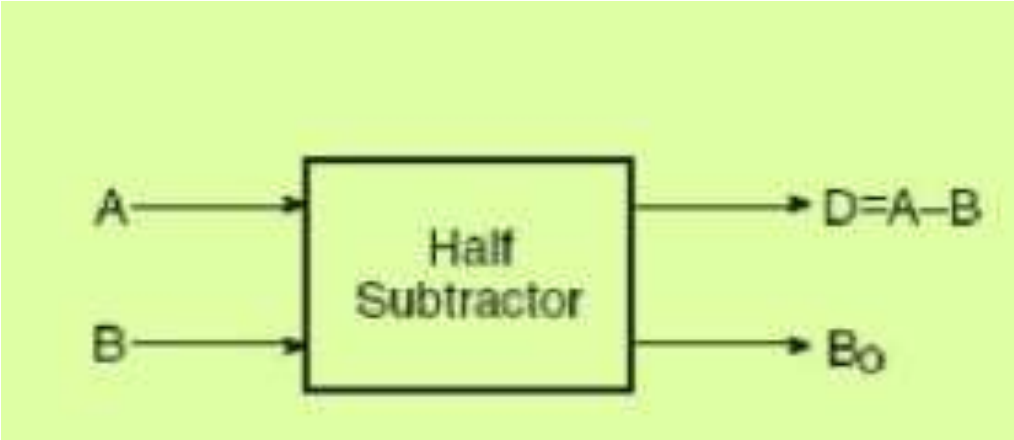


Full adder using 2 half adders




A
0
B
0
0
S
AND
OR

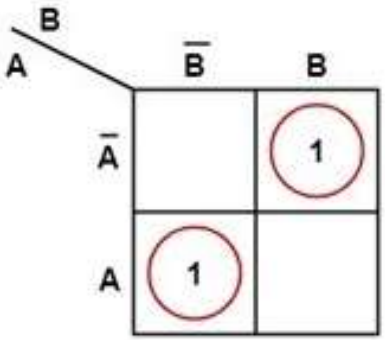
Half Subtractor



A	B	D	B_O
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

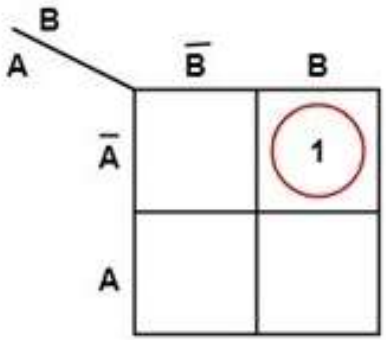
K map

For D:



$D = A \oplus B$

For b:



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

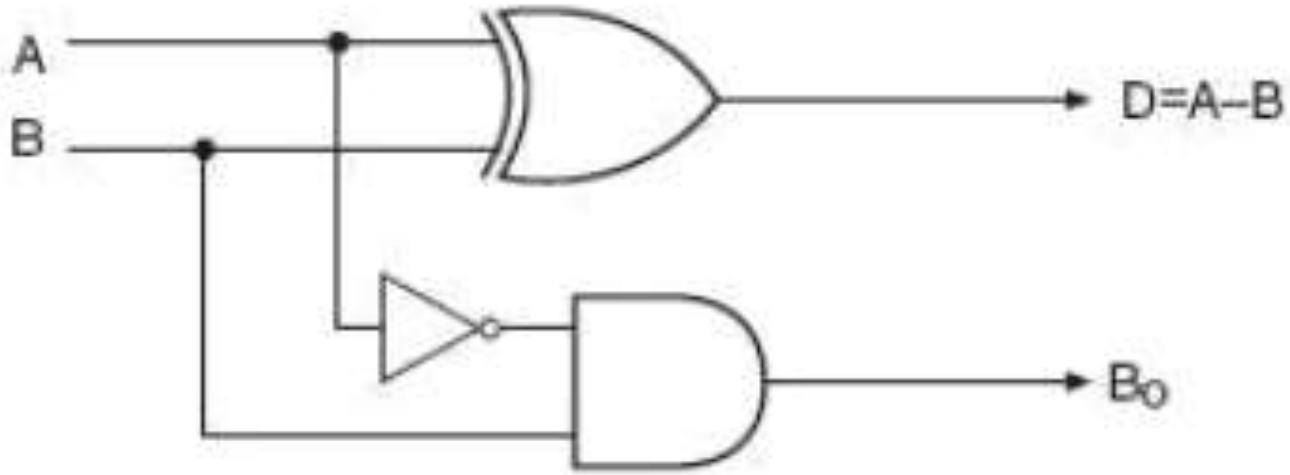
$$D = \overline{A}.B + A.\overline{B}$$

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

Half Subtractor

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

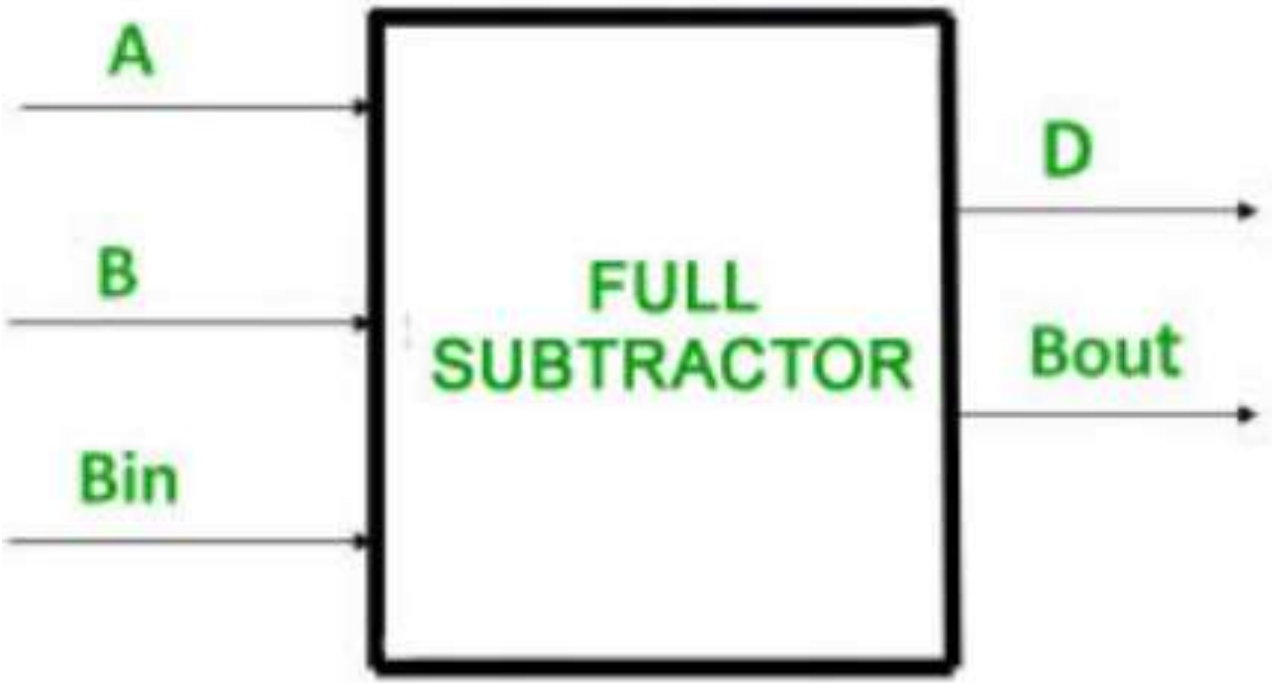
$$B_o = \overline{A}.B$$



Full Subtractor

- The Half Subtractor is used to subtract only two numbers. To overcome this problem, a full subtractor was designed. The full subtractor is used to subtract three 1-bit numbers A, B, and C, which are minuend, subtrahend, and borrow, respectively. The full subtractor has three input states and two output states i.e., difference and borrow.

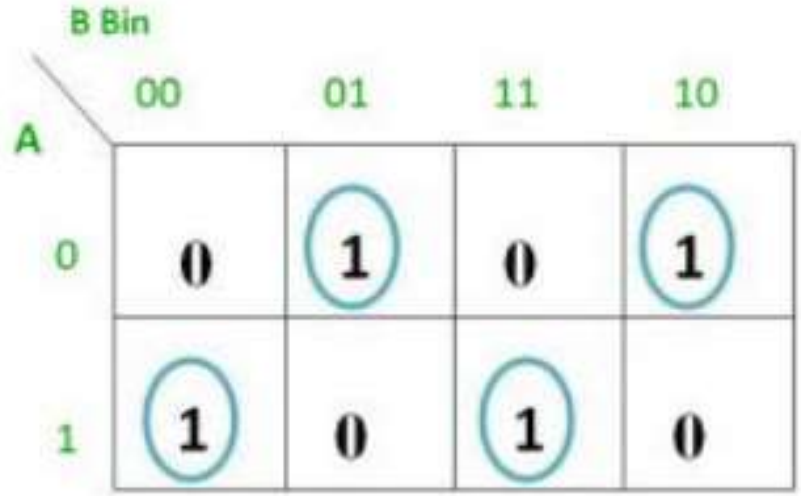
Full Subtractor



Truth Table

INPUT			OUTPUT	
A	B	Bin	D	Bout
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



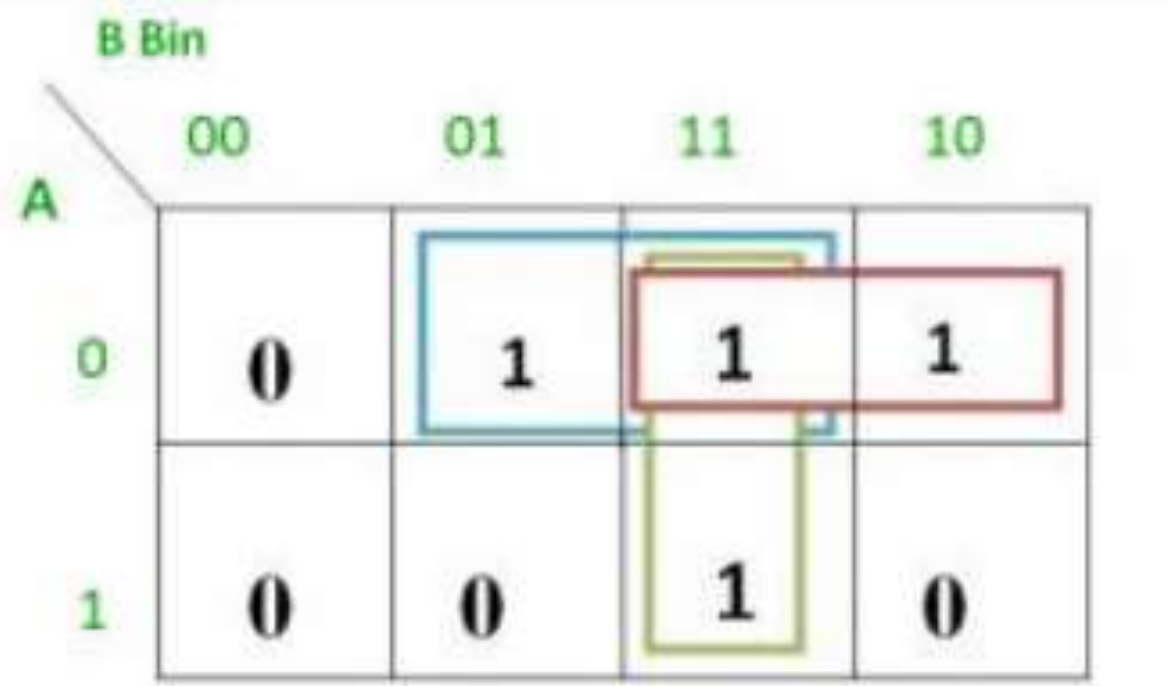
D=A'B'Bin + AB'Bin' + A'BBin' + ABBin

Bin Y'+Bin' Y= Bin XOR Y= Bin XOR A XOR B

$$
\begin{aligned}
D &= A'B'Bin + A'BBin' + AB'Bin' + ABBin \\
&= Bin(A'B' + AB) + Bin'(AB' + A'B) \\
&= Bin(\ A\ XNOR\ B) + Bin'(A\ XOR\ B) \\
&= Bin\ (A\ XOR\ B)' + Bin'(A\ XOR\ B) \\
&= Bin\ XOR\ (A\ XOR\ B) \\
&= (A\ XOR\ B)\ XOR\ Bin
\end{aligned}
$$

$$D = A \oplus B \oplus B_{in}$$

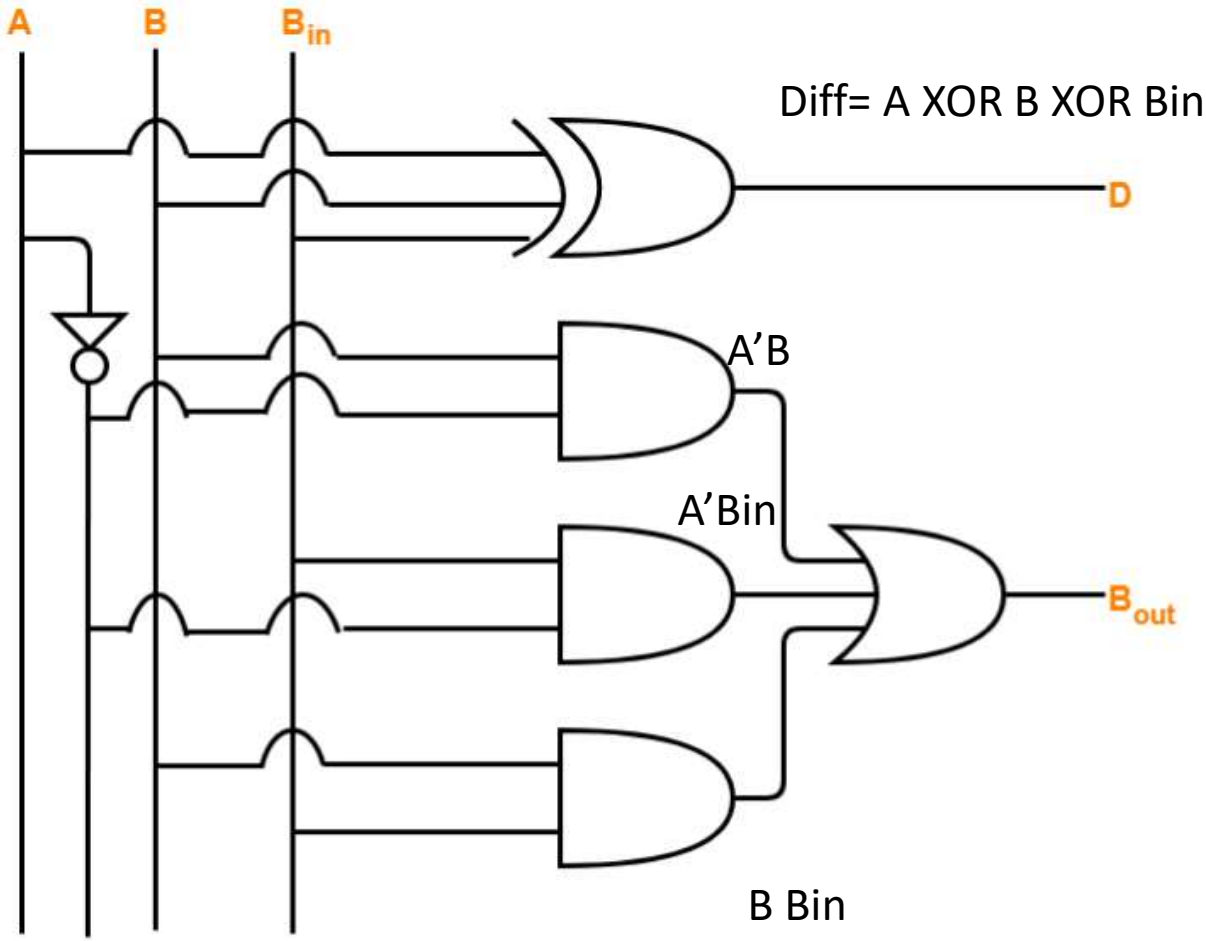
K map for borrow



$$B_{out} = A'Bin + A'B + BBin$$

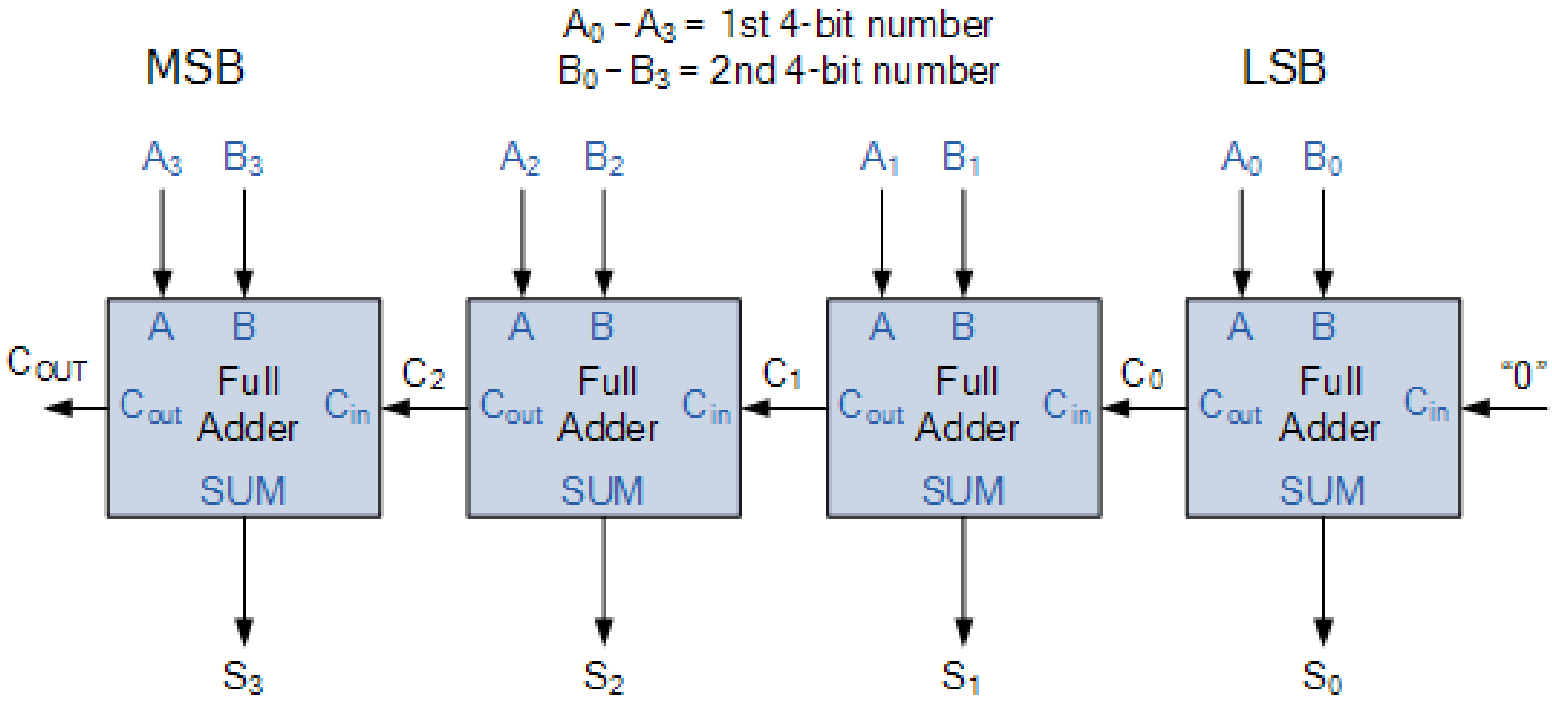
$$B_{out} = \overline{A}\,B + (\overline{A} + B)\,B_{in}$$

Full Subtractor



4 bit parallel adder

A 4-bit Ripple Carry Adder



A = 1 1 0 1

B= 1 0 0 1

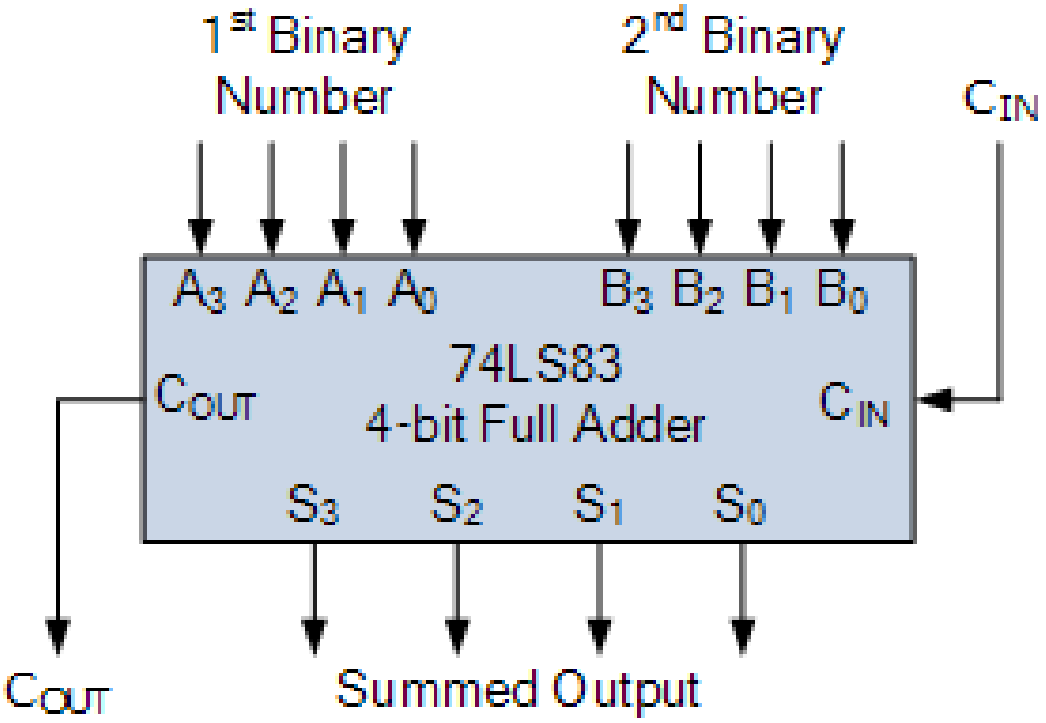
Sum= 1 0110

S3=0 Cout=1

S2 = 1 C2=0

S1= 1 C1=0

S0 =0 C0=1


1st Binary Number
2nd Binary Number
CIN
A3 A2 A1 A0
B3 B2 B1 B0
74LS83
4-bit Full Adder
COUT
CIN
S3 S2 S1 S0
COUT
Summed Output

BCD Numbers

BCD or **Binary Coded Decimal** is that number system or code which has the binary numbers or digits to represent a decimal number.
A decimal number contains 10 digits (0-9). Now the equivalent binary numbers can be found out of these 10 decimal numbers. In case of **BCD** the binary number formed by four binary digits, will be the equivalent code for the given decimal digits. In **BCD** we can use the binary number from 0000-1001 only, which are the decimal equivalent from 0-9 respectively. Suppose if a number have single decimal digit then it's equivalent **Binary Coded Decimal** will be the respective four binary digits of that decimal number and if the number contains two decimal digits then it's equivalent **BCD** will be the respective eight binary of the given decimal number. Four for the first decimal digit and next four for the second decimal digit.

Let, $(12)_{10}$ be the decimal number whose equivalent **Binary coded decimal** will be 0001 0010. Four bits from L.S.B is binary equivalent of 2 and next four is the binary equivalent of 1.
Table given below shows the binary and **BCD** codes for the decimal numbers 0 to 15.
From the table below, we can conclude that after 9 the decimal equivalent binary number is of four bit but in case of BCD it is an eight bit number. This is the main difference between Binary number and binary coded decimal. For 0 to 9 decimal numbers both binary and BCD is equal but when decimal number is more than one bit BCD differs from binary.

Decimal number	Binary number	Binary Coded Decimal(BCD)
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111
8	1000	1000
9	1001	1001

Decimal number	Binary number	Binary Coded Decimal(BCD)
10	1010	0001 0000
11	1011	0001 0001
12	1100	0001 0010
13	1101	0001 0011
14	1110	0001 0100
15	1111	0001 0101

Example 1

But the result of addition here is less than 9, which is valid for BCD numbers.

$$\begin{array}{r} 0001 \\ +\ 0101 \\ \hline 0110 \end{array}$$

Example 2

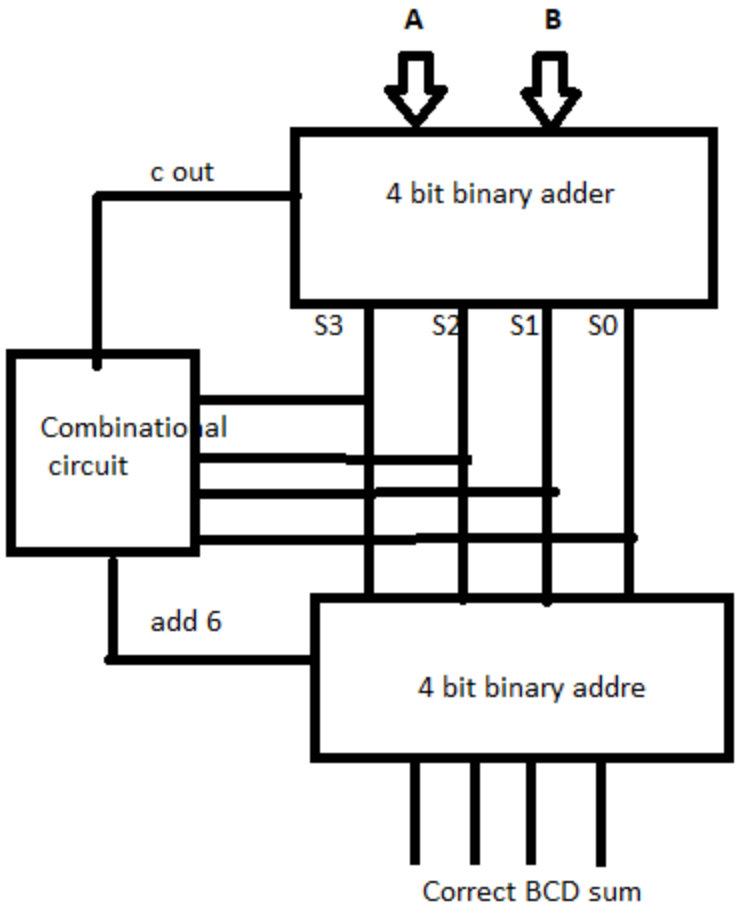
If the four bit result of addition is greater than 9 and if a carry bit is present in the result then it is invalid and we have to add 6 whose binary equivalent is $(0110)_2$ to the result of addition. Then the resultant that we would get will be a valid binary coded number. In case 2 the result was $(1111)_2$, which is greater than 9 so we have to add 6 or $(0110)_2$ to it.

$$\begin{array}{r} 1010 \\ +\ 0101 \\ \hline 1111 \end{array}$$

$$\begin{array}{r} 1111 \\ +\ 0110 \\ \hline 0001\ 0101 \\ 1 \quad\ 5 \end{array}$$

BCD ADDER

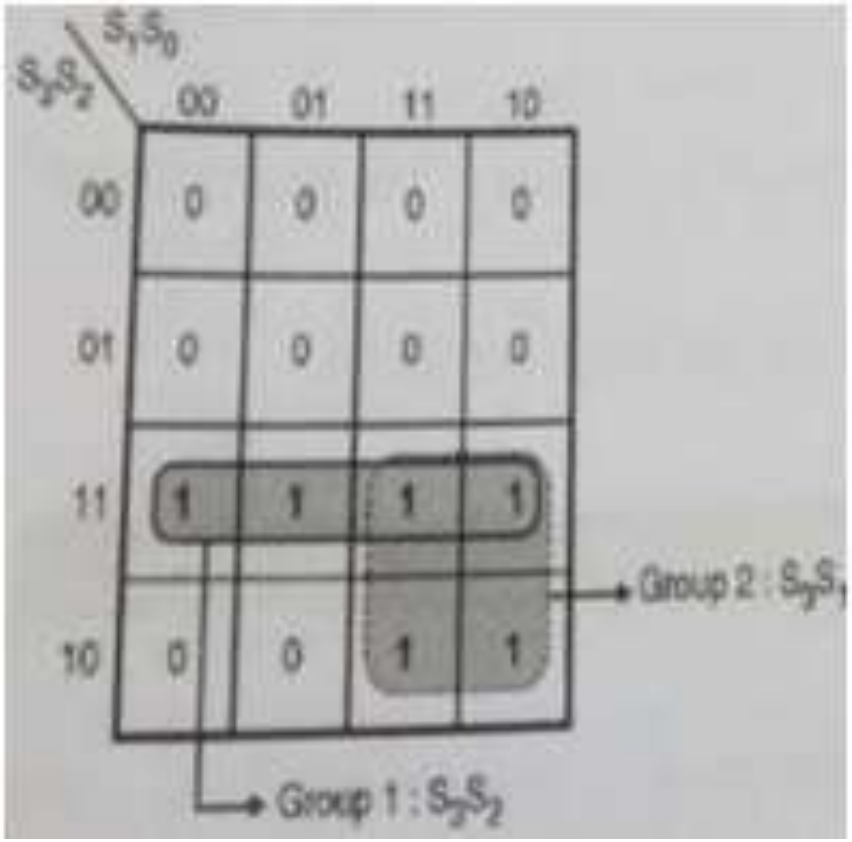
- A BCD adder adds two BCD digits and produces output as a BCD digit. A BCD or Binary Coded Decimal digit cannot be greater than 9.
- The two BCD digits are to be added using the rules of binary addition. If sum is less than or equal to 9 and carry is 0, then no correction is needed. The sum is correct and in true BCD form.
- But if sum is greater than 9 or carry =1, the result is wrong and correction must be done. The wrong result can be corrected adding six (0110) to it.
- For implementing a BCD adder using a binary adder circuit IC 7483, additional combinational circuit will be required, where the Sum output $S_3 - S_0$ is checked for invalid values from 10 to 15. The truth table and K-map for the same is as shown:


A
B
c out
4 bit binary adder
S3
S2
S1
S0
Combinational
circuit
add 6
4 bit binary addre
Correct BCD sum

TRUTH TABLE

I/P				O/P
S3	S2	S1	S0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

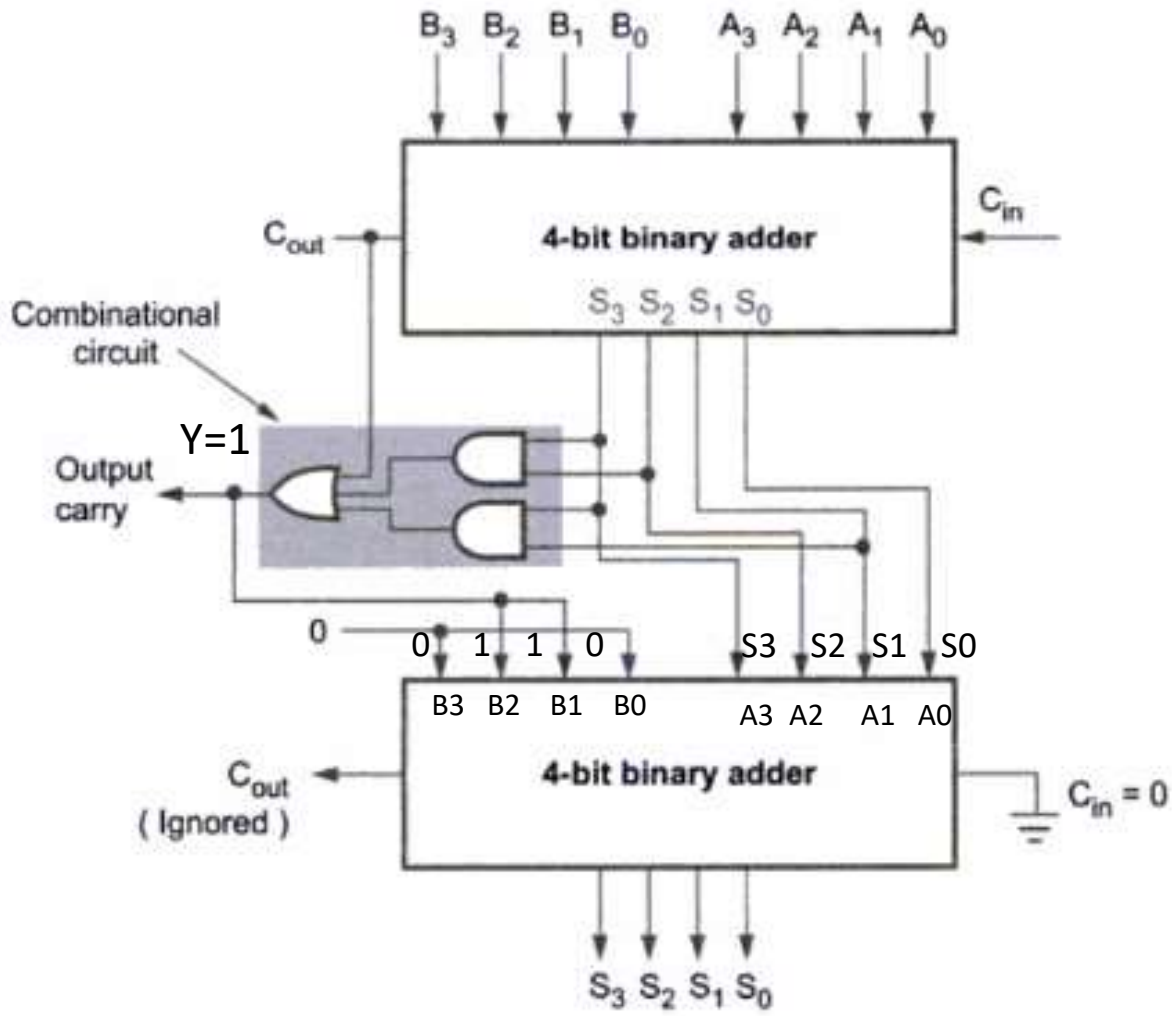
K map



The Boolean expression is, $Y = S_3S_2 + S_3S_1$

- The BCD adder is shown below. The output of the combinational circuit should be 1 if Cout of adder-1 is high. Therefore Y is ORed with Cout of adder 1.
- The output of combinational circuit is connected to B1B2 inputs of adder-2 and $B_3 = B_0 = 0$ as they are connected to ground permanently. This makes $B_3B_2B_1B_0 = 0110$ if Y' = 1.
- The sum outputs of adder-1 are applied to $A_3A_2A_1A_0$ of adder-2. The output of combinational circuit is to be used as final output carry and the carry output of adder-2 is to be ignored.

4 bit BCD ADDER



1. As shown in the Fig, the two BCD numbers, together with input carry, are first added in the top 4-bit binary adder to produce a binary sum.

2. When the output carry is equal to zero (i.e. when sum <= 9 and C_out = 0) nothing (zero) is added to the binary sum.

   When it is equal to one (i.e. when sum > 9 or C_out = 1), binary 0110 is added to the binary sum through the bottom 4-bit binary adder.

3. The output carry generated from the bottom binary adder can be ignored, since it supplies information already available at the output carry terminal.

Case1: sum <= 9 and carry=0

Case 2: sum > 9 and carry=0

Case 3: sum <= 9 but carry=1


thank
YOU

Code Converters

Gauri Rao

Binary To Gray Code Converter

Gray Code system is a binary number system in which every successive pair of numbers differs in only one bit. It is used in applications in which the normal sequence of binary numbers generated by the hardware may produce an error or ambiguity during the transition from one number to the next.

For example, the states of a system may change from 3(011) to 4(100) as- 011 — 001 — 101 — 100. Therefore there is a high chance of a wrong state being read while the system changes from the initial state to the final state.
This could have serious consequences for the machine using the information. The Gray code eliminates this problem since only one bit changes its value during any transition between two numbers.

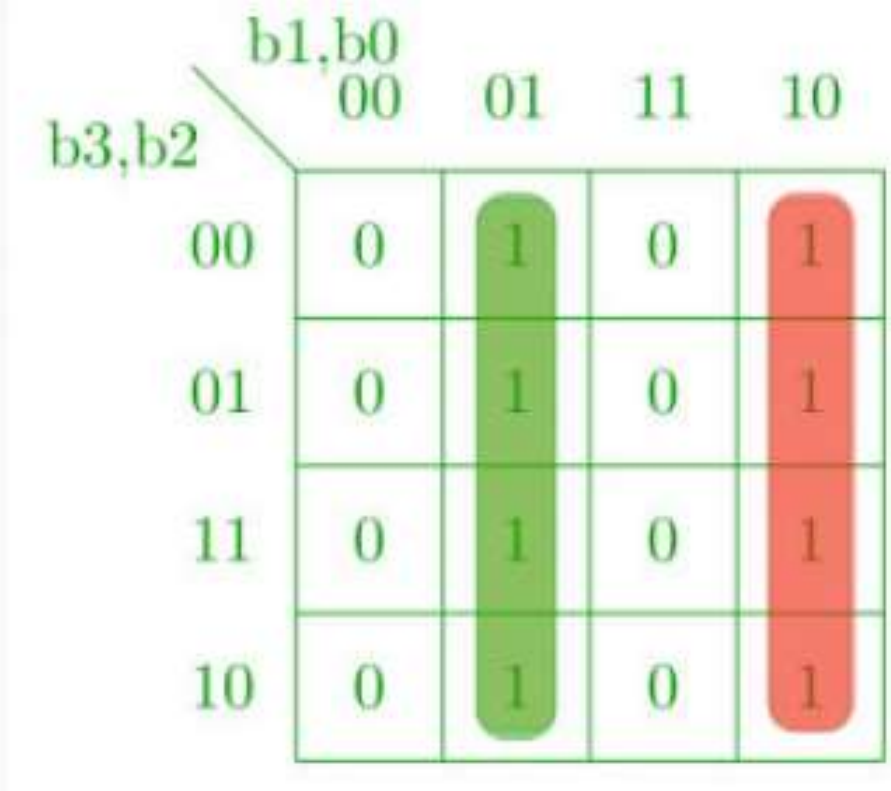
Binary to Gray Code converter Truth table

Binary				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
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

Let be the bits representing the binary numbers, where b0 is the LSB and b3 is the MSB, and
Let be the bits representing the gray code of the binary numbers, where g0 is the LSB and g3 is the MSB.

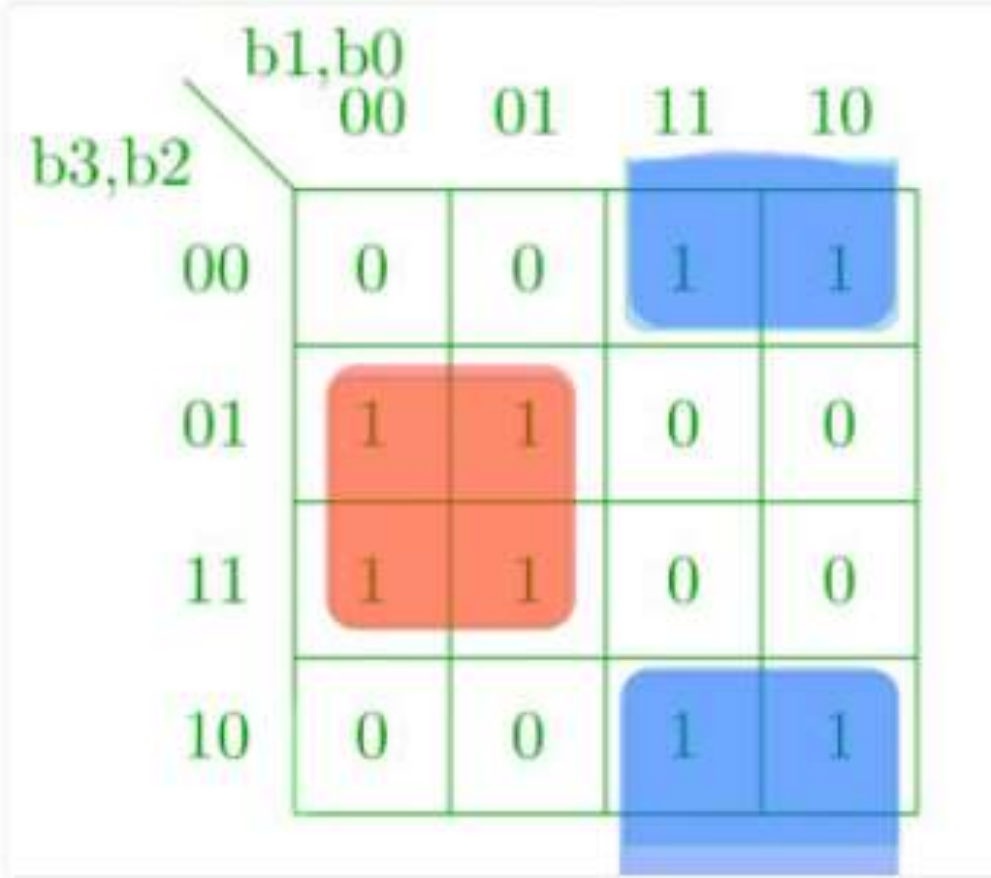
To find the corresponding digital circuit, we will use the K-Map technique for each of the gray code bits as output with all of the binary bits as input.

K map for g0



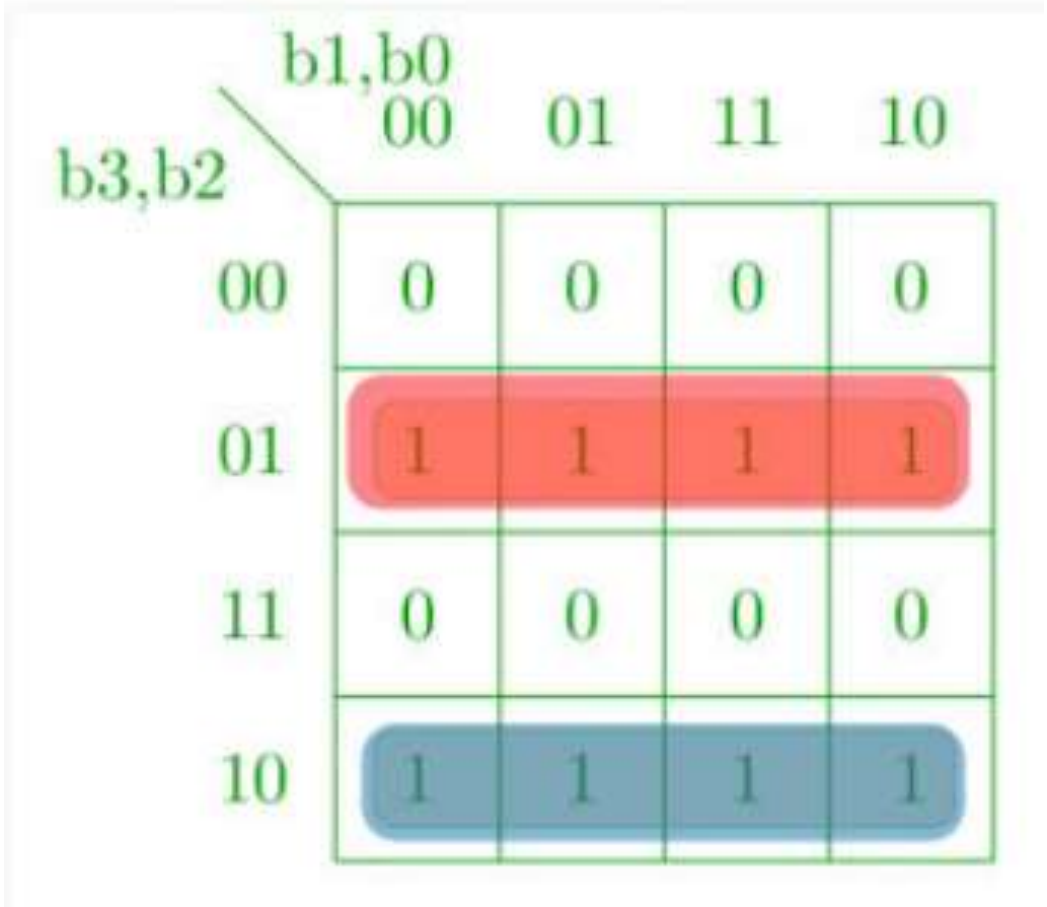
$$g_0 = b_0 b_1' + b_1 b_0' = b_0 \oplus b_1$$

K map for g1



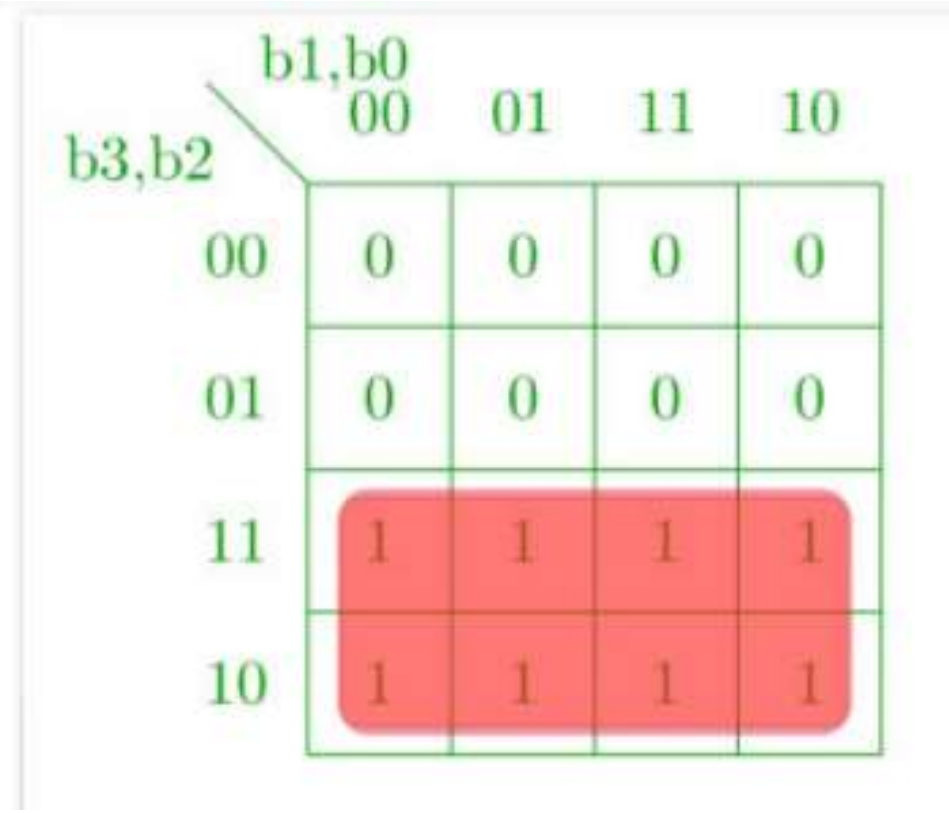
$$g_1 = b_2 b_1' + b_1 b_2' = b_1 \oplus b_2$$

K map for g2



$$g_2 = b_2 b_3' + b_3 b_2' = b_2 \oplus b_3$$

K map for g3



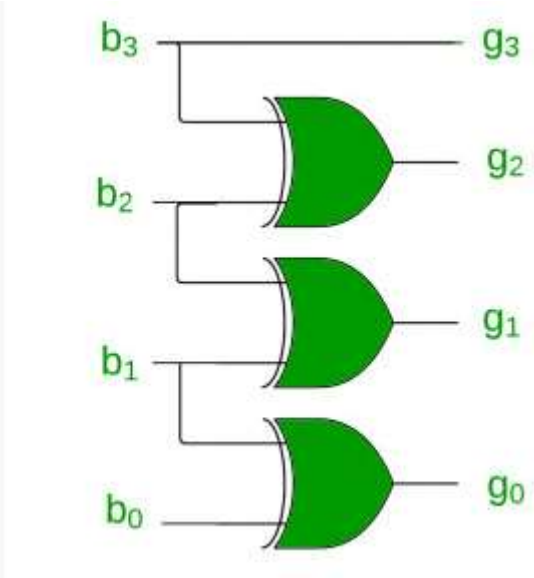
$$g_3 = b_3$$

Corresponding minimized boolean expressions for gray code bits

$$g_0 = b_0 b_1' + b_1 b_0' = b_0 \oplus b_1$$

$$g_1 = b_2 b_1' + b_1 b_2' = b_1 \oplus b_2$$
$$g_2 = b_2 b_3' + b_3 b_2' = b_2 \oplus b_3$$
$$g_3 = b_3$$

Code converter



BCD to excess-3 code converter

As is clear by the name, a BCD digit can be converted to it's corresponding Excess-3 code by simply adding 3 to it.

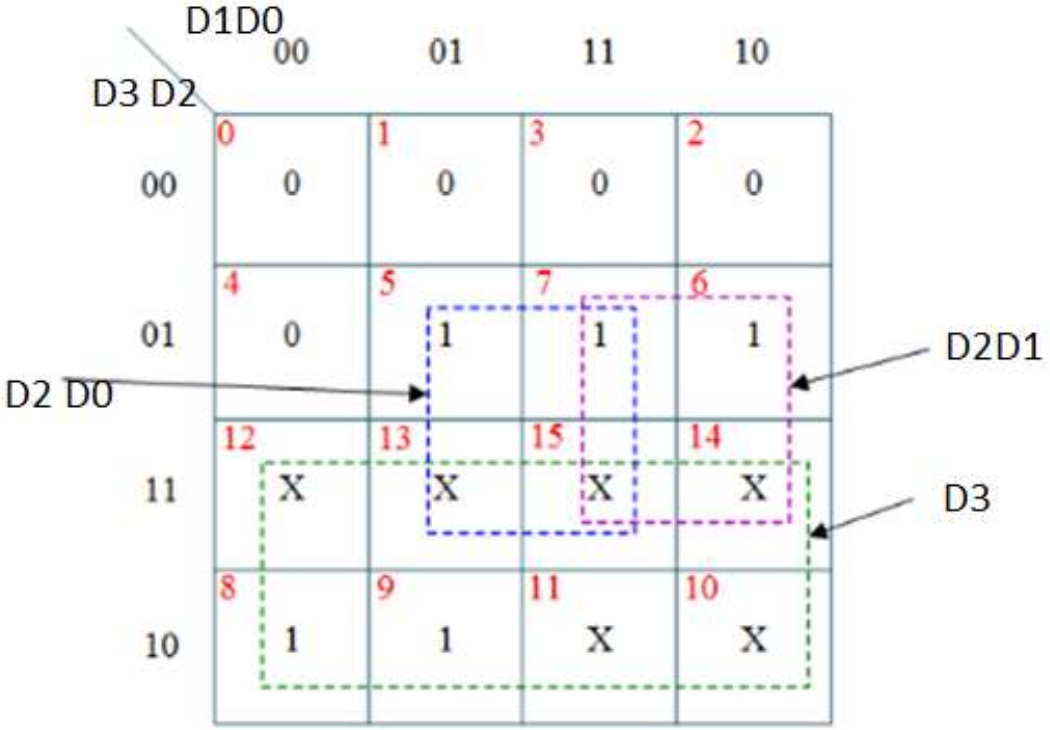
Let $A, B, C, and\ D$ be the bits representing the binary numbers, where D is the LSB and A is the MSB, and

Let $w, x, y, and\ z$ be the bits representing the gray code of the binary numbers, where z is the LSB and w is the MSB.

The truth table for the conversion is given below. The X's mark don't care conditions.

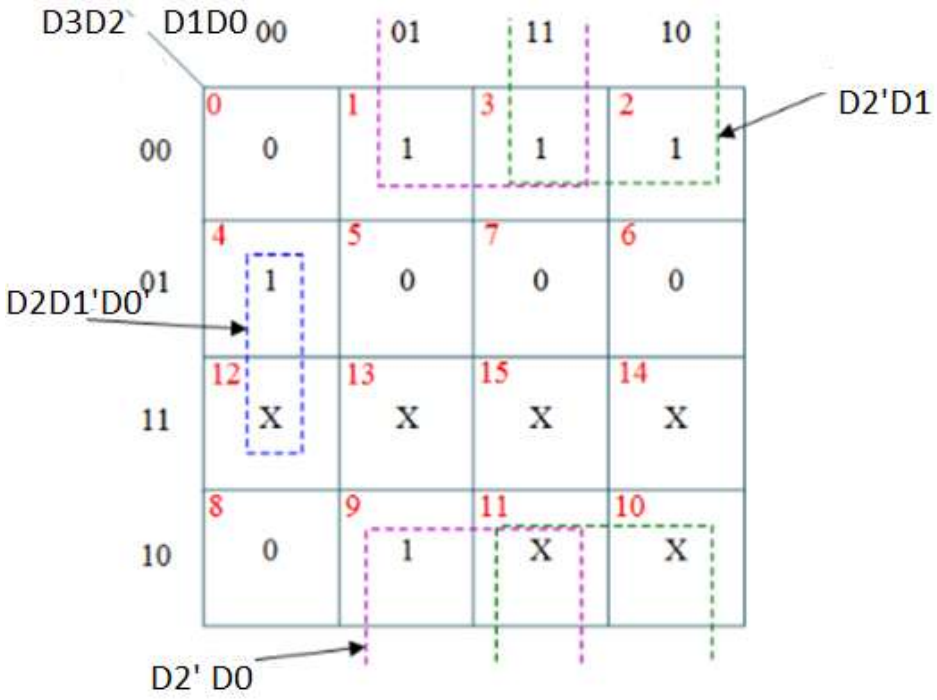
BCD(8421)				Excess-3			
D3	D2	D1	D0	E3	E2	E1	E0
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

K map for E3



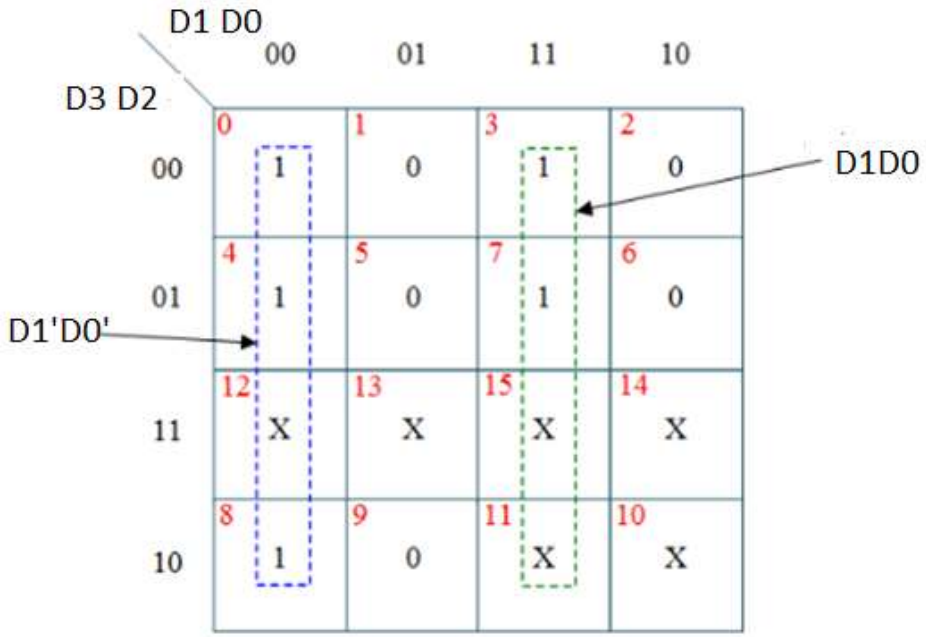
E3= D3+ D2D0+ D2D1

K map for E2



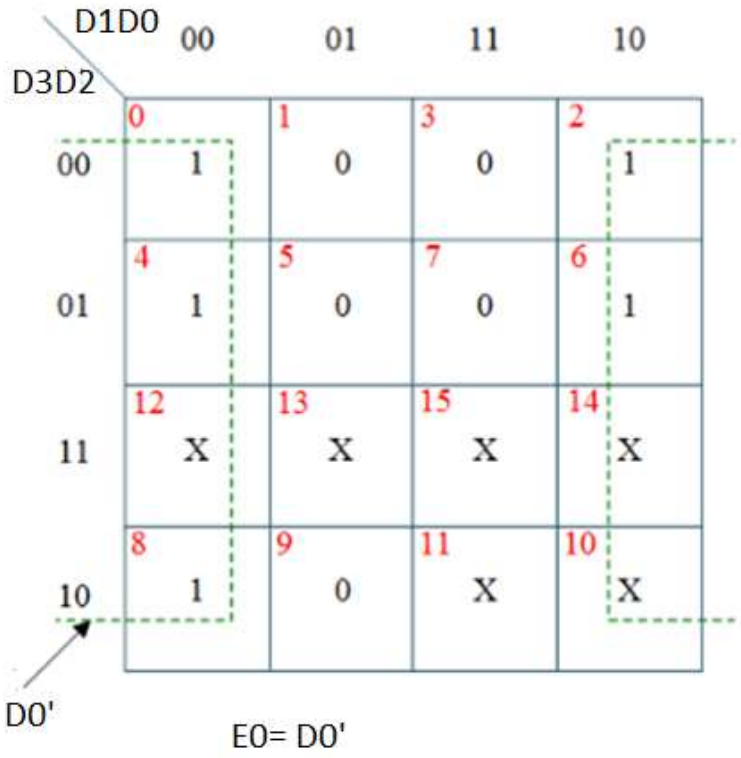
E2= D2'D1+D2'D0
+D2D1'D0'

K map for E1



E1=D1D0+D1'Do'

K map for z


D1D0
00
01
11
10
D3D2
00
01
11
10
0
1
3
2
4
5
7
6
12
13
15
14
8
9
11
10
D0'
E0= D0'

Minimized expressions

$E3 = D3 + D2D0 + D2D1$

$E2 = D2'D1 + D2'D0 + D2D1'D0'$

$E1 = D1D0 + D1'Do'$

$E0 = D0'$

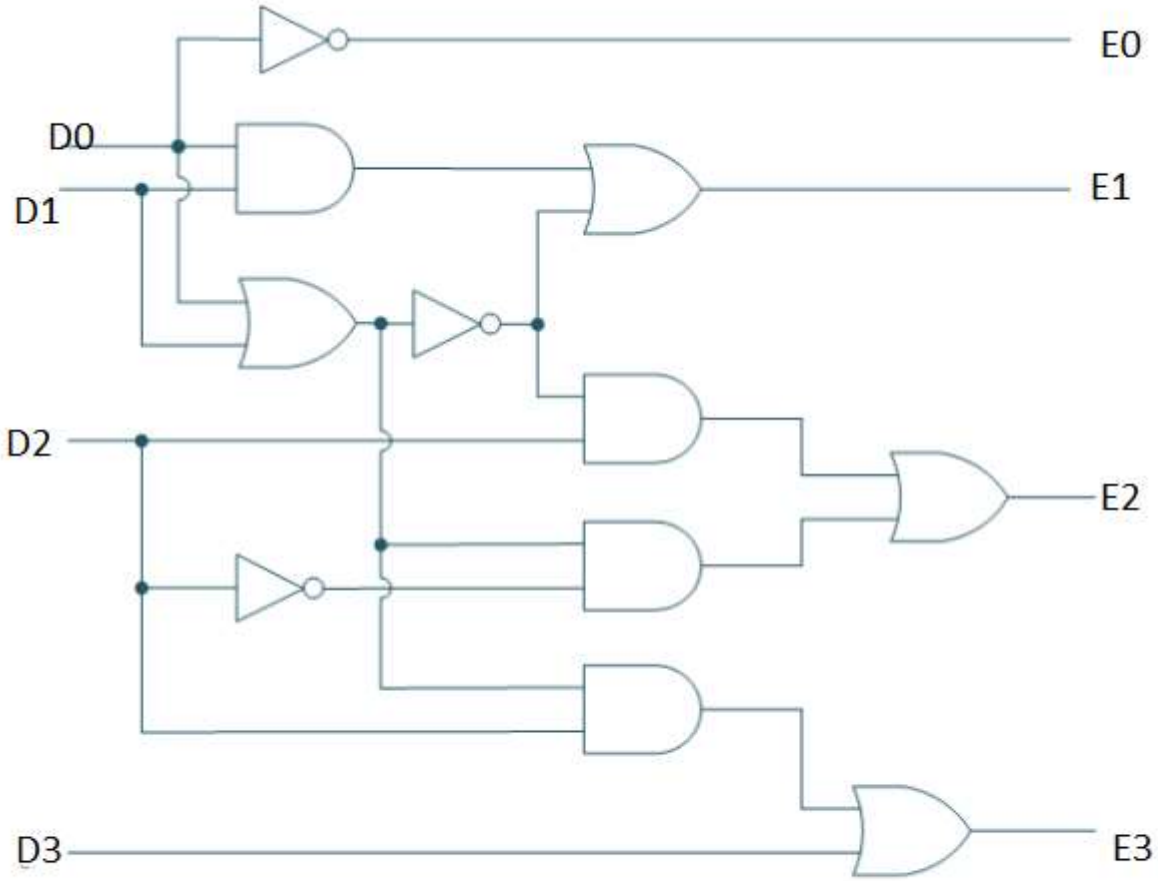
E3= D3+D2(D0+D1)

E2=D2’(D1+D0)+ D2D1’D0’

E1= D0 EXNOR D1

E0= D0’

BCD to Excess-3 code converter



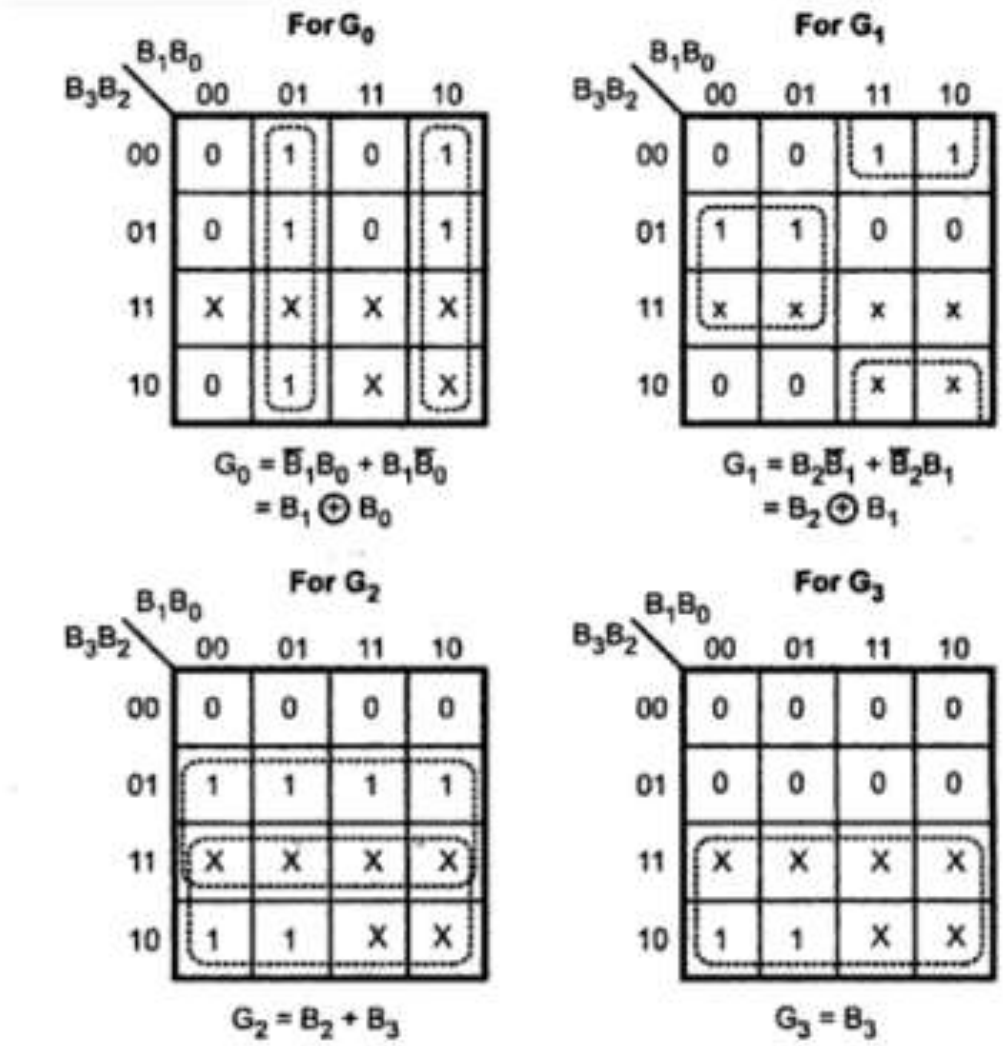
Practice Problem

BCD to Gray code converter

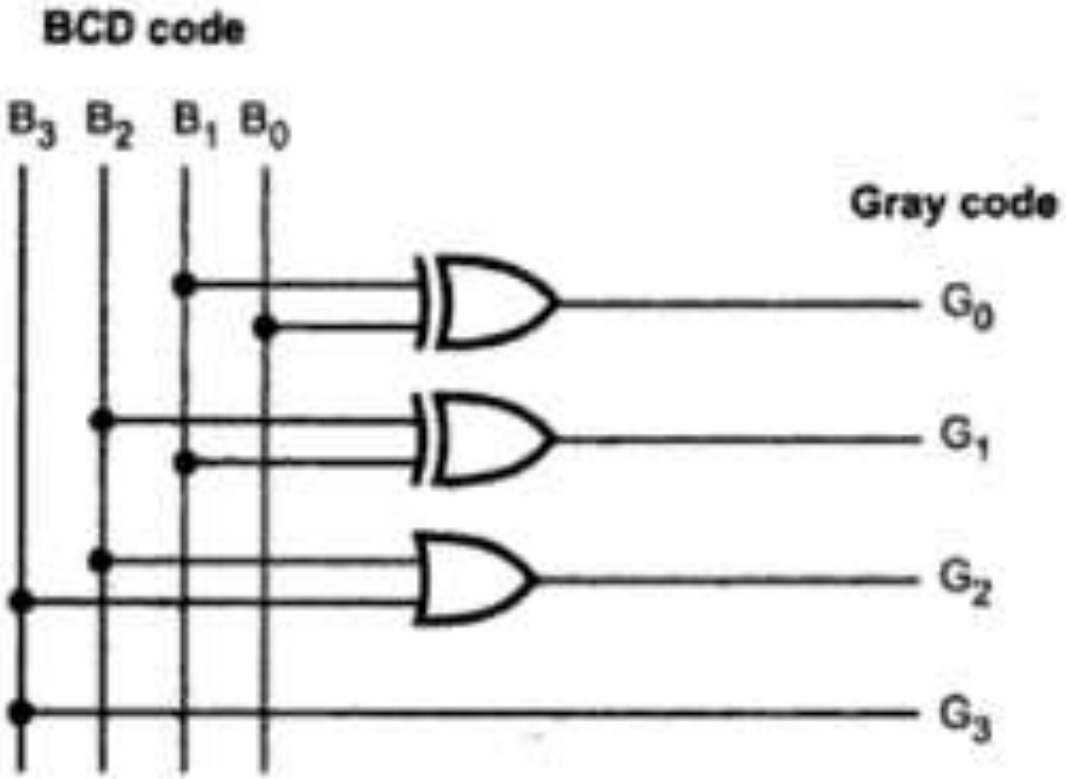
Truth table

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

K map simplification



Circuit diagram

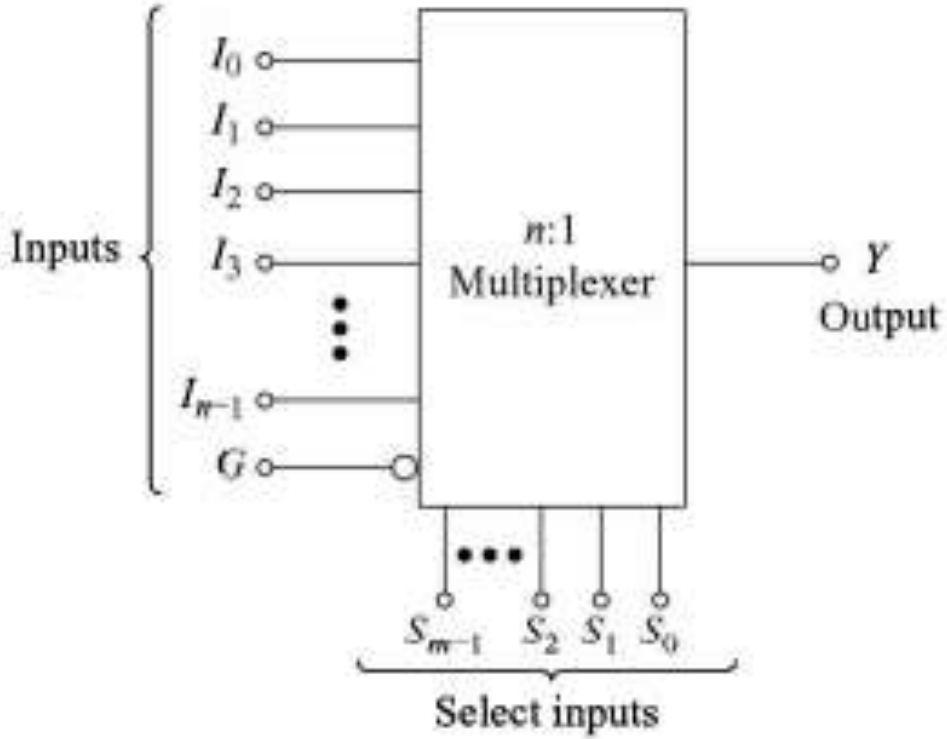


Thank you

Multiplexers

Gauri Rao

Block diagram



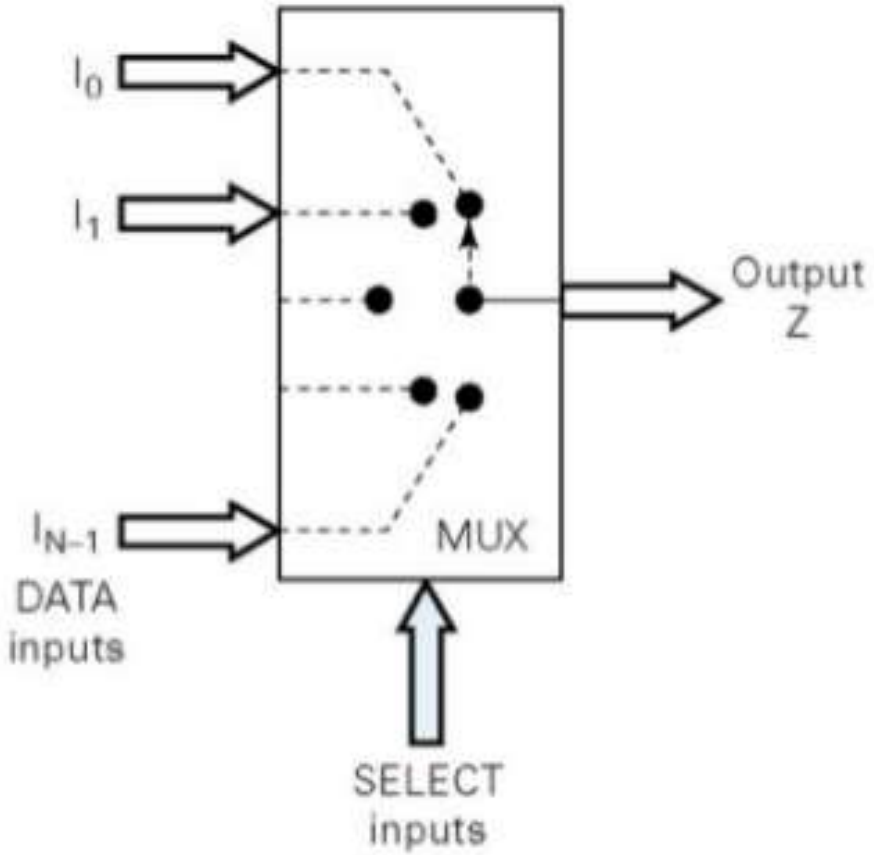
- Definition : A multiplexers (MUX) is a device that allows digital information from several sources to be routed onto a single line for transmission over that line to a common destination.

- Several data input lines
- Some select line (less than the no. of input lines)
- Single output line

- If there are n data input lines and m select lines, then

$$2^m = n$$

Data selector

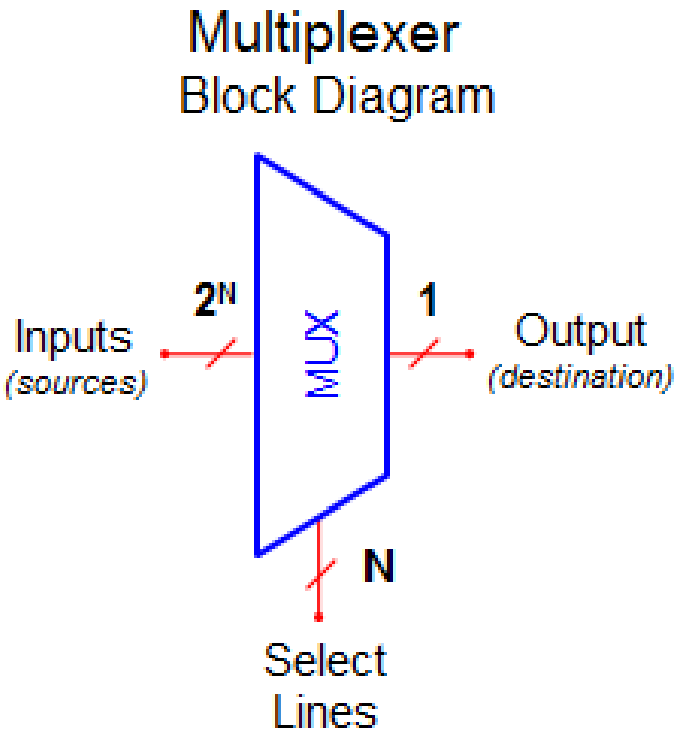


Multiplexer

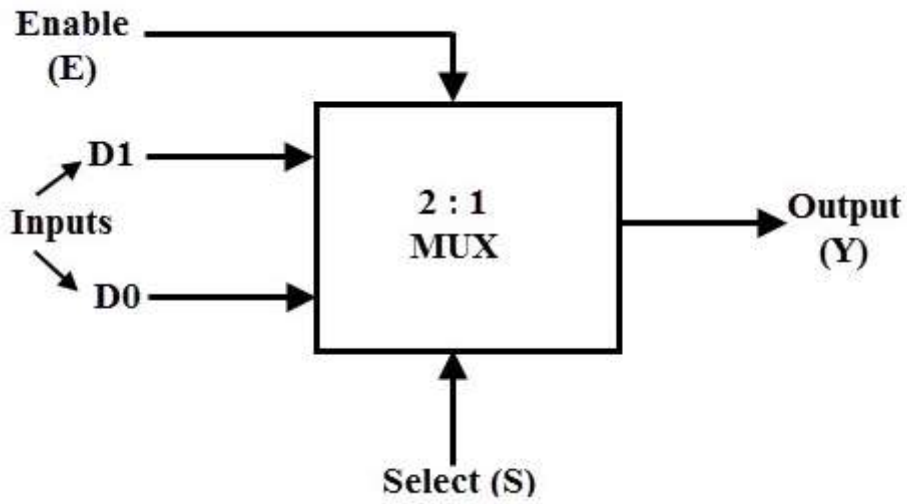
- Also called data selectors.
- Basic function: select one of its 2^n data input lines and place the corresponding information onto a single output line.
- n input bits needed to specify which input line is to be selected.
  - Place binary code for a desired data input line onto its n select input lines.

Multiplexer

- A MUX is a digital switch that has multiple inputs (sources) and a single output (destination).
- The select lines determine which input is connected to the output.
- MUX Types
  - → 2-to-1 (1 select line)
  - → 4-to-1 (2 select lines)
  - → 8-to-1 (3 select lines)
  - → 16-to-1 (4 select lines)



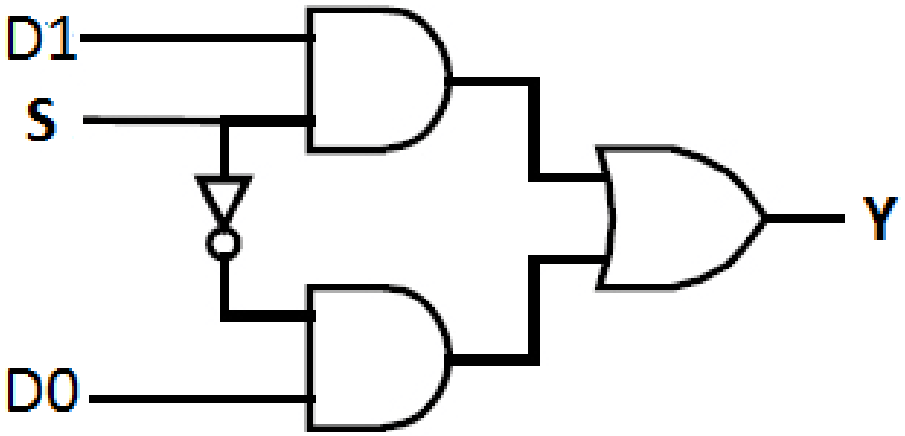
2:1 multiplexer



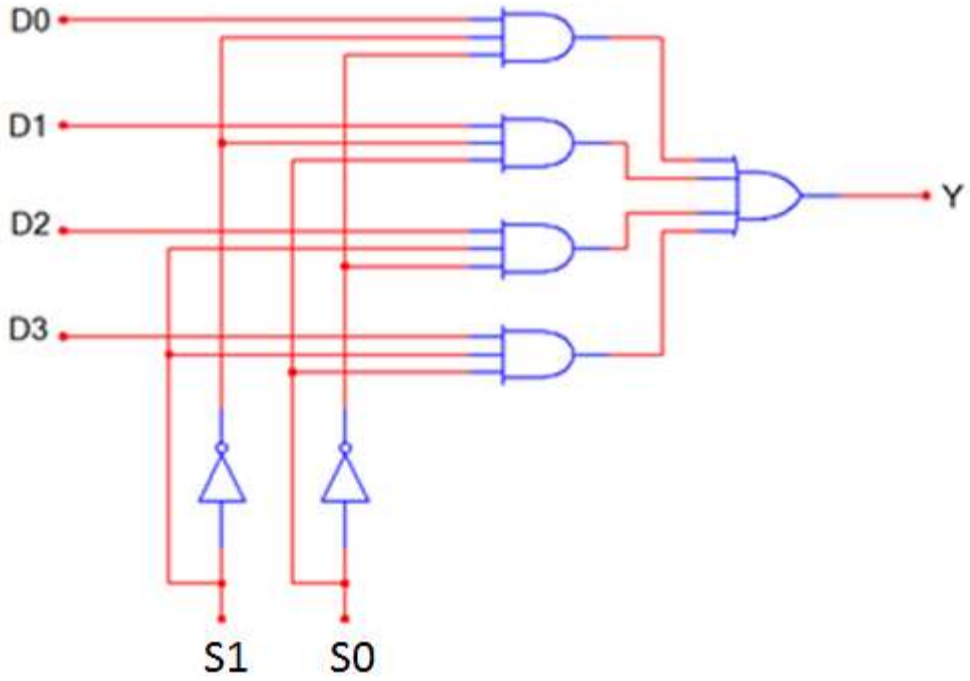
2:1 MUX

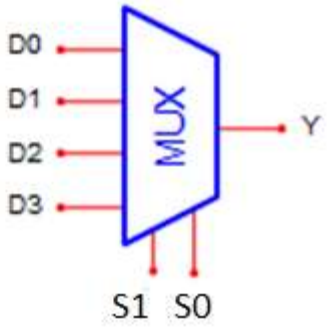
S	D0	D1	Y
0	0	X	0
0	1	X	1
1	X	0	0
1	X	1	1

$$Y = \bar{S}D0 + SD1$$



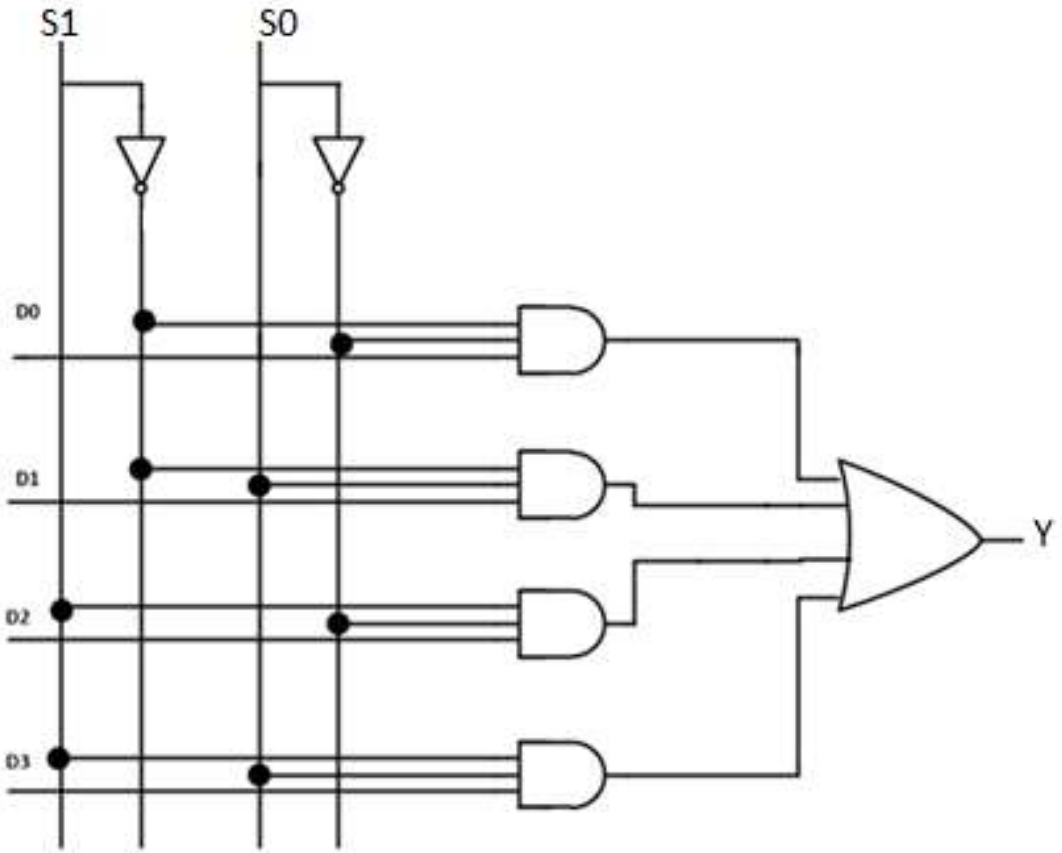
4:1 Multiplexer



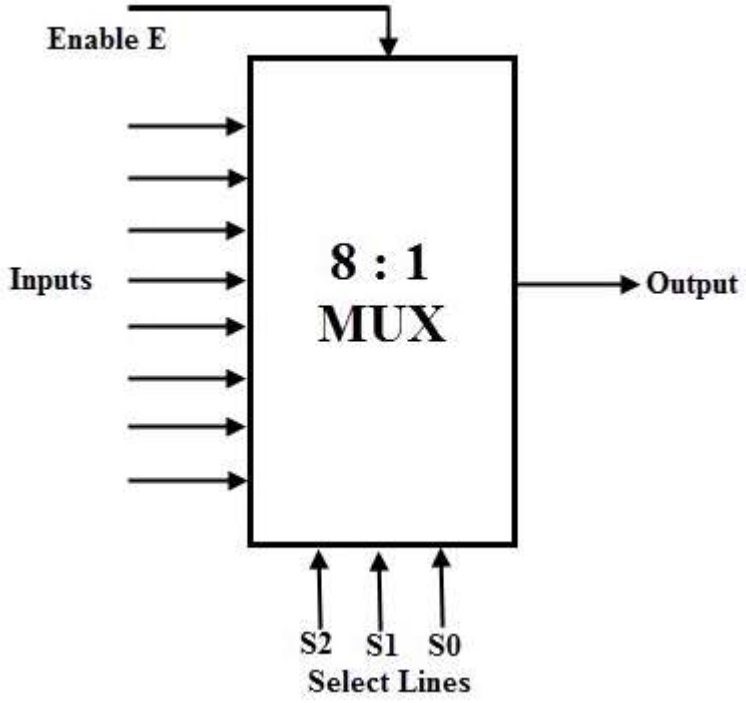


S1	S0	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3

4:1 MUX

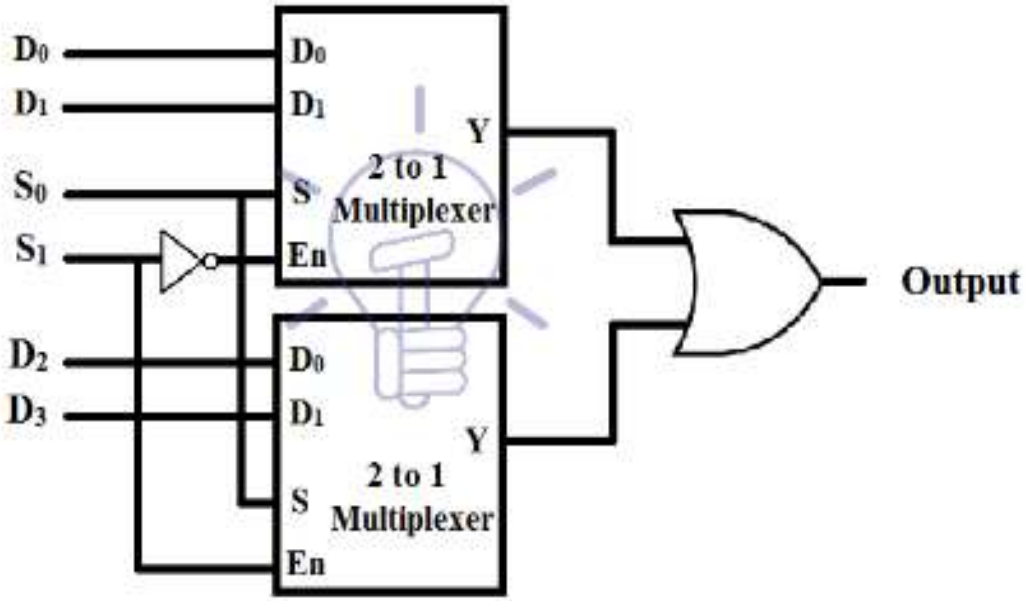


8:1 MUX



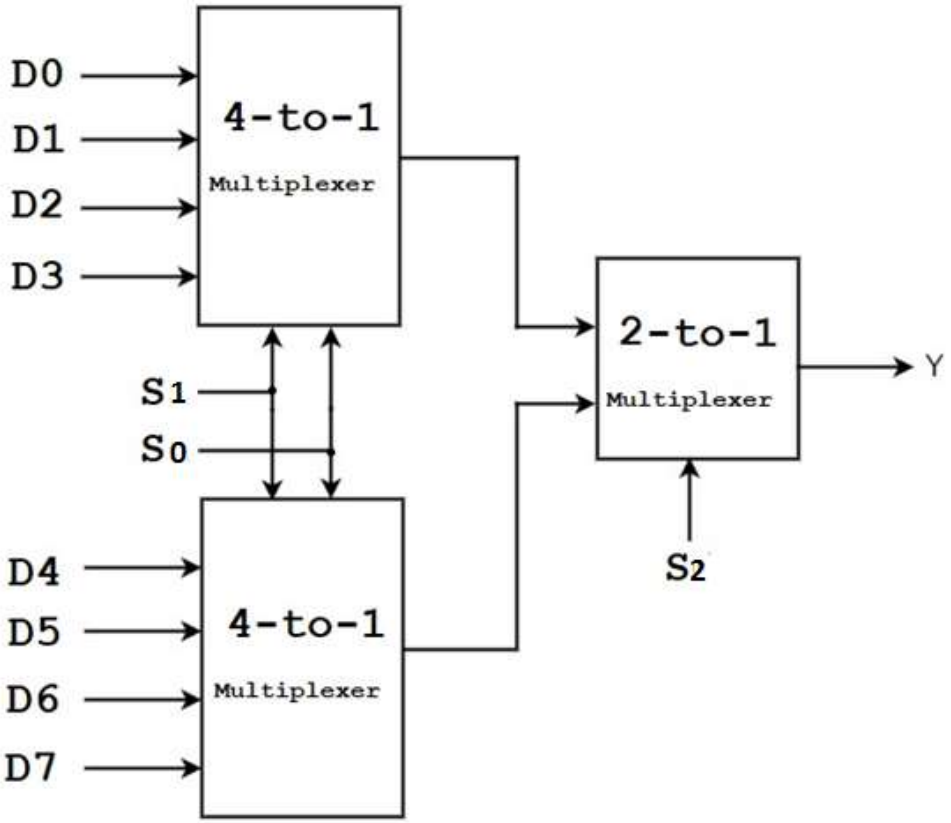
Select Data Inputs			Output
S_2	S_1	S_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

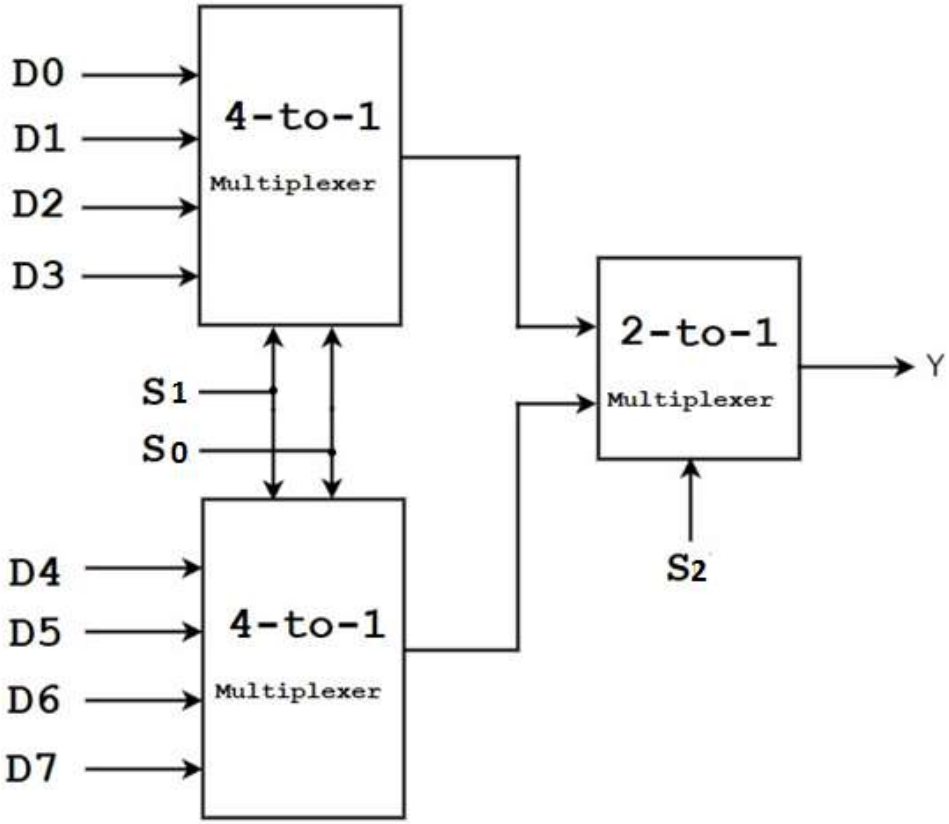
4:1 MUX USING 2:1 MUX



Select Data Inputs		Output
S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

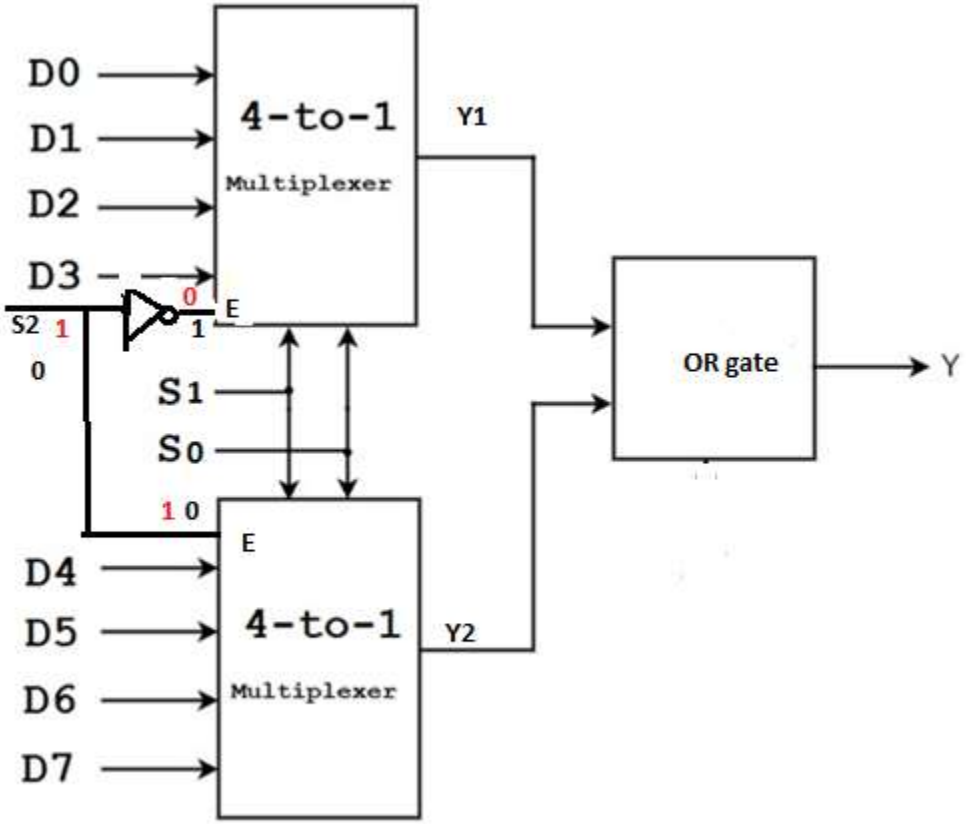
8:1 MUX


D0
D1
D2
D3
4-to-1
Multiplexer
S1
S0
D4
D5
D6
D7
4-to-1
Multiplexer
2-to-1
Multiplexer
Y
S2

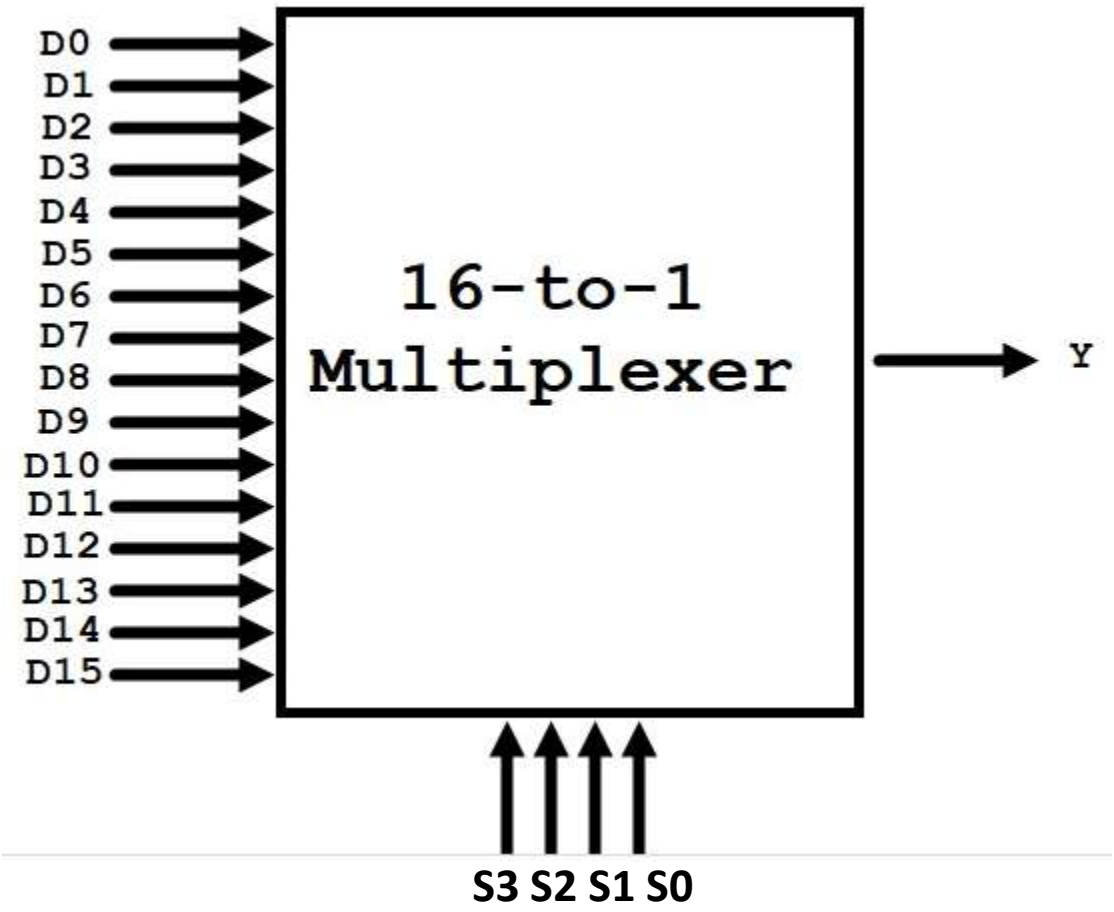


Select Data Inputs			Output
S_2	S_1	S_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

8:1 MUX using 4:1 MUX only



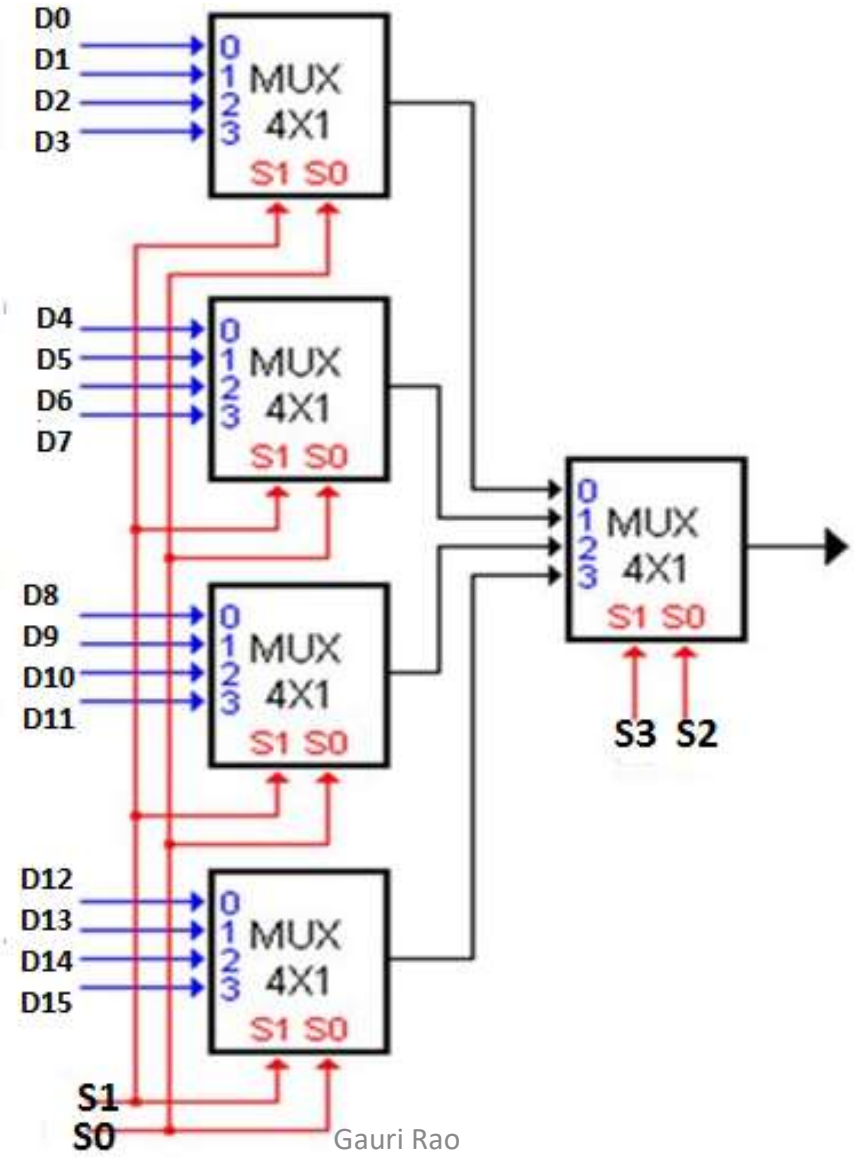
16:1 MUX


D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
D14
D15
16-to-1
Multiplexer
Y
S3 S2 S1 S0

S3	S2	S1	S0	Y
0	0	0	0	D0
0	0	0	1	D1
0	0	1	0	D2
0	0	1	1	D3
0	1	0	0	D4
0	1	0	1	D5
0	1	1	0	D6
0	1	1	1	D7
1	0	0	0	D8
1	0	0	1	D9
1	0	1	0	D10
1	0	1	1	D11
1	1	0	0	D12
1	1	0	1	D13
1	1	1	0	D14
1	1	1	1	D15

TRUTH TABLE

16:1 MUX USING ONLY 4:1



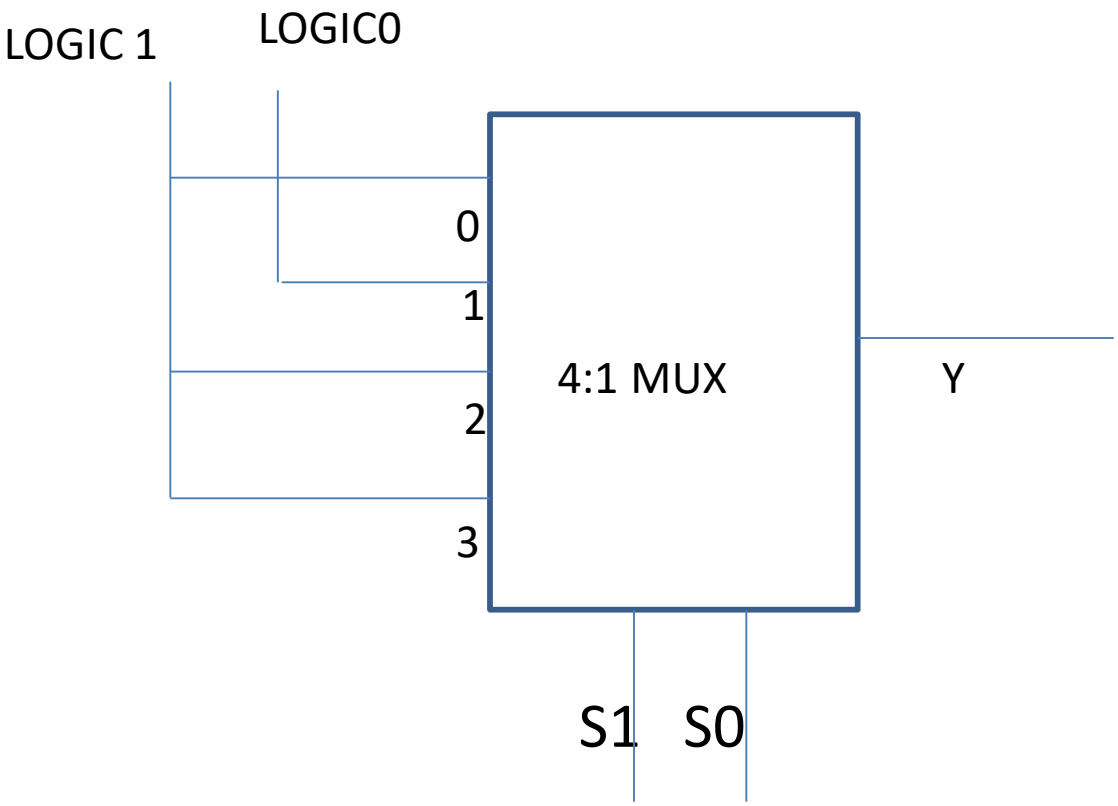
WHAT WILL BE THE OUTPUT IF

S3=1 S2=0 S1=1 S0=1

Y=?

D11

F(A,B)=(0,1,3)

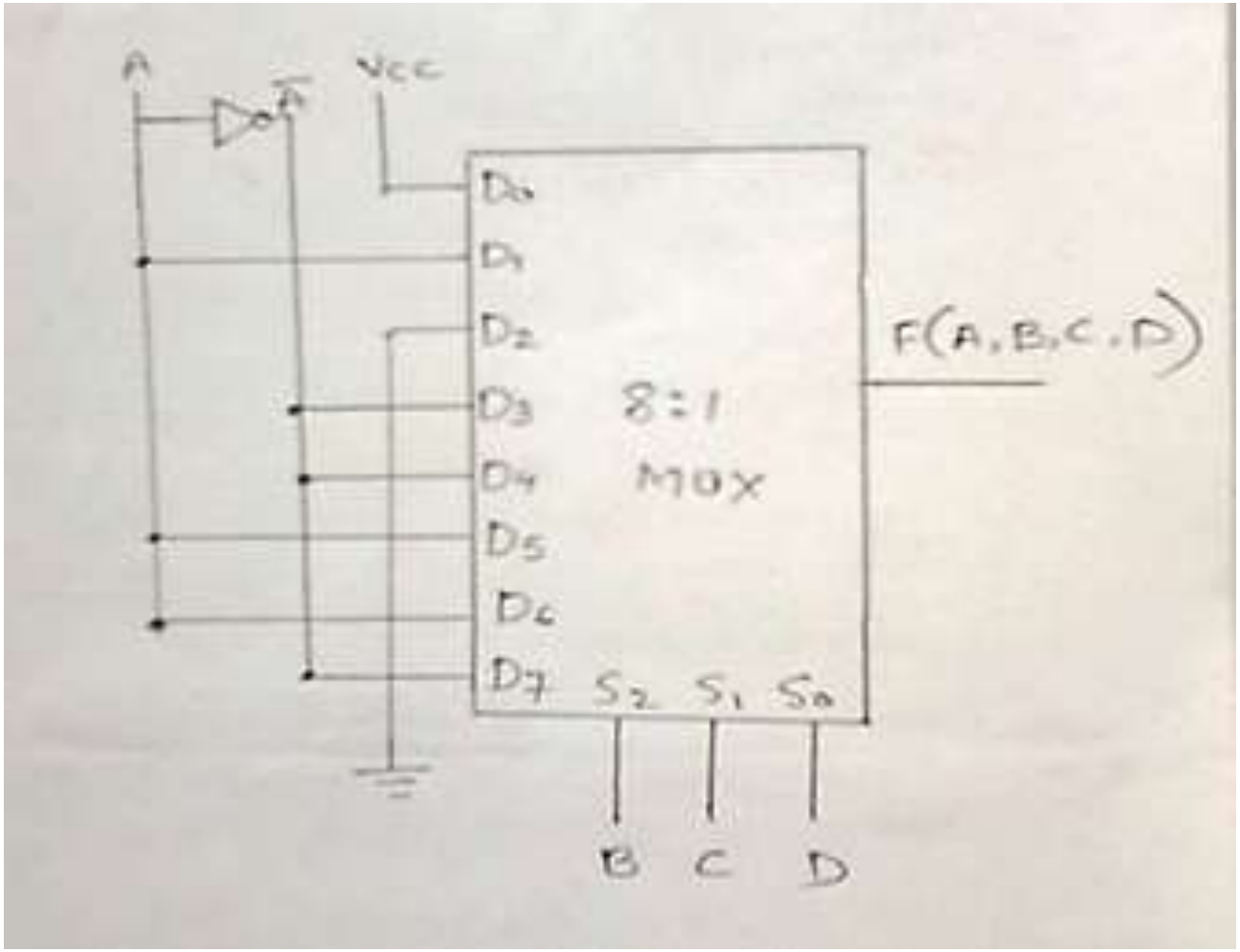


Implement the following function using 8:1 MUX

$$F(A,B,C,D) = \sum m\,(0,3,4,7,8,9,13,14)$$

The given function is in terms of minterms and is to be implemented using a 8:1 MUX. An 8:1 MUX has three select lines, whereas the given function is a 4 variable function. Hence a logic is needed to give combination of A as inputs while only B, C and D as select line inputs. The method for the same is described below.

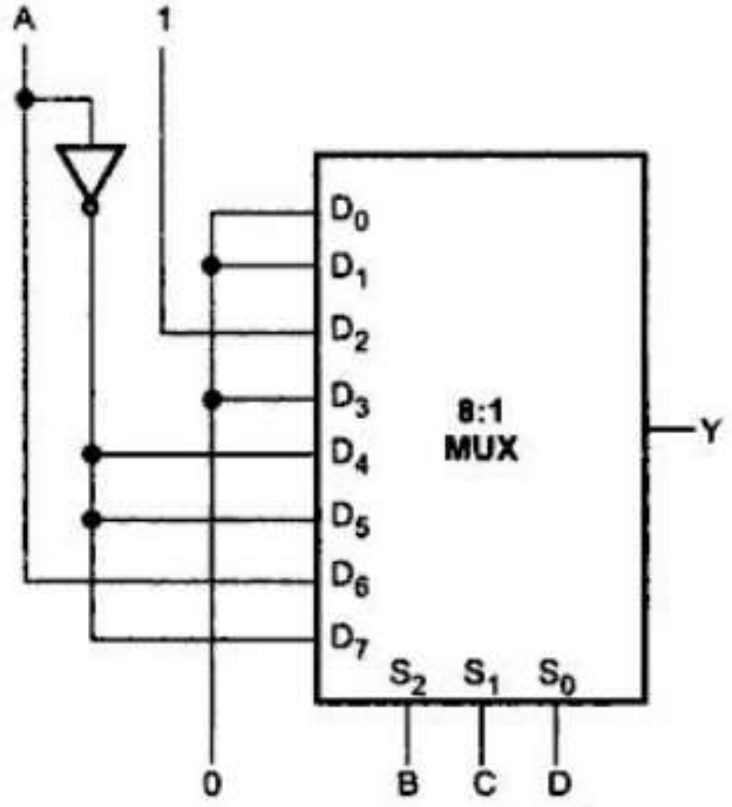
	D0	D1	D2	D3	D4	D5	D6	D7
A’	**0**	1	2	**3**	**4**	5	6	**7**
A	**8**	**9**	10	11	12	**13**	**14**	15
	1	**A**	**0**	**A’**	**A’**	**A**	**A**	**A’**


A
$\bar{A}$
Vcc
D0
D1
D2
D3
D4
D5
D6
D7
8:1
MUX
F(A,B,C,D)
S2
S1
S0
B
C
D

Implement the following using 8 : 1 MUX

i) $f(A, B, C, D) = \sum m\ (2, 4, 5, 7, 10, 14)$

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{A}$	0	1	(2)	3	(4)	(5)	6	(7)
A	8	9	(10)	11	12	13	(14)	15
	0	0	1	0	$\overline{A}$	$\overline{A}$	A	$\overline{A}$



Home work

Implement the following Boolean function using 8 :1 MUX.

$F(P, Q, R, S) = \sum m(0, 1, 3, 4, 8, 9, 15)$

Implementing boolean expressions using MUX

NPTEL video IIT Madras

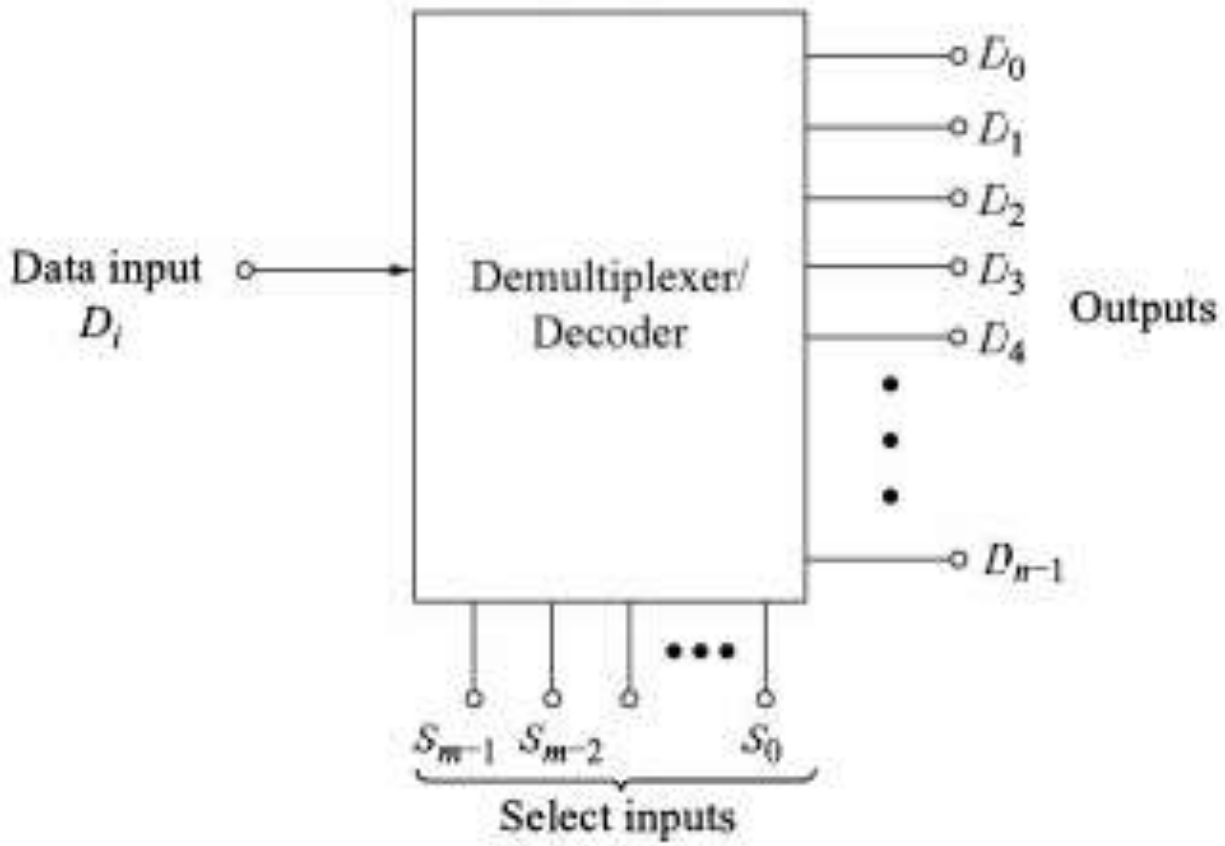
https://youtu.be/QrZgp0SAUFQ

https://www.youtube.com/watch?v=QrZgp0SAUFQ

Demultiplexer

Gauri Rao

Block diagram



De multiplexer

- Definition : A DEMULTIPLEXER (DEMUX) basically reverses the multiplexing function. It takes data from one line and distributes them to a given number of output lines. For this reason, the demultiplexers is also known as a data distributor.

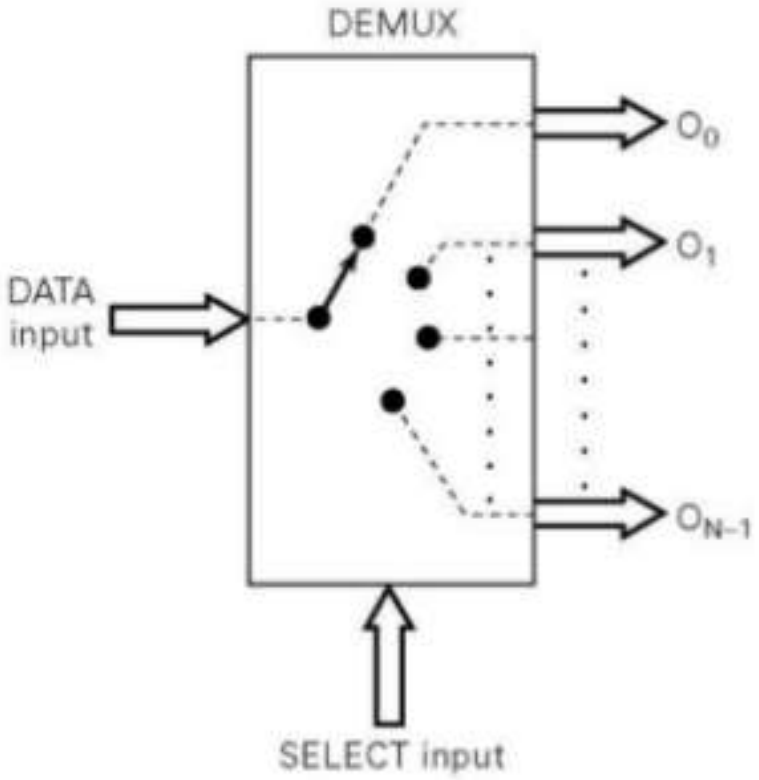
- Single data input lines
- Some select line (less than the no. of output lines)
- Several output line

- If there are n data output lines and m select lines, then

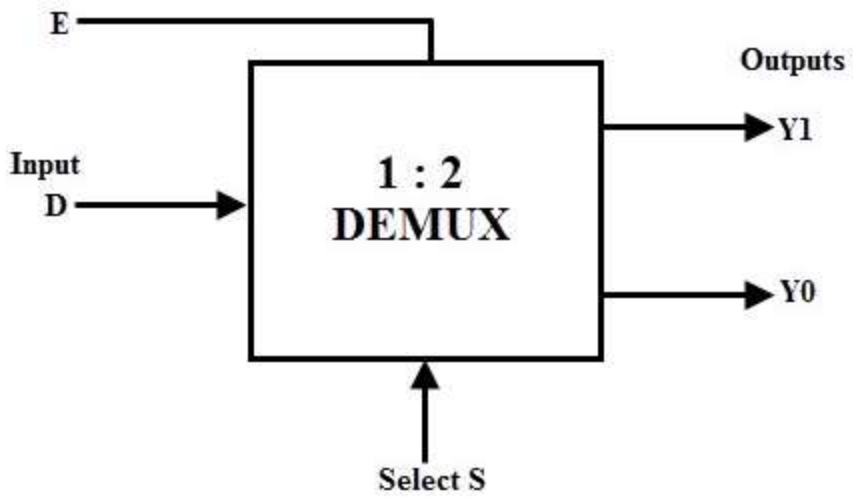
$$2^m = n$$

Functional Diagram

One: Many

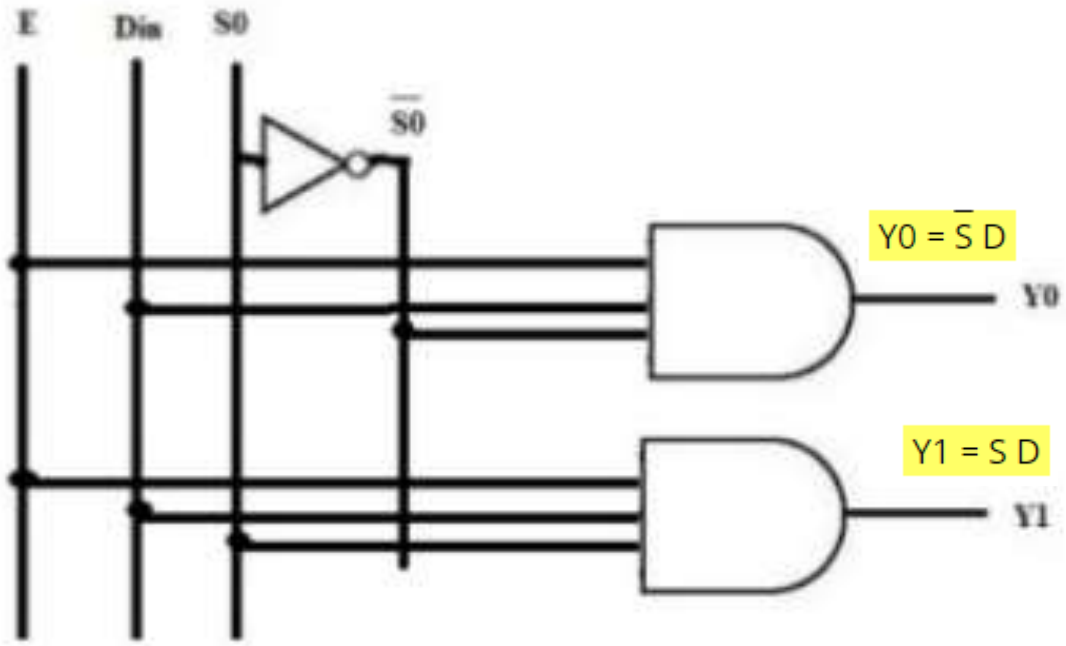


1:2 De MUX



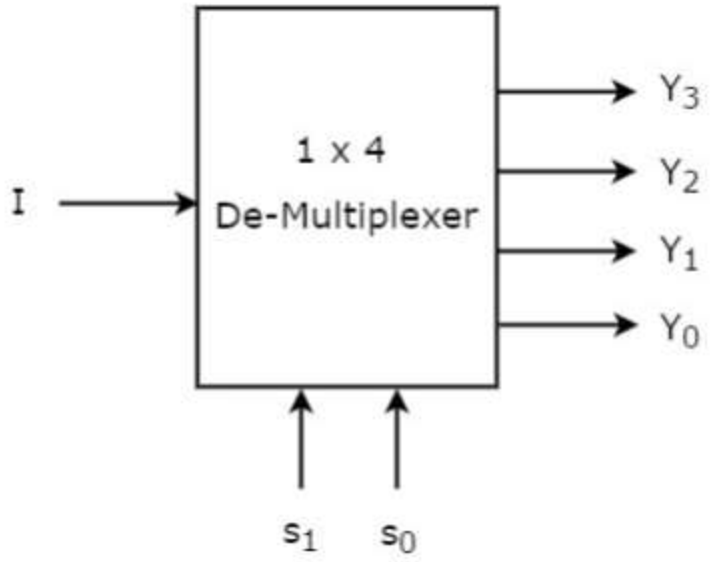
S	D	Y1	Y0
0	0	0	**0**
0	1	0	**1**
1	0	**0**	0
1	1	**1**	0

1:2 De MUX



S_0	Y_0	Y_1
0	D	0
1	0	D

1:4 De MUX



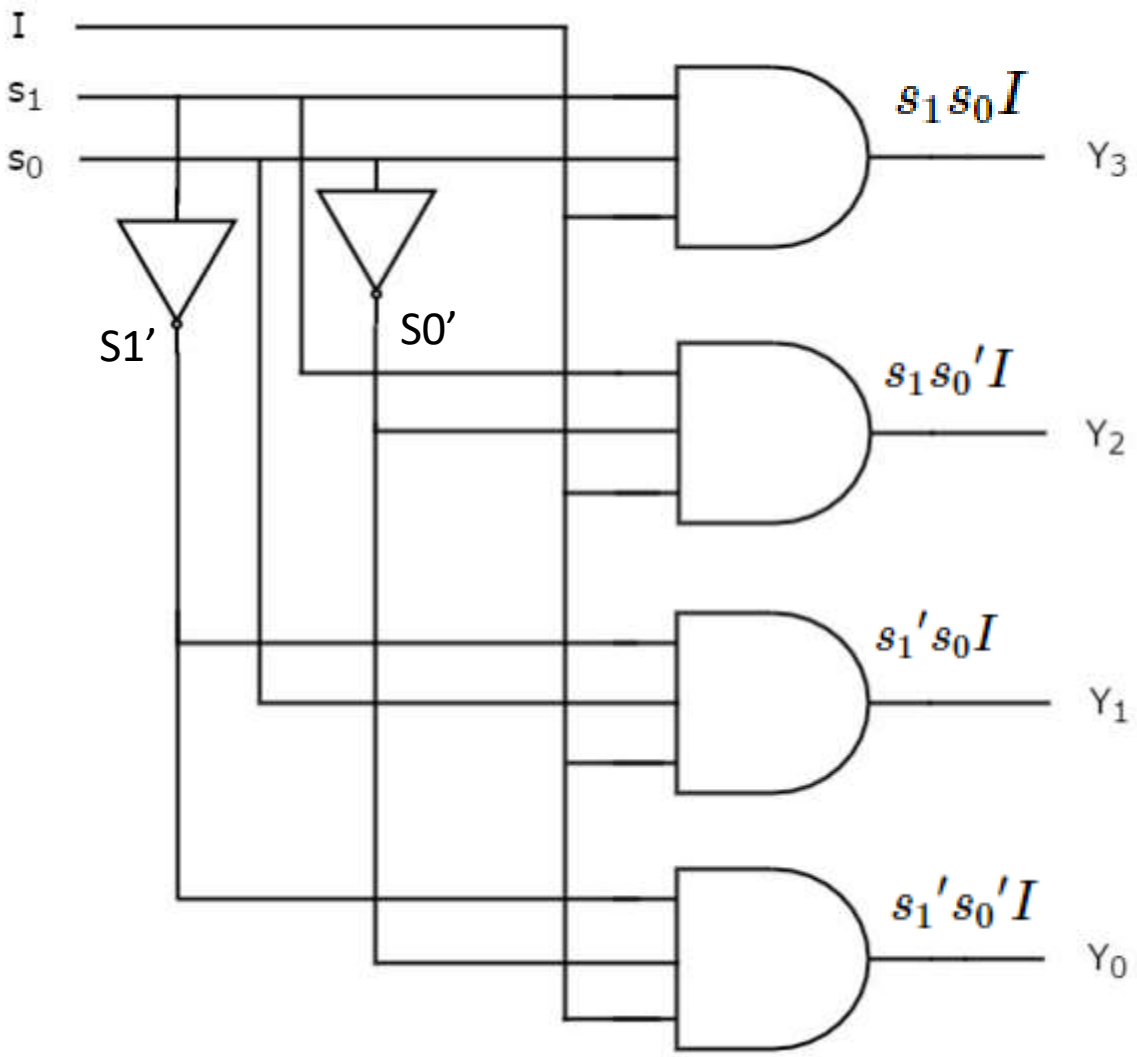
Selection Inputs		Outputs			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

$$Y_3 = s_1 s_0 I$$

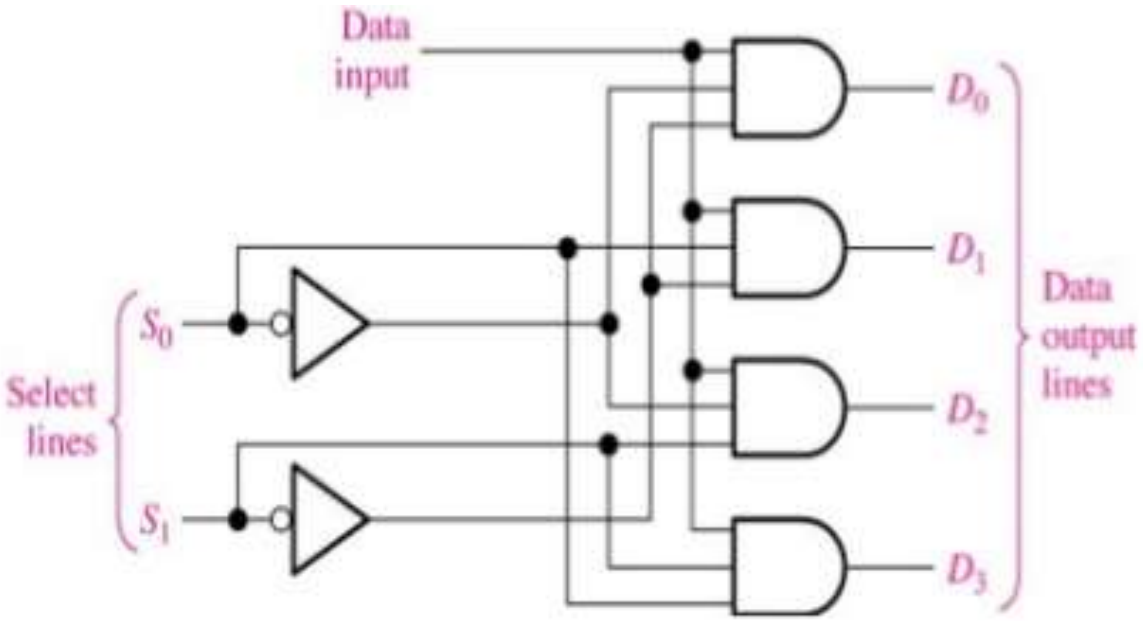
$$Y_2 = s_1 s_0' I$$

$$Y_1 = s_1' s_0 I$$

$$Y_0 = s_1' s_0' I$$

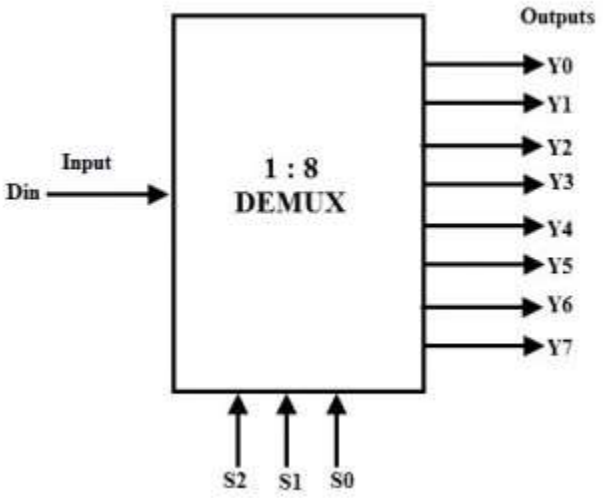

I
s_1
s_0
S1'
S0'
$s_1 s_0 I$
Y_3
$s_1 s_0' I$
Y_2
$s_1' s_0 I$
Y_1
$s_1' s_0' I$
Y_0

1:4 De Mux



S_0	S_1	O0	O1	O2	O3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

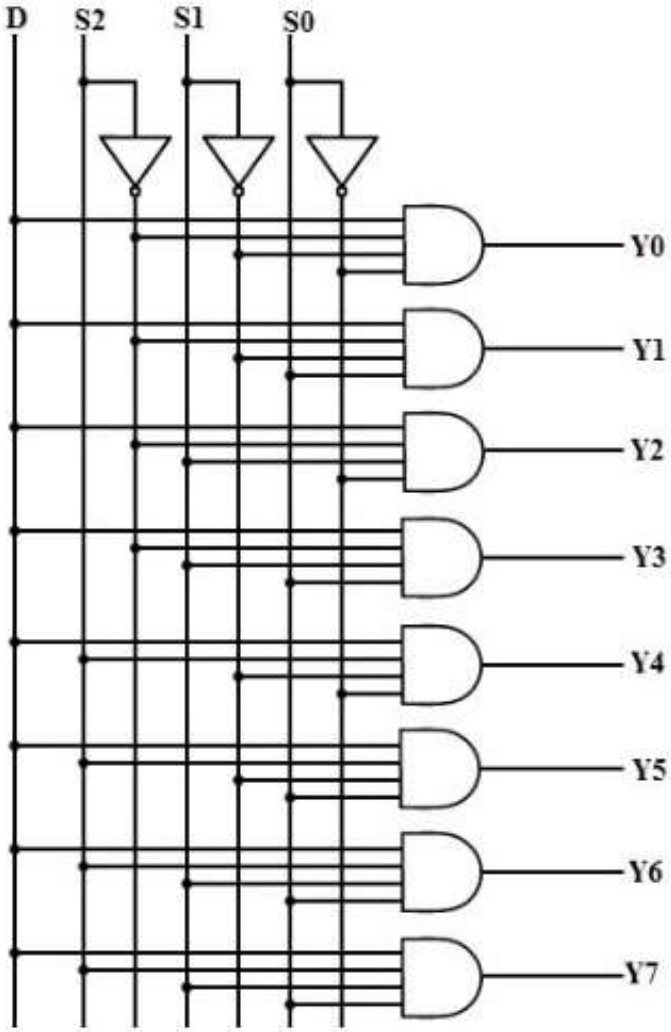
1:8 De MUX



Truth table

S2	S1	S0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
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

1:8 De MUX Circuit



$Y0 = \overline{S2}\ \overline{S1}\ \overline{S0}\ D$

$Y1 = \overline{S2}\ \overline{S1}\ S0\ D$

$Y2 = \overline{S2}\ S1\ \overline{S0}\ D$

$Y3 = \overline{S2}\ S1\ S0\ D$

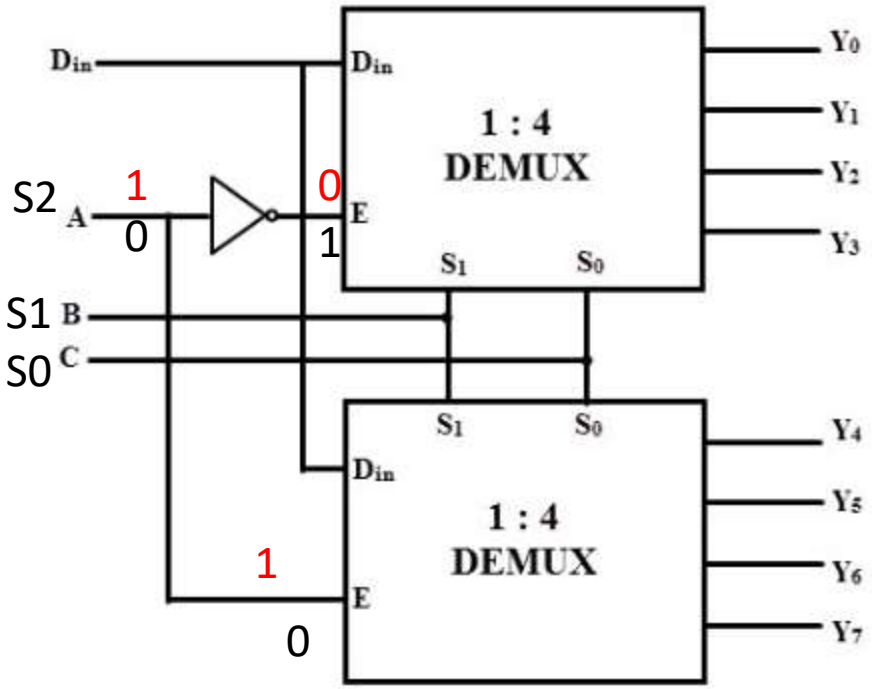
$Y4 = S2\ \overline{S1}\ \overline{S0}\ D$

$Y5 = S2\ \overline{S1}\ S0\ D$

$Y6 = S2\ S1\ \overline{S0}\ D$

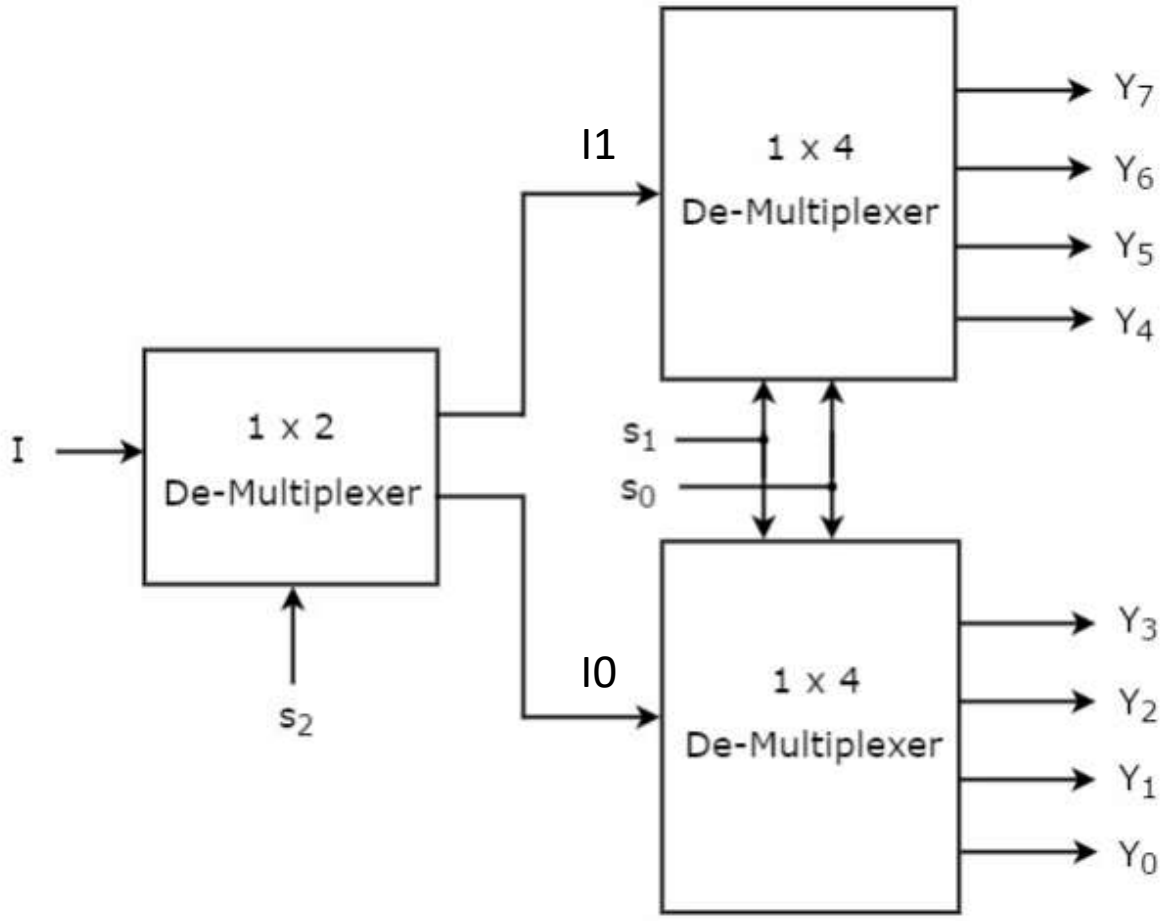
$Y7 = S2\ S1\ S0\ D$

1:8 using two 1:4 demux



S2	S1	S0	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
0	0	0	D	0	0	0	0	0	0	0
0	0	1		D						
0	1	0			D					
0	1	1				D				
1	0	0					**D**			
1	0	1						**D**		
1	1	0							**D**	
1	1	1								**D**

1:8 MUX using 1:4 and 1:2 MUX



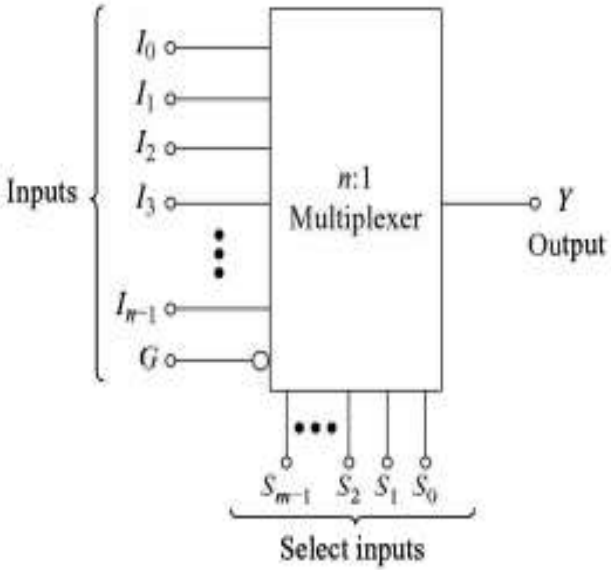
Truth table

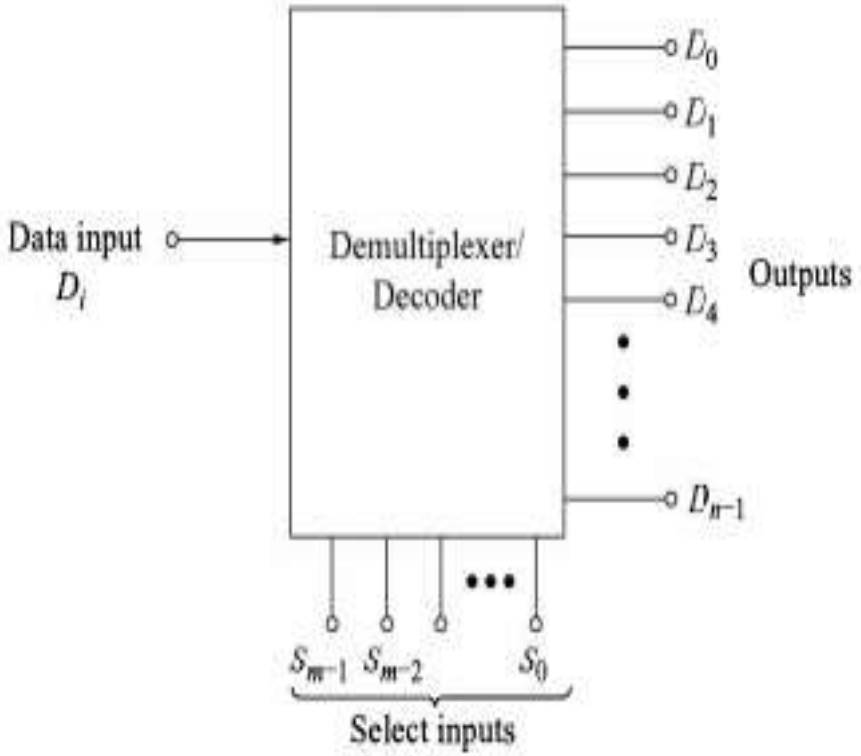
Selection Inputs			Outputs							
s_2	s_1	s_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	I
0	0	1	0	0	0	0	0	0	I	0
0	1	0	0	0	0	0	0	I	0	0
0	1	1	0	0	0	0	I	0	0	0
1	0	0	0	0	0	I	0	0	0	0
1	0	1	0	0	I	0	0	0	0	0
1	1	0	0	I	0	0	0	0	0	0
1	1	1	I	0	0	0	0	0	0	0

Comparison

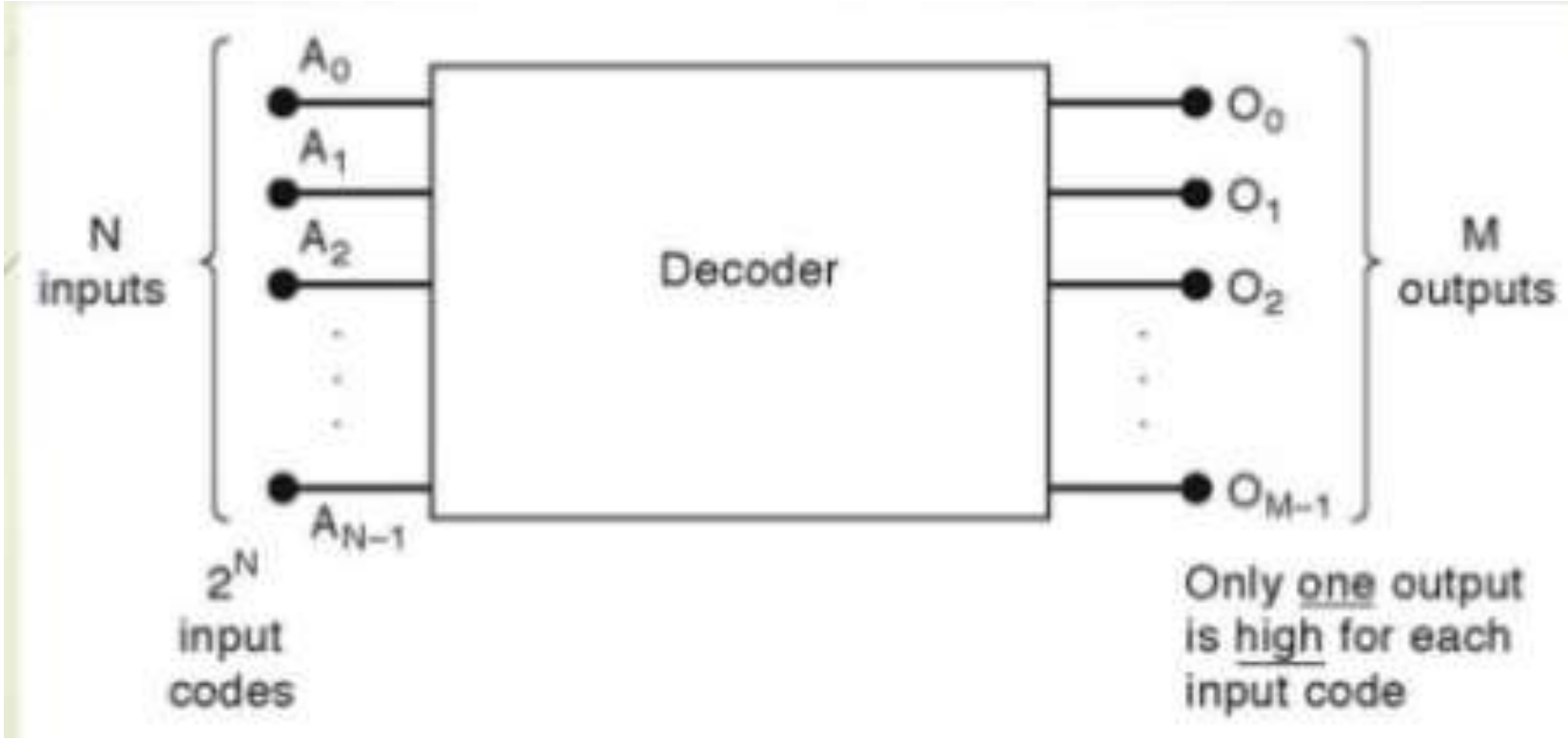
Sr. no.	Parameter	Multiplexer	De-multiplexer
1	Type of logic circuit	Combinational	Combinational
2	No. of data inputs	n	1
3	No. of select inputs	m	m
4	No. of data output	1	n
5	Relation between input/output lines & select lines	$n=2^m$	$n=2^m$
6	Operation principle	Many to 1 or as data selector	1 to many or data distributor

Block diagram



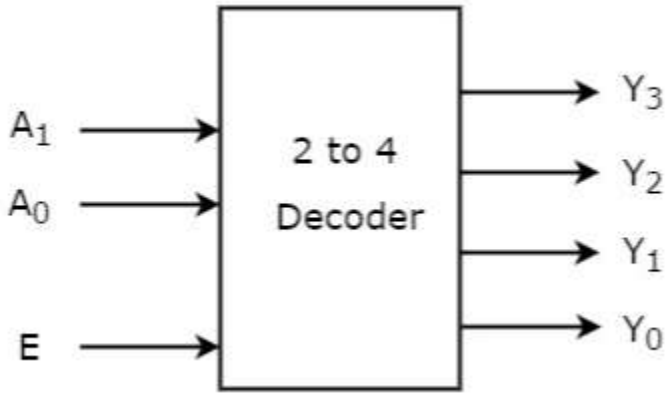


Decoder


N
inputs
A_0
A_1
A_2
A_{N-1}
Decoder
O_0
O_1
O_2
O_{M-1}
M
outputs
2^N
input
codes
Only one output
is high for each
input code

Decoder is a combinational circuit that has 'n' input lines and maximum of 2^n output lines. One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code. The outputs of the decoder are nothing but the **min terms** of 'n' input variables $lines$, when it is enabled.

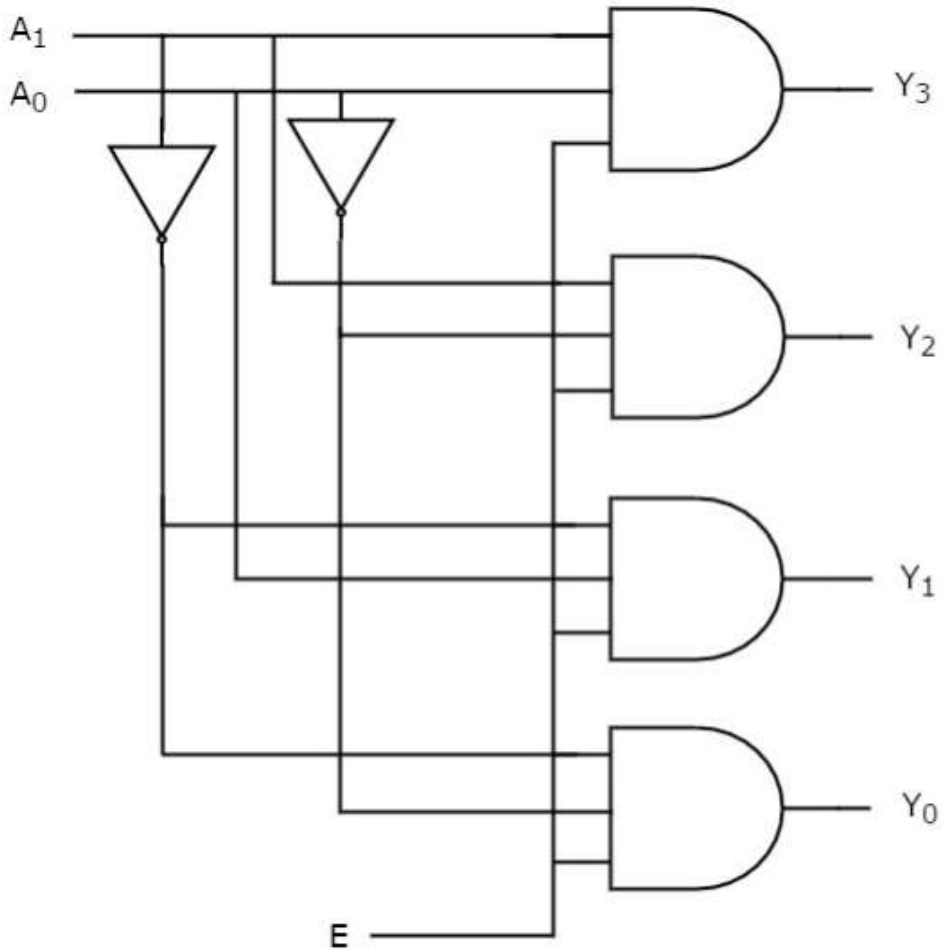
2:4 decoder



Truth table

Enable	Inputs		Outputs			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Boolean function Implementation



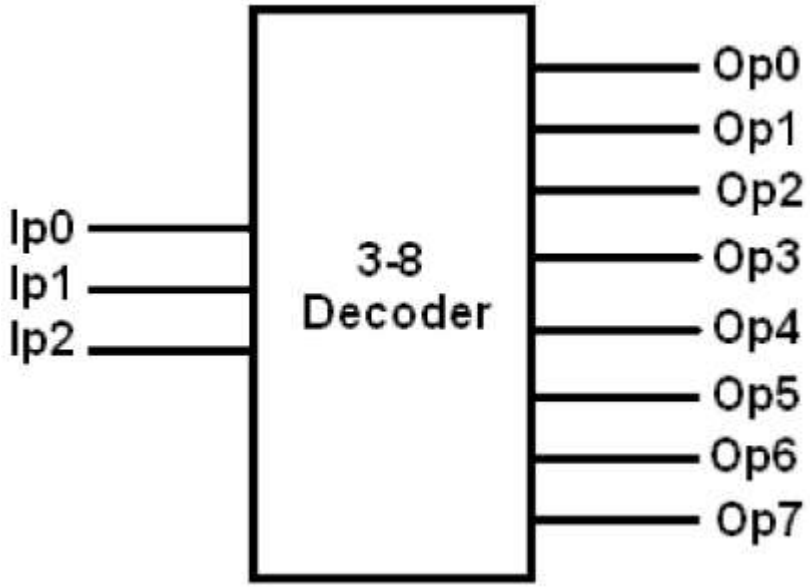
$$Y_3 = E.A_1.A_0$$

$$Y_2 = E.A_1.A_0'$$

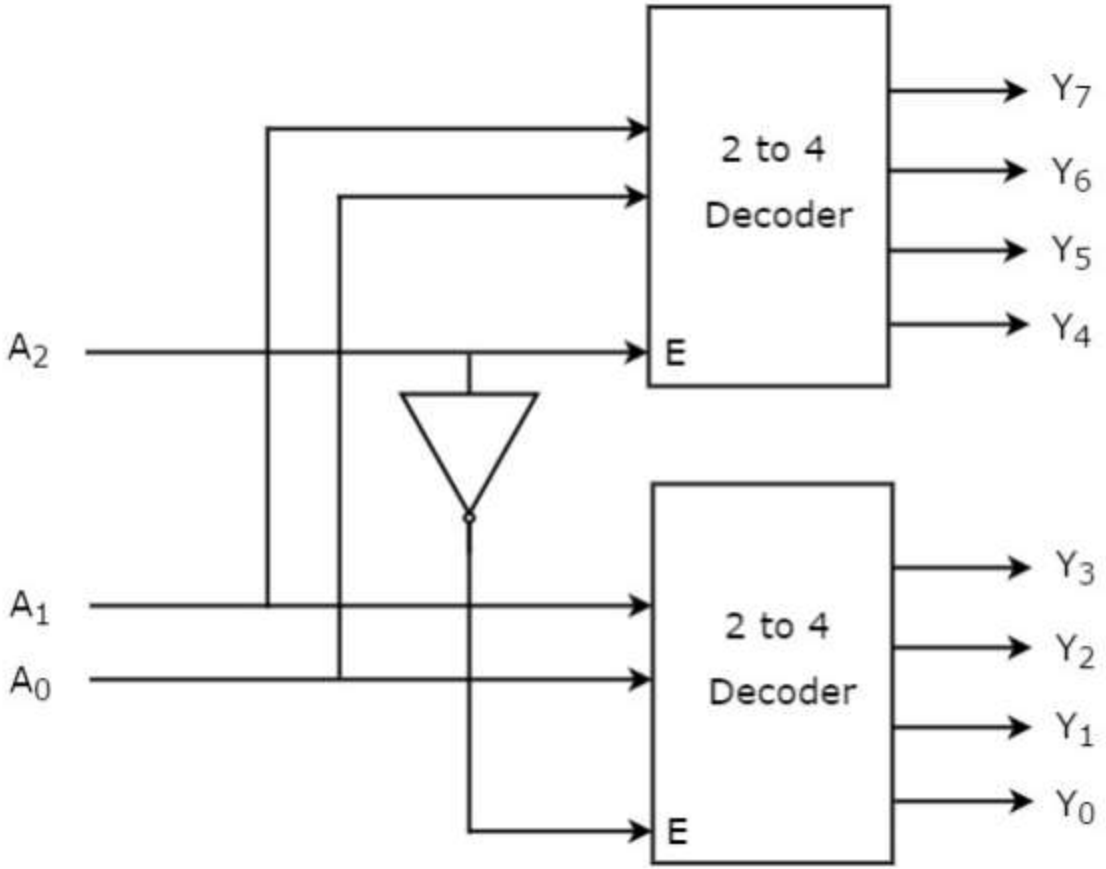
$$Y_1 = E.A_1'.A_0$$

$$Y_0 = E.A_1'.A_0'$$

3:8 decoder



3:8 decoder using 2:4 decoder

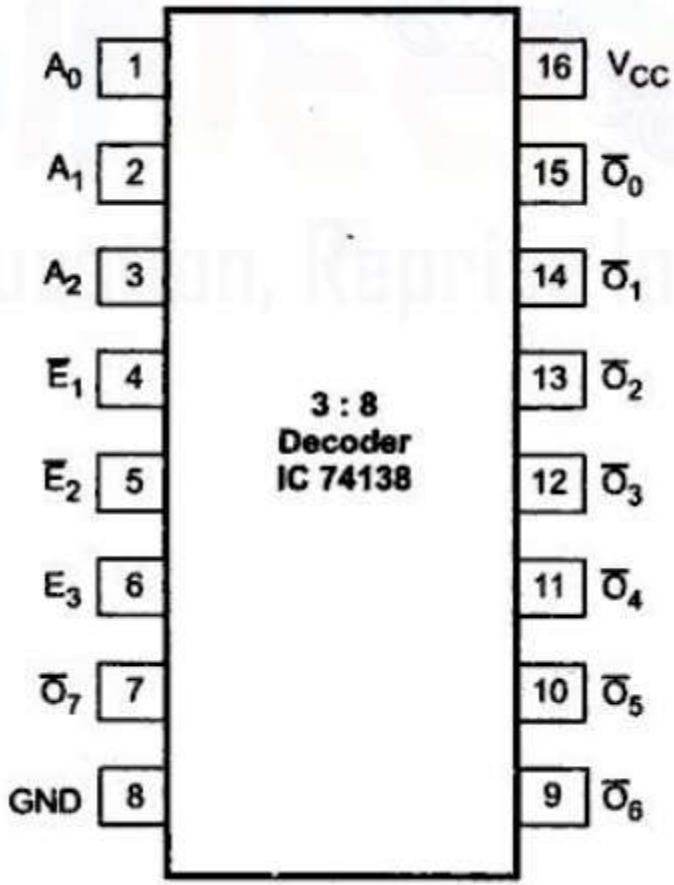


$$Required\ number\ of\ 2\ to\ 4\ decoders = \frac{8}{4} = 2$$

Therefore, we require two 2 to 4 decoders for implementing one 3 to 8 decoder.

The parallel inputs A_1 & A_0 are applied to each 2 to 4 decoder. The complement of input A_2 is connected to Enable, E of lower 2 to 4 decoder in order to get the outputs, Y_3 to Y_0. These are the **lower four min terms**. The input, A_2 is directly connected to Enable, E of upper 2 to 4 decoder in order to get the outputs, Y_7 to Y_4. These are the **higher four min terms**.

IC 74138

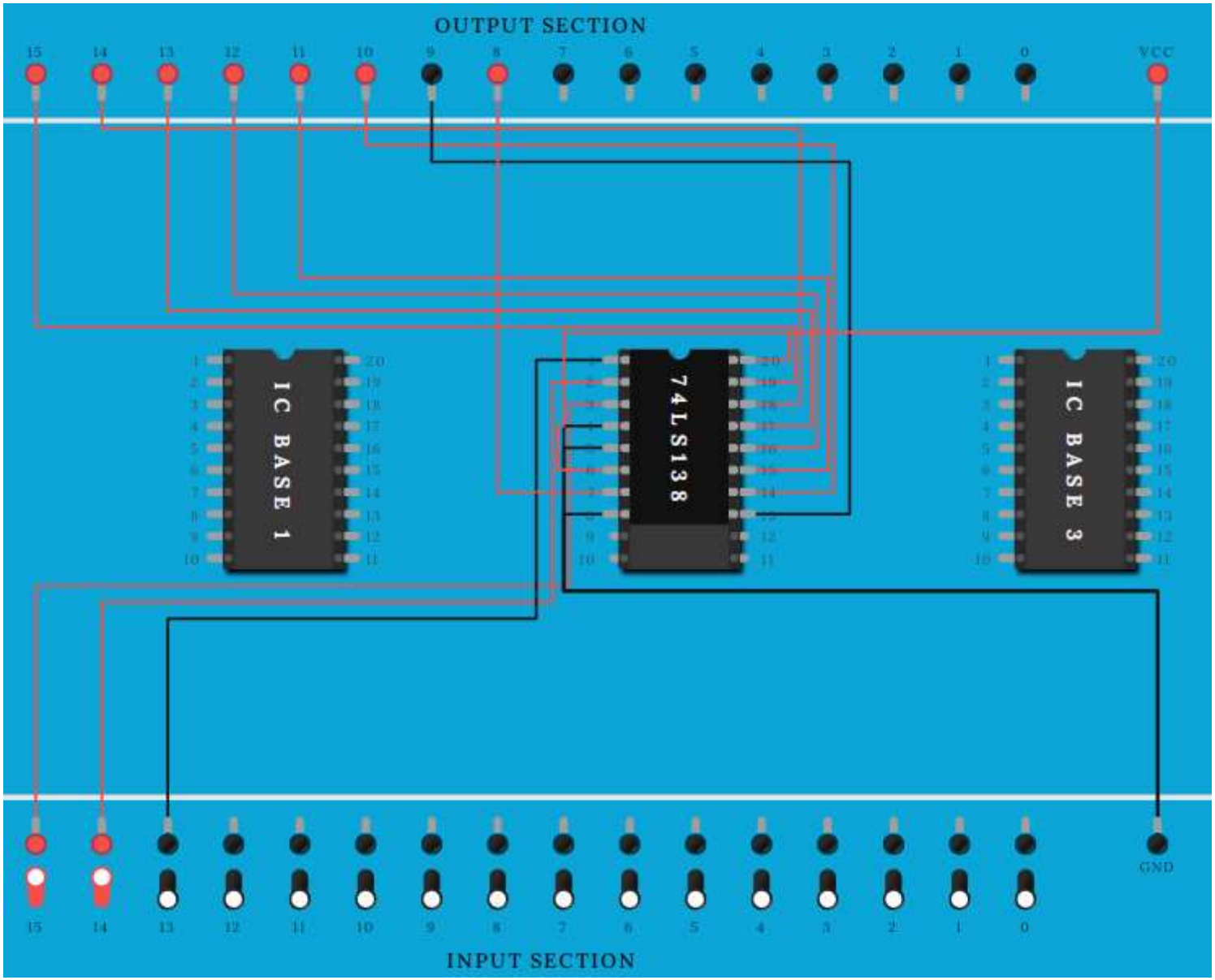


- Pin4 (G2A): Active low enable pin
- Pin5 (G2B): Active low enable pin
- Pin6 (G1): Active high enable pin

The IC 74LS138 is a 3 to 8 line decoder **integrated circuit** from the 74xx family of **transistor-transistor-logic-gates**. The main function of this IC is to decode otherwise demultiplex the applications. The setup of this IC is accessible with 3-inputs to 8-output setup. This IC is mainly used in applications like memory decoding with high performance otherwise data routing, etc. These ICs can be used for minimizing the system decoding effects in memory systems with high performance. This IC includes three enable pins (where two pins are active low and one is active high) decreases the necessity of outside gates. The implementation of 24 line decoder can be done without using outside inverters, as well as a 32-line decoder needs a single inverter

Truth table

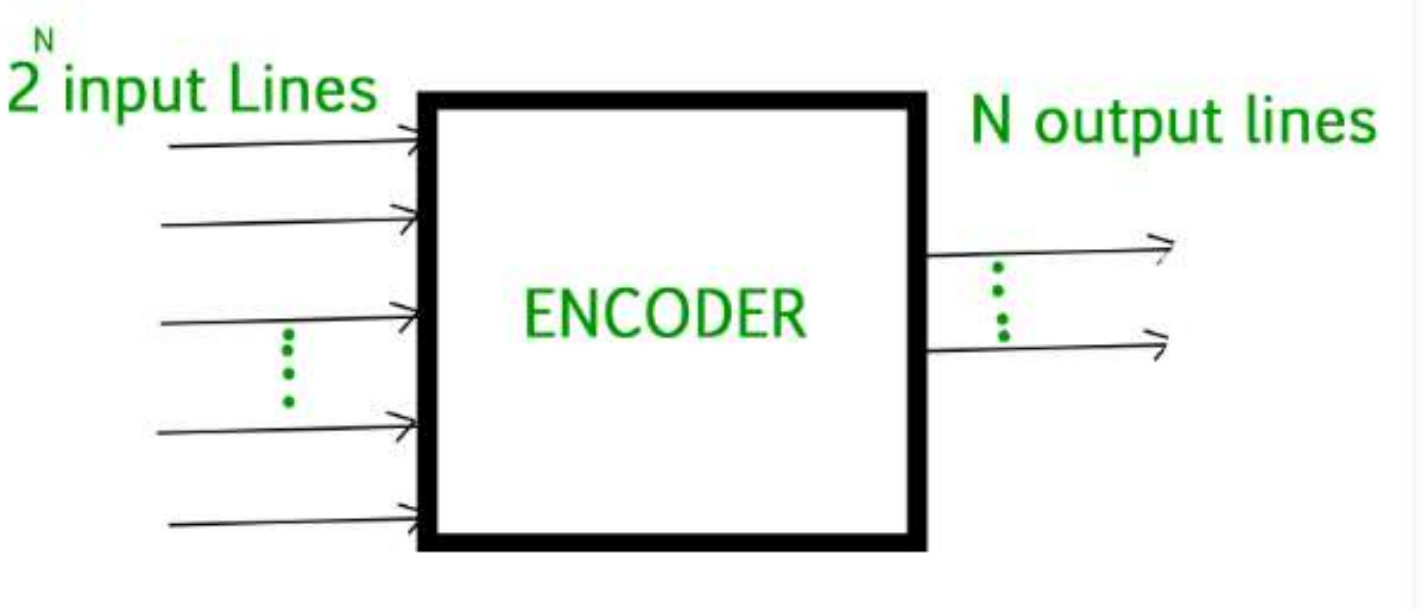
Inputs					Outputs							
Enable		Select										
G1	G2 (Note 1)	C	B	A	YO	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L


OUTPUT SECTION
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 VCC
IC BASE 1
74LS138
IC BASE 3
GND
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
INPUT SECTION

Encoder

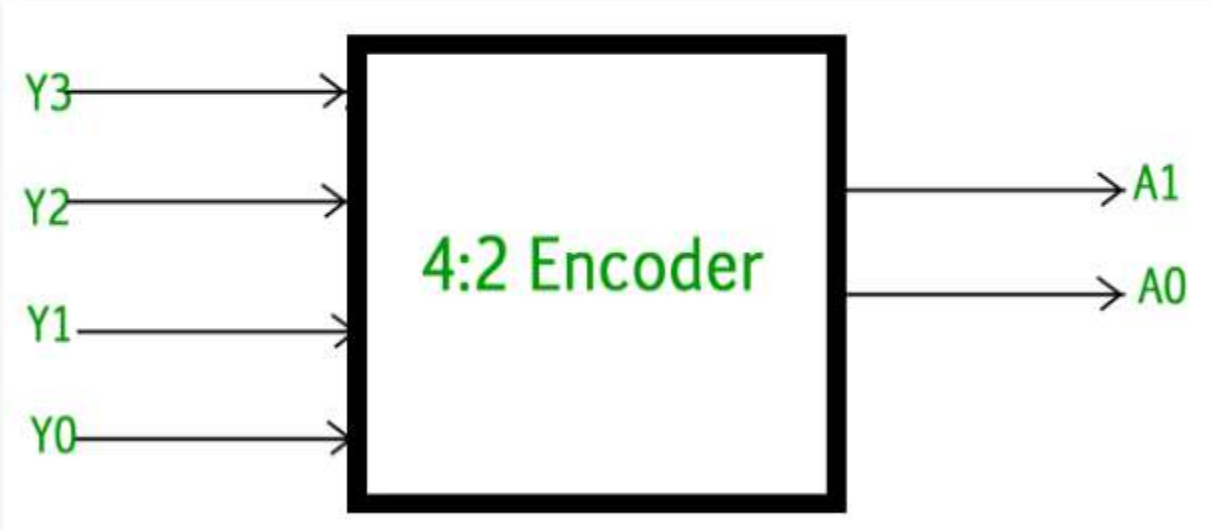
An Encoder is a **combinational circuit** that performs the reverse operation of Decoder.It has maximum of **2^n input lines** and **'n' output lines**, hence it encodes the information from 2^n inputs into an n-bit code. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes 2^n input lines with 'n' bits.

Block Diagram



4:2 Encoder

The 4 to 2 Encoder consists of **four inputs Y3, Y2, Y1 & Y0 and two outputs A1 & A0**. At any time, only one of these 4 inputs can be '1' in order to get the respectiv binary code at the output. The figure below shows the logic symbol of 4 to 2 encoder :

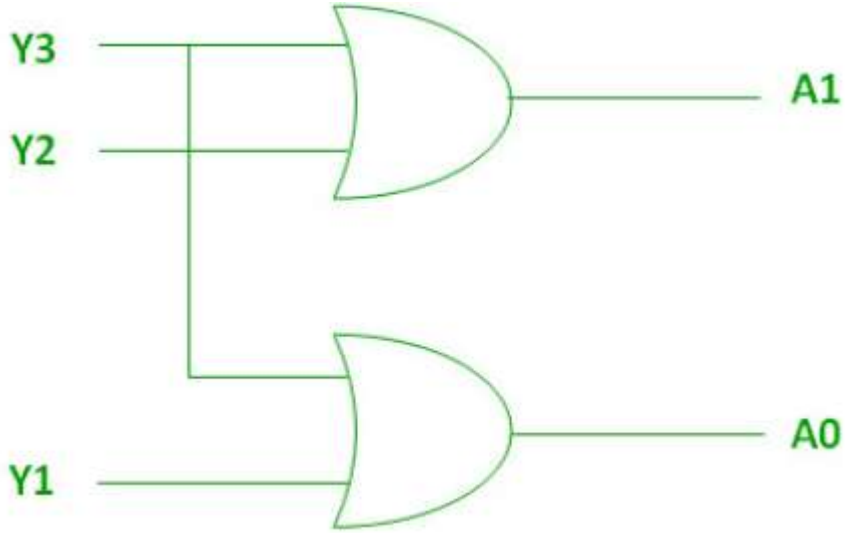


Truth Table

INPUTS				OUTPUTS	
Y3	Y2	Y1	Y0	A1	A0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Implementation

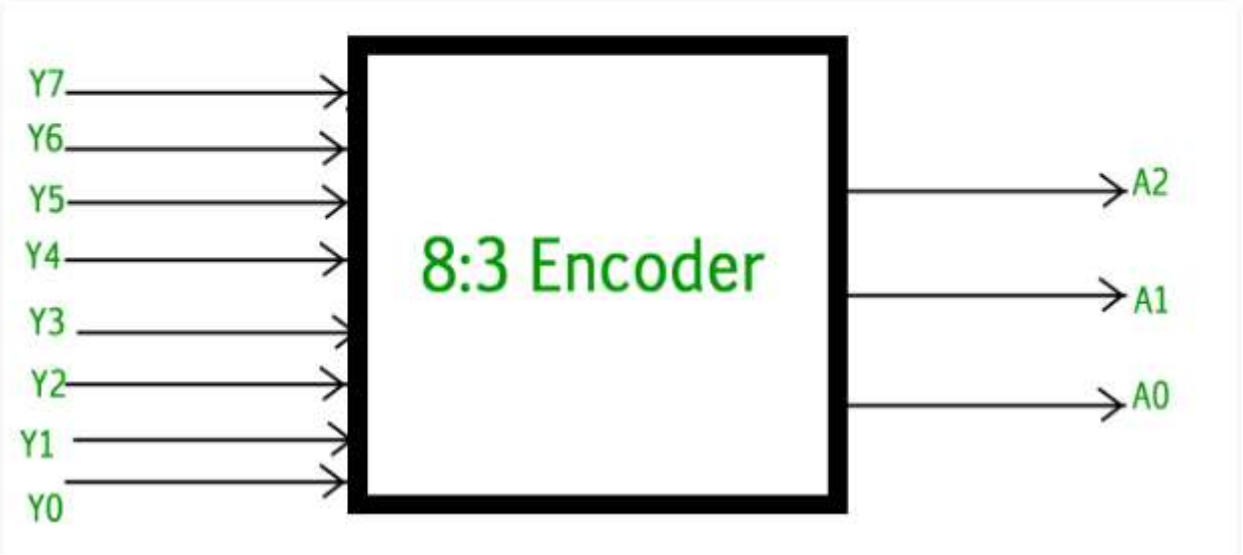
```
A1 = Y3 + Y2
A0 = Y3 + Y1
```



8:3 Encoder

The 8 to 3 Encoder or octal to Binary encoder consists of **8 inputs** : Y7 to Y0 and **3 outputs** : A2, A1 & A0. Each input line corresponds to each octal digit and three outputs generate corresponding binary code.

The figure below shows the logic symbol of octal to binary encoder:

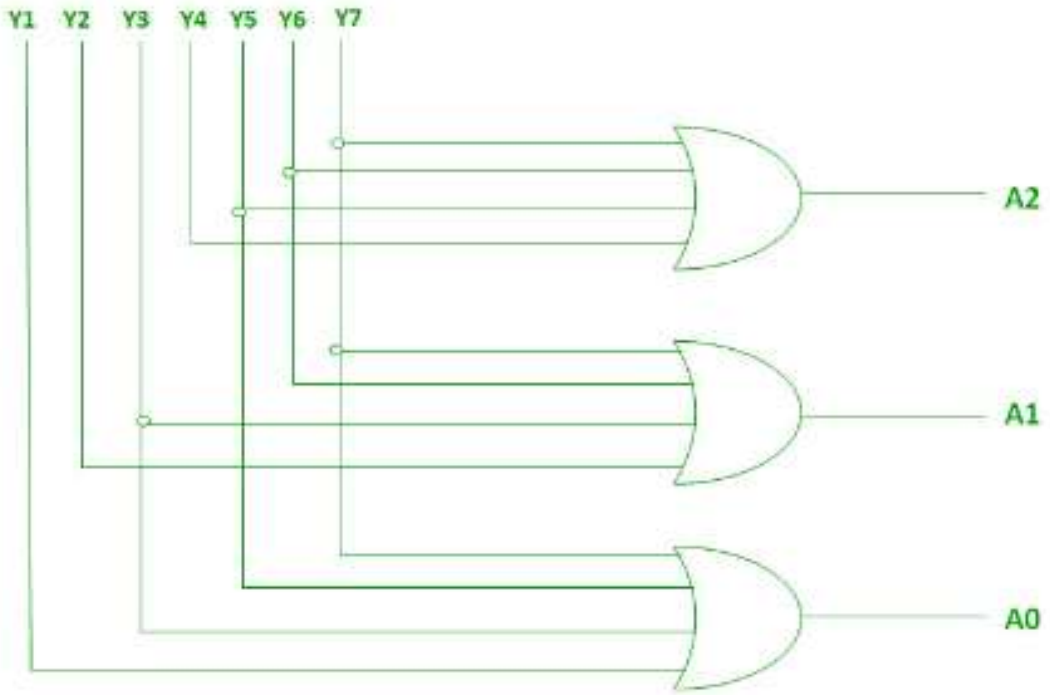


Truth table

INPUTS								OUTPUTS		
Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	A2	A1	A0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Implementation

```
A2 = Y7 + Y6 + Y5 + Y4
A1 = Y7 + Y6 + Y3 + Y2
A0 = Y7 + Y5 + Y3 + Y1
```



Priority Encoder

One of the main disadvantages of standard digital encoders is that they can generate the wrong output code when there is more than one input present at logic level "1". For example, if we make inputs D_1 and D_2 HIGH at logic "1" both at the same time, the resulting output is neither at "01" or at "10" but will be at "11" which is an output binary number that is different to the actual input present. Also, an output code of all logic "0"s can be generated when all of its inputs are at "0" OR when input D_0 is equal to one.

One simple way to overcome this problem is to "Prioritise" the level of each input pin. So if there is more than one input at logic level "1" at the same time, the actual output code would only correspond to the input with the highest designated priority. Then this type of digital encoder is known commonly as a **Priority Encoder** or **P-encoder** for short.

Priority Encoder

The **Priority Encoder** solves the problems mentioned above by allocating a priority level to each input. The *priority encoders* output corresponds to the currently active input which has the highest priority. So when an input with a higher priority is present, all other inputs with a lower priority will be ignored.

A 4 to 2 priority encoder has **4 inputs** : Y3, Y2, Y1 & Y0 and **2 outputs** : A1 & A0. Here, the input, Y3 has the **highest priority**, whereas the input, Y0 has the **lowest priority**. In this case, even if more than one input is ‘1’ at the same time, the output will be the (binary) code corresponding to the input, which is having **higher priority**.

Truth table

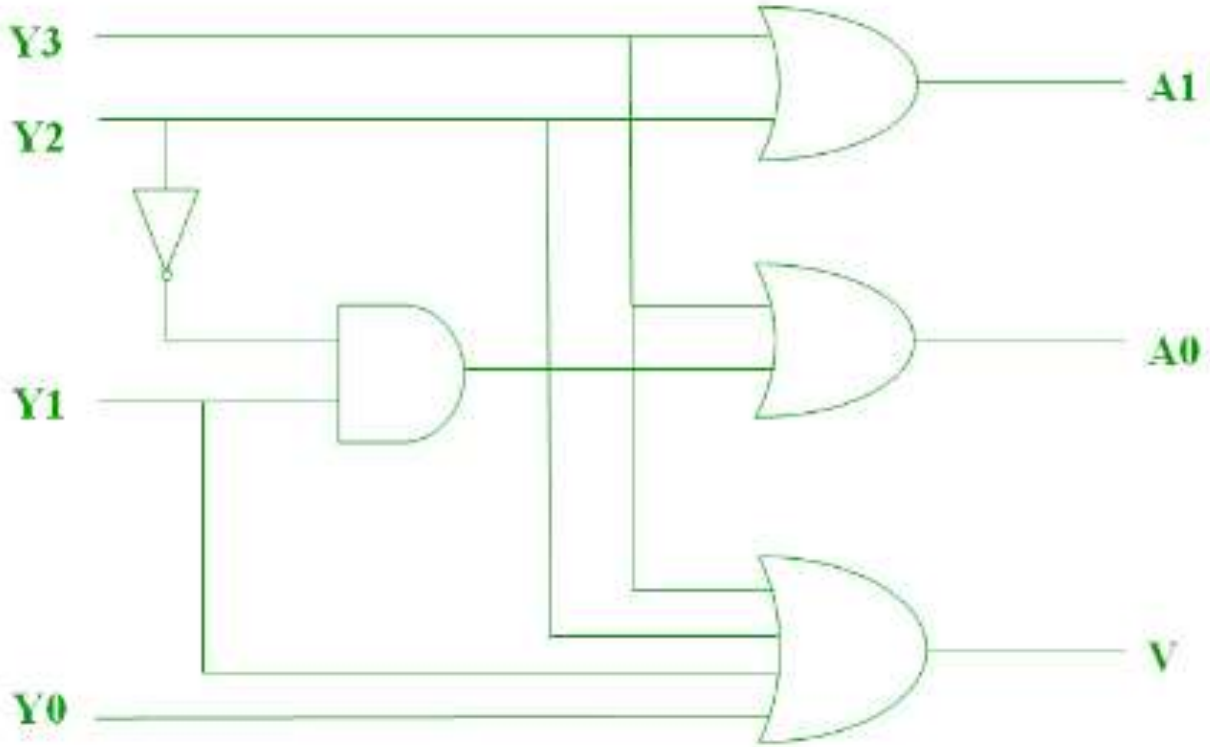
INPUTS				OUTPUTS		
Y3	Y2	Y1	Y0	A1	A0	V
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

Y3 Y2 \ Y1 Y0	00	01	11	10
00	X	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

A1=Y3 + Y2

Y3 Y2 \ Y1 Y0	00	01	11	10
00	X	0	1	1
01	0	0	0	0
11	X	X	X	X
10	1	1	1	1

A0=Y3 + Y2’ Y1


Y3
Y2
Y1
Y0
A1
A0
V

Drawbacks of Normal Encoders –

1. There is an ambiguity, when all outputs of encoder are equal to zero.
2. If more than one input is active High, then the encoder produces an output, which may not be the correct code.

So, to overcome these difficulties, we should assign priorities to each input of encoder. Then, the output of encoder will be the (code corresponding to the active High inputs, which has higher priority.

Uses of Encoders –

1. Encoders are very common electronic circuits used in all digital systems.
2. Encoders are used to translate the decimal values to the binary in order to perform the binary functions such as addition, subtraction, multiplication, etc.
3. Other applications especially for Priority Encoders may include detecting interrupts in microprocessor applications.

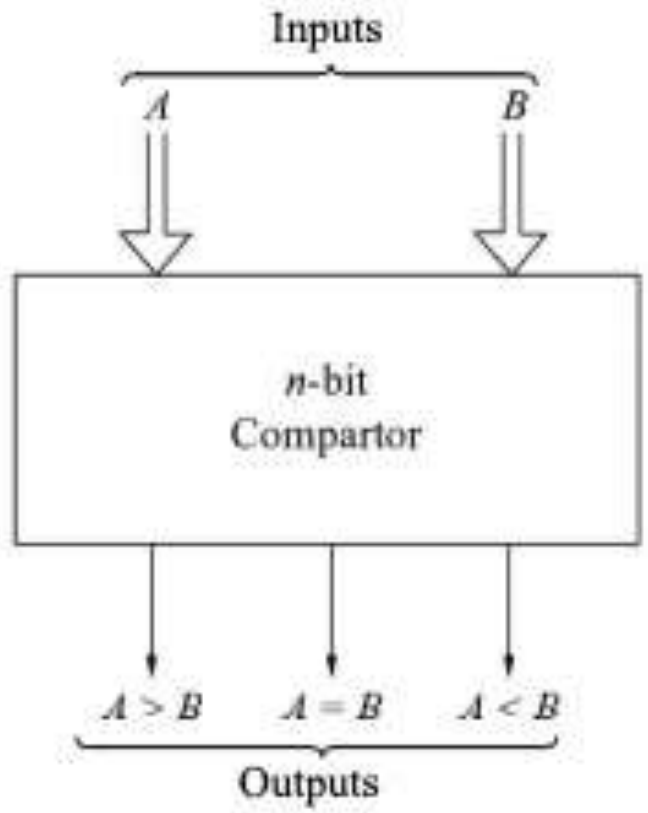
Digital Comparator

Gauri Rao

Digital Comparator

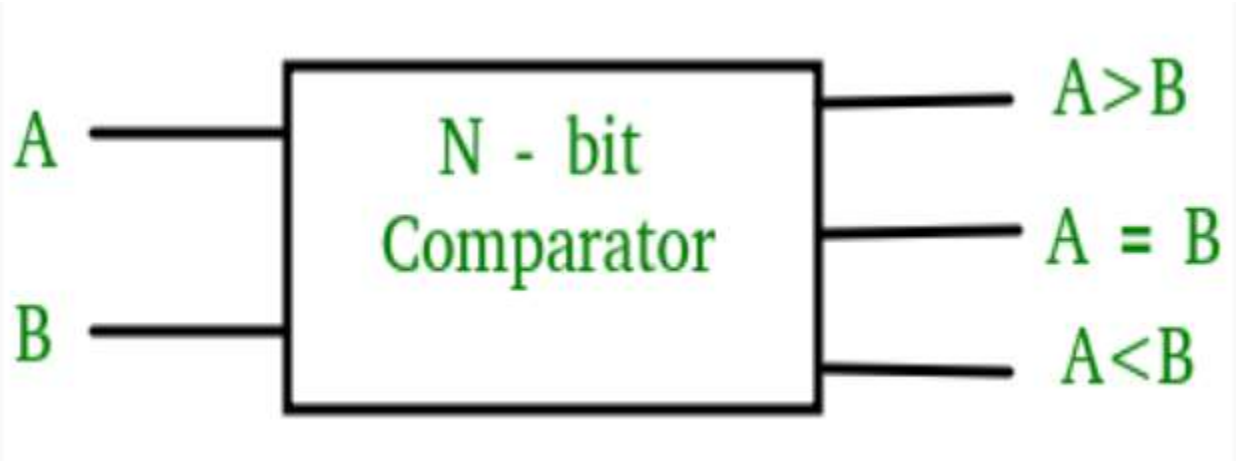
A magnitude digital Comparator is a combinational circuit that **compares two digital or binary numbers** in order to find out whether one binary number is equal, less than or greater than the other binary number. We logically design a circuit for which we will have two inputs one for A and other for B and have three output terminals, one for A > B condition, one for A = B condition and one for A < B condition.

Block diagram



1-Bit Magnitude Comparator –

A comparator used to compare two bits is called a single bit comparator. It consists of two inputs each for two single bit numbers and three outputs to generate less than, equal to and greater than between two binary numbers.



Truth table

A	B	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

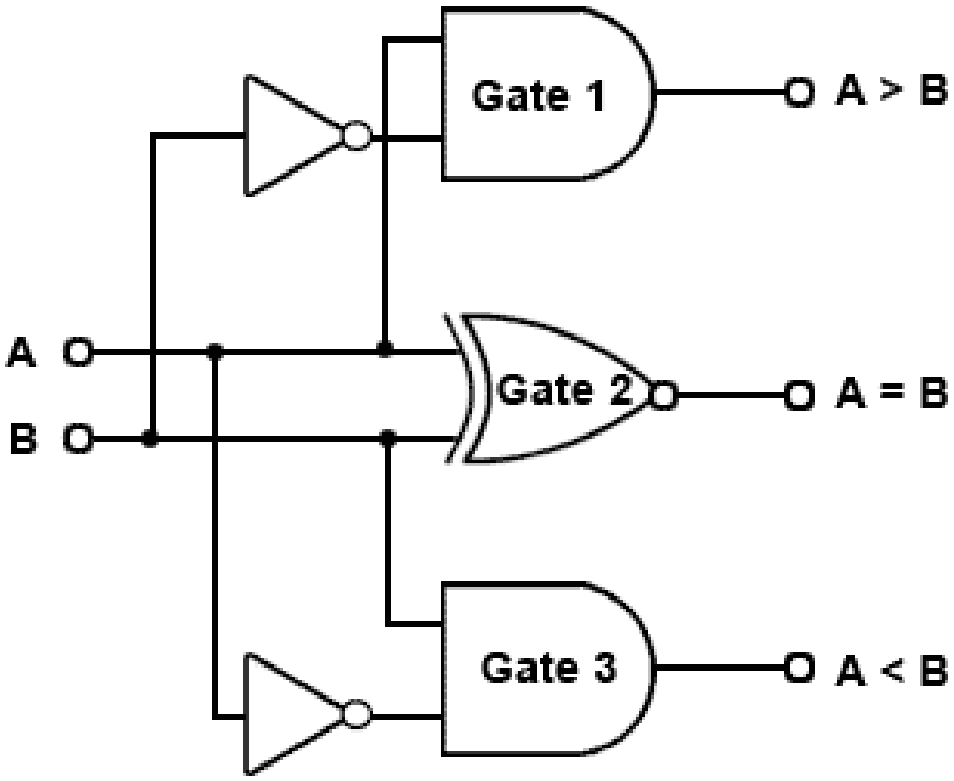
```
A>B: AB'

A<B: A'B

A=B: A'B' + AB
```

Circuit diagram of 1 bit Comparator

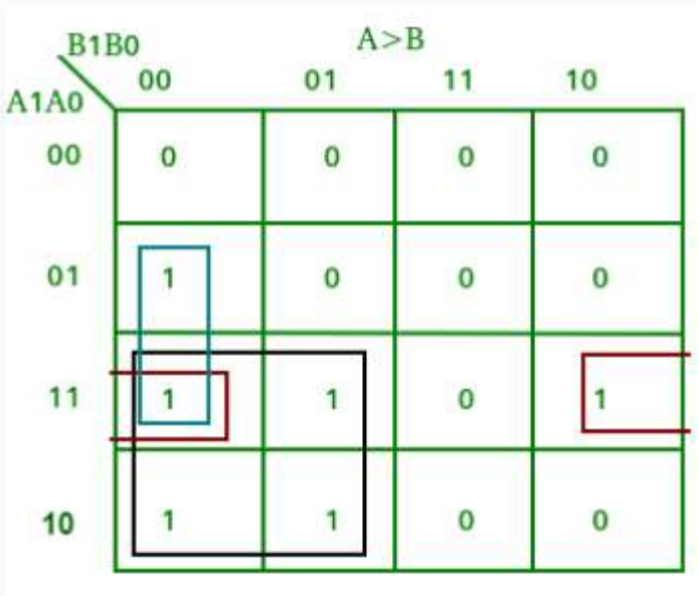
```
A>B: AB'
A<B: A'B
A=B: A'B' + AB
```

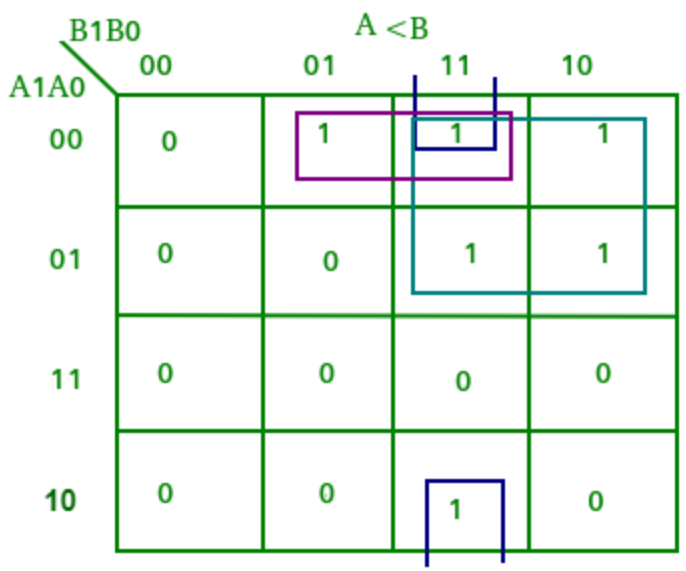


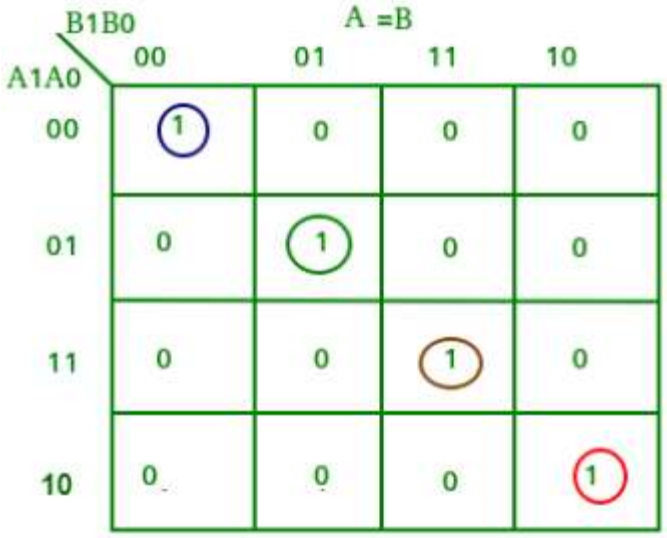
digital Comparator :- It receives two n-bit numbers A and B as inputs and the outputs are $A > B$, $A = B$, and $A < B$. Depending upon the relative magnitude of the two numbers, one of the outputs will be HIGH. As shown in the truth table of a 2-bit comparator. The reader is advised to simplify the expressions for $A > B$, $A = B$, and $A < B$ outputs using K-map and design the circuit using gates. However, 4-bit comparators are available in MSI (7485) which can compare straight binary and natural BCD codes. These ICs can be cascaded to compare words of greater lengths without external gates. The $A > B$, $A = B$, and $A < B$ outputs of a stage handling less-significant bits are connected to the corresponding $A > B$, $A = B$, and $A < B$ cascading inputs of the next stage handling more-significant bits. The stage handling the least-significant bits must have $A = B$ input connected to logic 1 level and $A > B$ and $A < B$ inputs connected to logic 0 or 1 level.

Truth table for 2 bit comparator

Inputs				Outputs		
A_1	A_0	B_1	B_0	$A > B$	$A = B$	$A < B$
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

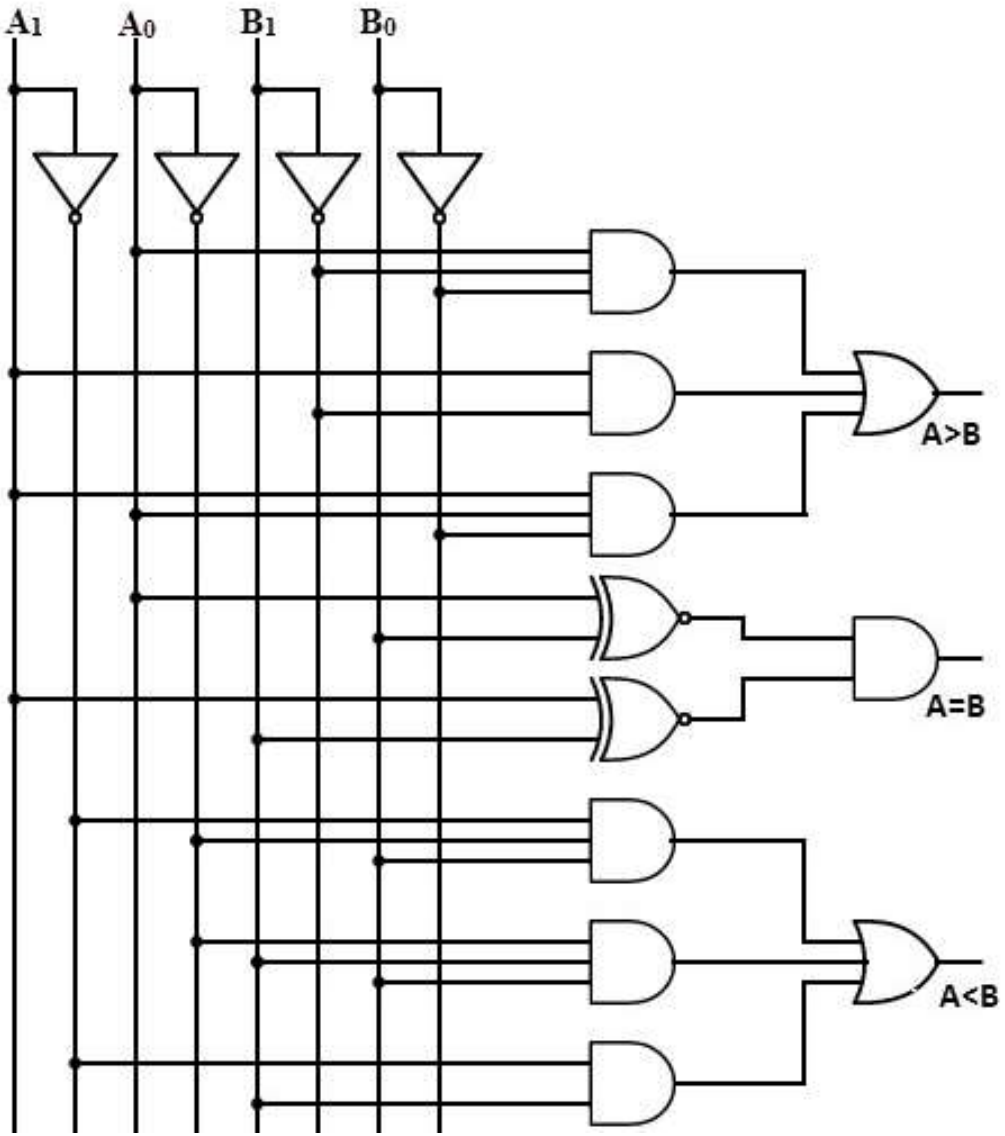

A>B
B1B0
A1A0
00
01
11
10
00 0 0 0 0
01 1 0 0 0
11 1 1 0 1
10 1 1 0 0


A <B
B1B0
A1A0
00
01
11
10
00 0 1 1 1
01 0 0 1 1
11 0 0 0 0
10 0 0 1 0

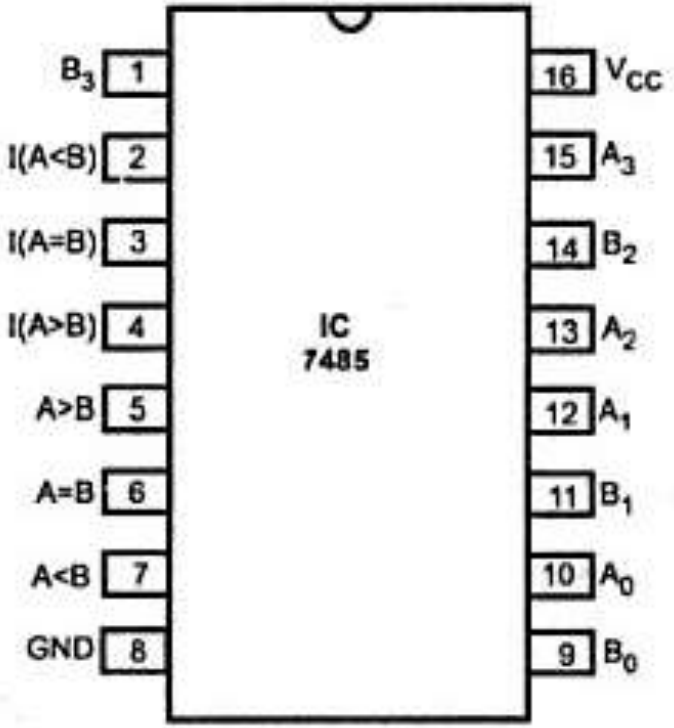

A =B
B1B0
A1A0
00
01
11
10
00 1 0 0 0
01 0 1 0 0
11 0 0 1 0
10 0 0 0 1

Equations

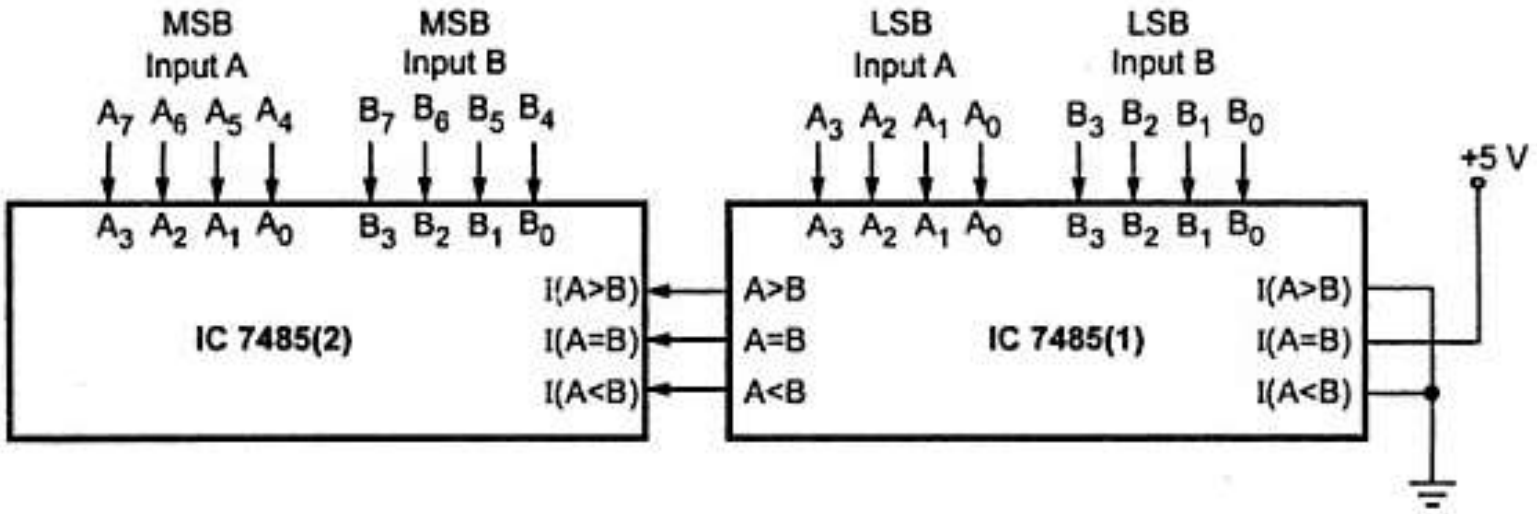
```
A>B:A1B1’ + A0B1’B0’ + A1A0B0’
A=B: A1’A0’B1’B0’ + A1’A0B1’B0 + A1A0B1B0 + A1A0’B1B0’
   : A1’B1’ (A0’B0’ + A0B0) + A1B1 (A0B0 + A0’B0’)
   : (A0B0 + A0’B0’) (A1B1 + A1’B1’)
   : (A0 Ex-Nor B0) (A1 Ex-Nor B1)
A<B:A1’B1 + A0’B1B0 + A1’A0’B0
```


A1
A0
B1
B0
A>B
A=B
A<B

Magnitude Comparator



Cascading


MSB
Input A
MSB
Input B
LSB
Input A
LSB
Input B
A7 A6 A5 A4
B7 B6 B5 B4
A3 A2 A1 A0
B3 B2 B1 B0
A3 A2 A1 A0
B3 B2 B1 B0
+5 V
I(A>B)
I(A=B)
I(A<B)
IC 7485(2)
A>B
A=B
A<B
IC 7485(1)
I(A>B)
I(A=B)
I(A<B)