

---

---

# Module 1:

# Fundamentals of Logic Design

---

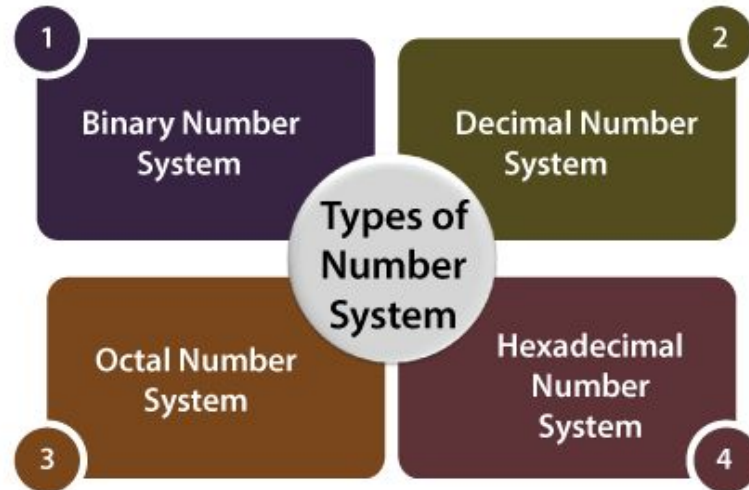
---

# Introduction to Number systems

- A number system is defined as the representation of numbers by using digits or other symbols in a consistent manner.
- The value of any digit in a number can be determined by a digit, its position in the number, and the base of the number system.
- In simple terms, for representing the information, we use the number system in the digital system.

# Types of Number systems

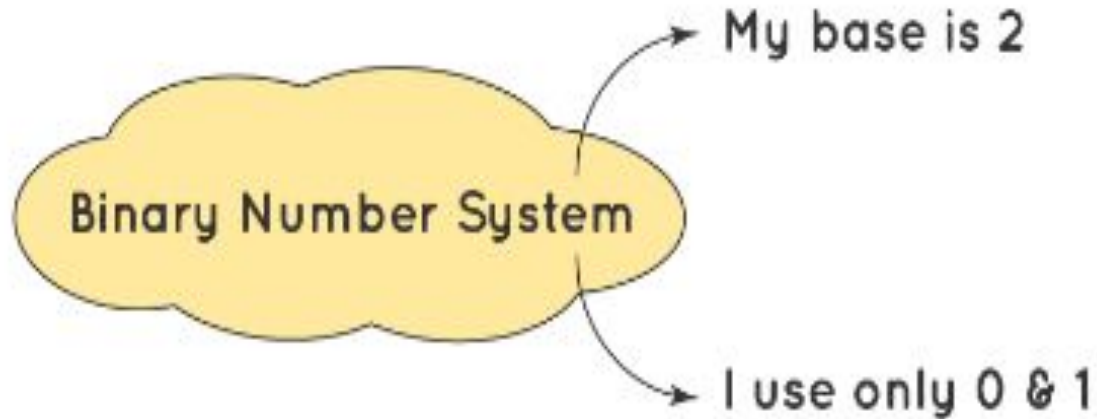
In the digital computer, there are various types of number systems used for representing information.



# Binary number system

- The binary number system uses only two digits: 0 and 1. The numbers in this system have a base of 2.
- Digits 0 and 1 are called bits and 8 bits together make a byte.
- The data in computers is stored in terms of bits and bytes.
- For example:  $1000_2$ ,  $111101_2$ ,  $1010101_2$

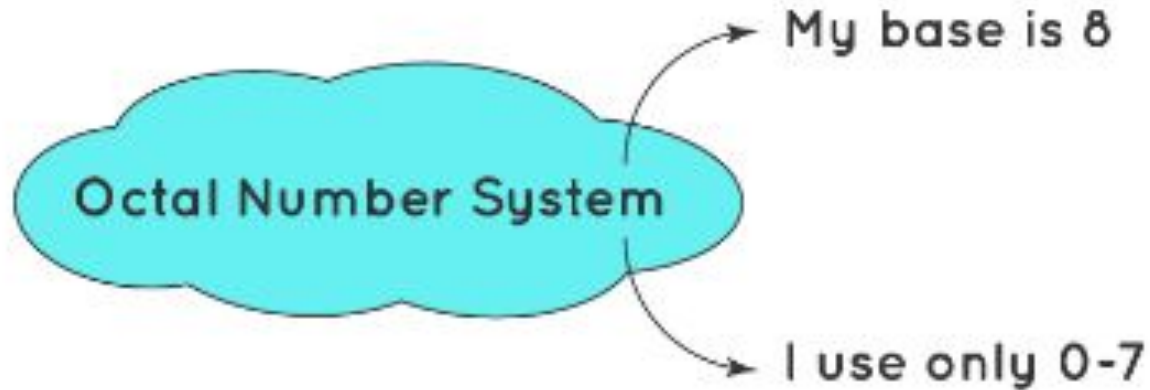
# Binary number system



# Octal Number System

- The octal number system uses eight digits: 0,1,2,3,4,5,6 and 7 with the base of 8.
- Digits like 8 and 9 are not included in the octal number system.
- Just as the binary, the octal number system is used in minicomputers but with digits from 0 to 7.
- For example:  $35_8$ ,  $23_8$ ,  $141_8$

# Octal Number System

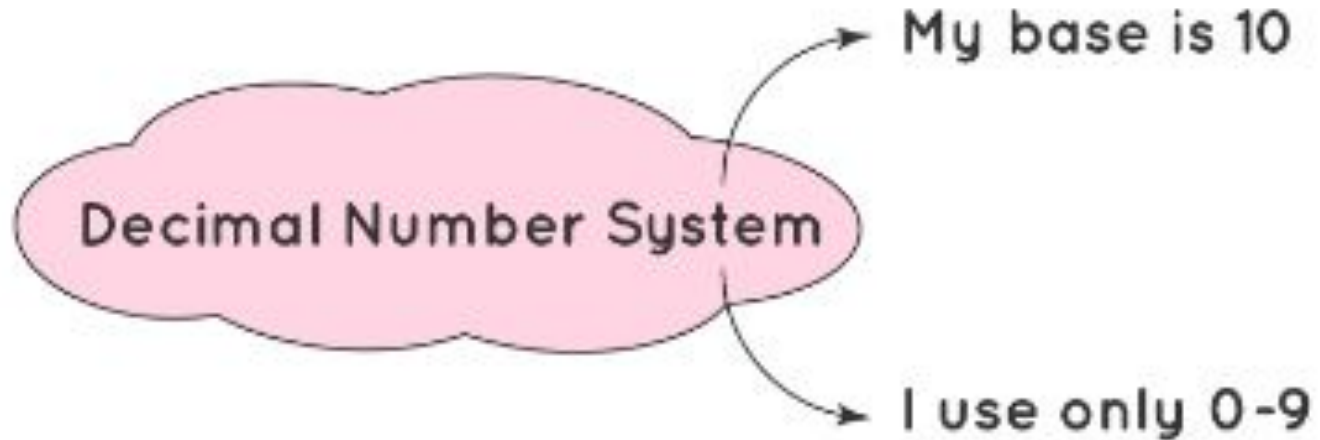


# Decimal Number System

- The decimal number system uses ten digits: 0,1,2,3,4,5,6,7,8 and 9 with the base number as 10.
- The decimal number system is the system that we generally use to represent numbers in real life.
- If any number is represented without a base, it means that its base is 10.
- For example:  $723_{10}$ ,  $32_{10}$ ,  $4257_{10}$



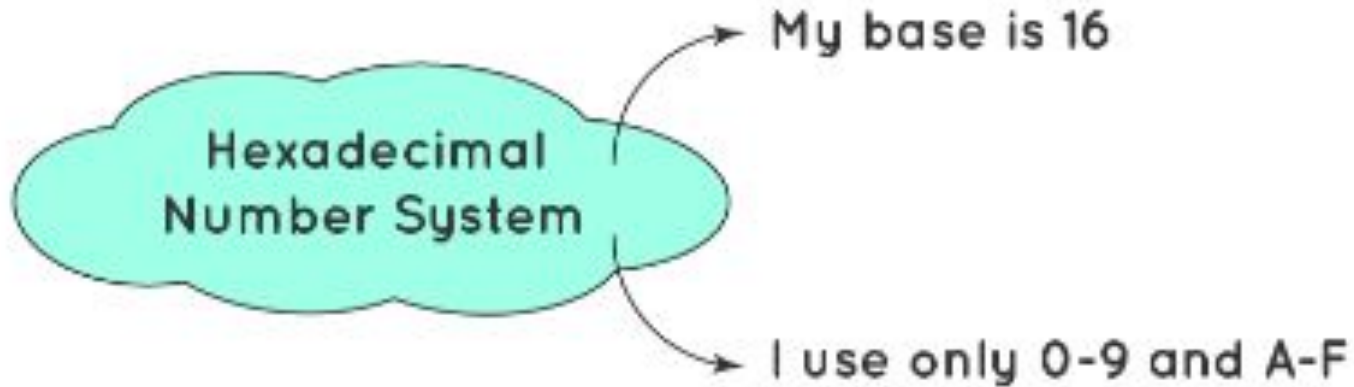
# Decimal Number System



# Hexadecimal Number System

- The hexadecimal number system uses sixteen digits/alphabets: 0,1,2,3,4,5,6,7,8,9 and A,B,C,D,E,F with the base number as 16.
- Here, A-F of the hexadecimal system means the numbers 10-15 of the decimal number system respectively.
- This system is used in computers to reduce the large-sized strings of the binary system.
- For example,  $7B3_{16}$ ,  $6F_{16}$ ,

# Hexadecimal Number System



# Conversion of Binary to Decimal Number System

**If the number is integer** - Multiply each digit of the given number by 2 from right to left, with the exponents of the base. The exponents should start with 0 and increase by 1 every time as we move from right to left i.e.,  $2^0$ ,  $2^1$ ,  $2^2$  and so on from right to left and add them.

# Conversion of Binary to Decimal Number System

## Example 1:

$$100111 = (1 \times 2^5) + (0 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$$

$$= (1 \times 32) + (0 \times 16) + (0 \times 8) + (1 \times 4) + (1 \times 2) + (1 \times 1)$$

$$= 32 + 0 + 0 + 4 + 2 + 1$$

$$= 39$$

$$\text{Thus, } 100111_2 = 39_{10}.$$

# Conversion of Binary to Decimal Number System

**Q.1: Convert the binary number 1001 to a decimal number.**

Solution: Given, binary number =  $1001_2$

Hence, using the binary to decimal conversion formula, we have:

$$1001_2 = (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$$

$$= 8 + 0 + 0 + 1$$

$$= (9)_{10}$$

**Q.2: Convert  $1101001_2$  into an equivalent decimal number.**

Solution: Using binary to decimal conversion method, we get;

$$(1101001)_2 = (1 \times 2^6) + (1 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$$

$$= 64 + 32 + 0 + 8 + 0 + 0 + 1$$

$$= (105)_{10}$$

# Conversion of Binary to Decimal Number System

If the number is fractional - for the fractional binary numbers to the right of the binary point, the weight of each digit becomes more negative giving:  $2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}$ , and so on as shown.

**Example 2:** Convert binary number 0.001(base 2) into decimal form

$$\begin{aligned} 0.001(\text{base } 2) &= 0 \times 2^0 + 0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\ &= 0 + 0 + 0 + 0.125 \\ &= 0.125 \end{aligned}$$

# Conversion of Binary to Decimal Number System

## Example 3:

$$(101.01)_2$$

$$= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$$

$$= 4 + 0 + 1 + 0 + 0.25$$

$$= (5.25)_{10}$$



# Conversion of Binary to Decimal Number System

**Example 4:**  $(11001011.01101)_2 = (?)_{10}$ .

**Soln. :**

$$\begin{aligned}(11001011)_2 &= 1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 \\ &\quad + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ &= 128 + 64 + 8 + 2 + 1 \\ &= (203)_{10}\end{aligned}$$

$$\begin{aligned}(0.01101)_2 &= 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5} \\ &= 0.25 + 0.125 + 0.03125 \\ &= (0.40625)_{10}\end{aligned}$$

$$\therefore (11001011.01101)_2 = (203.40625)_{10}$$



Scanned with CamScanner

# Conversion from decimal to binary number

Decimal numbers can be converted to binary by repeated division of the number by 2 while recording the remainder.

The remainders are to be read from bottom to top to obtain the binary equivalent.

# Conversion from decimal to binary number

**Example 1:** convert decimal number 43 to binary.

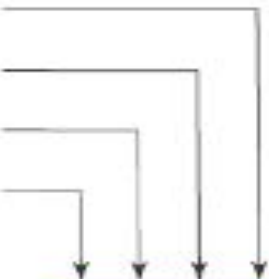
|   |    | Remainder |                              |
|---|----|-----------|------------------------------|
| 2 | 43 |           |                              |
| 2 | 21 | 1         | <div>MSB<br/>↑<br/>LSB</div> |
| 2 | 10 | 1         |                              |
| 2 | 5  | 0         |                              |
| 2 | 2  | 1         |                              |
| 2 | 1  | 0         |                              |
|   | 0  | 1         |                              |

The remainders are to be read from bottom to top to obtain the binary equivalent.  $43_{10} = 101011_2$

# Conversion from decimal to binary number

**Example 2:** convert the given decimal number 13 into a binary number.

**Step 1:** Divide the given number **13** repeatedly by 2 until you get '0' as the quotient

$$\begin{array}{lcl} 13 \div 2 = 6 & \text{(Remainder 1)} & \\ 6 \div 2 = 3 & \text{(Remainder 0)} & \\ 3 \div 2 = 1 & \text{(Remainder 1)} & \\ 1 \div 2 = 0 & \text{(Remainder 1)} & \end{array}$$


**Step 2:** Write the remainders in the reverse order **1 1 0 1**


$$\therefore 13_{10} = 1101_2$$

(Decimal)      (Binary)


# Conversion from decimal to binary number

**Example 3:** convert  $(10.625)_{10}$  to binary.

|   |    |   |
|---|----|---|
| 2 | 10 |   |
| 2 | 5  | 0 |
| 2 | 2  | 1 |
| 2 | 1  | 0 |
|   | 0  | 1 |



**0.625**

$$\begin{array}{l} 0.625 \times 2 = 1.250 \text{ ---- } 1 \\ 0.250 \times 2 = 0.500 \text{ ---- } 0 \\ 0.500 \times 2 = 1.0 \text{ ----- } 1 \\ 0 \times 2 \text{ end process} \end{array}$$



$$(10.625)_{10} = (1010.101)_2$$

**Practice:** Find the binary fraction equivalent of the decimal fraction:  $0.8125_{10}$


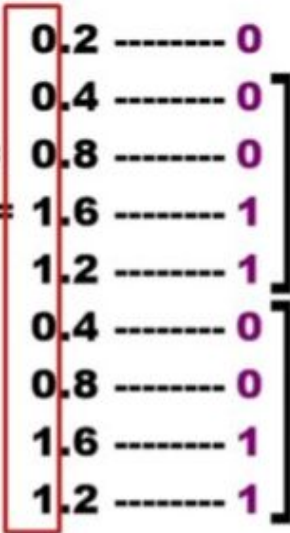
# Conversion from decimal to binary number

Example 4:  $(21.1)_{10} = (?)_2$

|   |    |   |
|---|----|---|
| 2 | 21 |   |
| 2 | 10 | 1 |
| 2 | 5  | 0 |
| 2 | 2  | 1 |
| 2 | 1  | 0 |
|   | 0  | 1 |



|                  |     |       |   |
|------------------|-----|-------|---|
| $0.1 \times 2 =$ | 0.2 | ----- | 0 |
| $0.2 \times 2 =$ | 0.4 | ----- | 0 |
| $0.4 \times 2 =$ | 0.8 | ----- | 0 |
| $0.8 \times 2 =$ | 1.6 | ----- | 1 |
| $0.6 \times 2 =$ | 1.2 | ----- | 1 |
| $0.2 \times 2 =$ | 0.4 | ----- | 0 |
| $0.4 \times 2 =$ | 0.8 | ----- | 0 |
| $0.8 \times 2 =$ | 1.6 | ----- | 1 |
| $0.6 \times 2 =$ | 1.2 | ----- | 1 |



Repeated

when Repeated process stopped

$$(21.1)_{10} = (10101.000110011)_2$$

# Conversion from decimal to octal number

**Example 1:**

|   |     |   |
|---|-----|---|
| 8 | 540 |   |
| 8 | 67  | 4 |
| 8 | 8   | 3 |
| 8 | 1   | 0 |
|   | 0   | 1 |

$(540)_{10} = (1034)_8$

**Practice:** Convert  $(127)_{10}$  to Octal.

Convert  $100_{10}$  to octal.

# Conversion from decimal to octal number

## Example 2:

**29.30**

|   |    |   |
|---|----|---|
| 8 | 29 |   |
| 8 | 3  | 5 |
|   | 0  | 3 |

$$(29.30)_{10} = (35.23146)_8$$

$$0.30 \times 8 = 2.4$$

$$0.40 \times 8 = 3.2$$

$$0.20 \times 8 = 1.6$$

$$0.60 \times 8 = 4.8$$

$$0.80 \times 8 = 6.4$$

$$0.40 \times 8 = 3.2$$

$$0.20 \times 8 = 1.6$$

$$0.60 \times 8 = 4.8$$

$$0.80 \times 8 = 6.4$$

Stop

Repeated



# Conversion from octal to decimal number

To obtain the decimal equivalent of an octal number, individual digits of octal number should be multiplied by powers of 8 starting with rightmost digit multiplied by  $8^0$ , second last digit multiplied by  $8^1$ , third last digit multiplied by  $8^2$  and so on up to the leftmost digit. For digits after the decimal point, the leftmost digit should be multiplied by  $8^{-1}$  and the next digit by  $8^{-2}$  and so on up to the rightmost digit.

**Example:**

**$(127.34)_8$**

$$= 1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 3 \times 8^{-1} + 4 \times 8^{-2}$$

$$= 64 + 16 + 7 + 0.375 + 0.0625$$

$$= (87.4375)_{10}$$

# Conversion from octal to binary number

**1 2 . 5**  
    /   /   /  
001 010 101  
 $(12.5)_8 = (001010.101)_2$

**7 2 1**  
  /   /   /  
111 010 001  
 $(721)_8 = (111010001)_2$

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

# Conversion from Decimal to Hexadecimal number

Repeatedly divide the decimal number by 16 until the quotient becomes less than 16 and record all the remainders. The remainders should be written bottom to upwards to get the hexadecimal equivalent of the decimal number.

Here,

- **10 = A**
- **11 = B**
- **12 = C**
- **13 = D**
- **14 = E**
- **15 = F**

# Conversion from Decimal to Hexadecimal number

**Example:**

**$(1973)_{10}$**

|    |      |           |
|----|------|-----------|
| 16 | 1973 |           |
| 16 | 123  | <b>5</b>  |
|    | 7    | <b>11</b> |

**Here, 11 = B**

**$(1973)_{10} = (7B5)_{16}$**

# Binary Arithmetic

# Binary Arithmetic

- Computer works only with binary number system.
- Although a computer accepts decimal input provides decimal output, but internally it works on binary.
- Operations to be done on binary data:
  1. Binary addition
  2. Binary subtraction

# Binary Addition

Truth table to add two binary numbers:

| Input A | Input B | Sum (S)<br>A+B | Carry (C) |
|---------|---------|----------------|-----------|
| 0       | 0       | 0              | 0         |
| 0       | 1       | 1              | 0         |
| 1       | 0       | 1              | 0         |
| 1       | 1       | 0              | 1         |





# Binary Subtraction

Truth table to subtract two binary numbers:

| Input A | Input B | Subtract<br>(S)<br>A-B | Borrow (B) |
|---------|---------|------------------------|------------|
| 0       | 0       | 0                      | 0          |
| 0       | 1       | 0                      | 1          |
| 1       | 0       | 1                      | 0          |
| 1       | 1       | 0                      | 0          |

# Binary Subtraction-Example

Subtract: 1100 - 1010

$$\begin{array}{r} \phantom{(-)} \phantom{1} \phantom{0} \phantom{1} \phantom{0} \\ \phantom{(-)} \phantom{1} \phantom{0} \phantom{1} \phantom{0} \\ (-) \phantom{1} \phantom{0} \phantom{1} \phantom{0} \\ \hline \phantom{(-)} \phantom{1} \phantom{0} \phantom{1} \phantom{0} \\ \phantom{(-)} \phantom{1} \phantom{0} \phantom{1} \phantom{0} \end{array}$$

← borrow

| Input A | Input B | Subtract<br>(S)<br>A-B | Borrow (B) |
|---------|---------|------------------------|------------|
| 0       | 0       | 0                      | 0          |
| 0       | 1       | 0                      | 1          |
| 1       | 0       | 1                      | 0          |
| 1       | 1       | 0                      | 0          |

# 1's Complement and 2's Complement

1s complement and 2s complement are way of representing the signed binary numbers.

In general, the binary number can be represented in two ways.

1. **Unsigned Binary Numbers:** Using unsigned binary number representation, only positive binary numbers can be represented.
2. **Signed Binary Numbers:** Using signed binary number representation both positive and negative numbers can be represented.

# 1's Complement and 2's Complement

**2. Signed Binary Numbers:** In signed binary number representation, **MSB of the number is a sign bit.** For **positive** numbers, the sign bit is **0** and for **negative** number, the sign bit is **1**. There are three different ways the signed binary numbers can be represented.

1. Signed Magnitude Form
2. 1's Complement Form
3. 2's Complement Form

**0 1 0 1**

**1 0 1 1**

first bit is sign bit:  
if sign bit is 0, number is  
positive.  
if sign bit is 1, number is  
negative.

# 1's Complement Representation

**1's complement** of a binary number is another binary number obtained by toggling all bits in it, i.e., transforming the 0 bit to 1 and the 1 bit to 0.

## Example:

1's complement of "0111" is "1000"

1's complement of "1100" is "0011"

# 1's Complement Representation

In this, the representation of the positive number is same as the negative number. But the representation of the negative number is different.

**Example:** if we want to represent -34 in 8-bit 1's complement form, then first write the positive number (+34). And invert all 1s in that number by 0s and 0s by 1s in that number. The corresponding inverted number represents the -34 in 1's complement form. It is also called 1s complement of the number +34.

# 1's Complement Representation Example

To represent **-34** in 1's complement form

$$+ 34 = \begin{matrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \end{matrix}$$

$$- 34 = \begin{matrix} 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{matrix} \quad (\text{1's complement of } + 34)$$

# 1's Complement Representation

1. Write the 1's complement of the subtrahend.
2. Then add the 1's complement subtrahend with the minuend.
3. If the result has a carryover, then add that carry over in the least significant bit.
4. If there is no carryover, then take the 1's complement of the resultant, and it is negative.



# 1's Complement Representation Example

**Subtract  $(1010)_2$  from  $(1111)_2$  using 1's complement method.**

**Step-1:** Find the 1's complement of 1010. The required 1's complement will be 0101.

**Step-2:** In this step, we need to add the value calculated in step-1 to 1111. This is shown below.

$$\begin{array}{r} \phantom{+} \phantom{0} 1 \phantom{0} 1 \phantom{0} 1 \phantom{0} 1 \\ + \phantom{0} 0 \phantom{0} 1 \phantom{0} 0 \phantom{0} 1 \\ \hline \text{Carry } \textcircled{1} \phantom{0} 0 \phantom{0} 1 \phantom{0} 0 \phantom{0} 0 \\ \phantom{+} \phantom{0} \phantom{0} + \phantom{0} \phantom{0} 1 \\ \hline \boxed{0 \phantom{0} 1 \phantom{0} 0 \phantom{0} 1} \text{ Answer} \end{array}$$

# 1's Complement Representation Example

**Subtract  $(1010)_2$  from  $(1000)_2$  using 1's complement method.**  
=> The 1's complement of  $(1010)_2$  is  $(0101)_2$ . Now, we will add this with the smaller number and finally take 1's complement of the result to get the answer.

|                  |       |   |   |   |                |
|------------------|-------|---|---|---|----------------|
|                  | 1     | 0 | 0 | 0 |                |
|                  |       |   |   |   | (+)            |
| 1's complement → | 0     | 1 | 0 | 1 |                |
|                  | <hr/> |   |   |   |                |
|                  | 1     | 1 | 0 | 1 |                |
|                  | <hr/> |   |   |   |                |
|                  |       |   |   |   | 1's complement |
|                  |       |   |   |   | ↓              |
|                  | 0     | 0 | 1 | 0 | <b>Answer</b>  |

# 1's Complement Representation Example

**Practice Example 1:**  $10101 - 00111$

**Practice Example 2:**  $10101 - 10111$

# 2's Complement Representation

2's complement of binary number is 1's complement of given number plus 1 to the least significant bit (LSB).

**For example:**

2's complement of binary number 10010 is  $(01101) + 1 = 01110$ .

# 2's Complement Representation

**Example-1** – Find 2's complement of binary number 10101110.

Simply invert each bit of given binary number, which will be 01010001. Then add 1 to the LSB of this result, i.e.,  $01010001 + 1 = 01010010$  which is answer.

**Example-2** – Find 2's complement of binary number 10001.001.

Simply invert each bit of given binary number, which will be 01110.110 Then add 1 to the LSB of this result, i.e.,  $01110.110 + 1 = 01110.111$  which is answer.

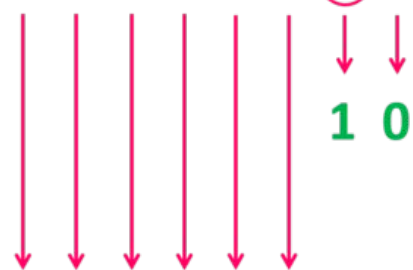
# 2's Complement Representation

But the representation of the **negative number** is different.

**Example:** if we want to represent -34 in 2's complement form then

To represent -34 in 2's complement form

+ 34 = 0 0 1 0 0 0 1 0



Copy all bits till first '1' is encountered in the number

1 1 0 1 1 1 1 0

Invert all 1s with 0s and 0s with 1s then after

- 34 = 1 1 0 1 1 1 1 0 2's Complement of +34

# 2's Complement Representation

The second way of representing -34 in 2's complement form is

To represent **-34** in 2's complement form

$$\begin{array}{r} +34 = 00100010 \\ \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\ 11011101 \quad \text{(1's complement of +34)} \\ + \qquad \qquad \qquad 1 \\ \hline -34 = 11011110 \quad \text{(2's complement of +34)} \end{array}$$

# 2's Complement Representation

**Example 1:**  $7 - 2 = ?$

$\Rightarrow 7 + (-2)$

2  $\rightarrow$  0 0 1 0

1 1 1 0 (2's complement of 2 i.e. -2)

7  $\rightarrow$  0 1 1 1

(-2)  $\rightarrow$  +1 1 1 0

0 1 0 1 (5)

*Sign bit is 0, so the number is positive*



## 2's Complement Representation

**Example 2:**  $3 - 6 = ?$

$\Rightarrow 3 + (-6)$

6  $\rightarrow$  0 1 1 0

1 0 1 0 (2's complement of 6 i.e. -6)

3  $\rightarrow$  0 0 1 1

(-6)  $\rightarrow$   $\begin{array}{r} +1\ 0\ 1\ 0 \\ \hline 1\ 1\ 0\ 1 \end{array}$

*Sign bit is 1, so the number is negative*

## Practice Example

Example 1: Subtraction of 01100-00011 using 2's complement method

Example 2: Perform the operation  $5 - 4$  using 2's complement.

## Practice Example

**Example1: Subtraction of 01100-00011 using 2's complement method**

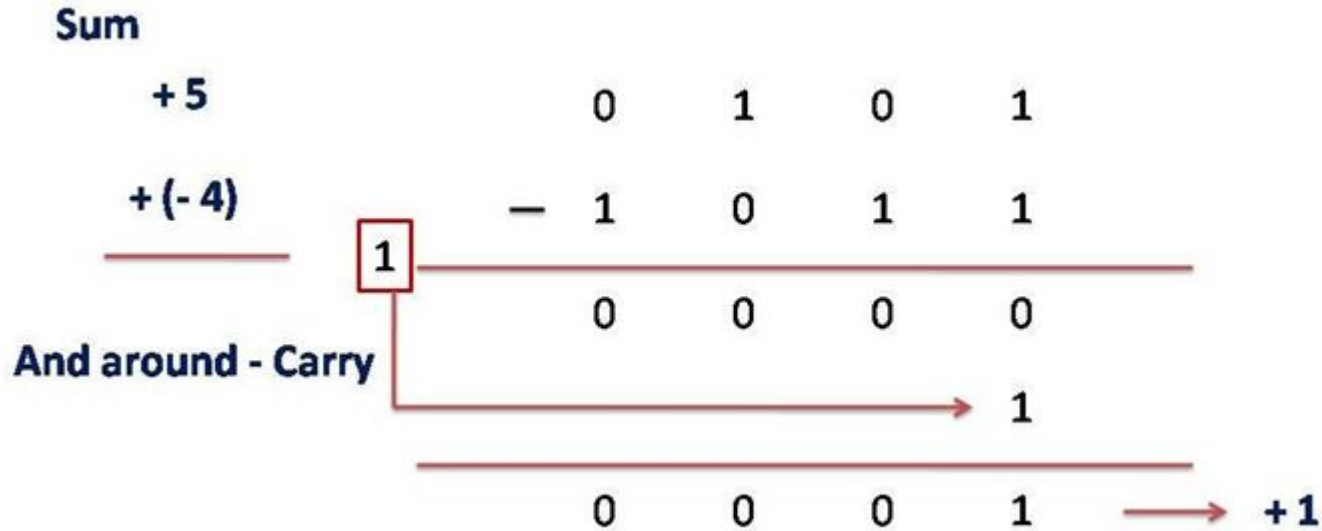
$$\begin{array}{rcl} 01100 & = & 01100 \\ - 00011 & = & + 11101 \text{ (2's complement)} \\ \hline & & 1\ 01001 = +9 \\ & \uparrow & \\ & \text{Ignore} & \end{array}$$

If a last carry is produced discard the carry and the answer is provided by the remaining bits that is positive that is,

$$(1001)_2 = (+9)_{10}.$$

# Practice Example

**Example2:** Perform the operation 5- 4 using 2's complement.



# Boolean Algebra

It is a mathematical system that defines a series of logical operations(AND,OR,NOT) performed on a set of variable (a,b,c,...).

In expression,  $Y=A+1$

Y is a function having value 0 or 1.

A is a variable having value 0 or 1.

1 is a constant.

The three important Boolean operators are:

AND (Conjunction)

OR (Disjunction)

NOT (Negation)

# Boolean Algebra

**Complement:** It is represented by “bar” over a letter. For example, complement of A will be  $\bar{A}$

So, If  $A = 0$  ,  $\bar{A} = 1$

If  $A = 1$  ,  $\bar{A} = 0$

**Boolean function:** Boolean expression is used to describe Boolean functions.

Example:

$F(A, B, C, D)$

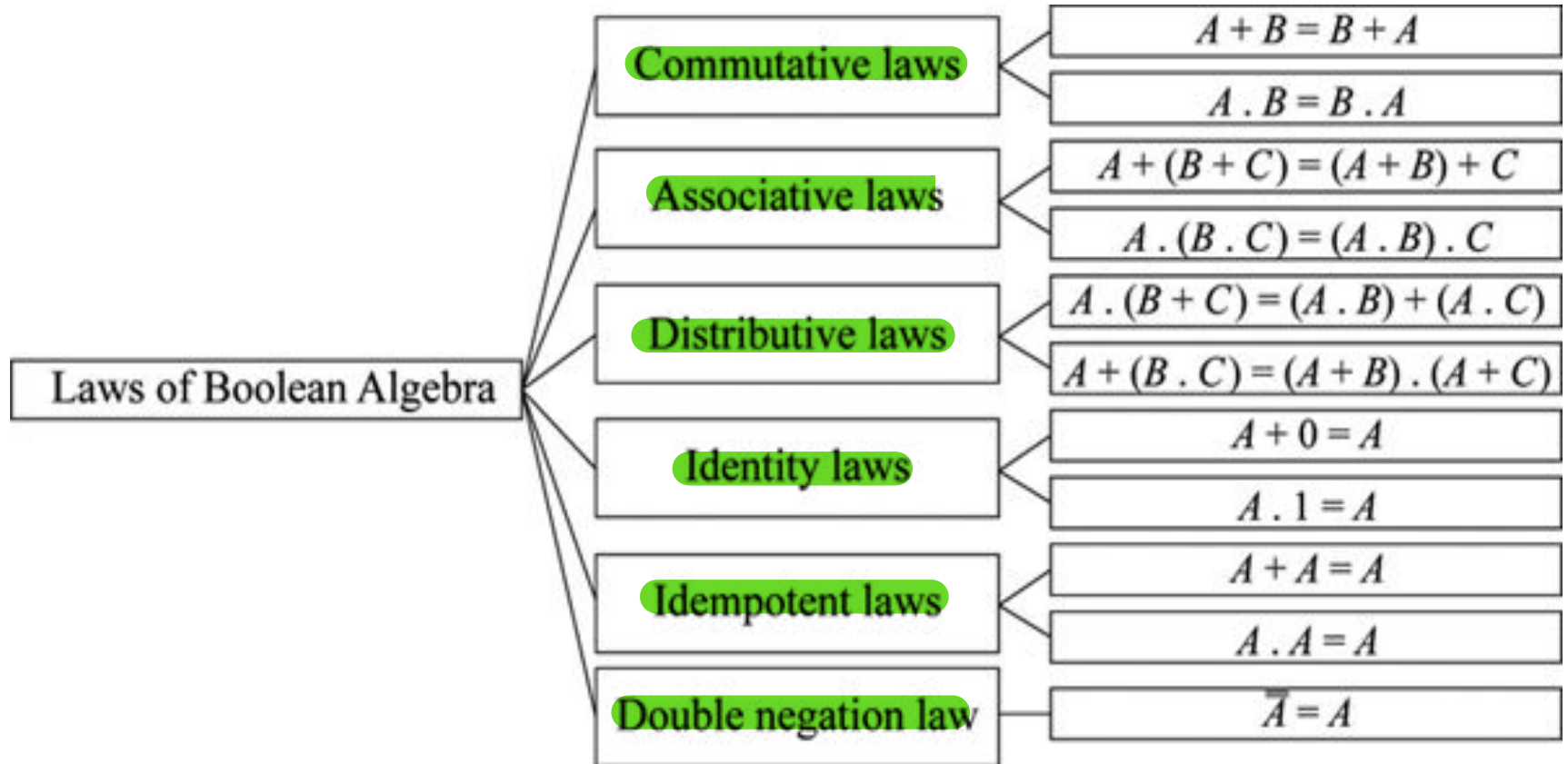
=

$A + \overline{BC} + ADC$

Boolean Function

Boolean Expression

# Boolean Laws



# Boolean Laws

## Basic Rules of Boolean Algebra

|                      |                                  |
|----------------------|----------------------------------|
| 1. $A + 0 = A$       | 7. $A \cdot A = A$               |
| 2. $A + 1 = 1$       | 8. $A \cdot \bar{A} = 0$         |
| 3. $A \cdot 0 = 0$   | 9. $\overline{\overline{A}} = A$ |
| 4. $A \cdot 1 = A$   | 10. $A + AB = A$                 |
| 5. $A + A = A$       | 11. $A + \bar{A}B = A + B$       |
| 6. $A + \bar{A} = 1$ | 12. $(A + B)(A + C) = A + BC$    |

$$A + AB$$



$$A(1 + B)$$



$$A(1)$$



$$A$$

## DeMorgan's Theorem

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

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



## Example 1

$$A + A \cdot B = A$$

Proof Steps

$$\begin{aligned} & A + A \cdot B \\ &= A \cdot 1 + A \cdot B \\ &= A \cdot (1 + B) \\ &= A \cdot 1 \\ &= A \end{aligned}$$

(Absorption Theorem)

Justification

Identity element:  $A \cdot 1 = A$

Distributive

$$1 + B = 1$$

Identity element

## Example 2

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

$$(A + B)(A + C) = AA + AC + AB + BC \text{ (Distributive law)}$$

$$= A + AC + AB + BC \quad (AA = A)$$

$$= A(1 + C) + AB + BC \quad (1 + C = 1)$$

$$= A \cdot 1 + AB + BC$$

$$= A(1 + B) + BC \quad (1 + B = 1)$$

$$= A \cdot 1 + BC \quad (A \cdot 1 = A)$$

$$= A + BC$$

## Sum of Product (SOP)

- The sum-of-products (**SOP**) form is a method (or form) of **simplifying the Boolean expressions of logic gates**.
- Sum and product derived from the symbolic representations of the OR and AND functions.
- OR (+) , AND ( . ) , addition and multiplication.

$$f(A,B,C) = ABC + A'BC'$$

Sum

Product terms

# Product of Sum (POS)

- When two or more sum terms are multiplied by a Boolean OR operation.
- Sum terms are defined by using OR operation and the product term is defined by using AND operation.

$$f(A,B,C) = (A'+B) \cdot (B+C')$$

Product

Sum terms

## Canonical form

- In SOP or POS form, all individual terms do not involve all literals.
- For example  $AB + A'BC$  the first product term do not contain literal C.
- If each term in SOP or POS contain all literals then the expression is known as standard or canonical form.

## Standard SOP

Convert the following Boolean expression into standard SOP form:

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

$$A\bar{B}C = A\bar{B}C(D + \bar{D}) = \boxed{A\bar{B}CD + A\bar{B}C\bar{D}}$$

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

$$\bar{A}\bar{B}C(D + \bar{D}) + \bar{A}\bar{B}\bar{C}(D + \bar{D}) = \boxed{\bar{A}\bar{B}CD + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D}}$$

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

## Standard POS

Convert the following Boolean expression into standard POS form:

$$(A + \bar{B} + C)(\bar{B} + C + \bar{D})(A + \bar{B} + \bar{C} + D)$$

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

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

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

$$(A + \bar{B} + C + D)(A + \bar{B} + C + \bar{D})(A + \bar{B} + C + \bar{D})(\bar{A} + \bar{B} + C + \bar{D})(A + \bar{B} + \bar{C} + D)$$

# Minterm and Maxterm

Minterm is **product term** in which the Boolean variable is either normal or in complemented form.

Each individual term in standard SOP form is called **Minterm**.

Maxterm is a **sum term** in which the Boolean variable is either normal or in complemented form.




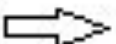

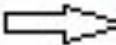
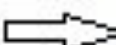
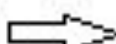
Each individual term in standard POS form is called **Maxterm**.



# Minterm and Maxterm

- Each individual term in the POS form is called **Maxterm**.
- Each individual term in the SOP form is called **Minterm**.
- In **Minterm**, we look for the functions where the output results is "1".
- while in **Maxterm** we look for function where the output results is "0".
- We perform **Sum of minterm** also known as **Sum of products (SOP)**.
- We perform **Product of Maxterm** also known as **Product of sum (POS)**.

# Minterm and Maxterm example

| A | B | C | Output |  |
|---|---|---|--------|--|
| 0 | 0 | 0 | 0      |  $A+B+C$  |
| 0 | 0 | 1 | 0      |  $A+B+C'$ |
| 0 | 1 | 0 | 0      |  $A+B'+C$ |
| 0 | 1 | 1 | 1      |  $A'BC$   |
| 1 | 0 | 0 | 0      |  $A'+B+C$ |
| 1 | 0 | 1 | 1      |  $AB'C$   |
| 1 | 1 | 0 | 1      |  $ABC'$   |
| 1 | 1 | 1 | 1      |  $ABC$    |

SOP Expression:  $A'BC+AB'C+ABC'+ABC$

POS Expression:  $(A+B+C)(A+B+C')(A+B'+C)(A'+B+C)$

# Minterm and Maxterm for 2 variable

Two variable minterms and maxterms.

| x | y | Index | Minterm                 | Maxterm                   |
|---|---|-------|-------------------------|---------------------------|
| 0 | 0 | 0     | $m_0 = \bar{x} \bar{y}$ | $M_0 = x + y$             |
| 0 | 1 | 1     | $m_1 = \bar{x} y$       | $M_1 = x + \bar{y}$       |
| 1 | 0 | 2     | $m_2 = x \bar{y}$       | $M_2 = \bar{x} + y$       |
| 1 | 1 | 3     | $m_3 = x y$             | $M_3 = \bar{x} + \bar{y}$ |

## Minterm and Maxterm for 3 variable

| x | y | z | Index | Minterm                         | Maxterm                             |
|---|---|---|-------|---------------------------------|-------------------------------------|
| 0 | 0 | 0 | 0     | $m_0 = \bar{x} \bar{y} \bar{z}$ | $M_0 = x + y + z$                   |
| 0 | 0 | 1 | 1     | $m_1 = \bar{x} \bar{y} z$       | $M_1 = x + y + \bar{z}$             |
| 0 | 1 | 0 | 2     | $m_2 = \bar{x} y \bar{z}$       | $M_2 = x + \bar{y} + z$             |
| 0 | 1 | 1 | 3     | $m_3 = \bar{x} y z$             | $M_3 = x + \bar{y} + \bar{z}$       |
| 1 | 0 | 0 | 4     | $m_4 = x \bar{y} \bar{z}$       | $M_4 = \bar{x} + y + z$             |
| 1 | 0 | 1 | 5     | $m_5 = x \bar{y} z$             | $M_5 = \bar{x} + y + \bar{z}$       |
| 1 | 1 | 0 | 6     | $m_6 = x y \bar{z}$             | $M_6 = \bar{x} + \bar{y} + z$       |
| 1 | 1 | 1 | 7     | $m_7 = x y z$                   | $M_7 = \bar{x} + \bar{y} + \bar{z}$ |

## Example 1: Express the Boolean function $F=A+BC$ as minterm

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

By complement law  $A+A'=1$ . Adding all the input variables in the product terms we get

$$F = (AB+AB')(C+C') + ABC + A'BC$$

$$F = ABC + ABC' + AB'C + ABC' + AB'C' + ABC + A'BC$$

There are two  $ABC$ s and  $ABC'$ . So let us consider one  $ABC$  and  $ABC'$

$$F = ABC + ABC' + AB'C + AB'C' + A'BC$$

In minterm the '1' input is written as such and the input '0' is complimented.

$$F = 111 + 110 + 101 + 100 + 011$$

$$F = m_7 + m_6 + m_5 + m_4 + m_3$$

$$F = \sum m(3, 4, 5, 6, 7)$$

## Example 2: Express the Boolean function $F=A+BC$ as maxterm

$$F=A+(BC)$$

After removing the bracket and expanding we get

$$F= (A+B).(A+C)$$

By complement law  $CC'=BB'=1$ . So after adding all input terms in the expression we get,

$$F= (A+B+CC').(A+C+BB')$$

$$F= (A+B+C).(A+B+C').(A+C+B).(A+C+B')$$

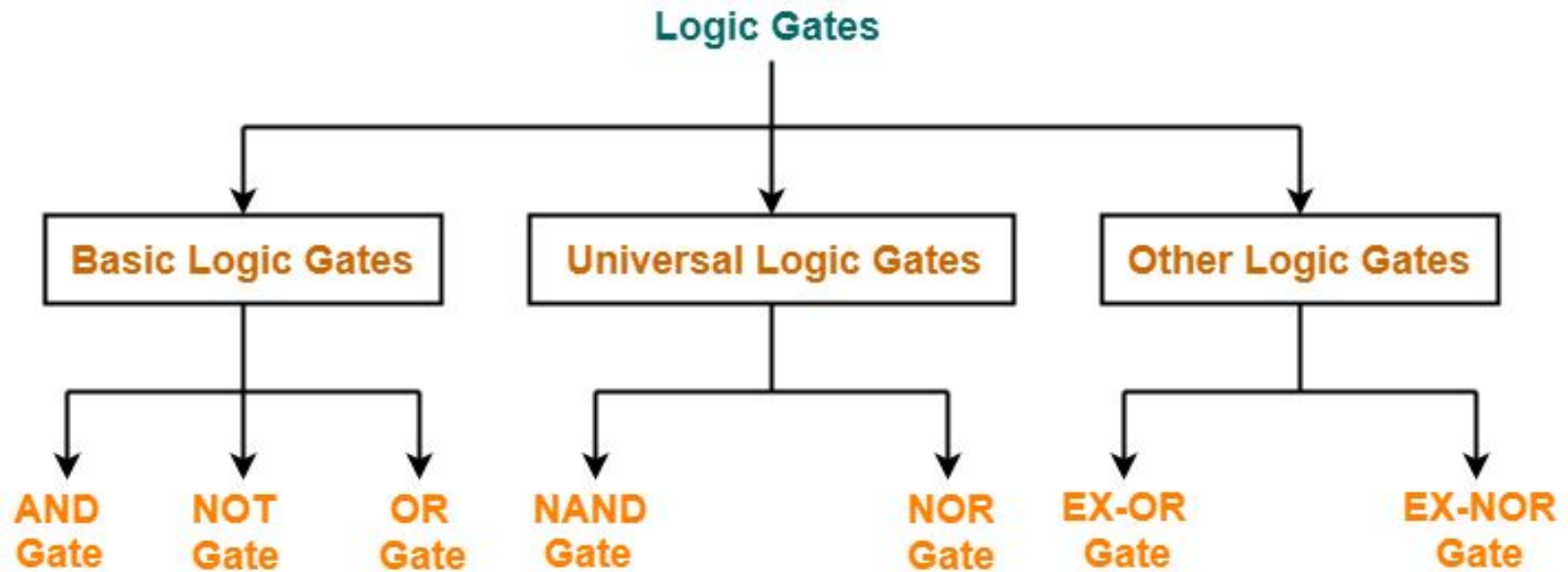
There are two  $(A+B+C)$ s. So let us consider one  $(A+B+C)$

$$F= (A+B+C).(A+B+C').(A+B'+C)$$

In Maxterm the input '0' is written as such and the input '1' is complemented.

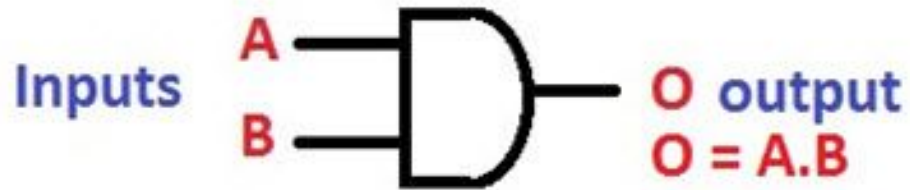
$$F=(0+0+0).(0+0+1).(0+1+0) = M_0.M_1.M_2 = \pi M(0,1,2)$$

# Logic gates



Types of Logic Gates

# Logic gates: AND Gate



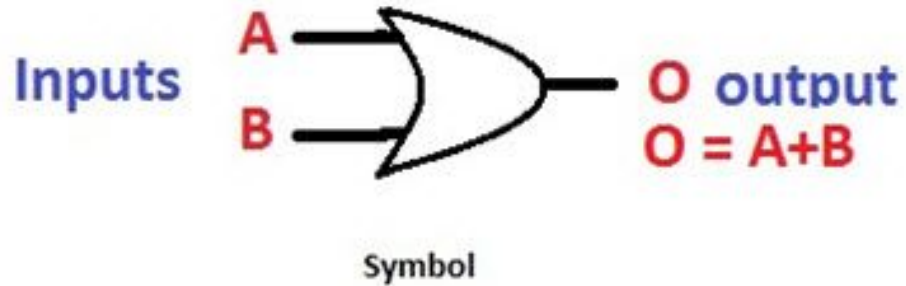
Symbol

| Inputs |   | Output |
|--------|---|--------|
| A      | B | O      |
| 0      | 0 | 0      |
| 0      | 1 | 0      |
| 1      | 0 | 0      |
| 1      | 1 | 1      |

Truth table



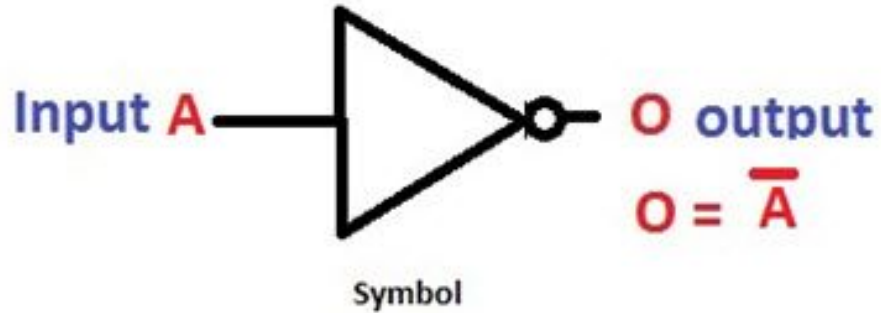
# Logic gates: OR Gate



| Inputs |   | Output |
|--------|---|--------|
| A      | B | O      |
| 0      | 0 | 0      |
| 0      | 1 | 1      |
| 1      | 0 | 1      |
| 1      | 1 | 1      |

Truth table

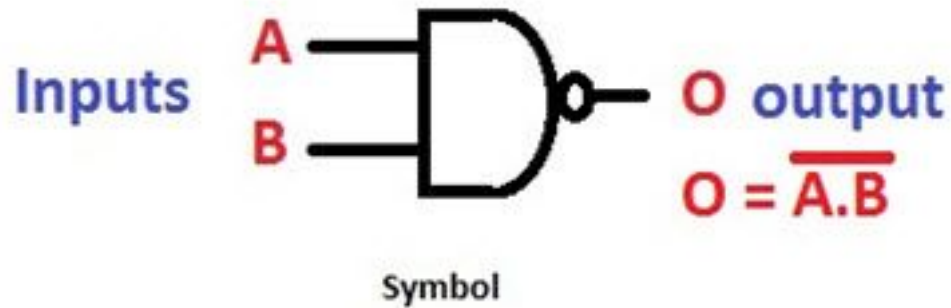
# Logic gates: NOT Gate



| Inputs   | Output   |
|----------|----------|
| <b>A</b> | <b>O</b> |
| 0        | 1        |
| 1        | 0        |

Truth table

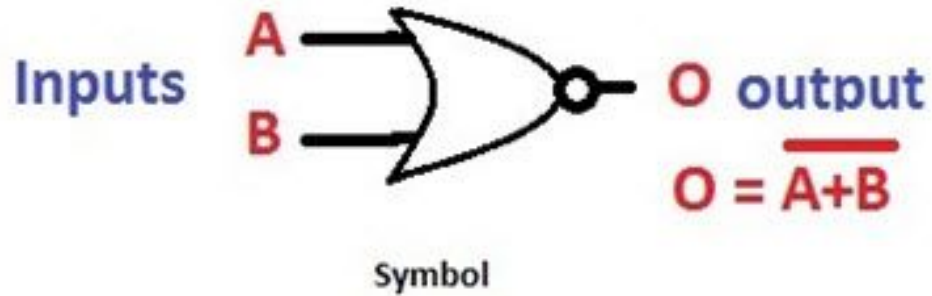
# Logic gates: NAND Gate



| Inputs |   | Output |
|--------|---|--------|
| A      | B | O      |
| 0      | 0 | 1      |
| 0      | 1 | 1      |
| 1      | 0 | 1      |
| 1      | 1 | 0      |

Truth table

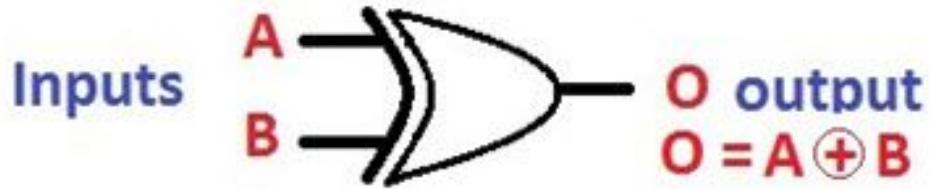
# Logic gates: NOR Gate



| Inputs |   | Output |
|--------|---|--------|
| A      | B | O      |
| 0      | 0 | 1      |
| 0      | 1 | 0      |
| 1      | 0 | 0      |
| 1      | 1 | 0      |

Truth table

# Logic gates: Exclusive-OR Gate

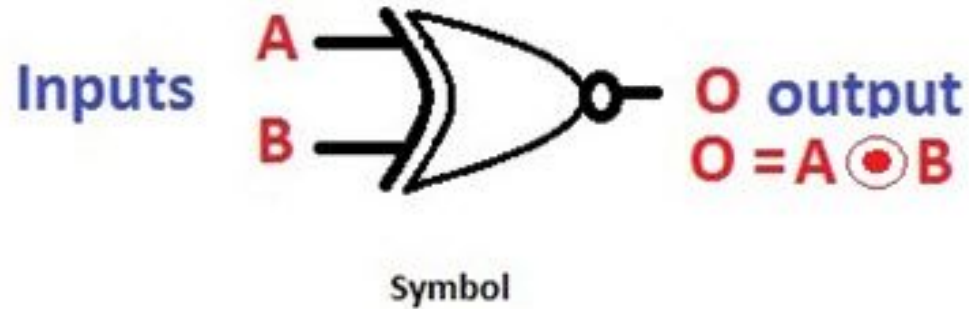


Symbol

| Inputs |   | Output |
|--------|---|--------|
| A      | B | O      |
| 0      | 0 | 0      |
| 0      | 1 | 1      |
| 1      | 0 | 1      |
| 1      | 1 | 0      |

Truth table

# Logic gates: Exclusive-NOR Gate



| Inputs |   | Output |
|--------|---|--------|
| A      | B | O      |
| 0      | 0 | 1      |
| 0      | 1 | 0      |
| 1      | 0 | 0      |
| 1      | 1 | 1      |

Truth table

# Adder

- An adder is a device that will add together two bits and give the result as the output.
- The bits being added together are called the "addends".
- Adders can be concatenated in order to add together two binary numbers of an arbitrary length.
- There are two kinds of adders - half adders and full adders.

# Half adder

- Half Adder is the digital logic circuit that is used to implement the **binary addition**.

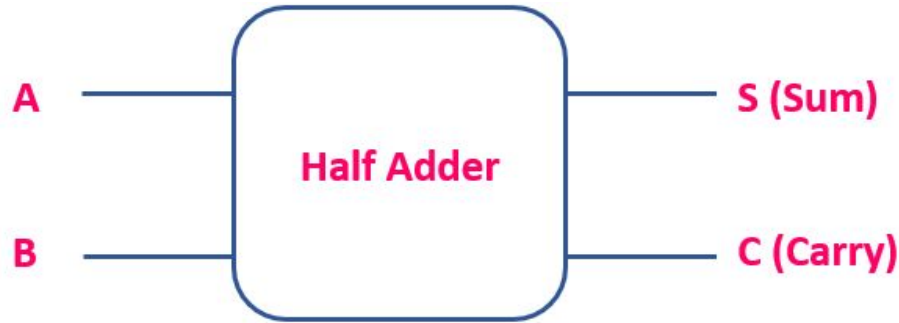


Fig: Block diagram



# Half adder circuit

- It adds the two bits and generates the sum bit (S) and carry bit (C) as an output.

| Inputs |   | Outputs |       |
|--------|---|---------|-------|
| A      | B | Sum     | Carry |
| 0      | 0 | 0       | 0     |
| 0      | 1 | 1       | 0     |
| 1      | 0 | 1       | 0     |
| 1      | 1 | 0       | 1     |

Fig: Truth Table

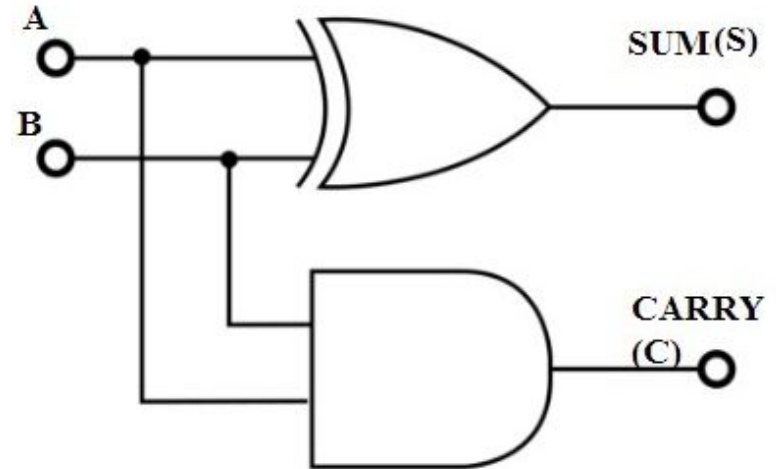


Fig: Half adder circuit

# Half adder

$$S = A \oplus B$$

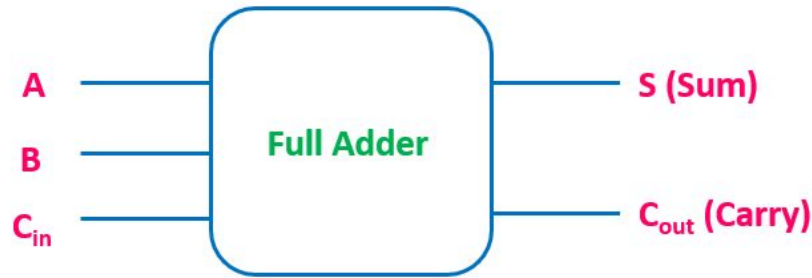
$$C = A \cdot B$$

The main disadvantage of this circuit is that it can only add two inputs and if there is any carry, it is neglected. Thus, the process is incomplete.

To overcome this difficulty Full Adder is designed.

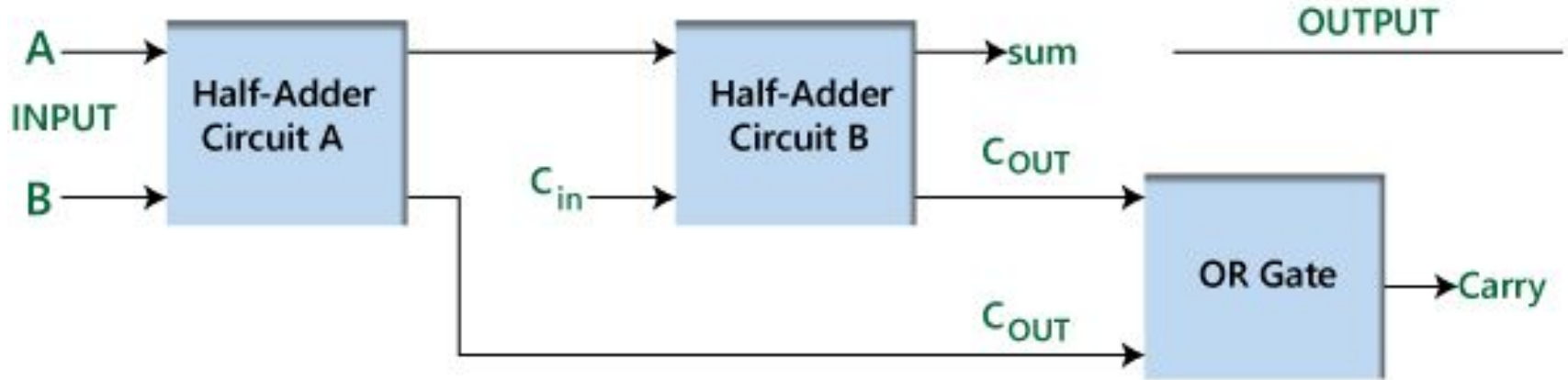
# Full adder

The full adder is the combinational circuit that adds the two bits along with the incoming carry ( $C_{in}$ ) and generates the sum bit ( $S$ ) and an outgoing carry bit ( $C_{out}$ ) as an output.

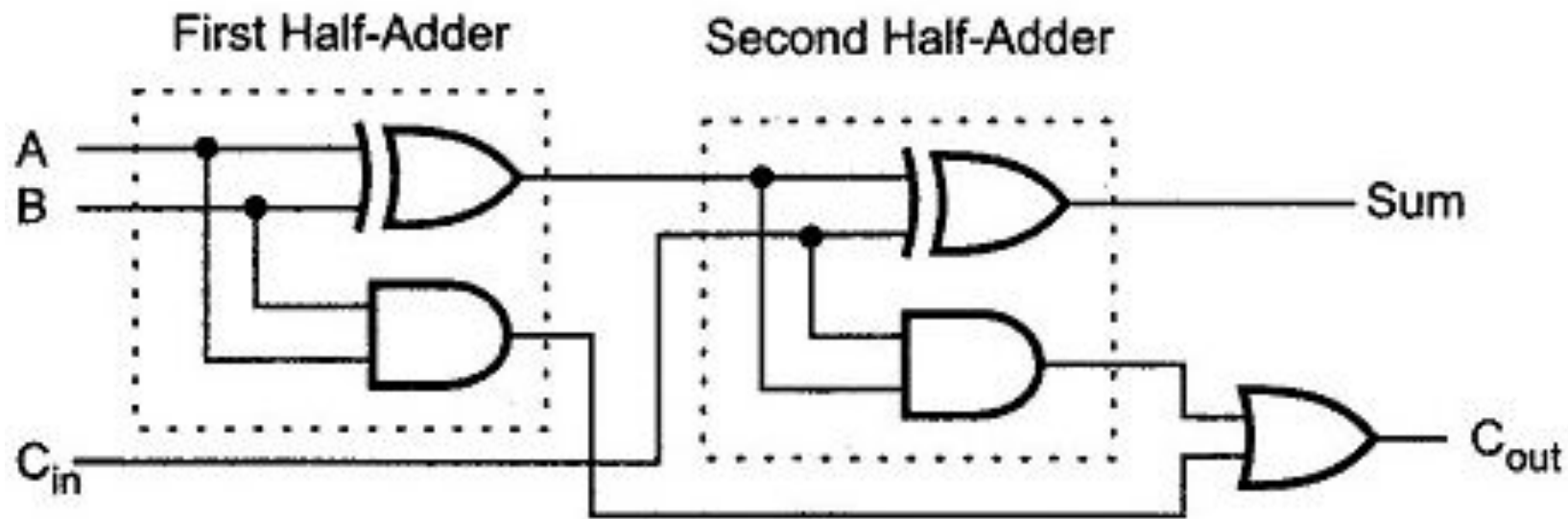


# Full adder

A full-adder can also be implemented with two half-adders and one OR gate



# Full adder



# Full adder truth table and sum expression

| Inputs |   |                 | Outputs |       |
|--------|---|-----------------|---------|-------|
| A      | B | C <sub>in</sub> | Sum     | Carry |
| 0      | 0 | 0               | 0       | 0     |
| 0      | 0 | 1               | 1       | 0     |
| 0      | 1 | 0               | 1       | 0     |
| 0      | 1 | 1               | 0       | 1     |
| 1      | 0 | 0               | 1       | 0     |
| 1      | 0 | 1               | 0       | 1     |
| 1      | 1 | 0               | 0       | 1     |
| 1      | 1 | 1               | 1       | 1     |

Boolean expression for the sum:

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

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

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

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

# Full adder truth table and carry expression

| Inputs |   |                 | Outputs |       |
|--------|---|-----------------|---------|-------|
| A      | B | C <sub>in</sub> | Sum     | Carry |
| 0      | 0 | 0               | 0       | 0     |
| 0      | 0 | 1               | 1       | 0     |
| 0      | 1 | 0               | 1       | 0     |
| 0      | 1 | 1               | 0       | 1     |
| 1      | 0 | 0               | 1       | 0     |
| 1      | 0 | 1               | 0       | 1     |
| 1      | 1 | 0               | 0       | 1     |
| 1      | 1 | 1               | 1       | 1     |

Boolean expression for carry:

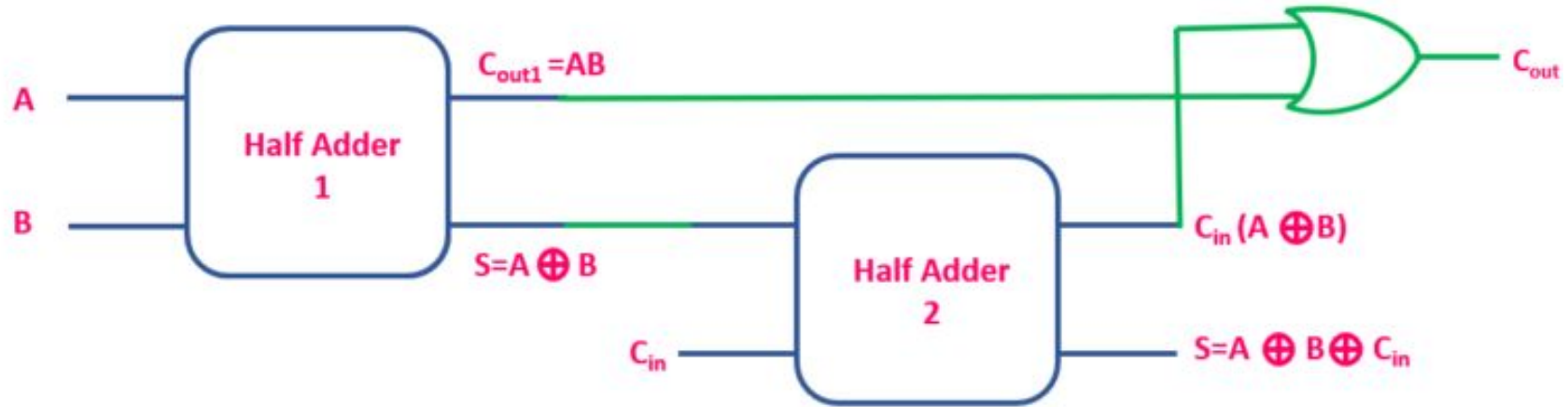
$$C_{out} = \bar{A} B C_{in} + A \bar{B} C_{in} + A B \bar{C}_{in} + A B C_{in}$$

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

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

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

# Full adder circuit with Boolean expression for sum and carry

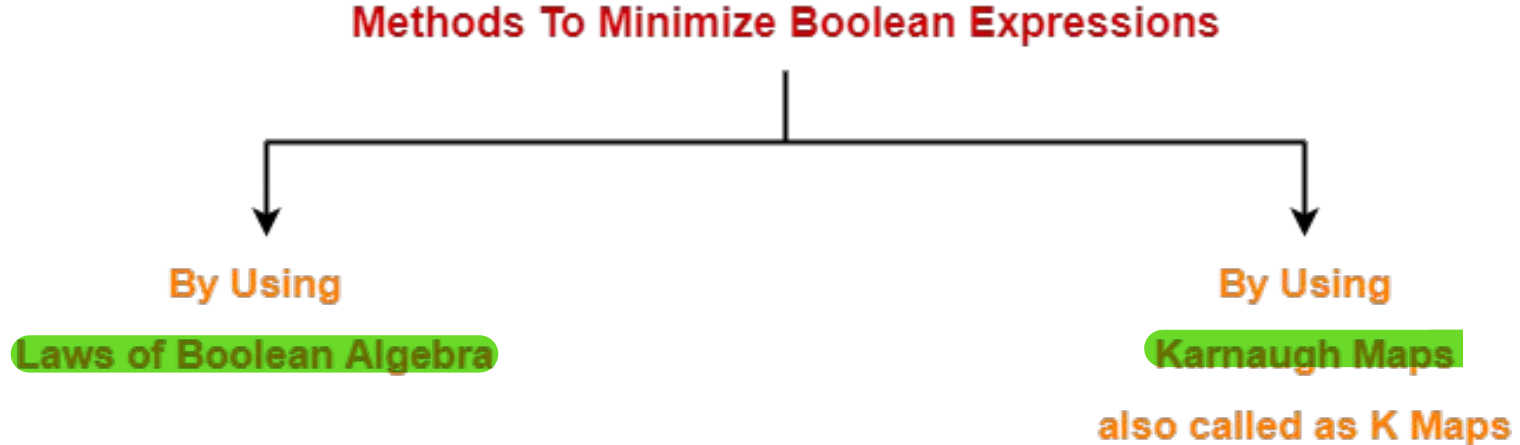




# Karnaugh Map / K-map method

K-map is a technique to simplify Boolean expression.

It is a graphical chart which contains boxes called cells.



# K-map method

K-map employs the use of two-dimensional tables to simplify the expressions, whose size increases at a very high rate with the increase in the number of variables.

| A \ B | 0 | 1 |
|-------|---|---|
| 0     | 0 | 1 |
| 1     | 2 | 3 |

(a) Two Variables

| A \ BC | 00 | 01 | 11 | 10 |
|--------|----|----|----|----|
| 0      | 0  | 1  | 3  | 2  |
| 1      | 4  | 5  | 7  | 6  |

(b) Three Variables

| AB \ CD | 00 | 01 | 11 | 10 |
|---------|----|----|----|----|
| 00      | 0  | 1  | 3  | 2  |
| 01      | 4  | 5  | 7  | 6  |
| 11      | 12 | 13 | 15 | 14 |
| 10      | 8  | 9  | 11 | 10 |

(c) Four Variables

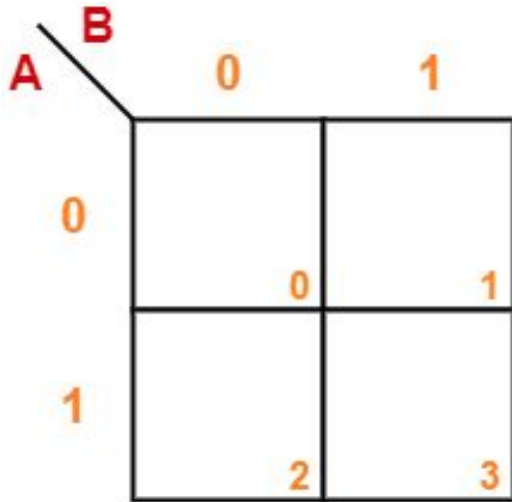
Figure. – Karnaugh Maps for (a) Two Variables (b) Three Variables (c) Four Variables

## K-map method

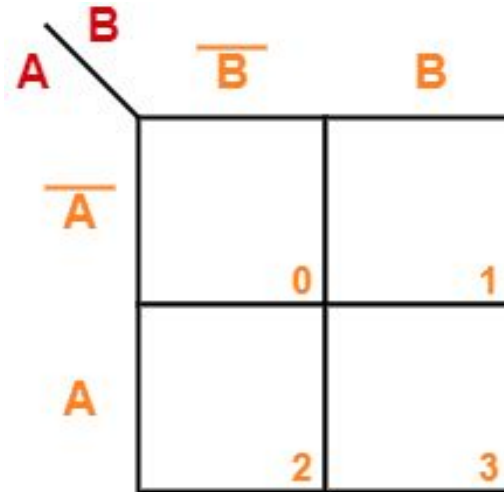
From the figure, it is evident that the number of cells in the K-map is a function of number of inputs. In general, if there are  $n$  inputs, then the corresponding K-map has to be of  $2^n$

cells. For example, if the number of input variables is 2, then we have to consider a K-map with 4 ( $=2^2$ ) cells, while if there are 3 input variables, then we require a 8 ( $=2^3$ ) cell K-map, and similarly for 4 inputs one gets 16 ( $=2^4$ ) cell K-map and so on.

# K-map method : 2-variable



OR



# K-map method : 3-variable

| A | BC |    |    |    |
|---|----|----|----|----|
|   | 00 | 01 | 11 | 10 |
| 0 | 0  | 1  | 3  | 2  |
| 1 | 4  | 5  | 7  | 6  |

OR

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

# K-map method : 4-variable

| AB \ CD |    |    |    |    |
|---------|----|----|----|----|
|         | 00 | 01 | 11 | 10 |
| 00      | 0  | 1  | 3  | 2  |
| 01      | 4  | 5  | 7  | 6  |
| 11      | 12 | 13 | 15 | 14 |
| 10      | 8  | 9  | 11 | 10 |

OR

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

# K-map method

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

Incorrect

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

Correct

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

Incorrect

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

Correct

# K-map method

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |

Incorrect

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |

Correct

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 |

Incorrect

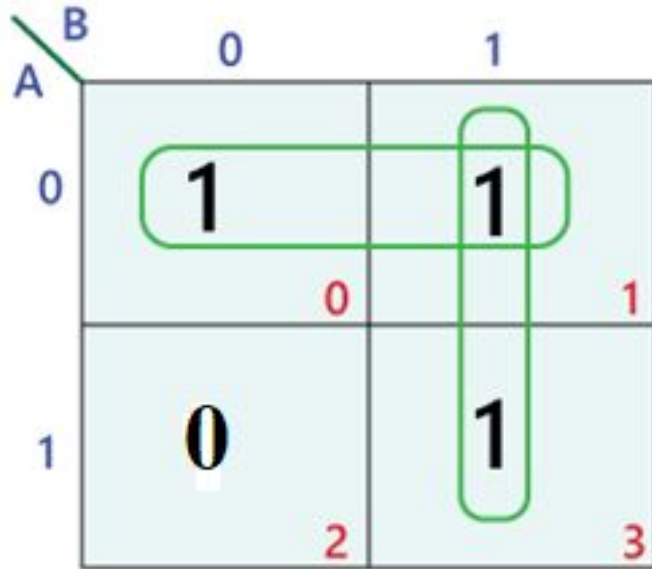
|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 |

Correct



# K-map method: SOP form

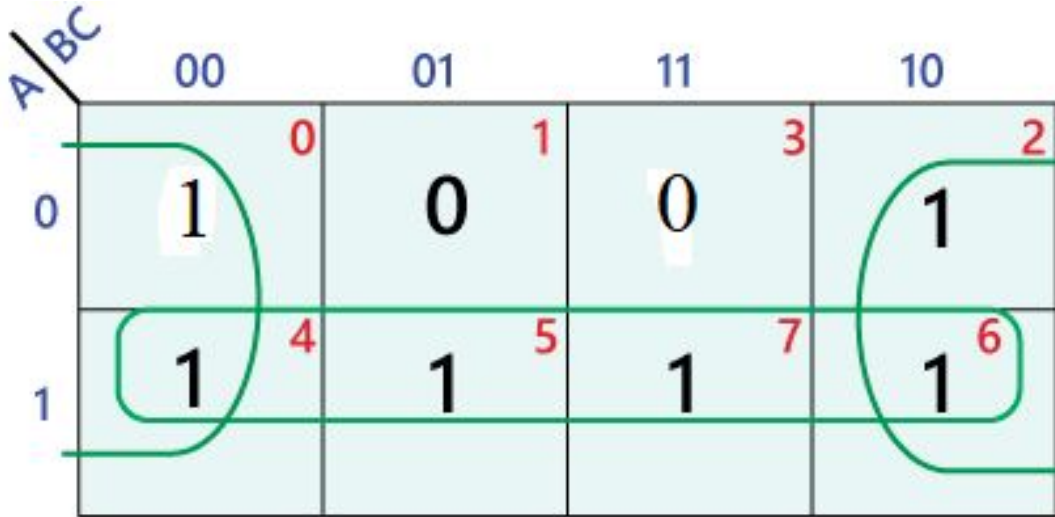
Plot Boolean expression  $Y = A'B' + A'B + AB$  on K-map



Simplified expression:  $Y = A' + B$

# K-map method for SOP

Plot Boolean expression  $Y = A'B'C' + A'BC' + AB'C' + AB'C + ABC' + ABC$  on K-map



Simplified expression:  $Y = A + C'$

## K-map method: Example

Plot the following Boolean expression on K-map

$$\begin{aligned} &\overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}CD \\ &+ A\overline{B}\overline{C}\overline{D} + A\overline{B}\overline{C}D + A\overline{B}C\overline{D} + A\overline{B}CD \end{aligned}$$

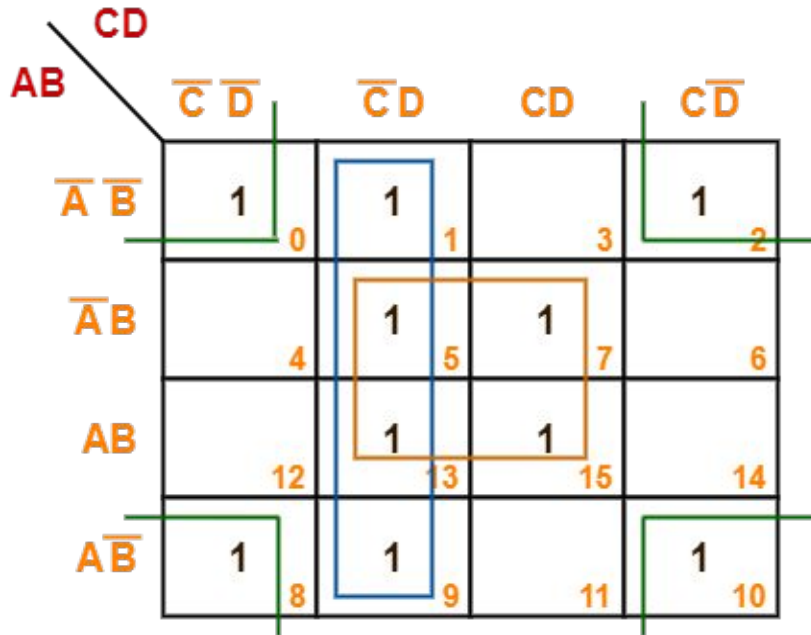
| $\begin{matrix} A \\ B \end{matrix} \backslash \begin{matrix} C \\ D \end{matrix}$ | 00 | 01 | 11 | 10 |
|--|----|----|----|----|
| 00   | 1  | 1  | 1  | 1  |
| 01   |    |    |    |    |
| 11   |    |    |    |    |
| 10   | 1  | 1  | 1  | 1  |

Simplified expression:  $Y=B'$

# K-map method: Example

Minimize the following boolean function-

$$F(A, B, C, D) = \sum m(0, 1, 2, 5, 7, 8, 9, 10, 13, 15)$$



Thus, minimized boolean expression is-

$$F(A, B, C, D) = BD + C'D + B'D'$$

# K-map method - Examples

Minimize the following boolean function-

$$F(A, B, C, D) = \sum m(0, 1, 3, 5, 7, 8, 9, 11, 13, 15)$$

| CD \ AB                    | $\overline{C}\overline{D}$ | $\overline{C}D$ | $CD$ | $C\overline{D}$ |
|----------------------------|----------------------------|-----------------|------|-----------------|
| $\overline{A}\overline{B}$ | 1                          | 1               | 1    |                 |
| $\overline{A}B$            |                            | 1               | 1    |                 |
| $AB$                       |                            | 1               | 1    |                 |
| $A\overline{B}$            | 1                          | 1               | 1    |                 |

Thus, minimized boolean expression is

$$F(A, B, C, D) = B'C' + D$$

## K-map method: POS form

Minimize the following boolean function-

$$F(A,B,C)=\pi(0,3,6,7)$$

|        | BC |    |    |    |
|--------|----|----|----|----|
|        | 00 | 01 | 11 | 10 |
| A<br>0 | 0  | 1  | 0  | 1  |
| A<br>1 | 1  | 1  | 0  | 0  |

Final expression is  $(A' + B')(B' + C')(A + B + C)$

## K-map method: POS form

Minimize the following boolean function-

$$F(A,B,C) = \pi (3, 5, 7, 8, 10, 11, 12, 13)$$

| AB \ CD | CD      |         |         |         |
|---------|---------|---------|---------|---------|
|         | 00      | 01      | 11      | 10      |
| 00      | 1<br>0  | 1<br>1  | 0<br>3  | 1<br>2  |
| 01      | 1<br>4  | 0<br>5  | 0<br>7  | 1<br>6  |
| 11      | 0<br>12 | 0<br>13 | 1<br>15 | 1<br>14 |
| 10      | 0<br>8  | 1<br>9  | 0<br>11 | 0<br>10 |

Final expression is:

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

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

## Practice example

1. Minimize the following 4 variable POS function using K map:  $F(A,B,C,D,E) = \pi M(0,1,3,5,6,8,9,11,15)$
2. Minimize the following 4 variable SOP function using K map:  $F(A,B,C,D,E) = \sum m(0,5,6,8,9,10,11,16)$
3. Minimize the boolean function  $Y = (A'+B'+C+D)(A+B'+C+D)(A+B+C+D')(A+B+C'+D')(A'+B+C+D')(A+B+C'+D)$ .
4. Minimize the boolean expression  $Y = ABC'D + ABC'D' + ABCD + A'BCD + ABCD' + A'BCD'$ .



# Don't care condition

- The "Don't care" condition says that we can use the blank cells of a K-map to make a group of the variables.
- To make a group of cells, we can use the "don't care" cells as either 0 or 1, and if required, we can also ignore that cell.
- We mainly use the "don't care" cell to make a large group of cells.

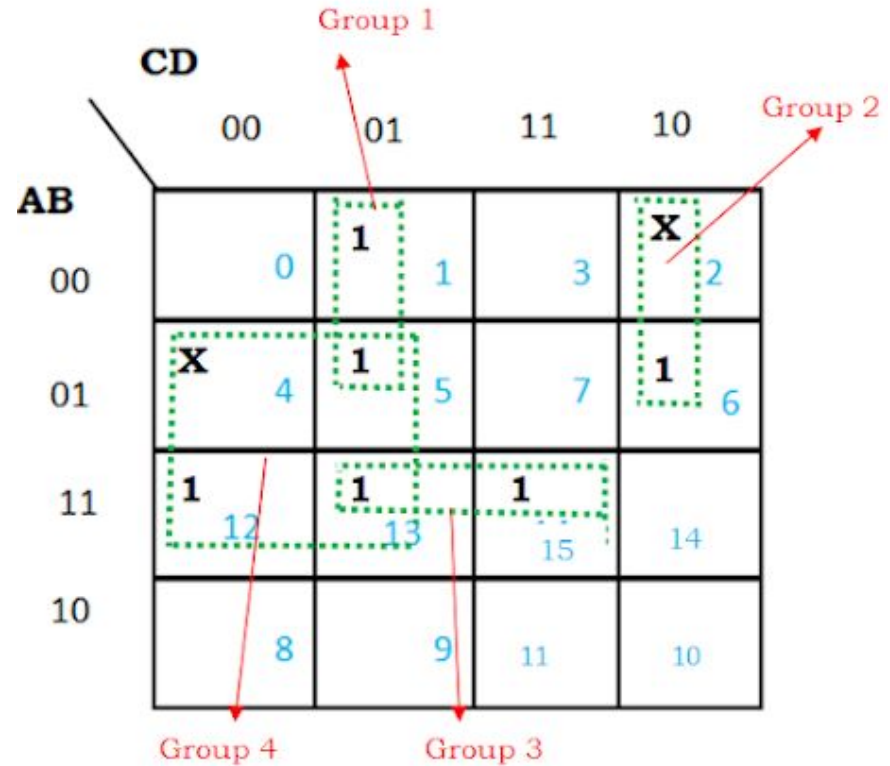
| AB<br>CD | 00 | 01 | 11 | 10 |
|----------|----|----|----|----|
| 00       |    |    | X  |    |
| 01       | X  | 1  | X  | 1  |
| 11       | 1  | 1  | X  | X  |
| 10       |    | 1  | X  | X  |

# Don't care condition-Example

Minimize the SOP expression using K map(with don't care conditions): $F(A,B,C,D)=\sum m(1,5,6,12,13,15)+d(2,4)$

=> So the minimized expression from K map is

$$F(A,B,C,D)=A'C'D+A'CD'+ABD+BC'$$



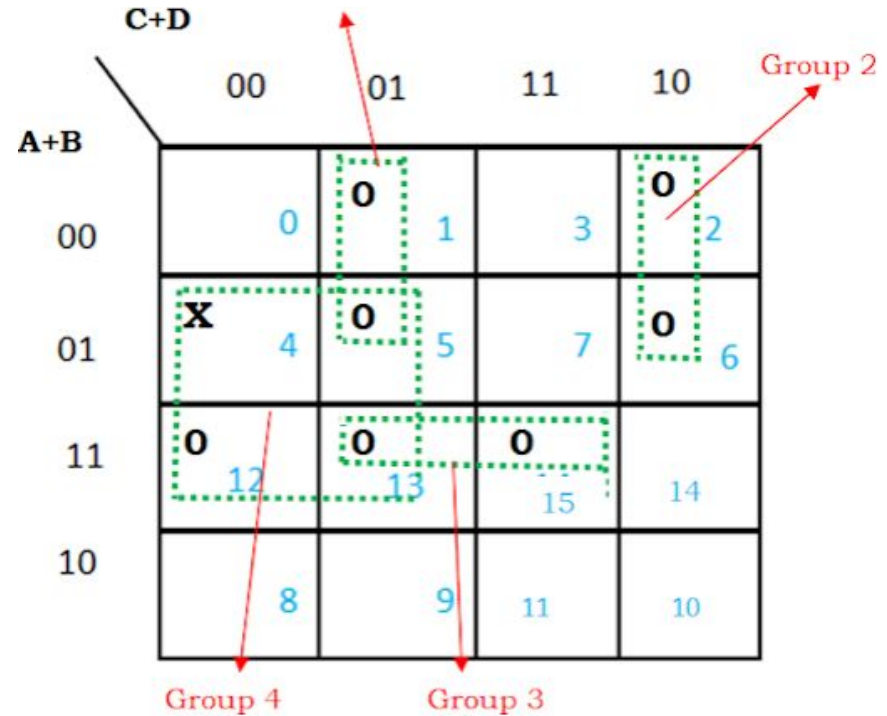
# Don't care condition-Example

Minimize the POS expression using K map(with don't care conditions):

$$f(A,B,C,D)=\pi M(1,5,6,12,13,15)+d(2,4)$$

=> So the minimized expression from K map is

$$f(A,B,C,D)=(A+C+D').(A+C'+D).(A'+B'+D').(B'+C)$$



## Don't care condition

Practice Example1: Minimize  $F = m(1, 5, 6, 12, 13, 14) + d(4)$  in SOP minimal form

Practice Example2: Minimize  $F(A, B, C, D) = M(6, 7, 8, 9) + d(12, 13, 14, 15)$  in POS form.

# Subtractor

Subtractor circuits take two binary numbers as input and subtract one binary number input from the other binary number input. Similar to adders, it gives out two outputs, difference and borrow (carry-in the case of Adder).

There are two types of subtractors.

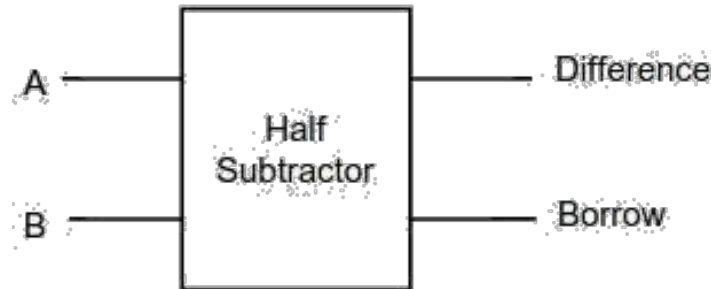
1. Half Subtractor
2. Full Subtractor

# Half Subtractor

The half subtractor is also a building block for subtracting two binary numbers.

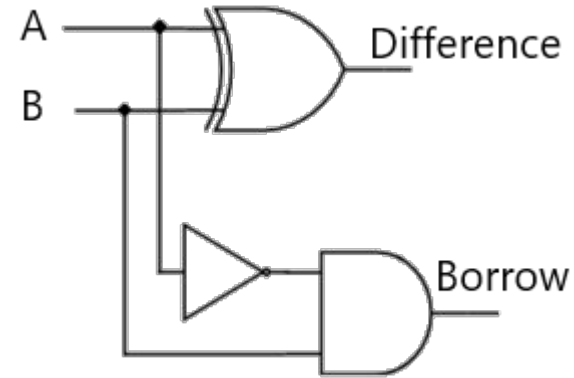
It has two inputs and two outputs.

This circuit is used to subtract two single bit binary numbers A and B. The 'diff' and 'borrow' are two output states of the half subtractor.



# Half Subtractor

| Inputs |   | Outputs    |        |
|--------|---|------------|--------|
| A      | B | Difference | Borrow |
| 0      | 0 | 0          | 0      |
| 0      | 1 | 1          | 1      |
| 1      | 0 | 1          | 0      |
| 1      | 1 | 0          | 0      |

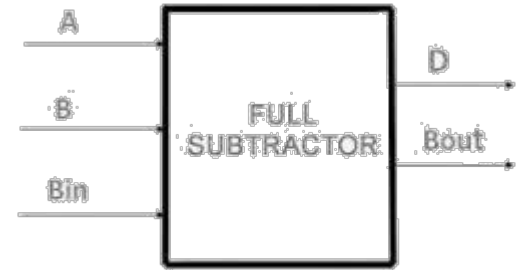


$$\text{Difference} = A \oplus B$$

$$\text{Borrow} = A' B$$

# Full Subtractor

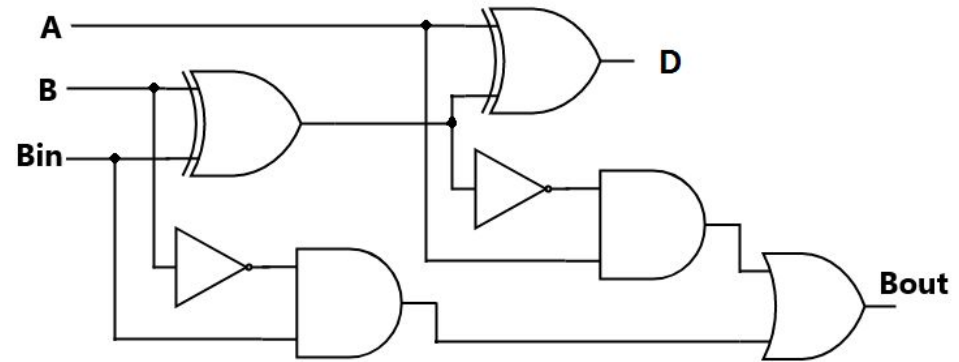
- A full subtractor is a combinational circuit that performs subtraction involving three bits, namely A (minuend), B (subtrahend), and Bin (borrow-in) .
- It accepts three inputs: A (minuend), B (subtrahend) and a Bin (borrow bit) and it produces two outputs: D (difference) and Bout (borrow out).





# Full Subtractor

| A | B | B <sub>in</sub> | D | B <sub>out</sub> |
|---|---|-----------------|---|------------------|
| 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                |



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

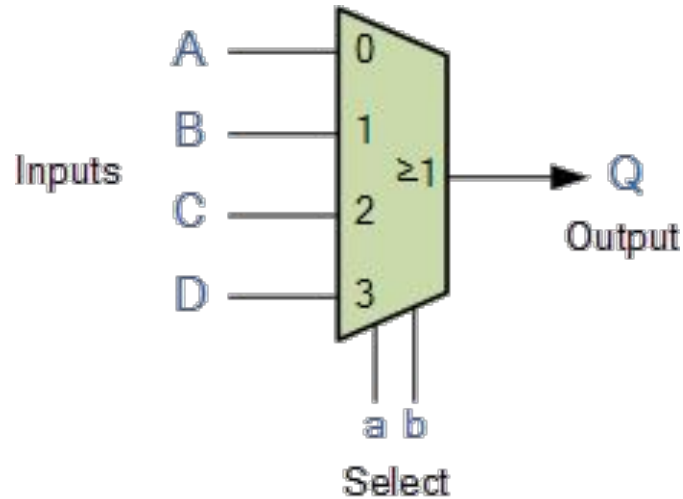
$$B_{out} = A' B_{in} + A' B + B B_{in}$$

# Multiplexer

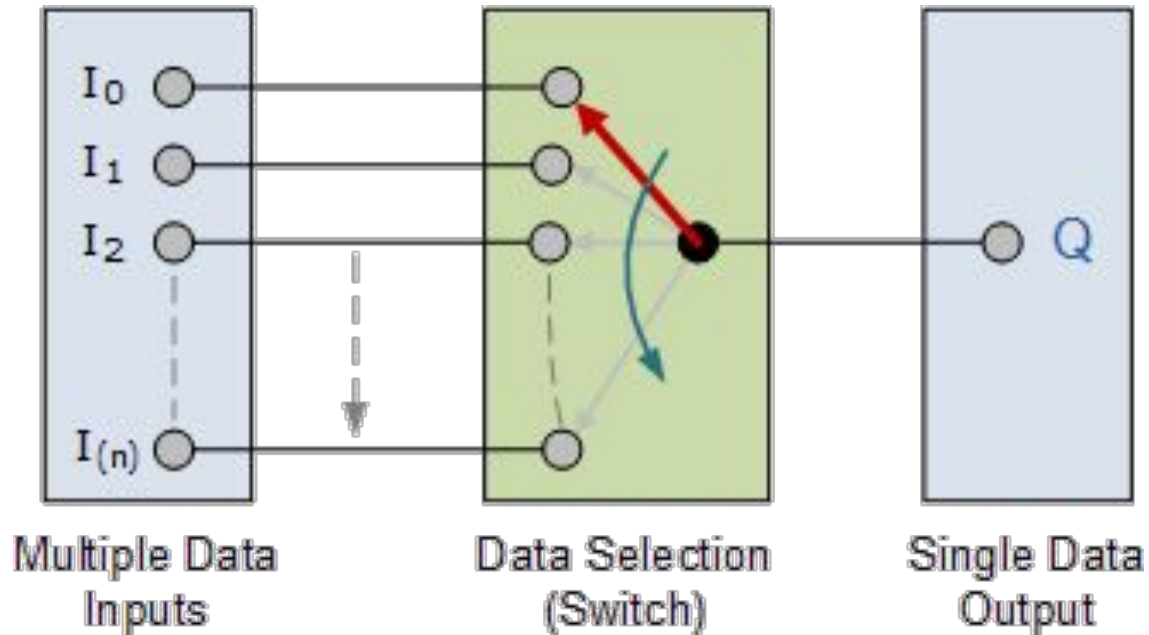
- A multiplexer is a combinational circuit that has  $2^n$  input lines and a single output line.
- Simply, the multiplexer is a multi-input and single-output combinational circuit.
- The binary information is received from the input lines and directed to the output line. On the basis of the values of the selection lines, one of these data inputs will be connected to the output. A multiplexer is also treated as **Mux**.

# Multiplexer

Multiplexer symbol-

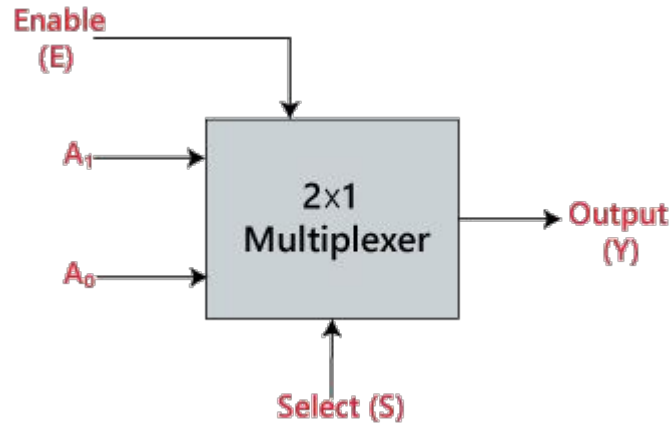


# Basic Multiplexing switch



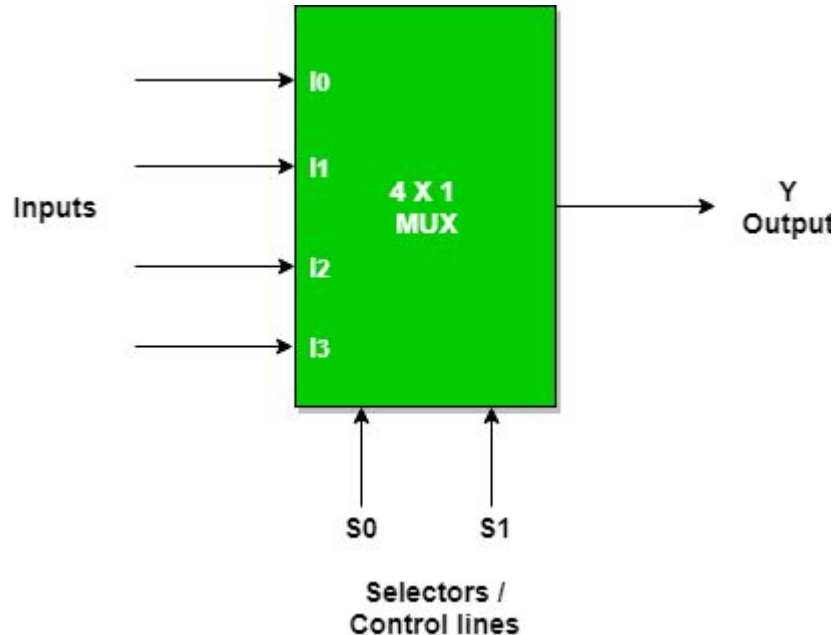
# 2X1 Multiplexer

In  $2 \times 1$  multiplexer, there are only two inputs, i.e.,  $A_0$  and  $A_1$ , 1 selection line, i.e.,  $S_0$  and single outputs, i.e.,  $Y$ .



# 4X1 Multiplexer

In  $4 \times 1$  multiplexer, there are only 4 inputs, 2 selection line and single outputs.



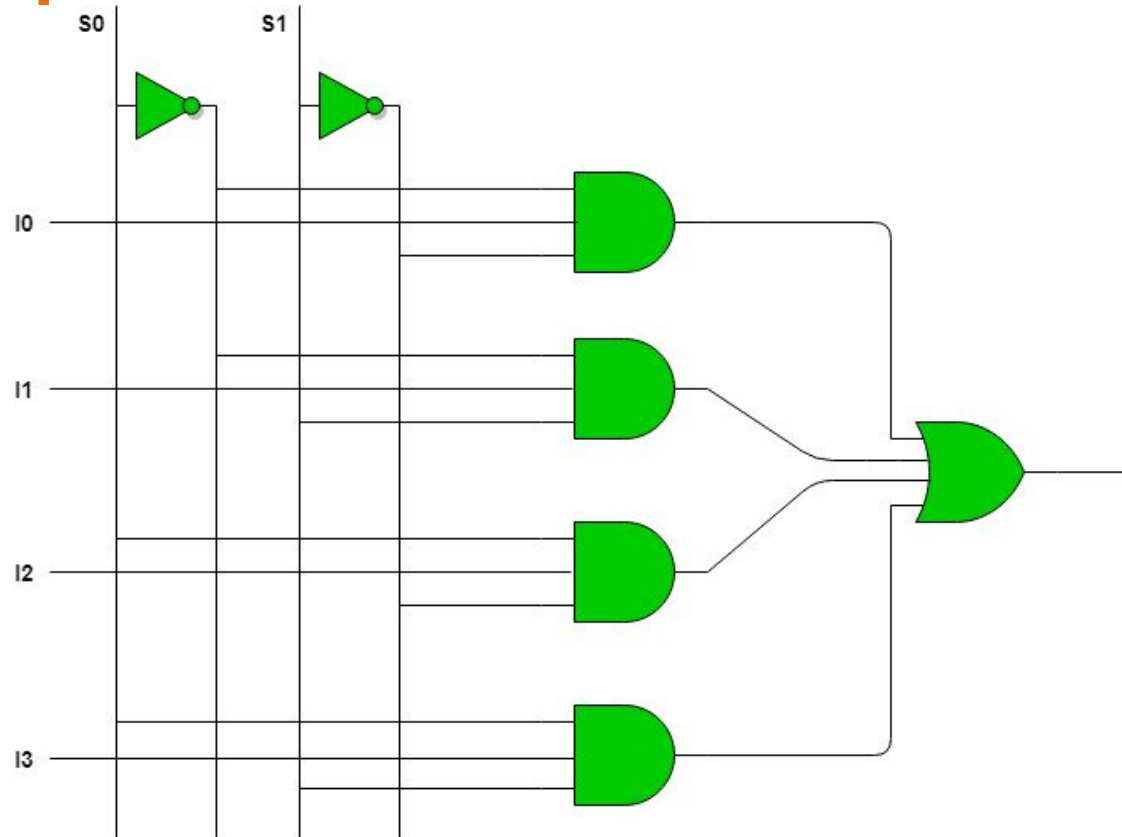
Truth Table

| $S_0$ | $S_1$ | $Y$   |
|-------|-------|-------|
| 0     | 0     | $I_0$ |
| 0     | 1     | $I_1$ |
| 1     | 0     | $I_2$ |
| 1     | 1     | $I_3$ |

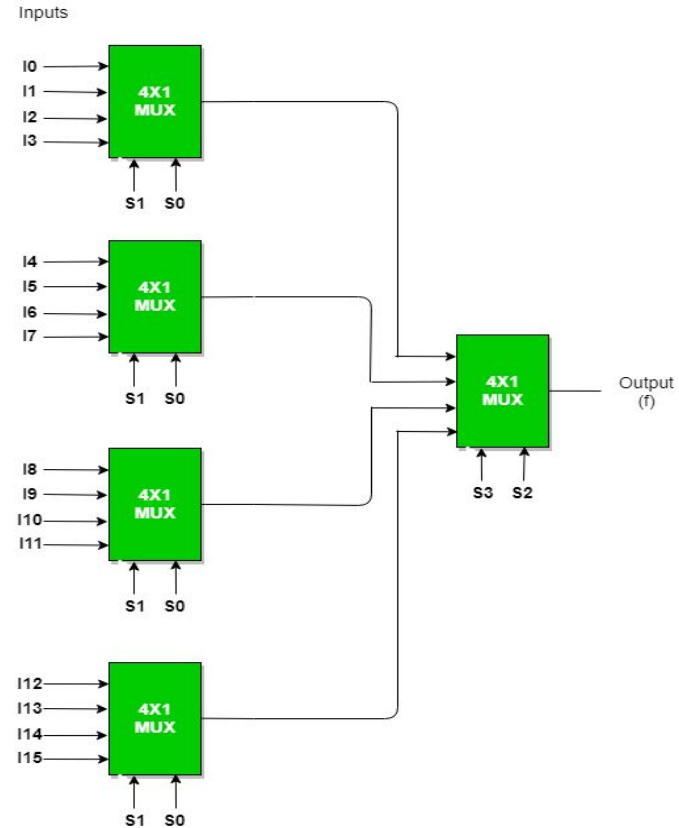
So, final equation,

$$Y = S_0'.S_1'.I_0 + S_0'.S_1.I_1 + S_0.S_1'.I_2 + S_0.S_1.I_3$$

# 4X1 Multiplexer



# 16:1 using 4:1 Multiplexer

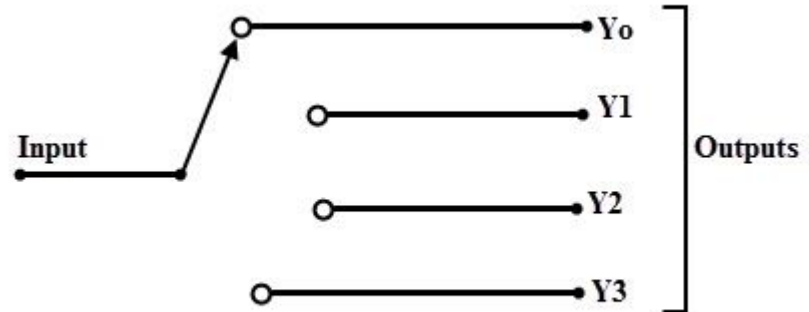
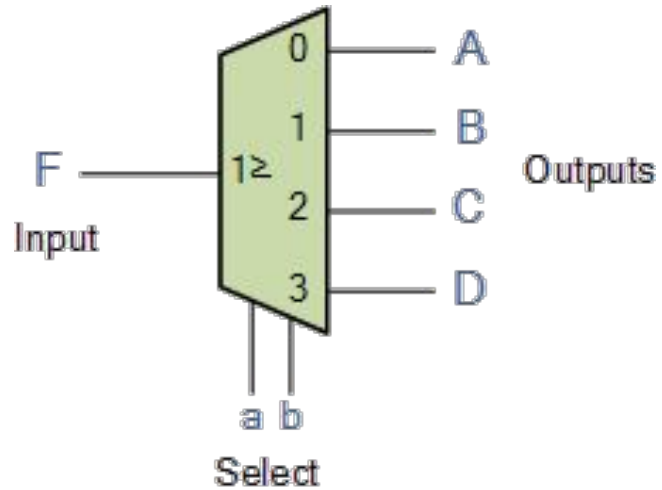




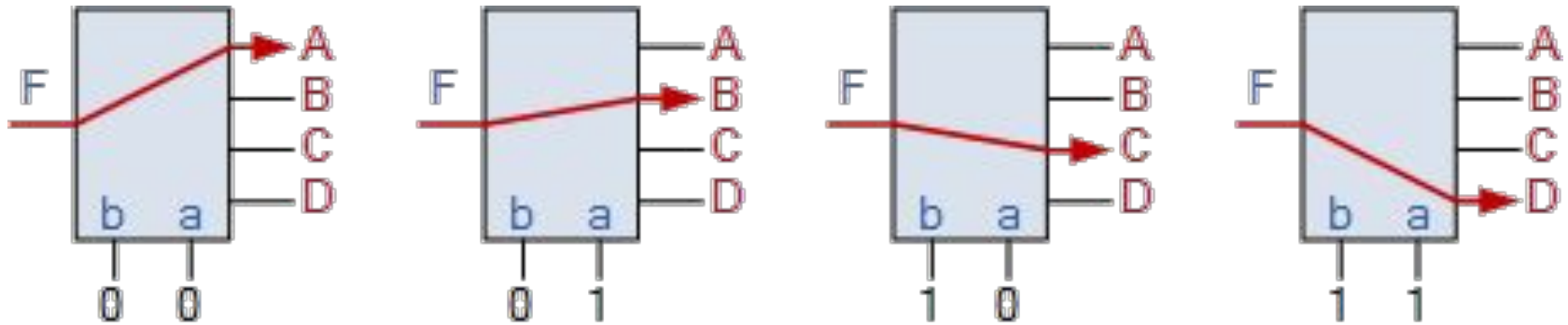
# Demultiplexer

- De-Multiplexer is a combinational circuit that performs the reverse operation of Multiplexer.
- It has single input, 'n' selection lines and maximum of  $2^n$  outputs.
- The input will be connected to one of these outputs based on the values of selection lines.
- Demultiplexer is also called as De-Mux.

# 1:4 Demultiplexer

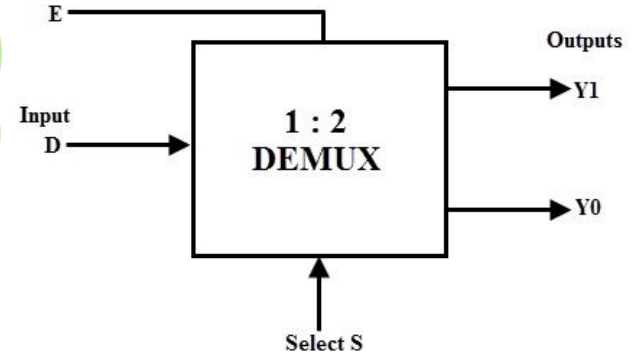


# Demultiplexer output line selection



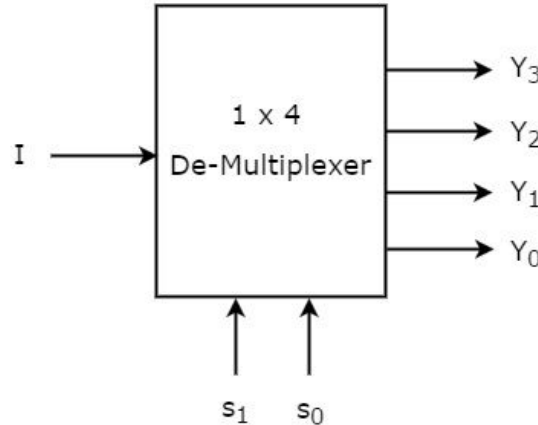
## 1:2 Demultiplexer

- A 1:2 demux consists of one input line, two output lines and one select line.
- The signal on the select line helps to switch the input to one of the two outputs.
- The figure below shows the block diagram of a 1-to-2 demultiplexer with additional enable input.



## 1:4 Demultiplexer

- 1x4 De-Multiplexer has one input I, two selection lines,  $s_1$  &  $s_0$  and four outputs  $Y_3$ ,  $Y_2$ ,  $Y_1$  &  $Y_0$ .
- The block diagram of 1x4 De-Multiplexer is shown in the following figure.



## 1:4 Demultiplexer

The single input 'I' will be connected to one of the four outputs,  $Y_3$  to  $Y_0$  based on the values of selection lines  $s_1$  &  $s_0$ . The **Truth table** of 1x4 De-Multiplexer is shown below.

| 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     |

# 1:4 Demultiplexer

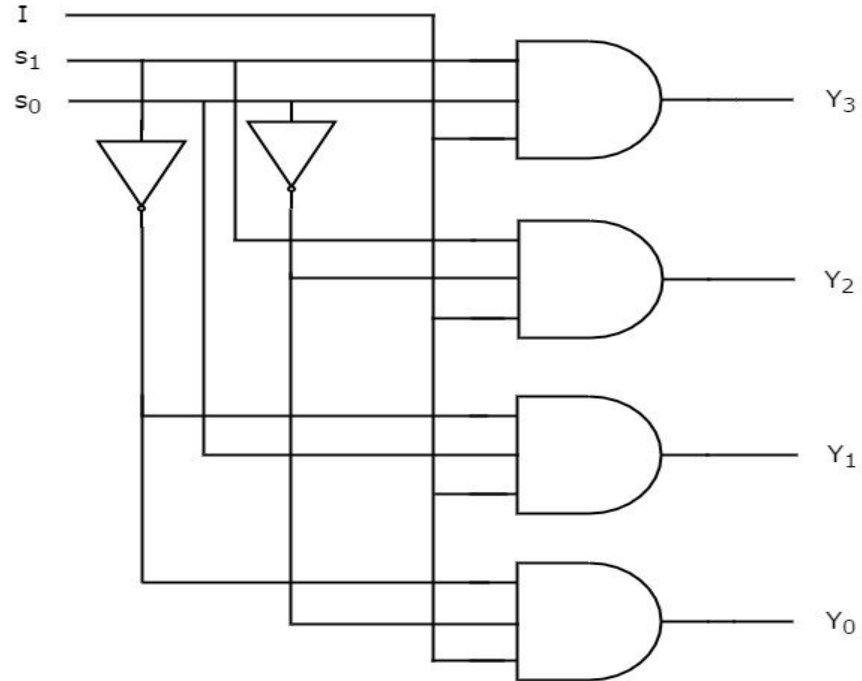
From the above truth table, the Boolean Expressions for the outputs as follows:

$$Y_0 = S_1'S_0'I$$

$$Y_1 = S_1'S_0I$$

$$Y_2 = S_1S_0'I$$

$$Y_3 = S_1S_0I$$



A large red square with a white border, centered on a white background. The text "Flip flops" is written in white inside the red square.

**Flip flops**

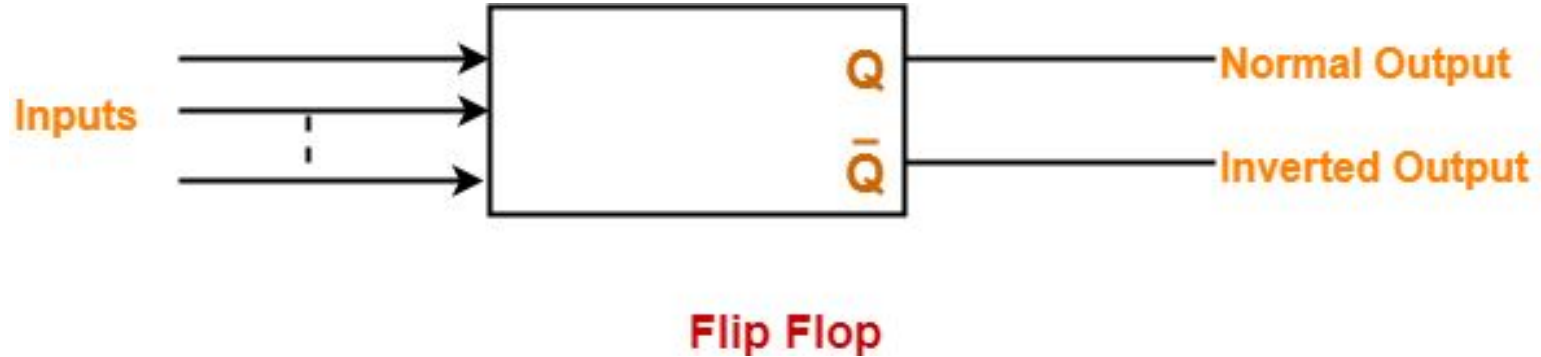


# Introduction

- A Flip Flop is an electronic circuit that is capable of storing one bit of information.
- It is a memory element that is capable of storing one bit of information.
- It has two stable states either 0 or 1.

# Introduction

A flip flop has two outputs as shown-



# Flip flop types

Flip flops are of different types depending on how their inputs and clock pulses cause transition between two states.

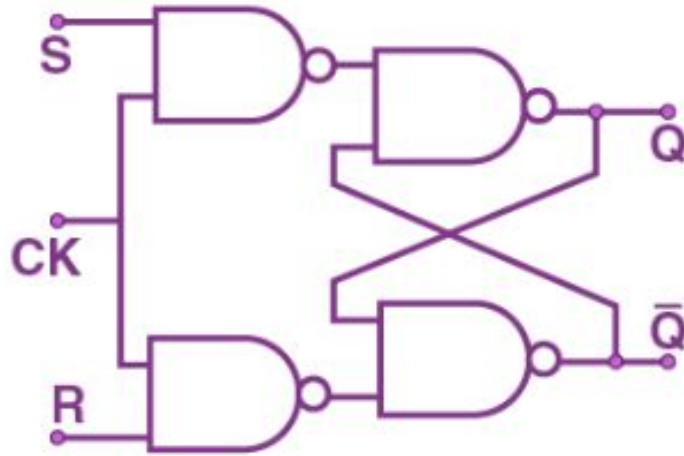
There are 4 basic types of flip flops-

1. SR Flip Flop
2. JK Flip Flop
3. D Flip Flop
4. T Flip Flop

# SR Flip flop

- SR flip flop is the simplest type of flip flops.
- It stands for **Set Reset flip flop**.
- It is a clocked flip flop.
- Following are the two methods for constructing a SR flip flop-
  1. By using NOR latch
  2. By using NAND latch

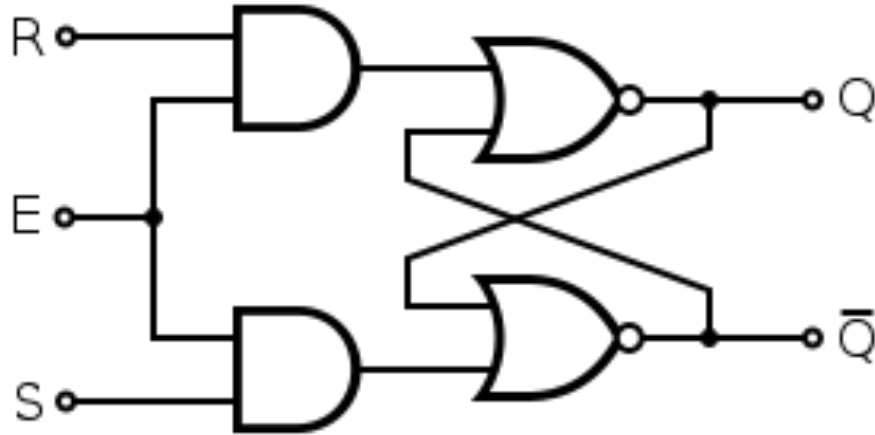
# 1. Construction of SR Flip Flop By Using NAND Latch-



Truth Table

| S | R | $Q_N$ | $Q_{N+1}$ |
|---|---|-------|-----------|
| 0 | 0 | 0     | 0         |
| 0 | 0 | 1     | 1         |
| 0 | 1 | 0     | 0         |
| 0 | 1 | 1     | 0         |
| 1 | 0 | 0     | 1         |
| 1 | 0 | 1     | 1         |
| 1 | 1 | 0     | -         |
| 1 | 1 | 1     | -         |

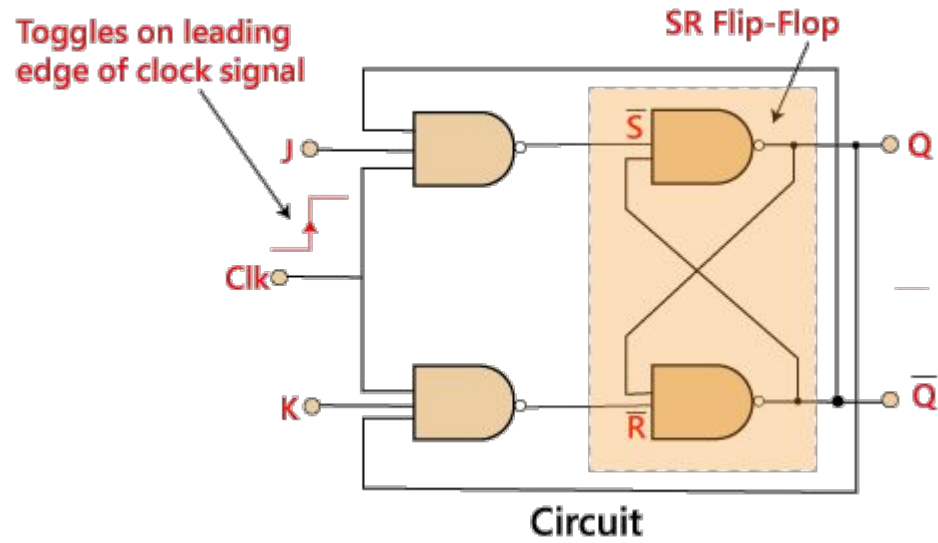
## 2. Construction of SR Flip Flop By Using NOR Latch-



# JK Flip flop

- JK flip flop is a refined & improved version of SR flip flop that has been introduced to solve the problem of indeterminate state that occurs in SR flip flop when both the inputs are 1.
- There are following two methods for constructing a JK flip flop-

# JK Flip flop



| J | K | $Q_n$ | $Q_{n+1}$ | State        |
|---|---|-------|-----------|--------------|
| 0 | 0 | 0     | 0         | $Q_n$ (Hold) |
| 0 | 0 | 1     | 1         |              |
| 0 | 1 | 0     | 0         | Reset        |
| 0 | 1 | 1     | 0         |              |
| 1 | 0 | 0     | 1         | Set          |
| 1 | 0 | 1     | 1         |              |
| 1 | 1 | 0     | 1         | Toggle       |
| 1 | 1 | 1     | 0         |              |



# JK Flip flop

| J | K | $Q_n$ | $Q_{n+1}$ | State        |
|---|---|-------|-----------|--------------|
| 0 | 0 | 0     | 0         | $Q_n$ (Hold) |
| 0 | 0 | 1     | 1         |              |
| 0 | 1 | 0     | 0         | Reset        |
| 0 | 1 | 1     | 0         |              |
| 1 | 0 | 0     | 1         | Set          |
| 1 | 0 | 1     | 1         |              |
| 1 | 1 | 0     | 1         | Toggle       |
| 1 | 1 | 1     | 0         |              |

|     |   | $KQ_n$ |    |    |    |
|-----|---|--------|----|----|----|
|     |   | 00     | 01 | 11 | 10 |
| $J$ | 0 |        | 1  |    |    |
|     | 1 | 1      | 1  |    | 1  |

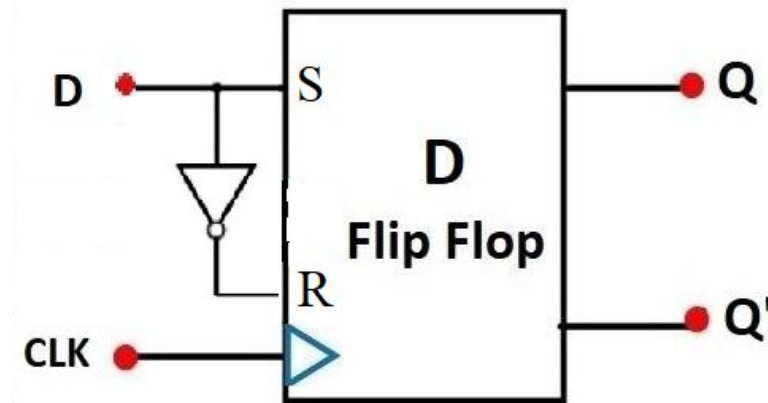
$\therefore$  Characterstic equation is  $Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$

# D Flip flop

- D stands for data thus it is used to store data.
- The basic building block of D flip flop is SR flip flop.
- In SR flip flop if both i/p are same, o/p is either no change or it is invalid (*i/o -> 00, No change and o/p -> 1 1, Invalid*).
- These conditions can be avoided by making them complement of each other. This modified SR flip flop is called as D flip flop.

# D Flip flop

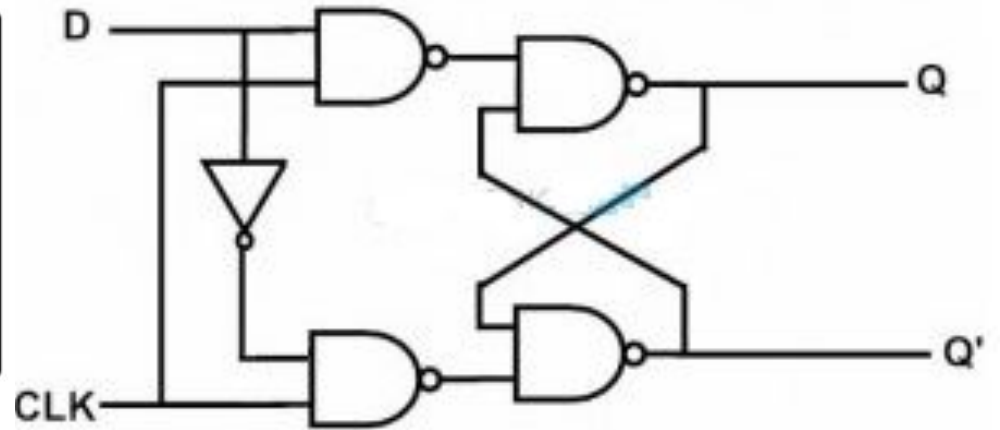
- The input D goes directly to i/p S and its complement is applied to R i/p. Due to this, only two i/p conditions exist:  $S=0 \ \& \ R=1$  or  $S=1 \ \& \ R=0$ .



# D Flip flop

| D | S | R | Q              | State     |
|---|---|---|----------------|-----------|
|   | 0 | 0 | Previous State | No Change |
| 0 | 0 | 1 | 0              | Reset     |
| 1 | 1 | 0 | 1              | Set       |
|   | 1 | 1 | ?              | Forbidden |

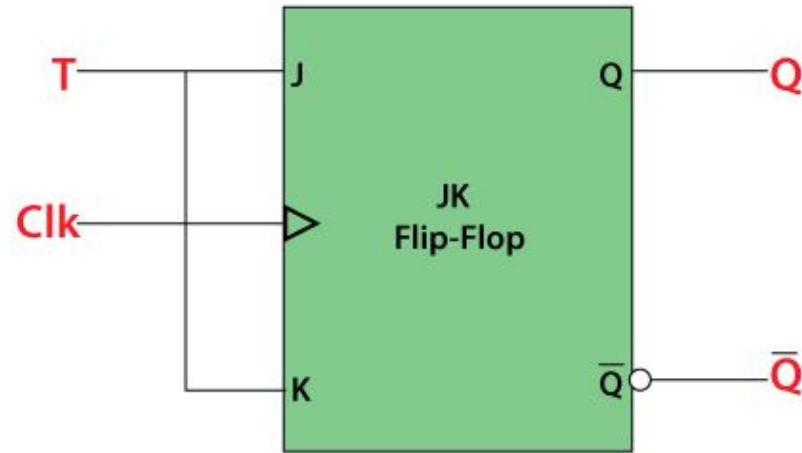
SR & D Flip Flop TruthTable



# T (Toggle) Flip flop

- We can design the T flip flop by making simple modifications to the JK flip flop.
- The T flip flop is a single input device and hence by connecting J and K inputs together and giving them with single input called T, we can convert a JK flip flop into T flip flop.
- So, a T flip flop is sometimes called as single input JK flip flop.

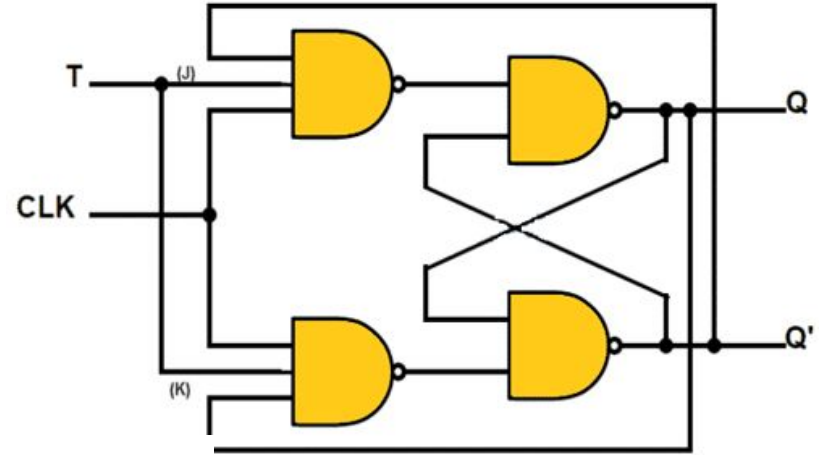
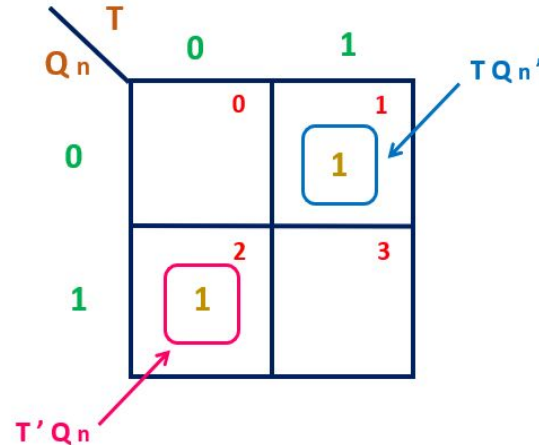
# T (Toggle) Flip flop



# T (Toggle) Flip flop

| $Q_n$ | $T$ | $Q_{n+1}$ |
|-------|-----|-----------|
| 0     | 0   | 0         |
| 0     | 1   | 1         |
| 1     | 0   | 1         |
| 1     | 1   | 0         |

| $T$ | $Q_{n+1}$ |
|-----|-----------|
| 0   | $Q_n$     |
| 1   | $Q_n'$    |



$$Q_{n+1} = TQ_n' + T'Q_n$$

$$= T \oplus Q_n$$