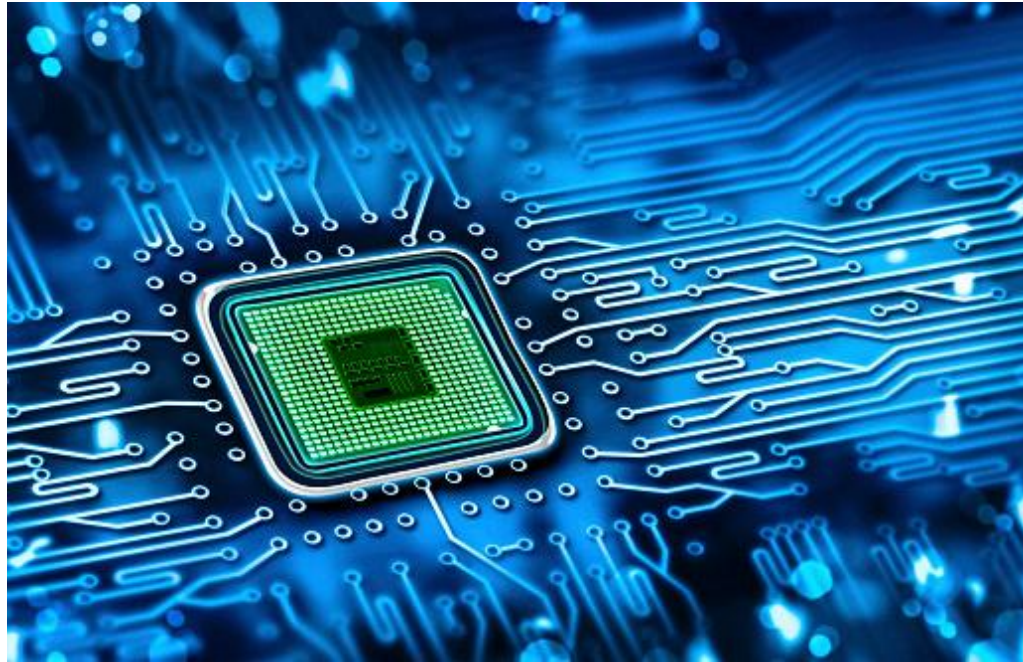


Digital Logic & Application (DLA)

UNIT I



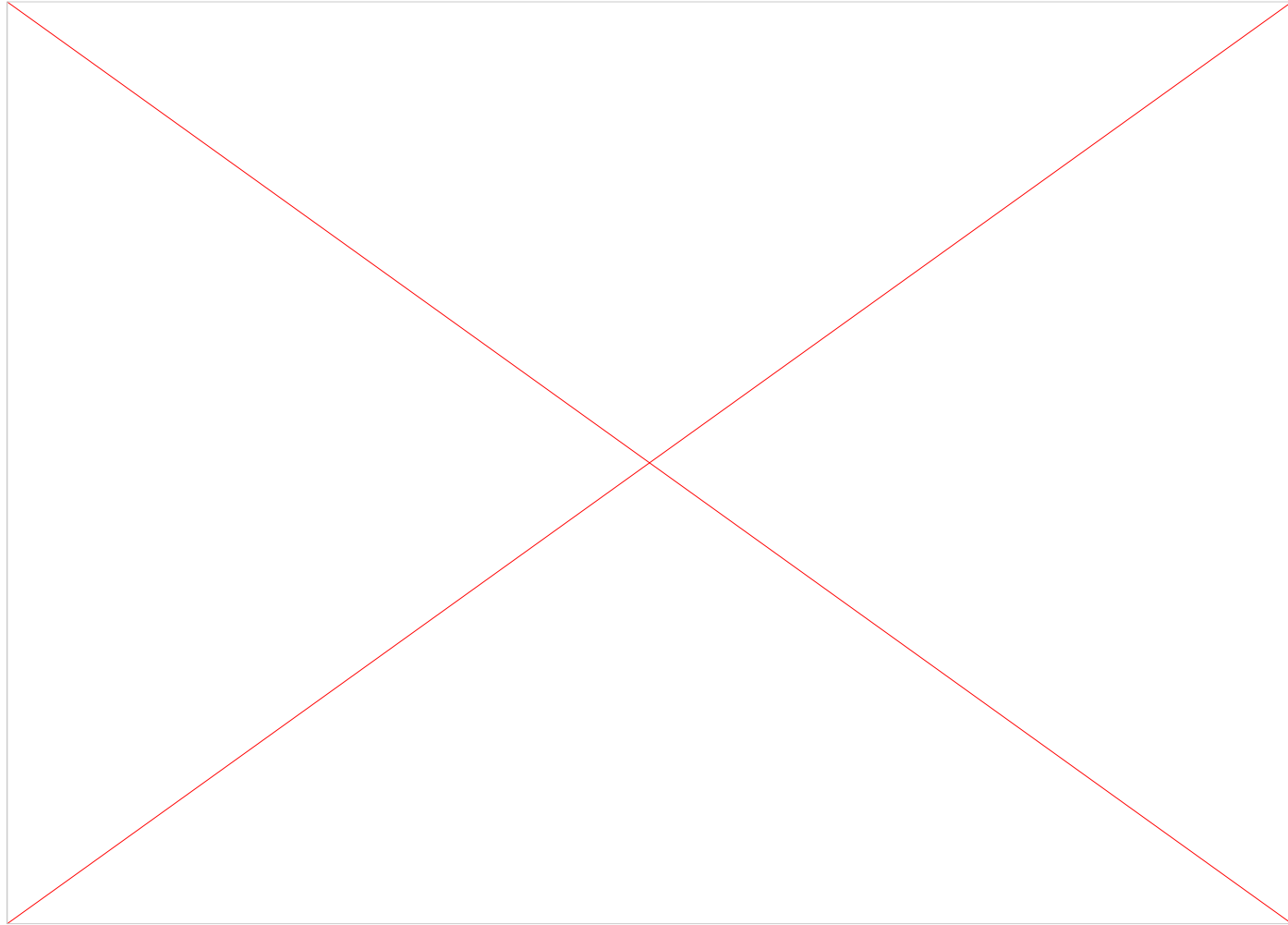
Digital Logic & Application (DLA)

- Digital logic is the underlying logic system that drives electronic circuit board design.
- Digital logic is the representation of signals and sequences of a digital circuit through numbers.
- Digital logic is the manipulation of binary values through printed circuit board technology that uses circuits and logic gates to construct the implementation of computer operations.

Analog v/s Digital

S. No.	Analog signal	Digital Signal
1	Analog signals are continuous signals	Digital signals are discrete signals.
2	Analog signal uses continuous values for representing the information.	A digital signal uses discrete values for representing the information.
3	Analog signals can be affected by the noise during the transmission.	Digital signals cannot be affected by the noise during transmission.
4	Accuracy of Analog signal is affected by the noise.	Digital signals are noise-immune hence their accuracy is less affected
5	Devices which are using analog signals are less flexible	Device using digital signals are very flexible
6	Analog signals consume less bandwidth	Digital signals consume more bandwidth.
7	Analog signals are stored in the form of continuous wave form.	Digital signals are stored in the form of binary bits "0", "1".
8	Analog signals have low cost.	Digital signals have high cost.
9	Analog signals are portable.	Digital signals are not Portable.
10	Analog signals give observation error	Digital Signals doesn't give observation error.

Analog v/s Digital



Analog v/s Digital – Systems

- The other possible way, referred to as digital, represents the numerical value of the quantity in steps of discrete values. The numerical values are mostly represented using binary numbers. For example, the temperature of the oven may be represented in steps of 1°C as 64°C , 65°C , 66°C and so on.
- To summarize, while an analogue representation gives a continuous output, a digital representation produces a discrete output.
- Analogue systems contain devices that process or work on various physical quantities represented in analogue form.
- Digital systems contain devices that process the physical quantities represented in digital form.
- Digital techniques and systems have the advantages of being relatively much easier to design and having higher accuracy, programmability, noise immunity, easier storage of data and ease of fabrication in integrated circuit form, leading to availability of more complex functions in a smaller size. The real world, however, is analogue.

1. Digital Systems and Binary numbers

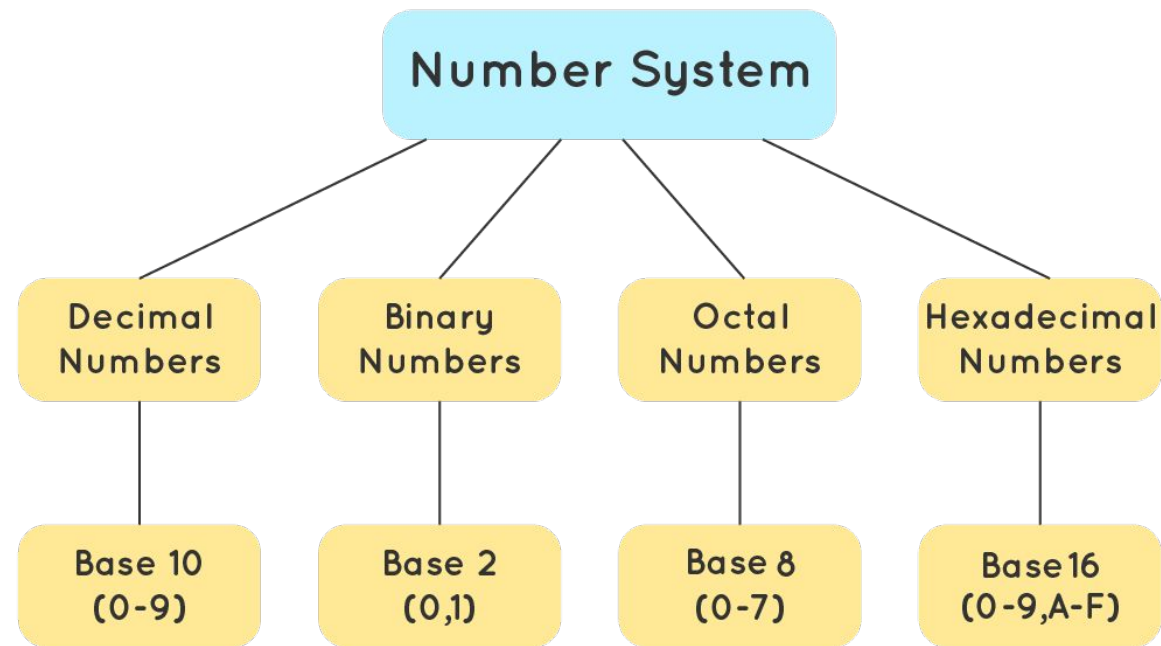
- [Introduction to Number systems.](#)
- [Positional Number systems, Conversions \(converting between bases\).](#)
- Non positional number systems.
- Unsigned and Signed binary numbers.
- Binary Codes.
- Number representation and storage in computer system

Introduction to Number systems

- A number system is defined as a system of writing to express numbers.
- It is the mathematical notation for representing numbers of a given set by using digits or other symbols in a consistent manner.
- It provides a unique representation of every number and represents the arithmetic and algebraic structure of the figures.
- 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.
- The numbers are represented in a unique manner and allow us to operate arithmetic operations like addition, subtraction, and division.
- **What is Base/Radix of a number?** □ Base is defined as a set of digits used to represent numbers, This is the total count of numbers of that particular number system.

2. Positional Number systems, Conversions (converting between bases).

Types of Number System



Types of Number Systems

- Decimal Number System.
- Binary Number System.
- Octal Number System.
- Hexadecimal Number System.

Decimal Number System

- The decimal number system is a radix-10 number system and therefore has 10 different digits or symbols.
- These are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. All higher numbers after '9' are represented in terms of these 10 digits only. The process of writing higher-order numbers after '9' consists in writing the second digit (i.e. '1') first, followed by the other digits, one by one, to obtain the next 10 numbers from '10' to '19'. The next 10 numbers from '20' to '29' are obtained by writing the third digit (i.e. '2') first, followed by digits '0' to '9', one by one.
- The process continues until we have exhausted all possible two-digit combinations and reached '99'. Then we begin with three-digit combinations. The first three-digit number consists of the lowest two-digit number followed by '0' (i.e. 100), and the process goes on endlessly.

Decimal Number System

- The place values of different digits in a mixed decimal number, starting from the decimal point, are 10^0 , 10^1 , 10^2 and so on (for the integer part) and 10^{-1} , 10^{-2} , 10^{-3} and so on (for the fractional part).
- The value or magnitude of a given decimal number can be expressed as the sum of the various digits multiplied by their place values or weights.
- As an illustration, in the case of the decimal number 3586.265, the integer part (i.e. 3586) can be expressed as

$$3586 = 6 \times 10^0 + 8 \times 10^1 + 5 \times 10^2 + 3 \times 10^3 = 6 + 80 + 500 + 3000 = 3586$$

and the fractional part can be expressed as

$$265 = 2 \times 10^{-1} + 6 \times 10^{-2} + 5 \times 10^{-3} = 0.2 + 0.06 + 0.005 = 0.265$$

Binary Number System

- The binary number system is a radix-2 number system with '0' and '1' as the two independent digits.
- All larger binary numbers are represented in terms of '0' and '1'. The procedure for writing higher order binary numbers after '1' is similar to the one explained in the case of the decimal number system.
- For example, the first 16 numbers in the binary number system would be 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, 1101, 1110 and 1111. The next number after 1111 is 10000, which is the lowest binary number with five digits. This also proves the point that a maximum of only 16 ($= 2^4$) numbers could be written with four digits.

Binary Number System

- Starting from the binary point, the place values of different digits in a mixed binary number are 2^0 , 2^1 , 2^2 and so on (for the integer part) and 2^{-1} , 2^{-2} , 2^{-3} and so on (for the fractional part).
- The left most bit in the binary number is called as the Most Significant Bit (MSB) and it has the largest positional weight. The right most bit is the Least Significant Bit (LSB) and has the smallest positional weight.
- Digits 0 and 1 are called bits and 8 bits together make a byte.

Octal Number System

- The octal number system has a radix of 8 and therefore has eight distinct digits.
- All higher-order numbers are expressed as a combination of these on the same pattern as the one followed in the case of the binary and decimal number systems.
- The independent digits are 0, 1, 2, 3, 4, 5, 6 and 7. The next 10 numbers that follow '7', for example, would be 10, 11, 12, 13, 14, 15, 16, 17, 20 and 21. In fact, if we omit all the numbers containing the digits 8 or 9, or both, from the decimal number system, we end up with an octal number system.
- The place values for the different digits in the octal number system are 8^0 , 8^1 , 8^2 and so on (for the integer part) and 8^{-1} , 8^{-2} , 8^{-3} and so on (for the fractional part).

Hexadecimal Number System

- The hexadecimal number system is a radix-16 number system and its 16 basic digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.
- The place values or weights of different digits in a mixed hexadecimal number are 16^0 , 16^1 , 16^2 and so on (for the integer part) and 16^{-1} , 16^{-2} , 16^{-3} and so on (for the fractional part). The decimal equivalent of A, B, C, D, E and F are 10, 11, 12, 13, 14 and 15 respectively, for obvious reasons.
- The hexadecimal number system provides a condensed way of representing large binary numbers stored and processed inside the computer. One such example is in representing addresses of different memory locations. Let us assume that a machine has 64K of memory.

Hexadecimal Number System

- Such a memory has 64K ($= 2^{16} = 65\,536$) memory locations and needs 65 536 different addresses. These addresses can be designated as 0 to 65 535 in the decimal number system and 00000000 00000000 to 11111111 11111111 in the binary number system.
- The decimal number system is not used in computers and the binary notation here appears too cumbersome and inconvenient to handle.
- In the hexadecimal number system, 65 536 different addresses can be expressed with four digits from 0000 to FFFF. Similarly, the contents of the memory when represented in hexadecimal form are very convenient to handle.

Table -- Binary, Octal, Hexadecimal equivalent of Decimal Numbers

Decimal	Binary	Octal	Hexadecimal
0	0000	000	0000
1	0001	001	0001
2	0010	002	0002
3	0011	003	0003
4	0100	004	0004
5	0101	005	0005
6	0110	006	0006
7	0111	007	0007
8	1000	010	0008
9	1001	011	0009
10	1010	012	A
11	1011	013	B
12	1100	014	C
13	1101	015	D
14	1110	016	E
15	1111	017	F

Conversions – Others to Decimal

- Binary to Decimal Conversion.
- Octal to Decimal Conversion.
- Hexadecimal to Decimal Conversion.

Conversions – Binary to Decimal

Steps	Methods
Step 1	Separate the number into Integral & Fraction part.
Step 2	Take the Integral part, multiple each binary number by 2^n , n value depends on the position of the binary number. Starting from Least Significant Bit (LSB) to Most Significant Bit (MSB) . $2^0 \times (\text{LSB}) + 2^1 \times (\text{next no. after LSB} - (2^{\text{nd}} \text{ digit})) + 2^2 \times (3^{\text{rd}} \text{ digit})$ & so on.
Step 3	Simplify the calculation, you will get the Integral part in Decimal number system
Step 4	Take the Fraction part, multiple each binary number by 2^{-n} , n value depends on the position of the binary number. Starting from first number after decimal, towards the right i.e. till the ending number $2^{-1} \times (\text{LSB}) + 2^{-2} \times (\text{next no. after LSB} - (2^{\text{nd}} \text{ digit})) + 2^{-3} \times (3^{\text{rd}} \text{ digit})$ & so on.
Step 5	Simplify the calculation, you will get the Fraction part in Decimal number system

Conversions – Binary to Decimal

The decimal equivalent of the binary number $(1001.0101)_2$ is determined as follows:

- The integer part = 1001
- The decimal equivalent = $1 \times 2^0 + 0 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 = 1 + 0 + 0 + 8 = 9$
- The fractional part = .0101
- Therefore, the decimal equivalent = $0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} = 0 + 0.25 + 0 + 0.0625 = 0.3125$
- Therefore, the decimal equivalent of $(1001.0101)_2 = 9.3125$

Conversions – Octal to Decimal

Steps	Methods
Step 1	Separate the number into Integral & Fraction part.
Step 2	Take the Integral part, multiple each binary number by 8^n , n value depends on the position of the Octal number. Starting from Right to Left . $8^0 \times (1^{\text{st}} \text{ digit}) + 8^1 \times (2^{\text{nd}} \text{ digit}) + 8^2 \times (3^{\text{rd}} \text{ digit})$ & so on.
Step 3	Simplify the calculation, you will get the Integral part in Decimal number system
Step 4	Take the Fraction part, multiple each binary number by 8^{-n} , n value depends on the position of the Octal number. Starting from first number after decimal, towards the right i.e. till the ending number $8^{-1} \times (1^{\text{st}} \text{ number}) + 8^{-2} \times (2^{\text{nd}} \text{ number}) + 8^{-3} \times (3^{\text{rd}} \text{ number})$ & so on.
Step 5	Simplify the calculation, you will get the Fraction part in Decimal number system

Conversions – Octal to Decimal

- The integer part = 137
- The decimal equivalent = $7 \times 8^0 + 3 \times 8^1 + 1 \times 8^2 = 7 + 24 + 64 = 95$
- The fractional part = .21
- The decimal equivalent = $2 \times 8^{-1} + 1 \times 8^{-2} = 0.265$
- Therefore, the decimal equivalent of $(137.21)_8 = (95.265)_{10}$

Conversions – Hexadecimal to Decimal

Steps	Methods
Step 1	Separate the number into Integral & Fraction part. (Also, change the value to it's number equivalent if a particular alphabet is given in Hexadecimal form, e.g.: write 12 rather than C)
Step 2	Take the Integral part, multiple each binary number by 16^n , n value depends on the position of the Hexadecimal number. Starting from Right to Left . $16^0 \times (1^{\text{st}} \text{ digit}) + 16^1 \times (2^{\text{nd}} \text{ digit}) + 16^2 \times (3^{\text{rd}} \text{ digit})$ & so on.
Step 3	Simplify the calculation, you will get the Integral part in Decimal number system
Step 4	Take the Fraction part, multiple each binary number by 8^{-n} , n value depends on the position of the Hexadecimal number. Starting from first number after decimal, towards the right i.e. till the ending number $16^{-1} \times (1^{\text{st}} \text{ number}) + 16^{-2} \times (2^{\text{nd}} \text{ number}) + 16^{-3} \times (3^{\text{rd}} \text{ number})$ & so on.
Step 5	Simplify the calculation, you will get the Fraction part in Decimal number system

Conversions – Hexadecimal to Decimal

The decimal equivalent of the hexadecimal number $(1E0.2A)_{16}$ is determined as follows:

- The integer part = 1E0
- The decimal equivalent = $0 \times 16^0 + 14 \times 16^1 + 1 \times 16^2 = 0 + 224 + 256 = 480$
- The fractional part = 2A
- The decimal equivalent = $2 \times 16^{-1} + 10 \times 16^{-2} = 0.164$
- Therefore, the decimal equivalent of $(1E0.2A)_{16} = (480.164)_{10}$

Conversions – Decimal to Others

- Decimal to Binary Conversion.
- Decimal to Octal Conversion.
- Decimal to Hexadecimal Conversion.

Conversions – Decimal to Binary

Steps	Methods
Step 1	Separate the number into Integral & Fraction part.
Step 2	Take the Integral part, Create 3 columns as – Divisor, Dividend & Remainder . We perform successive division (step by step). As we need to divide the given number write it under Dividend column. As it is to converted to Binary , <u>divide it by 2</u> . Perform the division as shown on <u>slide 27</u> .
Step 3	Now, write down the remainders obtain one by one in the Remainder column . Remember the last remainder will be the MSB while the very first remainder will be the LSB . This way the Integral part can be converted.
Step 4	Take the Fraction part, multiple it by 2. After multiplying it by 2 – note down the new number. Take only the Integral part of that number. i.e. number written before decimal, note it down.
Step 5	Take the fractional part from the previous new number obtained by multiplying by 2. Again repeated the same process. Note down the Integral part of the new number. Repeat this Steps until get the desired result.
Step 6	Now, from the very first noted number to the last noted number obtain by this fractional multiplication is the Fractional part of the number.

Conversions – Decimal to Binary

- The integer part = 13

Divisor	Dividend	Remainder
2	13	—
2	6	1
2	3	0
2	1	1
—	0	1

- The binary equivalent of $(13)_{10}$ is therefore $(1101)_2$
- The fractional part = .375
- $0.375 \times 2 = 0.75$ with a carry of 0
- $0.75 \times 2 = 0.5$ with a carry of 1
- $0.5 \times 2 = 0$ with a carry of 1
- The binary equivalent of $(0.375)_{10} = (.011)_2$
- Therefore, the binary equivalent of $(13.375)_{10} = (1101.011)_2$

Conversions – Decimal to Octal

Steps	Methods
Step 1	Separate the number into Integral & Fraction part.
Step 2	Take the Integral part, Create 3 columns as – Divisor, Dividend & Remainder . We perform successive division (step by step). As we need to divide the given number write it under Dividend column. As it is to converted to Octal , <u>divide it by 8</u> . Perform the division as shown on <u>slide 29</u> .
Step 3	Now, write down the remainders obtain one by one in the Remainder column . Remember <u>the last remainder will be the first number horizontally while the very first remainder will be the last number horizontally</u> . This way the Integral part can be converted.
Step 4	Take the Fraction part, <u>multiple it by 8</u> . After multiplying it by 8 – note down the new number. Take only the Integral part of that number. i.e. number written before decimal, note it down.
Step 5	Take the fractional part from the previous new number obtained by multiplying by 8. Again repeated the same process. Note down the Integral part of the new number. Repeat this Steps until get the desired result.
Step 6	Now, from the very first noted number to the last noted number obtain by this fractional multiplication is the Fractional part of the number.

Conversions – Decimal to Octal

- The integer part = 73

Divisor	Dividend	Remainder
8	73	—
8	9	1
8	1	1
—	0	1

- The octal equivalent of $(73)_{10} = (111)_8$
- The fractional part = 0.75
- $0.75 \times 8 = 6$ with a carry of 0
- The octal equivalent of $(0.75)_{10} = (.6)_8$
- Therefore, the octal equivalent of $(73.75)_{10} = (111.6)_8$

Conversions – Decimal to Hexadecimal

Steps	Methods
Step 1	Separate the number into Integral & Fraction part.
Step 2	Take the Integral part, Create 3 columns as – Divisor, Dividend & Remainder . We perform successive division (step by step). As we need to divide the given number write it under Dividend column. As it is to converted to Hexadecimal , <u>divide it by 16</u> . Perform the division as shown on <u>slide 31</u> .
Step 3	Now, write down the remainders obtain one by one in the Remainder column . Remember <u>the last remainder will be the first number horizontally while the very first remainder will be the last number horizontally</u> . This way the Integral part can be converted.
Step 4	Take the Fraction part, <u>multiple it by 16</u> . After multiplying it by 16 – note down the new number. Take only the Integral part of that number. i.e. number written before decimal, note it down.
Step 5	Take the fractional part from the previous new number obtained by multiplying by 16. Again repeated the same process. Note down the Integral part of the new number. Repeat this Steps until get the desired result.
Step 6	Now, from the very first noted number to the last noted number obtain by this fractional multiplication is the Fractional part of the number.

Conversions – Decimal to Hexadecimal

- The integer part = 82

Divisor	Dividend	Remainder
16	82	—
16	5	2
—	0	5

- The hexadecimal equivalent of $(82)_{10} = (52)_{16}$
- The fractional part = 0.25
- $0.25 \times 16 = 4$ with a carry of 0
- Therefore, the hexadecimal equivalent of $(82.25)_{10} = (52.4)_{16}$

Signed Binary numbers

In the decimal number system a plus (+) sign is used to denote a positive number and a minus (−) sign for denoting a negative number. The plus sign is usually dropped, and the absence of any sign means that the number has positive value. This representation of numbers is known as *signed number*. As is well known, digital circuits can understand only two symbols, 0 and 1; therefore, we must use the same symbols to indicate the sign of the number also. Normally, an additional bit is used as the *sign bit* and it is placed as the most significant bit. A 0 is used to represent a positive number and a 1 to represent a negative number. For example, an 8-bit signed number 01000100 represents a positive number and its value (magnitude) is $(1000100)_2 = (68)_{10}$. The left most 0 (MSB) indicates that the number is positive. On the other hand, in the signed binary form, 11000100 represents a negative number with magnitude $(1000100)_2 = (68)_{10}$. The 1 in the left most position (MSB) indicates that the number is negative and the other seven bits give its magnitude. This kind of representation for signed numbers is known as *sign-magnitude representation*. The user must take care to see the representation used while dealing with the binary numbers.

Signed Binary numbers – examples

Find the decimal equivalent of the following binary numbers assuming sign-magnitude representation of the binary numbers.

(a) 101100 (b) 001000 (c) 0111 (d) 1111

Solution

(a) Sign bit is 1, which means the number is negative.

$$\begin{aligned} \therefore \quad \text{Magnitude} &= 01100 = (12)_{10} \\ (101100)_2 &= (-12)_{10} \end{aligned}$$

(b) Sign bit is 0, which means the number is positive.

$$\begin{aligned} \therefore \quad \text{Magnitude} &= 01000 = 8 \\ (001000)_2 &= (+8)_{10} \end{aligned}$$

$$(c) (0111)_2 = (+7)_2$$

$$(d) (1111)_2 = (-7)_2$$

One's complement

In a binary number, if each 1 is replaced by 0 and each 0 by 1, the resulting number is known as the *one's complement* of the first number. In fact, both the numbers are complement of each other. If one of these numbers is positive, then the other number will be negative with the same magnitude and vice-versa. For example, $(0101)_2$ represents $(+5)_{10}$, whereas $(1010)_2$ represents $(-5)_{10}$ in this representation. This method is widely used for representing signed numbers. In this representation also, MSB is 0 for positive numbers and 1 for negative numbers.

Find the one's complement of the following binary numbers.

(a) 0100111001 (b) 11011010

Solution

(a) 1011000110 (b) 00100101

One's complement - examples

Represent the following numbers in one's complement form.

(a) +7 and -7 (b) +8 and -8 (c) +15 and -15

Solution

In one's complement representation,

$$(a) (+7)_{10} = (0111)_2 \quad \text{and} \quad (-7)_{10} = (1000)_2$$

$$(b) (+8)_{10} = (01000)_2 \quad \text{and} \quad (-8)_{10} = (10111)_2$$

$$(c) (+15)_{10} = (01111)_2 \quad \text{and} \quad (-15)_{10} = (10000)_2$$

Two's complement

If 1 is added to 1's complement of a binary number, the resulting number is known as the *two's complement* of the binary number. For example, 2's complement of 0101 is 1011. Since 0101 represents $(+5)_{10}$, therefore, 1011 represents $(-5)_{10}$ in 2's complement representation. In this representation also, if the MSB is 0 the number is positive, whereas if the MSB is 1 the number is negative. For an n -bit number, the maximum positive number which can be represented in 2's complement form is $(2^{n-1} - 1)$ and the maximum negative number is -2^{n-1} . Table 2.3 gives sign-magnitude, 1's and 2's complement numbers represented by 4-bit binary numbers. From the table, it is observed that the maximum positive number is $0111 = +7$, whereas the maximum negative number is $1000 = -8$ using four bits in 2's complement format.

Two's complement - example

(i) 01001110 (ii) 00110101

Solution

(i)	Number	0 1 0 0 1 1 1 0
	1's complement	1 0 1 1 0 0 0 1
	Add 1	$\begin{array}{r} 1 \\ \hline 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0 \end{array}$

(ii)	Number	0 0 1 1 0 1 0 1
	1's complement	1 1 0 0 1 0 1 0
	Add 1	$\begin{array}{r} 1 \\ \hline 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1 \end{array}$

Two's complement - example

Represent $(-17)_{10}$ in

- (i) Sign-magnitude,
- (ii) one's complement,
- (iii) two's complement representation.

Solution

The minimum number of bits required to represent $(+17)_{10}$ in signed number format is six.

\therefore

$$(+17)_{10} = (010001)_2$$

Therefore, $(-17)_{10}$ is represented by

- (i) 110001 in sign-magnitude representation.
- (ii) 101110 in 1's complement representation.
- (iii) 101111 in 2's complement representation.

One's & Two's complement representation

Decimal number	Binary number		
	Sign-magnitude	One's complement	Two's complement
0	0000	0000	0000
1	0001	0001	0001
2	0010	0010	0010
3	0011	0011	0011
4	0100	0100	0100
5	0101	0101	0101
6	0110	0110	0110
7	0111	0111	0111
-8	—	—	1000
-7	1111	1000	1001
-6	1110	1001	1010
-5	1101	1010	1011
-4	1100	1011	1100
-3	1011	1100	1101
-2	1010	1101	1110
-1	1001	1110	1111
-0	1000	1111	—

Binary Arithmetic

- **Augend** - The number to which another is added.
- **Addend** - A number which is added to another.
- **Sum** – Addition of that two numbers.
- **Carry** - A digit that is transferred from one column of digits to another column of more significant digits.

Binary Arithmetic – Binary Addition

Augend	Addend	Sum	Carry	Result
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	0	1	10

Binary Arithmetic – Binary Addition

Add the binary numbers:

(i) 1011 and 1100 (ii) 0101 and 1111

Solution

(i)

	1	0	1	1
(+)	1	1	0	0
<hr/>				
	1	0	1	1

↑
carry

(ii)

	(1)	(1)	(1)	← carry
	0	1	0	1
(+)	1	1	1	1
<hr/>				
	1	0	1	0

↑
carry

Binary Arithmetic – Binary Addition

- Try & Add this Binary number

Add the binary numbers:

0 1 1 0 1 0 1 0

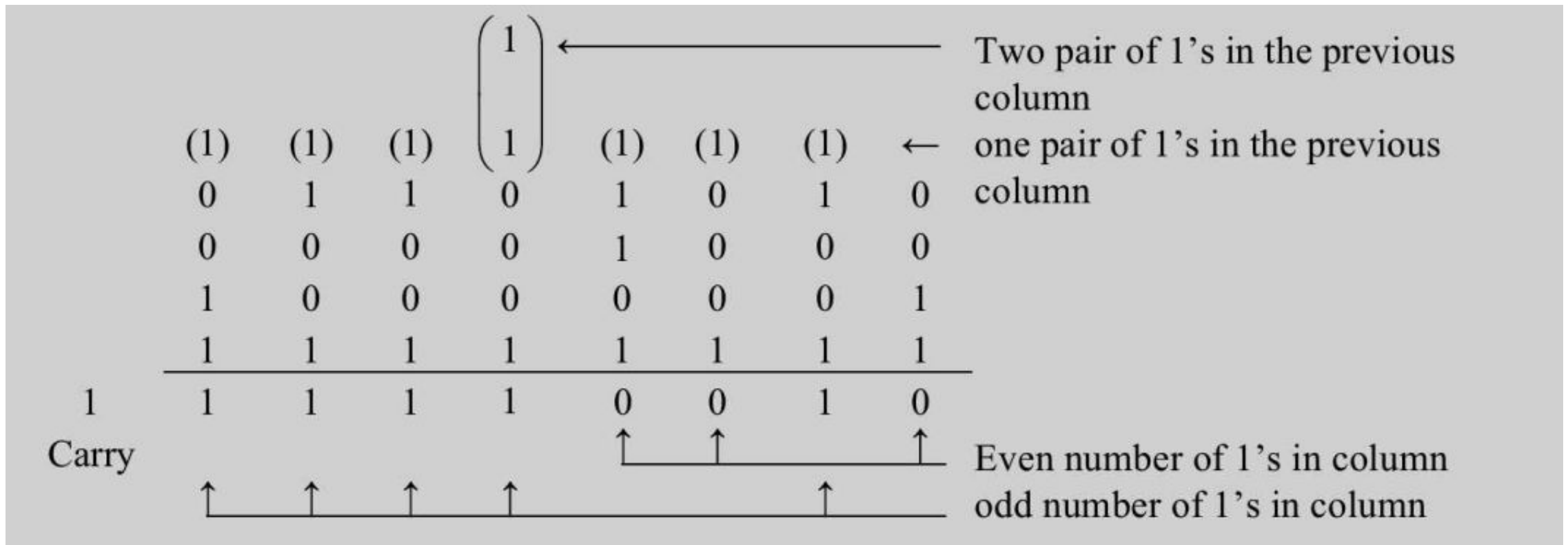
0 0 0 0 1 0 0 0

1 0 0 0 0 0 0 1

1 1 1 1 1 1 1 1

Binary Arithmetic – Binary Addition

- Solution : The sum is **111110010**



Binary Arithmetic – Binary Subtraction

- **Minuend** - A quantity or number from which another is to be subtracted.
- **Subtrahend** - A quantity or number to be subtracted from another.
- **Difference** – The quantity of numbers between two numbers.
- **Borrow** - to take (one) from a digit of the minuend in arithmetical subtraction in order to add as 10 to the digit holding the next lower place.

Binary Arithmetic – Binary Subtraction

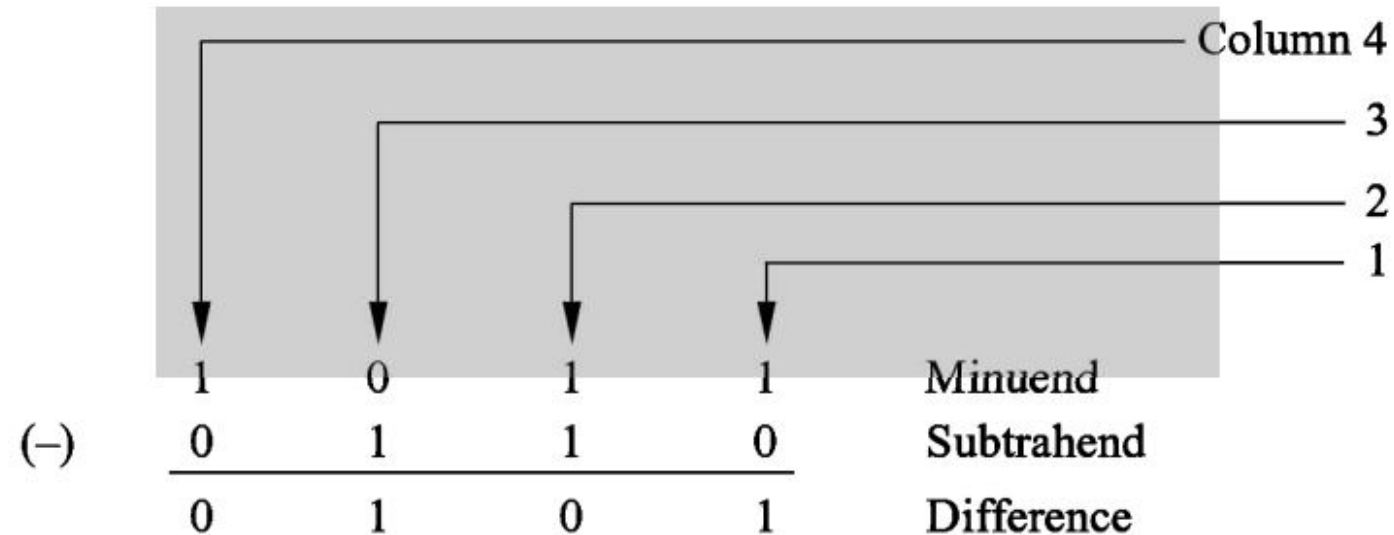
Minuend	Subtrahend	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Binary Arithmetic – Binary Subtraction

Perform the following subtraction:

$$\begin{array}{r} 1011 \\ - 0110 \\ \hline \end{array}$$

Solution



Here, in columns 1 and 2, borrow = 0 and in column 3 it is 1. Therefore, in column 4 first subtract 0 from 1 and from this result obtained subtract the borrow bit.

Binary Arithmetic – Binary Multiplication

- **Multiplicand** - A quantity which is to be multiplied by another (the multiplier)
- **Multiplier** - A quantity by which a given number (the multiplicand) is to be multiplied.
- **Product** – Multiplication of **Multiplicand & Multiplier**.

Binary Arithmetic – Binary Multiplication

Multiplicand	Multiplier	Product
0	0	0
0	1	0
1	0	0
1	1	1

Binary Arithmetic – Binary Multiplication

1 0 0 1		Multiplicand	
× 1 1 0 1		Multiplier	
<hr/>			
1 0 0 1	} I	Partial Products	
0 0 0 0			II
1 0 0 1			III
1 0 0 1			IV
<hr/>			
1 1 1 0 1 0 1		Final Product	

Binary Arithmetic – Binary Division


$$\begin{array}{r} \text{Divisor} \rightarrow 1001 \overline{) 1110101} \\ \underline{1001} \\ 1011 \\ \underline{1001} \\ 001001 \\ \underline{1001} \\ 0000 \end{array} \quad \begin{array}{l} \leftarrow \text{Quotient} \\ \leftarrow \text{Dividend} \end{array}$$

Binary Subtraction using 2's complement Representation.

Perform binary subtraction using 2's complement representation of negative numbers.

Solution

(i)

7		0 1 1 1	Minuend
<u>-5</u>		<u>(+) 1 0 1 1</u>	2's complement of subtrahend
+2		1 0 0 1 0	
		↑	
		Discard final carry	

The answer is 0 0 1 0 equivalent to $(+2)_{10}$.

Binary Subtraction using 2's complement Representation.

(ii)

$$\begin{array}{r} 5 \\ -7 \\ \hline -2 \end{array} \Rightarrow \begin{array}{r} 0 \ 1 \ 0 \ 1 \\ (+) \ 1 \ 0 \ 0 \ 1 \\ \hline 1 \ 1 \ 1 \ 0 \end{array}$$

Minuend
2's complement of subtrahend

The final carry = 0. Therefore, the answer is negative and is in 2's complement form. 2's complement of 1110 = 0010

Therefore, the answer is $(-2)_{10}$.

Binary Subtraction using 2's complement Representation.

Rules:

- 1. When first number i.e. Minuend is +ve & Second number i.e. Subtrahend is -ve, 2's complement is performed on the Second number i.e. Subtrahend & then both numbers are added.**
- 2. When first number i.e. Minuend is -ve & second number i.e. Subtrahend is +ve, 2's complement is performed on the first number i.e. Minuend & then both numbers are added.**
- 3. When both first number i.e. Minuend & Second number i.e. Subtrahend is -ve, 2's complement is performed on both first & second number & then both numbers are added.**

Binary Subtraction using 2's complement Representation – Rules & Overflow

- (a) If the two operands are of the opposite sign, the result is to be obtained by the rule of subtraction using 2's complement (Sec. 2.6.1)
- (b) If the two operands are of the same sign, the sign bit of the result (MSB) is to be compared with the sign bit of the operands. In case the sign bits are same, the result is correct and is in 2's complement form. If the sign bits are not same there is a problem of *overflow*, i.e. the result can not be accommodated using eight bits and the result is to be interpreted suitably. The result in this case will consist of nine bits, i.e. carry and eight bits, and the carry bit will give the sign of the number.

Binary Subtraction using 2's complement Representation - Examples

Perform the following operations using 2's complement method:

(i) $48 - 23$ (ii) $23 - 48$ (iii) $48 - (-23)$ (iv) $-48 - 23$

Use 8-bit representation of numbers.

Solution

(i) 2's complement representation of $+48 = 00110000$

2's complement representation of $-23 = 11101001$

$48 + (-23)$

$$\begin{array}{rcl} \Rightarrow & \begin{array}{r} 48 \\ + (-23) \\ \hline + 25 \end{array} & \Rightarrow \begin{array}{r} 00110000 \\ (+) 11101001 \\ \hline 100011001 \end{array} \end{array} \quad = +25$$

↑
Discard Carry

Binary Subtraction using 2's complement Representation - Examples

(ii) 2's complement representation of $+23 = 00010111$

2's complement representation of $-48 = 11010000$

$23 - 48 = 23 + (-48)$

$$\begin{array}{r} \Rightarrow \begin{array}{r} 23 \\ + (-48) \\ \hline -25 \end{array} \Rightarrow \begin{array}{r} 00010111 \\ (+) 11010000 \\ \hline 11100111 \end{array} \Rightarrow -25 \end{array}$$

(iii) $48 - (-23) = 48 + 23$

$$\begin{array}{r} \Rightarrow \begin{array}{r} 48 \\ + (23) \\ \hline +71 \end{array} \Rightarrow \begin{array}{r} 00110000 \\ (+) 00010111 \\ \hline 01000111 \end{array} \Rightarrow +71 \end{array}$$

Binary Subtraction using 2's complement Representation - Examples

(iv) $-48 - 23 = (-48) + (-23)$

\Rightarrow

-48	
$+ (-23)$	
<hr/>	
-71	

\Rightarrow

	1	1	0	1	0	0	0	0
(+)	1	1	1	0	1	0	0	1
<hr/>	1	1	0	1	1	1	0	0
	1	1	0	1	1	1	0	0

$\Rightarrow -71$

\uparrow
Carry be ignored

Weighted Codes

- Each digit position of the number represents a specific weight.
- Weighted binary codes are those binary codes which obey the positional weight principle.
- Each position of the number represents a specific weight. Several systems of the codes are used to express the decimal digits 0 through 9.
- In these codes each decimal digit is represented by a group of four bits.
- Example: **Straight Binary Code, BCD code.**

Non-Weighted Codes

- Non weighted codes are codes that are not placed weighted.
- It means that each position within the binary number is not assigned a fixed value. Excess-3 and Gray codes are examples of non-weighted binary codes.
- In an unweighted code, no weighting is assigned to the position of a bit.
- There is simply a look-up table for correspondences between the objects to be coded and their binary representation.
- Example: **Excess-3 Code, Gray Code.**

CODES

- Straight Binary Code**
- Binary-coded decimal Code(BCD Code)**
- Excess-3 Code**
- Gray Code**

Decimal Number	Binary				BCD				Excess-3				Gray			
	B_3	B_2	B_1	B_0	D	C	B	A	E_3	E_2	E_1	E_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
1	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	1
2	0	0	1	0	0	0	1	0	0	1	0	1	0	0	1	1
3	0	0	1	1	0	0	1	1	0	1	1	0	0	0	1	0
4	0	1	0	0	0	1	0	0	0	1	1	1	0	1	1	0
5	0	1	0	1	0	1	0	1	1	0	0	0	0	1	1	1
6	0	1	1	0	0	1	1	0	1	0	0	1	0	1	0	1
7	0	1	1	1	0	1	1	1	1	0	1	0	0	1	0	0
8	1	0	0	0	1	0	0	0	1	0	1	1	1	1	0	0
9	1	0	0	1	1	0	0	1	1	1	0	0	1	1	0	1
10	1	0	1	0									1	1	1	1
11	1	0	1	1									1	1	1	0
12	1	1	0	0									1	0	1	0
13	1	1	0	1									1	0	1	1
14	1	1	1	0									1	0	0	1
15	1	1	1	1									1	0	0	0

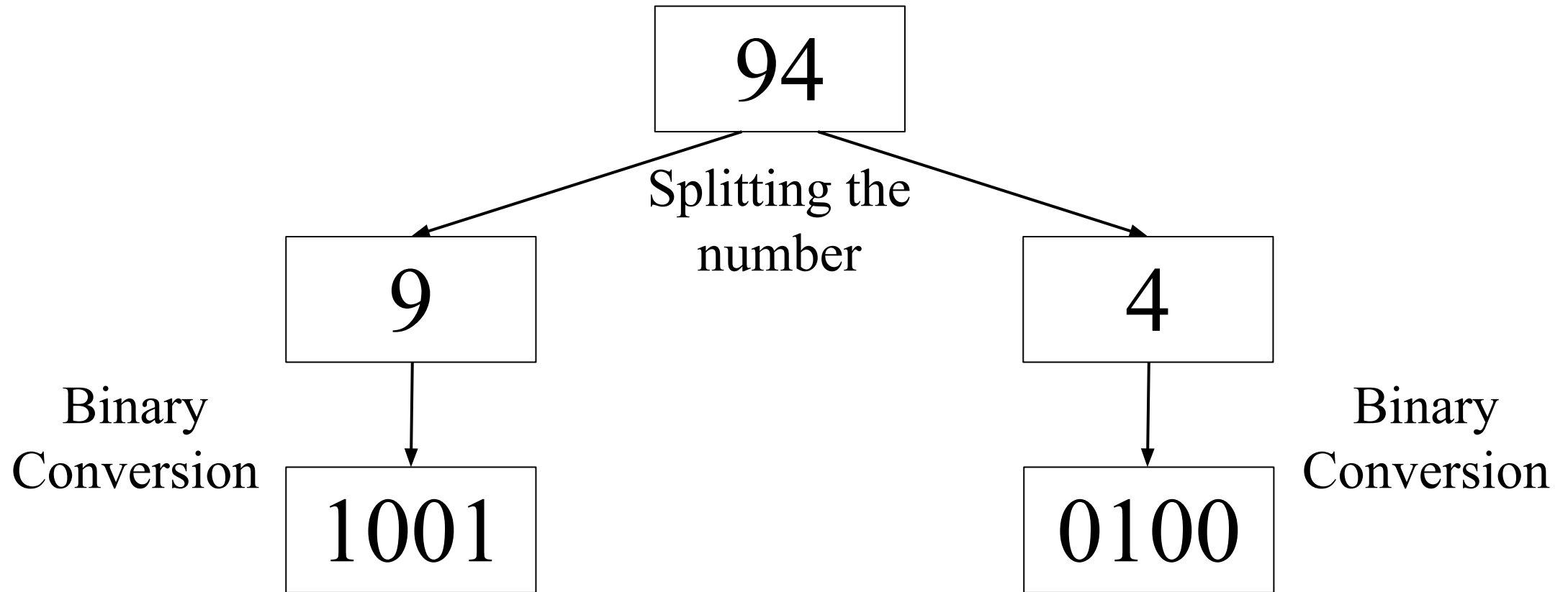
Straight Binary Code

- A binary code represents text, computer processor instructions, or any other data using a two-symbol system. The two-symbol system used is often "0" and "1" from the binary number system.
- **Straight Binary Codes** is a weighted code.

BCD Code

- Binary-coded decimal is a system of writing numerals that assigns a four-digit binary code to each digit 0 through 9 in a decimal (base 10) number.
- Simply put, binary-coded decimal is a way to convert decimal numbers into their binary equivalents.
- **BCD Code** is a weighted code.

BCD Code



Excess - 3 Code

- This is another form of BCD code, in which each decimal digit is coded into a 4-bit binary code. The code for each decimal digit is obtained by adding decimal 3 to the natural BCD code of the digit.
- For example, decimal 2 is coded as $0010 + 0011 = 0101$ in Excess-3 code.
- **Excess - 3 Code** is not a weighted code.
- Excess-3 codes for decimal digits 0 through 9 are given in Table on further slides.

Gray Code

- In unit distance code, bit patterns for two consecutive numbers differ in only one bit position.
- For example, the Gray code for decimal number 5 is 0111 and for 6 is 0101. These two codes differ by only one bit position (third from the left). This code is used extensively for shaft encoders because of this property.
- **Gray Code** is not a weighted code.
- It is also called as **Reflected Code** or **Cyclic Code**.

Gray Code Construction

(a) 1-bit Gray code is constructed using (i) above.

<i>Decimal number</i>	<i>Gray code</i>
0	0
1	1

(b) 2-bit Gray code is constructed using (ii) and (iii) above and Gray code of 1-bit

<i>Decimal number</i>	<i>Gray code</i>
0	0 0
1	<u>0</u> <u>1</u>
2	1 1
3	1 0

CODES – Gray Code Construction

(c) 3-bit Gray code is constructed using 2-bit Gray code.

<i>Decimal number</i>	<i>Gray code</i>
0	0 0 0
1	0 0 1
2	0 1 1
3	<u>0</u> <u>1</u> <u>0</u>
4	1 1 0
5	1 1 1
6	1 0 1
7	1 0 0

CODES – Gray Code Construction

(c) 3-bit Gray code is constructed using 2-bit Gray code.

<i>Decimal number</i>	<i>Gray code</i>
0	0 0 0
1	0 0 1
2	0 1 1
3	<u>0</u> <u>1</u> <u>0</u>
4	1 1 0
5	1 1 1
6	1 0 1
7	1 0 0

Code Conversions

- **Binary to Gray Code Conversion.**

~~Decimal~~ **to Gray Code** (Simply, do Binary to Decimal & vice versa as required)

- **Gray to Binary Code Conversion.**

~~Gray Code~~ **to Decimal** (Simply, do Binary to Decimal & vice versa as required)

- **Decimal to Excess – 3 Code Conversion.**

- **Excess – 3 to Decimal Code Conversion.**

Binary to Gray Code Conversion

- **Step 1** : The MSB in gray code is the same as in binary code.
- **Step 2** : Either perform – ‘**Binary Addition**’ or ‘**EX-OR gate operation**’

From Left to Right, go on performing **Binary Addition/ EX-OR gate operation** on each adjacent pair of binary bit to get next gray code & discard carry if performing Binary addition.

Binary to Gray Code Conversion

	B_4	B_3	B_2	B_1	B_0
Binary Code	1	1	0	0	1
Write MSB as it is, Perform EX-OR/Binary Addition for rest	B_4				
	1				
Gray Code	1	0	1	0	1

Gray to Binary Code Conversion

- **Step 1** : The MSB in binary code is the same as in gray code.
- **Step 2** : Either perform – ‘**Binary Addition**’ or ‘**EX-OR gate operation**’

From Left to Right, go on performing **Binary Addition/ EX-OR gate operation**, add each binary digit generated to the gray digit in the next adjacent position & discard carry.

NOTE: This is different then **Binary to Gray** as operation is performed on each generated binary & existing adjacent gray code.

Gray to Binary Code Conversion

	G_4	G_3	G_2	G_1	G_0
Gray Code	1	0	1	0	1
Write MSB as it is, Perform EX-OR/Binary Addition for rest	G_4				
	1				
Binary equivalent	B_4	B_3	B_2	B_1	B_0
Binary Code	1	1	0	0	1

Decimal to Excess – 3 Code Conversion

- This is another form of BCD code, in which each decimal digit is coded into a 4-bit binary code.
- **Step 1** : First add 3 to each decimal digit.
- **Step 2** : After adding 3 to each digit of Decimal number.
- **Step 3** : Convert each digit to BCD (i.e. write down the BCD value of every digit).

Decimal to Excess – 3 Code Conversion

$$\begin{array}{ccc} 4 & 6 & 5 \\ +3 & +3 & +3 \\ =7 & =9 & =8 \\ \underline{0111} & \underline{1001} & \underline{1000} \end{array}$$

$$\therefore (465)_{10} = (\underline{011110011000})_{XS3}$$

$$\begin{array}{ccc} 3 & 9 & 6 \\ +3 & +3 & +3 \\ =6 & =12 & =9 \\ \underline{0110} & \underline{1100} & \underline{1001} \end{array}$$

$$\therefore (396)_{10} = (\underline{011011001001})_{XS3}$$

Excess – 3 to Decimal Code Conversion

- **Conversion STEPS.**
- **Step 1 :** Separate group of 4 bits, each 4-bit group representing a Decimal digit.
- **Step 2 :** Convert each digit to Decimal.
- **Step 3 :** Subtract 3 from each decimal digit.

Excess – 3 to Decimal Code Conversion

$$(011011001001)_{\text{Excess 3}} = (\underline{\quad? \quad})_{10}$$

$$(011110011000)_{\text{XS3}} = (\underline{\quad\quad\quad})_{10}$$

Solution:

$$\begin{array}{ccc} \underline{0111} & \underline{1001} & \underline{1000} \end{array}$$

$$\begin{array}{ccc} 7 & 9 & 8 \\ -3 & -3 & -3 \\ \hline =4 & =6 & =5 \end{array}$$

$$\therefore (011110011000)_{\text{XS3}} = (\underline{465})_{10}$$

$$(011011001001)_{\text{XS3}} = (\underline{\quad\quad\quad})_{10}$$

$$\begin{array}{ccc} \underline{0110} & \underline{1100} & \underline{1001} \end{array}$$

$$\begin{array}{ccc} 6 & 12 & 9 \\ -3 & -3 & -3 \\ \hline =3 & =9 & =6 \end{array}$$

$$\therefore (011011001001)_{\text{XS3}} = (\underline{396})_{10}$$

Alphanumeric Codes

- It only represents the states “0” or “1”, a binary digit or bit can only represent two symbols which is insufficient to computer-to-computer communication because there are many more symbols required.
- **26 different alphabets** with capital & tiny letters, 0 to 9 digit numerals, punctuation marks, and other symbols must all be represented.
- **The code that stands for numbers & alphabetic characters are known as Alphanumeric Codes.**
- A minimum of 36 items – 10 numerals & 26 letters of the alphabet, shall be represented by alphanumeric code.

Alphanumeric Codes & it's types

- Three Alphanumeric Codes listed below are used for Data Representation.
- 1. American Standard Code for Information Technology Interchange (ASCII).**
 - 2. Extended Binary Coded Decimal Interchange Code (EBCDIC).**
 - 3. Five bit Baudot Code.**

1. American Standard Code for Information Technology Interchange (ASCII).

- It is **7-bit** or **8-bit** alphanumeric code.
- **7-bit** code is Standard ASCII supports **127 characters**.
- Standard ASCII series starts from **00h to 7Fh**, where 00h-1Fh are used as control characters and 20h-7Fh as graphic symbols.
- **8-bit** code is Extended ASCII supports **256 characters** where special graphics and math's symbols are added.
- Extended ASCII series starts from **80h to FFh**.

2. Extended Binary Coded Decimal Interchange Code (EBCDIC).

- It is **8-bit** alphanumeric code developed by IBM, .
- Mostly used in IBM mainframe computers, also on midrange computer OS.
- EBCDIC descended from code used with punched cards binary code used with most of IBM's computer peripherals(other areas) of the late 1950s & early 1960s.
- It is also supported on various non-IBM platforms such as Fujitsu-Siemens' BS2000/ OSD, OS-IV, MSP, and MSP-EX, the SDS Sigma series, and Unisys VS/9 and MCP.

3. Five bit Baudot Code

- The Baudot code is an early character encoding for telegraphy (the science or practice of using or constructing communication systems for the transmission or reproduction of information) invented by Émile Baudot in the 1870s.
- Baudot's **5-bit** code was adapted to be sent from a manual keyboard, and no teleprinter equipment was ever constructed that used it in its original form.
- It is binary code which uses crosses and dots. It was used for teleprinter messages instead of the Morse code and allowed to encode $2^5 = 32$ characters efficiently.

Number Representation & Storage in computer system.

- Computer system recognizes only binary numbering system. That is why we need an interpretation who can convert human terminology in binary and result of computer system's to human understandable language.
- In addition to Binary numbering system, other numbering system such as Hexadecimal, EBCDIC, ASCII etc. must have to be understand **EBCDIC & ASCII**.
- The **28 special** characters that can be used in a **6-bit BCD** code are frequently insufficient for modern data processing. (Some data processing tools might even require letters in both lower & uppercase.)

Number Representation & Storage in computer system.

- As a result the **8-bit** codes have been created . As demonstrated, the standard division of each coded character or byte into 4 zone bits & **8-4-2-1 bits**.
- Any set of **8-bit** is referred to as a “**byte**”.
- A byte can be represented by two Hexadecimal digits, the first of which stands for the zone bits and the second for the numeric bits.
- Today’s computer industry mostly uses two **8-bit BCD**

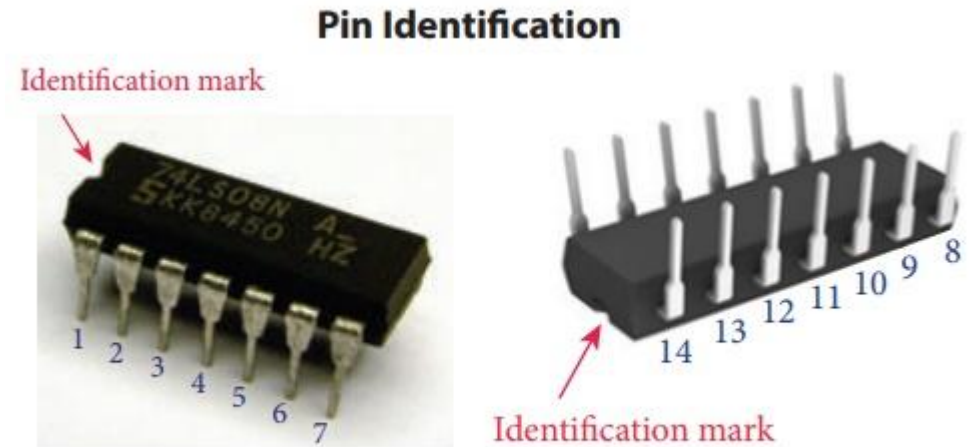
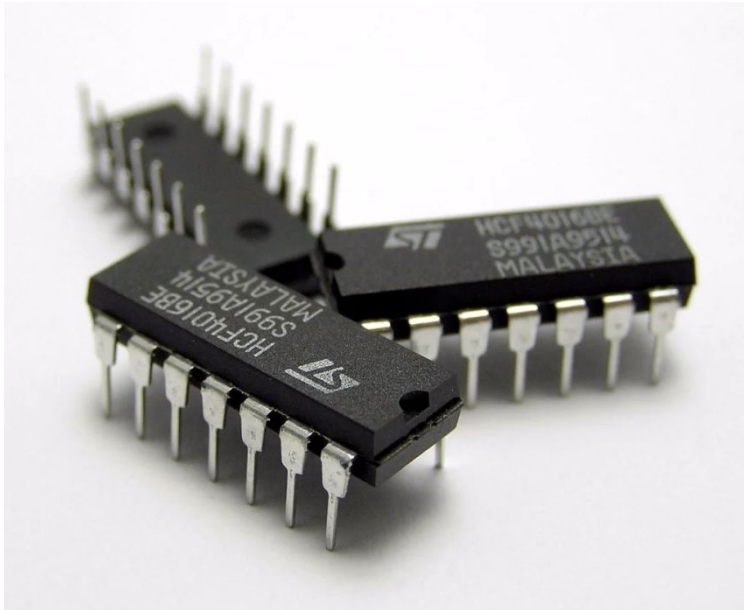
Z	Z	Z	Z	8	4	2	1
----------	----------	----------	----------	----------	----------	----------	----------

Number Representation & Storage in computer system.

- **Zone bit:** Four-bit zones, with one zone indicating the type of character, digit, punctuation mark, lowercase letter, capital letter, and so on, and the other zone.

2. Logic Gates and Related Devices

- Basic and Universal Gates



Positive & Negative Logic

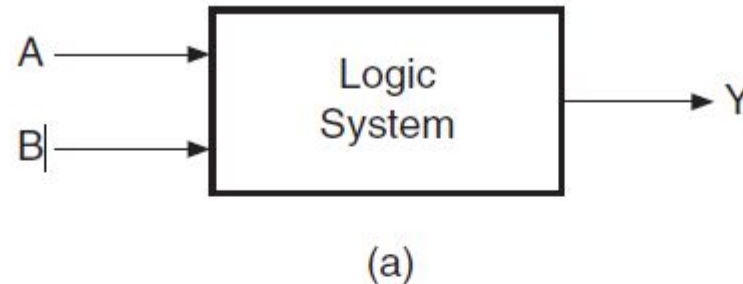
- The binary variables, as we know, can have either of the **two states, i.e. the logic '0' state or the logic '1' state.** These logic states in digital systems such as computers, for instance, are represented by two different voltage levels or two different current levels.
- If the more positive of the two voltage or current levels represents a logic '1' and the less positive of the two levels represents a logic '0', then the logic system is referred to as a **positive logic system.**
- If the more positive of the two voltage or current levels represents a logic '0' and the less positive of the two levels represents a logic '1', then the logic system is referred to as a **negative logic system.**

Truth Table

- A **truth table** lists all possible combinations of input binary variables and the corresponding outputs of a logic system.
- The logic system output can be found from the logic expression, often referred to as the Boolean expression, that relates the output with the inputs of that very logic system.
- When the number of input binary variables is only one, then there are only two possible inputs, i.e. '0' and '1'. If the number of inputs is two, there can be four possible input combinations, i.e. 00, 01, 10 and 11.

Truth Table

- This statement can be generalized to say that, if a logic circuit has n binary inputs, its truth table will have 2^n possible input combinations, or in other words 2^n rows.



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

Logic Gates

- The logic gate is the most basic building block of any digital system, including computers. Each one of the basic logic gates is a piece of hardware or an electronic circuit that can be used to implement some basic logic expression.
- While laws of Boolean algebra could be used to do manipulation with binary variables and simplify logic expressions, these are actually implemented in a digital system with the help of electronic circuits called logic gates.

Types of Logic Gates

Logic Gate Symbols



OR



NOR



AND



NAND



XOR



XNOR

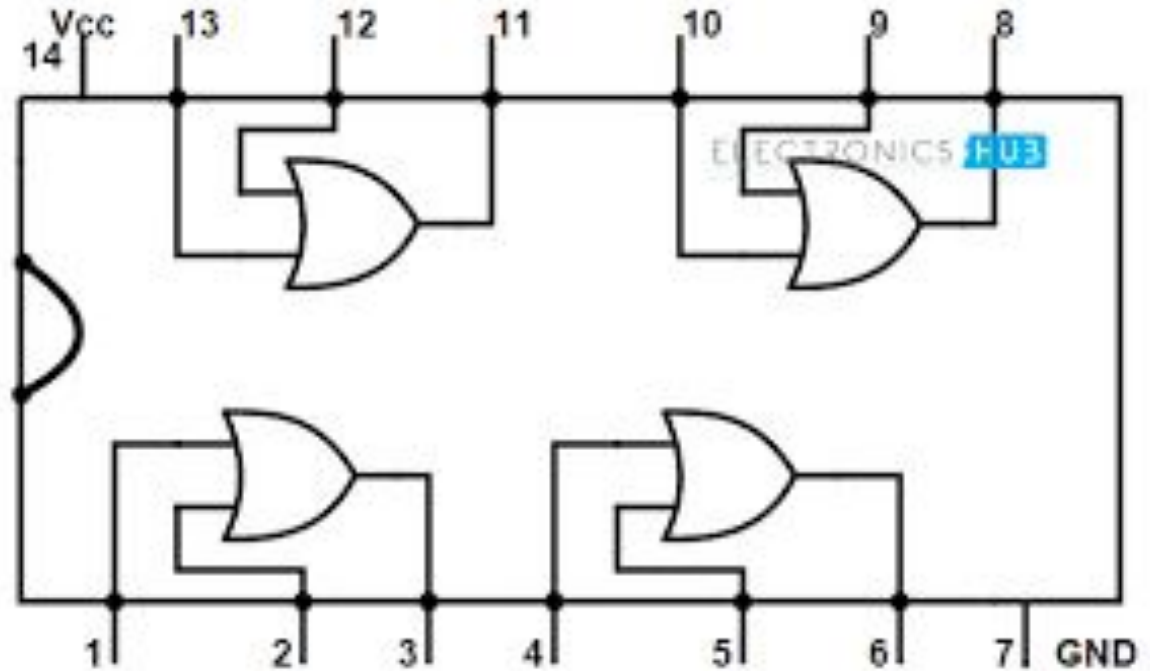
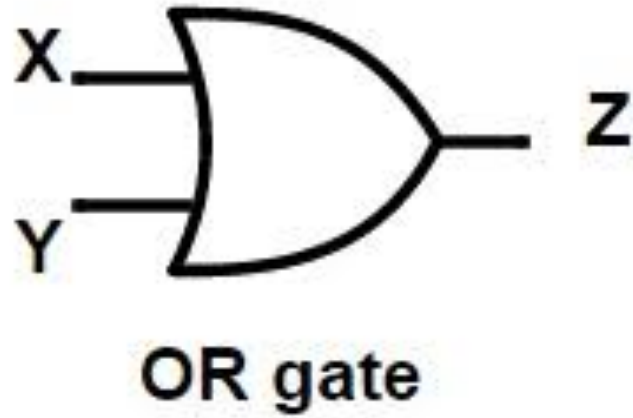


Buffer



NOT

1) OR Gate - 7432

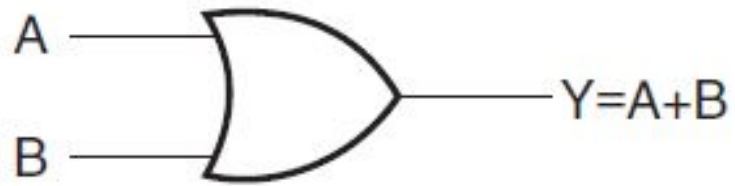


1) OR Gate - 7432

- The OR operation on two independent logic variables A and B is written as $Y = A+B$ and reads as Y equals A OR B and not as A plus B.
- An OR gate is a logic circuit with two or more inputs and one output. The output of an OR gate is LOW only when all of its inputs are LOW. For all other possible input combinations, the output is HIGH.
- The output of an OR gate is a logic '0' only when all of its inputs are at logic '0'. For all other possible input combinations, the output is a logic '1'.
- The operation of a two-input OR gate is explained by the logic expression

$$Y = A+B$$

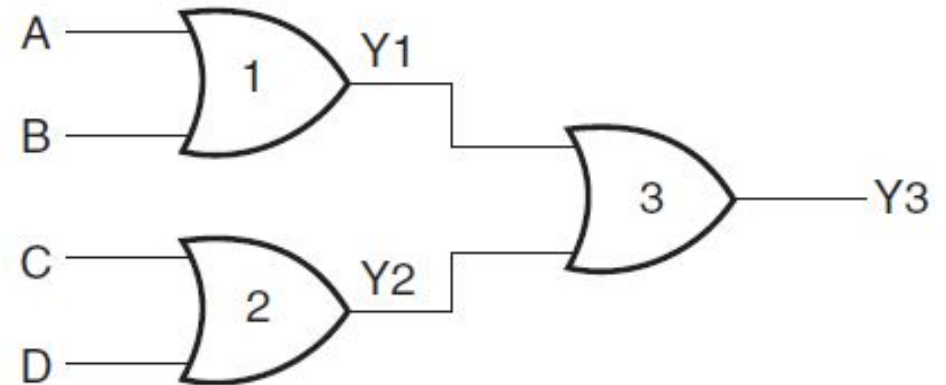
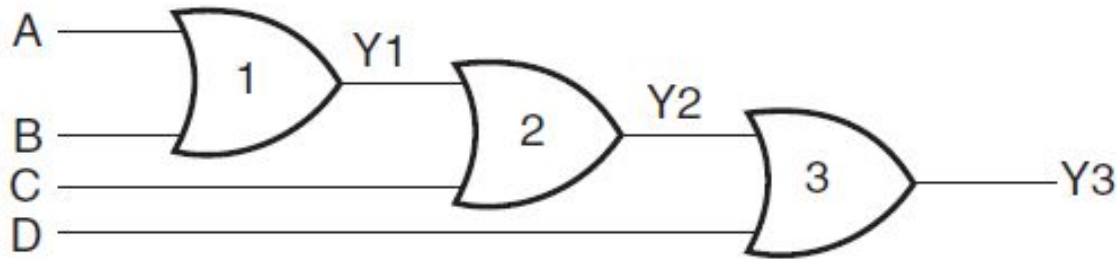
1) OR Gate - 7432



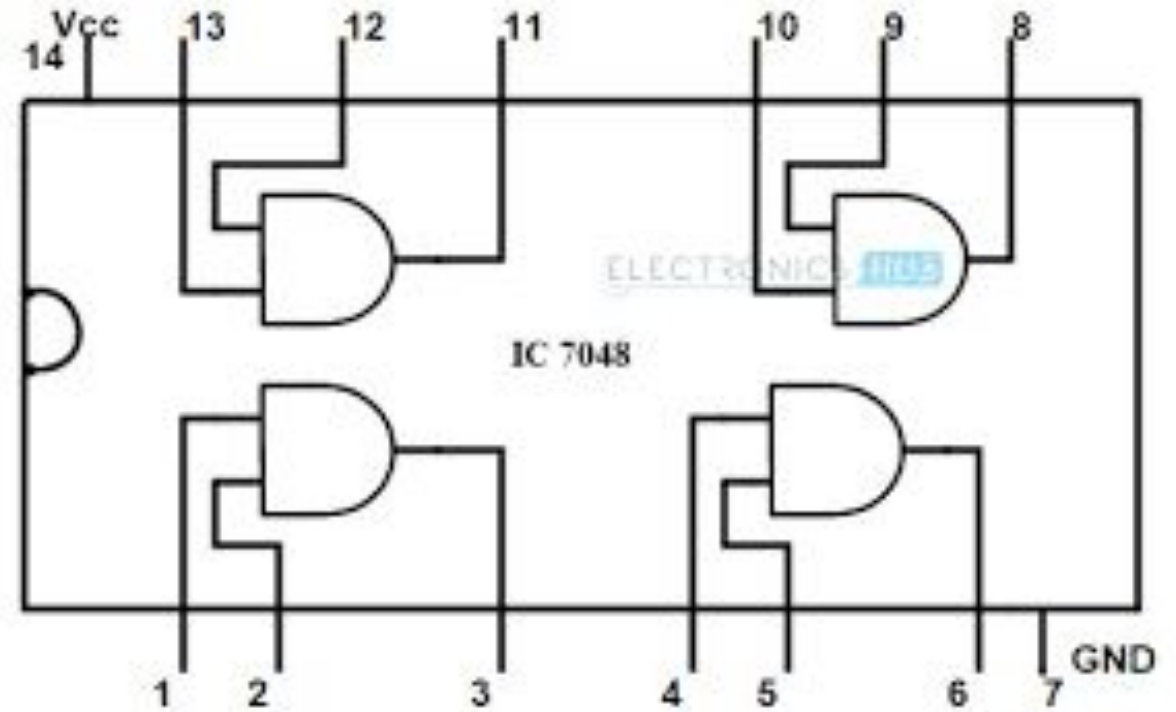
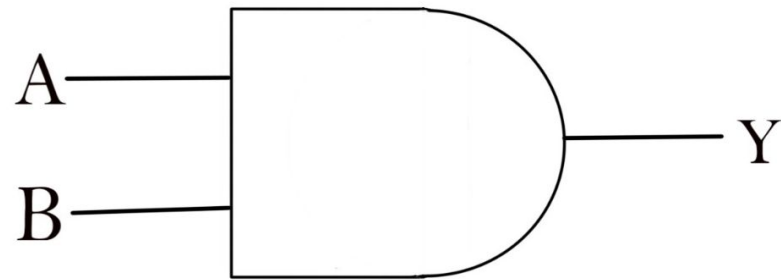
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

1) OR Gate - 7432

- For multiple input i.e. for 4 inputs - $2^n = 2^4 = 16$ output



2) AND Gate - 7408



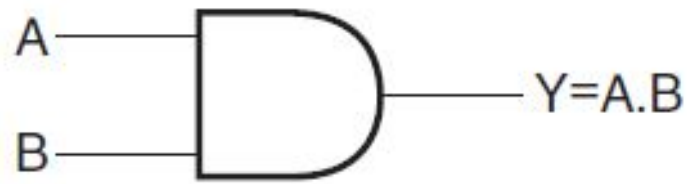
2) AND Gate - 7408

- The output of an AND gate is HIGH only when all of its inputs are in the HIGH state. In all other cases, the output is LOW.
- When interpreted for a positive logic system, this means that the output of the AND gate is a logic '1' only when all of its inputs are in logic '1' state. In all other cases, the output is logic '0'.
- The AND operation on two independent logic variables A and B is written as

$$Y = A.B$$

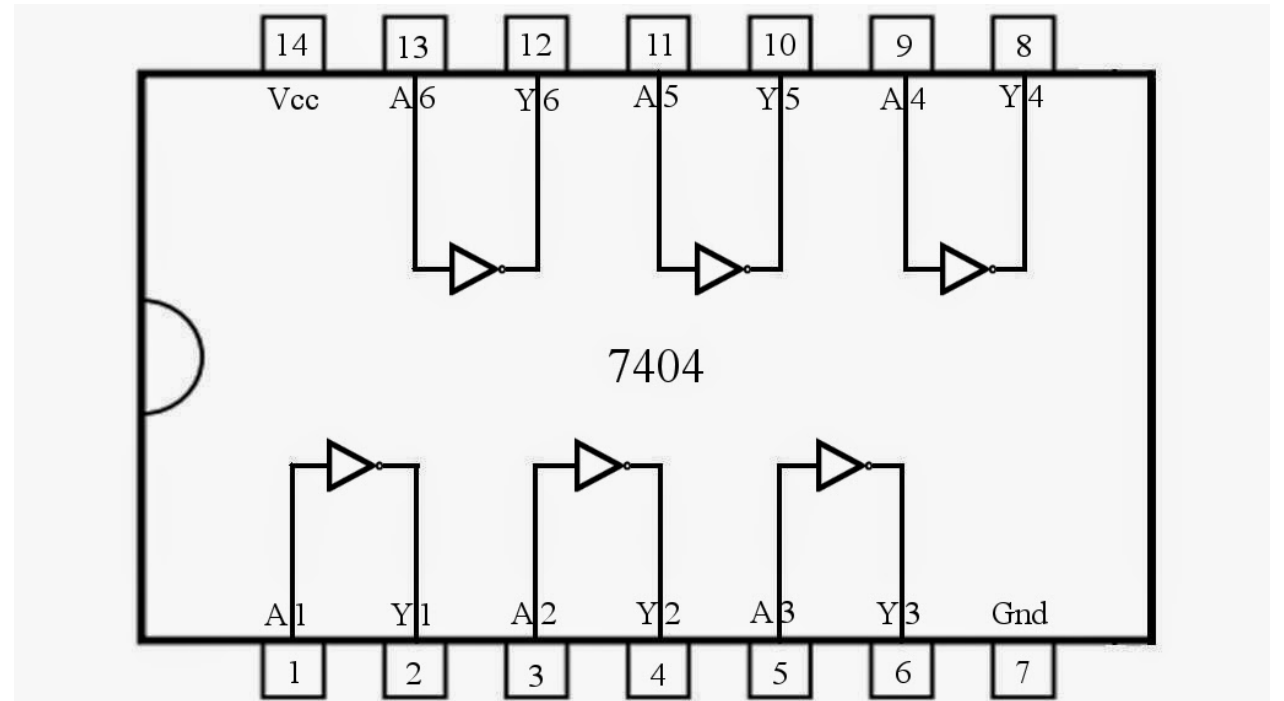
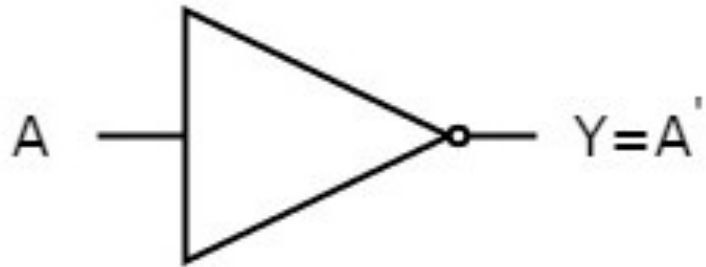
and reads as Y equals A AND B and not as A multiplied by B.

2) AND Gate - 7408



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

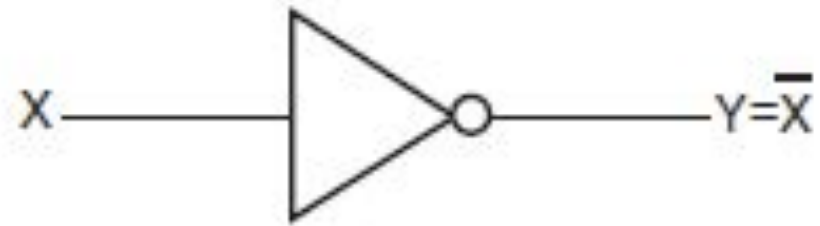
3) NOT Gate - 7404



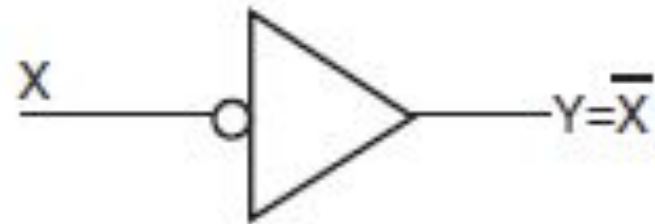
3) NOT Gate - 7404

- A NOT gate is a one-input, one-output logic circuit whose output is always the complement of the input. That is, a LOW input produces a HIGH output, and vice versa.
- When interpreted for a positive logic system, a logic '0' at the input produces a logic '1' at the output, and vice versa. It is also known as a **'complementing circuit'** or an **'inverting circuit'**.
- The NOT operation on a logic variable X is denoted as \bar{X} or X' . That is, if X is the input to a NOT circuit, then its output Y is given by $Y = \bar{X}$ or $Y = X'$ and reads as Y equals NOT X . Thus, if $X = 0$, $Y = 1$ and if $X = 1$, $Y = 0$.

3) NOT Gate - 7404



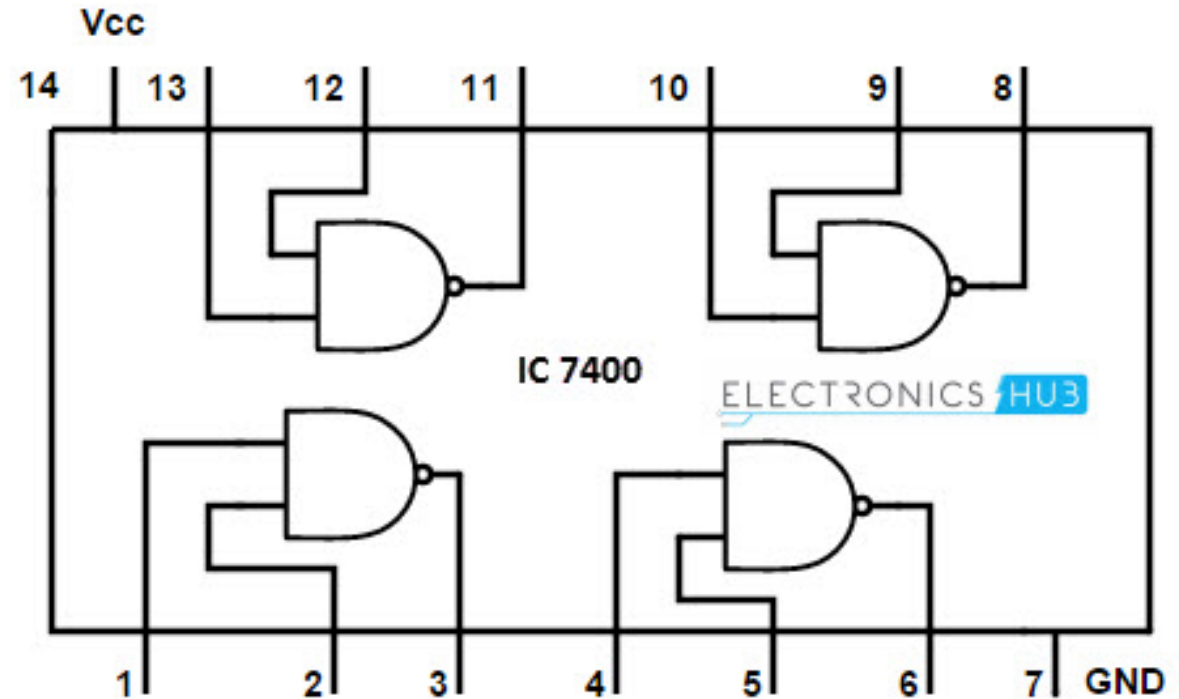
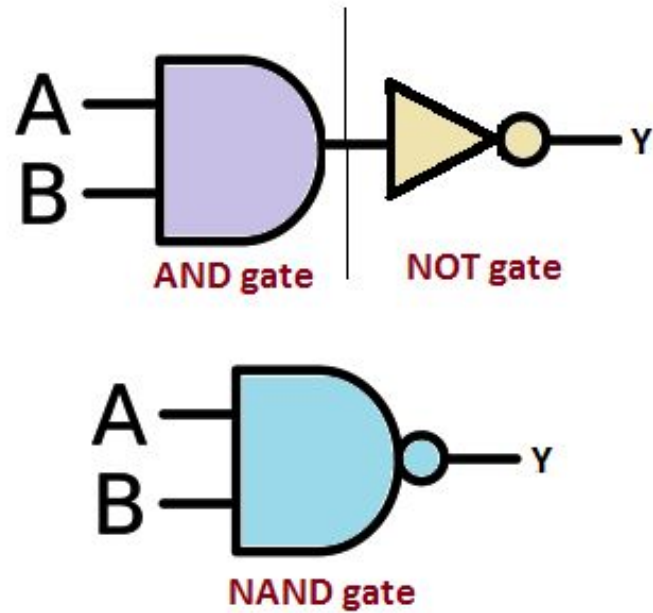
(a)



(b)

X	Y
0	1
1	0

4) NAND Gate - 7400



4) NAND Gate - 7400

- NAND stands for NOT AND. An AND gate followed by a NOT circuit makes it a NAND gate.
- NAND gate is obtained from the truth table of an AND gate by complementing the output entries.
- The output of a NAND gate is a logic '0' when all its inputs are a logic '1'. For all other input combinations, the output is a logic '1'. NAND gate operation is logically expressed as

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

4) NAND Gate - 7400

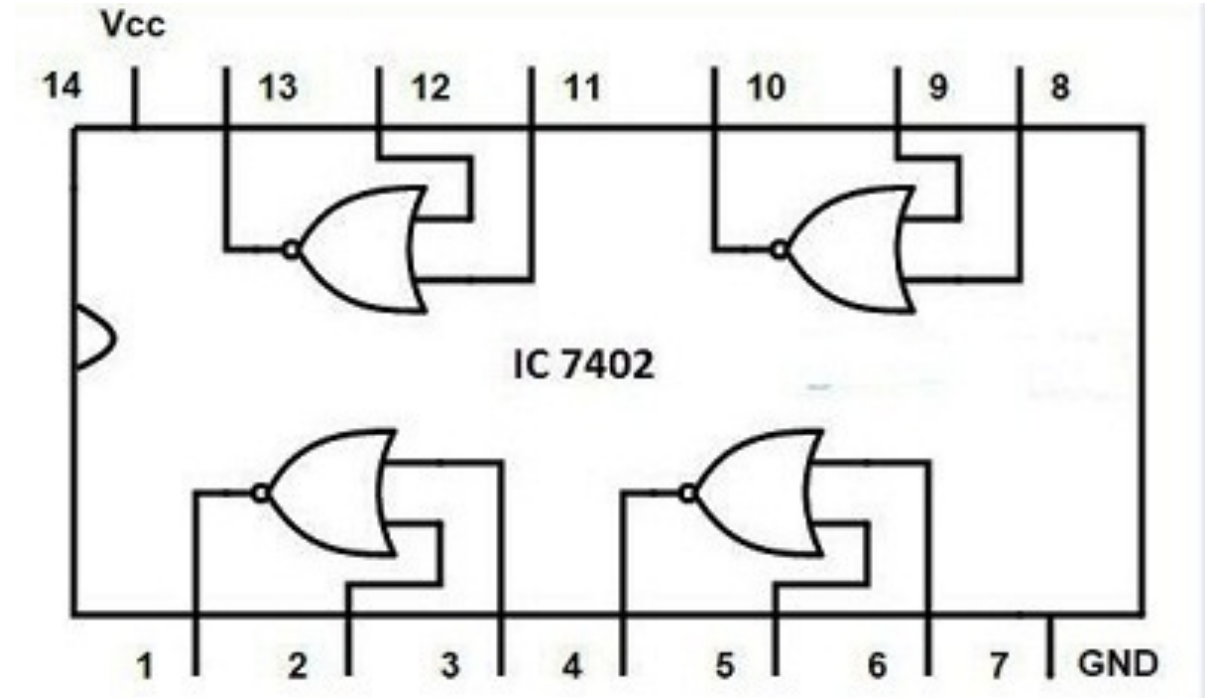
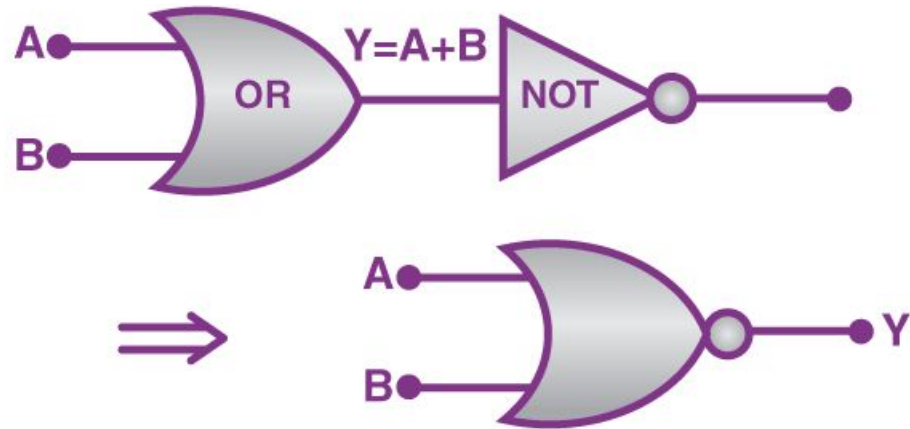


(b)

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

(c)

5) NOR Gate - 7402

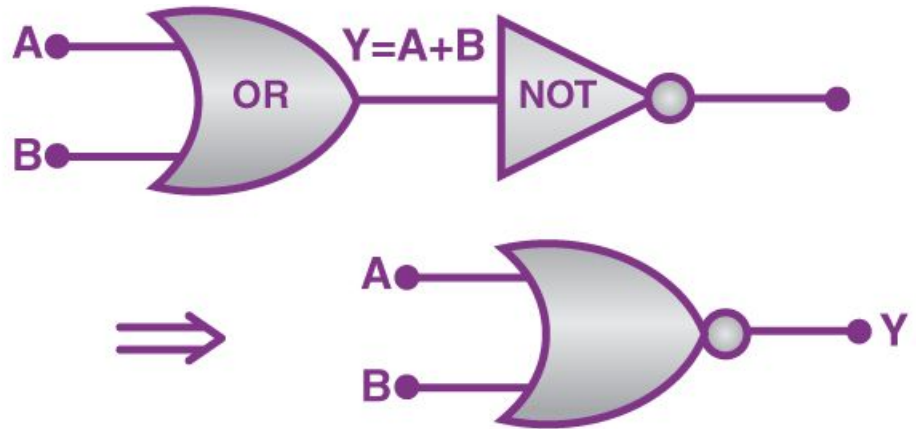


5) NOR Gate - 7402

- NOR stands for NOT OR. An OR gate followed by a NOT circuit makes it a NOR gate.
- The truth table of a NOR gate is obtained from the truth table of an OR gate by complementing the output entries. The output of a NOR gate is a logic '1' when all its inputs are logic '0'. For all other input combinations, the output is a logic '0'. The output of a two-input NOR gate is logically expressed as

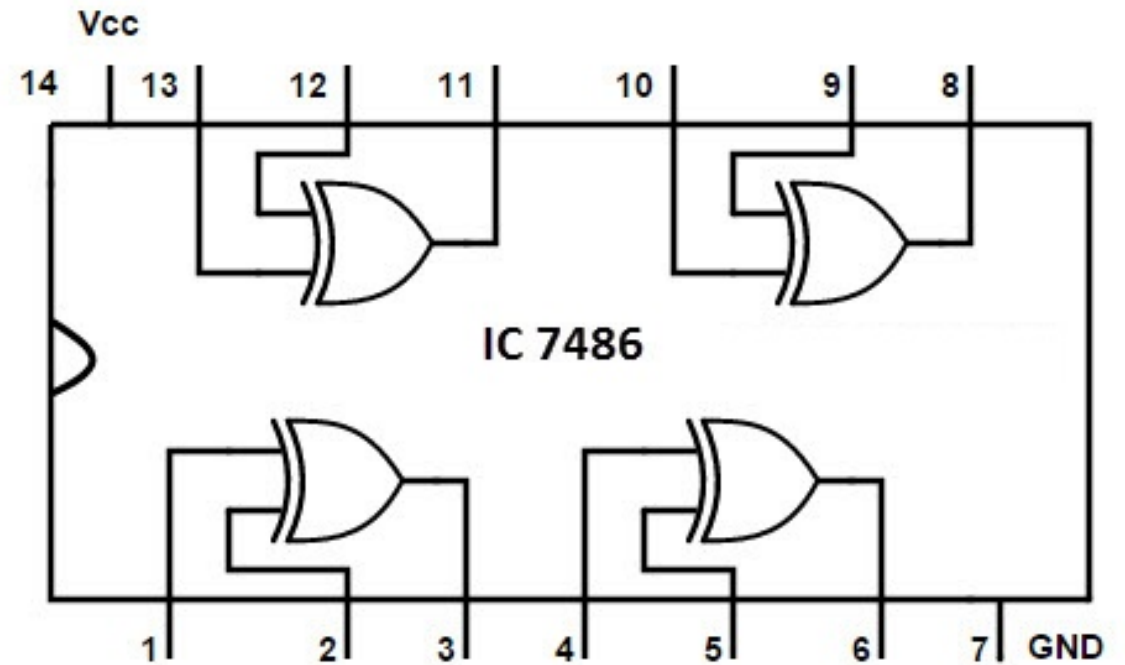
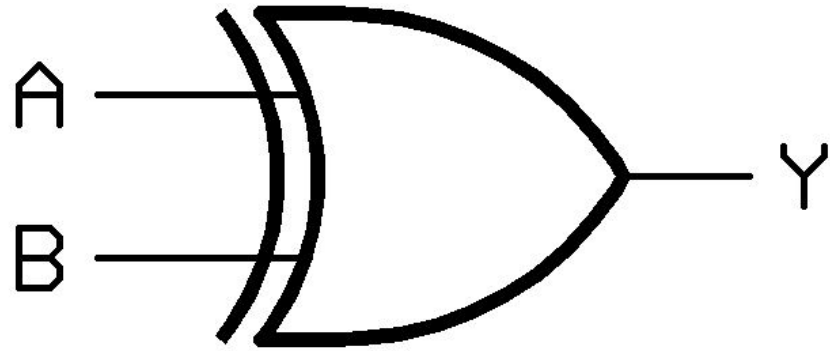
$$Y = \overline{A + B}$$

5) NOR Gate - 7402



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

6) EX-OR Gate - 7486

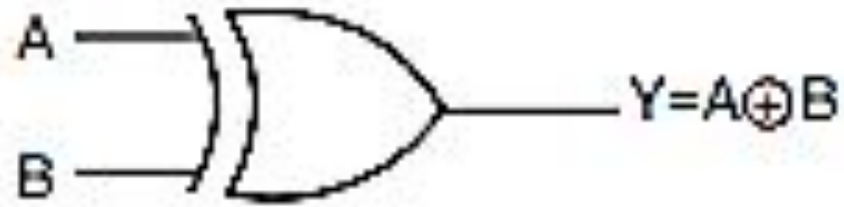


6) EX-OR Gate - 7486

- The EXCLUSIVE-OR gate, commonly written as EX-OR gate, is a two-input, one-output gate. The output of an EX-OR gate is a logic '1' when the inputs are unlike and a logic '0' when the inputs are like.
- Suppose, if both input A & B are assigned with values '1' & '1' respectively then the output will be '0'. Same output is obtained when the value entered is '0' for both the inputs.
- We get the output as '1' when both the inputs are opposite i.e. when $A = 0, B = 1$ – or – $A = 1, B = 0$. The logical,

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

6) EX-OR Gate - 7486

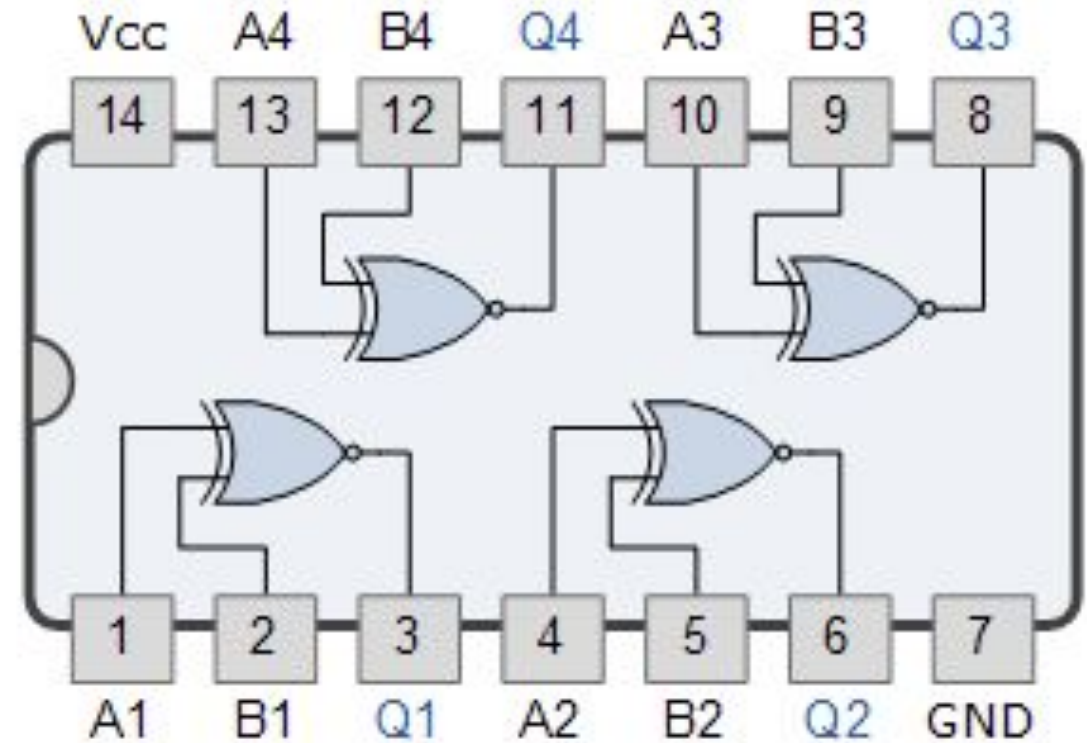
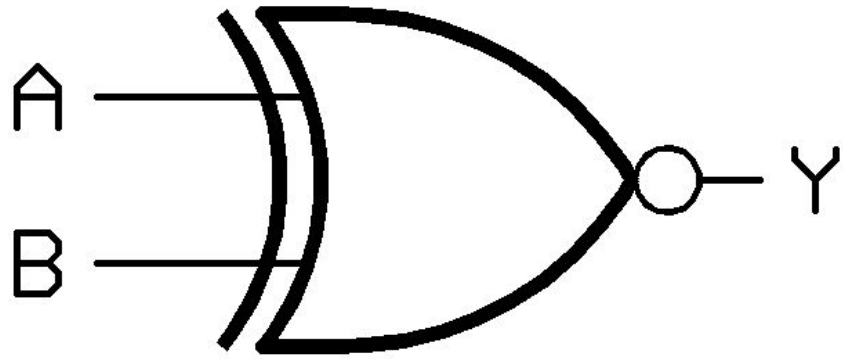


(a)

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

(b)

7) EX-NOR Gate - 74266



7) EX-NOR Gate - 74266

- EXCLUSIVE-NOR (commonly written as EX-NOR) means NOT of EX-OR, i.e. the logic gate that we get by complementing the output of an EX-OR gate. The truth table of an EX-NOR gate is obtained from the truth table of an EX-OR gate by complementing the output entries. Logically,

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

Universal Gates

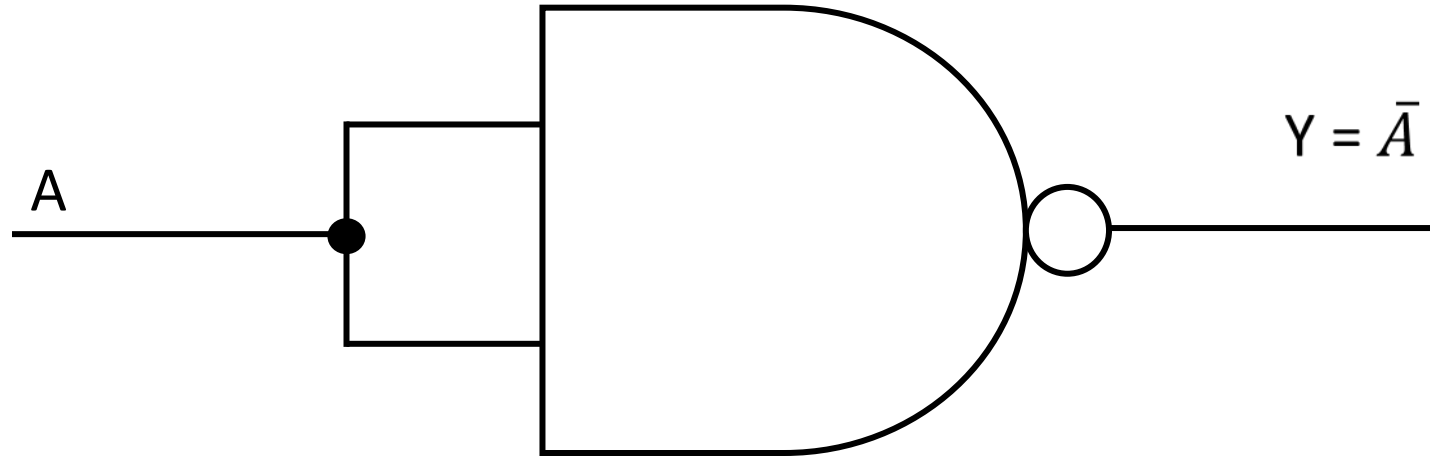
- NOR and NAND gates have the property that they individually can be used to hardware-implement a logic circuit corresponding to any given Boolean expression. That is, it is possible to use either only NAND gates or only NOR gates to implement any Boolean expression.
- This is so because a combination of NAND gates or a combination of NOR gates can be used to perform functions of any of the basic logic gates. It is for this reason that **NAND and NOR gates are universal gates.**

Other Logic gates using Universal Gates

- Various Other Basic Logic Gates can be obtained using **NAND Gate & NOR Gate**.
- Two-input NAND gates can be used to construct a NOT circuit by merging that two input together.
- So that the input provided to the merged input of **NAND Gate** is both the same i.e. either '0' or '1'.
- As we know, the truth table for **NAND Gate** concludes - The output of a NAND gate is a logic '0' when all its inputs are a logic '1'. For all other input combinations, the output is a logic '1'.

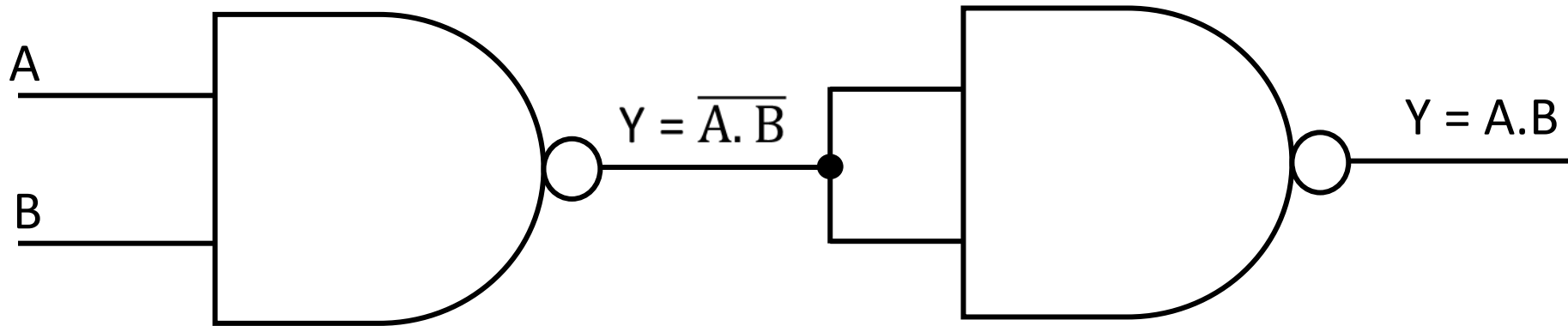
Other Logic gates using Universal Gates

- NOT Gate using NAND Gate



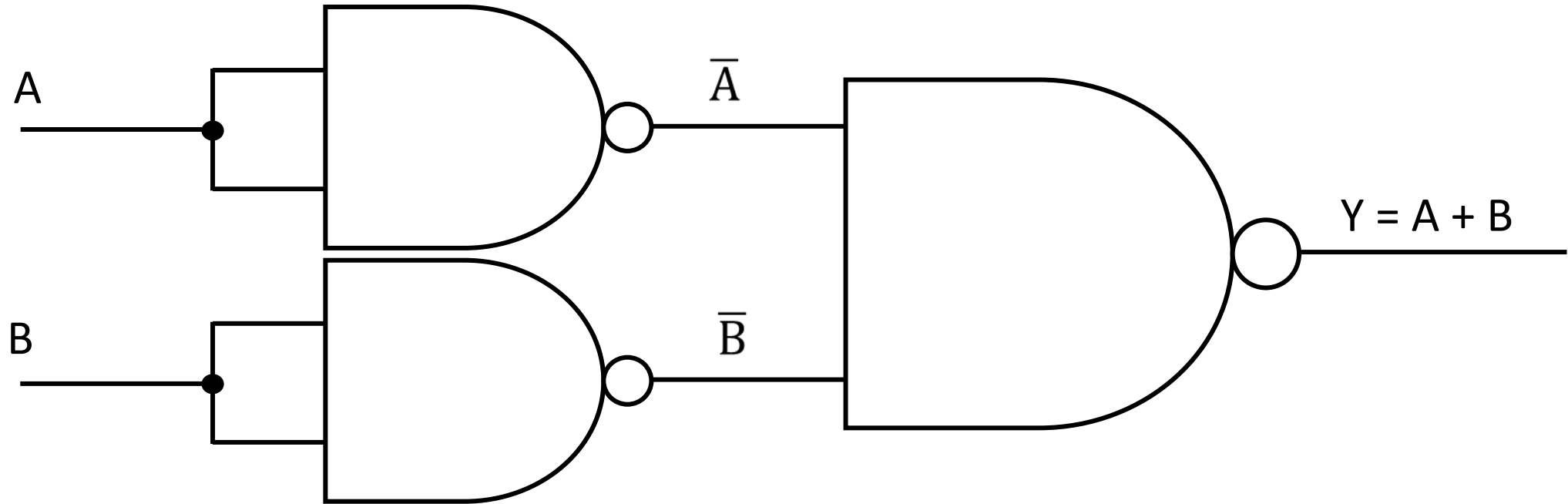
Other Logic gates using Universal Gates

- **AND Gate using NAND Gate**



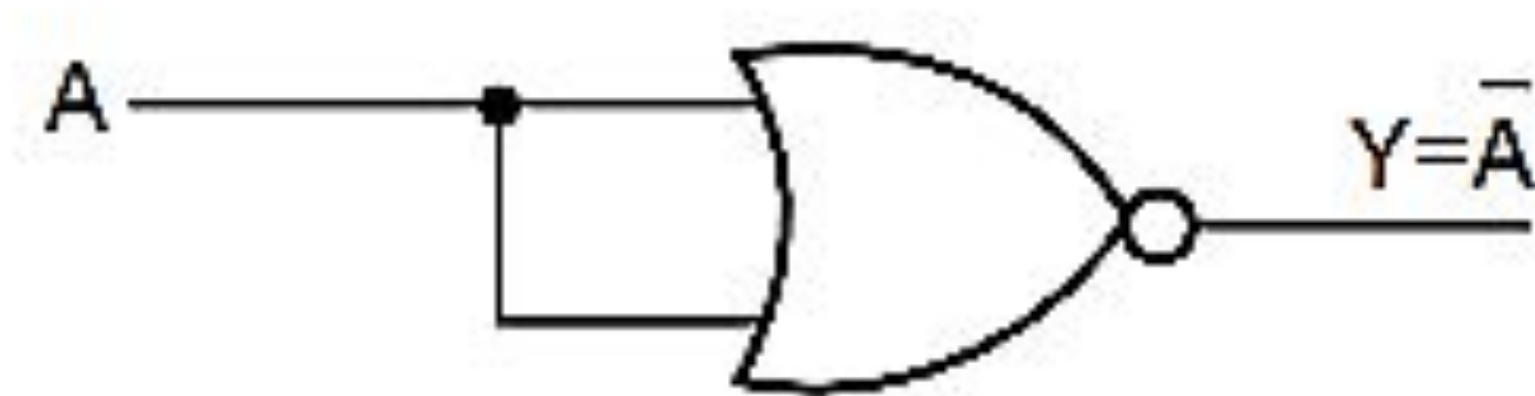
Other Logic gates using Universal Gates

- OR Gate using NAND Gate



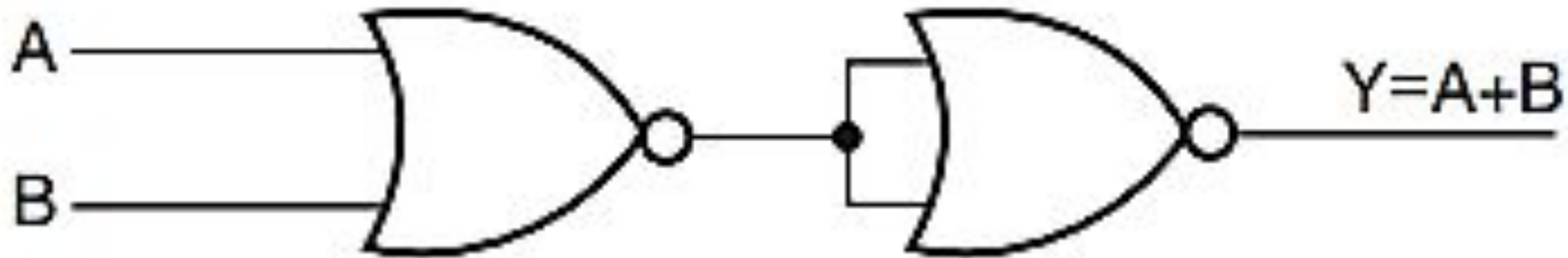
Other Logic gates using Universal Gates

- NOT Gate using NOR Gate



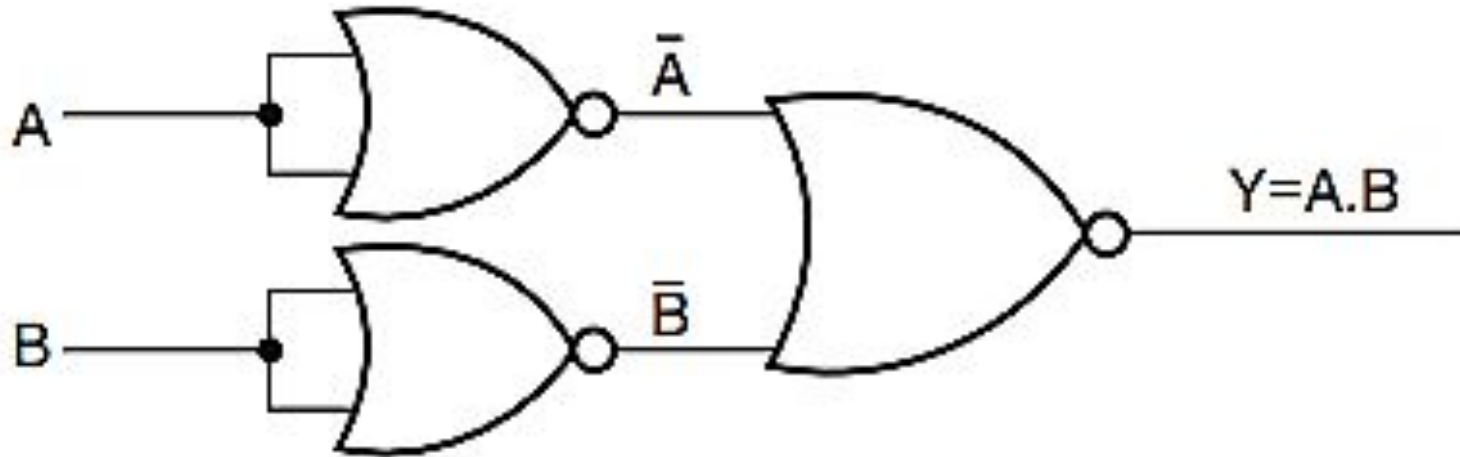
Other Logic gates using Universal Gates

- OR Gate using NOR Gate



Other Logic gates using Universal Gates

- **AND Gate using NOR Gate**



Boolean Algebra

- In the middle of 19th century, an English mathematician George Boole developed rules for manipulations of binary variables, known as **Boolean Algebra**.

- **Boolean Algebraic Theorems**

- i. $A + 0 = A$
- ii. $A + 1 = 1$
- iii. $A \cdot 1 = A$
- iv. $A \cdot 0 = 0$

Boolean Algebra

• Boolean Algebraic Theorems

v. $A + A = A$

vi. $A \cdot A = A$

vii. $A + \bar{A} = 1$

viii. $A \cdot \bar{A} = 0$

ix. $A \cdot (B + C) = AB + AC$

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

Boolean Algebra

• Boolean Algebraic Theorems

xi. $A + AB = A$

xii. $A(A + B) = A$

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

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

xv. $AB + A\overline{B} = A$

xvi. $(A + B) \cdot (A + \overline{B}) = A$

Boolean Algebra

• Boolean Algebraic Theorems

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

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

xix. $AB + \bar{A}C + BC = AB + \bar{A}C$

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

Boolean Algebra

• De Morgan's Theorems

$$1. \overline{A \cdot B} = \bar{A} + \bar{B}$$

$$2. \overline{A + B} = \bar{A} \cdot \bar{B}$$

Boolean Algebra

• De Morgan's Theorems – Truth Table

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