21CSS201T
COMPUTER ORGANIZATION AND ARCHITECTURE

UNIT-1

Computer Architecture Objectives

- Know the difference between computer organization and computer architecture.
- Understand units of measure common to computer systems
- Appreciate the evolution of computers.
- Understand the computer as a layered system.
- Be able to explain the von Neumann architecture and the function of basic computer components.

COMPUTER ARCHITECTURE AND ORGANIZATION OVERVIEW

- A modern computer is an electronic, digital, general purpose computing machine that automatically follows a step-by-step list of instructions to solve a problem. This step-by step list of instructions that a computer follows is also called an algorithm or a computer program.
- Why to study computer organization and architecture?
  - Design better programs, including system software such as compilers, operating systems, and device drivers.
  - Optimize program behavior.
  - Evaluate (benchmark) computer system performance.
  - Understand time, space, and price tradeoffs.
- Computer organization
  - Encompasses all physical aspects of computer systems.
  - E.g., circuit design, control signals, memory types.
  - *How does a computer work?*

COMPUTER ARCHITECTURE AND ORGANIZATION OVERVIEW

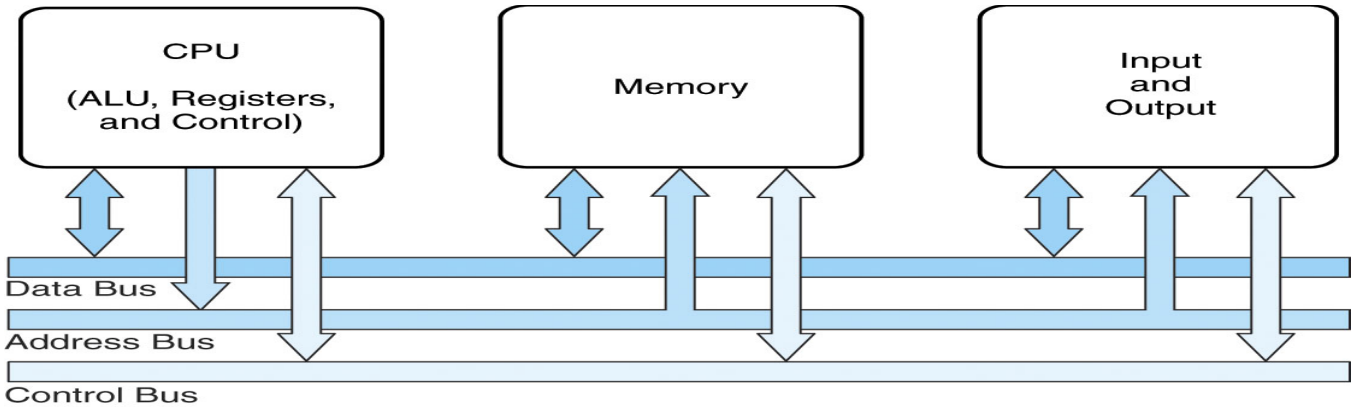
- Focuses on the **structure(**the way in which the components are interrelated**)** and behavior of the computer system and refers to the logical aspects of system implementation as seen by the programmer
- Computer architecture includes many elements such as

  instruction sets and formats, operation codes, data types, the number and types of registers, addressing modes, main memory access methods, and various I/O mechanisms.
- The architecture of a system directly affects the logical execution of programs.
- The computer architecture for a given machine is the combination of its hardware components plus its instruction set architecture (ISA).
- The ISA is the interface between all the software that runs on the machine and the hard
- **Studying computer architecture helps us to answer the question: How do I design a computer?**

COMPUTER ARCHITECTURE AND ORGANIZATION OVERVIEW

- In the case of the IBM, SUN and Intel ISAs, it is possible to purchase processors which execute the same instructions from more than one manufacturer
- All these processors may have quite different internal organizations but they all appear identical to a programmer, because their **instruction sets** are the same
- Organization & Architecture enables a family of computer models
  - Same Architecture, but with differences in Organization
  - Different price and performance characteristics
- When technology changes, only organization changes.
- This gives code compatibility (backwards)

COMPUTER COMPONENTS



At the most basic level, a computer is a device consisting of 3 pieces

A processor to interpret and execute programs

A memory (Includes Cache, RAM, ROM) to **store both data and program instructions**

A mechanism for transferring data to and from the outside world.

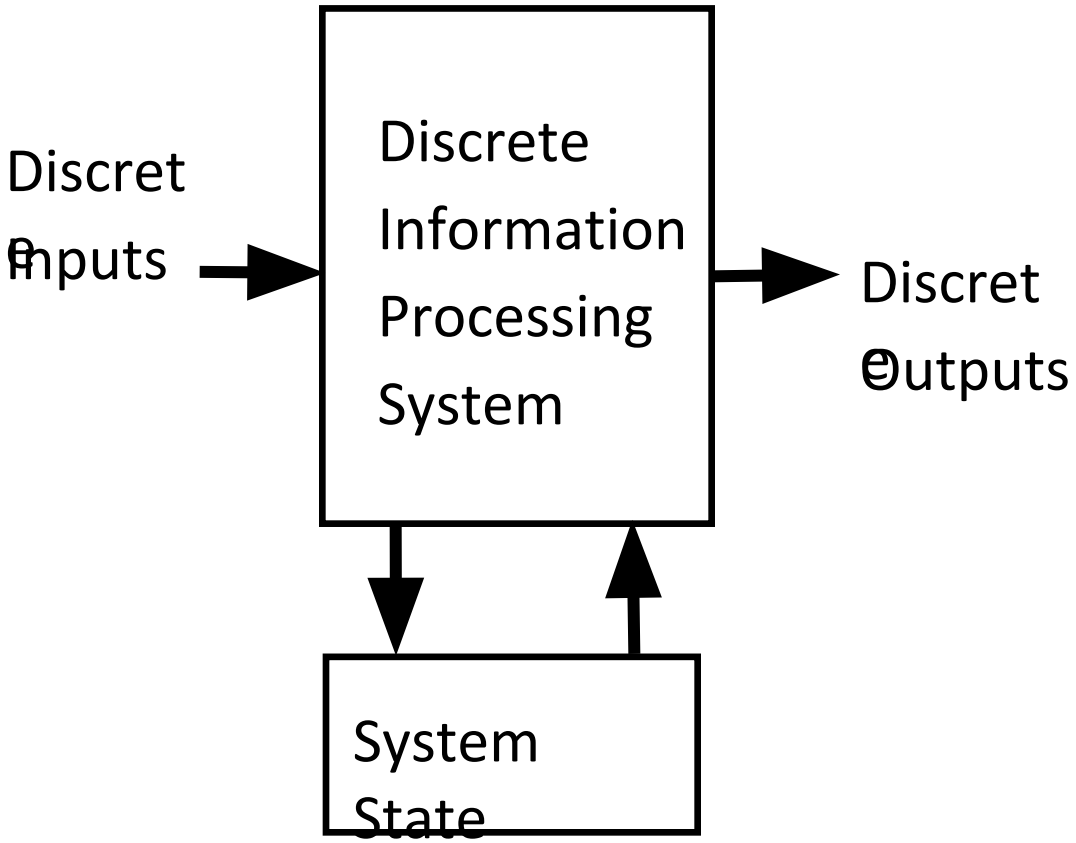
- I/O to communicate between computer and the world
- Bus to move info from one computer component to another

Contd..

- Computers with large main memory capacity can run larger programs with greater speed than computers having small memories.
- RAM is an acronym for random access memory. Random access means that memory contents can be accessed directly if you know its location.
- Cache is a type of temporary memory that can be accessed faster than RAM.

Digital System

- **Takes a set of discrete information inputs and discrete internal information (system state) and generates a set of discrete information outputs.**



Signal

- An information variable represented by physical quantity.
- For digital systems, the variable takes on discrete values.
- Two level, or binary values are the most prevalent values in digital systems.
- Binary values are represented abstractly by:
  - digits 0 and 1
  - words (symbols) False (F) and True (T)
  - words (symbols) Low (L) and High (H)
  - and words On and Off.
- Binary values are represented by values or ranges of values of physical quantities

Introduction to Number System and Logic Gates

- Number Systems- Binary, Decimal, Octal, Hexadecimal
- Codes- Grey, BCD,Excess-3,
- ASCII, Parity
- Binary Arithmetic- Addition, Subtraction, Multiplication, Division using Sign Magnitude
- 1’s complement, 2’s complement,
- BCD Arithmetic;
- Logic Gates-AND, OR, NOT, NAND,
- NOR, EX-OR, EX-NOR

Number System

- Integers are normally written using positional number system, in which each digit represents the coefficient in a power series
- $N = a_{n-1}r^{n-1} + a_{n-2}r^{n-2} + \cdots + a_2r^2 + a_1r^1 + a_0$

Where n *is the number of digit,* r *is the radix or base and* a_i *is the coefficient*

$$0 \leq a_i < r$$

Ex.

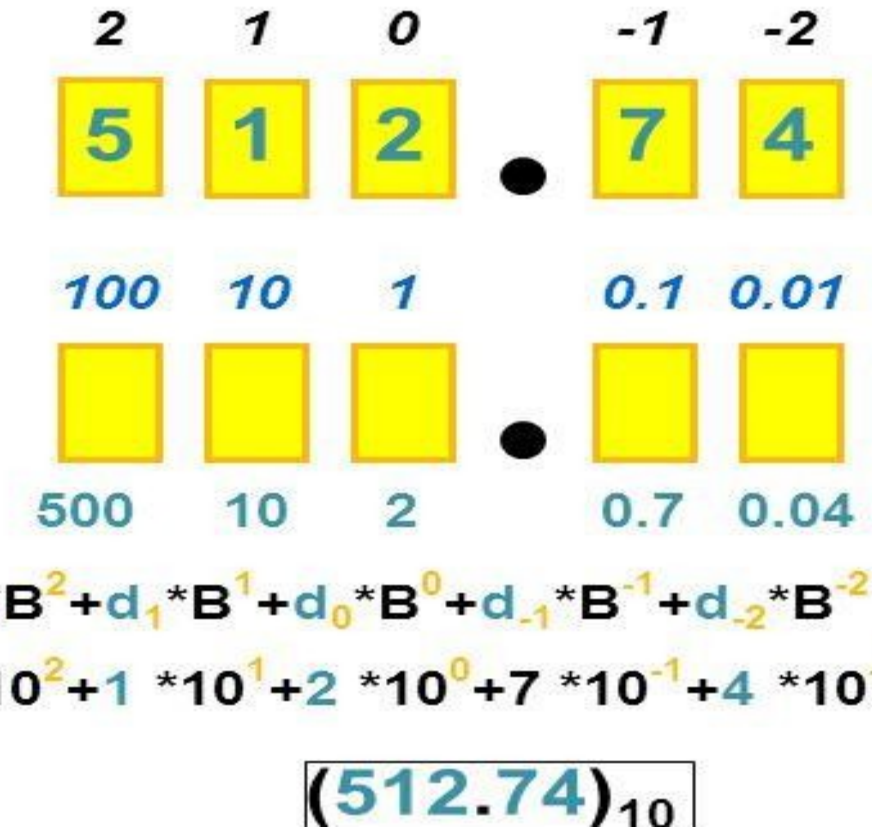
$$97142 = 9*10^4 + 7*10^3 + 1*10^2 + 4*10^1 + 2*10^0$$

There are four systems of arithmetic which are often used in digital circuits. These systems are:

- ***decimal****: it has a base* $(r=10)$ *and coefficients* (a) *are in the range 0 to 9*
- ***binary:*** *it has a base* $(r=2)$ *and coefficients* (a) *are all either 0 or 1*
- ***octal :*** *it has a base* $(r=8)$ *and coefficients* (a) *are in the range 0 to 7*
- ***Hexadecimal:*** *it has a base* $(r=16)$ *and coefficients* (a) *are in the range* { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F }

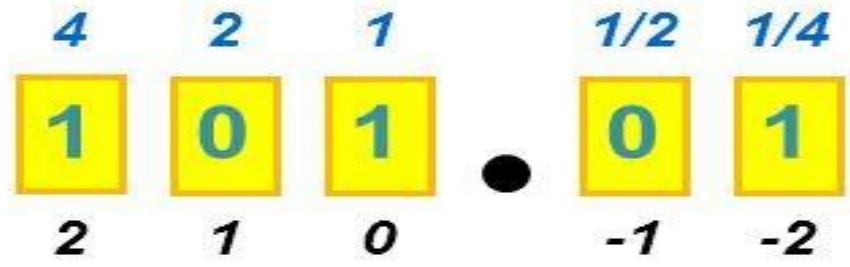
Decimal Number System

- Base (also called radix) = 10
  - 10 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
- Digit Position
  - Integer & fraction
- Digit Weight
  - Weight = (*Base*) ^{Position}
- Magnitude
  - Sum of "*Digit* x *Weight*"
- Formal Notation



Binary Number System

- Base = 2
  - 2 digits { 0, 1 }, called *b*inary dig*its* or "*bits*"
- Weights
  - Weight = (*Base*) ^{Position}
- Magnitude
  - Sum of "*Bit* x *Weight*"
- Formal Notation
- Groups of bits 4 bits = *Nibble*
  8 bits = *Byte*



$1*2^{2}+0*2^{1}+1*2^{0}+0*2^{-1}+1*2^{-2}$

$=(5.25)_{10}$

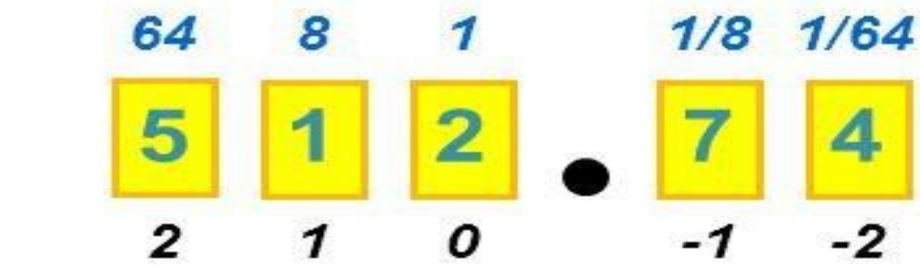
$(101.01)_{2}$

1 0 1 1

1 1 0 0 0 1 0 1

Octal Number System

- Base = 8
  - 8 digits { 0, 1, 2, 3, 4, 5, 6, 7 }
- Weights
  - Weight = (*Base*) ^{Position}
- Magnitude
  - Sum of "*Digit* x *Weight*"
- Formal Notation



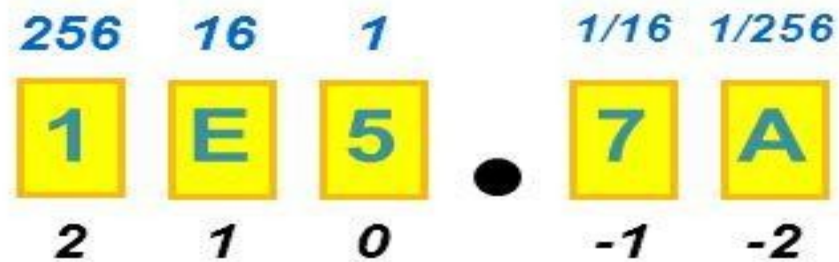
$5*8^{2}+1*8^{1}+2*8^{0}+7*8^{-1}+4*8^{-2}$

$=(330.9375)_{10}$

$(512.74)_{8}$

Hexadecimal Number System

- Base = 16
  - 16 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F }
- Weights
  - Weight = (*Base*) ^{Position}
- Magnitude
  - Sum of "*Digit* x *Weight*"
- Formal Notation

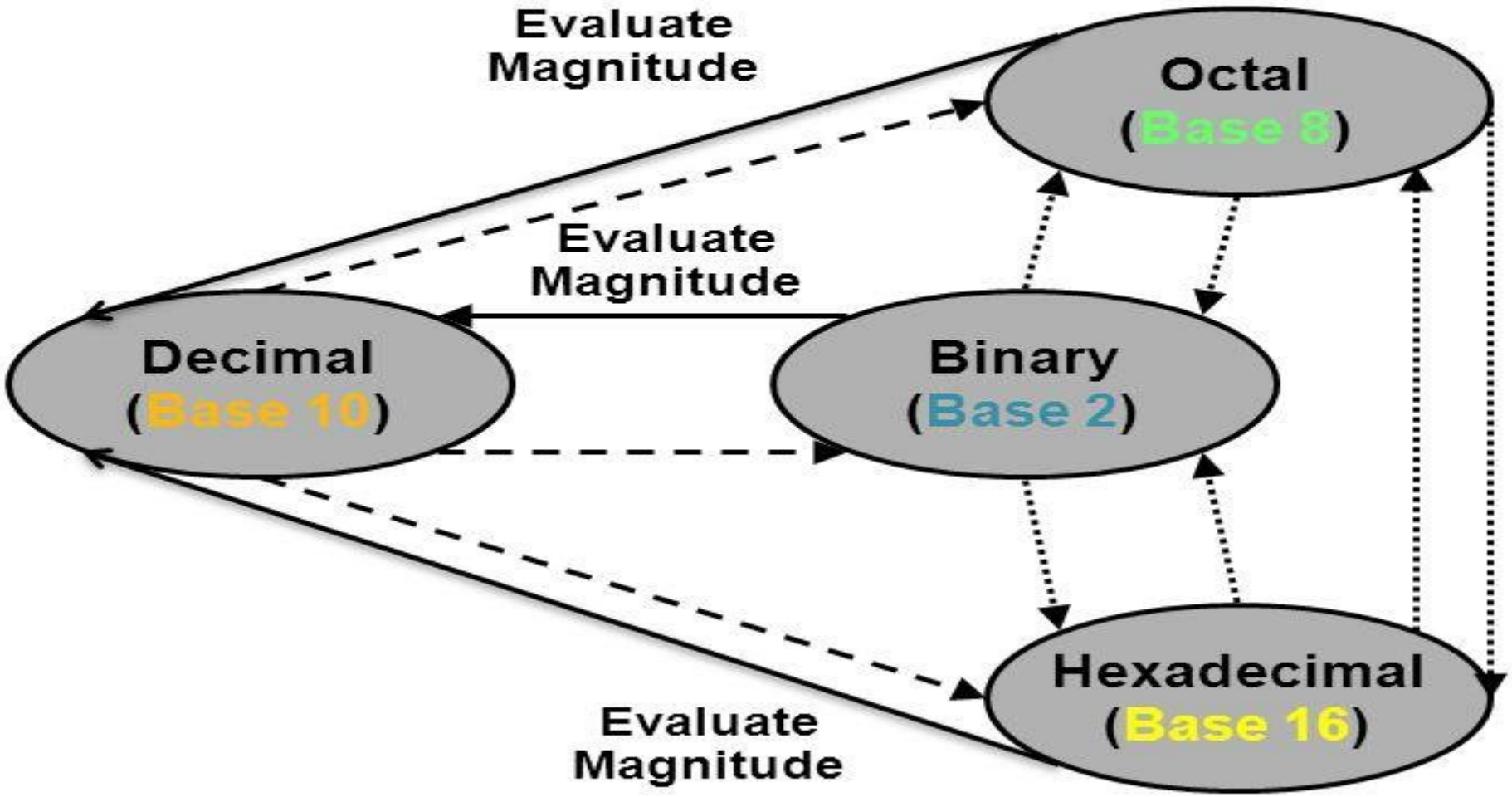


$$1*16^{2}+14*16^{1}+5*16^{0}+7*16^{-1}+10*16^{-2}$$

$$=(485.4765625)_{10}$$

$$(1E5.7A)_{16}$$

Number Base Conversions



Decimal (*Integer*) to Binary Conversion

- Divide the number by the 'Base' (=2)
- Take the remainder (either 0 or 1) as a coefficient
- Take the quotient and repeat the division

Example: $(13)_{10}$

	Quotient	Remainder	Coefficient
13 / 2 =	6	1	$a_0 = 1$
6 / 2 =	3	0	$a_1 = 0$
3 / 2 =	1	1	$a_2 = 1$
1 / 2 =	0	1	$a_3 = 1$

Answer: $(13)_{10} = (a_3\, a_2\, a_1\, a_0)_2 = (1101)_2$

MSB LSB

Decimal (*Integer*) to Binary Conversion

$25 \div 2 = 12 + \text{remainder of } 1$ — TOP
$12 \div 2 = 6 + \text{remainder of } 0$
$6 \div 2 = 3 + \text{remainder of } 0$
$3 \div 2 = 1 + \text{remainder of } 1$
$1 \div 2 = 0 + \text{remainder of } 1$
$\therefore \quad 25_{10} = 11001_2$ — BOTTOM

The above process may be simplified as under :

Successive Divisions		***Remainders***
2) 2 5		
2) 1 2	———	1
2) 6	———	0
2) 3	———	0
2) 1	———	1
2) 0	———	1

Reading the remainders from bottom to top, we get $25_{10} = 11001_2$

It may also be put in the following form :

$25 \div 2 = 12 + 1$
$12 \div 2 = 6 + 0$
$6 \div 2 = 3 + 0$
$3 \div 2 = 1 + 1$
$1 \div 2 = 0 + 1$
$\therefore$ decimal 25 = 1 1 0 0 1 binary

Decimal (*Fraction*) to Binary Conversion

- Multiply the number by the 'Base' (=2)
- Take the integer (either 0 or 1) as a coefficient
- Take the resultant fraction and repeat the division

Example: $(0.625)_{10}$

		Integer		Fraction	Coefficient
0.625	* 2 =	1	.	25	$a_{-1} = 1$
0.25	* 2 =	0	.	5	$a_{-2} = 0$
0.5	* 2 =	1	.	0	$a_{-3} = 1$

Answer: $(0.625)_{10} = (0.a_{-1}\, a_{-2}\, a_{-3})_2 = (0.101)_2$

MSB LSB

Decimal to Binary Conversion

(a) Integer

$25 \div 2 = 12 + 1$

$12 \div 2 = 6 + 0$

$6 \div 2 = 3 + 0$

$3 \div 2 = 1 + 1$

$1 \div 2 = 0 + 1$

$\therefore \quad 25_{10} = 11001_2$

(b) fraction

$0.625 \times 2 = 1.25 = 0.25 + 1$

$0.25 \times 2 = 0.5 = 0.5 + 0$

$0.5 \times 2 = 1.0 = 0.0 + 1$

$\therefore \quad 0.625_{10} = 0.101_2$

Table of binary equivalent decimal numbers

Decimal	Binary	Decimal	Binary	Decimal	Binary
1	1	11	1011	21	10101
2	10	12	1100	22	10110
3	11	13	1101	23	10111
4	100	14	1110	24	11000
5	101	15	1111	25	11001
6	110	16	10000	26	11010
7	111	17	10001	27	11011
8	1000	18	10010	28	11100
9	1001	19	10011	29	11101
10	1010	20	10100	30	11110

Decimal to Octal Conversion

Example: $(175)_{10}$

	Quotient	Remainder	Coefficient
175 / 8 =	21	7	$a_0 = 7$
21 / 8 =	2	5	$a_1 = 5$
2 / 8 =	0	2	$a_2 = 2$

Answer: $(175)_{10} = (a_2\, a_1\, a_0)_8 = (257)_8$

Example: $(0.3125)_{10}$

	Integer		Fraction	Coefficient
0.3125 * 8 =	2	.	5	$a_{-1} = 2$
0.5 * 8 =	4	.	0	$a_{-2} = 4$

Answer: $(0.3125)_{10} = (0.a_{-1}\, a_{-2}\, a_{-3})_8 = (0.24)_8$

Decimal to Hexadecimal Conversion

Example: $(1983)_{10}$

	Quotient	Remainder	Coefficient
1983 / 16 =	123	15	$a_0 = F$
123 / 16 =	7	11	$a_1 = B$
7 / 16 =	0	7	$a_2 = 7$

↑

Answer: $(1983)_{10} = (a_2\, a_1\, a_0)_{16} = (7BF)_{16}$

Example: $(0.5625)_{10}$

	Integer	Fraction	Coefficient
0.5625 *16 =	9	0	$a_{-1} = 9$

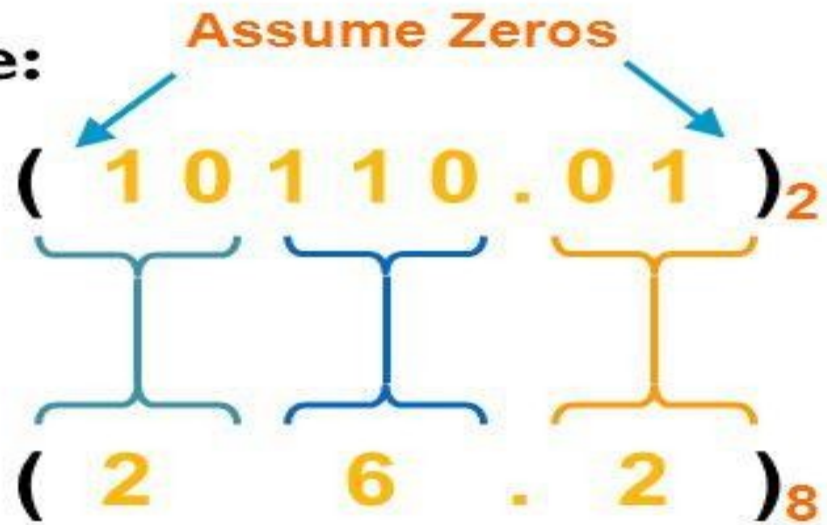
↓

Answer: $(0.5625)_{10} = (0.a_{-1}\, a_{-2}\, a_{-3})_{16} = (0.9)_{16}$

Binary – Octal Conversion

- $8 = 2^3$
- Each group of 3 bits represents an octal digit

Example:



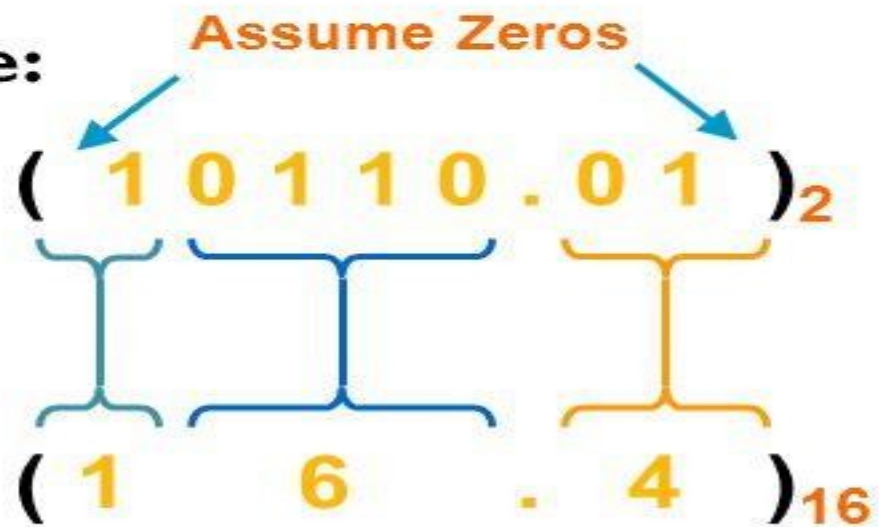
Octal	Binary
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

Works both ways (Binary to Octal & Octal to Binary)

Binary – Hexadecimal Conversion

- $16 = 2^4$
- Each group of 4 bits represents a hexadecimal digit

Example:



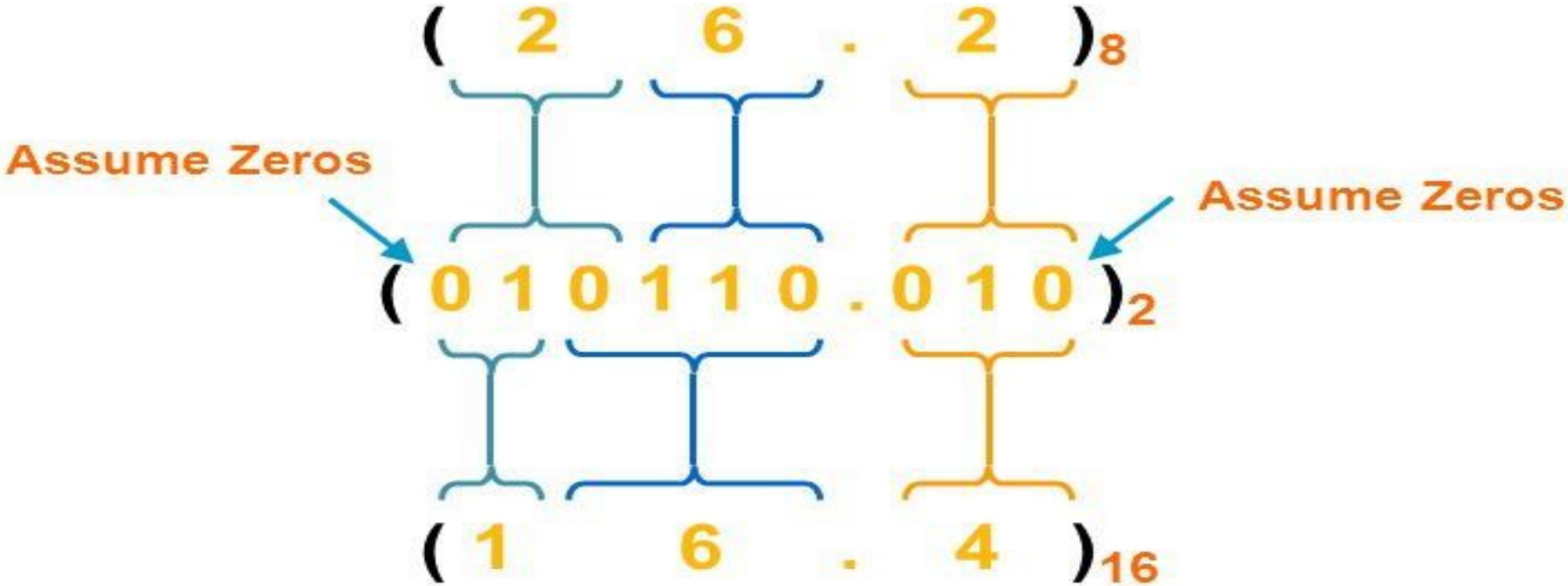
Hex	Binary
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
A	1 0 1 0
B	1 0 1 1
C	1 1 0 0
D	1 1 0 1
E	1 1 1 0
F	1 1 1 1

Works both ways (Binary to Hex & Hex to Binary)

Octal – Hexadecimal Conversion

- Convert to Binary as an intermediate step

Example:



Works both ways (Octal to Hex & Hex to Octal)

Decimal, Binary, Octal and Hexadecimal

Decimal	Binary	Octal	Hex
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Tutorial Problems

- Find the decimal equivalents of the following binary numbers:

(a) $(101)_2$ (b) $(1001)_2$ (c) $(10.011)_2$

[(a) $(5)_{10}$ (b) $(9)_{10}$ (c) $(3.0375)_{10}$]

- What are the decimal equivalents of the following binary numbers ?

(a) $(1111)_2$ (b) $(10100)_2$ (c) $(11011001)_2$ (d) $(10011001)_2$

[(a) $(15)_{10}$ (b) $(20)_{10}$ (c) $(109)_{10}$ (d) $(153)_{10}$]

- Express the following binary numbers into their equivalent decimal numbers :

(a) $(11.01)_2$ (b) $(101.11)_2$ (c) $(110.01)_2$

[(a) $(3.25)_{10}$ (b) $(5.75)_{10}$ (c) $(6.25)_{10}$]

- Convert the following decimal numbers into their binary equivalents:

(a) $(25)_{10}$ (b) $(125)_{10}$ (c) $(0.85)_{10}$

[(a) $(11001)_2$ (b) $(1111101)_2$ (c) $(0.110110)_2$]

- What are the binary equivalents of the following decimal numbers ?

(a) $(27)_{10}$ (b) $(92)_{10}$ (c) $(64)_{10}$

[(a) $(11011)_2$ (b) $(1011100)_2$ (c) $(1000000)_2$]

- Convert the following real numbers to the binary numbers:

(i) $(12.0)_{10}$ (ii) $(25.0)_{10}$ (iii) $(0.125)_{10}$

[(i) $(1100)_2$ (ii) $(11001)_2$ (iii) $(0.001)_2$]

Tutorial Problems

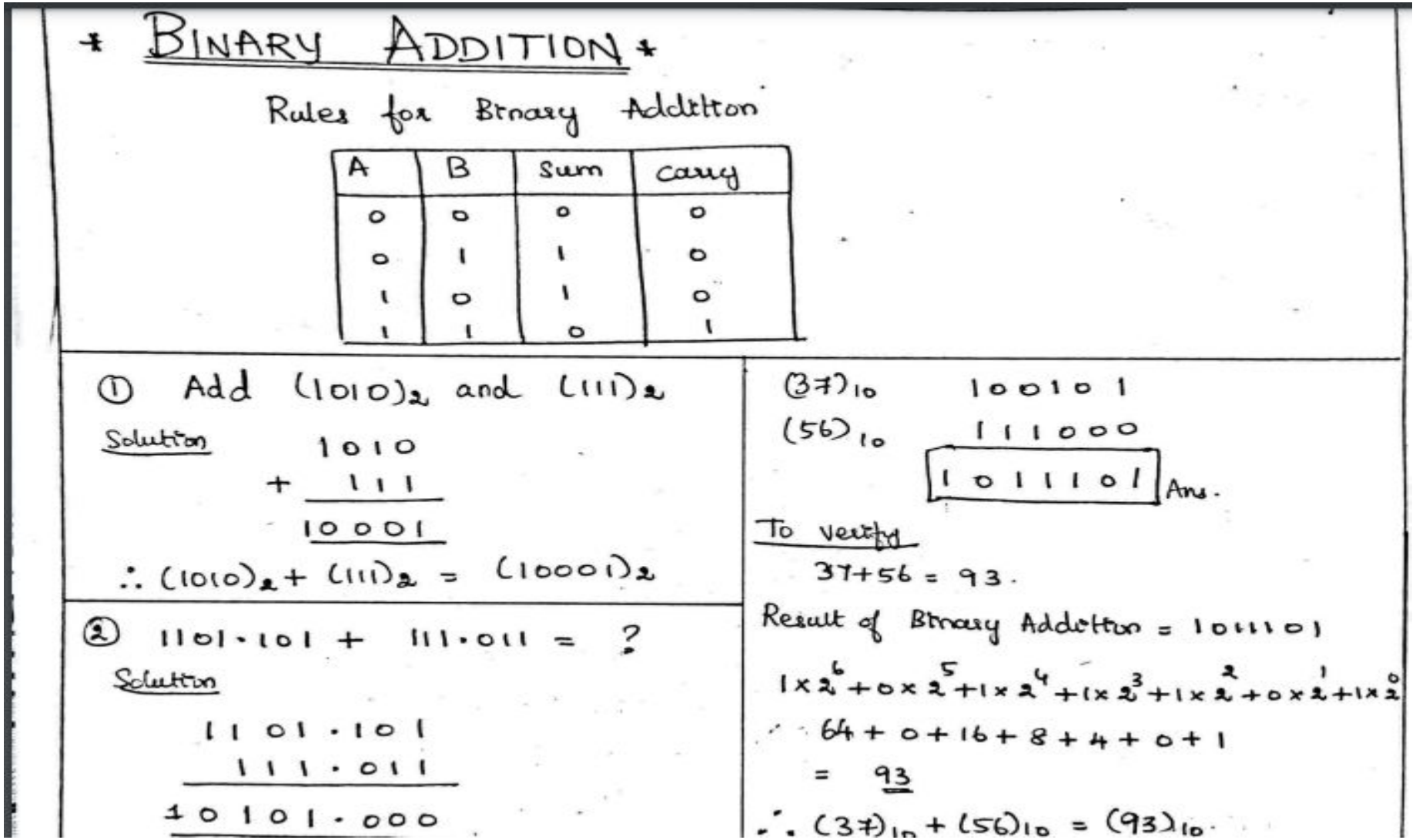
- Convert the following numbers :

(a) $(35)_8$ to decimal (b) $(6421)_8$ to decimal

(c) $(1359)_{10}$ to octal (d) $(7777)_{10}$ to octal

[(a) $(239)_{10}$ (b) $(3345)_{10}$ (c) $(2517)_8$ (d) $(17141)_8$]

BINARY ARITHMETIC



* BINARY ADDITION *

Rules for Binary Addition

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

① Add $(1010)_2$ and $(111)_2$

Solution

$$\begin{array}{r} 1010 \\ +\ \ 111 \\ \hline 10001 \end{array}$$

$\therefore (1010)_2 + (111)_2 = (10001)_2$

② $1101.101 + 111.011 = ?$

Solution

$$\begin{array}{r} 1101.101 \\ 111.011 \\ \hline 10101.000 \end{array}$$

$(37)_{10}$ 100101

$(56)_{10}$ 111000

1011101 Ans.

To verify

$37+56 = 93.$

Result of Binary Addition = 1011101

$1\times2^6+0\times2^5+1\times2^4+1\times2^3+1\times2^2+0\times2^1+1\times2^0$

$64+0+16+8+4+0+1$

$= 93$

$\therefore (37)_{10} + (56)_{10} = (93)_{10}$

* BINARY - MULTIPLICATION *

Rules for Binary Multiplication

A	B	A·B
0	0	0
0	1	0
1	0	0
1	1	1

① Multiply $(1101)_2$ with $(110)_2$

```
    1 1 0 1
  x   1 1 0
-----------
    0 0 0 0
  1 1 0 1
1 1 0 1
-----------
1 0 0 1 1 1 0
-----------
```

$\therefore (1101)_2 \times (110)_2 = (1001110)_2$

② Multiply 1011.101 with 101.01

```
    1 0 1 1 . 1 0 1
  x   1 0 1 . 0 1
-----------------
        1 0 1 1 1 0 1
      0 0 0 0 0 0 0
    1 0 1 1 1 0 1
  0 0 0 0 0 0
1 0 1 1 1 0 1
-----------------
1 1 1 1 0 1 . 0 0 0 1
-----------------
```

$\therefore (1011.101)_2 \times (101.01)_2 = (111101.00001)_2$

* BINARY DIVISION *

Rules for Binary Division

$0 \div 1 = 0$

$1 \div 1 = 1$

① Divide $(11011011)_2$ by $(110)_2$

```
        10010 0.1
    ------------------
110 ) 11011011
      110
    ------
       0110
        110
      ------
          0110
           110
          ----
             0
```

$\therefore (11011011) \div (110) = (10010.1)_2$

② $(110101.11)_2 \div (101)_2$.

```
          1010.11
    ------------------
101 ) 110101.11
      101
    ------
       110
       101
      ------
         111
         101
        -----
          101
          101
         -----
            0
```

0 - 0 = 0

0 - 1 = 1 borrow 1

1 - 0 = 1

1 - 1 = 0

$\therefore (110101.11)_2 \div (101)_2 = (1010.11)_2$

* BINARY SUBTRACTION *

Rules for Binary subtraction

A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

10-1 = 1

1) $(11101100)_2 - (00110010)_2$

```
  1 1 1 0 1 1 0 0
(-) 0 0 1 1 0 0 1 0
-----------------
  1 0 1 1 1 0 1 0
-----------------  Ans
```

$\therefore (11101100)_2 - (00110010)_2 = (10111010)_2$.

2) $(1110)_2 - (1101)_2$

```
  1 1 1 0
(-) 1 1 0 1
-----------
  0 0 0 1
-----------  Ans
```

$\therefore (1110)_2 - (1101)_2 = (0001)_2$

3) $(10001 \cdot 01)_2 - (1110 \cdot 10)_2$

```
  1 0 0 0 1 . 0 1
(-)   1 1 1 0 . 1 0
-------------------
  0 0 0 1 0 . 1 1
-------------------  Ans
```

$\therefore (10001 \cdot 01)_2 - (1110 \cdot 10)_2 = (00010 \cdot 11)_2$

4) $(1001 \cdot 11)_2 - (1000 \cdot 11)_2$

```
  1 0 0 1 . 1 1
(-) 1 0 0 0 . 1 1
-----------------
  0 0 0 1 . 0 0
-----------------  Ans
```

$\therefore (1001 \cdot 11)_2 - (1000 \cdot 11)_2 = (0001 \cdot 00)_2$

Signed Binary Numbers

- Two ways of representing signed numbers:
  - 1) Sign-magnitude form, 2) Complement form.
- Most of computers use complement form for negative number notation.
- 1’s complement and 2’s complement are two different methods in this type.

1’s Complement

- 1’s complement of a binary number is obtained by subtracting each digit of that binary number from 1.
- Example

```
  1 1 1 1          1 1 1 . 1 1
- 1 1 0 1        - 1 0 1 . 0 1
---------        -------------
  0 0 1 0          0 1 0 . 1 0
```

(1’s complement of 1101) (1’s complement of 101.01)

Shortcut: Invert the numbers from 0 to 1 and 1 to 0

2’s Complement

- 2’s complement of a binary number is obtained by adding 1 to its 1’s complement.
- Example

```
    1 1 1 1          1 1 1 . 1 1
  - 1 1 0 0        - 1 0 1 . 0 1
  ---------        -------------
    0 0 1 1          0 1 0 . 1 0
  +       1        +         1
  ---------        -------------
    0 1 0 0          0 1 0 . 1 1
```

(2’s complement of 1100) (2’s complement of 101.01)

Shortcut: Starting from right side, all bits are same till first 1 occurs and then invert rest of the bits

Subtraction using 1's complement

- Using 1's complement
  - Obtain 1's complement of subtrahend
  - Add the result to minuend and call it intermediate result
  - If carry is generated then answer is positive and add the carry to Least Significant Digit (LSD)
  - If there is no carry then answer is negative and take 1's complement of intermediate result and place negative sign to the result.

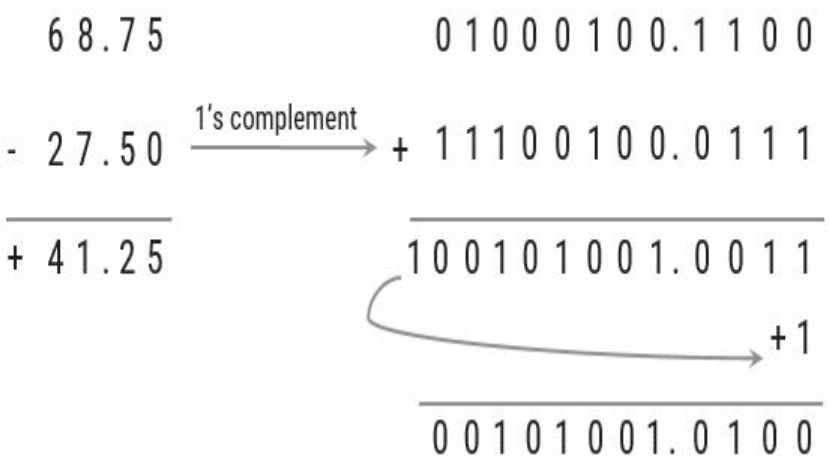
Subtraction using 2's complement

- Using 2's complement
  - Obtain 2's complement of subtrahend
  - Add the result to minuend
  - If carry is generated then answer is positive, ignore carry and result itself is answer
  - If there is no carry then answer is negative and take 2's complement of intermediate result and place negative sign to the result.

Subtraction using 1’s complement (Examples)

Example - 1

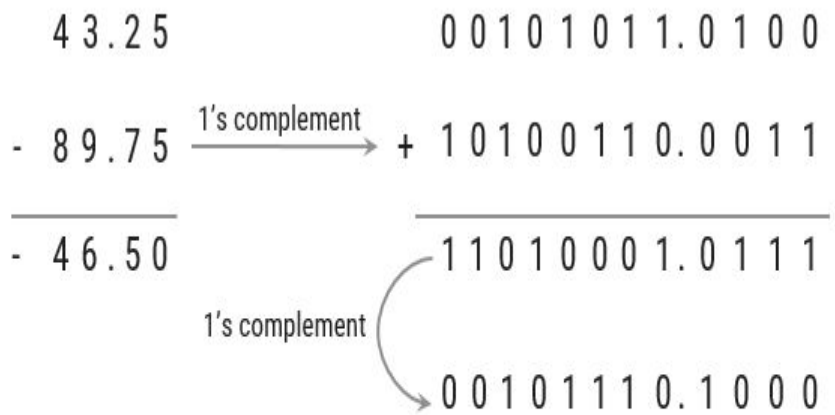
68.75 – 27.50



Subtraction using 1's complement (Examples)

Example - 2

43.25 - 89.75

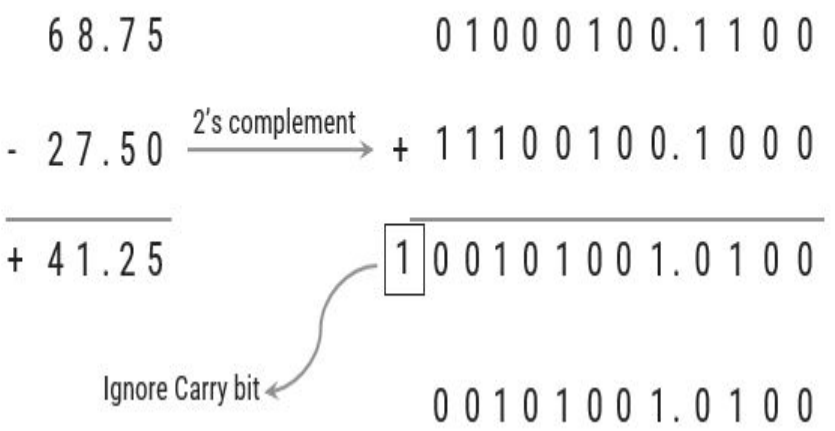


As carry is not generated, so take 1's complement of the intermediate result and add ' – ' sign to the result

Subtraction using 2's complement (Examples)

Example - 1

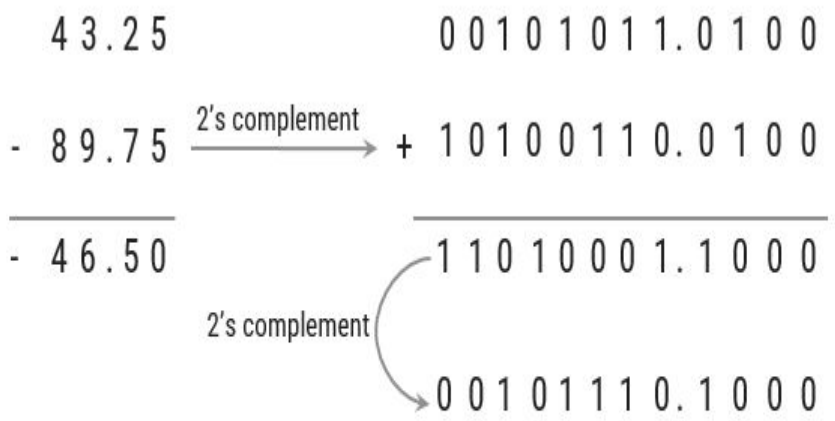
68.75 – 27.50



Subtraction using 2's complement (Examples)

Example - 2

43.25 - 89.75



As carry is not generated, so take 2's complement of the intermediate result and add ' – ' sign to the result

BCD ARITHMETIC

Decimal	Binary	BCD
0	0	0000
1	1	0001
2	10	0010
3	11	0011
4	100	0100
5	101	0101
6	110	0110
7	111	0111

Decimal	Binary	BCD
8	1000	1000
9	1001	1001
10	1010	0001 0000
11	1011	0001 0001
12	1100	0001 0010
13	1101	0001 0011
14	1110	0001 0100
15	1111	0001 0101

BCD Addition

Example - 1

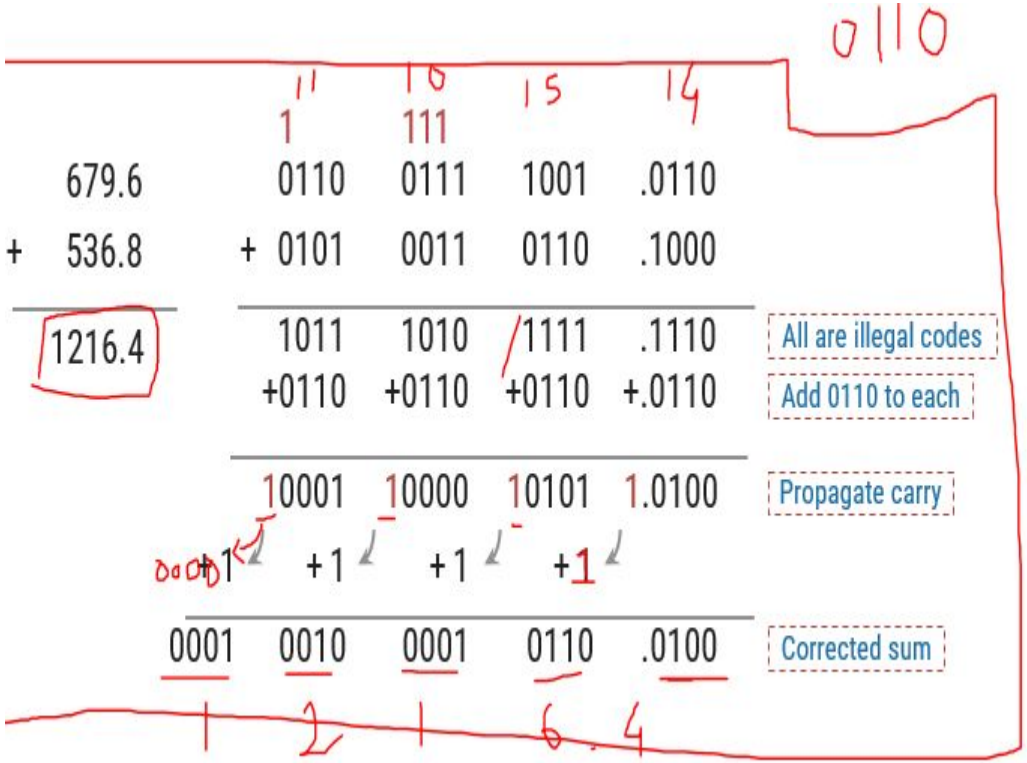
					1	1	1		
	2	5		0	0	1	0	0	1	0	1
+	1	3	+	0	0	0	1	0	0	1	1
	3	8		0	0	1	1	1	0	0	0

No carry, no illegal code. So, this is the correct sum.

Rule: If there is an illegal code or carry is generated as a result of addition, then add 0110 to particular that 4 bits of result.

BCD Addition

Example - 2



BCD Subtraction

Example - 1

	3	8			0	0	1	1	1 0 0 0
-	1	5		-	0	0	0	1	0 1 0 1
	2	3			0	0	1	0	0 0 1 1

No borrow. So, this is the correct difference.

Rule: If one 4-bit group needs to take borrow from neighbor, then subtract 0110 from the group which is receiving borrow.

BCD Subtraction

Example - 2

	206.7	0010	0000	0110	.0111
-	147.8	- 0001	0100	0111	.1000
	58.9	0000	1011	1110	.1111 ✓ Borrows are present
			-0110	-0110	-.0110 ✓ Subtract 0110
			0101	1000	.1001 ✓ Corrected difference
			5	8	9

Pg. No: 23

Examples for BCD Addition:

① 4+5

Solution:

```
  4          0100
+ 5        + 0101
---        ------
  9          1001   Ans BCD sum
             (9)
```

② 4+8

Solution:

```
  4            0100
+ 8          + 1000
---          ------
 12            1100   Binary Sum>9
             + 0110   Add 6
       0001    0010   Ans. BCD sum
         1       2
```

⑤ Using BCD, add 184+576

Solution:

```
  184              1         1
+ 576            0001      1000      0100
-----          + 0101 ↓    0111 ↓    0110   ↓
  760            ----      ----      ----
                 0111     10000      1010   Sum>9
               +           0110      0110   Add 6
                 ----      ----      ----
Ans BCD Sum =    0111      0110      0000
                   7         6         0
```

⑥ Using BCD, add 48+58

Solution:

```
  48               1
+ 58          0100      1000
----        + 0101 ↓    1000 ↓
 106          ----      ----
              1010     10000   Binary sum <9, with carry
            + 0110      0110   Add 6
              ----      ----
Ans   0001    0000      0110   BCD sum.
        1       0         6
```

⑦ Using BCD, add ...

③ 8 + 9

Solution

8	1000	
+9	+ 1001	
17	10001	Binary sum < 9, with carry
	+ 0110	Add 6
	0001 0111	Ans BCD sum
	1 7	

④ Using BCD Addition, add 24+18

Solution:

	1		
24	0010	0100	
+18	+ 0001 ↓	1000 ↓	
42	0100	1100	Binary sum > 9
	+	0110	Add 6
	0100	0010	Ans BCD sum
	4	2	

⑦ Using BCD, add 175 + 326

Solution:

		1	1	
175		0001	0111	0101
+326	+	0011 ↓	0010 ↓	0110 ↓
501		0101	1010	1011 Binary sum > 9,
	+		0110	0110 Add 6
Ans		0101	0000	0001 BCD sum
		5	0	1

⑧ Using BCD, add 589 + 199

Solution:

		1	1	
589		0101	1000	1001
+199	+	0001 ↓	1001 ↓	1001 ↓
788		0111	1 0010	1 0010 Sum < 9, carry
	+		0110	0110 Add 6
Ans		0111	1000	1000 BCD sum
		7	8	8

GRAY CODE

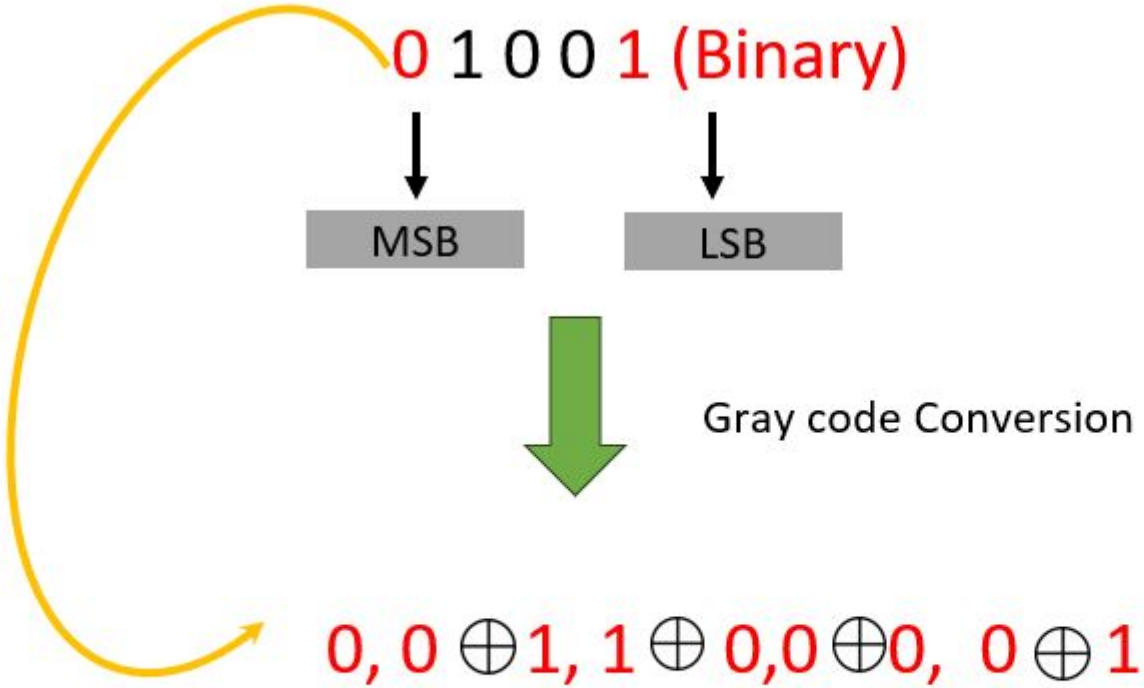
- Gray code is the arrangement of binary number system such that each incremental value can only differ by one bit.
- This code is also known as **Reflected Binary Code** (RBC), **Cyclic Code** and **Reflected Binary** (RB). The reason for calling this code as reflected binary code is the first N/2 values compared with those of the last N/2 values in reverse order.
- In gray code when transverse from one step to another step the only one bit will be change of the group. This means that the two adjacent code numbers differ from each other by only one bit.
- It is popular for unit distance code but it is not use from arithmetic operations. This code has some application like convert analog to digital, error correction in digital communication.

Binary- Gray code conversion

- **STEPS**
  - The most significant bit of gray code is equal to the first bit of the given binary bit.
  - The second bit of gray code will be exclusive-or (XOR) of the first and second bit of the given binary bit.
  - The third bit of gray code is equal to the exclusive-or (XOR) of the second and third binary bits. For father gray code result this process will be continuing.

Explanation

- The given binary digit is 01001



Truth Table of XOR

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

0, 0 ⊕ 1, 1 ⊕ 0,0 ⊕ 0, 0 ⊕ 1

0 , 1 , 1 , 0 , 1 (On concatening)

The gray code of the given binary code (01001)

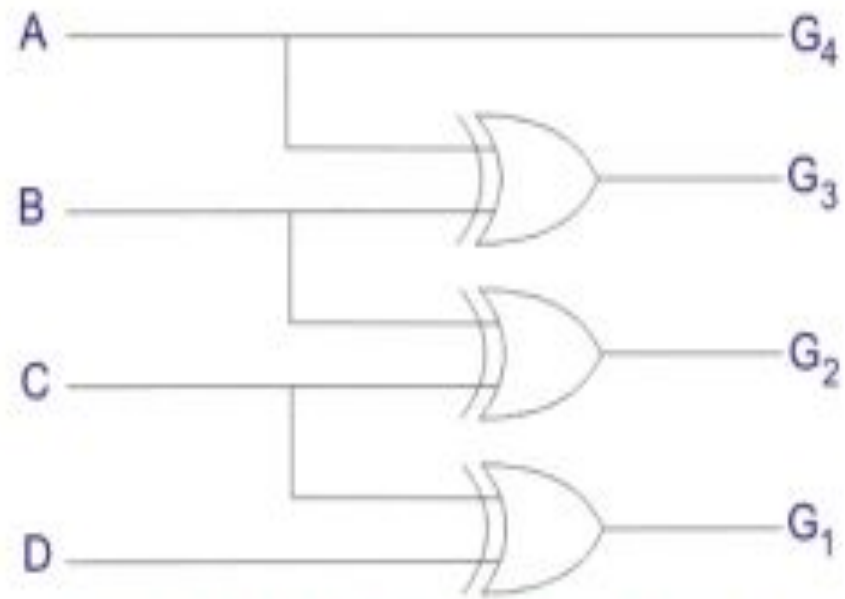
=0 1 **1** 0 1 (One bit changed)

28

Example

- The gray code of the given binary code is $(010.01)_2$??
  - The first MSB bit of binary is same in the first bit of gray code. In this example the binary bit is “0”. So, gray bit also “0”.
  - Next gray bit is equal to the XOR of the first and the second binary bit. The first bit is 0, and the second bit is 1. The bits are different so resultant gray bit will be “1” (second gray codes bit)
  - The XOR of the second and third binary bit. The second bit is 1 and third is 0. These bits are again different so the resultant gray bit will be 1 (third gray codes bit)
  - Next we perform the XOR operation on third and fourth binary bit. The third bit is 0, and the fourth bit is 0. The both bits are same than resultant gray codes will be 0 (fourth gray codes bit).
  - Take the XOR of the fourth and fifth binary bit. The fourth bit is 0 and fifth bit is 1. These bits are different than resultant gray codes will be 1 (fifth gray code bit)
  - The result of binary to gray codes conversion is 01101.

- You can convert n bit ($b_n b_{(n-1)} \ldots b_2 b_1 b_0$) binary number to gray code ($g_n g_{(n-1)} \ldots g_2 g_1 g_0$). For most significant bit $b_n = g_n$, and rest of the bit by XORing $b_{(n-1)} = g_{(n-1)} \oplus g_n$, ….



Logic Circuit for Binary to Gray Code Converter

GRAY CODE TABLE

The conversion in between decimal to gray and binary to gray code is given below

Decimal Number	Binary Number	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

The gray code of the given binary code (01001) =0 1 1 0 1. We can see one bit change in the next incremental value.

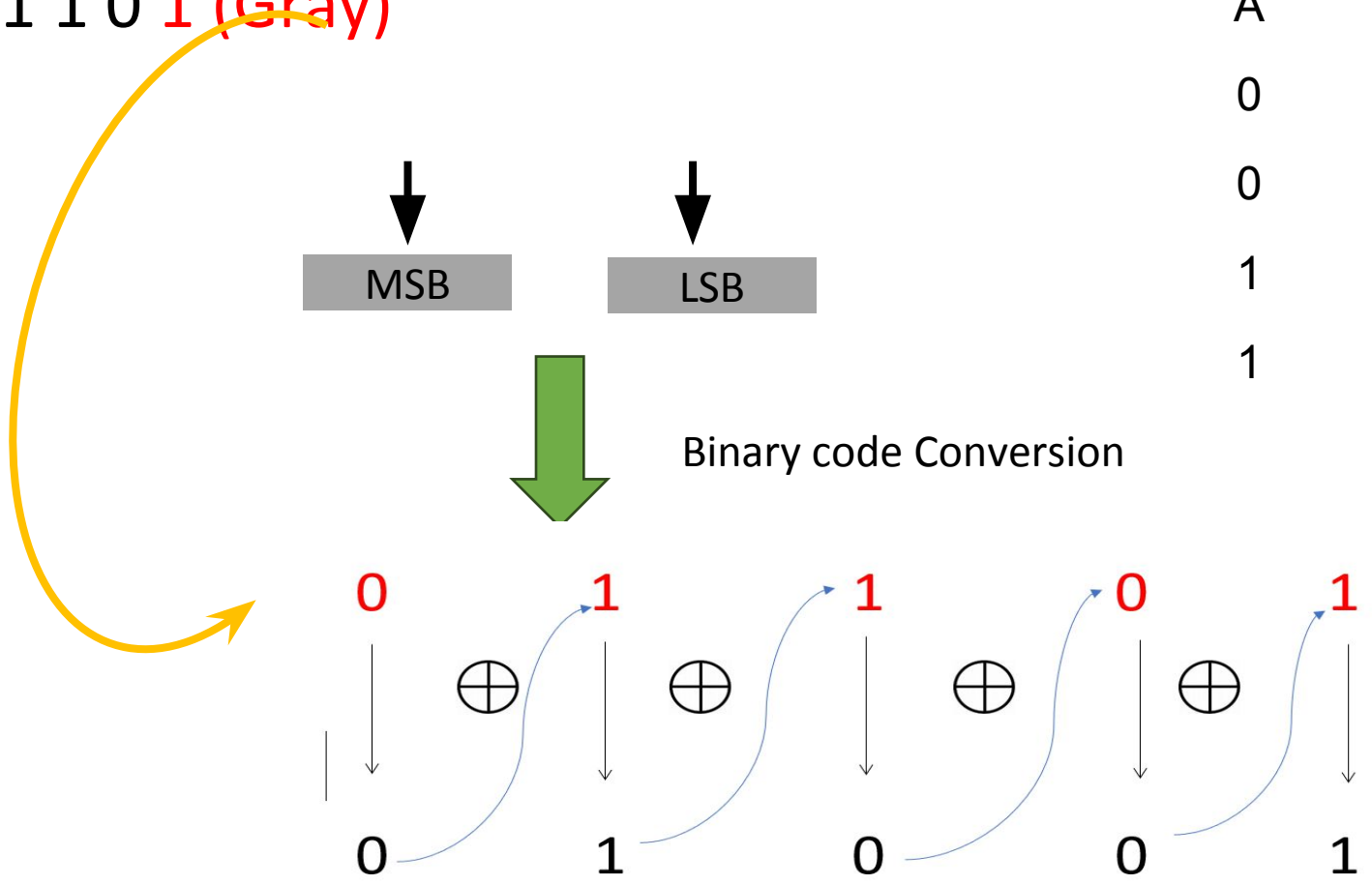
Gray-Binary code conversion

- **STEPS**
  1. The most significant bit of gray codes is equal in binary number.
  2. Now move to the next gray bit, as it is 1 the previous bit will be alter i.e it will be 1, thus the second binary bit will be 1.
  3. Next see the third bit, in this example the third bit is 1 again, the third binary bit will be alter of second binary bit and the third binary bit will be 0.
  4. Now fourth bit of the, here the fourth bit of gray code is 0. So the fourth bit will be same as a previous binary bit, i.e 4th binary bit will be 0.
  5. The last fifth bit of gray codes is 1; the fifth binary number is altering of fourth binary number.
  6. Therefore the gray code (01101) equivalent in binary number is (01001)

Explanation

- The given gray code is 01101

0 1 1 0 1 (Gray)



Truth Table of XOR

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

Merits & Demerits of Gray Code

Advantages of gray code

- It is best for error minimization in conversion of analog to digital signals.
- It is best for minimize a logic circuit
- Decreases the “Hamming Walls” which is undesirable state, when used in genetic algorithms
- It is useful in clock domain crossing

Disadvantages of gray code

- Not suitable for arithmetic operations
- It has limited use.

Binary Codes for Decimal Digits

- There are over 8,000 ways that you can chose 10 elements from the 16 binary numbers of 4 bits. A few are useful:

Decimal	8,4,2,1	Excess3	8,4,-2,-1	Gray
0	0000	0011	0000	0000
1	0001	0100	0111	0100
2	0010	0101	0110	0101
3	0011	0110	0101	0111
4	0100	0111	0100	0110
5	0101	1000	1011	0010
6	0110	1001	1010	0011
7	0111	1010	1001	0001
8	1000	1011	1000	1001
9	1001	1100	1111	1000

Binary Coded Decimal (BCD)

- The BCD code is the 8,4,2,1 code.
- This code is the simplest, most intuitive binary code for decimal digits and uses the same powers of 2 as a binary number, but only encodes the first ten values from 0 to 9.
- Example: 1001 (9) = 1000 (8) + 0001 (1)
- How many “invalid” code words are there?
- What are the “invalid” code words?

Excess 3 Code and 8, 4, –2, –1 Code

Decimal	Excess 3	8, 4, –2, –1
0	0011	0000
1	0100	0111
2	0101	0110
3	0110	0101
4	0111	0100
5	1000	1011
6	1001	1010
7	1010	1001
8	1011	1000
9	1100	1111

- **What interesting property is common to these two codes?**

Decimal to BCD

- *Convert* $(324)_{10}$ *in BCD*
- *From truth table the BCD value of*
  - $(3)_{10} = (0011)_{BCD}$
  - $(2)_{10} = (0010)_{BCD}$
  - $(4)_{10} = (0100)_{BCD}$

So, $(324)_{10} = (0011\ 0010\ 0100)_{BCD}$ → (A)

$(324)_{10} = (101000100)_2 \rightarrow (B)$

- On comparing (A) and (B) we understand that binary and BCD value of the given decimal is not same.

Binary to BCD conversion

- Step 1: Convert binary to decimal number
- Step 2 : Convert decimal number to BCD
  - Consider a binary number $(11101)_2$ and convert it to BCD.

Step 1: Convert binary to decimal number

$$(1\text{-}1\text{1}0\text{1}\text{-})_2 = ((1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))$$

$$= 16 + 8 + 4 + 0 + 1 = (29)_{10}\ (Decimal)$$

Step 2 : Convert decimal number to BCD (in 4 digit) (Write the 4 digit BCD value for individual digit in decimal)

$$(1\text{-}1\text{1}0\text{1}\text{-})_2 = (29)_{10} = (00\text{-}10\ 1\text{-}00\text{-}1)_{BCD}$$

Advantages of BCD

- Easy to **encode and Decode** decimals into BCD and Vice-versa
- Easy to implement a **hardware algorithm** for BCD converter
- Very useful in **digital systems** whenever decimal information reqd.
  - **Digital voltmeters, frequency converters** and **digital clocks** all use BCD as they display output information in decimal.

Disadvantages of BCD Code

- **Require more bits** than straight binary code
- Difficult to be used in **high speed digital computer** when the size and capacity of their internal registers are **restricted or limited**.
- The arithmetic operations using BCD code require a **complex design of Arithmetic and Logic Unit (ALU)** than the straight binary number system.
- The speed of the arithmetic operations that can be realized using BCD code is naturally slow due to the **complex hardware circuitry** involved.
-

Excess-3 code

- The excess-3 code is also treated as **XS-3 code**. The excess-3 code is a **non-weighted** and **self-complementary BCD code** used to represent the decimal numbers.
- This code has a **biased representation**. This code plays an **important role in arithmetic operations** because it resolves **deficiencies encountered** when we use the 8421 BCD code for adding two decimal digits whose sum is greater than 9.
- The Excess-3 code uses a **special type of algorithm**, which differs from the binary positional number system or normal non-biased BCD.

Decimal to Excess-3 code conversion

Step-1: We find the decimal number of the given binary number. Step-2: Then we add 3 in each digit of the decimal number.

Step-3: Now, we find the binary code of each digit of the newly generated decimal number.

Alternatively,

- We can also add 0011 in each 4-bit BCD code of the decimal number for getting excess-3 code.

Ex-1

- Convert decimal $(5)_{10}$ to Excess-3.
- Step 1: Add ‘3’ to the given decimal → 5+3=8
- Step 2: Find binary digit of a new number ‘8’ which is $(8)_2 = (1000)_2$.
- The Excess-3 code of given decimal $(5)_{10}$ is $(1000)_{Excess-3}$.

Ex-2

- Convert BCD $(0101)_{BCD}$ *to* Excess-3.

Step 1: Add BCD value of 3 (0011) to the given number.

0 1 0 1

0 0 1 1(+)

1 0 0 0

$(0101)_{BCD} = (1000)_{Excess-3}$

Ex-3

- Convert decimal $(26)_{10}$ *to* Excess-3.
- Step 1: Add '3' to the individual given decimal

2	6	
+3	+3	Add 3 to each digit
5	9	
↓	↓	
0101	1001	Convert to a 4-bit binary code.

- The Excess-3 code of given decimal $(26)_{10}$ is $(0101\ 1001)_{Excess-3}$.

Excess-3 code

Decimal	*BCD*	*Excess-3*
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Ex-4

- Convert decimal $(81.61)_{10}$ *to* Excess-3.
- Step 1:

Decimal	BCD
8	1000
1	0001
6	0110
1	0001

Step 2:

BCD+ 0011	Excess-3
1000+0011	1011
0001+0011	0100
0110+0011	1001
0001+0011	0100

$$(81.61)_{10} = (1011\ 0100.1001\ 0100)_{Excess-3}$$

Ex 5

- **Convert $(11110)_2$ to Excess-3 using binary**
- **Step 1: Convert binary to Decimal. $(11110)_2 = (30)_{10}$**
- **Step 2: Add '3' to individual digits to decimal number**

3 0

3 3

6 3

Step 3: Find binary values of $(63)_{10} = (01100011)_{\text{Excess-3}}$

Ex 6

- Convert $(01100011)_{\text{Excess-3}}$ to binary.
- **Step- 1 : Find the decimal by dividing it four digits**
  - $(01100011)_{\text{Excess-3}} = (0110\ 0011)_{\text{Excess-3}} = (30)_{10}$
- **Step-2 : Find the binary value of the decimal through division method.**

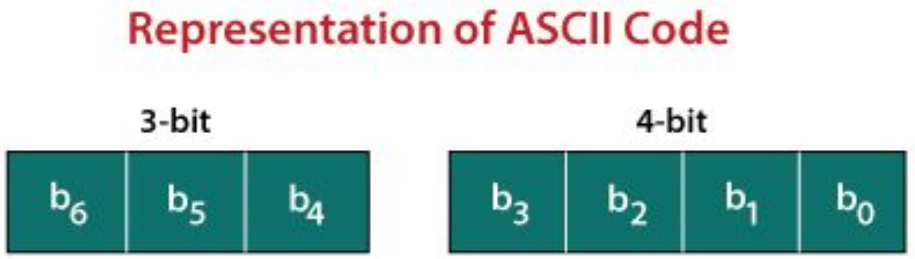
$$(01100011)_{\text{Excess-3}} = (11110)_2$$

Advantages

1. These codes are **self-complementary**.
2. These codes **use biased representation**.
3. The excess-3 code has **no limitation**, so that it considerably simplifies arithmetic operations.
4. The codes 0000 and 1111 can cause a fault in the transmission line. The excess-3 code doesn't use these codes and gives an advantage for memory organization.
5. These codes are usually **unweighted binary decimal codes**.
6. This code has a **vital role** in arithmetic operations. It is because it resolves deficiencies which are encountered when we use the 8421 BCD code for adding two decimal digits whose sum is greater than 9.

ASCII

- The ASCII stands for **American Standard Code for Information Interchange.** The ASCII code is an **alphanumeric code** used for data communication in digital computers.
- The ASCII is a 7-bit code capable of representing 2^7 or **128** number of different characters. The ASCII code is made up of a three-bit group, which is followed by a four-bit code.



ASCII Characters

- Control Characters-0 to 31 and 127
- Special Characters- 32 to 47, 58 to 64, 91 to 96, and 123 to 126
- Numbers Characters- 0 to 9
- Letters Characters - 65 to 90 and 97 to 122

Ex 1

- Encode (1001010110000111101101100001101010011100001101111 11010011 1101110111010010000000110001011001001100 11)$_2$ to ASCII.

- Step 1:
- The given binary data is grouped into 7-bits because the ASCII code is 7 bit.
- 1001010 1100001 1110110 1100001 1010100 1110000 1101111 1101001 1101110 1110100 1000000 0110001 0110010 0110011
- Step 2:
- Then, we find the equivalent decimal number of the binary digits either from the ASCII table or **64 32 16 8 4 2 1** scheme.

Cont'd

Binary	64	32	16	8	4	2	1	DECIMAL	ASCII
1100011	1	1	0	0	0	1	1	99	C
1101111	1	1	0	1	1	1	1	111	O
1101101	1	1	0	1	1	0	1	109	M
1110000	1	1	1	0	0	0	0	112	P
1110101	1	1	1	0	1	0	1	117	U
1110100	1	1	1	0	1	0	0	116	T
1100101	1	1	0	0	1	0	1	101	E
1110010	1	1	1	0	0	1	0	114	R

So the given binary digits results in ASCII Keyword COMPUTER

Parity Code

- The parity code is used for the purpose of **detecting errors** during the transmission of binary information. The parity code is a bit that is included with the binary data to be transmitted.
- The inclusion of a parity bit will make the number of **1's either odd** or **even.** Based on the number of 1's in the transmitted data, the parity code is of two types.
  - Even parity code
  - Odd parity code

- In even parity, the added parity bit will make the total number of 1's **an even number**.
- If the added parity bit make the total number of 1's as **odd number**, such parity code is said to be odd parity code.

Explanation

4-bit message

1	0	1	1

Adding 1 to the data to detect an error.
Total no. of 1's is an even number. So, it is Even parity

1	0	1	1	1

4-bit message

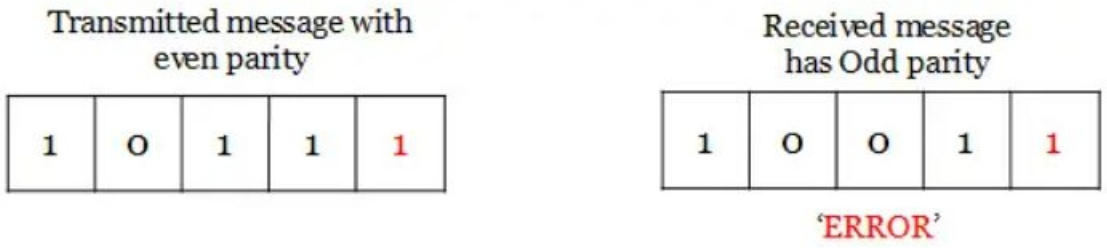
1	0	1	1

Adding 0 to the data to detect an error.
Total no. of 1's is an odd number. So, it is odd parity

1	0	1	1	0

Parity Bit

- On the receiver side, if the received data is other than the sent data, then it is an error. If the sent date is even parity code and the received data is odd parity, then there is an error.



- So, both even and odd parity codes are used only for the detection of error and not for the correction in the transmitted data. Even parity is commonly used and it has almost become a convention.

Logic Gates

SRM INSTITUTE OF SCIENCE & TECHNOLOGY (Deemed to be University u/s 3 of UGC Act, 1956)

Goal:

To understand how digital a computer can work, at the lowest level.

To understand what is possible and the limitations of what is possible for a digital computer.

Logic Gates

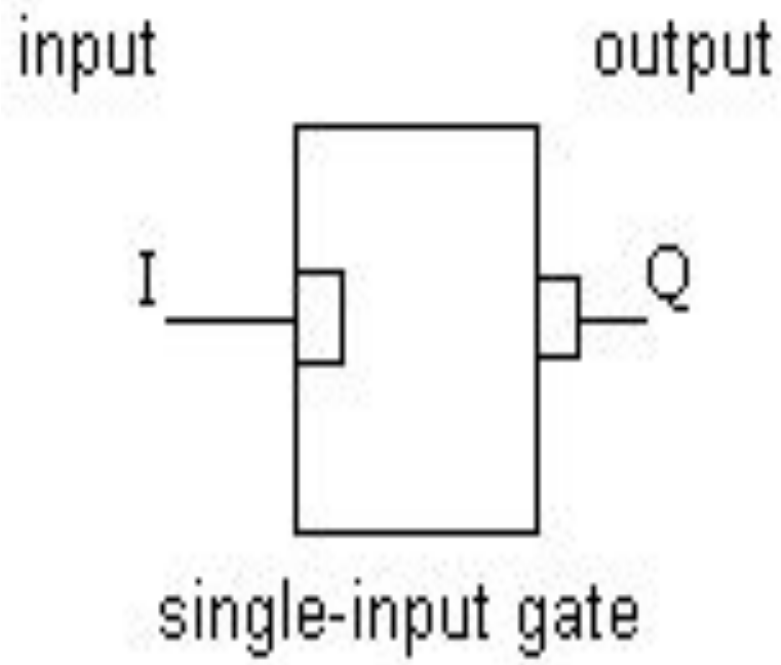
- All digital computers for the past 50 years have been constructed using the same type of components.
- These components are called logic gates.
- Logic gates have been implemented in many different ways.
- Currently, logic gates are most commonly implemented using electronic VLSI transistor logic.

Logic Gates

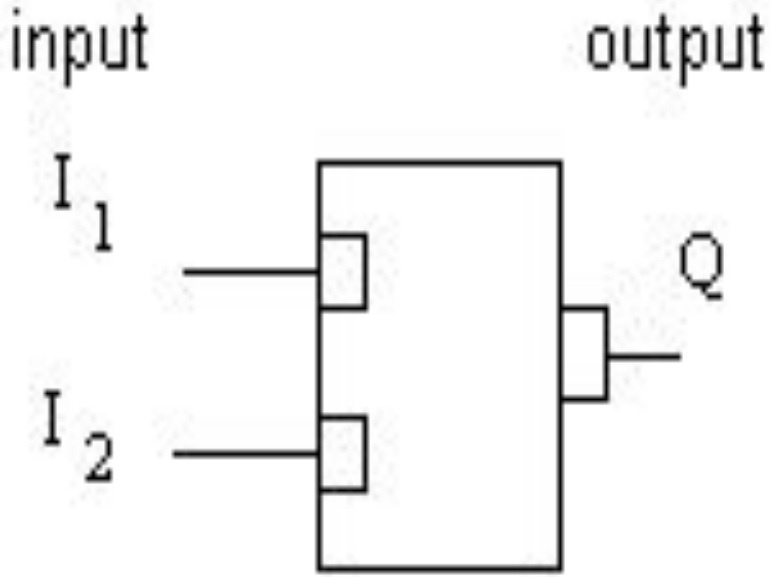
- A logic gate is a simple switching circuit that determines whether an input pulse can pass through to the output in digital circuits.
- The building blocks of a digital circuit are logic gates, which execute numerous logical operations that are required by any digital circuit.
- These can take two or more inputs but only produce one output.
- The mix of inputs applied across a logic gate determines its output. Logic gates use Boolean algebra to execute logical processes.
- Logic gates are found in nearly every digital gadget we use on a regular basis.
- Logic gates are used in the architecture of our telephones, laptops, tablets, and memory devices.

Logic Gates

All basic logic gates have the ability to accept either one or two input signals (depending upon the type of gate) and generate one output signal.



single-input gate

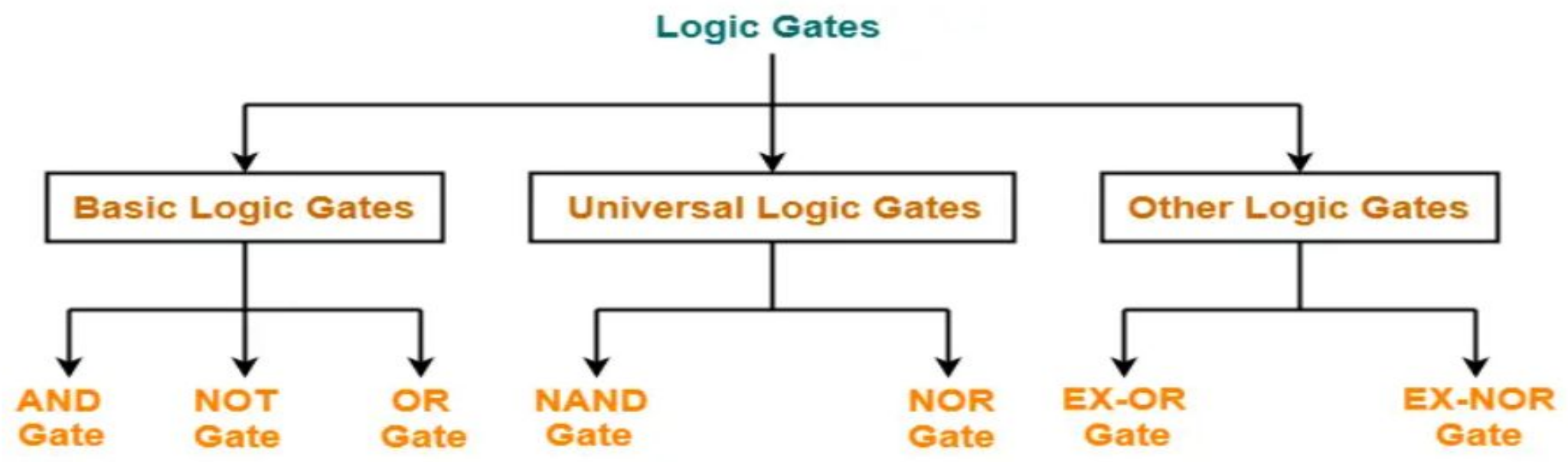


two-input gate

Truth Table

- The outputs for all conceivable combinations of inputs that may be applied to a logic gate or circuit are listed in a truth table.
- When we enter values into a truth table, we usually express them as 1 or 0, with 1 denoting True logic and 0 denoting False logic.

Classification



Types of Logic Gates

Basic Logic Gates-

- Basic Logic Gates are the fundamental logic gates using which universal logic gates and other logic gates are constructed.

They have the following properties-

- Basic logic gates are associative in nature.
- Basic logic gates are commutative in nature.

There are following three basic logic gates-

1. AND Gate
2. OR Gate
3. NOT Gate

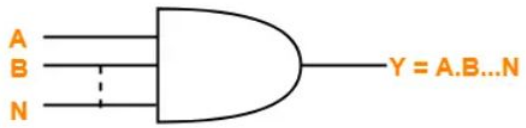
AND gate

- The output of AND gate is high (‘1’) if all of its inputs are high (‘1’).
- The output of AND gate is low (‘0’) if any one of its inputs is low (‘0’).

Logic Symbol-

The logic symbol for AND Gate is as shown below-

2-Input AND Gate



N-Input AND Gate

A	B	Y = A.B
0	0	0
0	1	0
1	0	0
1	1	1

Truth Table

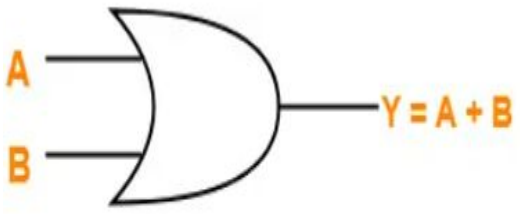
OR Gate

The output of OR gate is high (‘1’) if any one of its inputs is high (‘1’).

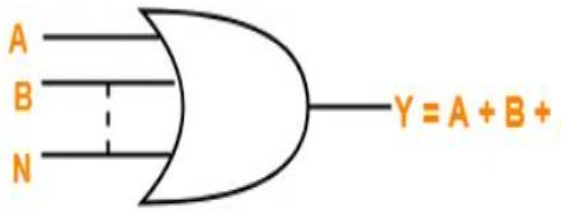
The output of OR gate is low (‘0’) if all of its inputs are low (‘0’).

Logic Symbol-

The logic symbol for OR Gate is as shown below-



2-Input OR Gate



N-Input OR Gate

A	B	Y = A + B
0	0	0
0	1	1
1	0	1
1	1	1

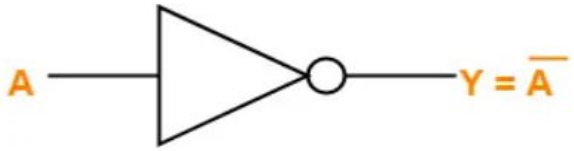
Truth Table

NOT Gate

- The output of NOT gate is high (‘1’) if its input is low (‘0’).
- The output of NOT gate is low (‘0’) if its input is high (‘1’).

From here-

- It is clear that NOT gate simply inverts the given input.
- Since NOT gate simply inverts the given input, therefore it is also known as **Inverter Gate**.



NOT Gate

A	Y = A’
0	1
1	0

Truth Table

Universal Logic Gates

Universal logic gates are the logic gates that are capable of implementing any Boolean function without requiring any other type of gate.

They are called as "**Universal Gates**" because-

- They can realize all the binary operations.
- All the basic logic gates can be derived from them.

They have the following properties-

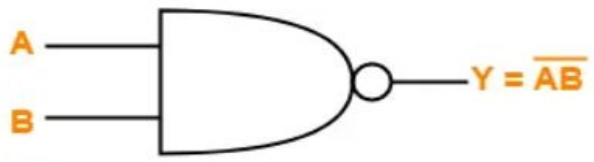
- Universal gates are not associative in nature.
- Universal gates are commutative in nature.

NAND GATE

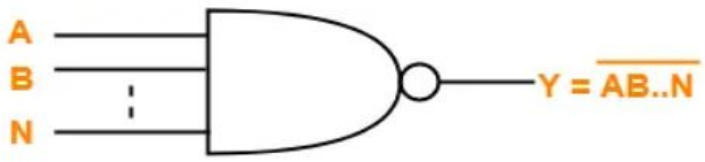
- A NAND Gate is constructed by connecting a NOT Gate at the output terminal of the AND Gate.
- The output of NAND gate is high (‘1’) if at least one of its inputs is low (‘0’).
- The output of NAND gate is low (‘0’) if all of its inputs are high (‘1’).

Logic Symbol-

The logic symbol for NAND Gate is as shown below-



2-Input NAND Gate



N-Input NAND Gate

A	B	Y = (A.B)’
0	0	1
0	1	1
1	0	1
1	1	0

Truth Table

NOR Gate

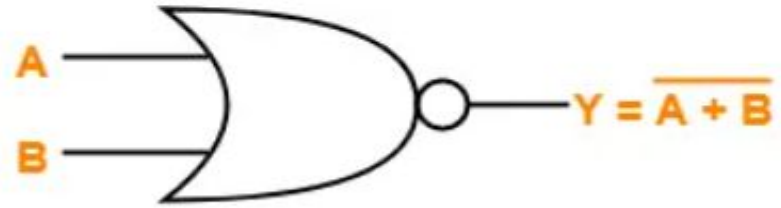
- A NOR Gate is constructed by connecting a NOT Gate at the output terminal of the OR Gate.
- The output of OR gate is high (‘1’) if all of its inputs are low (‘0’).
- The output of OR gate is low (‘0’) if any of its inputs is high (‘1’).

Logic Symbol-

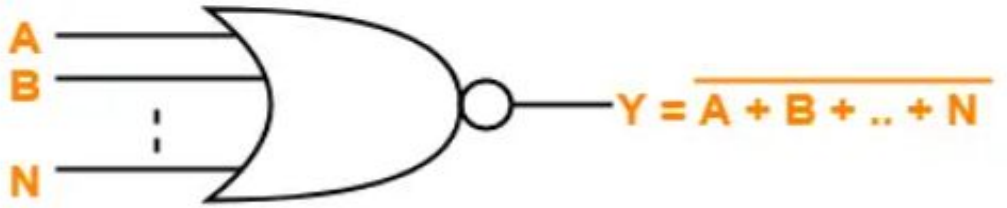
The logic symbol for NOR Gate is as shown below-

A	B	Y = A + B
0	0	1
0	1	0
1	0	0
1	1	0

Truth Table



2-Input NOR Gate



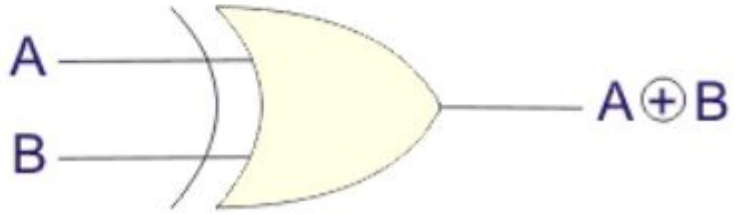
N-Input NOR Gate

EX-OR

- An XOR gate (also known as an EOR, or EXOR gate) – pronounced as “Exclusive OR gate” – is a digital logic gate that gives a true (i.e. a HIGH or 1) output when the number of true inputs is odd.
- An XOR gate implements an exclusive OR, i.e., a true output result occurs if one – and only one – of the gate’s inputs is true. If both inputs are false (i.e. LOW or 0) or both inputs are true, the output is false.

Logical Symbol of XOR Gate

An XOR gate is logically represented as,



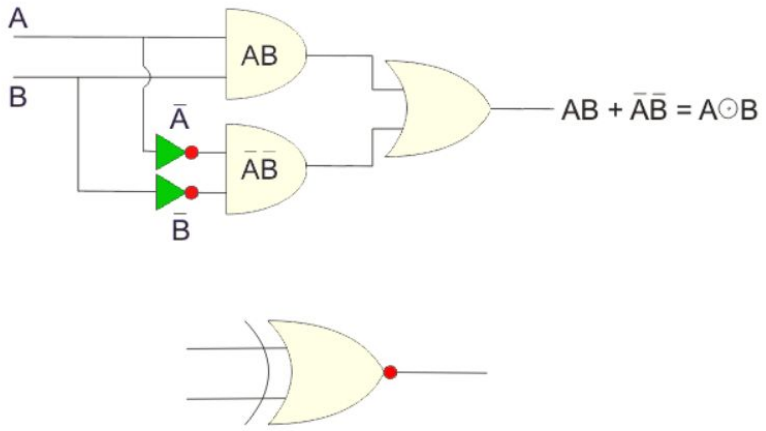
XOR Gate Truth Table

Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

EX-NOR

- The XNOR gate (also known as an XORN'T, ENOR, EXNOR or NXOR) – and pronounced as Exclusive NOR – is a digital logic gate whose function is the logical complement of the exclusive OR gate (XOR gate). Logically, an XNOR **gate** is a NOT gate followed by an XOR gate.
- The XOR operation of inputs A and B is $A \oplus B$; therefore, the XNOR operation of those inputs will be $\overline{(A + B)}$. That means the output of the XOR gate is inverted in the XNOR gate.

The symbol of the XNOR gate:



XNOR Gate Truth Table

Inputs		Output
A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

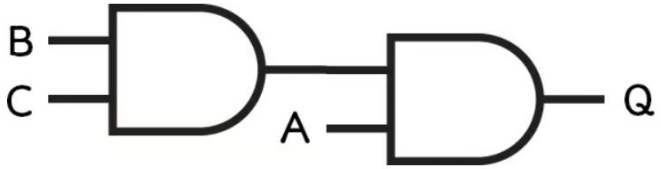
Example

Example 1

Q = A AND (B AND C)

Step 1 – Start with the brackets, this is the “B AND C” part.

Step 2 – Add the outer expression, this is the “A AND” part.

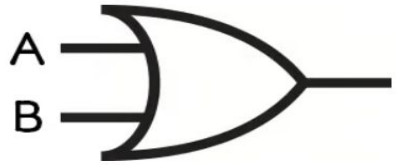


Example

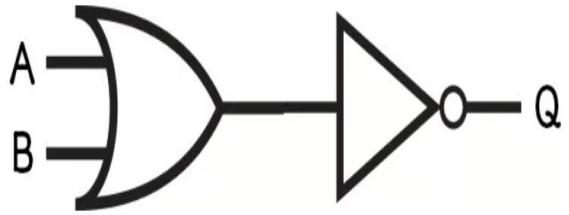
Example 2.

Q = NOT (A OR B)

Step 1 – Start with the brackets, this is the “A OR B” part

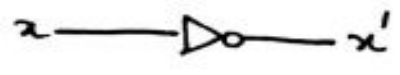
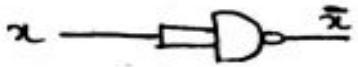


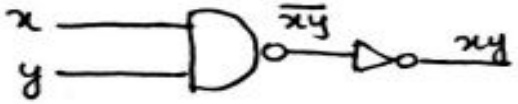
Step 2 – Add the outer expression, this is the “NOT” part.

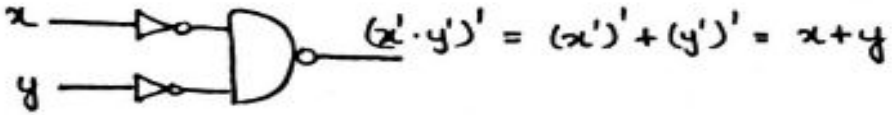


NAND CIRCUITS IMPLEMENTATION

* The NAND gate is said to be a 'universal gate' because any logic circuit can be implemented with it.
* The implementation of AND, OR, NOT operations with NAND gate are as follows:

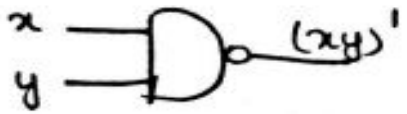
Inverter  (or) 

AND 

OR 

. Graphic symbols for NAND Gate.

① NAND Gate



≡

② Invert-OR Gate

x	y	xy	$(xy)'$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

$\equiv$

x	y	x'	y'	$x'+y'$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

$\therefore$ Invert-OR Gate is equivalent to NAND Gate.

NOR CIRCUITS IMPLEMENTATION

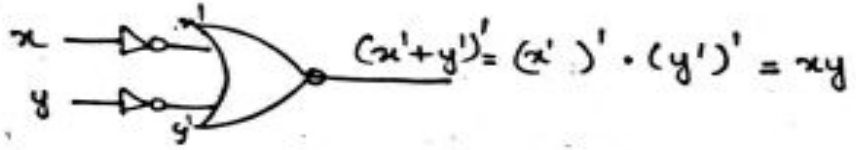
* The NOR gate is another universal gate that can be used to implement any Boolean function.

* The NOR operation is the dual of the NAND operation.

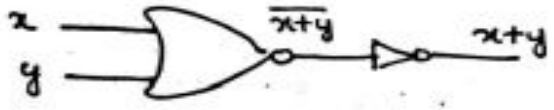
* The implementation of AND, OR and NOT operations with NOR gates are as follows.

Inverter

AND

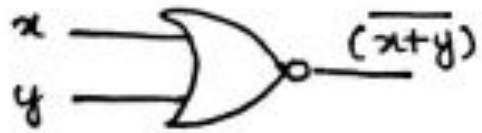


OR



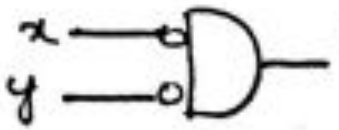
. Graphic symbols for NOR Gate .

① NOR Gate



≡

② Invert-AND gate



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

x	y	x+y	$\overline{x+y}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

≡

x	y	$\bar{x}$	$\bar{y}$	$\bar{x} \cdot \bar{y}$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

∴ Invert-AND is equivalent to a NOR Gate.