



# KTU NOTES

The learning companion.

**KTU STUDY MATERIALS | SYLLABUS | LIVE  
NOTIFICATIONS | SOLVED QUESTION PAPERS**

Website: [www.ktunotes.in](http://www.ktunotes.in)

# **CST 203 LOGIC SYSTEM DESIGN**

## **MODULE 1**

### **Number Systems Operations and Codes**

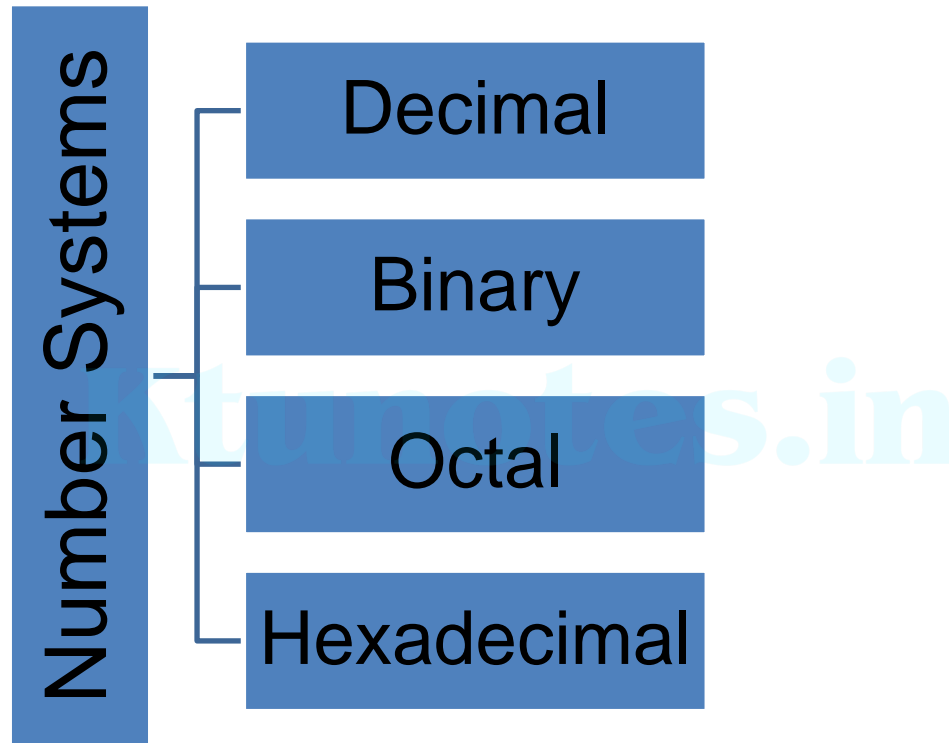
# LOGIC DESIGN

- Digital logic design is basis of electronic systems, such as computers and cell phones
- Digital Logic is rooted in binary code, a series of zeroes and ones
- Digital Logic Design is foundational to fields of electrical engineering and computer engineering
- Digital Logic designers build complex electronic components that use both electrical and computational characteristics
- Digital Logic Design is used to develop hardware, such as circuit boards and microchip processors

# SWITCHING THEORY

- Switching Theory is about using switches to implement Boolean expressions and logic gates for logic design of digital circuits
- It can be used to develop theoretical knowledge and concepts of digital circuits when viewed as an interconnection of input elements producing an output state
- Digital logic gates whose inputs and output can switch between two distinct logical values of 0 and 1

# DIGITAL NUMBER SYSTEMS



# DECIMAL NUMBER SYSTEMS

Decimal Numbering System (base 10)

Characters = 0,1,2,3,4,5,6,7,8,9

4	8	7	2	=	4x1000	+	8x100	+	7x10	+	2x1
Thousand's Place	Hundred's Place	Ten's Place	One's Place								

written  $4872_d$  or  $4872_{10}$

# DECIMAL NUMBER SYSTEMS (CONT...)

- Position of each digit in a weighted number system is assigned a weight based on **base** or **radix** of system
- Radix of decimal numbers is ten
- Ten symbols (0 through 9) are used to represent any number
- For fractional decimal numbers, column weights are negative powers of ten that decrease from left to right:

... $10^5$   $10^4$   $10^3$   $10^2$   $10^1$   $10^0$ .

- Column weights of decimal numbers are powers of ten that increase from right to left beginning with  $10^0 = 1$ :

$10^2$   $10^1$   $10^0$ .  $10^{-1}$   $10^{-2}$   $10^{-3}$   $10^{-4}$  ...

# DECIMAL NUMBER SYSTEMS (CONT...)

- Decimal numbers can be expressed as sum of products of each digit times column value for that digit
- Number 9240 can be expressed as

$$(9 \times 10^3) + (2 \times 10^2) + (4 \times 10^1) + (0 \times 10^0)$$

or

$$9 \times 1,000 + 2 \times 100 + 4 \times 10 + 0 \times 1$$

**Example**

Express number 480.52 as the sum of values of each digit

**Solution**

$$480.52 = (4 \times 10^2) + (8 \times 10^1) + (0 \times 10^0) + (5 \times 10^{-1}) + (2 \times 10^{-2})$$



# BINARY NUMBER SYSTEMS

Binary Numbering System (base 2)

Characters = 0 and 1

$$\begin{array}{cccc} 1 & 1 & 0 & 1 \\ \text{8's place} & \text{4's place} & \text{2's place} & \text{1's place} \end{array} = 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1$$

written  $1101_b$  or  $1101_2$

# BINARY NUMBER SYSTEMS (CONT...)

- For digital systems, binary number system is used
- Binary has a radix of two and uses digits 0 and 1
- Column weights of binary numbers are powers of two that increase from right to left beginning with  $2^0 = 1$ :

... $2^5$   $2^4$   $2^3$   $2^2$   $2^1$   $2^0$ .

- For fractional binary numbers, column weights are negative powers of two that decrease from left to right:

$2^2$   $2^1$   $2^0$ .  $2^{-1}$   $2^{-2}$   $2^{-3}$   $2^{-4}$  ...

# OCTAL NUMBER SYSTEMS (CONT...)

Octal Numbering System (base 8)

Characters = 0,1,2,3,4,5,6,7

4 3 7

$$= 4 \times 64 + 3 \times 8 + 7 \times 1$$

64's place   8's place   1's place

written  $437_6$  or  $437_8$

# OCTAL NUMBER SYSTEMS (CONT...)

- Octal is also a weighted number system
- Column weights are powers of 8, which increase from right to left

$$\text{Column weights} \begin{cases} 8^3 & 8^2 & 8^1 & 8^0 \\ 512 & 64 & 8 & 1 \end{cases}$$

**Example** Express  $3702_8$  in decimal.

**Solution** Start by writing the column weights:

$$\begin{array}{cccc} 512 & 64 & 8 & 1 \\ 3 & 7 & 0 & 2_8 \end{array}$$

$$3(512) + 7(64) + 0(8) + 2(1) = 1986_{10}$$

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

# OCTAL NUMBER SYSTEMS (CONT...)

- Octal uses eight characters the numbers 0 through 7 to represent numbers
- There is no 8 or 9 character in octal
- Binary number can easily be converted to octal by grouping bits 3 at a time and writing equivalent octal character for each group

**Example** Express  $1\ 001\ 011\ 000\ 001\ 110_2$  in octal:

**Solution** Group the binary number by 3-bits starting from the right. Thus,  
 $113016_8$

# HEXADECIMAL NUMBER SYSTEMS

Hexadecimal Numbering System (base 16)

Characters = 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

$$\begin{array}{ccc} \text{E} & 4 & \text{C} \\ \text{256's place} & \text{16's place} & \text{1's place} \end{array} = 14 \times 256 + 4 \times 16 + 12 \times 1$$

written  $\text{E4C}_h$  or  $\text{E4C}_{16}$

# HEXADECIMAL NUMBER SYSTEMS (CONT...)

- Hexadecimal is a weighted number system
- Column weights are powers of 16, which increase from right to left

Column weights  $\begin{cases} 16^3 & 16^2 & 16^1 & 16^0 \\ 4096 & 256 & 16 & 1 \end{cases}$

**Example** Express  $1A2F_{16}$  in decimal.

**Solution** Start by writing the column weights:

4096 256 16 1  
1 A 2  $F_{16}$

$$1(4096) + 10(256) + 2(16) + 15(1) = 6703_{10}$$

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

# HEXADECIMAL NUMBER SYSTEMS (CONT...)

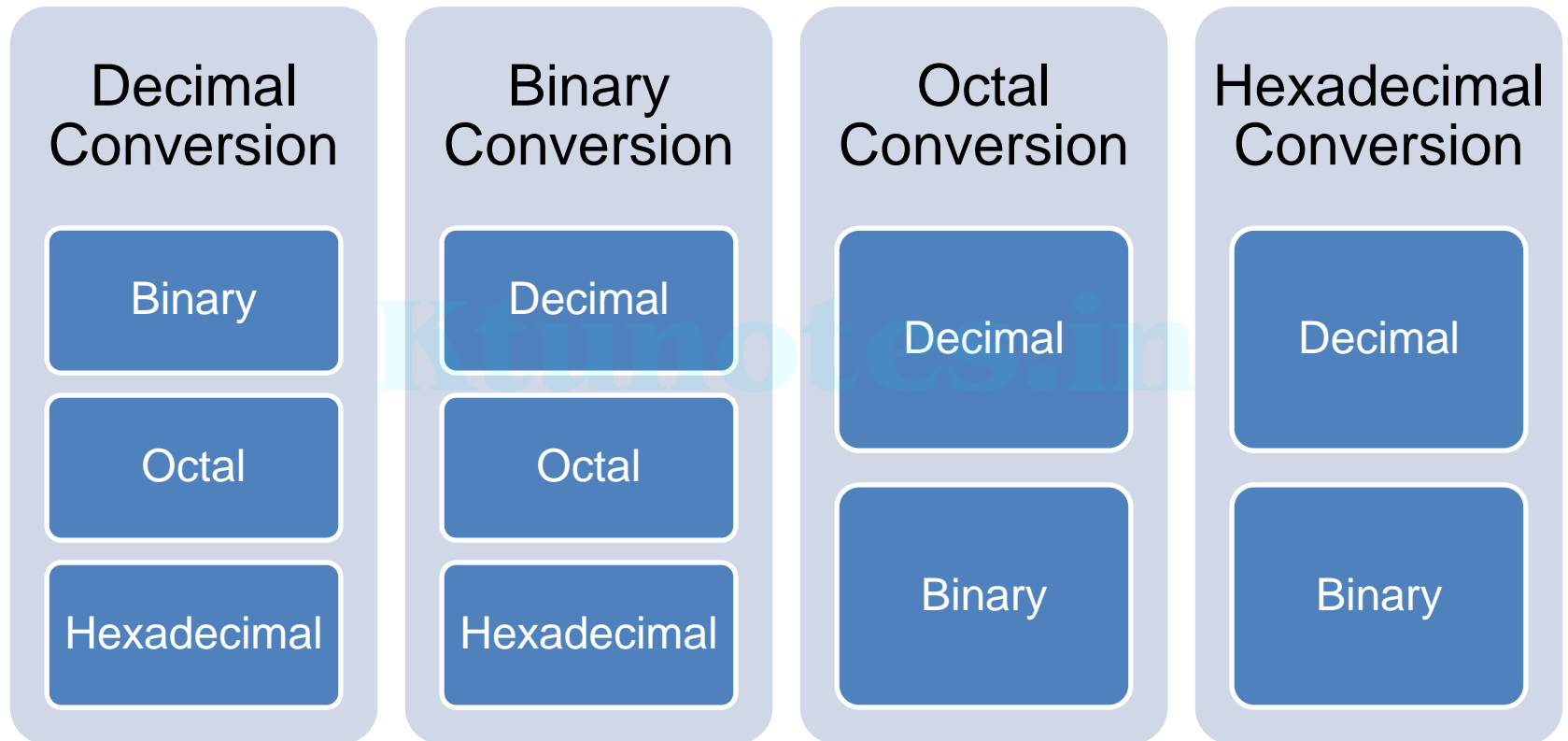
- Hexadecimal uses sixteen characters to represent numbers
- Numbers 0 through 9 and alphabetic characters A through F
- Large binary number can easily be converted to hexadecimal by grouping bits 4 at a time and writing equivalent hexadecimal character

**Example** Express  $1001\ 0110\ 0000\ 1110_2$  in hexadecimal:

**Solution** Group binary number by 4-bits starting from right. Thus, **960E**

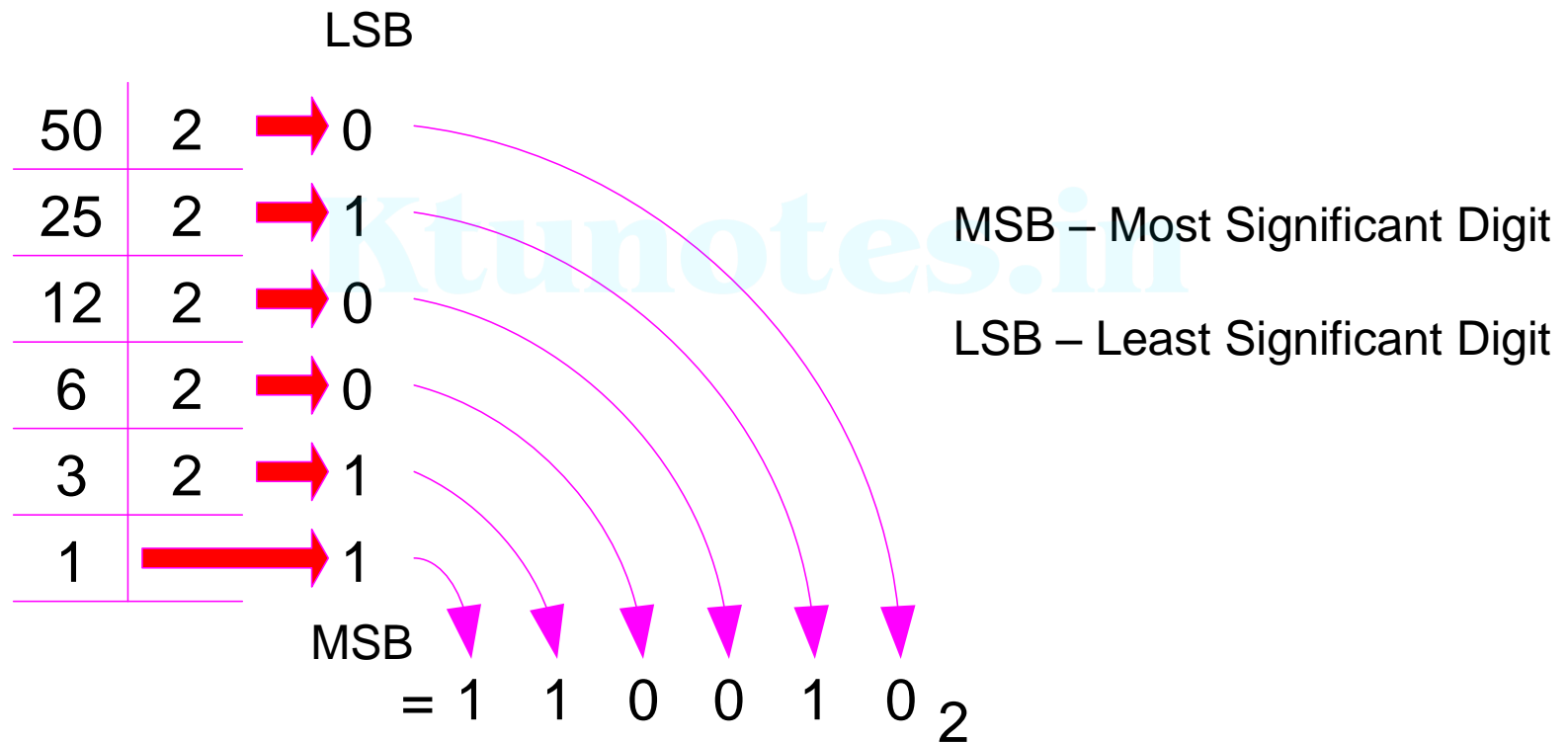


# CONVERSION BETWEEN NUMBER SYSTEMS



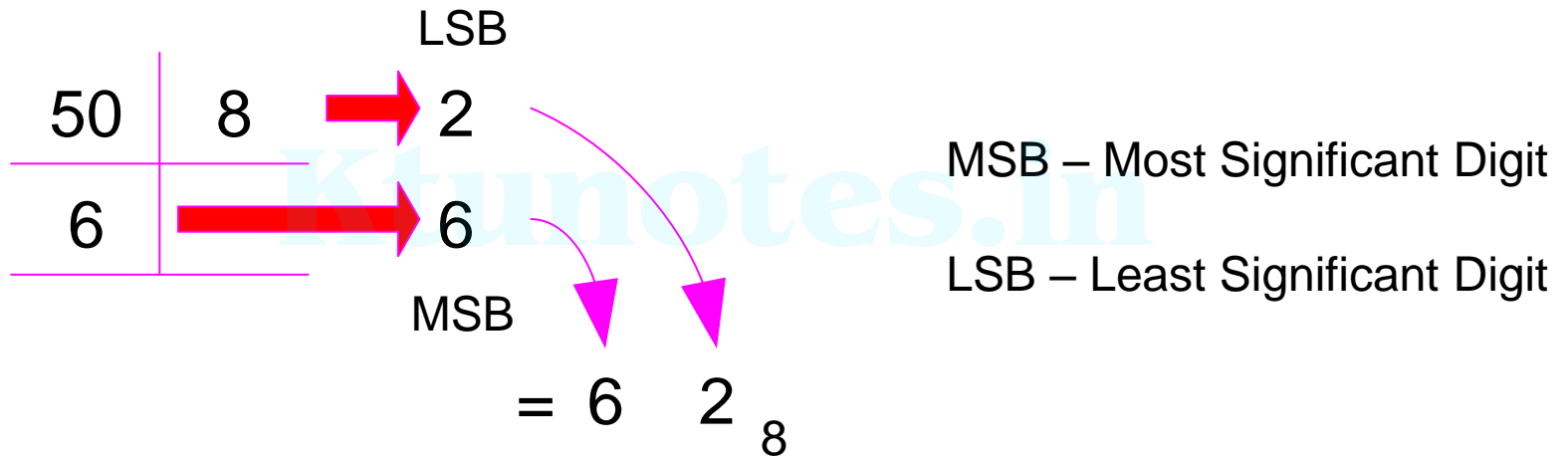
# DECIMAL TO BINARY

- Divide the decimal number by 2 until the quotient is 0.



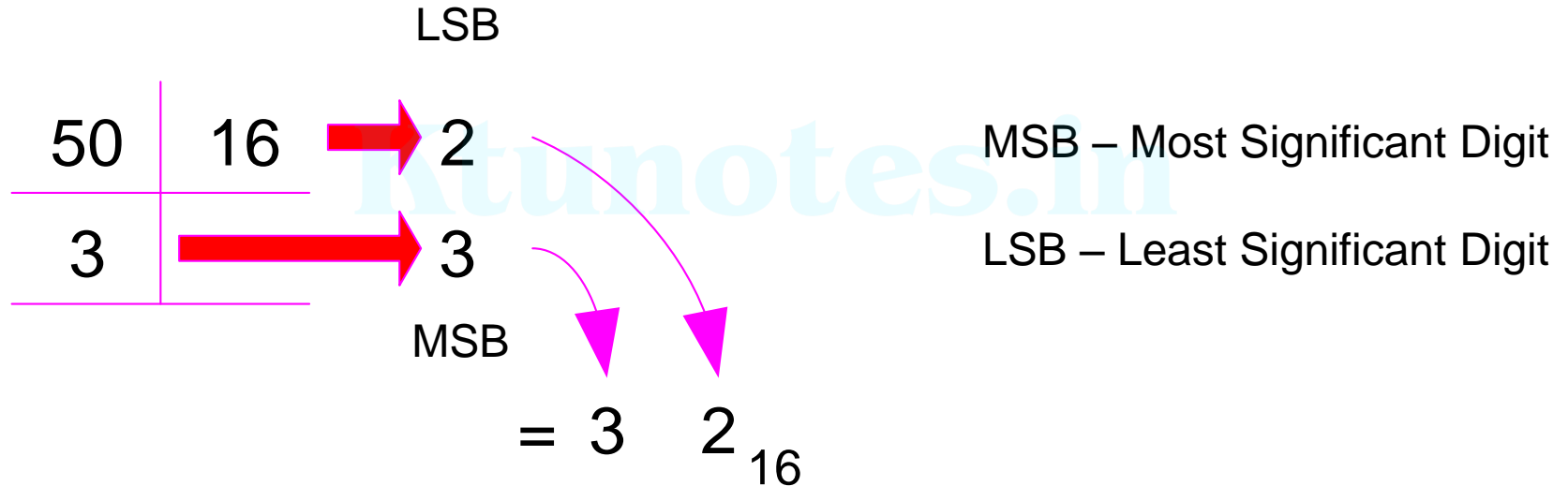
# DECIMAL TO OCTAL

- Divide the decimal number by 8 until the quotient is 0.



# DECIMAL TO HEXADECIMAL

- Divide the decimal number by 16 until the quotient is 0.





# 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 (*Fraction*) to Binary Conversion

**Example**

Convert the decimal fraction 0.188 to binary by repeatedly multiplying the fractional results by 2.

**Solution**

$0.188 \times 2 = 0.376$	carry = 0	MSB ↓
$0.376 \times 2 = 0.752$	carry = 0	
$0.752 \times 2 = 1.504$	carry = 1	
$0.504 \times 2 = 1.008$	carry = 1	
$0.008 \times 2 = 0.016$	carry = 0	

Answer = .00110 (for five significant digits)

# DECIMAL (*FRACTION*) TO OCTAL CONVERSION

Example:  $(0.3125)_{10}$

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

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

# BINARY TO DECIMAL

- Multiply each binary number by its weight and summing products

$$(11101)_2$$

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


$$= 16 + 8 + 4 + 1$$

$$= (29)_{10}$$



# BINARY TO OCTAL

- ✓ Grouped of three bits starting at the LSB
- ✓ Then convert each group to its octal equivalent

  $11111010_2 =$

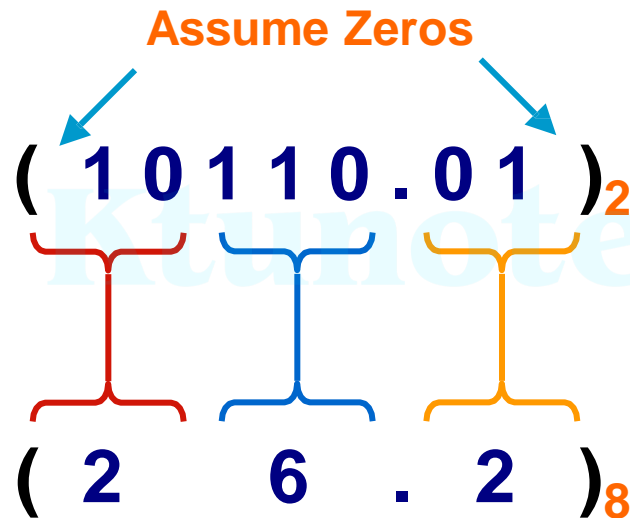
011		111		010	
3		7		2	

$= 372_8$

# BINARY TO OCTAL

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


Example:




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

# BINARY TO HEXADECIMAL

- Grouped of four bits starting at the LSB
- Then convert each group to its hexadecimal equivalent
- Zeros are added to make each group complete with 4 bits



  $1010101111_2 =$

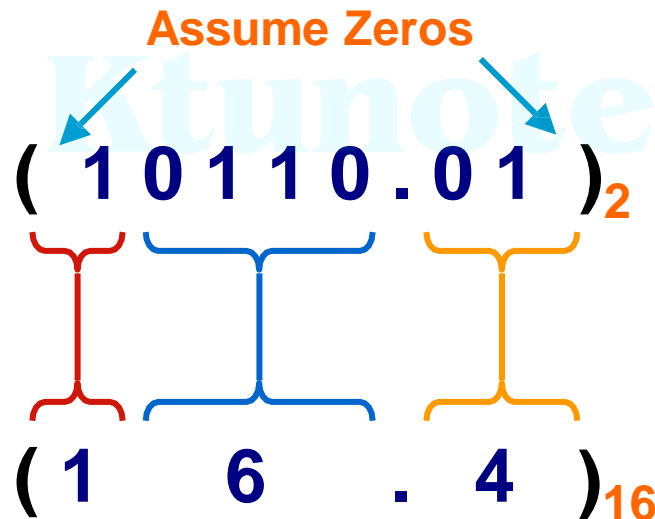
0010		1010		1111	
2		A		F	

$= 2AF_{16}$

# BINARY TO HEXADECIMAL

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

Example:



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

# OCTAL TO DECIMAL

- Multiply each octal number by its weight and summing products

$$\begin{aligned}(362)_8 \\&= (3 \times 8^2) + (6 \times 8^1) + (2 \times 8^0) \\&= 192 + 48 + 2 \\&= (242)_{10}\end{aligned}$$

# OCTAL TO BINARY

- Convert each octal digit to its three-bit binary equivalent

$$\begin{array}{ccc} 3 & 7 & 2 \\ 011 & 111 & 010 \end{array}$$

$= 011111010$

# HEXADECIMAL TO DECIMAL

- ✓ Multiply each hexadecimal number by its weight and summing products

$$\begin{aligned}(19B)_{16} &= (1 \times 16^2) + (9 \times 16^1) + (11 \times 16^0) \\ &= 256 + 144 + 11 \\ &= (411)_{10}\end{aligned}$$

# HEXADECIMAL TO BINARY

- ✓ Convert each hexadecimal digit to its four-bit binary equivalent

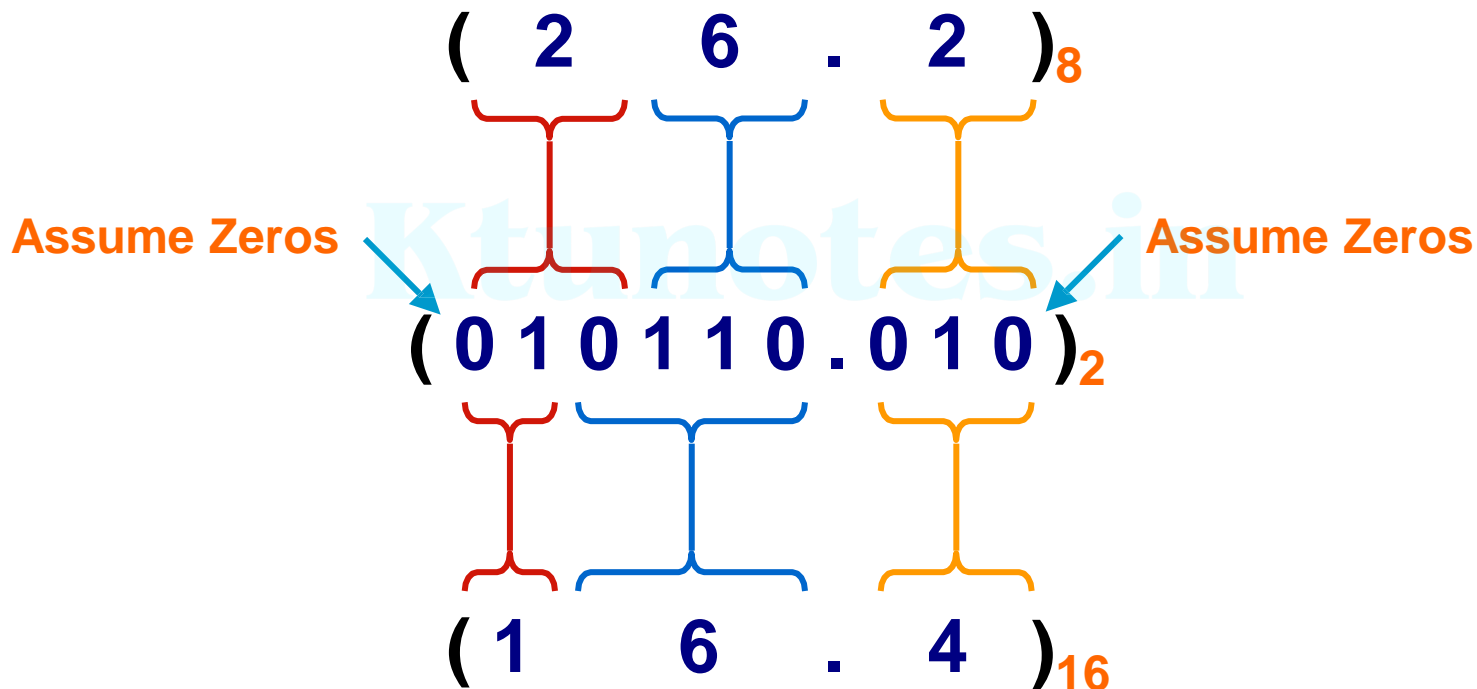
$$\begin{array}{ccccccc} 2AF_{16} & = & 2 & & A & & F \\ & & & & & & \\ & & 0010 & & 1010 & & 1111 \\ & & & & & & \\ & = & 1010101111_2 \end{array}$$



# OCTAL TO HEXADECIMAL

▣ Convert to **Binary** as an intermediate step

**Example:**



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

# BINARY ADDITION

The rules for binary addition are

$$0 + 0 = 0 \quad \text{Sum} = 0, \text{carry} = 0$$

$$0 + 1 = 1 \quad \text{Sum} = 1, \text{carry} = 0$$

$$1 + 0 = 1 \quad \text{Sum} = 1, \text{carry} = 0$$

$$1 + 1 = 10 \quad \text{Sum} = 0, \text{carry} = 1$$

When an input carry = 1 due to a previous result, the rules are

$$1 + 0 + 0 = 01 \quad \text{Sum} = 1, \text{carry} = 0$$

$$1 + 0 + 1 = 10 \quad \text{Sum} = 0, \text{carry} = 1$$

$$1 + 1 + 0 = 10 \quad \text{Sum} = 0, \text{carry} = 1$$

$$1 + 1 + 1 = 11 \quad \text{Sum} = 1, \text{carry} = 1$$

# BINARY ADDITION

**Example** Add the binary numbers 00111 and 10101 and show the equivalent decimal addition.

**Solution**

$$\begin{array}{r} \textcolor{red}{0\ 1\ 1\ 1} \\ 00111 \quad 7 \\ 10101 \quad 21 \\ \hline 11100 \textcolor{red}{=} 28 \end{array}$$

[Ktunotes.in](http://Ktunotes.in)

# BINARY ADDITION

$$\begin{array}{rcccccc} & 1 & 1 & 1 & 1 & 1 & 1 & & \\ & & 1 & 1 & 1 & 1 & 0 & 1 & = 61 \\ + & & 1 & 0 & 1 & 1 & 1 & & = 23 \\ \hline 1 & 0 & 1 & 0 & 1 & 0 & 0 & & = 84 \end{array}$$

# BINARY SUBTRACTON

The rules for binary subtraction are

$$0 - 0 = 0$$

$$1 - 1 = 0$$

$$1 - 0 = 1$$

$$10 - 1 = 1 \text{ with a borrow of } 1$$

**Example** Subtract the binary number 00111 from 10101 and show the equivalent decimal subtraction.

**Solution**

$$\begin{array}{r} \phantom{1}1\phantom{1}1 \\ \cancel{1}\cancel{0}\cancel{1}01 \quad 21 \\ \underline{00111} \quad \underline{7} \\ 01110 = 14 \end{array}$$

# BINARY SUBTRACTON

$$\begin{array}{r}
 \phantom{0}1 \phantom{00000}2 \\
 0 \cancel{2} 2 0 0 2 \quad \swarrow \\
 \cancel{1} 0 0 \cancel{1} \cancel{1} 0 1 \quad = 77 \\
 - \phantom{0} \phantom{0} 1 0 1 1 1 \quad = 23 \\
 \hline
 0 1 1 0 1 1 0 \quad = 54
 \end{array}$$

# BINARY MULTIPLICATION

- Multiplication follows the general principal of shift and add.
- The rules include:
  - $0 * 0 = 0$
  - $0 * 1 = 0$
  - $1 * 0 = 0$
  - $1 * 1 = 1$

# BINARY MULTIPLICATION

$$\begin{array}{r} \phantom{x} \phantom{00000} 1 \phantom{00} 0 \phantom{00} 1 \phantom{00} 1 \phantom{00} 1 \\ \phantom{x} \phantom{00000} 1 \phantom{00} 0 \phantom{00} 1 \phantom{00} 0 \\ \hline \phantom{x} 0 \phantom{00} 0 \phantom{00} 0 \phantom{00} 0 \phantom{00} 0 \\ \phantom{x} \phantom{00} 1 \phantom{00} 0 \phantom{00} 1 \phantom{00} 1 \phantom{00} 1 \\ \phantom{x} \phantom{00} 0 \phantom{00} 0 \phantom{00} 0 \phantom{00} 0 \phantom{00} 0 \\ \phantom{x} 1 \phantom{00} 0 \phantom{00} 1 \phantom{00} 1 \phantom{00} 1 \\ \hline 1 \phantom{00} 1 \phantom{00} 1 \phantom{00} 0 \phantom{00} 0 \phantom{00} 1 \phantom{00} 1 \phantom{00} 0 \end{array}$$



# BINARY DIVISION

- Binary division is similar to decimal division
- It is called as the long division procedure

The diagram illustrates the long division procedure for binary and decimal numbers, showing their similarity.

**Binary Division:**

$$\begin{array}{r} 101 \\ 101 \overline{) 11010} \\ \underline{-101} \phantom{0} \\ 11 \phantom{0} \\ \underline{-00} \phantom{0} \\ 110 \\ \underline{-101} \\ 1 \end{array}$$

Quotient: 101  
Remainder: 1

**Decimal Division:**

$$\begin{array}{r} 5 \\ 5 \overline{) 26} \\ \underline{-25} \\ 1 \end{array}$$

Quotient: 5  
Remainder: 1

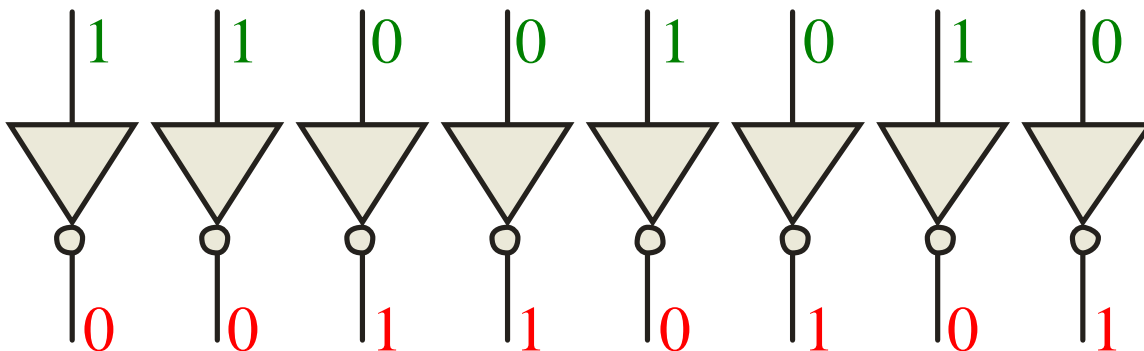
Arrows indicate the correspondence between the binary and decimal results: the quotient 101 (binary) corresponds to 5 (decimal), and the remainder 1 (binary) corresponds to 1 (decimal).

# 1'S COMPLEMENT

- The 1's complement of a binary number is just the inverse of the digits
- To form the 1's complement, change all 0's to 1's and all 1's to 0's

For example, the 1's complement of **11001010** is **00110101**

In digital circuits, the 1's complement is formed by using inverters:



# 2'S COMPLEMENT

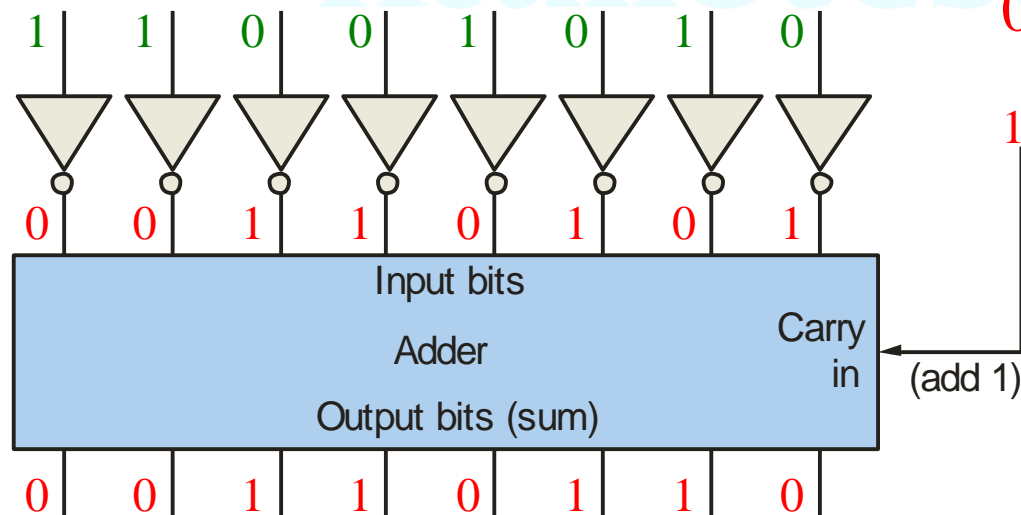
- The 2's complement of a binary number is found by adding 1 to the LSB of 1's complement

Recall that the 1's complement of **11001010** is

**00110101** (1's complement)

To form the 2's complement, add 1:

$$\begin{array}{r} 00110101 \\ +1 \\ \hline 00110110 \end{array}$$
  
(2's complement)



# SIGNED BINARY NUMBERS

- MSB in a signed number is the sign bit, that tells you if number is positive or negative
- Computers use a modified 2's complement for signed numbers
- Positive numbers are stored in *true* form (with a 0 for sign bit) and negative numbers are stored in *complement* form (with a 1 for sign bit)

For example, the positive number 58 is written using 8-bits as

00111010 (true form).

Sign bit      Magnitude bits

# SIGNED BINARY NUMBERS

- Negative numbers are written as the 2's complement of corresponding positive number

The negative number -58 is written as:

1's complement of +58 = 11000101

2's complement of +58 = 11000101 + 1 = 11000110

-58 = 11000110 (complement form)

Sign bit

Magnitude bits

An easy way to read a signed number that uses this notation is to assign the sign bit a column weight of -128 (for an 8-bit number).

Then add the column weights for the 1's.

**Example  
Solution**

Assuming that the sign bit = -128, show that 11000110 = -58 as a 2's complement signed number:

Column weights: -128 64 32 16 8 4 2 1.

1	1	0	0	0	1	1	0
-128	+64				+4	+2	
							= -58

# ARITHMETIC OPERATIONS WITH SIGNED NUMBERS

Rules for **addition**: Add the two signed numbers. Discard any final carries. The result is in signed form.

Examples:

$$00011110 = +30$$

$$00001111 = +15$$

---

$$00101101 = +45$$

$$00001110 = +14$$

$$11101111 = -17$$

---

$$11111101 = -3$$

$$11111111 = -1$$

$$11111000 = -8$$

---

$$11110111 = -9$$

Discard carry

# ARITHMETIC OPERATIONS WITH SIGNED NUMBERS

- Note that if the number of bits required for the answer is exceeded, overflow will occur
- This occurs only if both numbers have the same sign. The overflow will be indicated by an incorrect sign bit.

Two examples are:

$$\begin{array}{r} 10000001 = -127 \\ 10000001 = -127 \\ \hline 100000010 = +2 \end{array}$$

Discard carry →

**Wrong!** The answer is incorrect and the sign bit has changed.

# ARITHMETIC OPERATIONS WITH SIGNED NUMBERS

- Rules for **subtraction**: 2's complement subtrahend and add numbers
- Discard any final carries, result is in signed form

Repeat the examples done previously, but subtract:

$$\begin{array}{rcl}
 00011110 & (+30) & 00001110 & (+14) & 11111111 & (-1) \\
 - 00001111 & -(+15) & - 11101111 & -(-17) & - 11111000 & -(-8) \\
 \hline
 \end{array}$$

2's complement subtrahend and add:

$$\begin{array}{rcl}
 00011110 = +30 & 00001110 = +14 & 11111111 = -1 \\
 11110001 = -15 & 00010001 = +17 & 00001000 = +8 \\
 \hline
 \cancel{1}00001111 = +15 & 00011111 = +31 & \cancel{1}00000111 = +7
 \end{array}$$

Discard carry

Discard carry



# Hexadecimal Addition

- Start with least significant hexadecimal digits
- Let Sum= summation of two hex digits
- If sum is greater than or equal to 16, then

Sum=Sum-16 and carry 1

carry				1	1		1	
	9	C	3	7	2	8	6	5
+	1	3	9	5	E	8	4	B
<hr/>								
	A	F	C	D	1	0	B	0

$5 + B = 5 + 11 = 16$   
Since  $\text{Sum} \geq 16$   
 $\text{Sum} = 16 - 16 = 0$   
Carry = 1

# Hexadecimal Subtraction

- Start with least significant hexadecimal digits
- Let Difference=subtraction of two hex digits
- If Difference is negative, then

Difference=16+Difference and borrow=-1

borrow		-1		-1			-1	
	9	C	3	7	2	8	6	5
-	1	3	9	5	E	8	4	B
<hr/>								
	8	8	A	1	4	0	1	A

Since  $5 < B$ , Difference  $< 0$   
Difference =  $16+5-11 = 10$   
Borrow = -1

# Octal Addition

- Start with least significant octal digits
- Let Sum= summation of two octal digits
- If sum is greater than or equal to 8, then

Sum=Sum-8 and carry 1

$$\begin{array}{r} \text{1 1} \\ 6437_8 \\ + 2510_8 \\ \hline 99 \\ - 88 \quad \text{(subtract Base (8))} \\ \hline 11147_8 \end{array}$$

# Octal Subtraction

- Start with least significant octal digits
- Let Difference=subtraction of two octal digits
- If Difference is negative, then

Difference=8+Difference and borrow=-1

$$\begin{array}{r}
 \phantom{00}8 \\
 \phantom{00}008 \\
 \phantom{00}\diagup \diagup \\
 11147_8 \\
 \hline
 \phantom{00}89 \\
 - \phantom{00}6437_8 \\
 \hline
 \phantom{00}2510_8
 \end{array}$$

# 9's Complement

- 9's complement of a decimal number is found by subtracting each digit in number from 9
- 9's complement of each of decimal digits is as follows

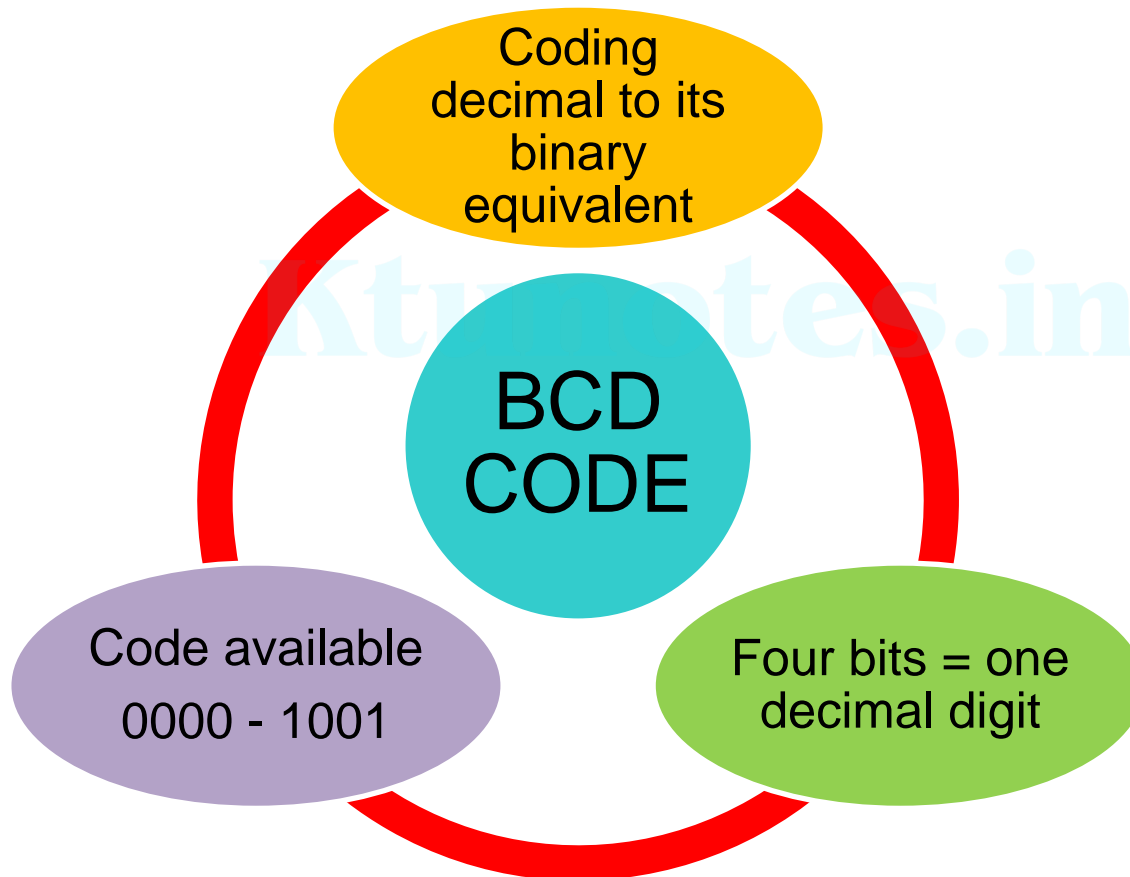
Digit	9's complement
0	9
1	8
2	7
3	6
4	5
5	4
6	3
7	2
8	1
9	0

# 10's Complement

- 10's complement of a decimal number can be found by adding 1 to 9's complement of that decimal number
- *10's complement = 9's complement + 1*
- Example: let us take a decimal number 456  
9's complement of this number will be  $999 - 456$   
which will be 543  
Now 10s complement will be  $543 + 1 = 544$

# BCD CODE

- Binary coded decimal (BCD) is a weighted code



# BCD code

- commonly known as a weighted 8421 BCD code, with 8, 4, 2 and 1 representing weights of different bits starting from most significant bit (MSB) and proceeding towards least significant bit (LSB)
- It is commonly used in digital systems when it is necessary to show decimal numbers such as in clock displays.

The table illustrates the difference between straight binary and BCD. BCD represents each decimal digit with a 4-bit code. Notice that the codes 1010 through 1111 are not used in BCD.

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

You can think of BCD in terms of column weights in groups of four bits. For an 8-bit BCD number, the column weights are: 80 40 20 10 8 4 2 1.

**Question:** What are the column weights for the BCD number 1000 0011 0101 1001?

**Answer:**

8000 4000 2000 1000 800 400 200 100 80 40 20 10 8 4 2 1

Note that you could add the column weights where there is a 1 to obtain the decimal number. For this case:

$$8000 + 200 + 100 + 40 + 10 + 8 + 1 = 8359_{10}$$

# BCD Addition

- Use binary arithmetic to add BCD digit
- If binary sum is less than or equal to 1010 (9 in decimal), corresponding BCD sum digit is correct
- If binary sum is more than 1010, must add 0110 (6 in decimal) to corresponding BCD sum digit in order to produce correct carry into digit to left

$\begin{array}{r} 8 \\ + 5 \\ \hline 13 \end{array}$	$\begin{array}{r} 1000 \\ + 0101 \\ \hline 1101 \\ + 0110 \\ \hline 1\ 0011 \\ 0001\ 0011 \end{array}$	$\begin{array}{r} \text{Eight} \\ \text{Plus 5} \\ \hline \text{is 13 (>9)} \\ \text{so add 6} \\ \hline \text{3 and carry} \\ \text{Final answer in BCD} \end{array}$
--	--	--

←

# BCD Addition

- Add 2905+1897 in BCD

	carry	+1	+1	+1	
+	0010	1001	0000	0101	
	0001	1000	1001	0111	
<hr/>					
	0100	10010	1010	1100	
digit correction		0110	0110	0110	
<hr/>					
	0100	1000	0000	0010	

- Final answer=4802

# BCD Subtraction Using 9's Complement

- BCD Subtraction using 9s Complement can be used to perform subtraction by adding minuend to 9s Complement of subtrahend
- In 9s Complement subtraction when 9s Complement of smaller number is added to larger number carry is generated, It is necessary to add this carry to result (this is called an end-around carry)
- When larger number is subtracted from smaller one, there is no carry, and result is in 9s Complement form and negative

# BCD Subtraction

## Regular Subtraction

(a)

$$\begin{array}{r} 8 \\ - 2 \\ \hline 6 \end{array}$$

(b)

$$\begin{array}{r} 28 \\ - 13 \\ \hline 15 \end{array}$$

(c)

$$\begin{array}{r} 18 \\ - 24 \\ \hline -6 \end{array}$$

## 9's Complement Subtraction

$$\begin{array}{r} 8 \\ + 7 \quad \text{9's complement of 2} \\ \hline 15 \\ \text{①} \downarrow + 1 \quad \text{Add carry to result} \\ \hline 6 \end{array}$$

$$\begin{array}{r} 28 \\ + 86 \quad \text{9's complement of 13} \\ \hline 114 \\ \text{①} \downarrow + 1 \quad \text{Add carry to the result} \\ \hline 115 \end{array}$$

$$\begin{array}{r} 18 \\ + 75 \quad \text{9's complement of 24} \\ \hline 93 \\ \downarrow \quad \text{9's complement of result} \\ \text{(No carry indicates that the} \\ \text{answer is negative and in} \\ \text{complement form)} \\ \hline -06 \end{array}$$

# BCD Subtraction Using 10's Complement

- BCD Subtraction using 10s Complement can be used to perform subtraction by adding minuend to 10s Complement of subtrahend and dropping carry

	<u>Regular Subtraction</u>	<u>10's Complement Subtraction</u>
(a)	$\begin{array}{r} 8 \\ - 2 \\ \hline 6 \end{array}$	$\begin{array}{r} 8 \\ + 8 \\ \hline \cancel{1}6 \end{array}$ <p>10's complement of 2 Drop carry</p>
(b)	$\begin{array}{r} 28 \\ - 13 \\ \hline 15 \end{array}$	$\begin{array}{r} 28 \\ + 87 \\ \hline \cancel{1}15 \end{array}$ <p>10's complement of 13 Drop carry</p>
(c)	$\begin{array}{r} 18 \\ - 24 \\ \hline -6 \end{array}$	$\begin{array}{r} 18 \\ + 76 \\ \hline 94 \\ \downarrow \\ -06 \end{array}$ <p>10's complement of 24 10's complement of result (No carry indicates that the answer is negative and in the 10's complement form)</p>

# Character Coding Schemes

- In computing, a single character such as a letter, a number or a symbol is represented by a group of bits
- Number of bits per character depends on *coding* scheme used
- Most common coding schemes are:
  1. Binary Coded Decimal (BCD)
  2. Extended Binary Coded Decimal Interchange Code (EBCDIC)
  3. American Standard Code for Information Interchange (ASCII)

# ASCII CODE

- ASCII (American Standard Code for Information Interchange) is a code for alphanumeric characters and control characters
- Every time a character is typed on a keyboard a code number is transmitted to computer
- Code numbers are stored in binary on computers as Character Sets called ASCII
- Table below shows a version of ASCII that uses 7 bits to code each character
- 128 different characters i.e.  $2^7$  can be represented using ASCII



# ASCII CODE

- Biggest number that can be held in 7-bits is 1111111 in binary (**127 in decimal**)
- More than enough to cover all of characters on a standard English-Language keyboard
- First 32 characters are control characters
- In 1981, IBM introduced extended ASCII, which is an 8-bit code and increased the character set to 256
- Characters from 0 to 31 are control characters, 32 to 64 special characters, 65 to 96 uppercase letters and few symbols, 97 to 127 lowercase letters and other symbols and 128 to 255 are other symbols

# ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

# ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]	48	30	110000	60	0	96	60	1100000	140	`
1	1	1	1	[START OF HEADING]	49	31	110001	61	1	97	61	1100001	141	a
2	2	10	2	[START OF TEXT]	50	32	110010	62	2	98	62	1100010	142	b
3	3	11	3	[END OF TEXT]	51	33	110011	63	3	99	63	1100011	143	c
4	4	100	4	[END OF TRANSMISSION]	52	34	110100	64	4	100	64	1100100	144	d
5	5	101	5	[ENQUIRY]	53	35	110101	65	5	101	65	1100101	145	e
6	6	110	6	[ACKNOWLEDGE]	54	36	110110	66	6	102	66	1100110	146	f
7	7	111	7	[BELL]	55	37	110111	67	7	103	67	1100111	147	g
8	8	1000	10	[BACKSPACE]	56	38	111000	70	8	104	68	1101000	150	h
9	9	1001	11	[HORIZONTAL TAB]	57	39	111001	71	9	105	69	1101001	151	i
10	A	1010	12	[LINE FEED]	58	3A	111010	72	:	106	6A	1101010	152	j
11	B	1011	13	[VERTICAL TAB]	59	3B	111011	73	;	107	6B	1101011	153	k
12	C	1100	14	[FORM FEED]	60	3C	111100	74	<	108	6C	1101100	154	l
13	D	1101	15	[CARRIAGE RETURN]	61	3D	111101	75	=	109	6D	1101101	155	m
14	E	1110	16	[SHIFT OUT]	62	3E	111110	76	>	110	6E	1101110	156	n
15	F	1111	17	[SHIFT IN]	63	3F	111111	77	?	111	6F	1101111	157	o
16	10	10000	20	[DATA LINK ESCAPE]	64	40	1000000	100	@	112	70	1110000	160	p
17	11	10001	21	[DEVICE CONTROL 1]	65	41	1000001	101	A	113	71	1110001	161	q
18	12	10010	22	[DEVICE CONTROL 2]	66	42	1000010	102	B	114	72	1110010	162	r
19	13	10011	23	[DEVICE CONTROL 3]	67	43	1000011	103	C	115	73	1110011	163	s
20	14	10100	24	[DEVICE CONTROL 4]	68	44	1000100	104	D	116	74	1110100	164	t
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]	69	45	1000101	105	E	117	75	1110101	165	u
22	16	10110	26	[SYNCHRONOUS IDLE]	70	46	1000110	106	F	118	76	1110110	166	v
23	17	10111	27	[ENG OF TRANS. BLOCK]	71	47	1000111	107	G	119	77	1110111	167	w
24	18	11000	30	[CANCEL]	72	48	1001000	110	H	120	78	1111000	170	x
25	19	11001	31	[END OF MEDIUM]	73	49	1001001	111	I	121	79	1111001	171	y
26	1A	11010	32	[SUBSTITUTE]	74	4A	1001010	112	J	122	7A	1111010	172	z
27	1B	11011	33	[ESCAPE]	75	4B	1001011	113	K	123	7B	1111011	173	{
28	1C	11100	34	[FILE SEPARATOR]	76	4C	1001100	114	L	124	7C	1111100	174	
29	1D	11101	35	[GROUP SEPARATOR]	77	4D	1001101	115	M	125	7D	1111101	175	}
30	1E	11110	36	[RECORD SEPARATOR]	78	4E	1001110	116	N	126	7E	1111110	176	~
31	1F	11111	37	[UNIT SEPARATOR]	79	4F	1001111	117	O	127	7F	1111111	177	[DEL]
32	20	100000	40	[SPACE]	80	50	1010000	120	P					
33	21	100001	41	!	81	51	1010001	121	Q					
34	22	100010	42	"	82	52	1010010	122	R					
35	23	100011	43	#	83	53	1010011	123	S					
36	24	100100	44	\$	84	54	1010100	124	T					
37	25	100101	45	%	85	55	1010101	125	U					
38	26	100110	46	&	86	56	1010110	126	V					
39	27	100111	47	'	87	57	1010111	127	W					
40	28	101000	50	(	88	58	1011000	130	X					
41	29	101001	51	)	89	59	1011001	131	Y					
42	2A	101010	52	*	90	5A	1011010	132	Z					
43	2B	101011	53	+	91	5B	1011011	133	[					
44	2C	101100	54	,	92	5C	1011100	134	\					
45	2D	101101	55	-	93	5D	1011101	135	]					
46	2E	101110	56	.	94	5E	1011110	136	^					
47	2F	101111	57	/	95	5F	1011111	137	_					

# EXTENDED ASCII TABLE

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	Ł	224	E0	α
129	81	ü	161	A1	í	193	C1	ł	225	E1	β
130	82	é	162	A2	ó	194	C2	ṽ	226	E2	Γ
131	83	â	163	A3	û	195	C3	ṭ	227	E3	π
132	84	ä	164	A4	ñ	196	C4	—	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	†	229	E5	σ
134	86	å	166	A6	*	198	C6	‡	230	E6	μ
135	87	ç	167	A7	°	199	C7	‡	231	E7	ι
136	88	ê	168	A8	¿	200	C8	Ł	232	E8	Φ
137	89	ë	169	A9	ƒ	201	C9	ƒ	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	Ł	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	ƒ	235	EB	δ
140	8C	î	172	AC	¼	204	CC	‡	236	EC	∞
141	8D	ì	173	AD	ı	205	CD	=	237	ED	∞
142	8E	Ā	174	AE	«	206	CE	‡	238	EE	ε
143	8F	Ă	175	AF	»	207	CF	Ł	239	EF	Π
144	90	É	176	B0	☐	208	D0	Ł	240	FO	≡
145	91	æ	177	B1	☐	209	D1	ƒ	241	F1	±
146	92	Æ	178	B2	☐	210	D2	π	242	F2	≥
147	93	ô	179	B3		211	D3	Ł	243	F3	≤
148	94	ö	180	B4	†	212	D4	Ł	244	F4	[
149	95	ò	181	B5	†	213	D5	ƒ	245	F5	]
150	96	û	182	B6	‡	214	D6	π	246	F6	÷
151	97	ù	183	B7	π	215	D7	‡	247	F7	≈
152	98	ÿ	184	B8	ƒ	216	D8	†	248	F8	*
153	99	Ö	185	B9	‡	217	D9	ƒ	249	F9	*
154	9A	Ü	186	BA		218	DA	ƒ	250	FA	.
155	9B	◊	187	BB	π	219	DB	■	251	FB	√
156	9C	£	188	BC	‡	220	DC	■	252	FC	∞
157	9D	¥	189	BD	‡	221	DD	■	253	FD	∞
158	9E	ℳ	190	BE	‡	222	DE	■	254	FE	■
159	9F	f	191	BF	ƒ	223	DF	■	255	FF	□

# EBCDIC CODE

- It is pronounced as “ebb-see-dick
- Extended Binary Coded Decimal Interchange code (EBCDIC) is an 8-bit character-coding scheme used primarily on IBM computers
- A total of 256 ( $2^8$ ) characters can be coded using this scheme. For example, symbolic representation of letter A using Extended Binary Coded Decimal Interchange code is  $11000001_2$

# EBCDIC CODE

- In EBCDIC code eight bits for each character is divided into two four-bit zones
- One zone indicating type of character, digit, punctuation mark, lowercase letter, capital letter, and so on, and other zone indicating value (that is, specific character within this type)
- Below is given some characters of **EBCDIC** code to get familiar with it



# EBCDIC CODE TABLE

Char	EBCDIC	HEX	Char	EBCDIC	HEX	Char	EBCDIC	HEX
A	1100 0001	C1	P	1101 0111	D7	4	1111 0100	F4
B	1100 0010	C2	Q	1101 1000	D8	5	1111 0101	F5
C	1100 0011	C3	R	1101 1001	D9	6	1111 0110	F6
D	1100 0100	C4	S	1110 0010	E2	7	1111 0111	F7
E	1100 0101	C5	T	1110 0011	E3	8	1111 1000	F8
F	1100 0110	C6	U	1110 0100	E4	9	1111 1001	F9
G	1100 0111	C7	V	1110 0101	E5	blank	...	...
H	1100 1000	C8	W	1110 0110	E6	.	...	...
I	1100 1001	C9	X	1110 0111	E7	(	...	...
J	1101 0001	D1	Y	1110 1000	E8	+	...	...
K	1101 0010	D2	Z	1110 1001	E9	\$	...	...
L	1101 0011	D3	0	1111 0000	F0	*	...	...
M	1101 0100	D4	1	1111 0001	F1	)	...	...
N	1101 0101	D5	2	1111 0010	F2	-	...	...
O	1101 0110	D6	3	1111 0011	F3	/		

# BINARY CODES

- A **binary code** represents text, computer processor instructions, or any other data using a two-symbol system
- Two-symbol system used is often "0" and "1" from binary number system
- For example, a binary string of eight bits can represent any of 256 possible values and can, therefore, represent a wide variety of different items
- Following are some binary codes
  1. Decimal codes
  2. Error detection codes
  3. Reflected code



# ERROR DETECTION CODES

- These are used to detect errors present in received data bitstream
- These codes contain some bits, which are appended to original bit stream
- These codes detect error, if it is occurred during transmission of original data bit stream
- **Example** – Parity code, Checksum, Cyclic Redundancy Check

# PARITY CODE

- It is easy to *append* one parity bit either to left of MSB or to right of LSB of original bit stream
- There are two types of parity codes, namely even parity code and odd parity code based on type of parity being chosen
- Parity bit helps to check if any error occurred in data during transmission.

Ktunotes.in

# EVEN PARITY CODE

- Value of even parity bit should be zero, if even number of ones present in the binary code. Otherwise, it should be one
- So that, even number of ones present in **even parity code**
- Even parity code contains data bits and even parity bit
- Following table shows **even parity codes** corresponding to each 3-bit binary code
- Here, even parity bit is included to right of LSB of binary code

# EVEN PARITY CODE

Binary Code	Even Parity bit	Even Parity Code
000	0	0000
001	1	0011
010	1	0101
011	0	0110
100	1	1001
101	0	1010
110	0	1100
111	1	1111

# ODD PARITY CODE

- Value of odd parity bit should be zero, if odd number of ones present in binary code. Otherwise, it should be one. So that, odd number of ones present in **odd parity code**
- Odd parity code contains data bits and odd parity bit
- Following table shows **odd parity codes** corresponding to each 3-bit binary code
- Here, odd parity bit is included to right of LSB of binary code

# ODD PARITY CODE

Binary Code	Odd Parity bit	Odd Parity Code
000	1	0001
001	0	0010
010	0	0100
011	1	0111
100	0	1000
101	1	1011
110	1	1101
111	0	1110

# PARITY CHECKING STEPS

Error detection using single parity check involves following steps

## Step-01:

At sender side,

- Total number of 1's in data unit to be transmitted is counted.
- Total number of 1's in the data unit is made even in case of even parity.
- Total number of 1's in the data unit is made odd in case of odd parity.
- This is done by adding an extra bit called as **parity bit**.

# PARITY CHECKING STEPS

## Step-02:

- Newly formed code word (Original data + parity bit) is transmitted to receiver

## Step-03:

Ktunotes.in

At receiver side,

- Receiver receives transmitted code word
- Total number of 1's in the received code word is counted



# PARITY CHECKING STEPS

Then, following cases are possible

- If total number of 1's is even and even parity is used, then receiver assumes that no error occurred.
- If total number of 1's is even and odd parity is used, then receiver assumes that error occurred.
- If total number of 1's is odd and odd parity is used, then receiver assumes that no error occurred.
- If total number of 1's is odd and even parity is used, then receiver assumes that error occurred.

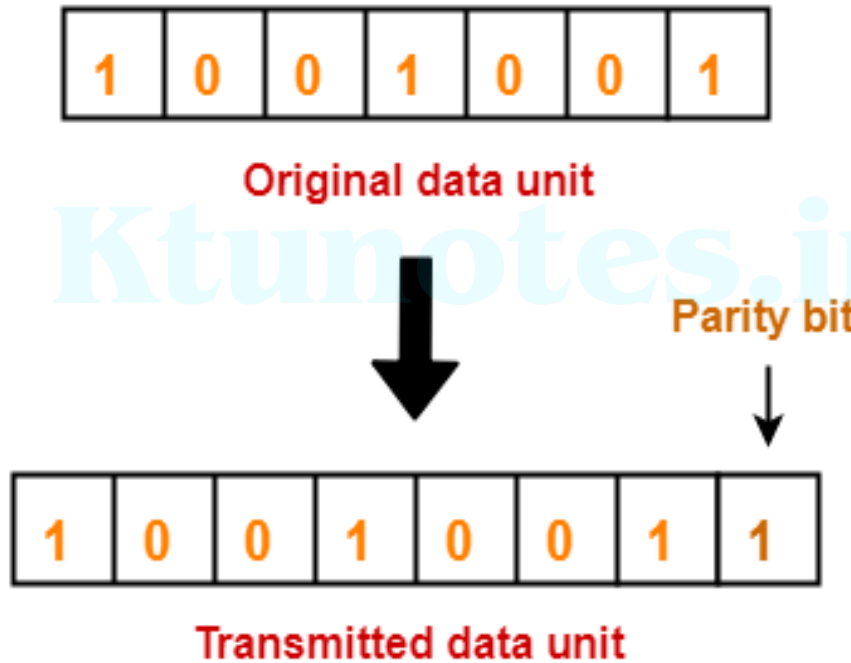
# PARITY CHECK EXAMPLE

- Consider the data unit to be transmitted is 1001001 and even parity is used

## **At Sender Side-**

- Total number of 1's in the data unit is counted.
- Total number of 1's in the data unit = 3.
- Clearly, even parity is used and total number of 1's is odd.
- So, parity bit = 1 is added to the data unit to make total number of 1's even.
- Then, the code word 10010011 is transmitted to the receiver.

# PARITY CHECK EXAMPLE



# PARITY CHECK EXAMPLE

## At Receiver Side-

- After receiving the code word, total number of 1's in the code word is counted.
- Consider receiver receives the correct code word = 10010011.
- Even parity is used and total number of 1's is even.
- So, receiver assumes that no error occurred in the data during the transmission.

# PARITY CHECK EXAMPLE

## **Advantage-**

- This technique is guaranteed to detect an odd number of bit errors (one, three, five and so on).
- If odd number of bits flip during transmission, then receiver can detect by counting the number of 1's.

## **Limitation-**

- This technique can not detect an even number of bit errors (two, four, six and so on).
- If even number of bits flip during transmission, then receiver can not catch the error.

# PARITY CHECK EXAMPLE

- Consider data unit to be transmitted is 10010001 and even parity is used.
- Then, code word transmitted to the receiver = 100100011
- Consider during transmission, code word modifies as 101100111.  
(2 bits flip)
- On receiving the modified code word, receiver finds the number of 1's is even and even parity is used.
- So, receiver assumes that no error occurred in the data during transmission though the data is corrupted.

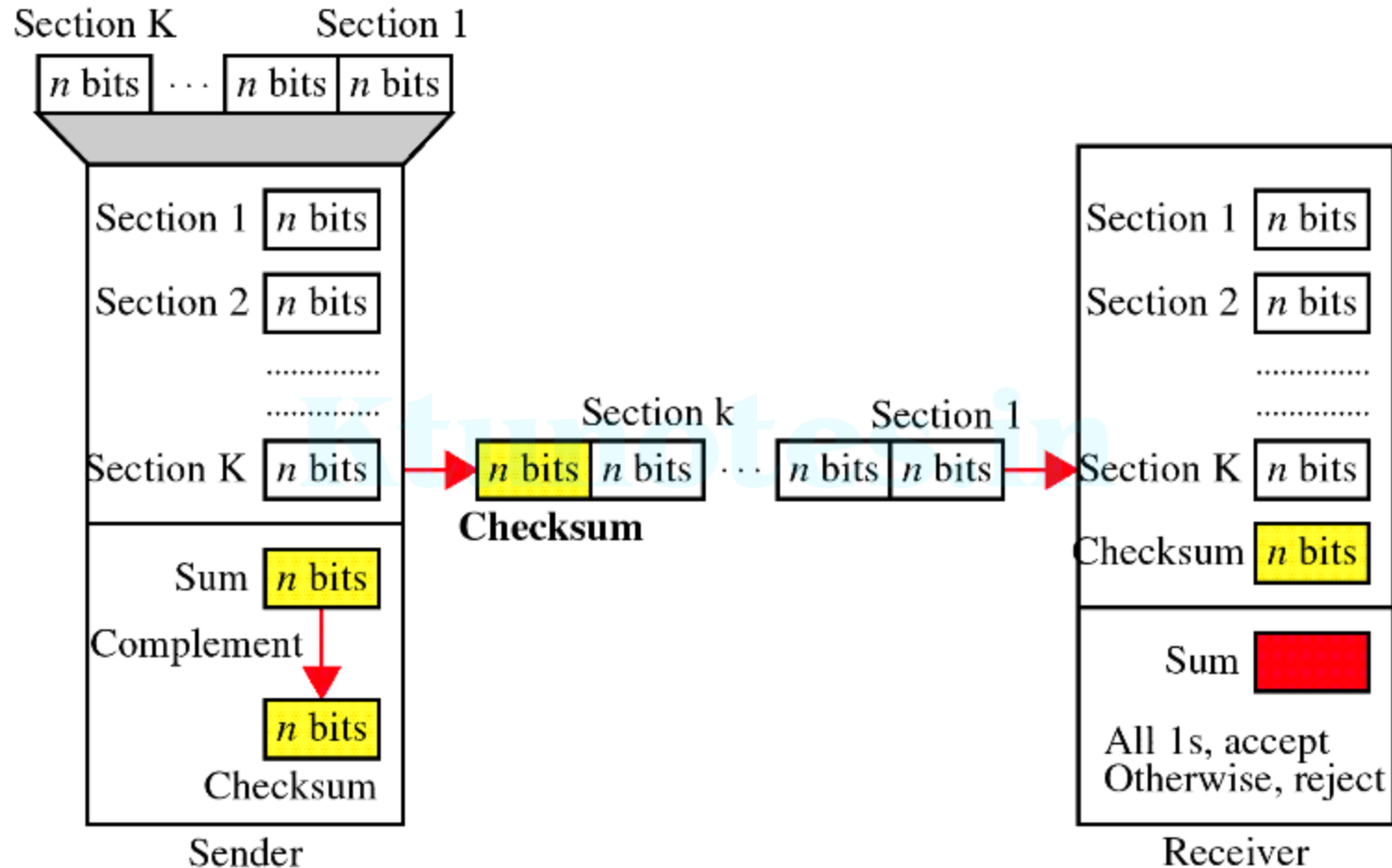
# CHECKSUM

- It also uses same method of parity bits transmitter send data followed by bits known as checksum bits

## **Algorithm:-**

1. In checksum error detection scheme, data is divided into  $k$  segments each of  $m$  bits.
2. In sender end segments are added using complement arithmetic to get the sum, sum is complemented to get Checksum.
3. Checksum segment is sent along with data segments.
4. At receiver end, all received segments are added using 1s complement arithmetic to get sum. Sum is complemented
5. If result is zero, received data is accepted; otherwise discarded

# CHECKSUM





# CHECKSUM EXAMPLE

Original Data

10011001	11100010	00100100	10000100
----------	----------	----------	----------

1

2

3

4

k=4, m=8

Sender

```

1  10011001
2  11100010
   -----
   ①01111011
     1
   -----
    01111100
3  00100100
   -----
    10100000
4  10000100
   -----
   ①00100100
     1
   -----
Sum: 00100101
Checksum: 11011010
    
```

Receiver

```

1  10011001
2  11100010
   -----
   ①01111011
     1
   -----
    01111100
3  00100100
   -----
    10100000
4  10000100
   -----
   ①00100100
     1
   -----
    00100101
    11011010
   -----
Sum: 11111111
Complement: 00000000
Conclusion: Accept Data
    
```

# GRAY CODE/REFLECTED CODE

- Reflected binary code or Gray code is an ordering of binary numeral system such that two successive values differ in only one bit (binary digit)
- Gray code is not weighted that means it does not depends on positional value of digit
- Gray code also known as reflected binary code, because the first  $(n/2)$  values compare with those of the last  $(n/2)$  values, but in reverse order.

# GRAY CODE/REFLECTED CODE

Decimal	Binary Code (input)	Gray Code (output)
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

# GRAY CODE

Consider gray code sequence

0000  
0001  
0011  
0010  
0110  
0111  
0101  
0100  
1100  
1101  
1111  
1110  
1010  
1011  
1001  
1000

Remove most significant bit and you obtain a nice reflected sequence:

x000

x001

x011

x010

x110

x111

x101

x100

---mirror

x100

x101

x111

x110

x010

x011

x001

x000

same kind of *reflection* can be found for Gray sequences of any width

# GRAY CODE

## **Convert Binary to Gray Code**

- MSB (Most Significant Bit) of the gray code will be exactly equal to first bit of the given binary number
- Second bit of gray code will be exclusive-or (XOR) of first and second bit of given binary number, i.e if both the bits are same the result will be 0 and if they are different the result will be 1.
- Third bit of gray code will be equal to exclusive-or (XOR) of the second and third bit of the given binary number
- Thus binary to gray code conversion goes on

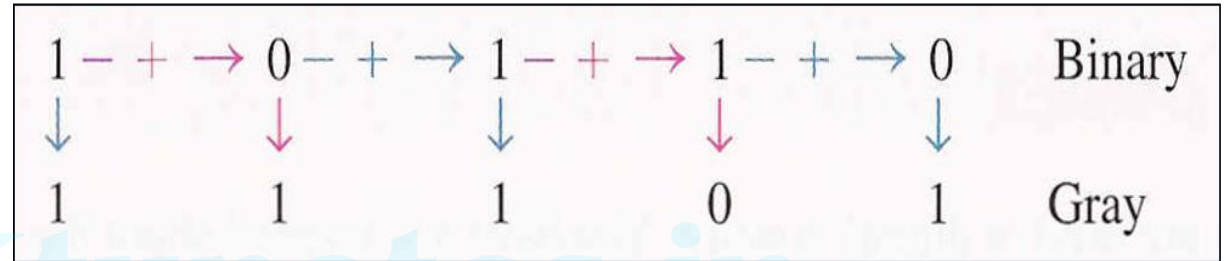
# GRAY CODE

## Gray Code to Binary Conversion

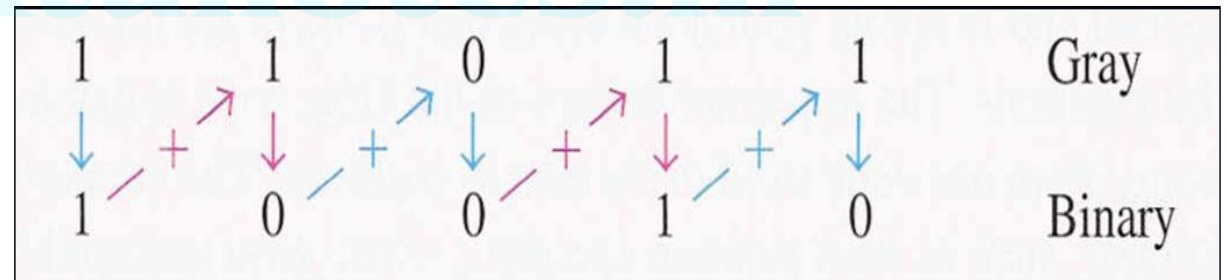
- MSB of binary number will be equal to the MSB of the given gray code.
- Second bit of binary code will be exclusive-or (XOR) of first bit of binary code and second bit of given gray code, i.e if both the bits are same the result will be 0 and if they are different the result will be 1.
- Third bit of binary code will be equal to exclusive-or (XOR) of second bit of binary code and third bit of the given gray code
- Thus binary to gray code conversion goes on

# GRAY CODE

Binary to  
Gray Code



Gray to  
Binary  
Code



**END.....**

**Ktunotes.in**