Number System

Prepared By:
Karuna Patel

Introduction of Number System

The technique to represent and work with numbers is called **number system**. **Decimal number system** is the most common number system. Other popular number systems include **binary number system, octal number system, hexadecimal number system**.

- **Decimal Number System**

Decimal number system is a **base 10** number system having 10 digits from 0 to 9. This means that any numerical quantity can be represented using these 10 digits. Decimal number system is also a **positional value system**.

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

Binary Number System

The easiest way to vary instructions through electric signals is two-state system – on and off. ON is represented as 1 and OFF as 0, though 0 is not actually no signal but signal at a lower voltage. The number system having just these two digits – 0 and 1 – is called **binary number system**.

Each binary digit is also called a **bit**. Binary number system is also positional value system, where each digit has a value expressed in powers of 2, as displayed here.

2^5	2^4	2^3	2^2	2^1	2^0

In any binary number, the rightmost digit is called **least significant bit (LSB)** and leftmost digit is called **most significant bit (MSB)**.

And decimal equivalent of this number is sum of product of each digit with its positional value.

- $11010_2 = 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0$
- $= 16 + 8 + 0 + 2 + 0$
- $= 26_{10}$

Computer memory is measured in terms of how many bits it can store. Here is a chart for memory capacity conversion.

- 1 byte (B) = 8 bits
- 1 Kilobytes (KB) = 1024 bytes
- 1 Megabyte (MB) = 1024 KB
- 1 Gigabyte (GB) = 1024 MB
- 1 Terabyte (TB) = 1024 GB
- 1 Exabyte (EB) = 1024 PB
- 1 Zettabyte = 1024 EB
- 1 Yottabyte (YB) = 1024 ZB

- **Octal number system** - It has eight digits – 0, 1, 2, 3, 4, 5, 6 and 7. Octal number system is also a positional value system with where each digit has its value expressed in powers of 8, as shown here

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

Decimal equivalent of any octal number is sum of product of each digit with its positional value.

- $726_8 = 7\times8^2 + 2\times8^1 + 6\times8^0$
- $= 448 + 16 + 6$
- $= 470_{10}$

- **Hexadecimal Number System -** has 16 symbols – 0 to 9 and A to F where A is equal to 10, B is equal to 11 and so on till F.

Hexadecimal number system is also a positional value system with where each digit has its value expressed in powers of 16, as shown here –

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

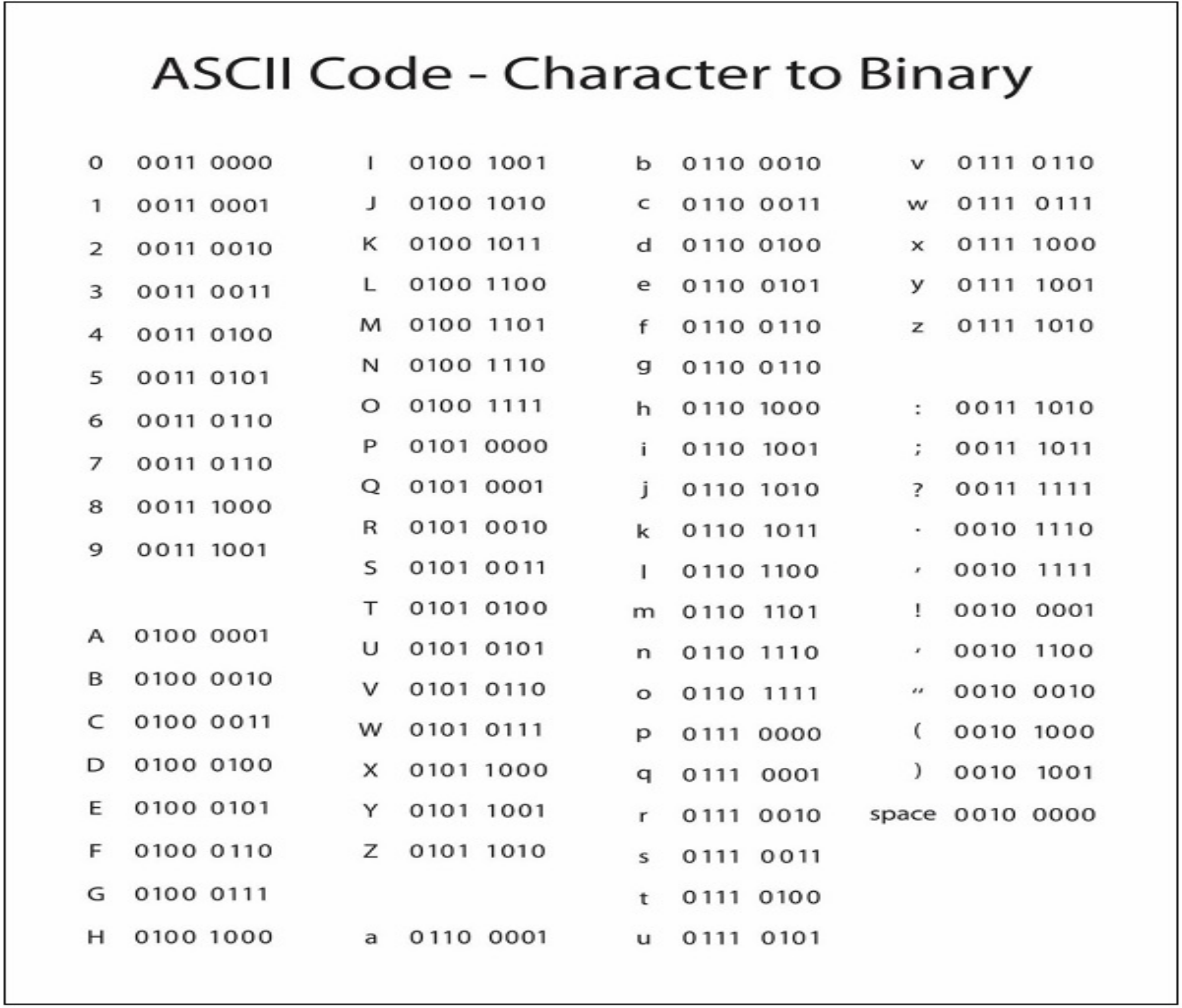
Decimal equivalent of any hexadecimal number is sum of product of each digit with its positional value.

- $27FB_{16} = 2\times16^3 + 7\times16^2 + 15\times16^1 + 11\times16^0$
- $= 8192 + 1792 + 240 + 11$
- $= 10235_{10}$

HEXADECIMAL	DECIMAL	OCTAL	BINARY
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	8	10	1000
9	9	11	1001
A	10	12	1010
B	11	13	1011
C	12	14	1100
D	13	15	1101

ASCII

- Besides numerical data, computer must be able to handle alphabets, punctuation marks, mathematical operators, special symbols, etc. that form the complete character set of English language. The complete set of characters or symbols are called alphanumeric codes. The complete alphanumeric code typically includes −
- 26 upper case letters
- 26 lower case letters
- 10 digits
- 7 punctuation marks
- 20 to 40 special characters
- Now a computer understands only numeric values, whatever the number system used. So all characters must have a numeric equivalent called the alphanumeric code. The most widely used alphanumeric code is American Standard Code for Information Interchange (ASCII). ASCII is a 7-bit code that has 128 (27) possible codes.



ASCII Code - Character to Binary

0	0011 0000	I	0100 1001	b	0110 0010	v	0111 0110
1	0011 0001	J	0100 1010	c	0110 0011	w	0111 0111
2	0011 0010	K	0100 1011	d	0110 0100	x	0111 1000
3	0011 0011	L	0100 1100	e	0110 0101	y	0111 1001
4	0011 0100	M	0100 1101	f	0110 0110	z	0111 1010
5	0011 0101	N	0100 1110	g	0110 0110		
6	0011 0110	O	0100 1111	h	0110 1000	:	0011 1010
7	0011 0110	P	0101 0000	i	0110 1001	;	0011 1011
8	0011 1000	Q	0101 0001	j	0110 1010	?	0011 1111
9	0011 1001	R	0101 0010	k	0110 1011	.	0010 1110
		S	0101 0011	l	0110 1100	,	0010 1111
A	0100 0001	T	0101 0100	m	0110 1101	!	0010 0001
B	0100 0010	U	0101 0101	n	0110 1110	,	0010 1100
C	0100 0011	V	0101 0110	o	0110 1111	"	0010 0010
D	0100 0100	W	0101 0111	p	0111 0000	(	0010 1000
E	0100 0101	X	0101 1000	q	0111 0001	)	0010 1001
F	0100 0110	Y	0101 1001	r	0111 0010	space	0010 0000
G	0100 0111	Z	0101 1010	s	0111 0011		
H	0100 1000			t	0111 0100		
		a	0110 0001	u	0111 0101		

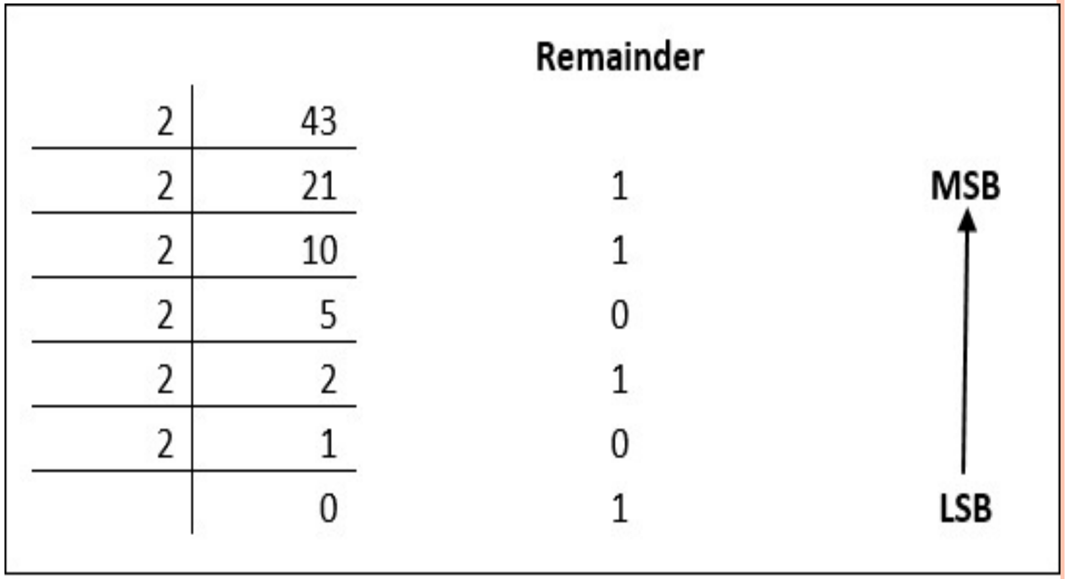
Number System Conversion

- **Decimal to Binary**

Decimal numbers can be converted to binary by repeated division of the number by 2 while recording the remainder. Let's take an example to see how this happens.

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

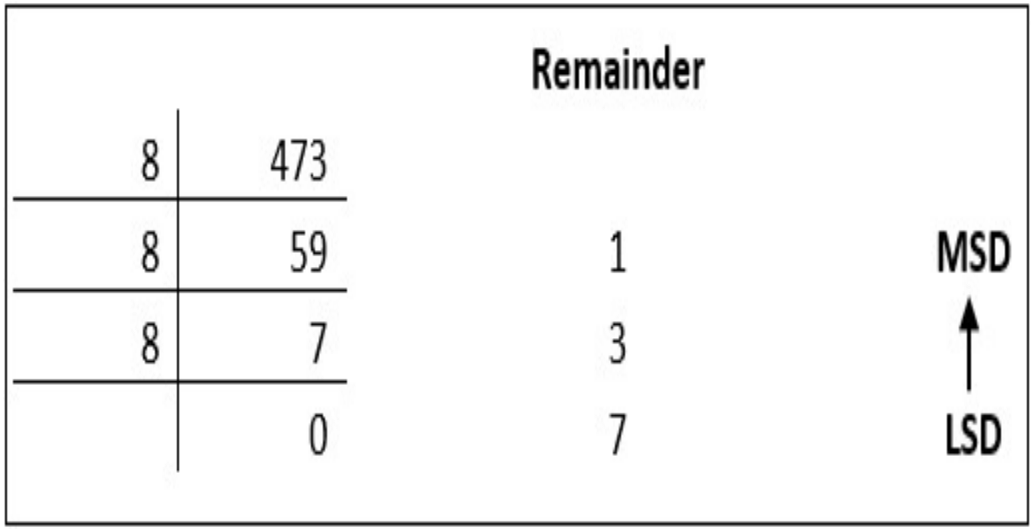
$43_{10} = 101011_2$



- **Decimal to Octal**

  Decimal numbers can be converted to octal by repeated division of the number by 8 while recording the remainder.
- Reading the remainders from bottom to top,

$473_{10} = 731_8$

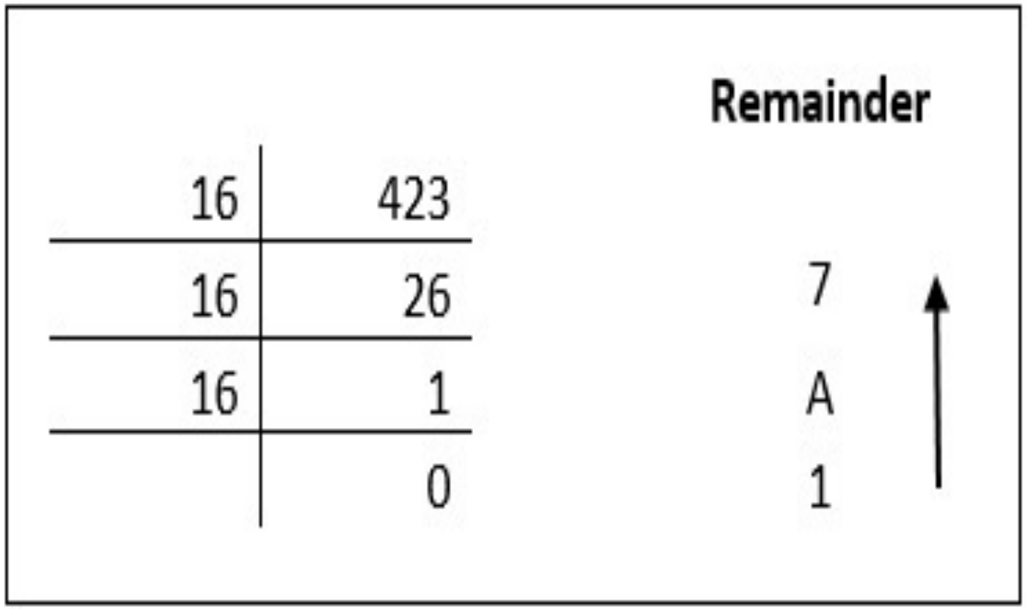


		Remainder	
8	473		
8	59	1	MSD
8	7	3	↑
	0	7	LSD

- **Decimal to Hexadecimal**

  Decimal numbers can be converted to octal by repeated division of the number by 16 while recording the remainder.
- Reading the remainders from bottom to top we get,

  $423_{10} = 1A7_{16}$



- **Binary to Decimal**

This method is used positional representation of the binary.

- **Step 1**: Write down the binary number.
- **Step 2**: Starting with the least significant digit (LSB - the rightmost one), multiply the digit by the value of the position. Continue doing this until you reach the most significant digit (MSB - the leftmost one).
- **Step 3**: Add the results and you will get the decimal equivalent of the given binary number.

Now, let's apply these steps to, for example, the binary number above, which is $(1010)_2$

- **Step 1**: Write down $(1010)_2$ and determine the positions, namely the powers of 2 that the digit belongs to.

- **Step 2**: Represent the number in terms of its positions. $(1 * 2^3) + (0 * 2^2) + (1 * 2^1) + (0 * 2^0)$

- **Step 3**:
- $(1 * 8) + (0 * 4) + (1 * 2) + (0 * 1)$
- $8 + 0 + 2 + 0 = 10$
- Therefore, $(1010)_2 = (10)_{10}$

- Binary to Octal and Vice Versa

  To convert a binary number to octal number, these steps are followed −

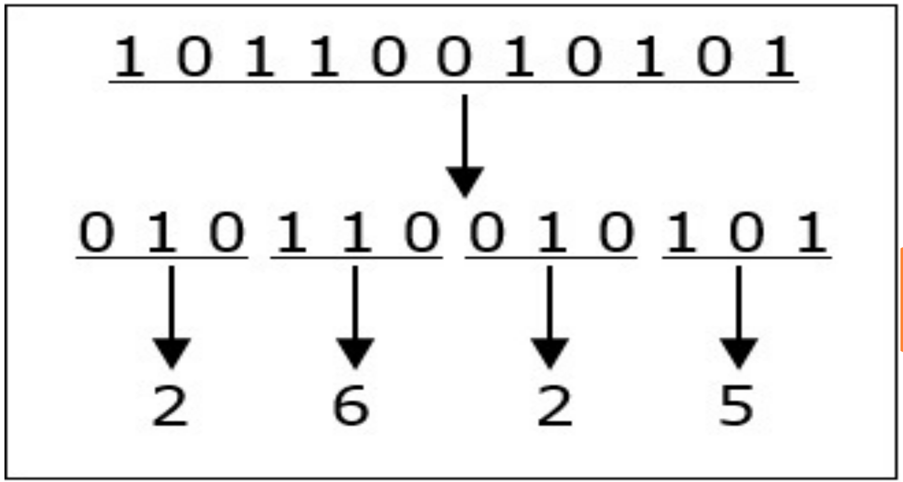
Step -1 Starting from the least significant bit, make groups of three bits.

Step -2 If there are one or two bits less in making the groups, 0s can be added after the most significant bit.

Step -3 Convert each group into its equivalent octal number

Let's take an example to understand this.

- $1011001010_{12} = 2625_8$



- To convert an **octal number to binary**, each octal digit is converted to its 3-bit binary equivalent according to this table.

Octal Digit	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Binary Equivalent	000	001	010	011	100	101	110	111

Example:
$54673_8 = 101100110111011_2$

Binary to Hexadecimal

To convert a binary number to hexadecimal number, these steps are followed −

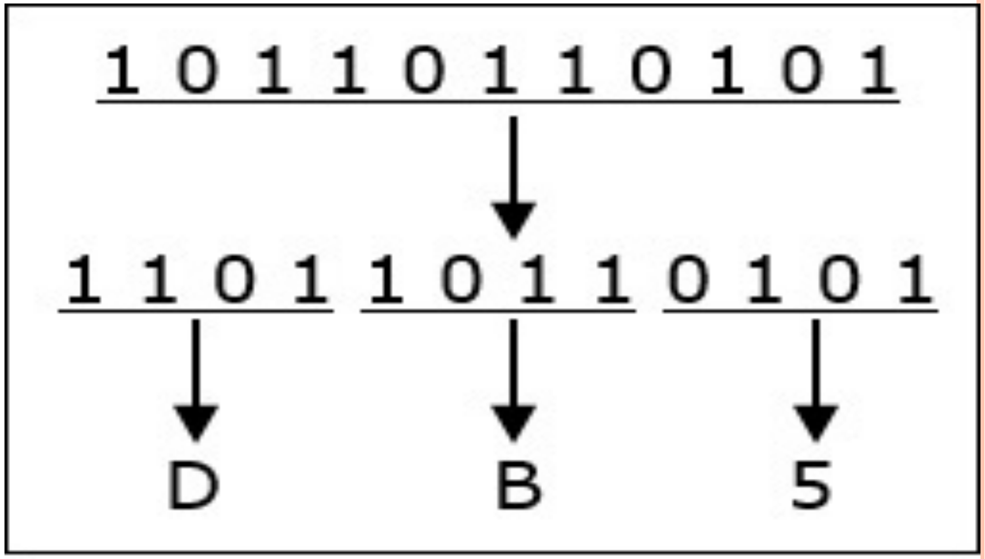
Step-1 Starting from the least significant bit, make groups of four bits.

Step- 2 If there are one or two bits less in making the groups, 0s can be added after the most significant bit.

Step-3 Convert each group into its equivalent octal number.

Example:

$10110110101_2 = DB5_{16}$



- There are only 16 digits (from 0 to 7 and A to F) in hexadecimal number system, so we can represent any digit of hexadecimal number system using only 4 bit as following below.

Hexa	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Binary	0000	0001	0010	0011	0100	0101	0110	0111

Hexa	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
Binary	1000	1001	1010	1011	1100	1101	1110	1111

To convert an **Hexadecimal number to binary**, each Hexa digit is converted to its 4-bit binary equivalent according to this table.

Example:
$59BB_{16} = 101100110111011_2$

- **Octal to Decimal**

The conversion can be performed in the conventional mathematical way, by showing each digit place as an increasing power of 8.

Example : $(345)_8 = (229)_{10}$

$= (5 * 8^0) + (4 * 8^1) + (3 * 8^2)$

$= 5 + 32 + 192$

$= 229$

Octal to Hexadecimal

- When converting from octal to hexadecimal, it is often easier to first convert the octal number into binary and then from binary into hexadecimal. For example, to convert 345 octal into hex:
- Octal=345
- Binary=011100101

Drop any leading zeros or pad with leading zeros to get groups of four binary digits (bits):
Binary 011100101 = 1110 0101

Binary:	**0000**	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**
Hexadecimal:	0	1	2	3	4	5	6	7
Binary:	1000	1001	1010	1011	1100	1101	1110	1111
Hexadecimal:	8	9	A	B	C	D	E	F

Binary =1110 0101
E   5
Hexadecimal=E5
Therefore, through a two-step conversion process, octal 345 equals binary 011100101 equals hexadecimal E5.

- **Hexadecimal to Octal**

When converting from hexadecimal to octal, it is often easier to first convert the hexadecimal number into binary and then from binary into octal. For example, to convert A2DE hex into octal:

(from the previous example)

Hexadecimal =A2DE

Binary =1010001011011110

Binary Add leading zeros or remove leading zeros to group into sets of three binary digits.

Binary: 1010001011011110

= 001 010 001 011 011 110

According to this group you can apply the conversion of Octal to Binary.

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

Binary =	001	010	001	011	011	110
Octal =	1	2	1	3	3	6

=$(121336)_8$

Hexadecimal to Decimal

Converting hexadecimal to decimal can be performed in the conventional mathematical way, by showing each digit place as an increasing power of 16.

A2DE hexadecimal:

= ((A) * 16^3) + (2 * 16^2) + ((D) * 16^1) + ((E) * 16^0)

= (10 * 16^3) + (2 * 16^2) + (13 * 16^1) + (14 * 16^0)

= (10 * 4096) + (2 * 256) + (13 * 16) + (14 * 1)

= 40960 + 512 + 208 + 14

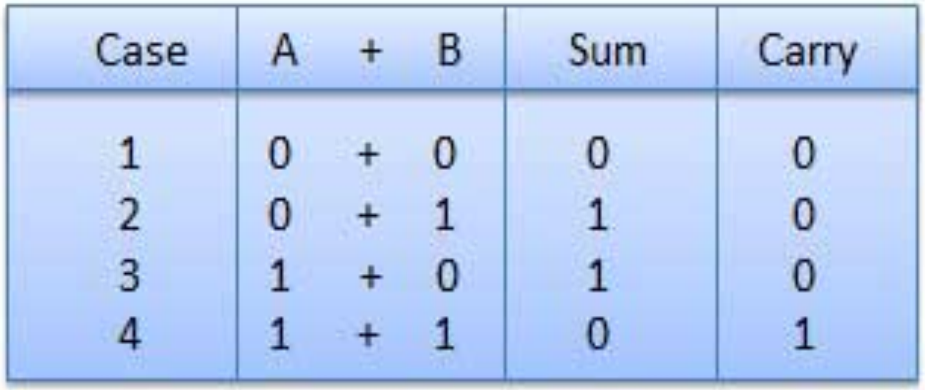
= 41694 decimal

Binary Addition

- For example: in decimal addition, if you add 8 + 2 you get ten, which you write as 10; in the sum this gives a digit 0 and a carry of 1. Something similar happens in binary addition when you add 1 and 1; the result is two (as always), but since two is written as 10 in binary, we get, after summing 1 + 1 in binary, a digit 0 and a carry of 1.
- Therefore in binary:
  0 + 0 = 0
  0 + 1 = 1
  1 + 0 = 1
  1 + 1 = 10 (which is 0 carry 1)

- **Example.** Suppose we would like to add two binary numbers 10 and 11. We start from the last digit. Adding 0 and 1, we get 1 (no carry). That means the last digit of the answer will be one. Then we move one digit to the left: adding 1 and 1 we get 10. Hence, the answer is 101. Note that binary 10 and 11 correspond to 2 and 3 respectively. And the binary sum 101 corresponds to decimal 5: is the binary addition corresponds to our regular addition.

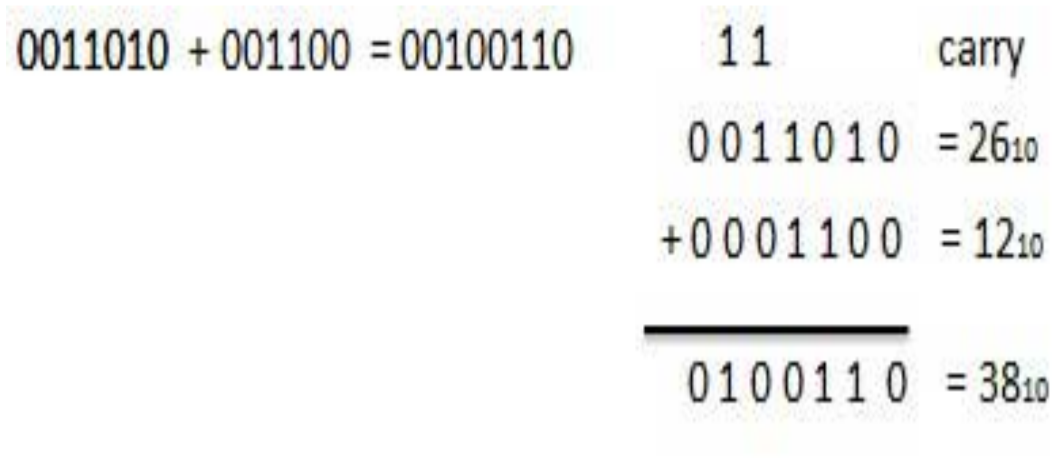
- t is a key for binary subtraction, multiplication, division. There are four rules of binary addition.



Case	A + B	Sum	Carry
1	0 + 0	0	0
2	0 + 1	1	0
3	1 + 0	1	0
4	1 + 1	0	1

- In fourth case, a binary addition is creating a sum of (1 + 1 = 10) i.e. 0 is written in the given column and a carry of 1 over to the next column.

Example – Addition

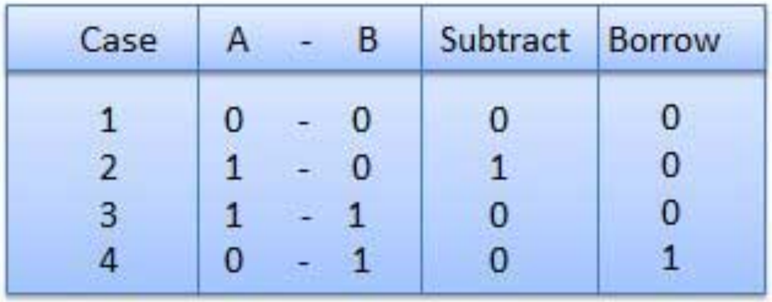


0011010 + 001100 = 00100110

```
  1 1        carry
 0011010   = 26₁₀
+0001100   = 12₁₀
 -------
 0100110   = 38₁₀
```

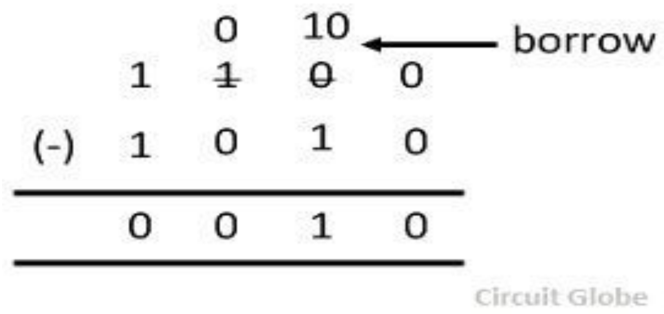
Binary Subtraction

- The subtraction of the binary digit depends on the four basic operations
- 0 – 0 = 0
  1 – 0 = 1
  1 – 1 = 0
  0 – 1 = 0 borrow 1

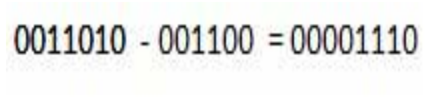


Case	A	-	B	Subtract	Borrow
1	0	-	0	0	0
2	1	-	0	1	0
3	1	-	1	0	0
4	0	-	1	0	1

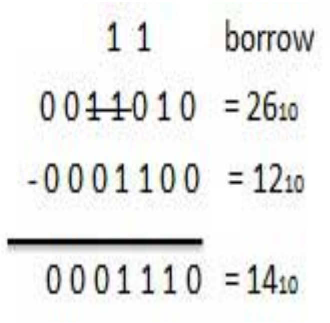
- The above first three operations are easy to understand as they are identical to decimal subtraction. The fourth operation can be understood with the logic two minus one is one.
- For a binary number with two or more digits, the subtraction is carried out column by column as in decimal subtraction. Also, sometimes one has to borrow from the next higher column. Consider the following example.



```
          0   10  ← borrow
      1   ~~1~~   ~~0~~   0
(-)   1   0   1   0
  ________________
      0   0   1   0
```



0011010 - 001100 = 00001110

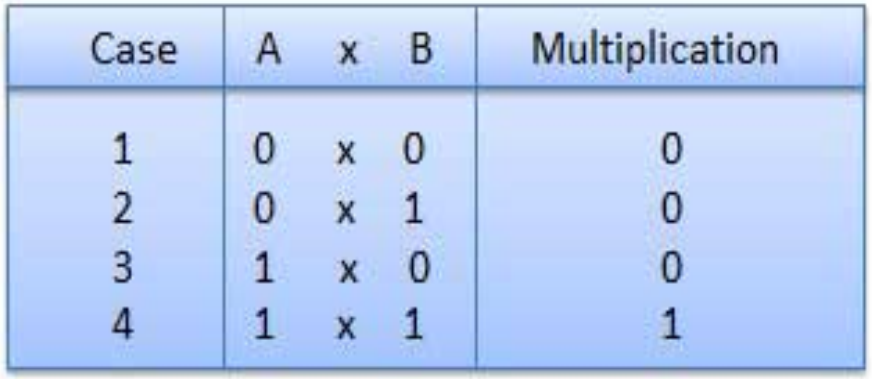


```
    1 1      borrow
 00~~11~~010  = 26₁₀
-0001100  = 12₁₀
 ________
 0001110  = 14₁₀
```

- The above subtraction is carried out through the following steps.
- 0 – 0 = 0
- For 0 – 1 = 1, taking borrow 1 and then 10 – 1 = 1
- For 1 – 0 , since 1 has already been given, it becomes 0 – 0 = 0
- 1 – 1 = 0
- Therefore the result is 0010.

Binary Multiplication

- Binary multiplication is similar to decimal multiplication. It is simpler than decimal multiplication because only 0s and 1s are involved. There are four rules of binary multiplication.



Case	A x B	Multiplication
1	0 x 0	0
2	0 x 1	0
3	1 x 0	0
4	1 x 1	1

- To perform a **binary multiplication** problem, we need to understand how addition works with binary numbers and follow the same process of multiplication and addition we would use with decimal numbers.

- Example:we want to multiply 4 by 3, which in binary will be 100 :

```
x    100
     011
   ------
```

- We start with the first digit, in this case 1, and multiply 100 by 1, which gives the exact same number.

```
x    100
     011
   ------
     100
```

- Then we move to the second digit and proceed to do the same. Since it's also 1, the number will remain intact, we just need to shift the number one digit to the left:

```
x    100
     011
   ------
     100
    100
```

- Then we move to the second digit and proceed to do the same. Since it's also 1, the number will remain intact, we just need to shift the number one digit to the left:

```
x     100
      011
     -----
      100
     100    +
     -----
```

Later, since the last digit is a 0, we can skip it and start with the addition:

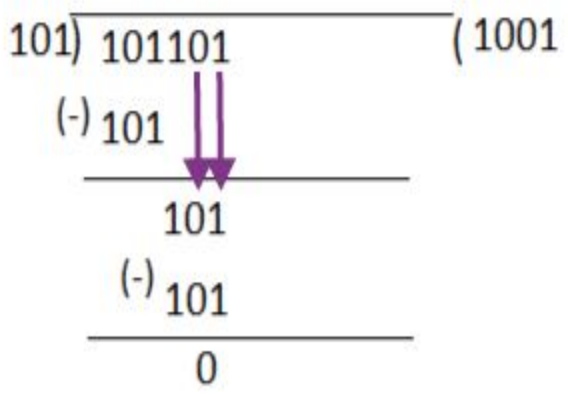
```
x     100
      011
     -----
      100
     100    +
     -----
     1100
```

Binary Division

- **Binary Division Rules**

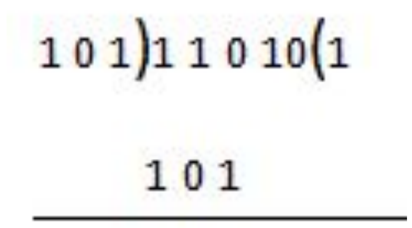
  The binary division is much easier than the decimal division when you remember the following division rules. The main rules of the binary division include:
- 1÷1 = 1
- 1÷0 = Meaningless
- 0÷1 = 0
- 0÷0 = Meaningless

- **Example:** Solve 101101 ÷ 101



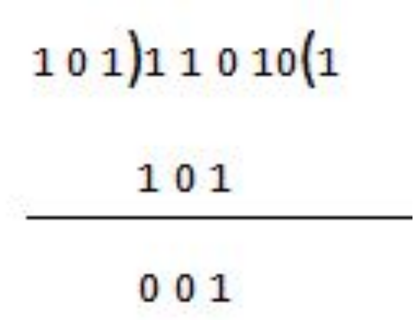
Let us take another example A = 11010 and B = 101, where we want to divide A by B.

- The structure of the operation of binary division is similar to that of decimal division, now we will look into the operation step by step to make it understanding as much as possible.



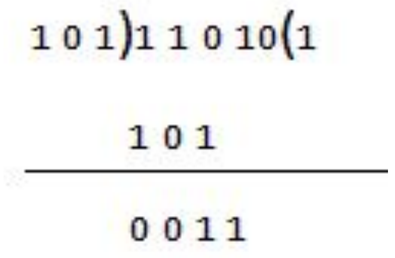
```
101)11010(1
    101
   ------
```

In the first step the left most digits of dividend i.e. A are considered and depending upon the value the divisor is multiplied with 1 and the result which is the result of multiplication of 101 and 1 are written. As we already know that 1 × 1 = 1, 1 × 0 = 0 and 1 × 1 = 1 that is exactly what is written.



```
101)11010(1
    101
   ------
    001
```

In this step 101 is subtracted from 110. This step is also very easy to understand as we already know binary subtraction method. Now going into the next step.



```
101)11010(1
    101
   ------
    0011
```

As of the rules of division the next least significant bit comes down and we try to multiply 1 with divider i.e. B but the result is bigger than the minuend so this step cannot be completed and we have to go to the next step.

```
101)11010(10
     101
  ---------
     00110
```

0 is inserted into the quotient and the least significant bit comes down now we can proceed to the next step.

```
101)11010(101
     101
  ---------
     00110
       101
  ---------
```

Now again the divisor is multiplied with 1 and the result is written, the result is similar to the first one because all the numbers are same. Now we are going into the final step.

```
101)11010(101→ quotient
     101
  ---------------
     00110
       101
  ---------------
       001→remainder
```

In the final step binary subtraction is done and we get the remainder and the operation of **binary division** is completed and we get the following result. Quotient = 101 and remainder = 1.

1's and 2's complement of a Binary Number

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

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

1's complement of "1100" is "0011“

- **2's complement** of a binary number is 1 added to the 1's complement of the binary number.
- Examples:

2's complement of "0111" is "1001"

2's complement of "1100" is "0100"

Subtractions by 1's Complement:

- The algorithm to subtract two binary number using 1's complement is explained as following below:
- Take 1's complement of the subtrahend
- Add with minuend
- If the result of above addition has carry bit 1, then add it to the least significant bit (LSB) of given result
- If there is no carry bit 1, then take 1's complement of the result which will be negative
- Note that subtrahend is number that to be subtracted from the another number, i.e., minuend.

- **Example (Case-1: When Carry bit 1):**
  Evaluate 10101 - 00101
- According to above algorithm,
- Take 1’s complement of subtrahend 00101, which will be 11010,
- Then add both of these.
- So, 10101 + 11010 =**1** 01111 .
- Since, there is carry bit 1, so add this to the LSB of given result.
- i.e., 01111+**1**=10000 which is the answer.
- **Example (Case-2: When no Carry bit):**
- Evaluate 11001 with 11110
- Take 1’s complement of subtrahend 11110, which will be 00001.
- Then add both of these, So, 11001 + 00001 =11010 .
- Since there is no carry bit 1, so take **1’s complement of above result**, which will be 00101, and this is negative number, i.e., 00101, which is the answer.

Additions by 1's Complement:

- There are difference scenario for addition of two binary numbers using 1's complement. These are explained as following below.
- **Case-1: Addition of positive and negative number when positive number has greater magnitude:**
- When positive number has greater magnitude, then take simply 1's complement of negative number and the end-around carry of the sum is added to the least significant bit (LSB).
- **Example:** Add 1110 and -1101.
- So, take 1's complement of 1101, which will be 0010, then add with given number.
- So, 1110+0010=1 0000 , then add this carry bit to the LSB, 0000+1=0001 , which is the answer.
- Note that if the register size is big then fill the same value of MSB to preserve sign magnitude for inputs and output.

- **Case-2: Addition of positive and negative number when negative number has greater magnitude:**
- When the negative number has greater magnitude, then take 1’s complement of negative number and add with given positive number. Since there will not be any end-around carry bit, so take 1’s complement of the result and this result will be negative.
- **Example:** Add 1010 and -1100 in five-bit registers.
- Note that there are five-bit registers, so these new numbers will be 01010 and -01100. Now take 1’s complement of 01100 which will be 10011 and add 01010+10011=11101 . Then take 1’s complement of this result, which will be 00010 and this will be negative number, i.e., -00010, which is the answer.

- **Case-3: Addition of two negative numbers:**
- You need to take 1's complement for both numbers, then add these 1's complement of numbers. Since there will always be end-around carry bit, so add this again to the MSB of result. Now, take 1's complement also of previous result, so this will be negative number.
- Alternatively, you can add both negative number directly, and get this result which will be negative only.
- **Example:** add -1010 and -0101 in five bit-register.
- These five bit numbers are -01010 and -00101. Add complements of these numbers,

  10101+11010 =1 01111 . Since, there is a carry bit 1, so add this to the LSB of result, i.e., 01111+1=10000 . Now take the 1's complement of this result, which will be 01111 and this number is negative, i.e, -01111, which is answer.

Subtractions by 2's Complement

- The algorithm to subtract two binary number using 2's complement is explained as following below –
- Take 2's complement of the subtrahend
- Add with minuend
- If the result of above addition has carry bit 1, then it is dropped and this result will be positive number.
- If there is no carry bit 1, then take 2's complement of the result which will be negative

Note that subtrahend is number that to be subtracted from the another number, i.e., minuend.

Also, note that adding *end-around carry-bit* occurs only in 1's complement arithmetic operations but not 2's complement arithmetic operations.

- **Example (Case-1: When Carry bit 1)** −
- Evaluate 10101 - 00101
- According to above algorithm, take 2’s complement of subtrahend 00101, which will be 11011,
- Then add both of these. So, 10101 + 11011 =1 10000. Since, there is carry bit 1,
- so dropped this carry bit 1, and take this result will be 10000 will be positive number.
- **Example (Case-2: When no Carry bit)** −
- Evaluate 11001 - 11100
- According to above algorithm, take 2’s complement of subtrahend 11110, which will be 00100.
- Then add both of these, So, 11001 + 00100 =11101. Since there is no carry bit 1,
- so take 2’s complement of above result, which will be 00011, and this is negative number, i.e, 00011, which is the answer.

Additions by 2's Complement

- **Case-1 − Addition of positive and negative number when positive number has greater magnitude:**

  When positive number has greater magnitude, then take simply 2's complement of negative number and carry bit 1 is dropped and this result will be positive number.

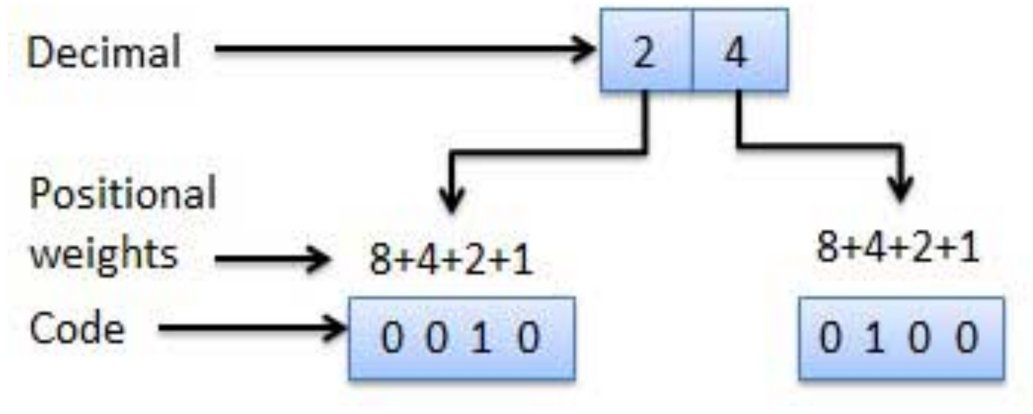
- **Example** −Add 1110 and -1101.
- So, take 2's complement of 1101, which will be 0011, then add with given number. So, 1110+0011=1 0001, and carry bit 1 is dropped and this result will be positive number, i.e., +0001.
- Note that if the register size is big then use sign extension method of MSB bit to preserve sign of number.

- **Case-2 − Addition of positive and negative number when negative number has greater magnitude −**
- When the negative number has greater magnitude, then take 2's complement of negative number and add with given positive number. Since there will not be any end-around carry bit, so take 2's complement of the result and this result will be negative.
- **Example** −Add 1010 and -1100 in five-bit registers.
- Note that there are five-bit registers, so these new numbers will have 01010 and -01100. Now take 2's complement of 01100 which will be 10100 and add 01010+10100=11110. Then take 2's complement of this result, which will be 00010 and this will be negative number, i.e., -00010, which is the answer.

- **Case-3 − Addition of two negative numbers −**
- You need to take 2's complement for both numbers, then add these 2's complement of numbers. Since there will always be end-around carry bit, so it is dropped. Now, take 2's complement also of previous result, so this will be negative number.
- Alternatively, you can add both of these Binary numbers and take result which will be negative only.
- **Example** − add -1010 and -0101 in five bit-register.
- These five bit numbers are -01010 and -00101. Add 2's complements of these numbers, 10110+11011 =1 10001. Since, there is a carry bit 1, so it is dropped. Now take the 2's complement of this result, which will be 01111 and this number is negative, i.e, -01111, which is answer.

Weighted Codes

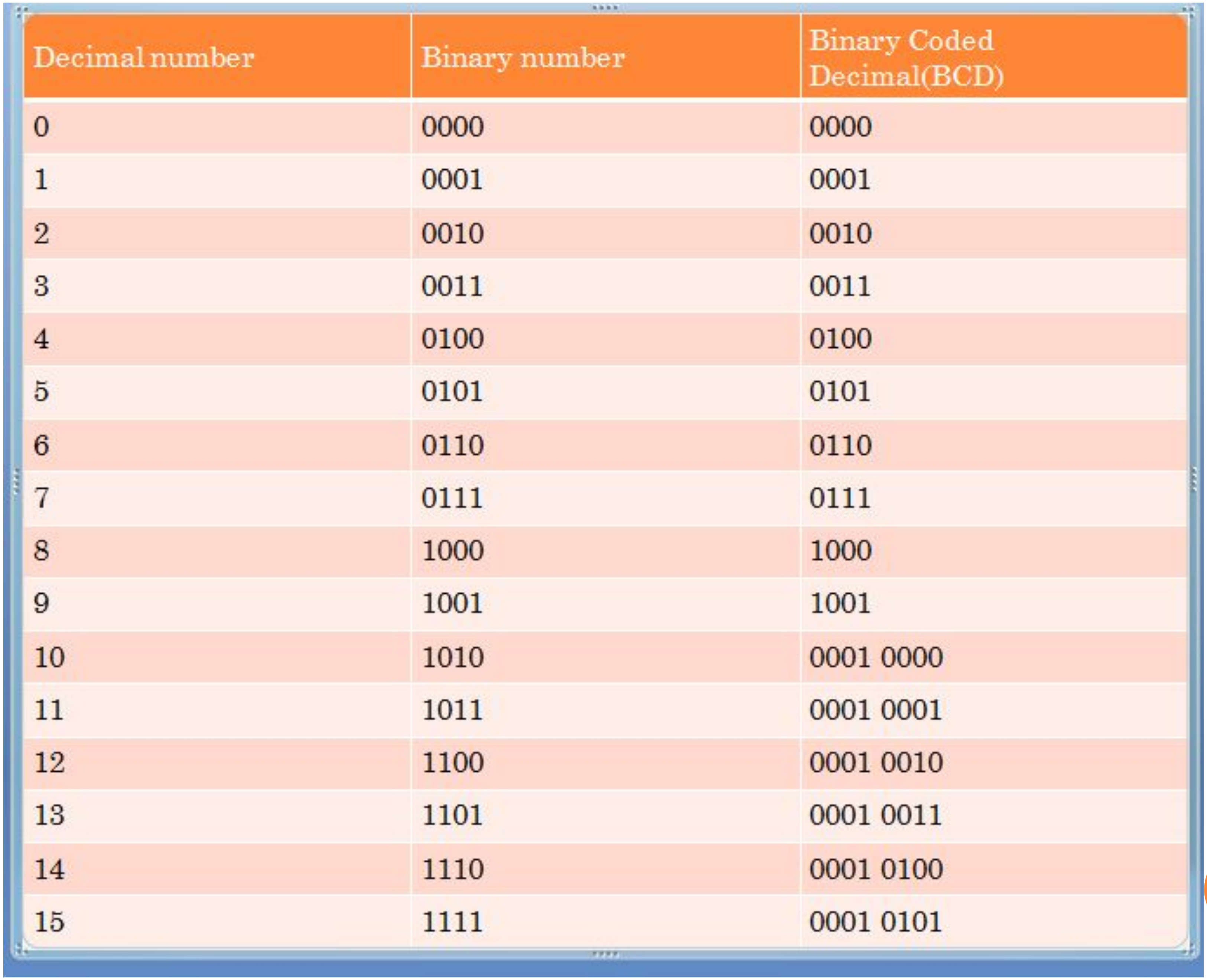
- Weighted binary codes are those binary codes which obey the positional weight principle. Each position of the number represents a specific weight. Several systems of the codes are used to express the decimal digits 0 through 9. In these codes each decimal digit is represented by a group of four bits.



Binary Coded Decimal (BCD) code

- In this code each decimal digit is represented by a 4-bit binary number. BCD is a way to express each of the decimal digits with a binary code. In the BCD, with four bits we can represent sixteen numbers (0000 to 1111). But in BCD code only first ten of these are used (0000 to 1001). The remaining six code combinations i.e. 1010 to 1111 are invalid in BCD.
- **Advantages of BCD Codes**
- ✓ It is very similar to decimal system.
- ✓ We need to remember binary equivalent of decimal numbers 0 to 9 only.
- **Disadvantages of BCD Codes**
- ✓ The addition and subtraction of BCD have different rules.
- ✓ The BCD arithmetic is little more complicated.
- ✓ BCD needs more number of bits than binary to represent the decimal number. So BCD is less efficient than binary.

- In case of **BCD** the binary number formed by four binary digits, will be the equivalent code for the given decimal digits. In **BCD** we can use the binary number from 0000-1001 only, which are the decimal equivalent from 0-9 respectively.
- Let, $(12)_{10}$ be the decimal number whose equivalent **Binary coded decimal** will be 00010010. Four bits from L.S.B is binary equivalent of 2 and next four is the binary equivalent of 1.
- Table given below shows the binary and **BCD** codes for the decimal numbers 0 to 15.
- 0 to 9 the decimal equivalent binary number is of four bit but in case of BCD it is an eight bit number. This is the main **difference** between Binary number and binary coded decimal. For 0 to 9 decimal numbers both binary and BCD is equal but when decimal number is more than one bit BCD differs from binary.



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

- BCD is a numerical code which has several rules for addition. The rules are given below in three steps with an example to make the idea of **BCD Addition** clear.
- Step-1:At first the given number are to be added using the rule of binary.
- For example: **Case-1** **Case-2**

```
  1010          0001
+ 0101        + 0101
  ----          ----
  1111          0110
```

- Step-2: We have to judge the result of addition. Here two cases are shown to describe the rules of **BCD Addition**.
- In **case 1** the result of addition of two binary number is greater than 9, which is **not valid** for BCD number. But the result of addition in **case 2** is less than 9, which is **valid** for BCD numbers.

- Step-3: If the four bit result of addition is greater than 9 and if a carry bit is present in the result then it is invalid and we have to add 6 whose binary equivalent is $(0110)_2$ to the result of addition. Then the resultant that we would get will be a valid **binary coded number**. In **case 1** the result was $(1111)_2$, which is greater than 9 so we have to add 6 or $(0110)_2$ to it.
- Example:

(1111) + (0110) = 0001 0101

Why 6 we have to add to find the BCD which value is more than 9?.

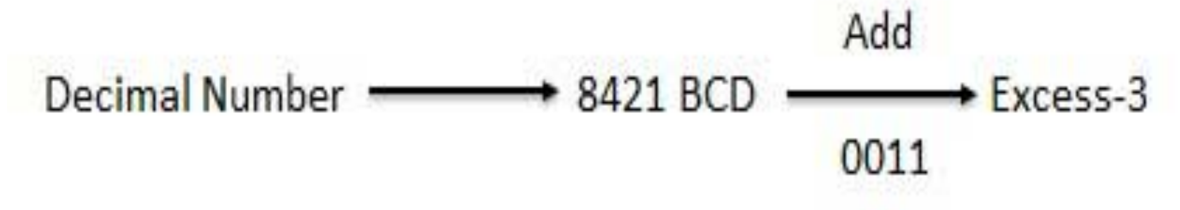
- It is done to skip the six invalid states of binary coded decimal i.e from 10 to 15 and again return to

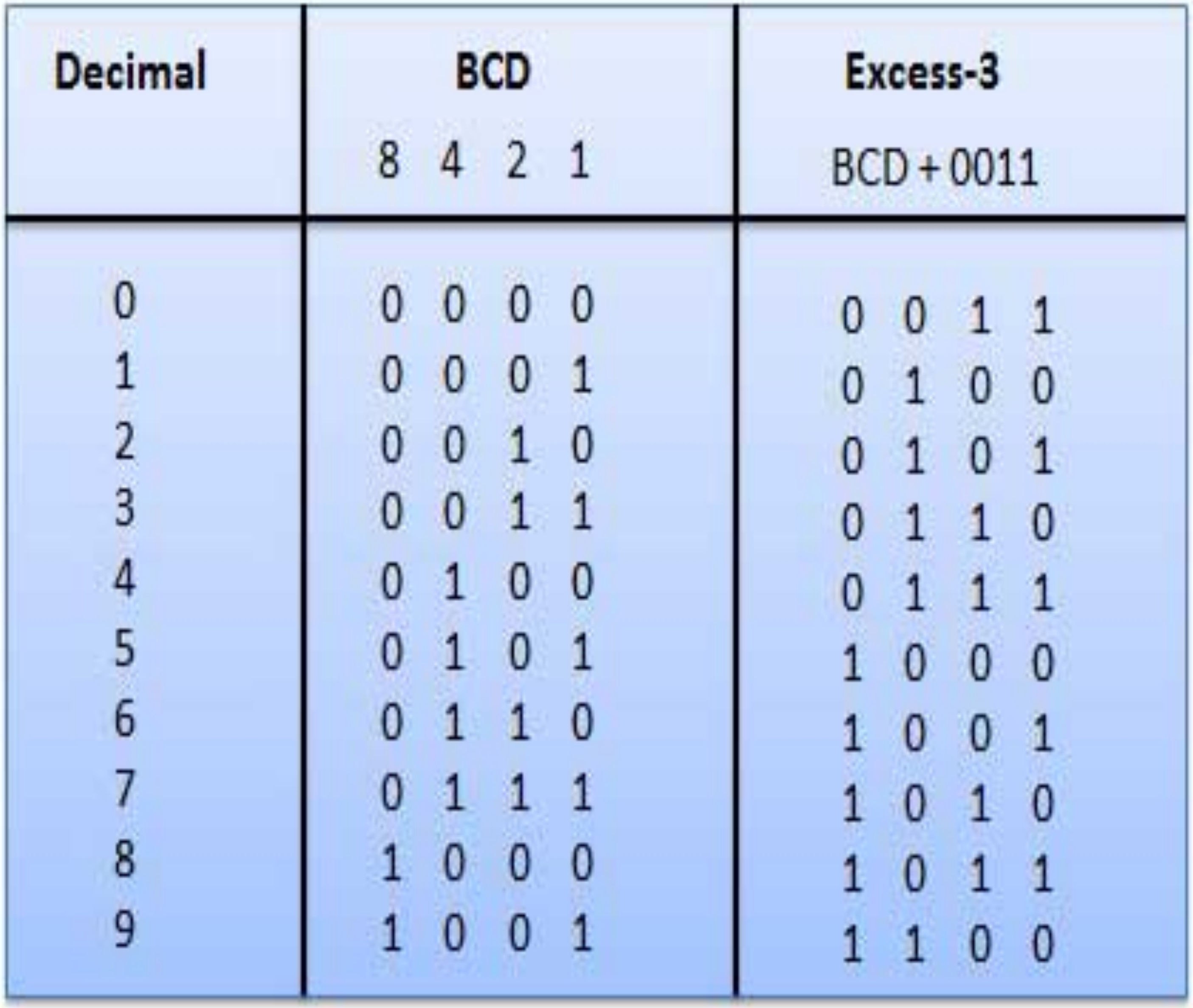
  the BCD codes.

Non-Weighted Number System Applications

Excess-3 code

- The Excess-3 code is also called as XS-3 code. It is non-weighted code used to express decimal numbers. The Excess-3 code words are derived from the 8421 BCD code words adding $(0011)_2$ or (3)10 to each code word in 8421.
- The excess-3 codes are obtained as follows −





Decimal	BCD	Excess-3
	8 4 2 1	BCD + 0011
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	1 0 0 1
7	0 1 1 1	1 0 1 0
8	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 0 0

Gray Code

It is the non-weighted code and it is not arithmetic codes. That means there are no specific weights assigned to the bit position.

It has a very special feature that, only one bit will change each time the decimal number is incremented.

As only one bit changes at a time, the gray code is called as a unit distance code.

The gray code is a cyclic code. Gray code cannot be used for arithmetic operation.

Gray code is not weighted that means it does not depends on positional value of digit. This cyclic variable code that means every transition from one value to the next value involves only one bit change.

- Gray code can be difficult to understand initially, but becomes much easier to understand when looking at the gray code tables below.
- Parallel encoder output provides a stream of bits in quick succession. When the data is in binary format, multiple bits may change per step and in some instances, all bits may change between each read.
- The Reflected Binary Code specified by **Frank Gray** in his patent application is determined using the following steps:
- Starting with bits in the first column:

0
1

- Take the reflection, as if a mirror were held up to the column:

0
1______mirror
1
0

- This results in a column with 4 entries, but the first and last are the same, as are the middle ones, so another column and alternate bits are added:

00
01
11
10

- Then reflect that:

  00
  01
  11
  10_______mirror
  10
  11
  01
  00

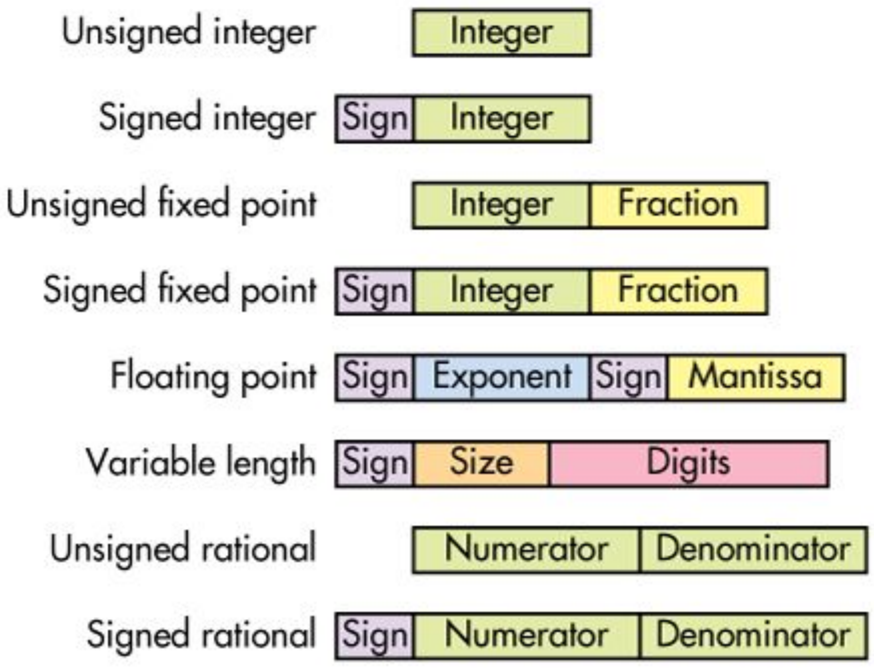
- and add another column with alternate bits:

  000
  001
  011
  010
  110
  111
  101
  100

- and it continues.
- A Gray Code is useful when rapidly changing values could result in errors due to hardware and interfacing constraints.

FIXED-POINT & FLOATING-POINT

- Embedded C and C++ programmers are familiar with signed and unsigned integers and floating-point values of various sizes, but a number of numerical formats can be used in embedded applications.



- Fixed-point formats offer an alternative to floating-point values.
- A fixed point consists of an integer and fraction portion.
- The number of bits used for each relates to the definition and implementation.
- Fixed point isn’t directly supported by programming languages like C and C++, although libraries are available that provide this support.
- FPGAs typically support fixed-point operations in addition to integer and floating-point operations.
- Floating-point support in an FPGA often uses more than 100 times as many gates compared to fixed-point support.