Unit1: Data Representation

A data-type defines the type of value an object can have and what operations can be performed on it. A data type should be declared first before being used. Different programming languages support different data-types. For example,

* C supports char, int, float, long, etc.
* Python supports String, List, Tuple, etc.

In a broad sense, there are three types of data types −

* **Fundamental data types** − These are the predefined data types which are used by the programmer directly to store only one value as per requirement, i.e., integer type, character type, or floating type. For example − int, char, float, etc.
* **Derived data types** − These data types are derived using built-in data type which are designed by the programmer to store multiple values of same type as per their requirement. For example − Array, Pointer, function, list, etc.
* **User-defined data types** − These data types are derived using built-in data types which are wrapped into a single a data type to store multiple values of either same type or different type or both as per the requirement. For example − Class, Structure, etc.

Number Systems

When we type some letters or words, the computer translates them in numbers as computers can understand only numbers. A computer can understand the positional number system where there are only a few symbols called digits and these symbols represent different values depending on the position they occupy in the number.

The value of each digit in a number can be determined using −

* The digit
* The position of the digit in the number
* The base of the number system (where the base is defined as the total number of digits available in the number system)

Decimal Number System

The number system that we use in our day-to-day life is the decimal number system. Decimal number system has base 10 as it uses 10 digits from 0 to 9. In decimal number system, the successive positions to the left of the decimal point represent units, tens, hundreds, thousands, and so on.

Each position represents a specific power of the base (10). For example, the decimal number 1234 consists of the digit 4 in the units position, 3 in the tens position, 2 in the hundreds position, and 1 in the thousands position. Its value can be written as

(1 x 1000)+ (2 x 100)+ (3 x 10)+ (4 x l)

(1 x 103)+ (2 x 102)+ (3 x 101)+ (4 x l00)

1000 + 200 + 30 + 4

1234

As a computer programmer or an IT professional, you should understand the following number systems which are frequently used in computers.

|  |  |
| --- | --- |
| **S.No.** | **Number System and Description** |
| 1 | **Binary Number System**Base 2. Digits used : 0, 1 |
| 2 | **Octal Number System**Base 8. Digits used : 0 to 7 |
| 3 | **Hexa Decimal Number System**Base 16. Digits used: 0 to 9, Letters used : A- F |

## Binary Number System

Characteristics of the binary number system are as follows −

* Uses two digits, 0 and 1
* Also called as base 2 number system
* Each position in a binary number represents a **0** power of the base (2). Example 20
* Last position in a binary number represents a **x** power of the base (2). Example 2x where **x** represents the last position - 1.

### **Example**

Binary Number: 101012

Calculating Decimal Equivalent −

|  |  |  |
| --- | --- | --- |
| **Step** | **Binary Number** | **Decimal Number** |
| Step 1 | 101012 | ((1 x 24) + (0 x 23) + (1 x 22) + (0 x 21) + (1 x 20))10 |
| Step 2 | 101012 | (16 + 0 + 4 + 0 + 1)10 |
| Step 3 | 101012 | 2110 |

**Note** − 101012 is normally written as 10101.

## Octal Number System

Characteristics of the octal number system are as follows −

* Uses eight digits, 0,1,2,3,4,5,6,7
* Also called as base 8 number system
* Each position in an octal number represents a **0** power of the base (8). Example 80
* Last position in an octal number represents a **x** power of the base (8). Example 8x where **x** represents the last position - 1

### **Example**

Octal Number: 125708

Calculating Decimal Equivalent −

|  |  |  |
| --- | --- | --- |
| **Step** | **Octal Number** | **Decimal Number** |
| Step 1 | 125708 | ((1 x 84) + (2 x 83) + (5 x 82) + (7 x 81) + (0 x 80))10 |
| Step 2 | 125708 | (4096 + 1024 + 320 + 56 + 0)10 |
| Step 3 | 125708 | 549610 |

**Note** − 125708 is normally written as 12570.

## Hexadecimal Number System

Characteristics of hexadecimal number system are as follows −

* Uses 10 digits and 6 letters, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
* Letters represent the numbers starting from 10. A = 10. B = 11, C = 12, D = 13, E = 14, F = 15
* Also called as base 16 number system
* Each position in a hexadecimal number represents a **0** power of the base (16). Example, 160
* Last position in a hexadecimal number represents a **x** power of the base (16). Example 16x where **x** represents the last position - 1

### **Example**

Hexadecimal Number: 19FDE16

Calculating Decimal Equivalent −

|  |  |  |
| --- | --- | --- |
| **Step** | **Binary Number** | **Decimal Number** |
| Step 1 | 19FDE16 | ((1 x 164) + (9 x 163) + (F x 162) + (D x 161) + (E x 160))10 |
| Step 2 | 19FDE16 | ((1 x 164) + (9 x 163) + (15 x 162) + (13 x 161) + (14 x 160))10 |
| Step 3 | 19FDE16 | (65536+ 36864 + 3840 + 208 + 14)10 |
| Step 4 | 19FDE16 | 10646210 |

**Note** − 19FDE16 is normally written as 19FDE.

## Decimal Representation

## What are Decimal Numbers?

A decimal number is a number where the integer part is separated from the fractional part with the help of a decimal point. The digits are placed to the left and to the right of the decimal to represent numbers greater than or less than one.

There are certain rules to be followed while reading a decimal number. For e.g. 1.23 is read as one point two three and not one point twenty-three.

## What is Decimal Representation?

In Number system, each and every real number can be represented in the form of a decimal. The decimal representation of any non-negative real numbers “r” is an expression which is in the form of series and is traditionally written as:

r=∑i=0∞ai10i

where ai is a non-negative integer

a1, a2, …are the integers satisfying the condition 0≤ ai≤ 9

For example, if we take the rational number 1/2, it can also be written as 0.5. In the given example,

“0” represents the whole number part

“.” represents a decimal point

“5” represents the decimal part.

For any [rational number](https://byjus.com/maths/rational-numbers/) which is the ratio of any two numbers, can have either a terminating decimal representation or a repeating representation. Similarly, for any fraction of number, we can represent the number using the decimal value. Some of the examples of the recurring decimal representations are as follows:

1/3 = 0.33333..

1/7 =  0.142857142857…

In order to understand the concept of decimals, let us make a square table with 1 row and 10 columns, as shown in the figure.



Let us fill four of these blocks, as shown below



The fraction of colored blocks to the total blocks can be written as 4/10. Another representation for the same can be given in terms of decimals, like 0.4. Here 0.4 = 4 \* or can be written as 4 tenths.

Similarly, if we take a square of 10 rows and 10 columns, we get 100 small squares. If we colour 27 of these blocks, the fractional representation can be written as 27/100.



Here 27/100 = 27\*1/100 = 27 hundredth which is represented in the decimal form as 0.27.

So when we have a number 2.34, it is equivalent to 2 + 3 tenths and 4 hundredths.

## How to Represent the Decimals on the Number Line?

We know how to represent whole numbers on the [number line](https://byjus.com/maths/number-lines/). Let us consider the image shown below. Here, the digits 0 and 1 are represented on the number line.



If we divide the number line into two equal parts, as shown in the figure below, what value does mid-point hold? We will say half of what the graduation between 0 and 1 holds.

So, the point represents (1-0)/2 = 0.5



In order to represent decimals on the number line, we divide the section between two whole numbers as per the places after decimal present in the number to be represented.

### **Decimal Representation on Number Line Examples**

**Example 1:**

Represent 0.8 on the number line.

**Solution:**

As we know, the number 0.8 is equivalent to 8 tenths, so we divide the section between 0 and 1 into 10 equal parts. Now, stepping 8 points from 0 towards 1 gives us 0.8.



**Example 2:**

Represent 8.6 on the number line.

**Solution:**

The number 8.6 = 8 + 0.6

We start from the number 8 on the number line and divide the section between 8 and 9 into 10 equal parts. Now, taking 6 steps from 8 towards 9 gives us the representation of 8.6



To solve more problems on the topic like the [square root of decimals](https://byjus.com/maths/square-root-of-decimals/), you can visit us www.byjus.com also, download BYJU’S – The Learning App to learn Maths with ease.

## Alphanumeric Representation

## Alphanumeric Representation

## • Characters and symbols are encoded as numbers – 1s and 0s as a byte – Note: the conversion to the character occurs at the peripheral device. The machine itself has no perception of what the bits stand for.

## • Standard encoding is called ASCII (7 bits, extended to 8 later) or ANSI – American Standard Code for Information Interchange – American National Standards Institute

## • Strings – A set of characters with length and location – In C, string always terminated by NULL char – In Pascal, a string defined as a pointer to an array with first byte giving the length – In assembly, a string defined as a adjacent set of bytes (terminated or not by a null char) in memory with address of the first byte recorded in a label e.g. a pointer to an array of bytes

## ASCII

### What is ASCII?

ASCII (American Standard Code for Information Interchange) is the most common character encoding format for text data in computers and on the internet. In standard ASCII-encoded data, there are unique values for 128 alphabetic, numeric or special additional characters and control codes.

ASCII encoding is based on character encoding used for telegraph data. The [American National Standards Institute](https://www.techtarget.com/searchdatacenter/definition/ANSI) first published it as a standard for computing in 1963.

Characters in ASCII encoding include upper- and lowercase letters A through Z, numerals 0 through 9 and basic punctuation symbols. It also uses some non-printing control characters that were originally intended for use with teletype printing terminals.

ASCII characters may be represented in the following ways:

* as pairs of [hexadecimal](https://www.techtarget.com/whatis/definition/hexadecimal) digits -- base-16 numbers, represented as 0 through 9 and A through F for the decimal values of 10-15;
* as three-digit octal (base 8) numbers;
* as decimal numbers from 0 to 127; or
* as 7-bit or 8-bit [binary](https://www.techtarget.com/whatis/definition/binary)

For example, the ASCII encoding for the lowercase letter "m" is represented in the following ways:

| **Character** | **Hexadecimal** | **Octal** | **Decimal** | **Binary (7 bit)** | **Binary (8 bit)** |
| --- | --- | --- | --- | --- | --- |
| m | 0x6D | /155 | 109 | 110 1101 | 0110 1101 |

ASCII characters were initially encoded into 7 bits and stored as 8-bit characters with the [most significant bit](https://www.techtarget.com/whatis/definition/most-significant-bit-or-byte) -- usually, the left-most bit -- set to 0.

### Why is ASCII important?

ASCII was the first major character encoding standard for data processing. Most modern computer systems use [Unicode](https://www.techtarget.com/whatis/definition/Unicode), also known as the Unicode Worldwide Character Standard. It's a character encoding standard that includes ASCII encodings.

The Internet Engineering Task Force ([IETF](https://www.techtarget.com/whatis/definition/IETF-Internet-Engineering-Task-Force)) adopted ASCII as a standard for internet data when it published "ASCII format for Network Interchange" as [RFC 20](https://datatracker.ietf.org/doc/rfc20/) in 1969. That request for comments ([RFC](https://www.techtarget.com/whatis/definition/Request-for-Comments-RFC)) document standardized the use of ASCII for internet data and was accepted as a full standard in 2015.

ASCII encoding is technically obsolete, having been replaced by Unicode. Yet, ASCII characters use the same encoding as the first 128 characters of the Unicode Transformation Format 8, so ASCII text is compatible with UTF-8.

In 2003, the IETF standardized the use of UTF-8 encoding for all web content in [RFC 3629](https://datatracker.ietf.org/doc/html/rfc3629).

Almost all computers now use ASCII or Unicode encoding. The exceptions are some IBM mainframes that use the proprietary 8-bit code called Extended Binary Coded Decimal Interchange Code ([EBCDIC](https://www.techtarget.com/whatis/definition/EBCDIC-Extended-Binary-Coded-Decimal-Interchange-Code)).

### How does ASCII work?

ASCII offers a universally accepted and understood character set for basic data communications. It enables developers to design interfaces that both humans and computers understand. ASCII codes a string of data as ASCII characters that can be interpreted and displayed as readable plain text for people and as data for computers.

Programmers use the design of the ASCII character set to simplify certain tasks. For example, using ASCII character codes, changing a single bit easily converts text from uppercase to lowercase.

The capital letter "A" is represented by the binary value:

0100 0001

The lowercase letter "a" is represented by the binary value:

0110 0001

The difference is the third most significant bit. In [decimal and hexadecimal](https://www.theserverside.com/tip/Binary-and-hexadecimal-numbers-explained-for-developers), this corresponds to:

| **Character** | **Binary** | **Decimal** | **Hexadecimal** |
| --- | --- | --- | --- |
| A | 01**0**0 0001 | 65 | 0x41 |
| a | 01**1**0 0001 | 97 | 0x61 |

The difference between upper- and lowercase characters is always 32 (0x20 in hexadecimal), so converting from upper- to lowercase and back is a matter of adding or subtracting 32 from the ASCII character code.

Similarly, hexadecimal characters for the digits 0 through 9 are as follows:

| **Character** | **Binary** | **Decimal** | **Hexadecimal** |
| --- | --- | --- | --- |
| 0 | 0011 0000 | 48 | 0x30 |
| 1 | 0011 0001 | 49 | 0x31 |
| 2 | 0011 0010 | 50 | 0x32 |
| 3 | 0011 0011 | 51 | 0x33 |
| 4 | 0011 0100 | 52 | 0x34 |
| 5 | 0011 0101 | 53 | 0x35 |
| 6 | 0011 0110 | 54 | 0x36 |
| 7 | 0011 0111 | 55 | 0x37 |
| 8 | 0011 1000 | 56 | 0x38 |
| 9 | 0011 1001 | 57 | 0x39 |

Using this encoding, developers can easily convert ASCII digits to numerical values by stripping off the four most significant bits of the binary ASCII values (0011). This calculation can also be done by dropping the first hexadecimal digit or by subtracting 48 from the decimal ASCII code.

Developers can also check the most significant bit of characters in a sequence to verify that a data stream, [string](https://www.techtarget.com/whatis/definition/string) or file contains ASCII values. The most significant bit of basic ASCII characters will always be 0; if that bit is 1, then the character is not an ASCII-encoded character.

### ASCII variants and Unicode

When it was first introduced, ASCII supported English language text only. When 8-bit computers became common during the 1970s, vendors and standards bodies began extending the ASCII character set to include 128 additional character values. Extended ASCII incorporates non-English characters, but it is still insufficient for comprehensive encoding of text in most world languages, including English. Different extended ASCII character sets are common, depending on the vendor, language and country.

Initially, other character encoding standards were adopted for other languages. In some cases, the standards were designed for other countries with different requirements. In other cases, the encodings were hardware manufacturers' proprietary designs.

Unicode defines codespaces for the implementation of character encodings for different languages. Characters can be mapped to encodings using one of the following two methods:

1. UTF
2. Universal Coded Character Set (UCS)

Depending on the language and the mapping used, characters can be expressed in one to four 8-bit bytes (UTF-8), in two 16-bit units ([UTF-16](https://www.techtarget.com/whatis/definition/UTF-16-16-bit-Unicode-Transformation-Format)) or in a single 32-bit unit (UTF-32).

The UCS standard is maintained as an [ISO](https://www.techtarget.com/searchdatacenter/definition/ISO) (International Organization for Standardization) standard, ISO/IEC 10646. As of this writing, there are 143,859 different characters defined in version 13.0 of the Unicode standard.

### ASCII advantages and disadvantages

After more than half a century of use, the advantages and disadvantages of using ASCII character encoding are well understood. That is one of the encoding format's great strengths.

#### Advantages

* **Universally accepted.** ASCII character encoding is universally understood. Except for the IBM mainframes that use EBCDIC encoding, it is universally implemented in computing through the Unicode standard. Unicode character encoding replaces ASCII encoding, but it is backward-compatible with ASCII.
* **Compact character encoding.** Standard codes can be expressed in 7 bits. This means data that can be expressed in the standard ASCII character set requires only as many bytes to store or send as the number of characters in the data.
* **Efficient for programming.** The character codes for letters and numbers are well adapted to programming techniques for manipulating text and using numbers for calculations or storage as raw data.

#### Disadvantages

* **Limited character set.** Even with extended ASCII, only 255 distinct characters can be represented. The characters in a standard character set are enough for English language communications. But even with the diacritical marks and Greek letters supported in extended ASCII, it is difficult to accommodate languages that do not use the Latin alphabet.
* **Inefficient character encoding.** Standard ASCII encoding is efficient for English language and numerical data. Representing characters from other alphabets requires more overhead such as escape codes.

### Converting text to ASCII code in Windows

There is more than one way to display text as ASCII codes in Windows. To use the Windows PowerShell command Format-Hex to display ASCII encoding for a text file, perform the following steps.

**Open the Windows PowerShell application.** Click on the search box in the lower left of your Windows 10 desktop. Type PowerShell and click on the PowerShell icon to start the application.

# **9's and 10's Complement**

If the number is binary, then we use 1's complement and 2's complement. But in case, when the number is a decimal number, we will use the 9's and 10's complement. The 10's complement is obtained from the 9's complement of the number, and we can also find the 9's and 10's complement using the r's and (r-1)'s complement formula.

## 9's Complement

The 9's complement is used to find the subtraction of the decimal numbers. The 9's complement of a number is calculated by subtracting each digit of the number by 9. For example, suppose we have a number 1423, and we want to find the 9's complement of the number. For this, we subtract each digit of the number 1423 by 9. So, the 9's complement of the number 1423 is 9999-1423= 8576.

### **Subtraction using 9's complement**

With the help of the 9's complement, the process of subtraction is done in a much easier way. Generally, we subtract the subtrahend from the minuend, but in a case when we perform subtraction using 9's complement, there is no need to do the same.

For subtracting two numbers using 9's complement, we first have to find the 9's complement of the subtrahend and then we will add this complement value with the minuend. There are two possible cases when we subtract the numbers using 9's complement.

**Case 1: When the subtrahend is smaller than the minuend.**

For subtracting the smaller number from the larger number using 9's complement, we will find the 9's complement of the subtrahend, and then we will add this complement value with the minuend. By adding both these values, the result will come in the formation of carry. At last, we will add this carry to the result obtained previously.



**Case 2: When the subtrahend is greater than the minuend.**

In this case, when we add the complement value and the minuend, the result will not come in the formation of carry. This indicates that the number is negative, and for finding the final result, we need to find the 9's complement of the result.



## 10's Complement

The 10's complement is also used to find the subtraction of the decimal numbers. The 10's complement of a number is calculated by subtracting each digit by 9 and then adding 1 to the result. Simply, by adding 1 to its 9's complement we can get its 10's complement value. For example, suppose we have a number 1423, and we want to find the 10's complement of the number. For this, we find the 9's complement of the number 1423 that is 9999-1423= 8576, and now we will add 1 to the result. So the 10's complement of the number 1423 is 8576+1=8577.

### **Subtraction using 10's complement**

For subtracting two numbers using 10's complement, we first have to find the 10's complement of the subtrahend, and then we will add this complement value with the minuend. There are two possible cases when we subtract the numbers using 10's complement.

**Case 1: When the subtrahend is smaller than the minuend.**

For subtracting the smaller number from the larger number using 10's complement, we will find the 10's complement of the subtrahend and then we will add this complement value with the minuend. By adding both these values, the result will come in the formation of carry. We ignore this carry and the remaining digits will be the answer.



**Case 2: When the subtrahend is greater than the minuend.**

In this case, when we add the complement value and the minuend, the result will not come in the formation of carry. This indicates that the number is negative and for finding the final result, we need to find the 10's complement of the result obtained by adding complement value of subtrahend and minuend.

## 9's and 10's Complement

# 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.In the 1’s complement format , the positive numbers remain unchanged . The negative numbers are obtained by taking the 1’s complement of positive counterparts.

for example +9 will be represented as 00001001 in eight-bit notation and -9 will be represented as 11110110, which is the 1’s complement of 00001001.

**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. In the 2’s complement representation of binary numbers, the MSB represents the sign with a ‘0’ used for plus sign and a  ‘1’ used for a minus sign. the remaining bits are used for representing magnitude. positive magnitudes are represented in the same way as in the case of sign-bit or 1’s complement representation.  Negative magnitudes are represented by the 2’s complement of their positive counterparts.

**Examples:**

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

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

#### Another trick to finding two’s complement:

**Step 1:** Start from the Least Significant Bit and traverse left until you find a 1.  Until you find 1, the bits stay the same

**Step 2:**Once you have found 1, let the 1 as it is, and now

**Step 3:**Flip all the bits left into the 1.

Differences between 1’s complement and 2’s complement

These differences are given as following below −

| **1’s complement** | **2’s complement** |
| --- | --- |
| To get 1’s complement of a binary number, simply invert the given number. | To get 2’s complement of a binary number, simply invert the given number and add 1 to the least significant bit (LSB) of given result. |
| 1’s complement of binary number 110010 is 001101 | 2’s complement of binary number 110010 is 001110 |
| Simple implementation which uses only NOT gates for each input bit. | Uses NOT gate along with full adder for each input bit. |
| Can be used for signed binary number representation but not suitable as ambiguous representation for number 0. | Can be used for signed binary number representation and most suitable as unambiguous representation for all numbers. |
| 0 has two different representation one is -0 (e.g., 1 1111 in five bit register) and second is +0 (e.g., 0 0000 in five bit register). | 0 has only one representation for -0 and +0 (e.g., 0 0000 in five bit register). Zero (0) is considered as always positive (sign bit is 0) |
| For k bits register, positive largest number that can be stored is (2(k-1)-1)  and negative lowest number that can be stored is -(2(k-1)-1). | For k bits register, positive largest number that can be stored is (2(k-1)-1) and negative lowest number that can be stored is -(2(k-1)). |
| *end-around-carry-bit*addition occurs in 1’s complement arithmetic operations. It added to the LSB of result. | *end-around-carry-bit*addition does not occur in 2’s complement arithmetic operations. It is ignored. |
| 1’s complement arithmetic operations are not easier than 2’s complement because of  addition of *end-around-carry-bit.* | 2’s complement arithmetic operations are much easier than 1’s complement because of there is no addition of *end-around-carry-bit.* |
| Sign extension is used for converting a signed integer from one size to another. | Sign extension is used for converting a signed integer from one size to another. |

## 2’s Complement Addition

## Addition using 2's complement

There are three different cases possible when we add two binary numbers using 2's complement, which is as follows:

**Case 1: Addition of the positive number with a negative number when the positive number has a greater magnitude.**

Initially find the 2's complement of the given negative number. Sum up with the given positive number. If we get the end-around carry 1 then the number will be a positive number and the carry bit will be discarded and remaining bits are the final result.

**Example: 1101 and -1001**

1. First, find the 2's complement of the negative number 1001. So, for finding 2's complement, change all 0 to 1 and all 1 to 0 or find the 1's complement of the number 1001. The 1's complement of the number 1001 is 0110, and add 1 to the LSB of the result 0110. So the 2's complement of number 1001 is 0110+1=0111
2. Add both the numbers, i.e., 1101 and 0111;
1101+0111=1  0100
3. By adding both numbers, we get the end-around carry 1. We discard the end-around carry. So, the addition of both numbers is 0100.

**Case 2: Adding of the positive value with a negative value when the negative number has a higher magnitude.**

Initially, add a positive value with the 2's complement value of the negative number. Here, no end-around carry is found. So, we take the 2's complement of the result to get the final result.

#### **Note: The resultant is a negative value.**

**Example: 1101 and -1110**

1. First, find the 2's complement of the negative number 1110. So, for finding 2's complement, add 1 to the LSB of its 1's complement value 0001.
0001+1=0010
2. Add both the numbers, i.e., 1101 and 0010;
1101+0010= 1111
3. Find the 2's complement of the result 1110 that is the final result. So, the 2's complement of the result 1110 is 0001, and add a negative sign before the number so that we can identify that it is a negative number.

**Case 3: Addition of two negative numbers**

In this case, first, find the 2's complement of both the negative numbers, and then we will add both these complement numbers. In this case, we will always get the end-around carry, which will be added to the LSB, and forgetting the final result, we will take the2's complement of the result.

#### **Note: The resultant is a negative value.**

**Example: -1101 and -1110 in five-bit register**

1. Firstly find the 2's complement of the negative numbers 01101 and 01110. So, for finding 2's complement, we add 1 to the LSB of the 1's complement of these numbers. 2's complement of the number 01110 is 10010, and 01101 is 10011.
2. We add both the complement numbers, i.e., 10001 and 10010;
10010+10011= 1 00101
3. By adding both numbers, we get the end-around carry 1. This carry is discarded and the final result is the 2.s complement of the result 00101. So, the 2's complement of the result 00101 is 11011, and we add a negative sign before the number so that we can identify that it is a negative number.

## Subtraction using 2's complement

These are the following steps to subtract two binary numbers using 2's complement

* In the first step, find the 2's complement of the subtrahend.
* Add the complement number with the minuend.
* If we get the carry by adding both the numbers, then we discard this carry and the result is positive else take 2's complement of the result which will be negative.

**Example 1:**10101 - 00111

We take 2's complement of subtrahend 00111, which is 11001. Now, sum them. So,

10101+11001 =1 01110.

In the above result, we get the carry bit 1. So we discard this carry bit and remaining is the final result and a positive number.

**Example 2:**10101 - 10111

We take 2's complement of subtrahend 10111, which comes out 01001. Now, we add both of the numbers. So,

10101+01001 =11110.

In the above result, we didn't get the carry bit. So calculate the 2's complement of the result, i.e., 00010. It is the negative number and the final answer.

In the coding, when numbers, letters or words are represented by a specific group of symbols, it is said that the number, letter or word is being encoded. The group of symbols is called as a code. The digital data is represented, stored and transmitted as group of binary bits. This group is also called as **binary code**. The binary code is represented by the number as well as alphanumeric letter.

## Advantages of Binary Code

Following is the list of advantages that binary code offers.

* Binary codes are suitable for the computer applications.
* Binary codes are suitable for the digital communications.
* Binary codes make the analysis and designing of digital circuits if we use the binary codes.
* Since only 0 & 1 are being used, implementation becomes easy.

## Classification of binary codes

The codes are broadly categorized into following four categories.

* Weighted Codes
* Non-Weighted Codes
* Binary Coded Decimal Code
* Alphanumeric Codes
* Error Detecting Codes
* Error Correcting Codes

## 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.



## Non-Weighted Codes

In this type of binary codes, the positional weights are not assigned. The examples of non-weighted codes are Excess-3 code and Gray code.

### **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 −



### **Example**



### **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 shown in fig. 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.



### **Application of Gray code**

* Gray code is popularly used in the shaft position encoders.
* A shaft position encoder produces a code word which represents the angular position of the shaft.

## 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.

## Alphanumeric codes

A binary digit or bit can represent only two symbols as it has only two states '0' or '1'. But this is not enough for communication between two computers because there we need many more symbols for communication. These symbols are required to represent 26 alphabets with capital and small letters, numbers from 0 to 9, punctuation marks and other symbols.

The alphanumeric codes are the codes that represent numbers and alphabetic characters. Mostly such codes also represent other characters such as symbol and various instructions necessary for conveying information. An alphanumeric code should at least represent 10 digits and 26 letters of alphabet i.e. total 36 items. The following three alphanumeric codes are very commonly used for the data representation.

* American Standard Code for Information Interchange (ASCII).
* Extended Binary Coded Decimal Interchange Code (EBCDIC).
* Five bit Baudot Code.

ASCII code is a 7-bit code whereas EBCDIC is an 8-bit code. ASCII code is more commonly used worldwide while EBCDIC is used primarily in large IBM computers.

# **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.

We can easily get an excess-3 code of a decimal number by simply adding 3 to each decimal digit. And then we write the 4-bit binary number for each digit of the decimal number. We can find the excess-3 code of the given binary number by using the following steps:

1. We find the decimal number of the given binary number.
2. Then we add 3 in each digit of the decimal number.
3. Now, we find the binary code of each digit of the newly generated decimal number.

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

|  |  |  |
| --- | --- | --- |
| **Decimal Digit** | **BCD Code** | **Excess-3 Code** |
| 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 |

In excess-3 code, the codes 1111 and 0000 are never used for any decimal digit. Let's take some examples of Excess-3 code.

### **Example 1: Decimal number 31**

1. We find the BCD code of each digit of the decimal number.

|  |  |
| --- | --- |
| **Digit** | **BCD** |
| 3 | 0011 |
| 1 | 0001 |

Error-detecting codes are a sequence of numbers generated by specific procedures for detecting errors in data that has been transmitted over computer networks.

When bits are transmitted over the computer network, they are subject to get corrupted due to interference and network problems. The corrupted bits leads to spurious data being received by the receiver and are called errors.

Error – detecting codes ensures messages to be encoded before they are sent over noisy channels. The encoding is done in a manner so that the decoder at the receiving end can detect whether there are errors in the incoming signal with high probability of success.

## Features of Error Detecting Codes

* Error detecting codes are adopted when backward error correction techniques are used for reliable data transmission. In this method, the receiver sends a feedback message to the sender to inform whether an error-free message has been received or not. If there are errors, then the sender retransmits the message.
* Error-detecting codes are usually block codes, where the message is divided into fixed-sized blocks of bits, to which redundant bits are added for error detection.
* Error detection involves checking whether any error has occurred or not. The number of error bits and the type of error does not matter.

## Error Detection Techniques

There are three main techniques for detecting errors

## Parity Check

Parity check is done by adding an extra bit, called parity bit to the data to make number of 1s either even in case of even parity, or odd in case of odd parity.

While creating a frame, the sender counts the number of 1s in it and adds the parity bit in following way

* In case of even parity: If number of 1s is even then parity bit value is 0. If number of 1s is odd then parity bit value is 1.
* In case of odd parity: If number of 1s is odd then parity bit value is 0. If number of 1s is even then parity bit value is 1.

On receiving a frame, the receiver counts the number of 1s in it. In case of even parity check, if the count of 1s is even, the frame is accepted, otherwise it is rejected. Similar rule is adopted for odd parity check.

Parity check is suitable for single bit error detection only.

## Checksum

In this error detection scheme, the following procedure is applied

* Data is divided into fixed sized frames or segments.
* The sender adds the segments using 1’s complement arithmetic to get the sum. It then complements the sum to get the checksum and sends it along with the data frames.
* The receiver adds the incoming segments along with the checksum using 1’s complement arithmetic to get the sum and then complements it.
* If the result is zero, the received frames are accepted; otherwise they are discarded.

## Cyclic Redundancy Check (CRC)

Cyclic Redundancy Check (CRC) involves binary division of the data bits being sent by a predetermined divisor agreed upon by the communicating system. The divisor is generated using polynomials.

* Here, the sender performs binary division of the data segment by the divisor. It then appends the remainder called CRC bits to the end of data segment. This makes the resulting data unit exactly divisible by the divisor.
* The receiver divides the incoming data unit by the divisor. If there is no remainder, the data unit is assumed to be correct and is accepted. Otherwise, it is understood that the data is corrupted and is therefore rejected.