**Codes**

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.

**Classification of binary codes**

The codes are broadly categorized into following four categories.

* Weighted Codes
* Non-Weighted Codes
* Reflective code
* Sequential 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. For example, the decimal number 539, 5 have weight 100, 3 have 10 and 9 have 1. For the 4-bit binary code each digit has weight 8,4,2 and1.

The codes 8421,2421,5211 are all weighted codes.

**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.

**Reflective Code**

The code that contain the complement of each code. For example, the complement of code 0 0 0 0 is 1 1 1 1 is the reflective code. The other reflective codes are 2421, 5211 and excess-3 code. 8421 is not a reflective code.

**Sequential Code**

A code is said to be sequential when each succeeding code is one binary number greater than the preceding code. 8421 and E-3 codes are the examples of this code.

**Alphanumeric Code**

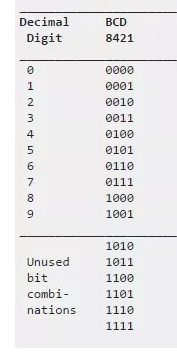
These codes contain both numeric numbers and alphabetic characters. Used for data transformation. Examples of these codes are ASCII (American Standard Code for Information Interchange) and EBCDIC (Extended Binary coded decimal interchange code)

**Error Detecting and Error Correcting Code**

When the digital data or information is transmitted from one system to other, an unwanted noise due to electric disturbance is added. This noise change the digital information i.e. 1 changes to 0 or 0 to 1. This error is corrected and detected by using some special codes called error detecting and error correcting codes.

**Binary Coded Decimal (BCD) code or 8 4 2 1 code**

It is a 4-bit binary code. It is positional code having weights 23 , 22 , 21 and 20. 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.



**Example**

Convert (73)10 and (894)10 into BCD

Sol

(73)10 = ( )BCD

dec 7 3

BCD 0 1 1 1 0 0 1 1

Thus (73)10 = ( 0 1 1 1 0 0 1 1)BCD

(894)10 = ( )BCD

Dec 8 9 4

BCD 1000 1001 0100

(894)10 = ( 1000 1001 0100 )BCD

The following table describes the relation between Decimal, Binary and **8421 BCD numbers**.

Decimal Number Binary Number 8421 BCD number

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

From 0 to 9 BCD and binary are same. From 10 to 15 BCD and binary are different. In this code there are six invalid BCD i.e. are **1010, 1011, 1100, 1101, 1110 and 1111**.

**NOTE 1100 0011 1010 1000**

Is the following BCD is correct or not?

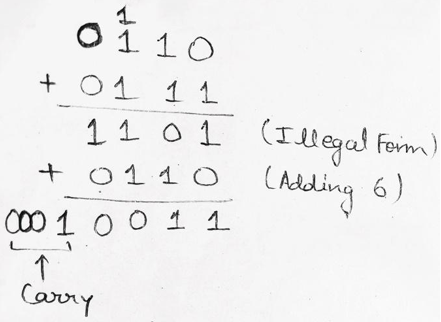
(1 0 1 0 1 1 0 0 0 0 1 1) BCD

The following is not correctbecause it contains invalid BCD i.e.1 1 0 0

**BCD addition**

**Example: Perform BCD Addition of 6 and 7.**

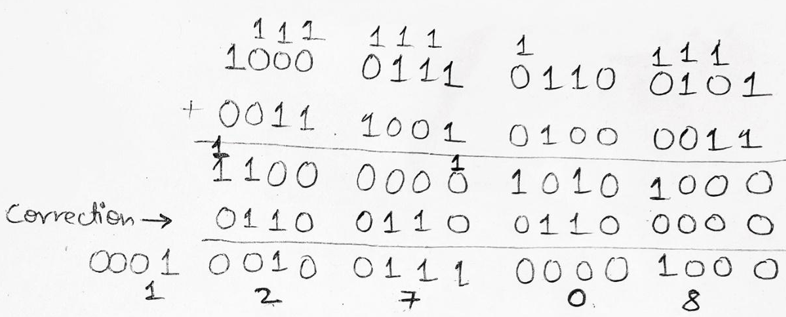
**Solution:** BCD representation of **6** is given as **0110** and for **7** it is **0111**.



**Example 2: Perform BCD Addition of 8765 and 3943.**

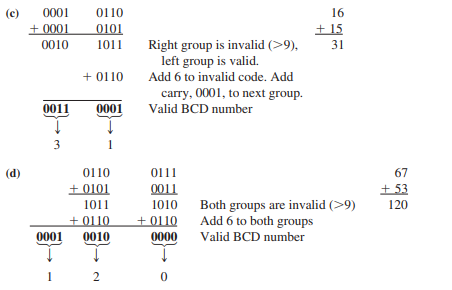
**Solution:**

BCD representation of **8765** is given as **1000 0111 0110 0101** and for **3943** it is **0011 1001 0 100 0011**.



Add the following BCD numbers





**BCD Subtraction**

**(541)10 – (216)10**

i) First change decimal to BCD

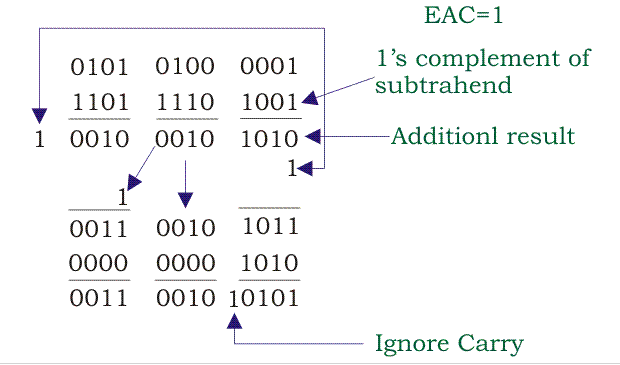
541)10 = 0101 0100 0001)BCD

216)10 = 0010 0001 0110)BCD

In this example 0010 0001 0110 is subtracted from 0101 0100 0001.

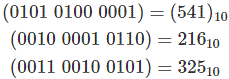
1’s compliment of the subtrahend is done, which is 1101 1110 1001

Added to 0101 0100 0001.



Therefore





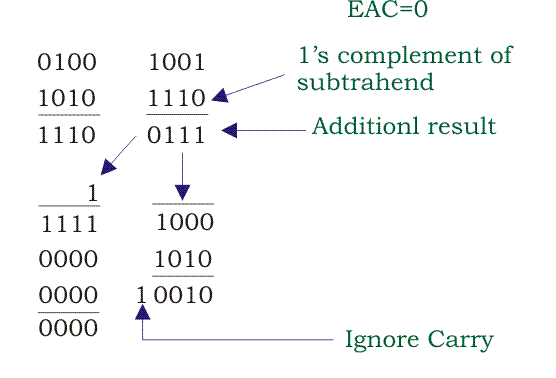
Example 2

Perform (49)10 –(51)10

The 1/s complement is

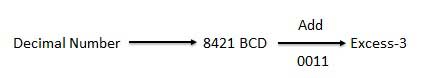
49) =0100 1001)BCD

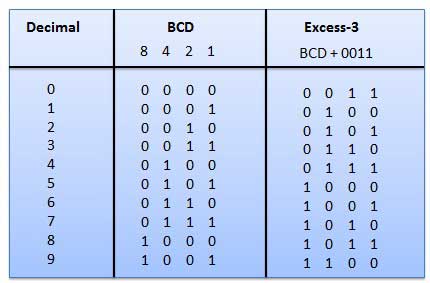
51) = 0101 0001)BCD



**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.it is a 4-bit code The excess-3 codes are obtained as follows −





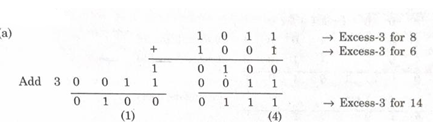
Like BCD there are six invalid E-3 codes. These codes are 0000,0001,0010,1101,1110 and 1111

**Excess-3 Addition**

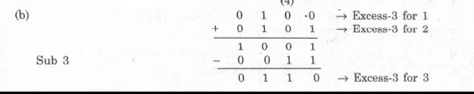
The following steps are used to add E-3 numbers

1. Write E-3 codes of the numbers to be added
2. If carry is = 1 add 3(0011) to the sum and if carry = 0 subtract 3(0011)

Example. Add (8)10 + (6)10



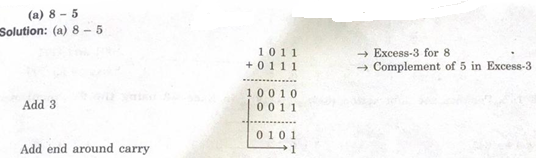
Add (1)10 + (2)10

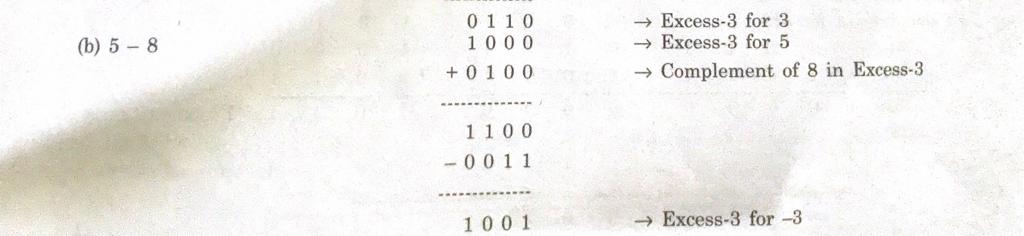


**E-3 Subtraction**

To perform E-3 subtraction, we have to

1. Complement of subtrahend
2. Add complemented subtrahend to minusend
3. If carry =1, result is positive. Add 3 and end around carry
4. If carry is zero, result is negative. Subtract 3





Examples

### Other 4-bit Code

### 2 4 2 1 code

* The weights of this code are 2, 4, 2 and 1.
* This code has all positive weights. So, it is a **positively weighted code**.
* it is a **self-complementing** code, i.e. its 1’s complement exits.

Dec number 2 4 2 1 code

1. 0 0 0 0
2. 0 0 0 1
3. 0 0 1 0
4. 0 0 1 1
5. 0 1 0 0
6. 1 0 1 1
7. 1 1 0 0
8. 1 1 0 1
9. 1 1 1 0
10. 1 1 1 1

### 8 4 -2 -1 code

* The weights of this code are 8, 4, -2 and -1.
* This code has negative weights along with positive weights. So, it is a **negatively weighted code**.
* It is an **unnatural BCD** code.
* It is a **self-complementing** code.

**Decimal Number 8 4 -2 -1 Code**

**0 0 0 0 0**

**1 0 1 1 1**

**2 0 1 1 0**

**3 0 1 0 1**

**4 0 1 0 0**

**5 1 0 1 1**

**6 1 0 1 0**

**7 1 0 0 1**

**8 1 0 0 0**

**9 1 1 1 1**

**Gray Code**

**Gray code** – also known as **Cyclic Code**, **Reflected Binary Code.** It is the non-weighted code and it is not arithmetic codes In gray code, while traversing from one step to another step only one bit in the code group changes. That is to say that two adjacent code numbers differ from each other by only one bit. 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.

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

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.

**Gray Code to Binary Conversion**

**Gray code to binary conversion** is again a very simple and easy process. Following steps can make your idea clear on this type of conversions.

1. The MSB of the binary number will be equal to the MSB of the given gray code.
2. Now if the second gray bit is 0, then the second binary bit will be the same as the previous or the first bit. If the gray bit is 1 the second binary bit will alter. If it was 1 it will be 0 and if it was 0 it will be 1.
3. This step is continued for all the bits to do **Gray code to binary conversion**. **1 1 0 0 1 1 0**



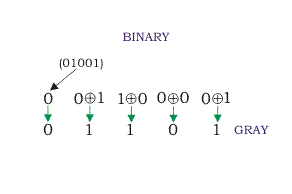
**Binary to Gray Code Conversion**

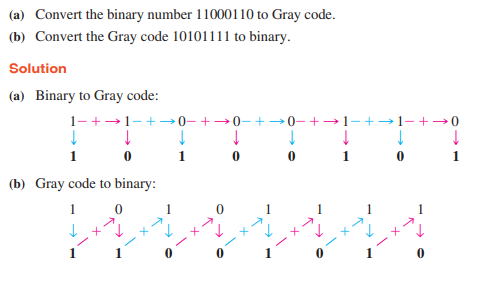
Say we have a binary number 01001 which we wish to convert to gray code. Let’s go through an example of how we would perform this conversion:

1. The MSB is kept the same. As the MSB of the binary is 0, the MSB of the gray code will be 0 as well (first gray bit)
2. Next, take the XOR of the first and the second binary bit. The first bit is 0, and the second bit is 1. The bits are different so the resultant gray bit will be 1 (second gray bit)

Next, take the XOR of the second and third binary bit. The second bit is 1, and the third bit is 0. These bits are again different so the resultant gray bit will be 1 (third gray bit)

Next, take the XOR of third and fourth binary bit. The third bit is 0, and the fourth bit is 0. As these are the same, the resultant gray bit will be 0 (fourth gray bit)





**Error Detecting and Error Correcting Code**

We know that the bits 0 and 1 corresponding to two different range of analog voltages. So, during transmission of binary data from one system to the other, the noise may also be added. Due to this, there may be errors in the received data at other system.

That means a bit 0 may change to 1 or a bit 1 may change to 0. We can’t avoid the interference of noise. But we can get back the original data first by detecting whether any errors present and then correcting those errors. For this purpose, we can use the following codes.

* Error detection codes
* Error correction codes

Therefore, to detect and correct the errors, additional bit is added to the data bits at the time of transmission. This method is called parity method.

Parity Code

It is easy to include one parity bit either to the left of MSB or to the right of LSB of original bit stream. There are two types of parity codes, namely even parity code and odd parity code based on the type of parity being chosen.

**Even Parity Code**

The 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 the data bits and even parity bit.

The following table shows the **even parity codes** corresponding to each 3-bit binary code. Here, the even parity bit is included to the right of LSB of binary 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 |

Here, the number of bits present in the even parity codes is 4. So, the possible even number of ones in these even parity codes are 0, 2 & 4.

### Odd Parity Code

The value of odd parity bit should be zero, if odd number of ones present in the binary code. Otherwise, it should be one. So that, odd number of ones present in **odd parity code**. Odd parity code contains the data bits and odd parity bit.

The following table shows the **odd parity codes** corresponding to each 3-bit binary code. Here, the odd parity bit is included to the right of LSB of binary 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 |

Here, the number of bits present in the odd parity codes is 4. So, the possible odd number of ones in these odd parity codes are 1 & 3.

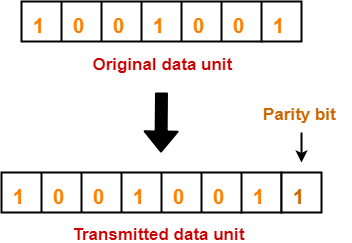
**Parity Check Example-**

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

Then,

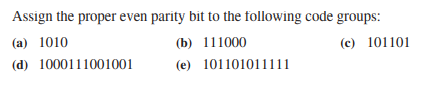
**At Sender Side-**

* 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.



**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.



Alphanumeric codes

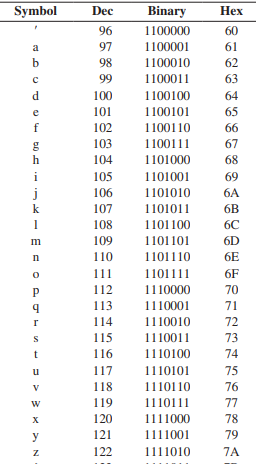
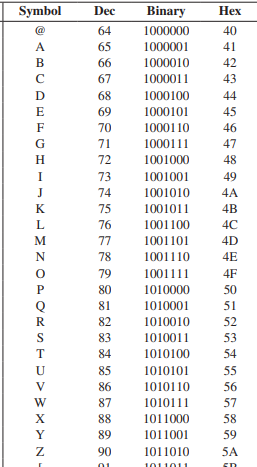
A binary digit or bit can represent only two symbols as it has only two states '0' or '1'. But this is not enough for communication between two computers because there we need many more symbols for communication. These symbols are required to represent 26 alphabets with capital and small letters, numbers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

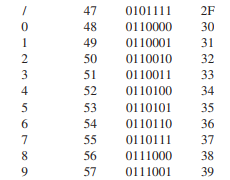
The alphanumeric codes are the codes that represent numbers and alphabetic characters. Mostly such codes also represent other characters such as symbol and various instructions necessary for conveying information. An alphanumeric code should at least represent 10 digits and 26 letters of alphabet i.e.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

* 1. American Standard Code for Information Interchange (ASCII).
  2. Extended Binary Coded Decimal Interchange Code (EBCDIC).

ASCII code is a 7-bit code and 128 characters whereas EBCDIC is an 8-bit code, 256 characters. ASCII code is more commonly used worldwide while EBCDIC is used primarily in large IBM computers.

**ASCII CODE**



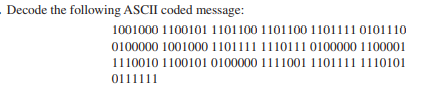


**Example**

Convert the following decimal to ASCII

1. 107 b) 3CD

**Example**



**Assignment**

1. **Add (569)10 and (687)10 in BCD**
2. **Subtract (645) and (319) using E-3**
3. Perform decimal addition in E-3 of following
4. (5)10+(4)10 b) (16)10+(29)1 0 c) (103)10+(287)10

iv) Perform decimal subtraction in E-3 of following

(645)10 - (319)10