CHAPTER 2

NUMBER SYSTEMS, OPERATIONS, AND CODES

2.1 Number Systems

A digital system can understand the positional number system only where there are 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:

A. The digit.
B. The position of the digit in the number.
C. The base of the number system (where the base is defined as the total number of digits available in the number system).

2.2 Types of the Number System in Computer

There are mainly four types of the number system in computer:

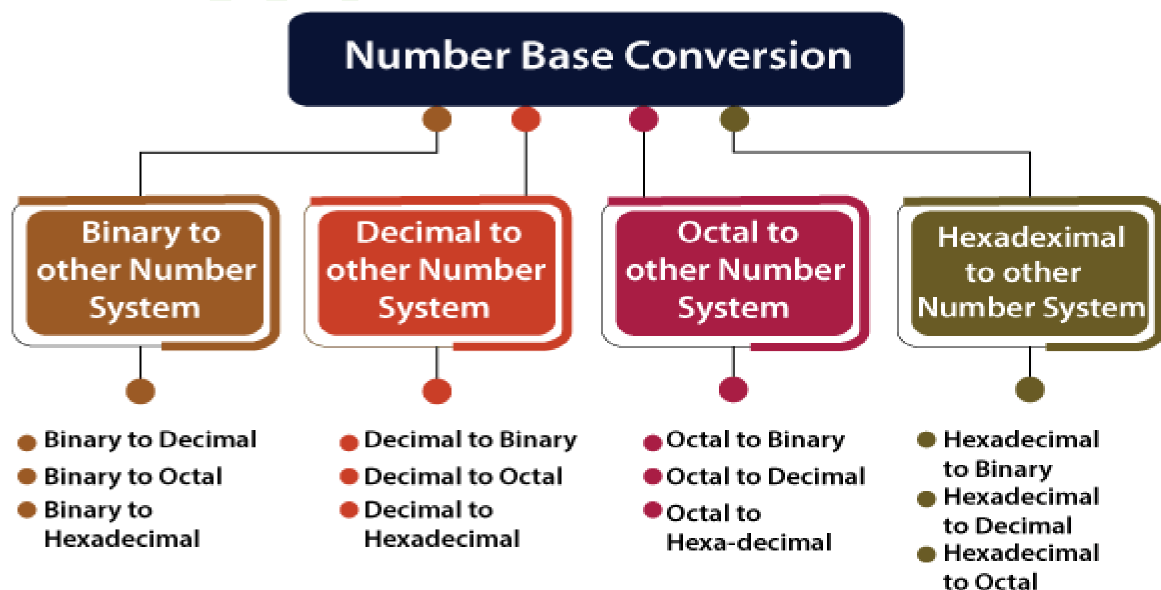
1. Binary Number System: The binary number system is the most fundamental number system used in computer science. It uses only two digits, 0 and 1, to represent all numbers and data.
2. Decimal Number System: The decimal number system is also used in computer science, but it is not as fundamental as the binary system. It uses ten digits, 0 through 9, to represent numbers.
3. Octal Number System: The octal number system uses eight digits, 0 through 7, to represent numbers. It is commonly used in computer programming and digital electronics.
4. Hexadecimal Number System: The hexadecimal number system uses 16 digits, including 0 through 9 and A through F, to represent numbers. It is often used in computer programming and digital electronics.

Number System

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

2.3 Conversion of the Number System

Each one of the four different types can be converted into the remaining three systems. There are the following conversions possible in Number System:



2.4 Binary-to-Decimal Conversion

Example: Convert the binary whole number 10101_2 to decimal.

Sol.

2^4	2^3	2^2	2^1	2^0
1	**0**	**1**	**0**	**1**

$10101_2 = ((1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10}$

$= (16 + 0 + 4 + 0 + 1)_{10} = 21_{10}$

Example: Convert the fractional binary number 0.1011 to decimal.

Sol.

Weight:	2^{-1}	2^{-2}	2^{-3}	2^{-4}
Binary number:	0.1	0	1	1

$0.1011 = 2^{-1} + 2^{-3} + 2^{-4} = 0.5 + 0.125 + 0.0625 = 0.6875$

2.5 Decimal-to-Binary Conversion

Example: Decimal number: $29_{10.}$ Calculating binary equivalent.

Sol.

Using the Repeated Division method.

29 / 2 = 14 Remainder 1 (LSB)

14 / 2 = 7 Remainder 0

7 / 2 = 3 Remainder 1

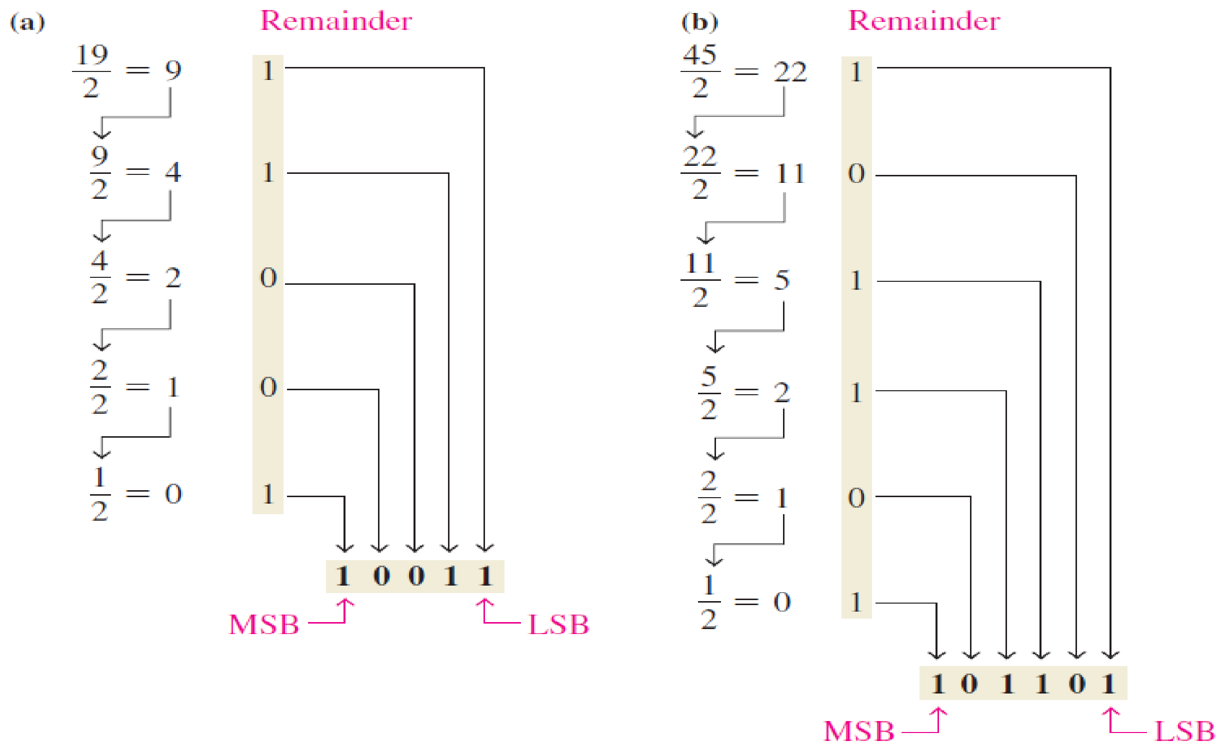
3 / 2 = 1 Remainder 1

1 / 2 = 0 Remainder 1 (MSB) $(29_{10}) = (11101_2)$

Dr. Intehaa Aljuboury

Example: Convert the following decimal numbers to binary: (a) 19 (b) 45.

Sol.



2.6 Binary Arithmetic

Binary arithmetic is essential in all types of digital systems. To understand these systems, you must know the basics of binary addition, subtraction, multiplication, and division.

2.7 Binary Addition

The four basic rules for adding binary digits (bits) are as follows:

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

Dr. Intehaa

Example (1): Add 11 + 1

Sol.

Carry Carry

	1	1	
	0	1	1
+	0	0	1
	1	0	0

In the right column, 1 + 1 = 0 with a carry of 1 to the next column to the left. In the middle column, 1 + 1 + 0 = 0 with a carry of 1 to the next column to the left. In the left column, 1 + 0 + 0 = 1.

Carry bits

1 + 0 + 0 = 01 Sum of 1 with a carry of 0

1 + 1 + 0 = 10 Sum of 0 with a carry of 1

1 + 0 + 1 = 10 Sum of 0 with a carry of 1

1 + 1 + 1 = 11 Sum of 1 with a carry of 1

Example (2): Add 111 + 11

Sol.

Carry Carry

		1	1	
		1	1	1
+			1	1
1	0	1	0	

Dr. Intehaa Aljuboury

2.8 Binary Subtraction

The four basic rules for subtracting bits are as follows:

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

When subtracting numbers, you sometimes have to borrow from the next column to the left. A borrow is required in binary only when you try to subtract a 1 from a 0. In this case, when a 1 is borrowed from the next column to the left, a 10 is created in the column being subtracted, and the last of the four basic rules just listed must be applied.

Example: Subtract 011_2 from 101_2.

Sol.

Left column:

When a 1 is borrowed,

a 0 is left, so 0 - 0 = 0.

Middle column:

Borrow 1 from next column

to the left, making a 10 in

this column, then 10 - 1 = 1.

```
  0
  1̸  10  1
- 0   1  1
---------
  0   1  0
```

Dr . Intehaa Aljuboury

2.9 Binary Multiplication

The four basic rules for multiplying bits are as follows:

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

Example**:** Perform the following binary multiplications:

$11_2 \times 11_2$

Sol.

$11_2 \times 11_2$

$$\begin{array}{r} 1\;1 \\ \times\quad 1\;1 \\ \hline 1\;1 \\ +\,1\;1 \\ \hline 1\;0\;0\;1 \end{array}$$

Partial products

2.10 Binary Division

Division in binary follows the same procedure as division in decimal

Example**:** Perform the following binary divisions: $110_2 \div 11_2$

Sol.

$$\begin{array}{r|l} & \;\;10 \\ 11 & 110 \\ & 11\; - \\ \hline & 000 \end{array}$$

Dr. Intehaa Aljuboury

2.11 One's complement representation

Complementing a number is a technique where all the digits of the number are reversed. A signed binary number can be represented in 1'complement by changing all the 0's to 1 and 1's to 0 (i.e. Change each bit in a number to get the 1's complement).

Example: Find the 1's complement of 00010011.

Sol.

Change each bit in a number to get the 1's complement. The 1's complement of a binary number is found by changing all 1s to 0s and all 0s to 1s, as illustrated below:

0	0	0	1	0	0	1	1	Binary number
↓	↓	↓	↓	↓	↓	↓	↓	
1	1	1	0	1	1	0	0	1's complement

2.12 Two's complement representation

The 2's complement of a binary number is found by adding 1 to the LSB of the 1's complement. The following steps are used to represent in the 2's complement method:

1. If the given number is positive, it is converted to its binary equivalent. If the given number is negative, the positive value of the number in binary is chosen.
2. Now the number is converted to its 1's complement form.
3. Finally, 1 is added to the 1's complement form.

Dr. Intenaa Ahboury

Example: Represent −2 in 2's complement.

Sol.

2	=	0	0	1	0	
		1	1	0	1	1's complement
+					1	results in 2's complement
		1	1	1	0	= -2

2.13 Digital Codes

Coding is a process of using a specific group of characters, alphabets, symbols, or numbers to represent specific information. The digital data are represented, stored, and transmitted as a group of binary bits. This group of binary bits is also called as the binary code. The binary codes represent numbers as well as alphanumeric letters. They are widely used for analyzing and designing digital circuit since only 0 and 1 are being used, which can be implemented easily. Many specialized codes are used in a digital system. They are as follows:

- Weighted codes
- Non-weighted codes
- Binary coded decimal code
- Alphanumeric codes
- Error detecting codes
- Error correcting codes

Weighted binary codes are those binary codes which have a specific weight for each position of the given number. Each decimal digit of the given number is represented as a group of 4-bit number. Several systems of the codes are used to

Dr. Intehaa Aljuboury

express the decimal digits 0 through 9. The weighted codes are 8421 code, 84-2-1 code, 2421 code, and 5211 code. The negative sign is represented as a bar on the number.

2.14 Binary Coded Decimal (BCD)

Binary coded decimal (BCD) is a way to express each of the decimal digits with a binary code. There are only ten code groups in the BCD system, so it is very easy to convert between decimal and BCD. Because we like to read and write in decimal, the BCD code provides an excellent interface to binary systems. Examples of such interfaces are keypad inputs and digital readouts. The binary coded decimal code, abbreviated as BCD, is a method that uses binary digits "0" and "1". ON state represents "1" and OFF state represents "0". Totally they can represent 16 numbers (0000–1111), bute in BCD code, only 10 of these are used. The remaining six code combinations are invalid in BCD. The commonly used BCD codes include 8421BCD code, 5421BCD code, 2421BCD code, and excess-3 code. Some codes In the table below such as 8421, 5421, and 2421 codes, are weighted codes, while some codes, such as excess-3 code, are nonweighted codes. In a weighted code, each digit position has a weight and the sum of all digits multiplied by their weight represents its corresponding decimal number. In a nonweighted code, no specific weights are assigned to bit position.

Dr. Intehaa Aljuboury

Table below commonly used BCD codes.

Decimal	8421 BCD	5421 BCD	2421 BCD	BCD2421 *	Excess-3 code
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 0 1 0	0 0 1 0	0 0 1 0	0 1 0 1
3	0 0 1 1	0 0 1 1	0 0 1 1	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0	0 1 0 1	1 0 1 1	1 0 0 0
6	0 1 1 0	1 0 0 1	0 1 1 0	1 1 0 0	1 0 0 1
7	0 1 1 1	1 0 1 0	0 1 1 1	1 1 0 1	1 0 1 0
8	1 0 0 0	1 0 1 1	1 1 1 0	1 1 1 0	1 0 1 1
9	1 0 0 1	1 1 0 0	1 1 1 1	1 1 1 1	1 1 0 0

2.15 The 8421 BCD code

The 8421 BCD code is the most commonly used BCD code. Each digit is called a bit. The designation 8421 indicates the binary weights of the four bits (23, 22, 21, 20). 4-bits are called nibble and are used to represent each decimal digit (0 through 9). This coding system has been used since the first computer. This coding system deals with decimal and binary numbers. The ease of conversion between 8421 code numbers and the familiar decimal numbers is the main advantage of this code. All you have to remember are the ten binary combinations that represent the ten decimal digits as shown in the table below. The 8421 code is the predominant BCD code, and when we refer to BCD, we always mean the 8421 code unless otherwise stated.

Decimal/BCD conversion.

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

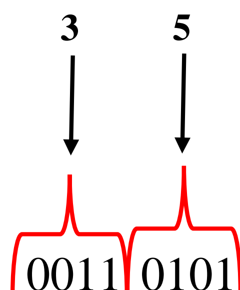
Note:

From four bits, sixteen numbers (0000 through 1111) can be represented but that, in the 8421 code, only ten of these are used. The six code combinations that are not used—1010, 1011, 1100, 1101, 1110, and 1111—are invalid in the 8421 BCD code. To express any decimal number in BCD, simply replace each decimal digit with the appropriate 4-bit code, as shown by Example:

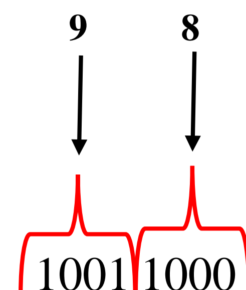
Example**:** Convert each of the following decimal numbers to BCD :**(a)** 35, (b) 98

Sol.

(a)



(b)



Dr. Intehaa Aliubouri

Related Problem

Convert the decimal number 9673 to BCD

2.16 The 5421 and 2421 BCD codes

The 5421 and 2421 BCD codes are weighted codes too, and their weights from MSB to LSB are 5, 4, 2, 1 and 2, 4, 2, 1, respectively. For the 5421 and 2421 BCD codes, one decimal digit may be represented by different binary numbers. For instance, 5 can be represented by either 1011 or 0101 in the 2421 BCD code; likewise, 5 can be represented by either 1000 or 0101 in the 5421 BCD code. However, the 5421 and 2421 BCD codes in the table listed in section 2.14 have been generally accepted, and other forms are no longer used. In addition, it can be observed that in BCD 2421*, the code for decimal 0 is the complement of the code for decimal 9; this also holds true for the codes for decimal 1 and 8, 2 and 7, 3 and 6, and 4 and 5. This property is called the nine's complement of a decimal number, that is, bitwise complementation of a code will produce the nine's complement of the decimal number, which makes hardware implementation of arithmetic operations much simpler in digital systems.

2.17 The Excess-3 Code

The excess-3 code is a nonweighted code used to express decimal numbers. The Excess-3 code is derived from the 8421 BCD code adding $(0011)_2$ or $(3)_{10}$ to each code in 8421 BCD. When the addition of two excess-3 codes produces a carry, the carry signal can be directly obtained from the MSB. In addition, excess-3 code also has the property of the nine's complement of the decimal number, and has been commonly used in the arithmetic operation circuitry of BCD codes. There are only ten codes in the BCD system, and so it is very easy to convert between decimal number and BCD. To convert any decimal number in BCD,

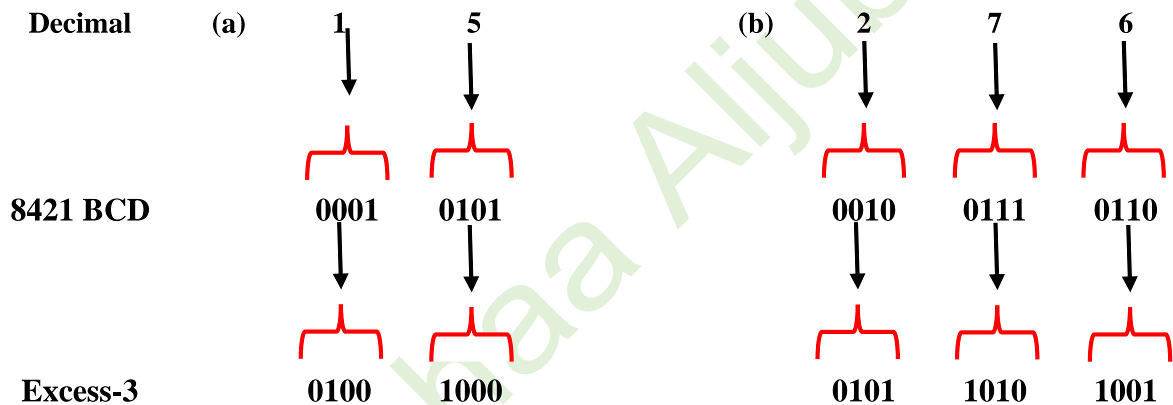
Dr. Intenaa Aljuboury

simply replace each decimal digit with the corresponding four-bit binary code. To convert a BCD number to a decimal, you can break the code into groups of four bits, starting from the LSB, and then write the corresponding decimal digit represented by each four-bit group.

Example**:** Covert the following decimal numbers to the 8421 BCD codes and the excess-3 codes, respectively.

(a) 15 **(b)** 276

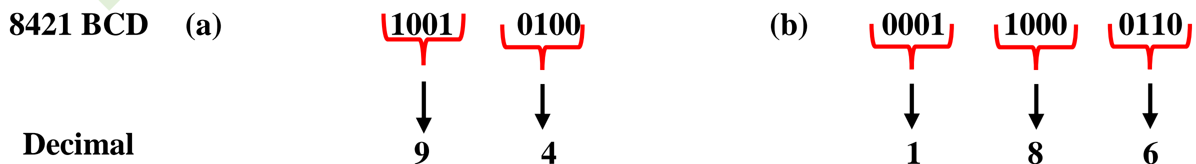
Sol.



Example**:** Covert the following the 8421 BCD codes to decimal numbers.

(a) 10010100 **(b)** 000110000110

Sol.



Dr. Intehaa Aljuboury

Advantages of BCD Codes

- BCD coding is similar to the binary equivalent of decimal numbers 0–9.
- BCD has no limitation for number size.
- It is easier to convert decimal numbers from or to BCD than to binary form.

Disadvantages of BCD Codes

- Each decimal number requires four bits to be represented in the BCD code.
- Arithmetic operations in BCD or weighted codes are much complicated as it deals with more number of bits and also it has different set of rules.
- BCD is less efficient than binary.

Applications

- They are mainly used to give instructions to microcomputers.
- It is used for interfacing devices.
- It is used in the transfer of information from keyboards to computer displays and printers.

2.18 Error Codes

In this section, three methods for adding bits to codes to detect a single-bit error are discussed. The parity method of error detection is introduced, and the cyclic redundancy check is discussed. Also, the Hamming code for error detection and correction is presented.

2.19 Parity Method for Error Detection

Many systems use a parity bit as a means for bit error detection. Any group of bits contain either an even or an odd number of 1s. A parity bit is attached to a group of bits to make the total number of 1s in a group always even or always odd. An even parity bit makes the total number of 1s even, and an odd parity bit makes the total odd.

A given system operates with even or odd parity, but not both. For instance, if a system operates with even parity, a check is made on each group of bits received to make sure the total number of 1s in that group is even. If there is an odd number of 1s, an error has occurred.

Dr. Intehaa Alluboury

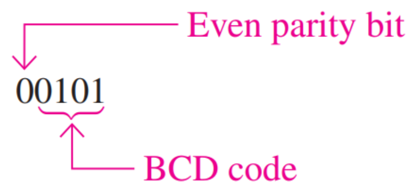
As an illustration of how parity bits are attached to a code, table below lists the parity bits for each BCD number for both even and odd parity. The parity bit for each BCD number is in the *P* column.

Even Parity		Odd Parity	
P	**BCD**	***P***	**BCD**
0	0000	1	0000
1	0001	0	0001
1	0010	0	0010
0	0011	1	0011
1	0100	0	0100
0	0101	1	0101
0	0110	1	0110
1	0111	0	0111
1	1000	0	1000
0	1001	1	1001

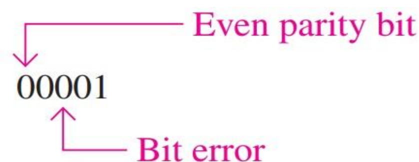
The parity bit can be attached to the code at either the beginning or the end, depending on system design. Notice that the total number of 1s, including the parity bit, is always even for even parity and always odd for odd parity.

2.20 Detecting an Error

A parity bit provides for the detection of a single bit error (or any odd number of errors, which is very unlikely) but cannot check for two errors in one group. For instance, let's assume that we wish to transmit the BCD code 0101. (Parity can be used with any number of bits; we are using four for illustration.) The total code transmitted, including the even parity bit, is



Now let's assume that an error occurs in the third bit from the left (the 1 becomes a 0).



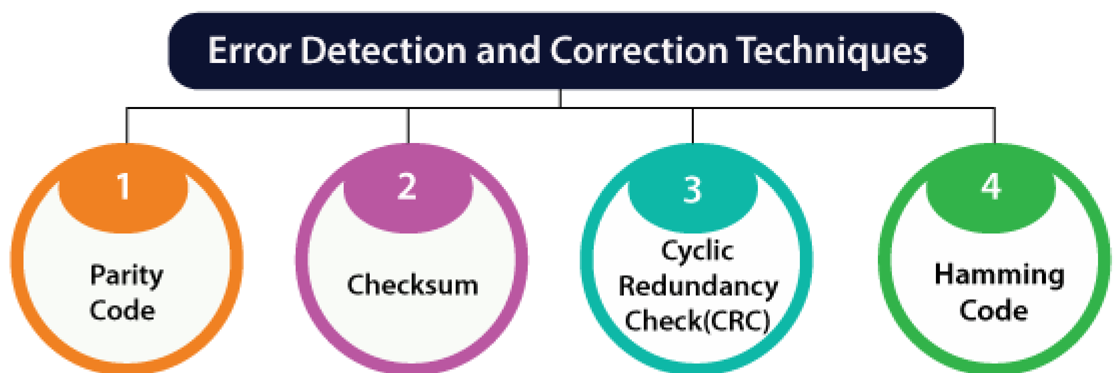
Dr. Intehaa Aljuboury

When this code is received, the parity check circuitry determines that there is only a single 1 (odd number), when there should be an even number of 1s. Because an even number of 1s does not appear in the code when it is received, an error is indicated.

An odd parity bit also provides in a similar manner for the detection of a single error in a given group of bits.

2.21 Error Codes

It is impossible to avoid the interference of noise, which causes errors in the received binary data at other systems. The bits of the data may change (either 0 to 1 or 1 to 0) during transmission. Therefore error detection and correction code play an important role in the transmission of data from one source to another. There are four method for detect errors:



2.22 Parity Method for Error Detection Binary

There are two parity methods, even and odd. In the even parity method, the value of the bit is chosen so that the total number of 1s in the transmitted signal, including the parity bit, is even. Similarly, with odd parity, the value of the bit is chosen so that the total number of 1s is odd.

Even parity (ep): makes the total no. of 1´s even

Odd parity (op): makes the total no. of 1´s odd

Example**:** Assign the proper even parity bit to the following code groups: (a) 1010 (b) 111000 (c) 101101 (d) 1000111001001 (e) 101101011111

Sol.

Make the parity bit either 1 or 0 as necessary to make the total number of 1s even. The parity bit will be the left-most bit (color).

(a) **0**1010 (b) **1**111000 (c) **0**101101 (d) **0**1000111001001 (e) **1**101101011111

Example**:** An odd parity system receives the following code groups: 10110, 11010, 110011, 110101110100, and 1100010101010. Determine which groups, if any, are in error.

Sol.

Since odd parity is required, any group with an even number of 1s is incorrect.
The following groups are in error:
110011 and 1100010101010.

2.23 Checksum Method

Checksums are similar to parity bits except, the number of bits in the sums is larger than parity and the result is always constrained to be zero. That means if the checksum is zero, an error is detected. A checksum of a message is an arithmetic sum of codewords of a certain length. The sum is stated by means of 1's complement and stored or transferred as a code extension of the actual code word. At the receiver, a new checksum is calculated by receiving the bit sequence from the transmitter.

Checksum of messages = M1 + M2 + M3 + M4 + … = 0 0 0 0 0

Example**:** If k = 4 and n = 8, find the checksum of four segments: (10110011 10101011 01011010 11010101), along with each transmitted message, the checksum of all the messages are also transmitted.

Dr. Intehaa Aljuboury

Sol.

k (number of messages) = 4, n (number of bits) = 8

At sender side	At receiver side
10110011	10110011
10101011	10101011
01011110	01011110
1	1
01011111	01011111
01011010	01011010
10111001	10111001
11010101	11010101
10001110	10001110
1	1
Sum: 10001111	10001111
Checksum: 01110000	01110000
	Sum: 11111111
	Complement = 00000000
	Conclusion = Accept data

At sender side *At receiver side*

2.24 Cyclic Redundancy Check (CRC)

In cyclic redundancy code (CRC), the transmitted bit sequence is:

TX	RX
Series data + CRC data	data + CRC data

The transmitted CRC is compared with the RX CRC and if they match then there are no errors. If they do not match then error is there.

Example: Determine the transmitted CRC for the following byte of data (D) and generator code (G). Verify that the remainder is 0. D: 11010011, G: 1010

Sol.
Since the generator code has four data bits, add four 0s (blue) to the data byte. The appended data (D´) is

D´ =110100110000

Divide the appended data by the generator code (red) using the modulo-2 operation until all bits have been used.

Dr. Intehaa Alfoury

$$\frac{D'}{G} = \frac{110100110000}{1010}$$

```
110100110000
1010
 1110
 1010
  1000
  1010
    1011
    1010
       1000
       1010
        100
```

Remainder = 0100. Since the remainder is not 0, append the data with the four remainder bits (blue). Then divide by the generator code (red). The transmitted CRC is **110100110100**.

```
110100110100
1010
 1110
 1010
  1000
  1010
    1011
    1010
       1010
       1010
         00
```

Remainder = 0

Example**:** During transmission, an error occurs in the second bit from the left in the appended data byte generated in Example above. The received data is D´ = 100100110100 Apply the CRC process to the received data to detect the error using the same generator code (1010).

Sol.

```
100100110100
1010
  1100
  1010
   1101
   1010
    1111
    1010
     1010
     1010
       0100
```

Remainder = 0100. Since it is not zero, an error is indicated.

Dr. Intehaa Aljubouri

2.25 Hamming Code (HC)

The Hamming code is used to detect and correct a single-bit error in a transmitted code. To accomplish this, 4 redundancy bits are introduced in a 7-bit group of data bits. These redundancy bits are interspersed at bit positions 2^n (n = 0, 1, 2, 3) within the original data bits. At the end of the transmission, the redundancy bits have to be removed from the data bits. A recent version of the Hamming code places all the redundancy bits at the end of the data bits, making their removal easier than that of the interspersed bits.

7-bits Hamming code is used commonly

Parity bits = 3 bits (p_1, p_2, p_3)

Data bits = 4 bits (d_1, d_2, d_3, d_4)

$2^P \geq n + P + 1$ Position of p from (2^0, 2^1, 2^2) 2^n where {n= 0, 1, ……., n}

$p_1 = d_1 \oplus d_2 \oplus d_4$ All d except d_3
$p_2 = d_1 \oplus d_3 \oplus d_4$ All d except d_2
$P_3 = d_2 \oplus d_3 \oplus d_4$ All d except d_1

Example: Determine the Hamming code word (7, 4) for the transmitted data 1011 over the noisy communication channel.

Sol.

Step 1: Find parity bit to constructing bit location table.

Bits position	1	2	3	4	5	6	7
Status	**P_1**	**P_2**	**d_1**	**P_3**	**d_2**	**d_3**	**d_4**
Code word	**0**	**1**	**1**	**0**	**0**	**1**	**1**

$p_1 = d_1 \oplus d_2 \oplus d_4 \rightarrow 1 \oplus 0 \oplus 1 = \mathbf{0}$
$p_2 = d_1 \oplus d_3 \oplus d_4 \rightarrow 1 \oplus 1 \oplus 1 = \mathbf{1}$
$P_3 = d_2 \oplus d_3 \oplus d_4 \rightarrow 0 \oplus 1 \oplus 1 = \mathbf{0}$

Step 2: Enter data in the syndrome.

$A = p_1 \oplus d_1 \oplus d_2 \oplus d_4 \rightarrow 0 \oplus 1 \oplus 0 \oplus 1 = \mathbf{0}$
$B = p_2 \oplus d_1 \oplus d_3 \oplus d_4 \rightarrow 1 \oplus 1 \oplus 1 \oplus 1 = \mathbf{0}$
$C = P_3 \oplus d_2 \oplus d_3 \oplus d_4 \rightarrow 0 \oplus 0 \oplus 1 \oplus 1 = \mathbf{0}$
CBA = **000** No errore

Dr. Intehaa Aljuboury