Chapter – 2

Number Systems

2.1 Introduction.

This chapter is dedicated to a explain computer arithmetic. Our goal is to introduce the fundamental issues related to the arithmetic operations and number conversion used to support computation in computers. Our coverage starts with an introduction to number systems. In particular, we introduce issues such as number representations and base conversion. This is followed by a discussion on integer arithmetic. In this regard, we are performing Different number systems addition, subtraction, multiplication, and division, and then we find the complements of number systems for negative representation, and convert decimal code in to different binary Weighted and non-weighted codes. Finally we discussed error detecting and error correcting code for finding and correct the error of transmitted data in to the network.

2.1 Number System: A number system uses a specific radix (base). Radices that are power of 10 are widely used in general mathematical and statistical calculation, while 2 are widely used in digital systems. These radices include binary (base 2), quaternary (base 4), octagonal (base 8), and hexagonal (base 16). The base 2 binary system is dominant in computer systems. For each number representation it has important objectives.

- Any number system position of symbols, base, and radix of a number system is presented.
- Each radix is indicating the representation of the number of symbols.
- In binary number system negative number are represented by sign bit, while in decimal system it represented by '-'sign.
- The decimal number use bi-stable devices coded with BCD system.
- Any number always read from left to right digit, where left most beat is MSB (Most significant Bit) and right most beat is (Least Significant Bit).

2.2 Decimal number System

Any number can be represented with the base so, it identify that the radix of that digit, as we know that radix is represented the number of symbols to interprets that digits, i.e. when we write 196 in decimal system we should write it in 196_{10} but gradually we use decimal number so no need to write base 10 with the digit.

In Decimal number system, we use 10 symbols are use to represent any number (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) .which are call a Arabic numerals. If we use roman number system we use I, II , V ,X etc. for number 10 we use X and for 13 we use XIII now when we multiply this X * XIII it is very small digit still it is difficult to interpretation of other number system.

Here the important of decimal number system is 10 symbols and the positional notational system for count any desired figure. If we multiply 15 by 13, result is 195. Here it can be seen very clear that, it is spoken as "One Hundred Ninety Five "this number is contradiction of 1*100 + 9*10 + 5*1. The general rules for representing numbering decimal system by using positional notation (n) is as follows:

$A_{n-1} * 10^{n-1} + A_{n-2} * 10^{n-2} + A_{n-3} * 10^{n-3} \ldots\ldots\ldots\ldots + A_1 * 10^1 + A_0 * 10^0$

Examples 2.1 Here some Decimal numbers are represented in polynomial form with base (10)

1. $9763_{10} = 9 * 10^3 + 7 * 10^2 + 6 * 10^1 + 3 * 10^0 = 9000+700+60+3$
2. $635.25_{10} = 6 * 10^2 + 3 * 10^1 + 5 * 10^0 . 2 * 10^{-1} + 5 * 10^{-2} = 600+50+3+.20+.05$
3. $-56.75_{10} = -(5 * 10^1 + 6 * 10^0 . 7 * 10^{-1} + 5 * 10^{-2}) = -(50+6+0.70+0.05)$
4. $1010_{10} = 1 * 10^3 + 0 * 10^2 + 1 * 10^1 + 0 * 10^0 = 1000+000+10+0$

Here, $\log_{10} 5 = 10000$

2.3 Bi-stable Devices

The basic element in computer or in any electronics devices which has two stages either it is on or it is off i.e. switches and relays, operation of switches and relays are defines they are bi-stable devices. The switch is either on or off (1) or (0). The principal circuit elements in more modern computers are transistor, which also has on or off state. The electric bulb has also two states on or off. So electronics circuit has also manufacturing number of bi-stable devices which has two possible states.

Computer devices (Machine) can understand two state 0 or 1, this binary language is also known as a machine language, and the o and 1 known as bit. Computer can perform operation on binary data so It accept binary data and output in binary format, it store data on disk or in chips also in binary so we can say that computer is a bi-stable device. Binary information in digital computer is represented by physical quantity called *signals*.

2.4 Binary, Octal and Hexadecimal numbers.

ıĕI

Decimal r=10	Binary r=2	Octal r=8	Hexadecimal r=16
Ĭ	Ĭ	Ĭ	Ĭ
I	I	I	I
ı	IĬ	ı	ı
Ŏ	II	Ŏ	Ŏ
ŏ	IĬĬ	ŏ	ŏ
	IĬI		
	IIĬ		
	III		

Decimal r=10	Binary r=2	Octal r=8	Hexadecimal r=16
	IĬĬĬ	IĬ	
	IĬĬI	II	
IĬ	IĬIĬ	Iı	
II	IĬII	IŎ	
Iı	IIĬĬ	Iŏ	
IŎ	IIĬI	I	
Iŏ	IIIĬ	I	
I	IIII	I	
I	IĬĬĬĬ	IĬĬ	IĬ

The polynomial representation of the earlier number is:

$$N = \sum_{i=-m}^{n} a_i * r^i$$

Consider an integer with n digits. A finite range of values can be represented by this integer. The smallest value in this range is 0 and corresponds to each digit of the n-digit integer being equal to 0. When each digit corresponds in value to r-1, the highest digit in the number system, the n-digit number attains the highest value in the range. This value is equal to r^n-1. *Table 2.1* lists the first few numbers in various systems. We will discuss binary, octal, and hexadecimal systems next.

Binary Number: In this number system, the radix is 2 and the two allowed digits are 0 and 1. BInary digi**T** is abbreviated as BIT. A typical binary number is shown in the positional notation are as below.

Example 2.2

N = $(1\ 1\ 0\ 1\ 0\ .\ 1\ 1\ 0\ 1\ )_2$
$2^4\ 2^3\ 2^2\ 2^1\ 2^0\ .2^{-1}2^{-2}2^{-3}2^{-4}$ -Weight
16 8 4 2 1 . ½ ¼ 1/8 1/16 -Weight in Decimal

Notes:

- Weights double for each move to left from the binary point
- Weights are halved for each move to right from the binary point

The polynomial form of given number is:

$$N = 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 + 1*2^{-1} + 1*2^{-2} + 0*2^{-3} + 1*2^{-4}$$

$$= 16+8+0+2+0+1/2+1/4+0+1/16$$

$$=26 + ½ + ¼ + {}^{1}/_{16} \quad =(26\ {}^{13}/_{26}\ )_{10}$$

Octal Number system : In this number system, the radix is 8 and the eight allowed digits are 0,1,2,3,4,5,6,and 7. We show the first eight octal numbers are equal to decimal number but eight numbers for octal is representing 10. Octal number has base of 8, octal number is shown in the positional notation are as below.

Example 2.3

N = $(1\ 6\ 5\ .\ 5\ 3\ )_8$
$8^2\ 8^1\ 2^0\ .8^{-1}8^{-2}$ -Weight
64 8 1 . 1/8 1/64 -Weight in Decimal

Notes:

- Weights double for each move to left from the binary point
- Weights are halved for each move to right from the binary point

$$N = 1*8^2 + 6*8^1 + 5*8^0 + 5*8^{-1} + 3*8^{-2} + 0*2^{-3} + 1*2^{-4}$$

$$= 64+48+5+5/8+3/64 =(117\ {}^{43}/_{64}\ )_{10}$$

Hexadecimal Number system: In this number system, the radix is 16 and the 16 allowed digits are 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,and F. We show the first 16 hexadecimal numbers in table there other A to F is used to represent 10, 11, 12, 13, 14, and 15 numbers respectively. Hexadecimal number is shown in the positional notation are as below.

Example 2.4

N = $(3\ \ A\ \ 8\ )_{16}$
$16^2\ 16^1\ 16^0$ -Weight
256 16 1 -Weight in Decimal

Notes:

- Weights double for each move to left from the hexadecimal point
- Weights are halved for each move to right from the hexadecimal point

2.5 Number base conversions

Any number in base-n, the least significant digit holds the units (i.e. n^0 the next the number of n's (i.e. n^1) and the next the number of n^2. So in general the value of the three-digit number abc in base n, i.e. abc,,, is given by

abc = (a x n^2) + (b x $n^{1'}$) + (c x n^0)

To convert numbers from a non-decimal system to decimal, we simply expand the give n number as a polynomial and evaluate the polynomial using decimal arithmetic c, as show n in Examples 2.1 through 2.4. When a decimal number is converted to any other system, the integer and fraction portions of the number are handled separately. The radix divide technique is used to convert the integer portion, and the radix multiply technique is used for the fraction portion.

2.5.1 Conversion from Decimal to n-base.

In binary code this means that successive digits hold the number of 1' s, 2' s, 4's, 8's etc., that is quantities represented by 2 raised to successively higher powers. In the above text and examples where it was necessary to use numeric representation (e.g. of 10^2 = 100 and 2^3 =8) then base-10 was used. It should be appreciated that any base could have been chosen, with base-10 selected simply because it is the one we are most familiar with. Remember that any written number is basically a shorthand way of recording the total number of units with successive single digits representing larger and larger quantities (i.e. l's, 10's, 100's etc., for base- 10).

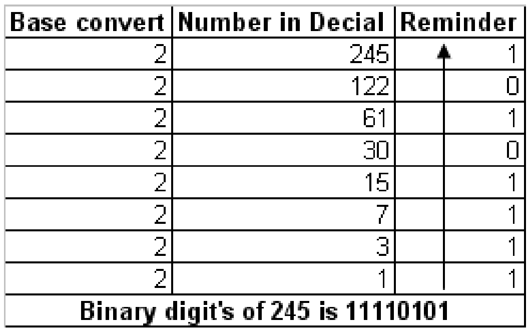
- **Radix Divide Technique**

1. Divide the given integer successively by the required radix, noting the remainder at each step. The quotient at each step becomes the new dividend for subsequent division. Stop the division process when the quotient becomes zero.

2. Collect the remainders from each step (last to first) and place them left to right to form the required number. The following examples illustrate the procedure.

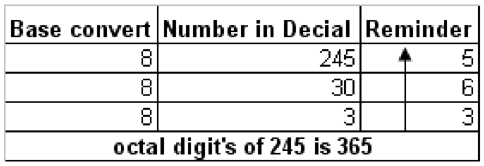
Examples 2.5 **Convert Decimal Number to Binary, Octal and Hexadecimal**

1). 245_{10} to binary



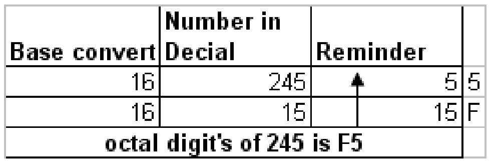
Base convert	Number in Decial	Reminder	
2	245	↑	1
2	122		0
2	61		1
2	30		0
2	15		1
2	7		1
2	3		1
2	1		1
Binary digit's of 245 is 11110101			

2). 245_{10} to Octal



Base convert	Number in Decial	Reminder	
8	245	↑	5
8	30		6
8	3		3
octal digit's of 245 is 365			

3) 245_{10} to Hexadecimal



Base convert	Number in Decial	Reminder		
16	245	↑	5	5
16	15		15	F
octal digit's of 245 is F5				

- **Radix Multiply Technique**

As we move each position to the right of the radix point, the weight corresponding to each bit in the binary fraction is halved. The radix multiply technique uses this fact and multiplies the given decimal number by 2 (i.e., divides the given number by half) to obtain each fraction bit. The technique consists of the following steps:

1. Successively multiply the given fraction by the required base, noting the integer portion of the product at each step. Use the fractional part of the product as the multiplicand for subsequent steps. Stop when the fraction either reaches 0 or recurs.

2. Collect the integer digits at each step from first to last and arrange them left to right.

If the radix multiplication process does not converge to 0, it is not possible to represent a decimal fraction in binary exactly, and then depends on the number of bits used to represent the fraction. Some examples follow.

Examples 2.6 **Convert Floating Decimal Number to Binary, Octal and Hexadecimal.**

1). 0.675 $_{10}$ to binary

Base convert	Number in Decial	Integer
2	0.675	1
2	0.3500	0
2	0.7000	1
2	0.4000	0
2	0.8000	1
2	0.6000	1
2	0.2000	0
2		
Binary digit's of 0.675 is (0.1010110)		

2). 0.545$_{10}$ to Octal

Base convert	Number in Decial	Integer
8	0.5450	4
8	0.3600	2
8	0.8800	7
8	0.0400	0
8	0.3200	2
8	0.5600	4
8	0.4800	3
Binary digit's of 0.545 is (0.427024..)		

3) 0.480$_{10}$ to Hexadecimal

Base convert	Number in Decial	Integer	
16	0.4800	7	
16	0.6800	10	A
16	0.8800	14	E
16	0.0800	1	
16	0.2800	4	
16	0.4800	7	
16	0.6800	10	A
Binary digit's of 0.48 is (0.7AE14....)			

When a number is converted from base x to base y, the number in base x is divided (or multiplied) by y in base x arithmetic. Because of our familiarity with decimal arithmetic, these methods are convenient when x = 10. In general, it is easier to convert a base x number to base y ($x \neq 10$, $y \neq 10$) by first converting the number to decimal from base x and then converting that decimal number to base y (i.e., $(N)x \rightarrow (?)_{10} \rightarrow (?)y$), as shown by the following example.

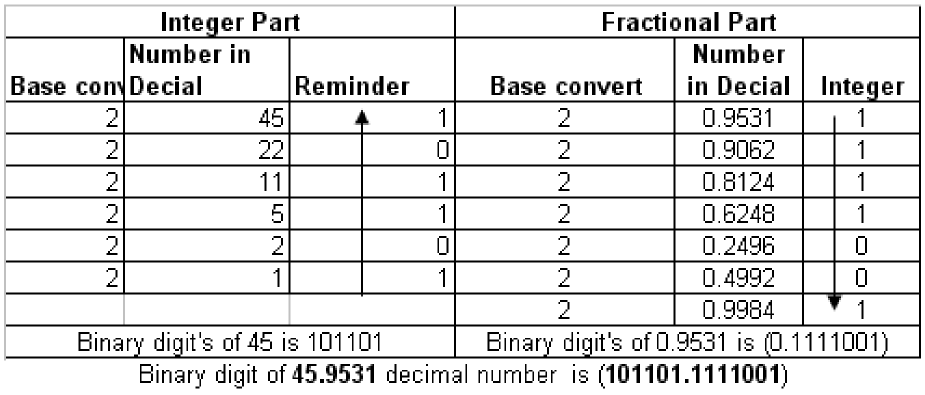
Examples 2.7 **Convert $(48.75)_8$ Digit in to Decimal and convert in Binary and hexadecimal.**

$$N_8 = \begin{array}{llllll} (4 & 8 & . & 7 & 5)_8 & \\ 8^1 & 8^0 & . & 8^{-1} & 8^{-2} & \text{-Weight} \end{array}$$

N_{10} $= 4 * 8^1 + 8 * 8^0 + 7 * 8^{-1} + 5 * 8^{-2}$

$= 32+8+5+7/8+5/64 = (45\ {}^{61}/_{64})_{10} = 45.9531_{10}$

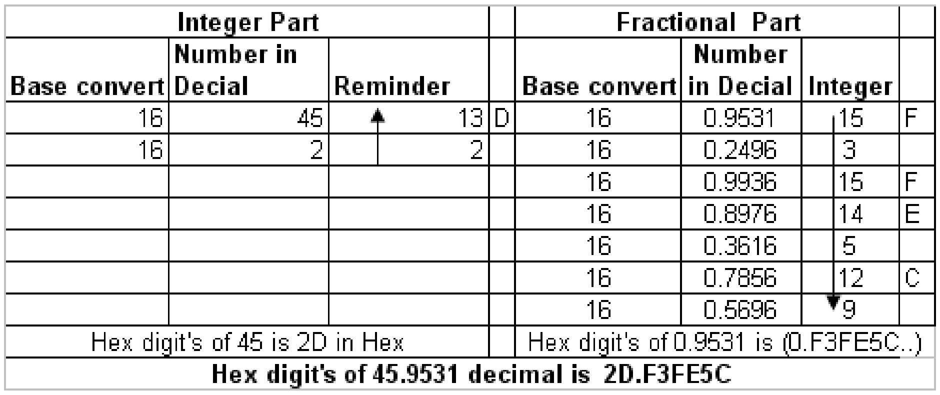
N_2 $= (101101.1111001)_2$



Integer Part			Fractional Part		
Base con	Number in Decial	Reminder	Base convert	Number in Decial	Integer
2	45	1	2	0.9531	1
2	22	0	2	0.9062	1
2	11	1	2	0.8124	1
2	5	1	2	0.6248	1
2	2	0	2	0.2496	0
2	1	1	2	0.4992	0
			2	0.9984	1
Binary digit's of 45 is 101101			Binary digit's of 0.9531 is (0.1111001)		

Binary digit of **45.9531** decimal number is (**101101.1111001**)

N_{16} $= (101101.1111001)_2$



Integer Part					Fractional Part			
Base convert	Number in Decial	Reminder			Base convert	Number in Decial	Integer	
16	45	13	D		16	0.9531	15	F
16	2	2			16	0.2496	3	
					16	0.9936	15	F
					16	0.8976	14	E
					16	0.3616	5	
					16	0.7856	12	C
					16	0.5696	9	
Hex digit's of 45 is 2D in Hex					Hex digit's of 0.9531 is (0.F3FE5C..)			
Hex digit's of 45.9531 decimal is 2D.F3FE5C								

2.5.2 Radix 2k Conversion

Each of the eight octal digits can be represented by a 3-bit binary number. Similarly, each of the 16 hexadecimal digits can be represented by a 4-bit binary number. In general, each digit of the base x number system, where x is an integral power k of 2, can be represented by a k-bit binary number.

In converting a base x number to base y, if x and y are both integral powers of 2, the base x number can first be converted to binary, and this can in turn be converted to base y by inspection. This conversion procedure is called the base 2k conversion.

Examples 2.8 **Convert $(32A5F.6A)_{16}$ Digit in to octal and Binary digit.**

N = **(3 2 A 5 F . 6 A)** $_{16}$

Convert all digits in to binary 2^4

3	2	A	5	F	.	6	A
0011	0010	1011	0101	1111	.	0110	1010

So the binary of $32A5F.6A_{16}$ is 00110010101101011111.01101010

Now, convert this binary digit in Octal make all bits in group of 3-bits from the radix so Integer and Decimal both make different pairs if require then add 0 to MSB or LSB i.e. here fractional part of eight bit so add 0 in to the LSB and integer part of 20 bits so add 0 to MSB fractional part.

N_8 =	000	110	010	101	101	011	111 . 011	010	100
=	0	6	2	5	5	3	7 . 3	2	4

N_8 = 625537.324

2.6 Binary Addition, Subtraction, Multiplication and Division

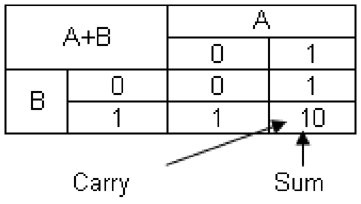
Arithmetic in all other number systems follows the same general rules as in decimal. Binary arithmetic is simpler than decimal arithmetic since only two digits (0 and 1) are involved. In this section, we will describe binary arithmetic in detail.

In the so-called fixed-point representation of binary numbers in digital systems, the radix point is assumed to be either at the right end or the left end of the field in which the number is represented. In the first case, the number is an integer, and in the second it is a fraction.

2.6.1 Binary Arithmetic:

Addition

In Table 2.2, note that 0 + 0=0, 0 + 1=1, 1 + 0=1, and 1 + 1=10. Thus, the addition of two 1s results in a SUM of 0 and a CARRY of 1. When two binary numbers are added, the carry from any position is included in the addition of bits in the next most significant position, as in decimal arithmetic. Example 2.9 illustrates this.



A+B		A	
		0	1
B	0	0	1
	1	1	10

Table 2.2 Binary Addition

Examples 2.9 **Add two Binary digit 1111 $(15)_{10}$ and 10100 $(20)_{10}$**

```
     1   1   1             Carry
         0   1   1   1   1 Augend     15
+        1   0   1   0   0 Addend     20
    ----------------------------------
     1   0   0   0   1   1 Sum        35
```

Here, bits in the LSB position (i.e., position 0) are first added, resulting in a sum bit of 1 and a carry of 0. The carry is included in the addition of bits at position 1. The 3 bits in position 1 are added using two steps (0 + 1=1, 1 + 1=10), resulting in a sum bit of 0 and a carry bit of 1 to the next most significant position (position 2). This process is continued through the most significant bit (MSB).

Subtraction

From Table 2.3., we can see that 0 – 0 = 0, 1- 0= 1, 1 - 1= 0, and 0 - 1= 1 with a BORROW of 1. That is, subtracting a 1 from a 0 results in a 1 with a borrow from the next most significant position, as in decimal arithmetic. Subtraction of two binary numbers is performed stage by stage as in decimal arithmetic, starting from the LSB to the MSB. Some examples follow.

A - B		A	
		0	1
B	0	0	1
	1	11	0

Borrow Difference

Table 2.3 Binary Subtraction

Examples 2.9 **Subtract Binary digit 110101 $(53)_{10}$ from 1000000 $(64)_{10}$**

```
    0   1   1   1   1        Borrow
   *1  *0  *0  *0  *0    0 Minued        64
—   0   1   0   1   0    1 Subtrahend    53
   ---------------------------------------
    0   0   1   0   1    1 Difference    11
```

Bit 2 requires a borrow from bit 3, minuend bit 3 is 0. Then, bit 3 requires a borrow. Because bits 4 and 5 of the minuend are zeros, borrowing is from bit 6. In this process, the intermediate minuend bits 2, 3, 4 and 5 each attain a value of 0 (compare this with the decimal subtraction). The subtraction continues through the MSB.

Multiplication

Binary multiplication is similar to decimal multiplication. From Table 2.4, we can see that 0 X 0=0, 0X1=0, 1X0=0, and 1X1=1. An example follows.

A X B		A	
		0	1
B	0	0	0
	1	0	1

Table 2.4 Binary Multiplication

Examples 2.9 **Multiply binary digit 1001 $(9)_{10}$ with 111 $(7)_{10}$**

```
    1   0   0   1         Multiplicand   9
X               1   1   1 Multiplier     7
-------------------------
            1   0   0   1                  1st position
        1   0   0   1                      2nd position
    1   0   0   1                          3rd position
-------------------------
    1   1   1   1   1   1 Product       63
```

In general, the product of two n-bit numbers is 2n bits long. In Example 2.9, there are three nonzero bits in the multiplier, following shift-and-add algorithm can be adopted to multiply two n-bit numbers A and B, where B=(b_{n_1} b_{n_2} . . . b_1b_0).

1. Start with a 2n-bit product with a value of 0.
2. For each bi ($0 \leq i \leq n\text{-}1$) $\neq 0$, shift A i positions to the left and add to the product.

Division

The division procedure of binary number is also like a decimal, as shown in Example 2.10.

Examples 2.10 **Divide binary 11001 $(25)_{10}$ by 101 $(5)_{10}$ divisor.**

```
              1 0 1 Quotient
        -------------
  1 0 1 | 1 1 0 0 1
        | 1 0 1
        | -----
        | 0 0 1 0
        |   0 0 0
        |-----------
        |     1 0 1
        |     1 0 1
        |     -----
        |     0 0 0 Reminder
```

In this procedure, the divisor is compared with the dividend at each step. If the divisor is greater than the dividend, the corresponding quotient bit is 0, otherwise the quotient bit is 1, and the divisor is subtracted from the dividend. The compare and subtract process is continued until the LSB of the dividend. The procedure is formalized in the following steps.

1. Align the divisor (Y) with the most significant end of the dividend. Let the portion of the dividend from its MSB to its bit aligned with the LSB of the divisor be denoted. We will assume that there are n bits in the divisor and 2n bits in the dividend. Let i=0.

2. Compare X and Y. If $X \geq Y$, the quotient bit is 1: perform X-Y. If $X < Y$, the quotient bit is 0.

3. Set i=i + 1. If $i \geq n$, stop. Otherwise, shift Y 1 bit to the right and go to step 2.

For the purposes of illustration, this procedure assumed the division of integers. If the divisor is greater than the dividend, the quotient is 0, and if the divisor is 0, the procedure should be stopped since dividing by 0 results in an error.

Shifting

Generally, shifting a base r number left by one position (and inserting a 0 into the vacant LSD position) is equivalent to multiplying the number by r. Shifting the number right by one position (inserting a 0 into the vacant MSD position) generally is equivalent to dividing the number by r. In binary system, each left shift multiplies the number by 2, and each right shift divides the number by 2, as shown in Example 2.21.

Example 2.11 **Shift left and right of $(011001.11)_2$.**

	Binary	**Decimal**
N	011001.11	25.75
2* N	110011.10	51.50
N÷2	001100.111	12.875

If the MSB of an n-bit number is not 0, shifting it left would result in a number larger than the magnitude that can be accommodated in n bits and the 1 shifted out of the MSB position cannot be discarded. If nonzero bits shifted out of the LSB position during a right shift are discarded, the accuracy is lost. Later in this chapter, we will discuss shifting in further detail.

Negative number

All numbers have two magnitudes either it is negative or it may be positive. Standard and conventional method for a negative number is representing by placing sign symbol before the number, i.e. negative decimal number 27 is written as -27, if -27 is to be added to + 53, we write + 53 + (-27) = 26 , and negative number is subtracted from positive number is +53 – (- 27) = +53 +27= 80.

The technique use to representing a negative number in digital system, in binary code bi-stable devices can store binary digit, i.e. set of five switches can store the value of 00000 to 11111, now if we desire to increase the total range of numbers in negative from 00000 to -11111 one more bit required, which bit indicate the sign of number. Generally when sign bit is 0 it indicate number is positive and sign bit is 1 indicate the negative number representation.

If any binary number is in unsigned number in this situation this number always indicate only positive magnitude, so the range of this number is doubled comparing with sign number,

Example 2.12

Unsigned N = 10101010 so, N10= 170 (Range is 0 to 255)

While, Sign N = **1**0101010 so, N10= - 42 (Range is +127 to -128)

Sign N = **0**0101010, N_{10} = + 42

The Bold and Gray bit represent the sign of the bit each binary bit is simply store in a individual bi-stable device. So here each number have separator to indicate the sign bit of the number, i.e. -,*, .(dot) , which indicate digit 1-1010,1*1010,1.1010 indicate the -10 number.

2.7 Complements

Complements are used in digital computer system for simplifying the subtraction operation and for logical manipulation. There are two types of complements for each base-r system. There are r's complement and (r-1)'s complement. For binary number referred as a 2's complement and 1's complement., For decimal number referred as 10's complement and 9's complement, For Octal number referred as a 8's complement and 7's complement and, for Hexadecimal number referred as a 16's complement and 15's complement respectively.

(r-1)'s Complement

Given a number N in base r having n digits, the(r-1)'s complement of N is defined as $(r^n -1)$-N for decimal number r=10, r-1 =9 so the 9's complement of N is $(10^{n} -1)$ –N. For example n=4 we have 10^4 -1 =9999. Which follows the 9's complement of the following decimal number is obtained by subtracting each digit from 9.

Example 2.13 **Find 9's complement of following digits.**

(1)	67896	:	99999 – 67896	= 32103
(2)	1658.39	:	9999.99 – 1658.39	= 8341.60

For binary number, r=2 and r-1=1, so the 1's complement of N is $(2^n - 1)$- N. Again, 2^n is represented by a binary number that consist of a 1 followed by n 0's. 2^n -1 is a binary number represented by n 1's.

For example, with n = 4, we have $2^4 = (10000)_2$ -1 = $(1111)_2$ thus 1's complement of binary number is obtain by subtracting of a binary digit from 1 cause the bit to change from 0 to 1 or from 1 to 0.

Example 2.14 **Find 1's complement of following digits**

(3)	10111010	:	11111111 – 10111010	= 01000101
(4)	1101.01	:	1111.11 – 1101.01	= 0010.10

The (r-1)'s complement of octal and hexadecimal numbers are obtained by subtracting each digit from 7 and F (15) respectively.

r's Complement

r's complement is obtain by adding 1 to (r-1)'s complement, so the $r^n - N = [(r^n - 1) - N] + 1$ thus 10's complement of the decimal number,

(1)	67896 :	32103 + 1	=	32104
(2)	1658.39 :	1658.39 +.01	=	1658.40

2's complement of Binary

(3)	10111010 :	01000101+1	=	01000110
(4)	1101.01 :	0010.10 +.01	=	0010.11

The r's complement of octal and hexadecimal numbers are obtained by adding 1 to (r-1)'s complement.

2.7.1 Use of complements to represent Negative Numbers

There are two basic type of complements which are useful in the binary and decimal number systems. For decimal number system two types are referred as 10's and 9's complement.

1) Decimal Subtraction using complement method.

Using 10's complement

- For subtract the one positive number (the minuend) from another (the subtrahend), first the 10's complement of subtrahend is formed, and then this 10's complement is added to this minuend.
- If there is a carry from the addition of the most significant digits then it is discarded, the difference is positive, and the result is correct.
- If the there is no carry the difference is negative the 10's complement of this number is formed, and a minus sign is placed before the result.

Using 9's complement

- For subtract the one positive number (the minuend) from another (the subtrahend), first the 9's complement of subtrahend is formed, and then added to this minuend.
- If there is a carry from the addition of the most significant digits then it is discarded from number and added to LSB of digit. It is your answer.
- If the there is no carry the difference is negative the 9's complement of the addition is formed, and a minus sign is placed before the result. It is your answer.

Example 2.15 **Find result using 10's and 9's complement.**

- **Subtraction using 10's complement.**

(1) Subtract 75 - 21

```
  75   Minuend
– 21   Subtrahend
―――――――――――――――
  54   Difference
```

9's complement of 21 is 99-21= 78
10's complement of 78 is 78 +1 = 79
Now, add 10's complement in to minuend 75 + 79 = 1 54
Here 1 carry generate so ignore it, and place '+' before result **+ 54** is answer.

(2) Subtract 45 – 54

```
  45   Minuend
– 54   Subtrahend
―――――――――――――――
 -11   Difference
```

9's complement of 54 is 99-54= 45
10's complement of 78 is 45 +1 = 46
Now, add 10's complement in to minuend 45 + 46 = 91
Here No carry so, Find 10's complement of 91 .
Find 9's complement of 91 is 99-91=8, 10's complement of 8 is 8+1= 9.
Here no carry generate place '-' before result – 9 is answer

- **Subtraction using 9's complement.**

(3) Subtract 75 - 21

```
  75   Minuend
– 21   Subtrahend
―――――――――――――――
  54   Difference
```

9's complement of 21 is 99-21= 78
Now, add 9's complement in to minuend 75 + 78 = 1 53
Here 1 carry generate so add it in your result of two digits is 53+1 = 54 answer.

(4) Subtract 45 – 54

```
  45   Minuend
– 54   Subtrahend
―――――――――――――――
 -11   Difference
```

9's complement of 54 is 99-54= 45
Now, add 9's complement in to minuend 45 + 45 = 90
Here No carry so, Find 10's complement of 91.
Find 9's complement of 99 is 99-90=9,
Here no carry generate place '-' before result – 9 is answer.

2.7.2 Binary number complement

- ❖ **Binary subtraction using complement method.**

Using 2's complement

- 2's complement of binary number is formed by simply subtracting each digit (bit) of the number the radix minus 1 and adding 1 to least significant bit.
- The rules for application are every 1 in the number is changed to a 0 and every o to 1. Then 1 is added to the least significant bit of the number formed.
- For instance, 2's complement of the number 10110 is 01001 add 1 to this number, and then the number formed is 01010.

Using 1's complement

- 1's complement system is actually the complement of the binary number.
- For instance, 1's complement of 11101 is 00010. When subtraction is performed in the 1's complement system, any end-around carry is added to the least significant bit.

Example 2.16 **Find result using 2s or 1s complement.**

- **Subtraction using 1's complement. (Minuend > Subtrahend).**

1. 10101 - 1010 using 1s

```
          10101       Minuend
        – 01010       Subtrahend
          01011       Difference
```

1's complement of 01010 is 10101
Now add , 10101 (Minuend)
+ 10101(1s Subtrahend)
(Avoid Carry 1) 01010 (1' complement make it 2's complement)
Add 1 in LSB 01011(Difference)
In **01011** Answer. 1 carry generated and ignore it, so that it is a '+' number.

- **Subtraction using 2's complement. (Minuend < Subtrahend).**

1. 1010 – 10101 using 1s

```
          01010       Minuend
        – 10101       Subtrahend
          10101       Difference (first 1 indicate negative number)
```

1's complement of 101011 is 01010and make 2s complement of it is '01011'
Now add , 01010 (Minuend)
+ 01011(2s of Subtrahend)
(No carry '-') **10101(Difference)**
In **10101** Answer. No carry generated, First bit 1 indicate it is a '-'number.

2.8 Binary codes

We are very comfortable with decimal code, but Digital system forced to use a binary number system. Although the binary number system has many practical advantages and is widely used in digital computer, Numeric decimal codes used to represent decimal digits are called **Binary Coded Decimal (BCD)** codes.

Binary code is representing numeric and various other symbols such as alphabetic characters and special characters (e.g. /, +, -, =) to represent information data in binary form. A digital system requires that all information be in binary form. These human friendly symbols have a unique pattern of 0s and 1s are assigned to represent each symbol. This pattern is the code word corresponding to that symbol. It is possible to represent 2n elements with a binary string containing n bits. That is, if n symbols are to be represented in binary form, the minimum number of bits n required in the code word is given by $2^{n-1} < n \le 2^n$.

The code word might be containing more than n bits to accommodate error detection and correction. Once the number of bits in the code word is set, the assignment of the code words to the symbols of information to be represented could be random (in which case a table associating each element with its code word is needed) or might follow some general rule.

For example, if the code word is required to represent four symbols—say, A, B, C, D we can use the four combinations of 2 bits (00, 01, 10, and11). Assignments of these combinations to the four symbols can be random, but after assignment it will no changed.

To represent the 9 digits of decimal system we would need 4 bit, to generate 16 possible combinations of 0s and 1s.and each code assign to each digit and still some codes are non used code, or we can say that it is non valid code.

The codes designed to represent only numeric data (i.e., decimal digits 0 through 9) can be classified into two categories:

1). Weighted
2). Non-weighted.

The alphanumeric codes can represent both alphabetic and numeric data. A third class of codes is designed for error-detection and -correction purposes.

2.8.1 Weighted codes (BCD)

A BCD code is one, in which the decimal digits are encoded –one at a time into group of four binary digits. As stated, we require at least 4 bits to represent the 10 decimal digits. But with 4 bits only 10 of the 16 possible combinations are used, numerous different codes are possible. *Table 2.5* shows some of these possibilities.

There are several weighted codes. The weighted codes are either *positively weighted* or *negatively weighted*.

Positively weighted codes are those in which all the weighs assign to the binary digits are positive. There are only 17 positively weighted codes. In these code first must be 1, second must be either 1 or 2, and the sum of all weighs must be equals to or grater than 9. The code 8421, 2421, 5211, 3321, 4311 are some of positively-weighted codes.

Negatively weighted codes some of the assign to the binary digits must be negative. The code 642-3, 631-1, 84-2-1, and 74-2-1 are some of negatively-weighted codes.

All of these codes given in tables are weighted codes, since each bit position in the code has a weight associated with it. The sum of weights corresponding to each nonzero bit in the code is the decimal digit represented by it.

Weights	8421	2421	5211	5421	642-3	84-2-1
Digits	Code	Code	Code	Code	Code	Code
0	0000	0000	0000	0000	0000	0000
1	0001	0001	0001	0001	0101	0111
2	0010	0010	0011	0010	0010	0110
3	0011	0011	0101	0011	1001	0101
4	0100	0100	0111	0100	0100	0100
5	0101	0101	1000	1000	1011	1011
6	0110	0110	1010	1001	0110	1010
7	0111	0111	1100	1010	1101	1001
8	1000	1110	1110	1011	1010	1000
9	1001	1111	1111	1100	1111	1111

Table 2.5 Weighted Code

Sequential Codes

Some weighted codes are one binary number grater than its preceding code word. Such as 8421 and XS-3 (see later) are sequential code.

Self-complementing codes

If the code word N of 9's complementing 9-N, can be obtain from the code word of the N by interchanging all 0s and 1s. Thus the code which has sum of numbers is 9 are self complementing code. Here, 8421, 5421 codes are not self complementing code, while 2421, 5211, 642-3, 84-2-1 are self complementing codes.

I.e. code 5211= 5+2+1+1 =9 so, it is self-complementing code.

Cyclic code

Each successive code word differs from the preceding one in only one bit position that is known as a cyclic code. They also called *unit distance codes.* Gray code is a cyclic code.

1. BCD Codes

- **The 8421 BCD code:** In this code every decimal digit coded in their own binary code, as per *table 2.5.* It is also called *natural binary* code. It is useful for mathematical operations. Main advantage of this code is it's easy to conversion to and from decimal. It is less efficient that the pure binary. The decimal number 11 represent in pure binary as a 1011 but in 8421 BCD code it is represent as 0001 0001. There are six illegal combinations 1010 (10),1011 (11), 1100(12), 1101(13), 1110 (14), 1111 (15) they are not part of 8421 BCD code.

BCD Arithmetic:

The rules of Binary addition and subtraction do not apply to entire 8421 number but only to individual 4 bit groups,

- In BCD addition a carry out of one group to next group or if the result is an illegal code, then 6 is add to the sum term of that group and the resulting carry is added to the next group (because to skip 6 illegal step)
- The BCD subtraction is performed by subtracting the digits of each 4-bit group of the subtrahend from corresponding 4-bit group of the minuend in binary starting from the LSD. If there is borrow from the next group, then 6_{10}(0110) is subtracted from the difference term of this group (This is done to skip 6 illegal states.)
- Since we subtracting decimal digit, we must from the 9s or 10s complement of the decimal subtrahend and encode that number in the 8421 code. The resulting BCD numbers are then added.

Example 2.17 **Perform the following decimal addition in the 8421 code.**

A) 24 + 11 B) 579.6 + 636.8

```
a)     24    - 8421 BCD code -      0010 0100
      +11    -8421 BCD code -       0001 0001
      _____________________________________
       35    -(No carry No some)-   0011  0101     - this is correct answer.

b)     579.6 - 8421 BCD code -      0101 0111 1001. 0110
      +636.8 - 8421 BCD code -      0110 0011 0110. 1000
      ___________________________________________________
      1216.4 -All are illegal code-  1011  1010  1111.  1110  (add 0110 to each)
                                    +0110+0110+0110.+0110
                               ______________________________
                                 00001000110000101011.0100 (propagate carry)
                                    +1    +1    +1    +1
                               ________________________________
                                 0001  0010  0001  0110  . 0100
                                    1   2     1      6   .  4    (Correct Sum)
```

Example 2.18 **Perform the following decimal subtraction in the 8421 code**

A) 24 - 11 B) 636.8 - 579.6

```
a)     24    - 8421 BCD code -      0010 0100
      -11    -8421 BCD code -       0001 0001
      _____________________________________
       13    -(No Borrow   )-       0001  0011     - this is correct answer
```

b)

```
 636 .8 - 8421 BCD code -    0110 0011 0110. 1000
-579.6 - 8421 BCD code -     0101 0111 1001. 0110
_____________________________________________________
057.2   - for borrow    -    00001101111101.0010  (Subtract 0110 to)
                                     - 0110-0110.
                             ________________________
                             0000 0101 0111. 0010
                             ________________________
                             0      5      7 .  2   (Correct  Difference)
```

Example 2.19 **Perform the following decimal subtraction using 9s 10s complement.**

A) 24 - 89   B)   636.8 - 579.6

a) Here, 24-89= -65 , using 10s complement here, 9s of subtrahend 89 is 10 and 10s of 10 is 11. Now, add 11 to minuend 24 so

```
 24     - 8421 BCD code -    0010 0100
+11     -8421 BCD code -     0001 0001
__________________________________________
 35     - (No Borrow  )-     0011  0101     - 9s complement
                             1001  1001 -
                             0110  0100    (64) +1 =0110 0101= -65
```

Here no carry generate so answer is negative now complement of 0110 0101

b) 636.8 - 579.6

Minuend is 636.8 and subtrahend is 579.6 subtraction is positive. First 9s of 579.6 then, add to 636.8

9s of 579.6 is 420.3 add to 636.8

```
  636 .8- 8421 BCD code -    0110 0011  0110. 1000
+420.3 - 8421 BCD code -     0100  0010 0000. 0011
1057.1- Ignore carry    -    1010 0101  0110. 1011 Add 0110 to some term
                           +10110           +1. 0110  Forward carry to term
                        0001 0000 0101  0111. 0001  Ignore First carry 1
```

We found the result with the use of 9s complement. carry generate so result is positive. So it is 57.1

2.8.2 Non-Weighted codes (BCD)

Table 2.6 shows two popular codes. They do not have any weight associated with each bit of the code word.

Gray Code				Decimal	4-bit binary	XS-3 Code	XS-3 GrayCod
1-bit	2-bit	3-bit	4-bit				
0	0 0	0 0 0	0 0 0 0	0	0 0 0 0	0011	0010
1	0 1	0 0 1	0 0 0 1	1	0 0 0 1	0100	0110
	1 1	0 1 1	0 0 1 1	2	0 0 1 0	0101	0111
	1 0	0 1 0	0 0 1 0	3	0 0 1 1	0110	0101
		1 1 0	0 1 1 0	4	0 1 0 0	0111	0100
		1 1 1	0 1 1 1	5	0 1 0 1	1000	1100
		1 0 1	0 1 0 1	6	0 1 1 0	1001	1101
		1 0 0	0 1 0 0	7	0 1 1 1	1010	1111
			1 1 0 0	8	1 0 0 0	1011	1110
			1 1 0 1	9	1 0 0 1	1100	1010
			1 1 1 1	10	1 0 1 0	Not define	Not define
			1 1 1 0	11	1 0 1 1	Not define	Not define
			1 0 1 0	12	1 1 0 0	Not define	Not define
			1 0 1 1	13	1 1 0 1	Not define	Not define
			1 0 0 1	14	1 1 1 0	Not define	Not define
			1 0 0 0	15	1 1 1 1	Not define	Not define

Table 2.6 Non-Weighted Codes and reflection of gray code

1. Excess Three (XS-3) Codes: Excess-3 is a 4-bit self-complementing code. It can be written as XS-3 code. The code for each decimal digit is obtained by adding 3 to the corresponding BCD code word. It is sequential code so it can be useful for arithmetic operations. Excess-3 code has been used in some older computer systems. In addition to being self-complementing, thereby making subtraction easier, this code enables simpler arithmetic hardware. This code is included here for completeness and is no longer commonly used. It has six invalid states those are 0000, 0001, 0010, 1101, 1110 and 1111. Those are not indicated in *table 2.6.*

In XS-3 Gray code each decimal digit is encoded with the Gray code pattern of the decimal digit that is greater by 3. it has a unit distance between the pattern for o to 9.table 2.6 XS-3 Gray code for decimal digits 0 through 9.

XS-3 Arithmetic:

XS-3 code has very important properties for performing arithmetic operation:

- **Addition:**
  - To add in XS-3 number by adding the 4-bit group in each column starting from LSD.
  - If there is no carry out from the addition of any of the 4 bit groups, subtract 0011 from the sum term of those groups. (*Two decimal are added in XS-3 and no carry the result is in XS-6*).

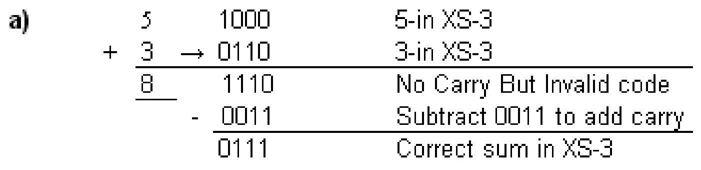
- If there is a carry out, add 0011 to the sum term of those groups (because when there is a carry, the invalid states are skipped and the result is in normal binary).

- **Subtraction:**

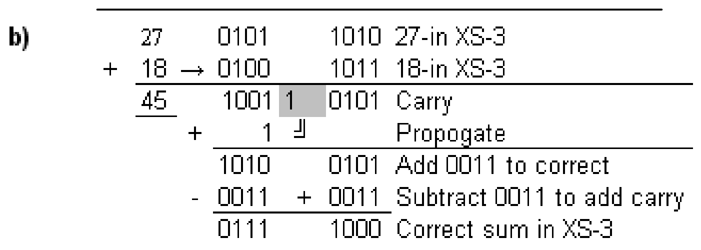
  - To subtract in XS-3 number by subtracting 4-bit group of the subtrahend from the corresponding 4-bit group of minuend starting from LSD.
  - If there is no borrow from the next 4-bit group, add 0011 to the difference term of such group (because when decimal digits are subtracted in XS-3 and there is no borrow, the result is in normal binary).
  - If there is a borrow, subtract 0011 from the difference term (because taking a borrow is equivalent to adding six invalid states, so the result is in XS-6).
  - Generally subtraction is performed by the 9's complement or 10's complement method.

Example 2.20 **Perform the following addition in XS-3 code.**

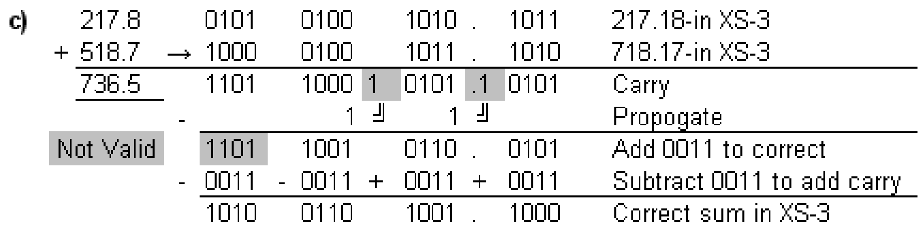
A) 5 + 3 B) 27 + 18 C) 217.8 +518.7



```
a)        5       1000            5-in XS-3
        + 3  →    0110            3-in XS-3
          8        1110           No Carry But Invalid code
                -  0011           Subtract 0011 to add carry
                  0111            Correct sum in XS-3
```



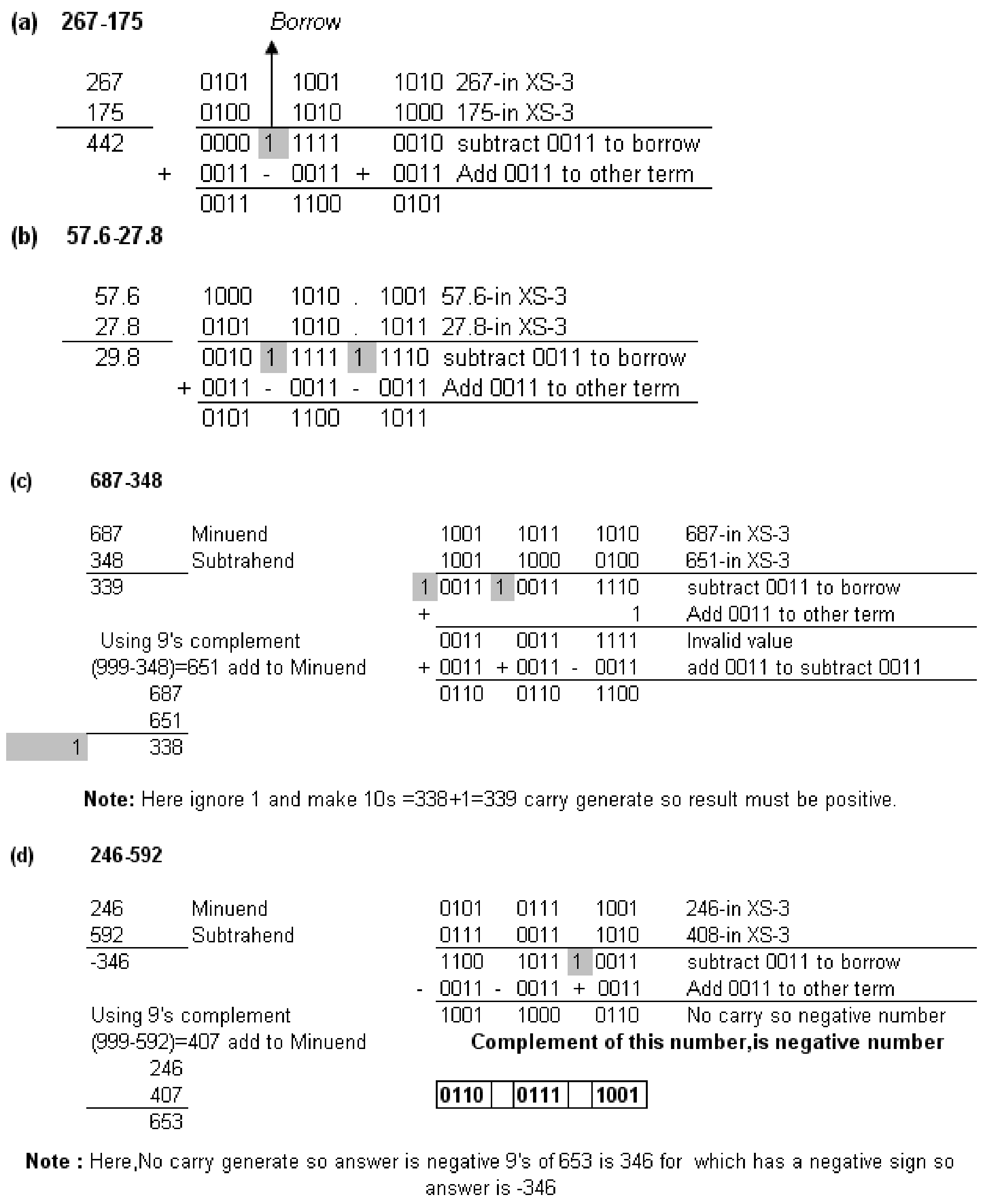
```
b)       27      0101        1010  27-in XS-3
       + 18  →   0100        1011  18-in XS-3
         45       1001  1    0101  Carry
               +     1 ┘           Propogate
                 1010        0101  Add 0011 to correct
               - 0011    +   0011  Subtract 0011 to add carry
                 0111        1000  Correct sum in XS-3
```



```
c)    217.8       0101    0100      1010 .    1011    217.18-in XS-3
    + 518.7  →    1000    0100      1011 .    1010    718.17-in XS-3
      736.5       1101    1000  1   0101  .1  0101    Carry
               -             1 ┘       1 ┘            Propogate
   Not Valid      1101    1001      0110 .    0101    Add 0011 to correct
               -  0011  - 0011  +   0011  +   0011    Subtract 0011 to add carry
                  1010    0110      1001 .    1000    Correct sum in XS-3
```

Example 2.21 **Perform the following subtraction in XS-3 code using 9's complement.**

(A) 267-175 (B) 57.6-27.8 (C) 687-348 (9's com) (D) 246-592(9's com)



(a) 267-175

```
                         Borrow
                           ↑
  267         0101    |   1001        1010  267-in XS-3
  175         0100    |   1010        1000  175-in XS-3
 -----        ---------------------------------------------
  442         0000   1    1111        0010  subtract 0011 to borrow
          +   0011   -    0011   +    0011  Add 0011 to other term
              ---------------------------------------------
              0011        1100        0101
```

(b) 57.6-27.8

```
  57.6        1000       1010   .   1001  57.6-in XS-3
  27.8        0101       1010   .   1011  27.8-in XS-3
 ------       -------------------------------------------
  29.8        0010   1   1111   1   1110  subtract 0011 to borrow
            + 0011   -   0011   -   0011  Add 0011 to other term
              -------------------------------------------
              0101       1100       1011
```

(c) 687-348

```
      687        Minuend                     1001    1011    1010     687-in XS-3
      348        Subtrahend                  1001    1000    0100     651-in XS-3
      -----------                         ---------------------------------------------
      339                                 1  0011  1 0011    1110     subtract 0011 to borrow
                                          +                     1     Add 0011 to other term
                                          ---------------------------------------------
       Using 9's complement                  0011    0011    1111     Invalid value
      (999-348)=651 add to Minuend        +  0011  + 0011  - 0011     add 0011 to subtract 0011
                   687                    ---------------------------------------------
                   651                       0110    0110    1100
           1       338
```

Note: Here ignore 1 and make 10s =338+1=339 carry generate so result must be positive.

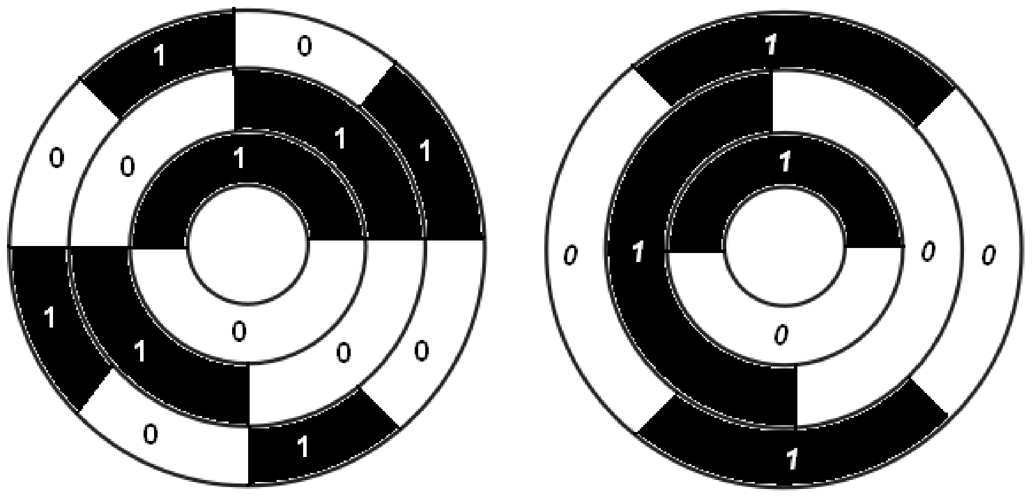
(d) 246-592

```
      246        Minuend                     0101    0111    1001     246-in XS-3
      592        Subtrahend                  0111    0011    1010     408-in XS-3
      -----------                         ---------------------------------------------
      -346                                   1100    1011  1 0011     subtract 0011 to borrow
                                          -  0011  - 0011  + 0011     Add 0011 to other term
                                          ---------------------------------------------
      Using 9's complement                   1001    1000    0110     No carry so negative number
      (999-592)=407 add to Minuend            Complement of this number,is negative number
                   246
                   407                     |0110|  |0111|  |1001|
      -----------------
                   653
```

Note : Here,No carry generate so answer is negative 9's of 653 is 346 for which has a negative sign so answer is -346

2. **Gray Code:** The gray code is not suitable for arithmetic operation it is cyclic unit distance and reflective code. It is a 4-bit code in which the 16 code words are selected such that there is a change in only 1-bit position as we move from one code word to the adjacent code word. Because only 1-bit changes from code word to code word, it is easier to detect an error if there is a change in more than 1 bit.
For example, consider the case of a shaft position indicator. Assume that the shaft position is divided into 16 sectors indicated by the gray code. As the shaft rotates, the code words change. If at any time there is a change in 2 bits of the code word compared with the previous one, there is an error. Several cyclic codes have been devised and are commonly used.

Another property of gray code is its corresponding to an decimal number 2 n-1 for any n differs from gray coded 0 in one bit position only. This property places the gray code for the largest N-bit binary number at unit distance from 0.it is easier to determine the pattern assignment corresponding to an random number, or decode an random number gray code pattern using the conversions to and from binary. Gray codes are used in instrumentation and data acquisition systems where linear or angular displacement is measured. They are also used in shaft encoder, I/O devices, A/D converters and other peripheral equipments.
Consider a rotating disk that provides an output of its position in 3-bit binary (*Fig2.1*). When the brushes are on the black part, they output a 1, and when they are on a white sector they output a 0. Still some errors are there, i.e. if the brushes are on the sector 010 and almost ready to enter 110 sectors and if the 4's brushes are slightly ahead, the position would be indicate by 110 instead of 010 resulting in a very small error. *Fig 2.1* illustrate this operation



a) Binary Code b) Gray Code

Figure 2.1 Code indication

➢ **Binary to Gray & Gray to Binary**

Binary code to Gray code

In n-bit binary number represented by $B_n\ B_{n-1}\ B_{n-2} \ldots\ldots B_1$ and it's gray code equivalent by $G_n\ G_{n-1}\ G_{n-2} \ldots\ldots G_2\ G_1$.Where B_n and G_n are MSB then gray code bits are obtained from the binary code as follows

Term .	1	$G_n = B_n$ ***(MSB)***
Term .	2	$G_n = B_n \oplus B_{n-1}$
Term .	3	$G_n = B_{n-1} \oplus B_{n-2}$
Term .	.	
Term .	.	
Term .	.	
Term .	n	$G_1 = B_2 \oplus B_1$ ***(LSB)***

a	b	a X-OR b
		$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.7 X-OR ($a \oplus b$) output

When $\oplus$ stands for Exclusive Or (X-OR). Truth table for output XOR is as per *table 2.7.*

1. Record the MSB of the Binary as the Gray code.
2. Add the MSB of the Binary to the next bit in Binary, recording the sum and ignoring the carry if any, i.e. X-Or the bit. This sum is the next bit of the Gray code.
3. Add the 2nd bit of the Binary to the 3rd bit of the Binary, the 3rd bit to the 4th bit and so on….
4. Record the successive sums as the successive bits of the Gray code until all the bits of the Binary number are exhausted.

Example 2.22 Convert the binary code 1001 to gray code.

1st Method -Using XOR

Binary code - $1 \rightarrow 0 \rightarrow 0 \rightarrow 1$

$\downarrow\ \downarrow\ \downarrow\ \downarrow$

Gray code - $1 \rightarrow 1 \rightarrow 0 \rightarrow 1$ = 1101. Gray code of 1001 Binary code

2nd Method –Shifted Binary

```
Binary code   -    1 0 0 1
Shifted binary -  + 1 0 0  1 ──► Ignore This bit.
                   1  1 0 1      Gray code of 1001 Binary code
```

1st Method: 8,4,2,1 bit binary code are represented like 8 bit gray code represented as 8 bit of binary bit, 4 bit of gray code represented as (8bit binary $\oplus$ 4-bit binary), 2 bit of gray code represented as (4-bit binary $\oplus$ 2-bit binary), 1 bit of gray code represented as (2-bit binary $\oplus$ 1-bit binary), and o this gray code as a **1101** as a result**.**

2nd Method: 8,4,2,1 bit binary code using shifted method as add binary code in to 1 bit right shifted binary code and exhausted last bit of code, so add 100(right shifted binary code) to 1001 and you find the gray code as a **1101** as a result.

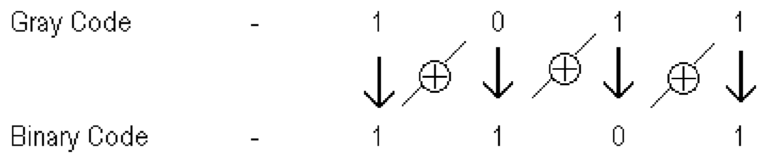
Gray code to Binary code

In n-bit Gray number represented by $G_n \ G_{n-1} G_{n-2} \ldots\ldots G_2 G_1$ and it's Binary equivalent number by $B_n \ B_{n-1} B_{n-2} \ldots\ldots B$. Where B_n and G_n are MSB then gray code bits are obtained from the Gray code as follows,

Term . 1	$B_n = G_n$ ***(MSB)***
Term . 2	$B_n = B_n \oplus G_{n-1}$
Term . 3	$B_n = B_{n-1} \oplus G_{n-2}$
Term . .	
Term . .	
Term . .	
Term . n	$B_1 = B_2 \oplus G_1$ ***(LSB)***

1. Record the MSB of the Gray code as the Binary code.
2. Add the MSB of the binary to the next significant bit of the Gray code, recording the sum and ignoring the carry if any, i.e. XOR the bit. This sum is the next bit of the gray code.
3. XOR the 2nd bit of the Binary to the 3rd bit of the Gray code, the 3rd bit of the Binary to 4th bit of the Gray code and so on….
4. Continue till all the gray bits are exhausted. The sequence of bit written down is the binary equivalent of the Gray code number.

Example 2.23 Convert the Gray code 1011 to Binary code.



Record the 8 bit MSB of the Gray code as the 8 bit of Binary code. XOR 8-bit of Binary to 4 bit of Gray code (1⊕ 0), 4 bit of Binary to 2 Bit of Gray code (1⊕ 1), 2-bit of Binary to 1-bit of gray code (0 ⊕ 1) where Gray code is exhausted and record your Binary code **1101** as a result.

2.8.3 Error detecting code and Error correcting codes

Network must be transfer data from inter communication of network devices, during the transmission of data accuracy must be acceptable to other device. A system must be guarantee to received data Identical to the data transmitted. In any condition data must be transmitted but not received, they can be corrupted in passage. Many factors can be altering one or more bits of massage. Some applications require a mechanism for detecting and correcting errors.

Types of Errors

Whenever bits flow from one point to another, they are subject to unpredictable changes because of interference. This interference can change the shape of signal. In a single bit error a 0 is changed to 1 or 1 to a 0.in a burst error, multiple bits are changed.

Error Detecting Code

1. Parity-check code:

Parity check code is most familiar code for error-detecting for your datawords. k-bit datawords can changed to an n-bit codeword where n=k+1. The extra bit, call the parity bit, is selected to make the total number of 1s in the codeword even. Although some implementations specify an odd number of 1s, parity check code is error detecting code in *fig 2.2* it can't be correct any error.

Single bit error:

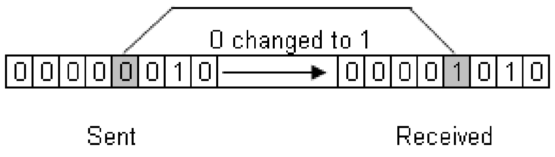


Figure 2.2 Single bit error

The encoder uses a generator that takes a copy of a 4-bit dataword (a0,a1,a2,a3) and generates a parity bit r0.the dataword bit and parity bits generate 5-bit of codeword as per *fig 2.3* there are two types of parity generated either it is even parity or odd parity.

r0=a3+a2+a1+a0

The sender sends the codeword which may be corrupted during transmission. The receiver receives a 5-bit word. The checker at the receiver does the same thing as the generator in the sender with one exception: the addition will done over all 5 bits. The result, which is called syndrome, is just 1 bit. The syndrome is 0 when the number of 1s in the received codeword is even; otherwise, it is 1.

s0=b3+b2+b1+b0+q0

The syndrome is passed to the decision logic analyzer. If the syndrome is 0, there is no error in the received codeword; the data portion of the received codeword is accepted as the dataword; if the syndrome is 1, the data portion of the received codeword is discarded. The dataword is not created.

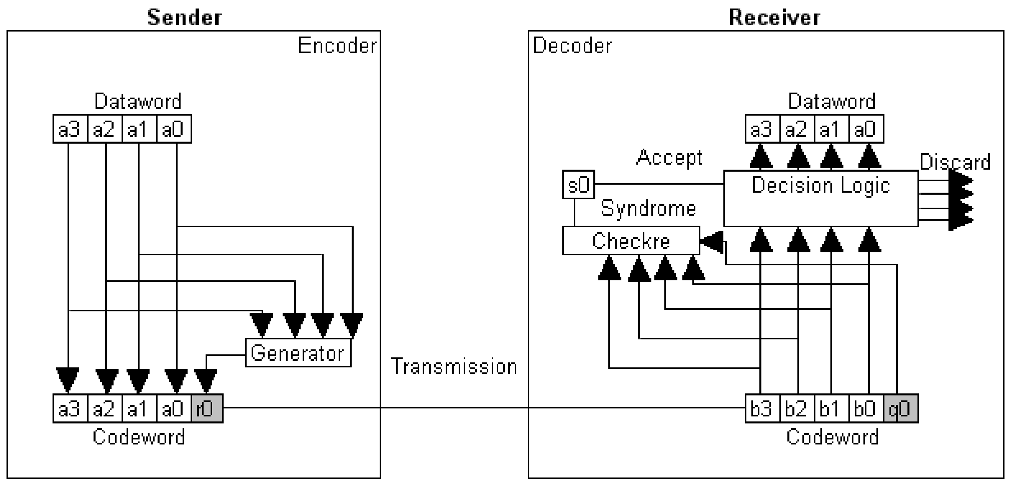


Figure 2.3 Encoder and Decoder for simple parity check

Decimal	8421 BCD	Even parity		Odd Parity	
	Datawords	P	Codewords	P	Codeword
0	0000	1	00001	0	00000
1	0001	0	00010	1	00011
2	0010	0	00100	1	00101
3	0011	1	00111	0	00110
4	0100	0	01000	1	01001
5	0101	1	01011	0	01010
6	0110	1	01101	0	01100
7	0111	0	01110	1	01111
8	1000	0	10000	1	10001
9	1001	1	10011	0	10010

Table 2.8 Even Odd Parity In 8421 BCD code

Example 2.23 **Let us look at some transmission of dataword 1101. Code word created form dataword which is set odd parity and the codeword sent to receiver. We examine different cases.**

- Case 1 No error occur: The received codeword is 11011. The syndrome is 0. The dataword 1011 is created.
- Case 2 Single bit error change dataword: The received codeword is 10011. The syndrome is 1. No dataword is created.
- Case 3 Single-bit error change parity: The received codeword is 11010. The syndrome is 1. No dataword is created.

- Case 4 Multiple-bit error (more then one error): If there is two bit error, the received codeword is 00110. The syndrome is 0. Dataword 0011 is created at the receiver. If there is three bit error in codeword is 00111. The syndrome is 1, no dataword is created.

Example 2.24 **In Even Parity scheme which word contain an error:**

No	Dataword	Even Parity	Error Status
1	10101010	1	No Error
2	11110100	0	No Error (ODD)
3	10111001	1	Error (ODD)

2. Check Sum:

Simple Parity can not detect the two error within the same word so difficulty is sort of two dimensional parity. As each word is transmitted, it is added to the sum of previously transmitted words, and sum retained at the transmitter end at the end of transmission, the sum (check sum) up to that time is sent to the receiver. The receiver can check its sum with the transmitted sum. If the two sums are the same then no error ware detected at the receiver end. If there is an error the receiving location can ask for retransmission of the entire data. This is the type of transmission used in teleprocessing systems.

3. Block Parity:

When several binary words are transmitted or store in succession the result collection of bits can be regarded as a block of data, having rows and columns parity bits can then be assign to both rows and columns.

I.e. 8-bit word in succession can be formed in to 6 x 8 block for transmission. Parity bits are added both rows and columns and the block is transmitted as a 7 x 9 block as shown in *Fig 2.9 (Table A)* at receiving end parity is checked both row and column wise, suppose errors are detect and corrected by complementing the error bit, in Fig 2.9 (Table-B) 3rd row 5th column, it can be corrected by complementing it. Two error as shown in table C can only detected but not corrected, parity error observed in both columns 2 and 4. It indicate that in one row there are two error.

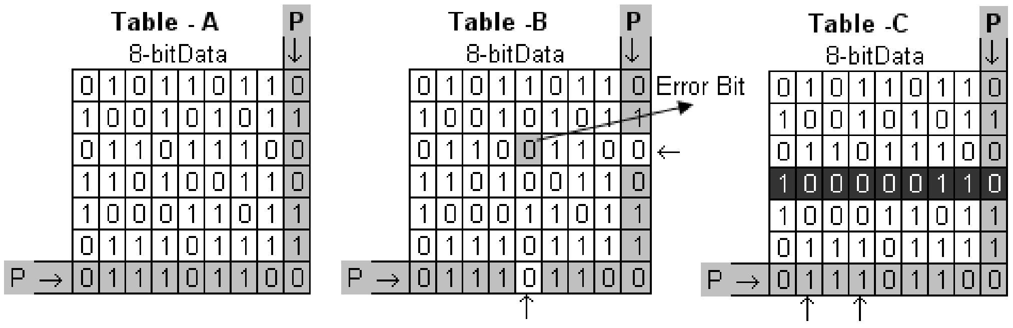


Table - A
8-bitData
P ↓
P →
Table -B
8-bitData
P ↓
Error Bit
←
P →
↑
Table -C
8-bitData
P ↓
P →
↑ ↑

Table - A

	8-bitData								P ↓
	0	1	0	1	1	0	1	1	0
	1	0	0	1	0	1	0	1	1
	0	1	1	0	1	1	1	0	0
	1	1	0	1	0	0	1	1	0
	1	0	0	0	1	1	0	1	1
	0	1	1	1	0	1	1	1	1
P →	0	1	1	1	0	1	1	0	0

Table -B

	8-bitData								P ↓	
	0	1	0	1	1	0	1	1	0	Error Bit
	1	0	0	1	0	1	0	1	1	
	0	1	1	0	0	1	1	0	0	←
	1	1	0	1	0	0	1	1	0	
	1	0	0	0	1	1	0	1	1	
	0	1	1	1	0	1	1	1	1	
P →	0	1	1	1	0	1	1	0	0	
					↑					

Table -C

	8-bitData								P ↓
	0	1	0	1	1	0	1	1	0
	1	0	0	1	0	1	0	1	1
	0	1	1	0	1	1	1	0	0
	1	0	0	0	0	0	1	1	0
	1	0	0	0	1	1	0	1	1
	0	1	1	1	0	1	1	1	1
P →	0	1	1	1	0	1	1	0	0
		↑		↑					

Figure 2.4 Block Parity

4. Five bit Codes:

Some 5-bits BCD code that have parity contained within each code for ease of error detection are shown in (*table 2.9*) the following 6310 is a weighted code (Except 0). It has the useful error detecting property that there are exactly two 1's in each code group. This code is used for storing data on magnetic tapes.

The two out of five is non weighted code it also has 1's in each code group this code is useful in telephone and communication industries at the receiving end, the receiver can check the number of 1's in each character received.

Shift counter code also called Johnson code has bit pattern produce by 5- bi Johnson counter. The 51111 code is similar to Johnson code but it is weighted code.

Decimal	63210	2-out of-5	Shift Counter	51111
0	00110	00011	00000	00000
1	00011	00101	00001	00001
2	00101	00110	00011	00011
3	01001	01001	00111	00111
4	01010	01010	01111	01111
5	01100	01100	11111	10000
6	10001	10001	11110	11000
7	10010	10010	11100	11100
8	10100	10100	11000	11110
9	11000	11000	10000	11111

Table 2.9 Five Bit BCD Code

5. The Bi-quinary code:

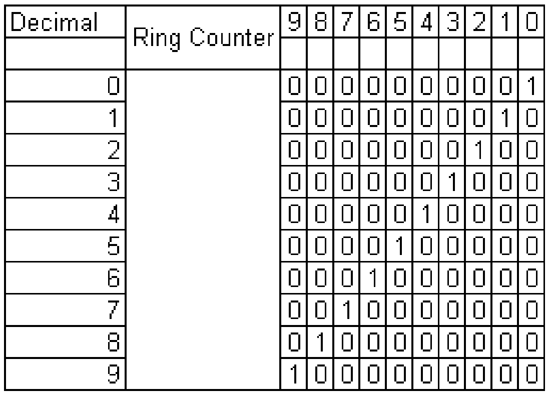
It is 7-bit BCD code It's a parity data code each code group regarded as consisting 2-bit subgroup and a 5-bit subgroup, and each of this subgroup contain a single 1. So, it has error checking feature. for each code group ha exactly two a's and each subgroup has exactly one 1.the weight of the bit position are as *table 2.10.*

Decimal	5	0	4	3	2	1	0
0	0	1	0	0	0	0	1
1	0	1	0	0	0	1	0
2	0	1	0	0	1	0	0
3	0	1	0	1	0	0	0
4	0	1	1	0	0	0	0
5	1	0	0	0	0	0	1
6	1	0	0	0	0	1	0
7	1	0	0	0	1	0	0
8	1	0	0	1	0	0	0
9	1	0	1	0	0	0	0

Table 2.10 Bi-Quinary Code

6. **The Ring counters code:**

10-bit ring counter code produce a sequence of 10-bit group having the property that each group has a single 1. It is weighted code because each bit weight equals to 1 of the 10 decimal digits although this code is inefficient (for 1024 encode in pure binary), it is excellent error detecting properties to implement ring counter.



Decimal	Ring Counter	9	8	7	6	5	4	3	2	1	0
0		0	0	0	0	0	0	0	0	0	1
1		0	0	0	0	0	0	0	0	1	0
2		0	0	0	0	0	0	0	1	0	0
3		0	0	0	0	0	0	1	0	0	0
4		0	0	0	0	0	1	0	0	0	0
5		0	0	0	0	1	0	0	0	0	0
6		0	0	0	1	0	0	0	0	0	0
7		0	0	1	0	0	0	0	0	0	0
8		0	1	0	0	0	0	0	0	0	0
9		1	0	0	0	0	0	0	0	0	0

Table 2.11 The Ring Counter

<u>Error correcting codes</u>

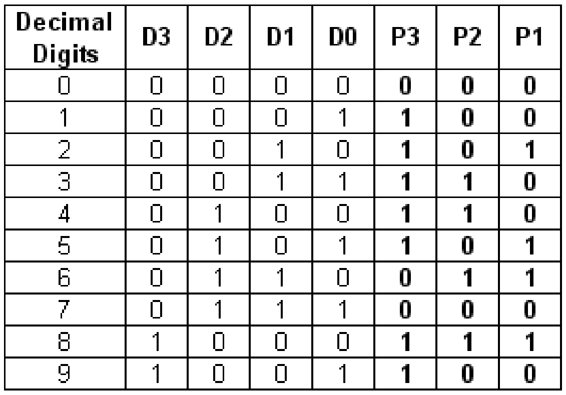
7-bit Hamming code: If the correct codeword is deduced form error word the minimum distance of that must be three the code. The key to error correction is that it must be possible to detect and locate erroneous digits. if location of error has been determined then by complementing the erroneous digit the message can be corrected. One type of error correcting code is hamming code.

Hamming code was detecting up to two errors and corrects one error; our focus is single bit correcting hamming code. Hamming code choose an integer m>=3. The value of n and k are then calculated from m as $n=2^m-1$and $k=n-m$. The number of check bits r=m.

In simple, to transmit 4 data bit three parity bits located at 2^0, 2^1 and 2^2 are added to make a 7- bit codeword which is then transmitted, the word format would be as shown below:

D3	D2	D1	D0	P2	P1	P0

Here the D are data bits and the P are parity bits, P0 set to 0 or 1 so it establishes even parity over D0,D1,D2. P1 set 0 or 1 to establish even parity over bits D1, D2, D3. P3 set to a 0 or 1 to establish even parity over D2, D3, D0, The bit hamming code for the decimal digits coded the BCD is shown in *table 2.12*,



Decimal Digits	D3	D2	D1	D0	P3	P2	P1
0	0	0	0	0	**0**	**0**	**0**
1	0	0	0	1	**1**	**0**	**0**
2	0	0	1	0	**1**	**0**	**1**
3	0	0	1	1	**1**	**1**	**0**
4	0	1	0	0	**1**	**1**	**0**
5	0	1	0	1	**1**	**0**	**1**
6	0	1	1	0	**0**	**1**	**1**
7	0	1	1	1	**0**	**0**	**0**
8	1	0	0	0	**1**	**1**	**1**
9	1	0	0	1	**1**	**0**	**0**

Table 2.12 The 7-bit Hamming code- Even parity

The receiving end the message received hamming code is decoded to receiving side generator creates three parity checks P0, P1 and P12 as shown below:

The checker is the decoder creates a 3-bit syndrome (s2, s1, s0) I which each bit is the parity check four 4 out of 7 bits in received codeword:

s0=d0+d1+d2+p0;
s1=d1+d2+d3+p1;
s2=d2+d3+d0+p2;
three parity-checks bi generate eight different combinations are

Syndrome	000	001	010	011	100	101	110	111
Error	None	p0	p1	d2	p2	d0	d3	d1

Example: 2.25 Let us trace some path sending dataword and receiving.

- The dataword 0100 become the codeword 0100011. The codeword 0100011 is received the syndrome is 000 the final dataword is 0100.
- The dataword 0111 become the codeword 0111001.the codeword 0011001 is received. The syndrome is 011. According to table d2 is in error after complementing b2 final dataword is 0111.
- The dataword 1101 become the codeword 1101000. The codeword 0001000 is received. The syndrome is 101, which means that b0 is in error. After complementing b0 we get 0000, the wrong dataword. This shows that our code cannot correct two errors.

Example: 2.26 Encode data bits 0011 in to 7-bit even parity hamming code:

0	0	1	1	P2	P1	P0

110+p0=1100
100+p1=1001
100+p2=0011

Let us trace some path sending dataword and receiving.

- The dataword 0011 become the codeword 0011110. The codeword 0011110 is received ,

  s0=1100=0
  s1=1001=0
  s2=0011=0

  Syndrome is 000 so final Dataword is 0011.
- The codeword 1011110 is received ,

  s0=1100=0
  s1=1011=1
  s2=1011=1

  Syndrome is 110 so final Dataword is D3 having an error replace by complementing it and code is 0011110.

Summary

Except these are the number systems are use for computer and decimal system is use in general sense general but some other number system are also use i.e. the quinary system, which has 5 for base, was prevalent among Eskimos and north American Indians. The other system is duodecimal system (base 12) may be seen in clock, inches and feet, dozens and grosses.