COURSE CODE: SCS1203

COURSE NAME: FUNDAMENTALS OF DIGITAL SYSTEMS

CHAPTER NAME: NUMBER SYSTEMS,COMPLIMENTS AND CODES

UNIT - I

Number Systems - Binary Numbers - Number base conversions - Octal and Hexa Decimal Numbers - Complements - Signed Binary Numbers - Binary Arithmetic - Binary Codes - Decimal Code - Error Detection code - Gray Code- Reflection and Self Complementary codes - BCD number representation - Alphanumeric codes ASCII/EBCDIC - Hamming Code- Generation, Error Correction.

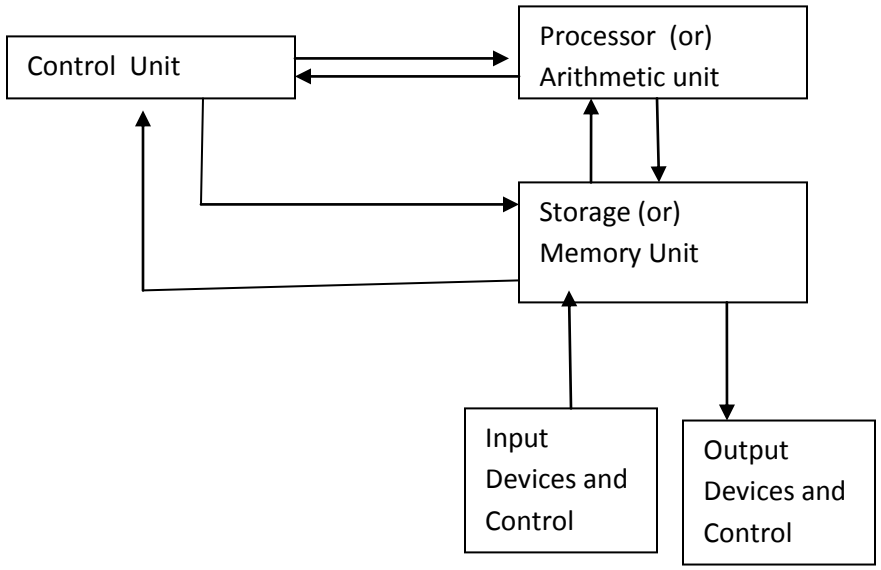
1. Number System

A number system relates quantities and symbols.In digital system how information is represented is key and there are different radices, i.e. number bases, that a numbering system can use.

1.1 Digital computer

Any class of devices capable of solving problems by processing information in discrete form.It operates on data,including letters and symbols,that are expressed in binary form i.e using only two digits 0 and 1.

The block diagram of digital computer is given below:



The memory unit stores programs as well as input, output and intermediate data. The processor unit performs arithmetic and other data processing tasks as specified by the program.The control unit supervises the flow of information between various units. The program and data prepared by the user are transferred into the memory unit by means of an input device such as punch card reader (or) tele typewriter. An output device, such as printer, receives the result of the computations and the printed results are presented to the user.

1.2 Number Representation:

It can have different base values like: binary (base-2), octal (base-8), decimal (base 10) and hexadecimal (base 16),here the base number represents the number of digits used in that numbering system. As an example, in decimal numbering system the digits used are: 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. Therefore the digits for binary are: 0 and 1, the digits for octal are: 0, 1, 2, 3, 4, 5, 6 and 7. For the hexadecimal numbering system, base 16, the digits are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

2. Binary numbers

Numbers that contain only two digit 0 and 1 are called Binary Numbers. Each 0 or 1 is called a Bit, from binary digit. A binary number of 4 bits is called a Nibble. A binary number of 8 bits is called a Byte. A binary number of 16 bits is called a Word on some systems, on others a 32-bit number is called a Word while a 16-bit number is called a Halfword.

Using 2 bit 0 and 1 to form

a binary number of 1 bit, numbers are 0 and 1

a binary number of 2 bit, numbers are 00, 01, 10, 11

a binary number of 3 bit, such numbers are 000, 001, 010, 011, 100, 101, 110, 111

a binary number of 4 bit, such numbers are 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100,1101,1110,1111

Therefore , using n bits there are 2n binary numbers of n bits

Each digit in a binary number has a value or weight. The LSB has a value of 1. The second from the right has a value of 2, the next 4 , etc.,

16	*8*	*4*	*2*	*1*
2^4	2^3	2^2	2^1	2^0

The binary equivalent for some decimal numbers are given below.

Decimal	0	1	2	3	4	5	6	7	8	9	10	11
Binary	0	1	10	11	100	101	110	111	1000	1001	1010	1011

3. Number Base Conversions

3.1 Conversion of decimal number to any number system

Step 1 convert the integer part by doing successive division using the radix of asked number systems.
Step 2 convert the fractional part by doing successive multiplication using radix of asked number system

3.2 Conversion of decimal to binary number system

The radix of asked number system is 2

Convert 87_{10} to $(\ )_2$

2	87	→ 1
2	43	→ 1
2	21	→ 1
2	10	→ 0
2	5	→ 1
2	2	→ 0
	1	

$(1010111)_2$

Convert $(14.625)_{10}$ decimal number to binary number

2	14
2	7-0
2	3-1
1	MSB

$1110)_2$

1st Multiplication Iteration

Multiply 0.625 by 2

0.625 x 2 = 1.25(Product) Fractional part=0.25 Carry=1 **(MSB)**

2nd Multiplication Iteration

Multiply 0.25 by 2

0.25 x 2 = 0.50(Product) Fractional part = 0.50 Carry = 0

3rd Multiplication Iteration

Multiply 0.50 by 2

0.50 x 2 = 1.00(Product) Fractional part = 1.00 Carry = 1 **(LSB)**

$(101)_2$

The binary number of $(16.625)_{10}$ is $(1110.101)_2$

3.3 Conversion of decimal to octal number system

The radix of asked number system is 8

Convert $(264)_{10}$ decimal number to octal number

$8\overline{)264_{10}}$ → 33

24

24

24

0 → 0 (LSD)

$8\overline{)33}$ → 4

32

1 → 1

$8\overline{)4}$ → 0

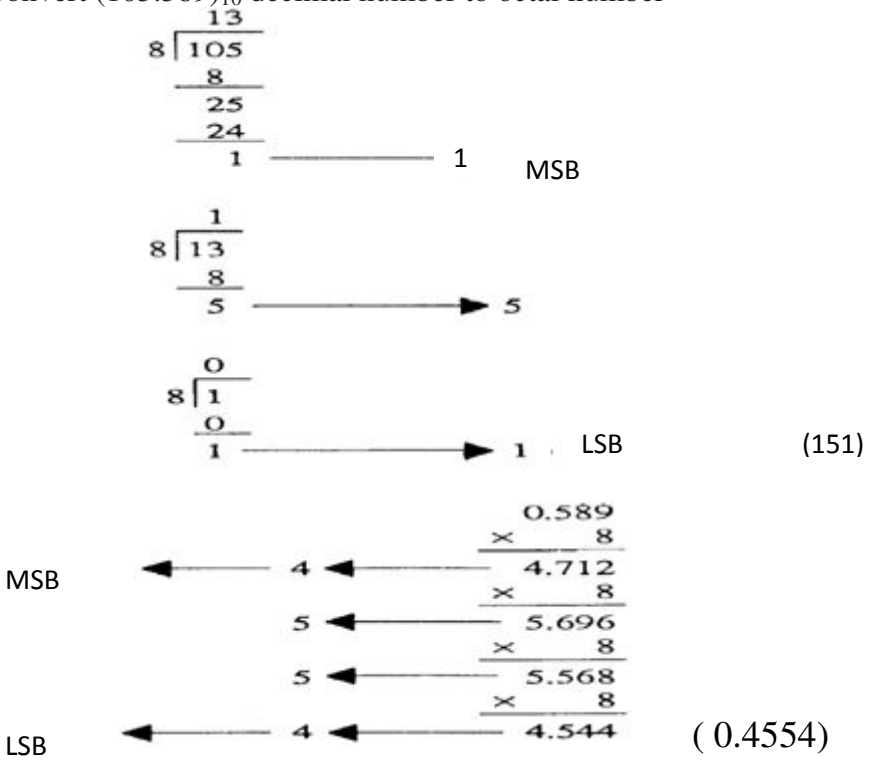
0

4 → 4 (MSD)

$(410)_8$

The octal number of $(264)_{10}$ is $(410)_8$

Convert $(105.589)_{10}$ decimal number to octal number



The octal number of $(105.589)_{10}$ is $(151.4554)_8$

3.4 Conversion of decimal to Hexadecimal number system

The radix of asked number system is 16

Convert $(1693)_{10}$ decimal number to Hexadecimal number

1693/16 = 105 Reminder (13) D (LSB)
105/16 = 6 Reminder 9
6/16 = 0 Reminder 6 (MSB)

(1693)10 (69D)16

Convert $(1693.0628)_{10}$ decimal fraction to hexadecimal fraction $(?)_{16}$

1693/16 = 105 Reminder (13) D (LSB)
105/16 = 6 Reminder 9
6/16 = 0 Reminder 6 (MSB)
(69D)

Multiply 0.0628 by 16

0.0628 x 16 = 1.0048(Product) Fractional part=0.0048 Carry=1 (MSB)

Multiply 0.0048 by 16

0.0048 x 16 = 0.0768(Product) Fractional part = 0.0768 Carry = 0

Multiply 0.0768 by 16

0.0768 x 16 = 1.2288(Product) Fractional part = 0.2288 Carry = 1

Multiply 0.2288 by 16

0.2288 x 16 = 3.6608(Product) Fractional part = 0.6608 Carry = 3 (LSB)
(.1013)

$(1693.0628)_{10} = (69D.1013)_{16}$

3.5 Conversion of any number system to decimal number system

In general the numbers can be represented as

$N = A_{n-1} r_{n-1} + = A_{n-2} r_{n-2} + \ldots\ldots + A_1 r^1 + A_0 r^0 + A_{-1} r^{-1} + A_{-2} r^{-2} + \ldots\ldots$

Where n= number in decimal
A= digit
r= radix of number system
n= The number of digits in the integer portion of number
m= the number of digits in the fractional portion of number

3.6 Conversion of binary to decimal number system

Convert $(101.101)_2 = (?)_{10}$

101.101

$= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 . \ 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$

$= 1 \times 4 + 0 \times 2 + 1 \times 1 . \ 1 \times (1 / 2) + 0 \times (1 / 4) + 1 \times (1 / 8)$

$= 4 + 0 + 1 . (1 / 2) + 0 + (1 / 8)$

$= 5 + 0.5 + 0.125$
$= 5 . 625$
Therefore $(1\ 0\ 1 . 1\ 0\ 1)_2 = (5.625)_{10}$

3.7 Conversion of octal to decimal number system

Convert $(128)_8 = (?)_{10}$

$123_8 = 1*8^2 + 2*8^1 + 3*8^0 = 64 + 16 + 3 = 73$

the decimal equivalent of the number 123_8 is 73_{10}

Convert $(2\ 1.\ 2\ 1)_8 = (?)_{10}$

2 1. 2 1

$= 2 \times 8^1 + 1 \times 8^0 . \ 2 \times 8^{-1} + 1 \times 8^{-2}$
$= 2 \times 8 + 1 \times 1 . \ 2 \times (1 / 8) + 1 \times (1 / 64)$
$= 16 + 1 . (0.\ 2\ 5) + (0.\ 0\ 1\ 5\ 6\ 2\ 5)$
$= 17 + 0.\ 265625$
$= 17.\ 265625$

Therefore $(2\ 1.\ 2\ 1)_8 = (1\ 7.\ 2\ 6\ 5\ 6\ 2\ 5)_{10}$

3.8 Conversion of hexadecimal to decimal number system

Convert $(E\ F.\ B\ 1)_{16} = (?)_{10}$

$= E \times 16^1 + F \times 16^0 . \ B \times 16^{-1} + 1 \times 16^{-2}$
$= 14 \times 16 + 15 \times 1 . \ 11 \times (1 / 16) + 1 \times (1 / 256)$
$= 224 + 15 + (0.\ 6\ 8\ 7\ 5) + (0.\ 0\ 0\ 3\ 9\ 0\ 6\ 2\ 5)$
$= 239 + 0.\ 6914$
$= 239.\ 691406$
Therefore $(E\ F.\ B\ 1)_{16} = (2\ 3\ 9.\ 6\ 9\ 1\ 4\ 0\ 6)_{10}$

Convert $(0.9D9)16 = (?)_{10}$

$= 0 \times 160. \ 9 \times 16\text{-}1 + D \times 16\text{-}2 + 9 \times 16\text{-}3$
$= 0 \times 1. \ 9 \times (1 / 16) + 13 \times (1 / 256) + 9 \times (1 / 4096)$
$= 0 . (0.\ 5625) + (0.\ 050781) + (0.\ 0021972)$
$= 0. \ (0.\ 6154782)$

= 0. 6154782

3.9 Conversion of binary to octal number system

Convert $(101101001)_2$ to ($)_8$

Divide the binary into group of three digits from LSB we will find the following pattern 101|101|001 Now writing the equivalent decimal number of each group we get 5 | 5 | 1 So the equivalent octal number is 551_8

Convert 11001100.101 to ($)_8$

011|001|100. |101|

3 1 4 . 5

So the equivalent octal number is 314.5

3.10 Conversion of binary to hexadecimal number system

Convert 111100010 to ($)_{16}$

Divide the binary into group of four digits from LSB

0001|1110|0010

Now writing the equivalent hexadecimal number of each group

1|E|2

So the equivalent Hexa decimal number is $1E2_{16}$

Convert 11000011001.101 to ($)_{16}$

0110|0001|1001|.1010|

6 1 9 . A

So the equivalent Hexa decimal number is $619.A_{16}$

3.11 Conversion of octal number system to hexa decimal number system

Convert $(25)_8$ to ($)_{16}$

First convert octal to binary

The binary equivalent of 25 is 010101

Divide the binary into group of four digits from LSB

0001|0101

1 5

So the equivalent Hexa decimal number is 15_{16}

3.12 Conversion of hexa decimal number system to octal number system

Convert $(1A.2B)_{16}$ to ($)_8$

First convert hexadecimal to binary

The binary equivalent of 1A.2B is 00011010.00101011

Divide the binary into group of Three digits

011|010|.|001|010|110

3 2 . 1 2 6

so the equivalent octal number is 32.126_8

4. COMPLEMENTS

In digital computers to simplify the subtraction operation and for logical manipulation complements are used . There are two types of complements for each radix system the radix complement and diminished radix complement. The first is referred to as the r's complement and the second as the (r-1)'s complement.

r's Complement

Given a positive number N in base r with an integer part of n digits, the r's complement of N is defined as r^n-N if N≠0 and 0 if N=0

(r-1)'s Complement

Given a positive number N in base r with an integer part of n digits and a fraction part of m digits, the (r-1)'s complement of N is defined as $r^n-r^{-m}-N$

Subtraction with r's complement

- The direct method of subtraction uses the borrow concept
- When subtraction is implemented by means of digital components, this method is found to be less efficient. So, instead the following procedure can be followed.

The subtraction of two positive numbers (M-N), both of base r, may be done as follows.

(1) Add the minuend M to the r's complement of the subtrahend N.
(2) Inspect the result obtained in step 1 for an end carry.
    - If an end-carry occurs, discard it.
    - If an end-carry does not occurs, take the r's complement of the number obtained in step 1 and place a negative sign in front.

Subtraction with (r-1)'s Complement

- The procedure for subtraction with (r-1)'s complement is same as r's complement except for end-around carry.
- The subtraction of M-N, both positive numbers in base r, may be calculated in the following manner.
    1. Add the minuend M to the (r-1)'s complement of the subtrahend N.
    2. Inspect the result obtained in step 1 for an end carry.
        - If an end-carry occurs, add 1 to the least significant digit (end-around carry)
        - If an end-carry does not occur, take the (r-1)'s complement of the number obtained in step 1 and place a negative sign in front.

It is classified into four types they are 1's complement , 2's complement , 9's complement and 10's complement.

4.1 <u>1's complement representation:</u> The 1's complement of a binary number is the number that results when we change all 1's to zeros and the zeros to ones.

2's complement representation:

The 2's complement is the binary number that results when we add 1 to the 1's complement.

Problems related to 1's complement and 2's complement :

1. *Express the following numbers in sign magnitude 1's and 2's complement :*
i) –56 ii)107

Solution : i) – 56

$56 = 0\,1\,1\,1\,0\,0\,0$

$-56 = 1\,0\,0\,0\,1\,1\,1$ 1's Complement

$+\,1$

$= 1\,0\,0\,1\,0\,0\,0$ 2's Complement

ii) 107 $107 = 0\,1\,1\,0\,1\,0\,1\,1$

$-107 = 1\,0\,0\,1\,0\,1\,0\,0$ 1's Complement

$+\,1$

$= 1\,0\,0\,1\,0\,1\,0\,1$ 2's Complement.

2. *Find 2's complement of* $(1\ 0\ 0\ 1)_2$

Solution :

1 0 0 1	number
0 1 1 0	1's complement
+ 1	
0 1 1 1	2's complement

3. *Find 2's complement of* $(1\ 0\ 1\ 0\ \ 0\ 0\ 1\ 1)_2$

Solution :

1 0 1 0 0 0 1 1	number
0 1 0 1 1 1 0 0	1's complement
+ 1	
0 1 0 1 1 1 0 1	2's complement

4.2 1's complement subtraction

Subtraction of binary numbers can be accomplished by the direct method by using the 1's complement method, which allows to perform subtraction using only addition . for subtraction of two numbers we have two cases.

1. Subtraction of smaller number from larger number and
2. Subtraction of larger number from smaller number.

1's complement Subtraction of smaller number from larger number

Method:

1. Determine the 1's complement of the smaller number.
2. Add the 1's complement to the larger number.
3. Remove the carry and add it to the result.
   This is called end -around carry.

4. *Subtract* 101011_2 *from* 111001_2 *using the 1's complement method.*

Solution :

```
    1 1 1 0 0 1
+   0 1 0 1 0 0   1's complement of 1 0 1 0 1 1
-----------------
  ① 0 0 1 1 0 1
    └──────→+ 1   Add end-around carry
-----------------
    0 0 1 1 1 0   Final answer
```

1's complement Subtraction of larger number from smaller number
Method:

1. Determine the 1's complement of the larger number.
2. Add the 1's complement to the smaller number.
3. Answer is in 1's complement form. To get the answer in true form take the 1's complement and assign negative sign to the answer.

5. *Subtract* 111001_2 *from* 101011_2 *using the 1's complement method.*

Solution :

```
    1 0 1 0 1 1
+   0 0 0 1 1 0   1's complement of 1 1 1 0 0 1
-----------------
    1 1 0 0 0 1   Answer in 1's complement form
-   0 0 1 1 1 0   Answer in true form
```

Advantages of 1's complement subtraction :

1. The 1's complement subtraction can be accomplished with an binary adder. Therefore , this method is useful in arithmetic logic circuits.
2. The 1's complement of a number is easily obtained by inverting each bit in the number.

4.3 2's complement Subtraction:

Like 1's complement subtraction, in 2's complement subtraction, the subtraction is accomplished by only addition.

2's complement Subtraction of smaller number from larger number
Method

1. Determine the 2's complement of the smaller number.
2. Add the 2's complement to the larger number.
3. Discard the carry.

6. *Subtract* 101011_2 *from* 111001_2 *using the 2's complement method.*

Solution :

```
    1 1 1 0 0 1
+   0 1 0 1 0 1   2's complement of 1 0 1 0 1 1
-----------------
    0 0 1 1 1 0
    0 0 1 1 1 0   Final answer
```

2's complement Subtraction of larger number from smaller number
Method:

1. Determine the 2's complement of the larger number.
2. Add the 2's complement to the smaller number.

3. Answer is in 2's complement form. To get the answer in true form take the 2's complement and assign negative sign to the answer.

7. *Subtract* 111001_2 *from* 101011_2 *using 2's complement method.*

Solution :

```
  1 0 1 0 1 1
+ 0 0 0 1 1 1     2's complement of 1 1 1 0 0 1
  ---------
  1 1 0 0 1 0     Answer in 2's complement form
- 0 0 1 1 1 0     Answer in true form
```

4.4 9's complement and 10's complement

Before knowing about 9's complement and 10's complement we should know why they are used and why their concept came into existence. Addition of signed BCD numbers can be performed by using 9's and 10's complement. The complements are used to make the arithmetic operations in digital system easier. Various topics and related problems we going to see here are

1. 9s complement
2. 10s complement
3. 9s complement subtraction
4. 10s complement subtraction

Now first of all let us know what 9's complement is and how it is done. To obtain the 9,s complement of any number we have to subtract the number with (10n - 1) where n = number of digits in the number, or in a simpler manner we have to divide each digit of the given decimal number with 9. The table 1. will explain the 9's complement more easily.

Table 1. 9's complement equivalent for decimalo numbers

Decimal digit	9s complement
0	9
1	8
2	7
3	6
4	5
5	4
6	3
7	2
8	1
9	0

Now coming to 10's complement, it is relatively easy to find out the 10's complement after finding out the 9,s complement of that number. We have to add 1 with the 9,s complement of any number to obtain the desired 10's complement of that number. Or if we want to find out the 10's complement directly, we can do it by following the formula, (10^n - number), where n = number of digits in the number. An example is given below to illustrate the concept of obtaining 10's complement

A decimal number 456, find 9's complement and 10's complement of this number

```
  999
(-) 456
-------
  543
```

10's complement of that no. is

```
  543
(+)1
-------
  544
```

In 9's complement subtraction when 9's complement of smaller number number is added to the larger number carry is generated. It is necessary to add this carry to the result. (this is called an end around carry).when larger number is subtracted from the smaller number, there is no carry, and the result is in 9's compliment form and negative. This is explained with following examples.

Subtraction using 9's complements:

	Regular Subtraction	9's Complement Subtraction	
(a)	8	8	
	− 2	+ 7	9's complement of 2
	6	(1) 5	
		+ 1	Add carry to result
		6	
(b)	9	9	
	− 5	+ 4	9' complement of 5
	4	(1) 3	
		+ 1	Add carry to result
		4	
(c)	4	4	
	− 8	+ 1	9' complement of 8
	−4	5	9's complement of result (No carry indicates that the answer is negative and in complement form)
		−4	

Steps for 9's complement BCD subtraction

1. Find the 9's complement of a negative number.
2. Add two numbers using BCD addition
3. If carry is generated add carry to the result otherwise find the 9's complement of the result.

9. *Perform each of the following decimal subtractions in 8-4-2-1 BCD using 9's complement method.* a) 79 − 26 b) 89 − 54

Solution :

a) 79 − 26

79		0	1	1	1		1	0	0	1	
− 26	+	0	1	1	1		0	0	1	1	73 − 9's complement for BCD 26
53		1	1	1	0		1	1	0	0	
							0	1	1	0	1100 > 9 so add 6
		1	1	1	0	1	0	0	1	0	
				+	1	←					Propagate carry
		1	1	1	1		0	0	1	0	
	+	0	1	1	0						Add 6
		1	0	1	0	1	0	0	1	0	
		→				+				1	End around carry
		0	1	0	1		0	0	1	1	BCD for 53

b) 89 − 54

89	1	0	0	0		1	0	0	1	
− 54	0	1	0	0		0	1	0	1	(45) 9's complement of 54 BCD
35	1	1	0	0		1	1	1	0	
					+	0	1	1	0	1110 > 9 so add 6
	1	1	0	0		1	0	1	0	0
			+	1	←					Propagate carry
	1	1	0	1			0	1	0	0
	0	1	1	0						Add 6
	1	0	0	1	1		0	1	0	0
	→					+				1	End around carry
	0	0	1	1			0	1	0	1	BCD for 35

Subtraction using 10's complements:

The 10's complement of the decimal is equal to 9's complement plus 1. The 10's complement can be used to perform subtraction by adding the minuend to the 10's complement of the subtrahend and dropping the carry. This is explained with following examples.

	Regular Subtraction	10's Complement Subtraction	
(a)	8	8	
	− 2	+ 8	10's complement of 2
	6	~~1~~6	Drop carry
(b)	9	9	
	− 5	+ 5	10's complement of 5
	4	~~1~~4	Drop carry
(c)	4	4	
	− 8	+ 2	10's complement of 8
	−4	6	10's complement of result
		↓	(No carry indicates that the answer is negative and in the 10's complement form)
		−4	

Steps for 10's complement BCD subtraction

1. Find the 10's complement of a negative number.
2. Add two numbers using BCD addition
3. If carry is not generated find the 10's complement of the result.

5.SIGNED NUMBERS

- Digital systems like computer, must be able to handle both positive and negative numbers.
- A signed binary number consists of both sign and magnitude information.
- The sign indicates whether a number is positive or negative.

5.1 Representation

There are three forms in which the signed integer (whole numbers) can be represented. They include,

1. Sign – Magnitude Form – Rarely used
2. 1's Complement Form
3. 2's Complement Form – Mostly used

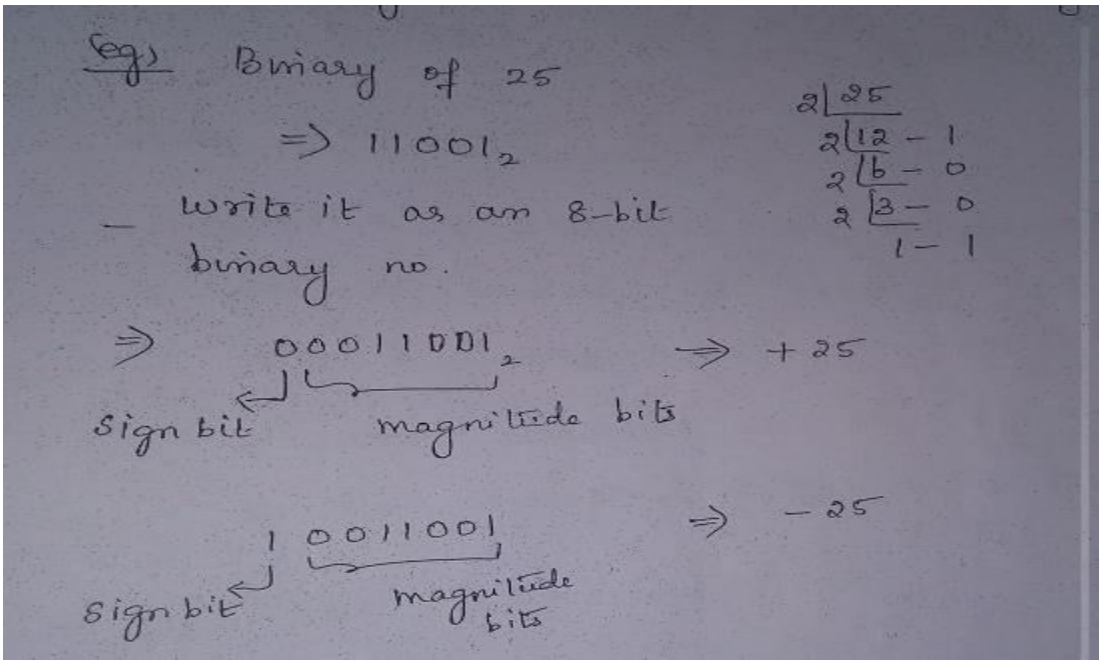
Note:

Sign bit – leftmost bit in a signed binary numbers

- 0 for positive, 1 for negative

5.11 Sign Magnitude Form

- Here, leftmost bit is the sign bit.
- Remaining bits are magnitude bits.
- Magnitude bits are in true binary.



5.12 1's Complement Form

- In this Form, positive numbers are represented the same way as positive sign-magnitude numbers.

- Negative numbers, are the 1's complement of the corresponding positive numbers.

(eg)

+25 is represented as,

00011001 → same as sign-magnitude form

-25 is represented as,

11100110 → 1's complement of +25

5.13 2's Complement Form

- Positive numbers in 2's complement form are represented as same as in sign-magnitude and 1's Complement Form.
- Negative numbers are the 2's complement of the corresponding positive numbers

(eg)

+25 is represented as,

00011001 → same as sign-magnitude form

-25 is represented as,

11100110 +

1

11100111_2 → 2's complement of +25

Decimal value of Signed Numbers

(1) Sign Magnitude

- Decimal values of positive and negative numbers in this form are determined by summing the weights in all the magnitude – bit positions.
- The sign is determined by examining the sign bit.

(eg) 1. Determine the decimal value of this signed binary number expressed in sign – magnitude. 10010101

Soln:

- The seven magnitude bits and their powers of 2 weights are as follows.

  1 0010101

  ↓ $2^6 2^5 2^4 2^3 2^2 2^1 2^0$

  Sign bit
- Summing weights where there are 1's.

  → 16+4+1 = 21
- Since, the sign bit is 1, the decimal number is -21

(2) 1's Complement

- Decimal values of positive numbers in this form are determined by summing the weights in all bit postions.
- Decimal values of negative numbers are determined by assigning a negative value to the weight of the sign bit, summing all the weights where there are 1's and adding 1 to the result.

(eg) Determine the decimal value of the signed binary number expressed in 1's complement 11101000

Soln:

- The bits and their powers- of- two weights are as follows.

Note: for sign bit, it is -2^7 (or) -128

1 1 1 0 1 0 0 0

$-2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

- Summing the weights where there are 1's

-128+64+32+8 = -24 (if +ve, write this as the result)

- Since, it is a negative number, add 1 to the result

-24+1 = -23

(3) 2's Complement

- Decimal values of positive and neagative numbers in this form are determined by summing the weights in all bit positions.
- The weight of the sign – bit in a negative number is given a negative value.

(eg): Determine the decimal values of the signed binary numbers expressed in 2's complement from 10101010

Soln:

- The bits and their corresponding powers of -2 weights are as follows

1 0 1 0 1 0 1 0

$-2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

- Summing weights where there are 1's

-128+32+8+2 = -86

Range of signed integer numbers that can be Represented

- Since 8-bit (1byte) grouping is common in most computers, the illustrations are all 8-bits. With 8-bits, we can represent 256 different numbers.
- With 16-bits (2 bytes), we can represent 65,536 different numbers.
- With 32-bits (4 bytes), we can represent 4.295×10^9 different numbers.

The formula for finding the number of different combinations of n-bits is,

Total combinations = 2^n

Range of values for n-bit numbers is,

$-(2^{n-1})$ to $+ (2^{n-1} - 1)$

So, for 8 bits the range is,

-128 to +127

For 16 bits the range is,

-32768 to +32767 etc

5.2 Arithmetic operations with Signed Numbers

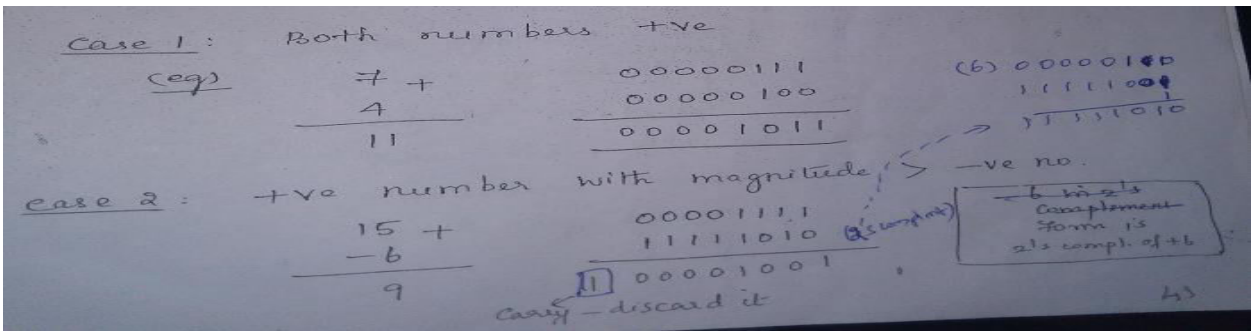
- Here, we use 2's complement representation

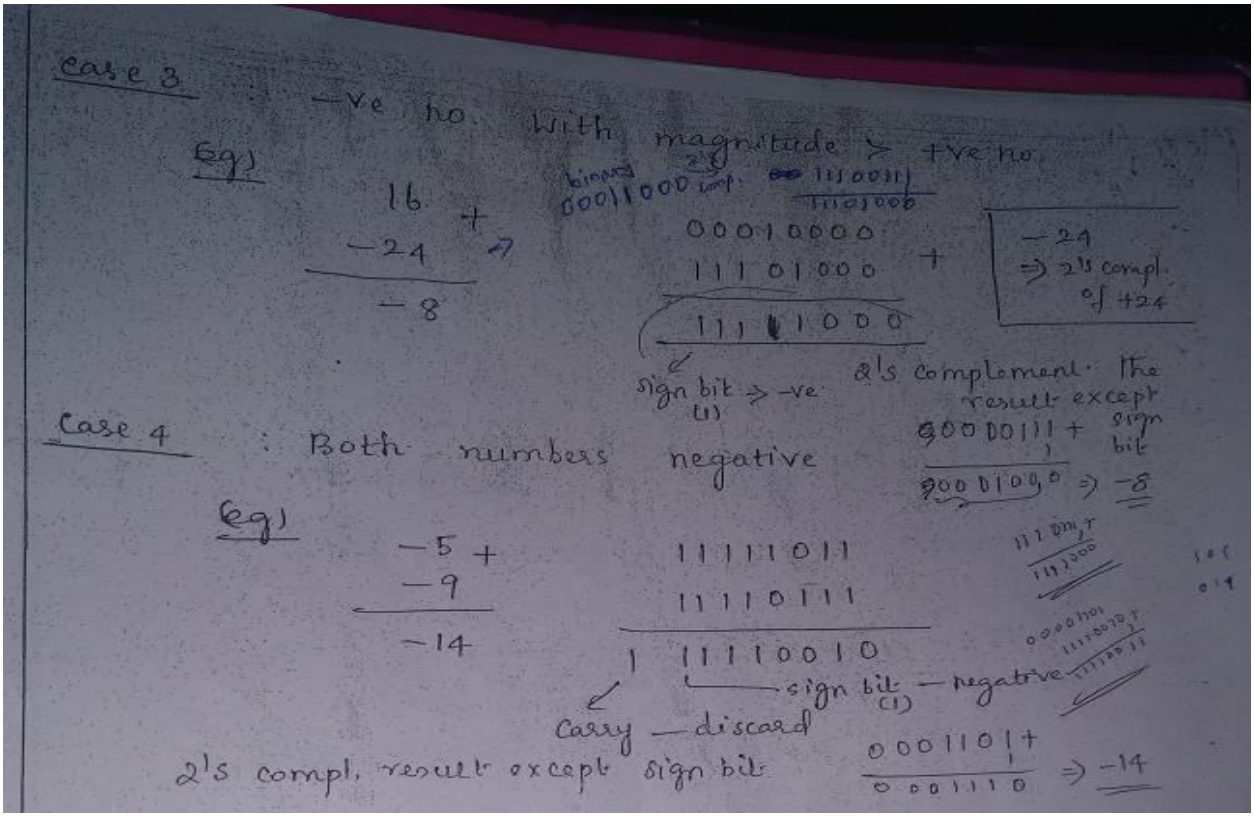
Addition

- The two numbers in an addition are the addend and the augend
- The result is sum.
- There are four cases that can occur when two signed binary numbers are added.

(1) Both numbers positive.
(2) Positive number with magnitude larger than negative number.
(3) Negative number with magnitude larger than positive number
(4) Both numbers negative.

Case 1: Both numbers +ve





Subtraction

- It is a special case of addition.
- The two numbers in subtraction are subtrahend and minuend.
- The result is the difference.
- To subtract +6 from +9, it is also equivalent to add -6 to +9.
- So, to subtract two signed numbers, take the 2's complement of the subtrahend and add. Discard any final carry bit.

6. BINARY ARITHMETIC

6.1 BINARY ADDITION

The binary addition table is as follows:

A+B	SUM	CARRY
0+0	0	0
0+1	1	0
1+0	1	0
1+1	0	1

Illustration 1:

Add $(1010)_2$ and $(0011)_2$

```
  1010   (Augend)
  0011   (Addend)
 ----------------------
  1101    (sum)
 ----------------------
```

The addition manipulated above as follows.

Step 1: The least significant bits are added, i.e. 0+1 =1 with a carry of 0

Step 2: The carry in the previous is added to the next higher significant bits, i.e. 0+1+1= 0 with a carry 1.
Step 3: The carry in the previous is added to the next higher significant bits, i.e. 1+0+0 =1 with a carry 0.
Step 4: The preceding carry is added to the most significant bit i.e. 0+1+0 = 1 with a carry 0.
Thus the sum is 1101.

6.2 BINARY SUBTRACTION

The binary subtraction table is as follows:

A-B	DIFFERENCE	BORROW
0-0	0	0
0-1	1	1
1-0	1	0
1-1	0	0

Illustration 1:

Subtract $(0101)_2$ from $(1011)_2$

```
1011    (Minuend)
0101    (Subtrahend)
-------------------------
0110    (Difference)
-------------------------
```

The steps are described below

Step1: the LSB in the first column are 1 and 1. Hence, the difference is 1 - 1 = 0
Step2: The column, the subtraction is performed as 1 – 0 = 1
Step3: In the third column, the difference is given by 0 – 1 =1
Step 4: In the fourth column (MSB), the difference is given by 0 – 0 = 0 since 1 is borrowed for third column.

6.3 BINARY MULTIPLICATION

The binary multiplication table is as follows:

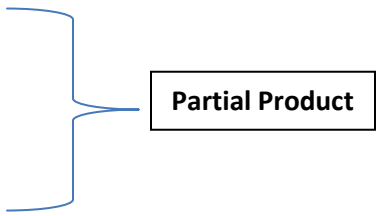
A *B	PRODUCT
0 * 0	0
0 * 1	0
1 * 0	0
1 * 1	1

- Binary multiplication uses add and shift process
- Binary multiplication is similar to decimal multiplication.

Illustration 1:

Multiplicand * Multiplier

```
        10110.1x01001.1
---------------------------------
                101101
          101101
           000000
          000000
          101101
          000000
---------------------------------
        011010101.11
---------------------------------
```



(Final product)

The steps are described below

Step 1: The LSB of the multiplier is taken. If multiplier bit is 1, the multiplicand is copied as such and if the multiplier bit is 0 zero is placed in all the bit positions.
Step 2: The next higher significant bit of the multiplier is taken and, the partial product is written with the shift to the left, as in step 1.
Step 3: step 2 is repeated for all other higher significant bits.
Step 4: The partial product terms are added which gives the actual product of multiplier and the multiplicand.

6.4 BINARY DIVISION:

The binary division table is as follows:

A÷B	Result
0÷0	Not allowed
0÷1	0
1÷0	Not allowed
1÷1	1

- Binary division uses subtract and shift process
- Binary division is similar to decimal division.
- Division by 0 is meaningless.

Illustration 1:

Dividend ÷ Divisor
11011.1÷ 101

```
        101.1          (QUOTIENT)
DIVISOR 101 √11011.1   (DIVIDEND)
             101
            ----------
              111
              101
            ----------
              101
              101
            ----------
                0
            ----------
```

7.BINARY CODES

Binary codes are codes which are represented in binary system with modification from the original one. The group of symbols is called as a code. The digital data is represented, stored and transmitted as group of binary bits. This group is also called as binary code. The binary code is represented by the number as well as alphanumeric letter.

Advantages of Binary Code

Following is the list of advantages that binary code offers.

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

7.1 Classification of binary codes:The codes are broadly categorized into following four categories.

- Weighted Codes
- Non-Weighted Codes
- Binary Coded Decimal Code
- Alphanumeric Codes

- Error Codes

7.1.1 Weighted codes: Weighted binary codes are those binary codes which obey the positional weight principle. Each position of the number represents a specific weight

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

For example, in 8421BCD code, 1001 the weights of 1, 0, 0, 1 (from left to right) are 8, 4, 2 and 1 respectively. The codes 8421BCD, 2421BCD, 5211BCD are all weighted codes.

7.1.2 Non-weighted codes: The non-weighted codes are not positionally weighted. In other words, each digit position within the number is not assigned a fixed value (or weight).

Examples are

- Excess-3
- Gray code

DECIMAL	EXCESS - 3	GRAY CODE
0	0011	0000
1	0100	0001
2	0101	0011

6.1.3 EXCESS – 3 CODES:-

- This is another form of BCD code, in which each decimal digit is coded into a 4-bit binary code.
- The code for each decimal digit is obtained by adding decimal 3 to the natural BCD code of the digit.

GRAY CONVERSION:-

- Record the mostsignificant bit add the binary MSB to the next significant bit of the Gray code.
- Record the result, ignoring carrier continue the process, until the LSB is reached.

REFLECTIVE CODES: A code is reflective when the code is self-complementing. In otherwords, when the code for 9 is the complement the code for 0, 8 for 1, 7 for 2, 6 for 3 and 5 for 4. 2421BCD, 5421BCD and Excess-3 code are reflective codes.

SEQUENTIAL CODES: In sequential codes, each succeeding 'code is one binary number greater than its preceding code. This property helps in manipulation of data. 8421 BCD and Excess-3 are sequential codes.

ALPHANUMERIC CODES: Codes used to represent numbers, alphabetic characters, symbols and various instructions necessary for conveying intelligible information. ASCII, EBCDIC, UNICODE are the most-commonly used alphanumeric codes.

8.Decimal code

Binary codes for decimal digits require a minimum of four bits. Numerous different codes can be obtained by arranging four or more bits in ten distinct possible combinations. A few possibilities are tabulated.

DECIMAL DIGIT	8421	84-2-1	7421	5421	2421	BIQUINARY
	8 4 2 1	8 4 -2 -1	7 4 2 1	5 4 2 1	2 4 2 1	5 0 4 3 2 1 0
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 0 0 0 1
1	0 0 0 1	0 1 1 1	0 0 0 1	0 0 0 1	0 0 0 1	0 1 0 0 0 1 0
2	0 0 1 0	0 1 1 0	0 0 1 0	0 0 1 0	0 0 1 0	0 1 0 0 1 0 0
3	0 0 1 1	0 1 0 1	0 0 1 1	0 0 1 1	0 0 1 1	0 1 0 1 0 0 0
4	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0	0 1 1 0 0 0 0
5	0 1 0 1	1 0 1 1	0 1 0 1	0 1 0 1	1 0 1 1	1 0 0 0 0 0 1
6	0 1 1 0	1 0 1 0	0 1 1 0	0 1 1 0	1 1 0 0	1 0 0 0 0 1 0
7	0 1 1 1	1 0 0 1	1 0 0 0	0 1 1 1	1 1 0 1	1 0 0 0 1 0 0
8	1 0 0 0	1 0 0 0	1 0 0 1	1 0 1 1	1 1 1 0	1 0 0 1 0 0 0
9	1 0 0 1	1 1 1 1	1 0 1 0	1 1 0 0	1 1 1 1	1 0 1 0 0 0 0

9.Error detection code

In data transmission, Interference and physical defects in the communication medium can cause random bit errors. As the signal is transmitted through a media, the signal gets corrupted because of noise and distortion. Therefore the media is not reliable. To achieve a reliable communication through this unreliable media, there is need for detecting the error in the signal so that suitable mechanism can be devised to take corrective actions.

Error coding is a method of detecting and correcting these errors to ensure information is transferred intact from its source to its destination

The errors can be divided into two types:

• Single-bit Error: only one bit of given data unit (such as a byte, character, or data unit) is changed from 1 to 0 or from 0 to 1.

• Burst Error: two or more bits in the data unit have changed from 0 to 1 or vice-versa. (Here doesn't necessary means that error occurs in consecutive bits)

Error Detecting Codes:

Basic approach used for error detection is the use of redundancy, where additional bits are added to facilitate detection and correction of errors.

Popular techniques are:

• Simple Parity check

• Two-dimensional Parity check

• Checksum

• Cyclic redundancy check

Detecting Errors using simple parity check

Suppose we are transmitting 7-bit ASCII characters. A parity bit is added to each character to make it 8 bits. Parity can detect all single-bit errors

–If even parity is used and a single bit changes, it will change the parity to odd, which will be detected at the receiver end

–The receiver end can detect the error, but cannot correct it because it does not know which bit is erroneous

Parity can also detect some multiple-bit errors

Table 1 shows the four bit data word and its corresponding code words

Decimal value	Data block	Parity bit	Code word
0	0000	0	00000
1	0001	1	00011
2	0010	1	00101

3	0011	0	00110
4	0100	1	01001
5	0101	0	01010
6	0110	0	01100
7	0111	1	01111
8	1000	1	10001
9	1001	0	10010
10	1010	0	10100
11	1011	1	10111
12	1100	0	11000
13	1101	1	11011
14	1110	1	11101
15	1111	0	11110

10.Gray Code- Reflection and Self Complementary codes

- Gray Code is a non-weighted code which belongs to a class of codes called minimum change codes.
- Gray Code is an alternative binary representation, devised such that, between any two adjacent numbers, ***only one bit*** changes at a time.

Binary	Dec	Gray
00000	0	00000
00001	1	00001
00010	2	00011
00011	3	00010
00100	4	00110
00101	5	00111
00110	6	00101
00111	7	00100
01000	8	01100
01001	9	01101
01010	10	01111
01011	11	01110
01100	12	01010
01101	13	01011
01110	14	01001
01111	15	01000

- To the left we see three columns of data. These are representations of the same numbers 0-15 in different ways.
    - In the middle is the decimal value.
    - On the left is positional notation binary
    - On the right is Gray code.
- You will notice that, on the right, each adjacent row is different from it's neighbours by no more than one bit.
- The term Gray code is often used to refer to a "reflected" code, or more specifically still, the binary reflected Gray code.

10.1 Self-complementary Code

• A code is said to be self-complementary if the code for 9's complement of N i.e. 9-N can be obtained by interchanging all 0s and 1s.
• Decimal 9 is the complement of code for 0, 8 for 1, 7 for 2 and so on.
• For a code to be self complementing, the sum of all its weights must be 9. digit.8421 and 5421 codes are not self complementing codes whereas 5211,2421,3321, 4321 are self complementing.
• In general, a code is self-complementary if we produce a code by taking the first complement of the digit which is same as 9's complement of the number.

10.2 Reflective code

- Imaged about the centre entries with one bit changed
- Example ï 9ís complement of a reflected BCD code word is formed by changing only one of its bits
- In the Gray code example shown below, the MSB bit alone is changing and the remaining bits is reflected mirror image about the centre. For clarity, the MSB is removed.
- Gray code Reflected property of Gray code

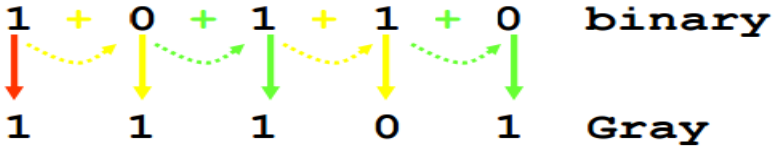
```
0000
0001
0011
0010
0110
0111
0101
0100
1100
1101
1111
1110
1010
1011
1001
1000
```

```
x000
x001
x011
x010
x110
x111
x101
x100
-------- mirror
x100
x101
x111
x110
x010
x011
x001
x000
```

Binary-to-Gray code conversion

- The MSB in the Gray code is the same as corresponding MSB in the binary number.
- Going from left to right, add each adjacent pair of binary code bits to get the next Gray code bit.
- Discard carries.

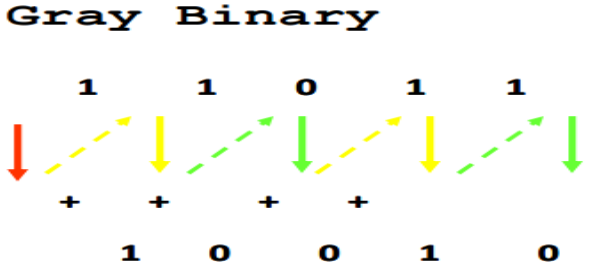
Problem: Convert 10110 to gray code



Gray-to-Binary Conversion

- The MSB in the binary code is the same as the corresponding bit in the Gray code.
- Add each binary code bit generated to the Gray code bit in the next adjacent position.
- Discard carries.

Problem: Convert the Gray code word 11011 to binary



11. Binary-Coded Decimal Code

Although the binary number system is the most natural system for a computer because it is readily represented in today's electronic technology, most people are more accustomed to the decimal system. One way to resolve this difference is to convert decimal numbers to binary, perform all arithmetic calculations in binary, and then convert the binary results back to decimal. This method requires that we store decimal numbers in the computer so that they can be converted to binary. Since the computer can accept only binary values, we must represent the decimal digits by means of a code that contains 1's and 0's. It is also possible to perform the arithmetic operations directly on decimal numbers when they are stored in the computer in coded form.

A binary code will have some unassigned bit combinations if the number of elements in the set is not a multiple power of 2. The 10 decimal digits form such a set. A binary code that distinguishes among 10 elements must contain at least four bits, but 6 out of the 16 possible combinations remain unassigned. Different binary codes can be obtained by arranging four bits into 10 distinct combinations. This scheme is called ***binary-coded decimal*** and is commonly referred to as **BCD**.

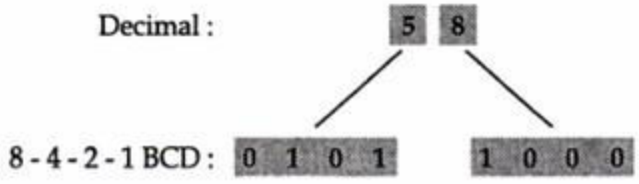
A number with k decimal digits will require $4k$ bits in BCD. Decimal 396 is represented in BCD with 12 bits as 0011 1001 0110, with each group of 4 bits representing one decimal digit. A decimal number in BCD is the same as its equivalent binary number only when the number is between 0 and 9. A BCD number greater than 10 looks different from its equivalent binary number, even though both contain 1's and 0's. Note that the BCD code is not self-complementing. Moreover, the binary combinations 1010 through 1111 are not used and have no meaning in BCD. Consider decimal 185 and its corresponding value in BCD and binary:

$(185)_{10} = (0001\ 1000\ 0101)_{BCD} = (10111001)_2$

Decimal	BCD Code			
Digit	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Table 1

In multi digit BCD coding

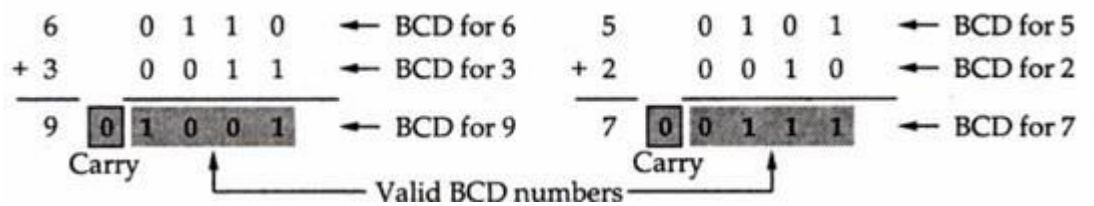


11.1 BCD addition:

The addition of two BCD numbers can be best understood by considering the three cases that occur when two BCD digits are added.

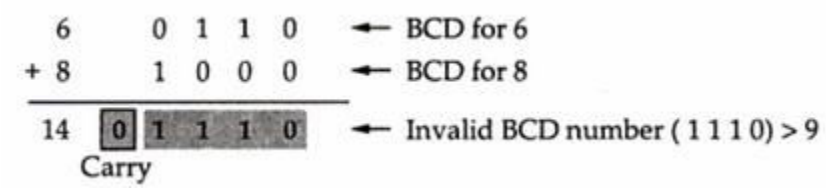
Sum equals 9 or less with carry 0

Let us consider additions of 3 and 6 in BCD.

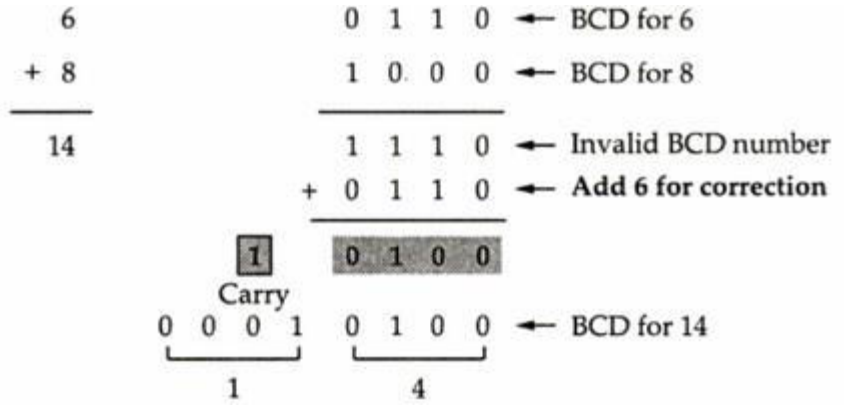


Sum greater than 9 with carry 0

Let us consider addition of 6 and 8 in BCD

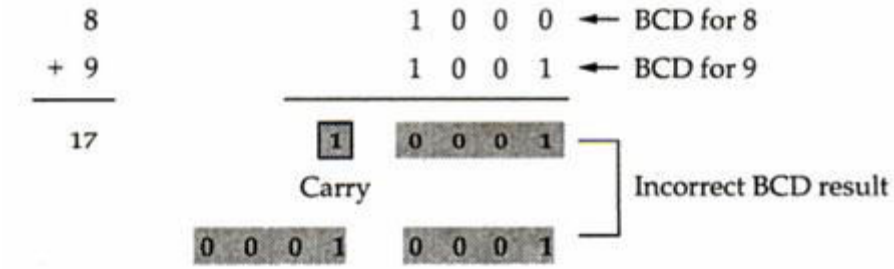


The sum 1110 is an invalid BCD number. This has occurred because the sum of the two digits exceeds 9. Whenever this occurs this occurs the sum has to be corrected by the addition of six (1110) in the invalid BCD number, as shown below

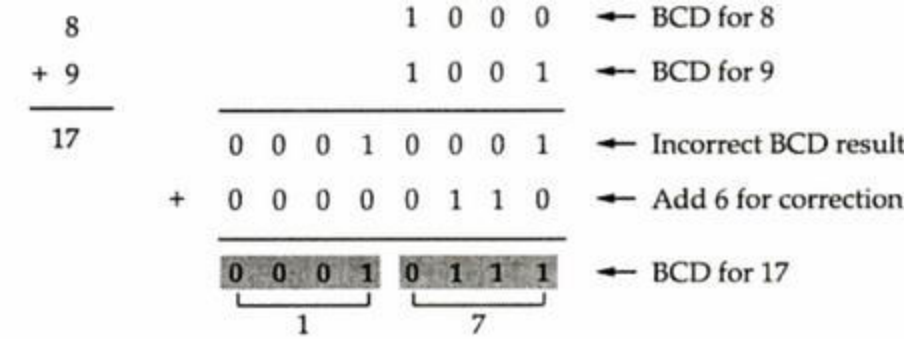


Sum equals 9 or less with carry 1

Let us consider addition of 8 and 9 in BCD



In this case, result (001 0001) is valid BCD number, but it is incorrect. To get the correct BCD result correction factor of 6 has to be added to the least significant digit sum, as shown.



BCD addition procedure

1. Add two BCD numbers using ordinary binary addition.
2. If four bit sum is equal to or less than 9, no correction is needed. The sum is in proper BCD form.

3. If the four bit sum is greater than 9 or if a carry is generated from the four-bit sum, the sum is invalid.
4. To correct the invalid sum, add 0110_2 to the four-bit sum. If a carry results from this addition, add it to the next higher-order BCD digit.

Example 1 : *Perform the code conversion :*

$(137)_{10}$ = *(?) NBCD*

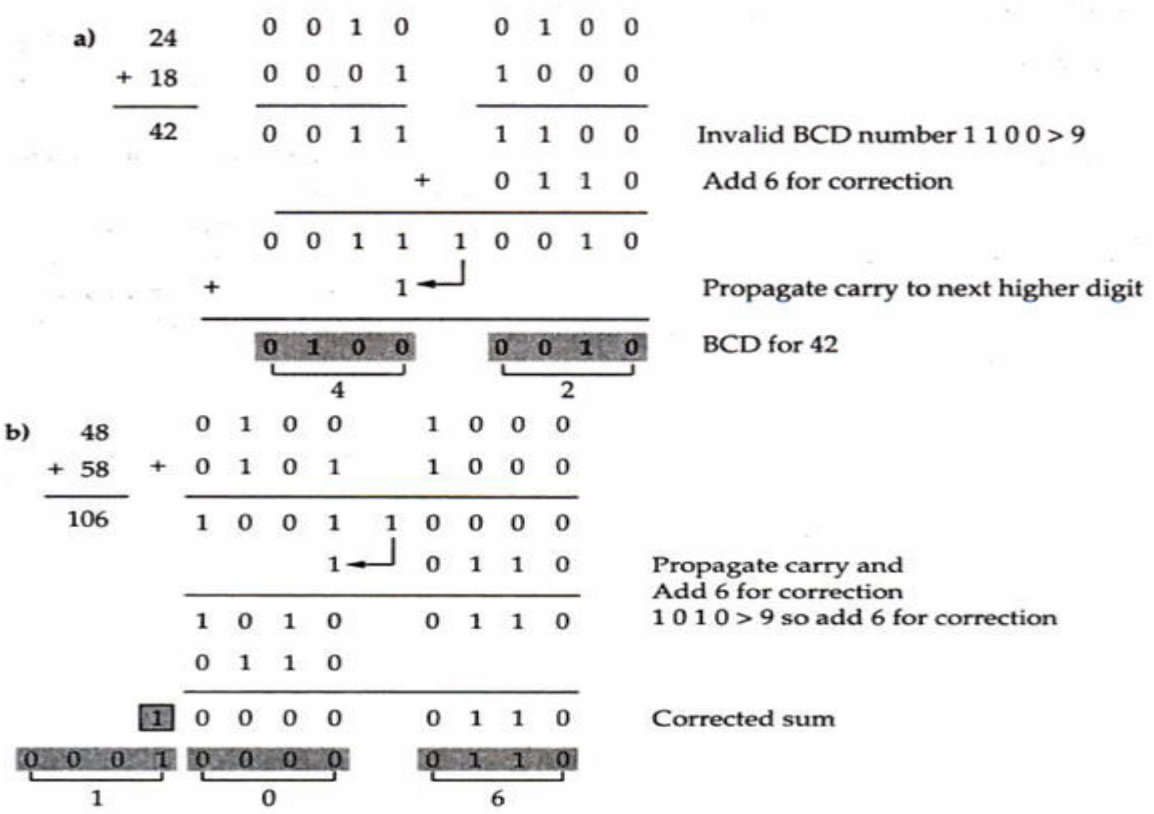
Solution : NBCD = 8421 BCD

$\therefore$ $(137)_{10} = (0001\ 0011\ 0111)_{NBCD}$

Example 2 : *Perform each of the following decimal additions in 8-4-2-1 BCD.*

a) $+\begin{matrix}24\\18\end{matrix}$ *b)* $+\begin{matrix}48\\58\end{matrix}$

Solution :



12.Alphanumeric codes

Alphanumeric codes are sometimes called character codes due to their certain properties. Now these codes are basically binary codes. We can write alphanumeric data, including data, letters of the alphabet, numbers, mathematical symbols and punctuation marks by this code which can be easily understandable and can be processed by the computers. Input output devices such as keyboards, monitors, mouse can be interfaced using these codes. 12-bit Hollerith code is the better known and perhaps the first effective code in the days of evolving computers in early days. During this period punch cards were used as the inputting and outputting data. But nowadays these codes are termed obsolete as many other modern codes have evolved. The most common **alphanumeric codes** used these days are **ASCII code**, **EBCDIC code** and Unicode.

12.1 ASCII Character Code

Many applications of digital computers require the handling not only of numbers, but also of other characters or symbols, such as the letters of the alphabet. For instance, consider a high-tech company with thousands of employees. To represent the names and other pertinent information, it is necessary to formulate a binary code for the letters of the alphabet. In addition, the same binary code must represent numerals and special characters (such as $). An alphanumeric character set is a set of elements that includes the 10 decimal digits, the 26 letters of the alphabet, and a number of special characters. Such a set contains between 36 and 64 elements if only capital letters are included, or between 64 and 128 elements if both uppercase and lowercase letters are included. In the first case, we need a binary code of six bits, and in the second, we need a binary code of seven bits. The standard binary code for the alphanumeric characters is the **American Standard Code for Information Interchange (ASCII)**, which uses seven bits to code 128 characters, as shown in Table below. The seven bits of the code are designated by $b1$ through $b7$, with $b7$ the most significant bit. The letter *A*, for example, is represented in ASCII as 1000001 (column 100, row 0001). The ASCII code also contains 94 graphic characters that can be printed and 34 nonprinting characters used for various control functions.

The graphic characters consist of the 26 uppercase letters (A through Z), the 26 lowercase letters (a through z), the 10 numerals (0 through 9), and 32 special printable characters, such as %, *, and $.characters. Format effectors are characters that control the layout of printing. They include the familiar word processor and typewriter controls such as backspace (BS), horizontal tabulation (HT), and carriage return (CR). Information separators are used to separate the data into divisions such as paragraphs and pages. They include characters such as record separator (RS) and file separator (FS). The communication-control characters are useful during the transmission of text between remote devices so that it can be distinguished from other messages using the same communication channel before it and after it. Examples of communication-control characters are STX (start of text) and ETX (end of text), which are used to frame a text message transmitted through a communication channel.

ASCII is a seven-bit code, but most computers manipulate an eight-bit quantity as a single unit called a *byte*. Therefore, ASCII characters most often are stored one per byte. The extra bit is sometimes used for other purposes, depending on the application.

For example, some printers recognize eight-bit ASCII characters with the most significant bit set to 0. An additional 128 eight-bit characters with the most significant bit set to 1 are used for other symbols, such as the Greek alphabet or italic type font.

DEC	OCT	HEX	BIN	Symbol	Description
0	000	00	00000000	NUL	Null char
1	001	01	00000001	SOH	Start of Heading
2	002	02	00000010	STX	Start of Text
3	003	03	00000011	ETX	End of Text
4	004	04	00000100	EOT	End of Transmission
5	005	05	00000101	ENQ	Enquiry
6	006	06	00000110	ACK	Acknowledgment
7	007	07	00000111	BEL	Bell
8	010	08	00001000	BS	Back Space
9	011	09	00001001	HT	Horizontal Tab
10	012	0A	00001010	LF	Line Feed
11	013	0B	00001011	VT	Vertical Tab
12	014	0C	00001100	FF	Form Feed
13	015	0D	00001101	CR	Carriage Return
14	016	0E	00001110	SO	Shift Out / X-On
15	017	0F	00001111	SI	Shift In / X-O

12.2 EBCDIC

The EBCDIC stands for Extended Binary Coded Decimal Interchange Code. IBM invented this code to extend the Binary Coded Decimal which existed at that time. All the IBM computers and peripherals use this code. It is an 8 bit code and therefore can accommodate 256 characters. Below is given some characters of **EBCDIC code** to get familiar with it.

Char	EBCDIC	HEX	Char	EBCDIC	HEX	Char	EBCDIC	HEX
A	1100 0001	C1	P	1101 0111	D7	4	1111 0100	F4
B	1100 0010	C2	Q	1101 1000	D8	5	1111 0101	F5
C	1100 0011	C3	R	1101 1001	D9	6	1111 0110	F6
D	1100 0100	C4	S	1110 0010	E2	7	1111 0111	F7
E	1100 0101	C5	T	1110 0011	E3	8	1111 1000	F8
F	1100 0110	C6	U	1110 0100	E4	9	1111 1001	F9
G	1100 0111	C7	V	1110 0101	E5	blank	...	...
H	1100 1000	C8	W	1110 0110	E6	.	...	...
I	1100 1001	C9	X	1110 0111	E7	(	...	...
J	1101 0001	D1	Y	1110 1000	E8	+	...	...
K	1101 0010	D2	Z	1110 1001	E9	$	...	...
L	1101 0011	D3	0	1111 0000	F0	*	...	...
M	1101 0100	D4	1	1111 0001	F1	)	...	...
N	1101 0101	D5	2	1111 0010	F2	-	...	...
O	1101 0110	D6	3	1111 0011	F3	/		

13. HAMMING CODE-ERROR DETECTION AND CORRECTION

Hamming code is a set of error-correction code s that can be used to detect and correct bit errors that can occur when computer data is moved or stored.

13.1 Error Detecting Codes

Basic approach used for error detection is the use of redundancy, where additional bits are added to facilitate detection and correction of errors. Popular techniques are: • Simple Parity check • Two-dimensional Parity check • Checksum • Cyclic redundancy check

Simple Parity Checking or One-dimension Parity Check The most common and least expensive mechanism for error- detection is the simple parity check. In this technique, a redundant bit called parity bit, is appended to every data unit so that the number of 1s in the unit (including the parity becomes even). Blocks of data from the source are subjected to a check bit or Parity bit generator form, where a parity of 1 is added to the block if it contains an odd number of 1's (ON bits) and 0 is added if it contains an even number of 1's. At the receiving end the parity bit is computed from the received data bits and compared with the received parity bit, as shown in Fig 1. This scheme makes the total number of 1's even, that is why it is called even parity checking. Considering a 4-bit word, different combinations of the data words and the corresponding code words are given in Table 1. Note that for the sake of simplicity, we are discussing here the even-parity checking, where the number of 1's should be an even number. It is also possible to use odd-parity checking, where the number of 1's should be odd.

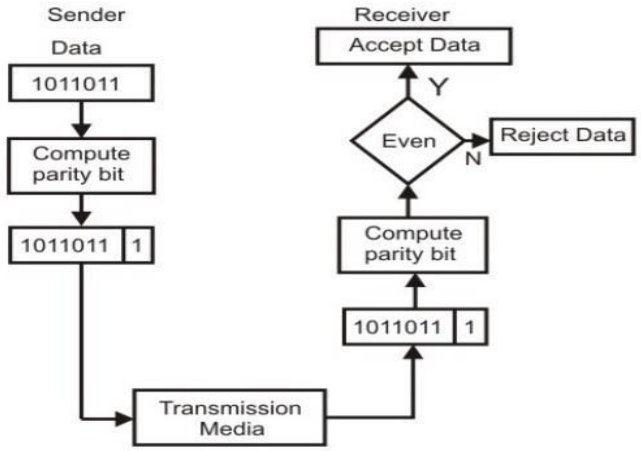


Fig 1) Even parity checking scheme

Decimal value	Data Block	Parity bit	Code word
0	0000	0	00000
1	0001	1	00011
2	0010	1	00101
3	0011	0	00110
4	0100	1	01001
5	0101	0	01010
6	0110	0	01100
7	0111	1	01111
8	1000	1	10001
9	1001	0	10010
10	1010	0	10100
11	1011	1	10111
12	1100	0	11000
13	1101	1	11011
14	1110	1	11101
15	1111	0	11110

Table 1:Possible 4 bit data words and corresponding code words

Two-dimension Parity Check

Performance can be improved by using two-dimensional parity check, which organizes the block of bits in the form of a table. Parity check bits are calculated for each row, which is equivalent to a simple parity check bit. Parity check bits are also calculated for all columns then both are sent along with the data. At the receiving end these are compared with the parity bitcalculated on the received data. This is illustrated in Fig. 2. Performance Two- Dimension Parity Checking increases the likelihood of detecting burst errors. As we have shown in Fig. 2, that a 2-D Parity check of n bits can detect a burst error of n bits. A burst error of more than n bits is also detected by 2-D Parity check with a highprobability. There is, however, one pattern of error that remains elusive. If two bits in one data unit are damaged and two bits in exactly same position in another data unit are also damaged, the 2-D Parity check checker will not detect an error. For example, if two data units: 11001100 and 10101100. If first and second from last bits in each of them is changed, making the data units as 01001110 and 00101110, the error cannot be detected by 2-D Parity check.

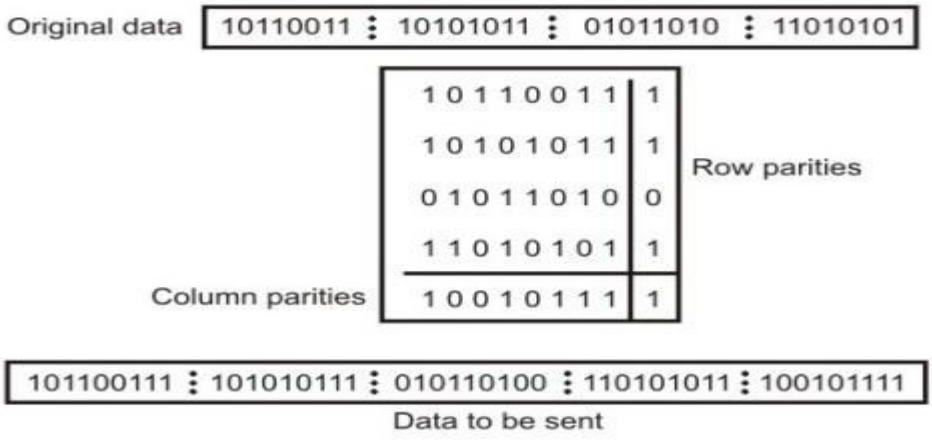


Fig 2) Two dimension parity checking

Example of Hamming Code Generation

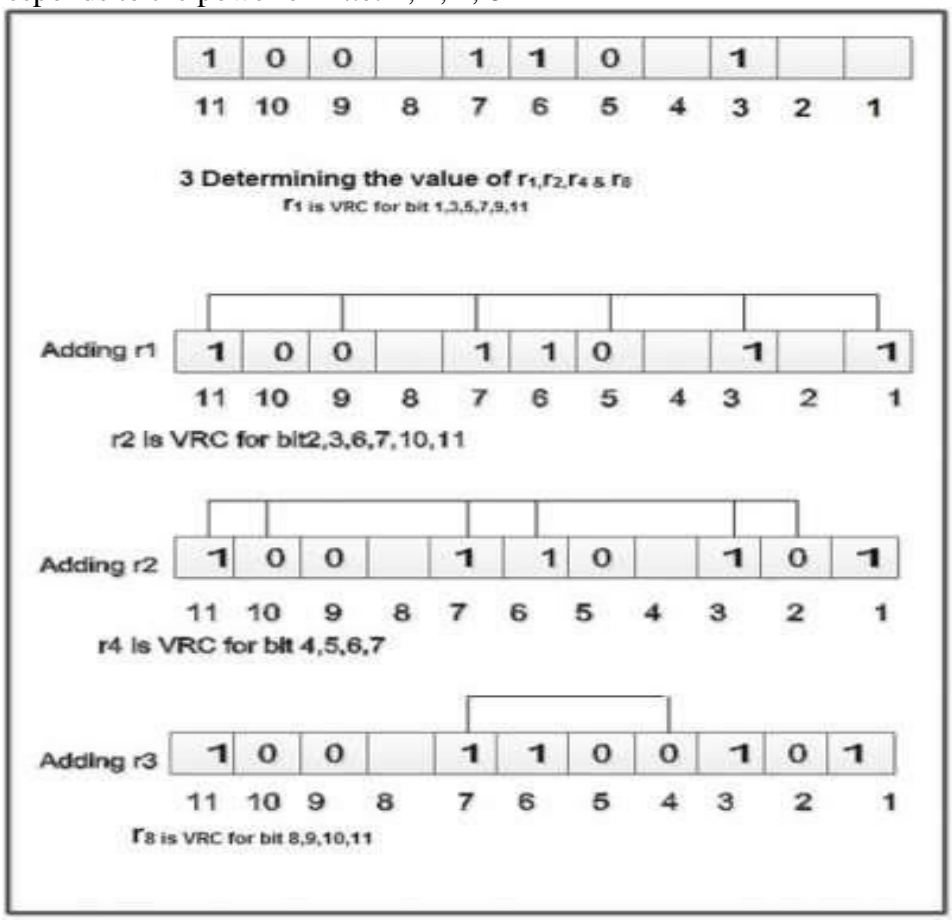
Suppose a binary data 1001101 is to be transmitted. To implement hamming code for this, following steps are used:

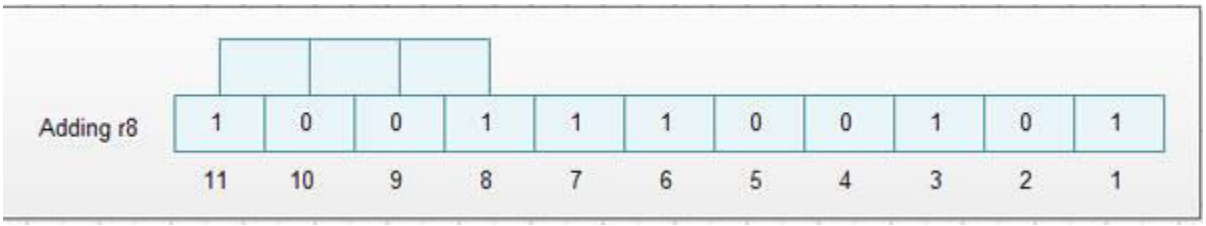
1. Calculating the number of redundancy bits required. Since number of data bits is 7, the value of r is calculated as

$$2^r \geq m + r + 1$$
$$2^4 \geq 7 + 4 + 1$$

Therefore no. of redundancy bits = 4

2. Determining the positions of various data bits and redundancy bits. The various r bits are placed at the position that corresponds to the power of 2 *i.e.* 1, 2, 4, 8

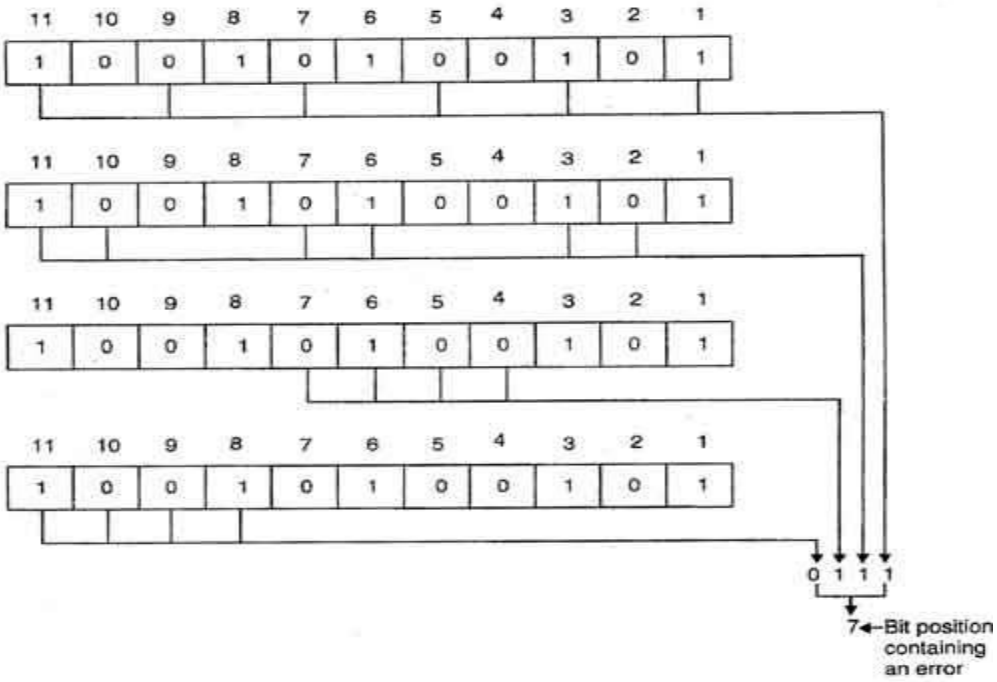




4. Thus data 1 0 0 1 1 1 0 0 1 0 1 with be transmitted.

13.1 Error Detection & Correction

Considering a case of above discussed example, if bit number 7 has been changed from 1 to 0.The data will be erroneous.



Data sent: 1 0 0 1 1 1 0 0 1 0 1

Data received: 1 00 1 0 1 00 1 0 1 (seventh bit changed)

The receive takes the transmission and recalculates four new VRCs using the same set of bits used by sender plus the relevant parity (r) bit for each set as shown in fig.

Then it assembles the new parity values into a binary number in order of r position (r_8, r_4, r_2, r_1).

In this example, this step gives us the binary number 0111. This corresponds to decimal 7. Therefore bit number 7 contains an error. To correct this error, bit 7 is reversed from 0 to 1.

References :

1. Moris Mano, “Digital Computer Fundamentals” TMH 3rd Edition
2. http://www.tutorialspoint.com/computer_logical_organization/number_system_conversion.htm
3. http://www.electronics-tutorials.ws/binary/signed-binary-numbers.html
4. HAMMING, R. W. “Error Detecting and Error Correcting Codes.” Bell System Tech. Jour., 29 (1950): 147–160.
5. A.P GODSE,D.A.GODSE .”Digital Systems”.Technical Publications.Pune.
6. http://www.tutorialspoint.com/computer_logical_organization/binary_codes.htm
7. http://nptel.ac.in/courses/Webcourse-contents/IIScBANG/Digital%20Systems/Digital%20Systems.pdf
8. Digital Logic Circuits by D.A.Godse A.P.Godse