fraction part

0.9375 × 8 = 7.5000 The integral part of the product is 7

0.5000 × 8 = 4.0000 The integral part of the product is 4

$\therefore (11.9375)_{10} = (13.74)_8$ Ans.

1.9 DOUBLE PRECISION NUMBERS

In a 16-bit computer, numbers from +32767 to -32768 can be expressed in each register. To store numbers longer than these, a provision has to be made. Double precision provides, the answer to this problem. In this system, two storage locations are used to represent each number. The format used is expressed as:

First word	S	High order Bits
Second word	O	Low order Bits

S is the sign bit and O represents zero. This permits a 31-bit number length using 16 bit registers.

Signed Binary Number System: In this form a positive or negative number is represented by a sign bit followed by its magnitude in binary form. Thus a decimal number +17 is represented by

$$\underbrace{\boxed{0}}_{\text{Sign bit}}\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{1}$$

and −14 is represented by

$$\underbrace{\boxed{1}}_{\text{Sign bit}}\boxed{0}\boxed{1}\boxed{1}\boxed{1}\boxed{0}$$

Normally 0 is used in the sign bit for positive numbers and 1 is used in the sign bit for negative numbers. Numbers written in this form are called signed binary numbers or signed magnitude numbers.

EXAMPLE 20: Represent the following decimal numbers in the signed binary number system.

(a) +29 (b) −29 (c) −19

Solution:

(a) +29 = 0 11101

(b) −29 = 1 11101

(c) −19 = 1 10011

1.10 FLOATING POINT REPRESENTATION

Consider th decimal number 128.456, which may be written as:

(i) 128.456

(ii) $.128456 \times 10^3$

If a register is availabe capable of storing 6 digits and a sign bit and this register is split into two parts — one part containing the integral portion of the number and other part containing the fractional portion and the decimal point located between the two parts of the register Fig. 1.1 shows this representation. The first drawback of this scheme is the need of the user to remember and keep track of the decimal point location. The second drawback is that the range of numbers which can represented using this scheme is limited to +999.999.

In the second method, called floating point representation, the number is written as a fractional multiplied by a power of 10. The fraction part is known as mantissa and the power of 10, which multiplies the fraction is known as exponent.

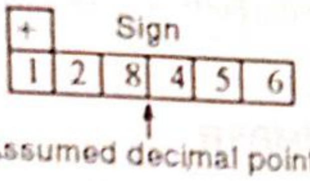


Fig. 1.1

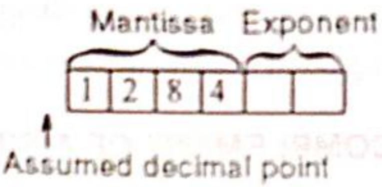


Fig. 1.2

The register is divided into two parts — the first part of 4 digits to contain the mantissa and the second part of 2 digits to hold the exponent. To store both positive and negative exponents, it is desired to split the range of 00 to 99 into two parts. Assuming 0 or origin at 50, all exponents greater than 50 are considered to be positive and all exponents less than 50 negative. The scheme is thus known as Floating point representation with exponent in excess 50 form. The range of exponent will be from — 50 to +49. In this scheme the number $.1284 \times 10^{3}$ can be stored in a register as in Fig. 1.3.

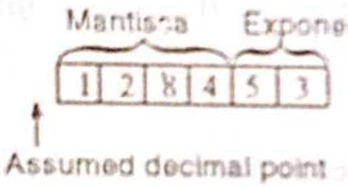


Fig. 1.3

To store the number — $.0001284 \times 10^{-6}$, the number is first written as -0.1284×10^{-9}, thus keeping all the significant digits in the mantissa, the process being called normalization. This number can then be stored in the register as in Fig. 1.4.

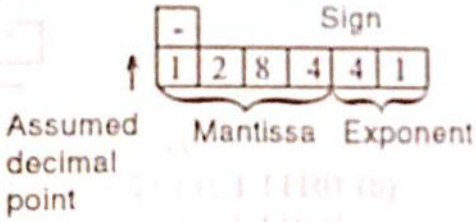
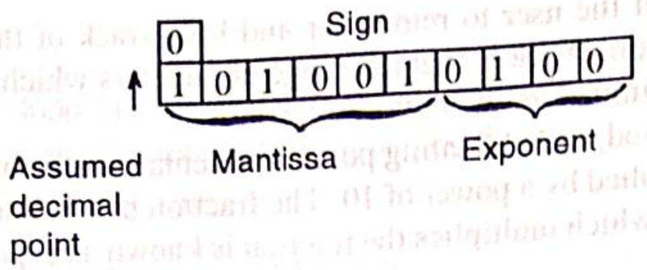


Fig. 1.4

A floating point number is said to be in the normalized form if the most significant position of the mantissa contains a non-zero digit.

A floating point binary number is also represented in a similar way except that here the base or radix is 2. For instance the number +1010.01 can be represented as a floating point binary number as.



Most computers and all electronic calculators have the built in capability of performing the floating point arithmetic operations.

1.11 1'S COMPLEMENT OF A BINARY NUMBER

The 1's complement of a binary number is obtained by changing each '0' to '1' and each '1' to '0'. Thus 1's complement of 010111 is 101000 and 1's complement of 1001.1110 is 0110.0001

2's complement of a Binary Number: The 2's complement of a binary number is obtained by adding 1 to the least significant bit of the 1's complement of that binary number. Thus 2's complement of 001=110 + 1=111. 2's complement of 010.111=101.000 + 1=101.001 2's complement of 110.11=001.00 + 1=001.01.

EXAMPLE 21: Perform the following subtraction using 1's complement arithmetic

(a) 11001 – 10110 (b) 1011 – 0101 (c) 11011 – 11001
(d) 10111.1 – 10011.1 (e) 11011.00 – 10011.11

Solution:
In the subtraction using 1's complement method, the 1's complement of the number to be subtracted (subtra hend) is formed by changing each 1 to 0 and each 0 to 1 and then performing the binary addition. Any end-around carry is then added to the least significant bit.

(a) $\begin{array}{r}11001\\-10110\end{array} = \begin{array}{r}11001\\+01001\\\hline 100010\\ \rightarrow 1\\\hline 00011\end{array}$ Ans.

(b) $\begin{array}{r}1011\\-0101\end{array} = \begin{array}{r}1011\\+\ 1010\\\hline 10101\\ \rightarrow 1\\\hline 0110\end{array}$ Ans.

(c) $\begin{array}{r}11011\\-11001\end{array} \quad \begin{array}{r}11011\\\times\ 00110\\\hline 100001\\ \rightarrow 1\\\hline 10\end{array}$ Ans.

(d) $\begin{array}{r}10111.1\\-10011.1\end{array} = \begin{array}{r}10111.1\\+01100.0\\\hline 100011.1\\ \rightarrow 1\\\hline 100.0\end{array}$ Ans.

(e) $\begin{array}{r}11011.00\\-10011.11\end{array} = \begin{array}{r}11011.00\\+\ 01100.00\\\hline 100111.00\\ \rightarrow 1\\\hline 111.01\end{array}$

EXAMPLE 22: Perform the following subtraction using 2's complementary arithmetic

(a) 11011 — 11001 (b) 11011.00 — 01100.00 (c) 0.01111 — 0.01001
(d) 111.01 — 010.111 (e) 111.01 — 110.11 (f) 10111.1— 10011.1

Solution:

In the subtraction using 2's complement method, first the 2's complement of the number to be subtracted (subtrahend) is formed by adding 1 to the least significant bit of 1's complement of subtrahend. Then binary addition of this 2's complement is performed with the given first number (minuend) and dropping or neglecting any end arround carry thus obtained i.e. dropping the final carry.

(a) 2's comlement of subtrahend = 11001 = 1's complement of 11001 + 1 = 00110 + 1 = 00111

$$\therefore \begin{array}{r} 11011 \\ -\,11001 \end{array} \quad = \begin{array}{r} 11011 \\ +\,00111 \\ \hline 100010 \end{array}$$

Carry is dropped

Thus the answer is 00010 = 10.

(b) 2's complement of subtrahend 01100.00
= 1's complement of 01100.00 + 1 = 10011.11 + 1 = 10100.00

$$\therefore \begin{array}{r} 11011.00 \\ -\,01100.00 \end{array} \quad = \begin{array}{r} 11011.00 \\ +\,10100.00 \\ \hline 101111.00 \end{array}$$

Carry is dropped

Thus the answer is 1111.00.

(c) 2's complement of subtrahend 0.01001 = .10110 + 1 = .10111

$$\therefore \begin{array}{r} .01111 \\ -\,.01001 \end{array} \quad = \begin{array}{r} .01111 \\ +\,.10111 \\ \hline 1.00110 \end{array}$$

Carry is dropped

Thus the answer is 0.00110.

(d) 2's complement of subtrahend 010.111
= 1's complement of 010.111 + 1 = 101.000 + 1 = 101.001

$$\therefore \begin{array}{r} 111.01 \\ -\,010.111 \end{array} \quad = \begin{array}{r} 111.010 \\ +\,101.001 \\ \hline 1100.011 \end{array}$$

Carry is dropped

Thus the answer is 100.011.

(e) 2's complement of subtrahend 110.11
= 1's complement of 110.11 + 1 = 001.00 + 1 = 001.01

$$\therefore \begin{array}{r} 111.01 \\ 110.11 \end{array} \quad = \begin{array}{r} 111.01 \\ +\,001.01 \\ \hline 1000.10 \end{array}$$

Carry is dropped

Thus the answer is 000.10.

(f) 2's complement of subtrahend 10011.1
= 1's complement of 10011.1 + = 01100.0 + 1 = 01100.1

```
∴  10111.1          10111.1
  −10011.1     =  + 01100.1
                  ────────
                  100100.0
                  └──→ Carry is dropped
```

Thus the answer is 100.0

~~1.12~~ BINARY CODED DECIMAL NUMBERS (BCD)

A code is certain special group of symbols used to represent numbers, letters, etc. In the BCD code each decimal digit of the number is represented by its binary equivalent as a nibble i.e., as a string of 4 bits each. Decimal numbers 5429 and 9637 are expressed in BCD numbers as follows:

5	4	2	9	Decimal
0101	0100	0010	1001	BCD
9	6	3	7	Decimal
1001	0110	0011	0111	

The BCD code is not a number system, but it is decimal system with each digit encoded, in its binary equivalent. In the binary number system, the whole of the decimal number is represented in binary, but in BCD code, each decimal digit is individually converted to its binary equivalent as a nibble. The advantage of the BCD code is an easy mode of conversion from decimal to binary and binary to decimal.

The main area of applications of BCD numbers is where decimal data is transferred into or out of digital processes. BCD numbers are processed by circuits of calculators, since decimal numbers are entered through the keyboard and decimal answers are seen on the LED or Liquid Crystal Display (LCD). Other devices, whose circuits can process BCD data or numbers are digital clocks, digital voltmeters, etc. Reverse conversion from BCD numbers to decimal numbers is also easy and is explained in the following example.

EXAMPLE 23: Convert the following BCD numbers to their decimal equivalent.
(a) 0100 0011 1001 (b) 0100 0010 0111 1000

Solution:
(a) Divide the given BCD number into groups of 4 bit each and then convert each nibble to decimal equivalent

$$\underbrace{0100}_{4}\quad\underbrace{0011}_{3}\quad\underbrace{1001}_{9}\ \text{Ans.}$$

(b) $$\underbrace{0100}_{4}\quad\underbrace{0010}_{2}\quad\underbrace{0111}_{7}\quad\underbrace{1000}_{8}\ \text{Ans.}$$

P ~~1.13~~ EXCESS-3 CODE

In the excess –3 code, 3 is added to each decimal digit and then each of the resulting digit is converted to an equivalent binary number written as a nibble, as is done in BCD code. For example decimal numbers 9 and 14 are written in excess –3 code as

$$\begin{array}{r} 9 \\ +\,3 \\ \hline 12 \end{array} = 1100 \qquad\qquad \begin{array}{r} 14 \\ +\,33 \\ \hline 47 \end{array} = 0100\ 0111$$

Excess –3 code in short is abbreviated as XS3.

P ~~1.14~~ GRAY CODE

In Gray Code also called cyclic or reflected code, only one bit in the code changes, when going from one step to the next, which means that only one bit will change each time the decimal number is incremented unlike in the binary system which requires one or more bits to change, each time the decimal is incremented. For instance when going from 7 to 8, the binary code changes from 0111 to 1000, which requires that all four bits change simultaneously.

To convert a binary number to its equivalent Gray Code, write the MSB as it is. Add this MSB to next binary position, write the sum and ignore any carry. Continue and write successive sums until process is completed.

EXAMPLE 24: Convert the following binary numbers to Gray Code.
(a) 110100 (b) 101101 (c) 1110010

Solution:

(a)

$$\begin{tikzcd}
\text{(MSB)} & & & & & & \text{(LSB)} \\
1 \arrow[r, bend left] \arrow[d] & 1 \arrow[r, bend left] \arrow[d] & 0 \arrow[r, bend left] \arrow[d] & 1 \arrow[r, bend left] \arrow[d] & 0 \arrow[r, bend left] \arrow[d] & 0 \arrow[d] & \text{Binary number} \\
1 & 0 & 1 & 1 & 1 & 0 & \text{Gray number Ans.}
\end{tikzcd}$$

Write MSB = 1 as it is. Add this 1 to next significant bit of binary to get 10, write 0 below and ignore carry of 1. Add 1 and next 0 to get 1. Again add 0 to 1 to get, 1 and 0 to get and 0 and 0 to get 0 in Gray Code

(b)

$$\begin{tikzcd}
\text{(MSB)} & & & & & & \text{(LSB)} \\
1 \arrow[r, bend left] \arrow[d] & 0 \arrow[r, bend left] \arrow[d] & 1 \arrow[r, bend left] \arrow[d] & 1 \arrow[r, bend left] \arrow[d] & 0 \arrow[r, bend left] \arrow[d] & 1 \arrow[d] & \text{Binary number} \\
1 & 1 & 1 & 0 & 1 & 1 & \text{Gray number Ans.}
\end{tikzcd}$$

(c)

$$\begin{tikzcd}
\text{(MSB)} & & & & & & & \text{(LSB)} \\
1 \arrow[r, bend left] \arrow[d] & 1 \arrow[r, bend left] \arrow[d] & 1 \arrow[r, bend left] \arrow[d] & 0 \arrow[r, bend left] \arrow[d] & 0 \arrow[r, bend left] \arrow[d] & 1 \arrow[r, bend left] \arrow[d] & 0 \arrow[d] & \text{Binary number} \\
1 & 0 & 0 & 1 & 0 & 1 & 1 & \text{Gray number Ans.}
\end{tikzcd}$$

To convert Gray number to equivalent binary, write the MSB as it is, add this binary MSB to next significant bit of Gray Code, write the result ignoring any carry.

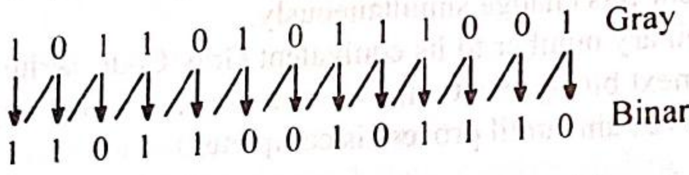
Continue the process till the LSB is reached.

EXAMPLE 25: (a) Subtract using 1's complements $(1011101)_2 - (1101100)_2$ [AMIE Summer 1993]
(b) A number written in Gray Code is 1011010111001. Write the equivalent binary code number [AMIE Summer 1993]
(c) Convert $(63547)_{10}$ to BCD [AMIE Winter 1993]
(d) Subtract 32 from 85 using 1's complement binary arithmetic [AMIE Winter 1993]

Solution:
(a) 1011101 — 1101100
1's complement of number to be subtracted (subtrahend) 1101100 is obtained as 0010011 and then adding.

```
∴   1011101          1011101
  - 1101100    =   + 0010011
                   ---------
                  ┌ 1110000
                  └───────→1
                   ---------
                    110001  Ans.
                   ---------
```

(b) (MSB)



Write MSB = 1 as it is. Add this 1 to next significant (0) of Gray Code number to get 1. Again add this 1 to next significant bit (1) of Gray Code to get 10, write 0 and ignore carry. Now add this 0 to next significant bit (1) of Gray Code to get 1. Add this 1 to next significant bit (1) of Gray Code to get 1 and so on to get equivalent binary code number 1101100101110 Ans.

(c) 6 3 5 4 7
Now representing each of the decimal digit of above number by its binary equivalent written as a nibble each.

6	3	5	4	7	Decimal
0110	0011	0101	0100	0111	BCD Ans.

(d) $(85)_{10} = 1010101$
$(32)_{10} = 0100000$

1's complement of $(32)_{10} = 1011111$

```
∴   1010101        1010101
  - 0100000    =  + 1011111
                  ---------
                 ┌ 10110100
                 └───────→ 1
                  ---------
                   0110101  Ans.
                  ---------
```

EXAMPLE 26: Write the equivalent in decimal for the following excess—3 code number:
[AMIE Summer 1994]

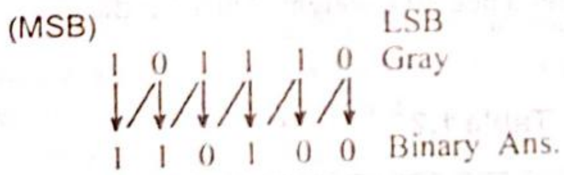
1000 0011 0100 0101 0111

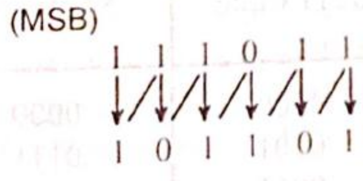
Solution:
Excess 3 code = (Decimal number + 3) written in binary. Therefore the equivalent decimal number of above excess 3 code numbers can be obtained by first writing the decimal number and then subtracting decimal 3 from each, we get

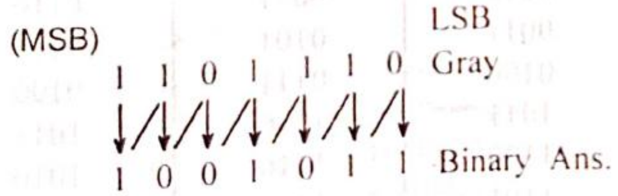
$$(8-3) \quad (3-3) \quad (4-3) \quad (5-3) \quad (7-3) \text{ Ans.}$$
$$= \quad 5 \quad 0 \quad 1 \quad 2 \quad 4$$

EXAMPLE 27: Convert the following Gray Code numbers into equivalent binary numbers
(a) 101110
(b) 111011
(c) 1101110

Solution:
(a) Write MSB = 1 as it is. Add this 1 to next significant bit (0) of gray to get 1. Again add this 1 to next significant bit (1) of gray to get 10, write 0 and ignore carry (1). Add this 0 to next significant bit (1) of gray to get 1. Add this 1 to next significant bit (1) of gray to get 10, write 0 and ignore carry (1). Add this 0 to next significant bit (0) of gray to get LSB = 0.

(a)


(b)


(c)


While converting gray (g) to binary (b), remember that if number of 1's preceding g_i is even, write $b_i = g_i$, if odd write $b_i = g_i$ i.e b_i = complement of g_i b_i is binary integer and gi is gray integer.

1.15 ALPHANUMERIC CODES

Apart from the numeral data, a computer should also be able to handle and recognize the codes which express letters of alphabet, certain special characters and punctuation marks. Such codes are called alphanumeric codes.

ASCII Code: ASCII code pronounced as Askee is the most popular and widely used alphanumeric code. Its full form is American Standard Code for Information Interchange. It is a 7-bit code whose arrangement is $X_6 X_5 X_4 X_3 X_2 X_1 X_0$, where each X is either 0 or 1. Letter A is coded as 1000001, and B as 1000010 and so on. It has $2^7 = 128$ possible code groups and as such can represent all the standard keyboard characters. This code permits the manufacturers to standardize I/O hardware such as keyboards, video display, printers, etc.

EBCDIC Code: Another code quite often used in modern, large digital computers is EBCDIC code pronounced as Ebsidik. It stands for Extended Binary Coded Decimal Interchange Code, which is an 8-bit code and uses binary coded decimal as the basis of binary arrangement. It is used for business oriented machines.

Hollerith Code: Hollerith code is used for punched card data. Each card has 12 horizontal rows, numbered 0 to 9,11,12 and 80 vertical columns numbered 1 to 80. The punched cards are used to input information or data to the computer.

15 P

1.16 WEIGHTED CODES

In weighted codes, each position of the number has a specific weight. The decimal value of a weighted code number is the algebraic sum of the weights of those positions in which a '1' appears. Table 1.2 shows a few weighted binary codes such as 8421 code, 2421 code, 5211 code and a weighted code having negative weights like $84\bar{2}\bar{1}$. The bar over a number indicates a negative weight. Thus $\bar{2}$ indicates —2 and 1 indicates —1 weight.

Table 1.2

Decimal Number	8421 Code	2421 Code	5211 Code	$84\bar{2}\bar{1}$ Code
0	0000	0000	0000	0000
1	0001	0001	0001	0111
2	0010	0010	0011	0110
3	0011	0011	0101	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1010	1010
7	0111	1101	1100	1001
8	1000	1110	1110	1000
9	1001	1111	1111	1111

Each decimal digit in the weighted codes is represented by a four bit group. For example a decimal number 493 is represented in the above weighted codes as follows:

In 8421 code : 0100 1001 0011
In 2421 code : 0100 1111 0011
In 5211 code : 0111 1111 0101
In $84\bar{2}\bar{1}$ code : 0100 1111 0101.

Reflective Codes: A code is called reflective or self-complementing if the code for 9 is the complement for the code for 0, code for 8 is the complement for the code for 1 and so on. The 2421, 5211, 8421 codes are reflective, whereas 8421 (BCD) code is not reflective.

Sequential Codes: A code is called sequential if each successive code is one binary number greater than its preceding code. The 8421 and × S3 codes are sequential, whereas 2421, 5211, 8421 codes are not sequential.

1.17 PARITY METHOD FOR ERROR DETECTION AND CORRECTION

In digital circuits and systems, a group of bits which is stored, operated and transmitted is called a word. When words or information is transmitted into or out of memory, there is every possibility that errors are introduced in the word, so that the receiver is not able to receive the identical or the correct information sent by the transmitter. Due to electrical noise or voltage or current transients, any '1' which occurs in the original word may be changed to '0'. This error which occurs in the system may cause a serious problem, because it will have a significant effect on the results. To avoid this disastrous effect of error, most digital systems use the most widely prevalent method for error detection and correction, called parity method.

One method of introducing error detecting ability into the code is to introduce an additional bit, called parity bit, so that the total number of 1's in the resulting code are either odd or even. If an additional bit is added to make the total number of 1's an odd number, then it is called an odd parity, whereas if the total number of 1's in the resulting code are even, then it is said to have an even parity. The additional parity is normally placed on to the left of the most significant bit. The following resulting code has odd parity.

Parity Bit	*Data*	*Even parity total number* of 1's
1	1010101	5
0	1101011	5
1	0100100	3
0	1010010	3
1	0000000	1

The resulting code has one parity bit and seven data bits, thus forming an eight-bit word, which is transmitted to the receiver. At the receiving, it is checked that each eight bit word has an odd number of 1's . For example if the first word given above i.e., 11010101 is received as 11010111 due to error because the equipment failure or noise in the transmission channel, on checking the word it will be found that at receiving location, it has even number of 1's, which is invalid because the resulting code has odd parity, thus detecting an error. But the error is not diagnosed—i.e., we do not know where the error has occured. This method is very good for detecting single bit error, if they occur. But if two errors occur in the word, they remain undetected. The method is quite widely used as even single bit errors are quite rare. Even parity can also be used. Here the parity bit is chosen so that the total number of 1's in the resulting code is even. The following resulting code has even parity.

Parity Bit	*Data*	*Even parity total number of 1's*
1	1101011	6
0	1011111	6
1	1001001	4
0	1001000	2

The word may have four data bits. It is only necessary to check the odd or even parity at the receiving location.

Hamming Code : R.W. Hamming developed a system in which he showed that by introducing more parity bits in the code it is not only possible to detect and locate the error but also to correct. Assuming four data bits to be transmitted, the word will be arranged as:

1	2	3	4	5	6	7
P_1	P_2	D_1	P_3	D_2	D_3	D_4

where D_1, D_2, D_3, D_4 are the four data bits and P_1, P_2, P_3 are parity bits. P_1 is selected so as to establish an even parity in positions 1,3,5,7. P_2 is selected so as to establish an even parity in positions 2,3,6,7 and P_3 is chosen to establish even parity in positions 4.5.6.7.

EXAMPLE 26: Construct an even parity seven bit Hamming code to transmit the data
(a) 0100 (b) 1110

Solution:

(a) Data bits 0100 can be filled up in the Hamming code format to give the word.

1	2	3	4	5	6	7
P_1	P_2	0	P_3	1	0	0

Now P_1 must be '1' in order to establish even parity at positions 1,3,5,7. P_2 must be '0' to establish even parity at positions 2,3,6,7 and P_3 must be '1' to establish even parity at positions 4,5,6,7. Thus the Hamming code is 1001100. Ans.

(b) Data bits 1110 can be filled up in the Hamming code format to give the word.

1	2	3	4	5	6	7
P_1	P_2	1	P_3	1	1	0

Now P_1 must be '0' to establish an even parity at positions 1,3,5,7. P_2 must be '0' to establish an even parity at positions 2,3,6,7 and P_3 must be '0' to establish an even parity at positions 4,5,6,7. Thus the seven bit Hamming code is 0010110. Ans.

EXAMPLE 27: A seven bit Hamming code is received as 1111001. Locate the error position and find the correct code.

Solution:

1	2	3	4	5	6	7
1	1	1	1	0	0	1

Check the even parity at positions 4,5,6,7, it passes giving even parity. Thus $C_1 = 0$. Check the even parity at positions 2,3,6,7, it fails due to odd number of 1's. Thus $C_2 = 1$. Check the even parity and positions 1,3,5,7 it fails again due to odd number of 1's. Thus $C_3 = 1$. Therefore position of error is $C_1 C_2 C_3 = 011$ which in decimal form is equal to 3. Thus error is located at third position. Therefore the bit at the third position should be complemented i.e., '1' at third position should be corrected to a '0'. Thus correct Hamming code is 1101001. Ans.

EXERCISES

1. Convert the following decimal numbers to their binary equivalent.
   (a) 25 (b) 21.6 [Ans. (a)10101, (b) 10101.10011]

2. Convert the following binary numbers to their equivalent decimal numbers.
   (a) 1111101 (b) 11.011 [Ans. (a) 125, (b) 3.375]

3. Convert decimal 63718 into equivalent (a) binary (b) octal (c) hexadecimal numbers. [Ans. (a) 1111100011100110(b) 174346, (c) F8E6]

4. Perform the following binary multiplications.$(11.110)_2 \times (100.1)_2$
   [Ans.10000.1110]

5. Convert $(AA1)_{16}$ to decimal and octal numbers. [Ans. $(2721)_{10}$, $(5241)_8$]

6. Convert the following octal numbers to equivalent decimal numbers.
   (a) $(567)_8$ (b) $(665)_8$ [Ans. (a) $(375)_{10}$, (b) $(437)_{10}$]

7. Perform the following subtraction using 1's complement arithmetic.
   (a) 0.1001 — 0.0110 (b) 0.01111 — 0.01001 [Ans.(a) 0.0011, (b) 0.00110]

8. Perform the following subtraction using 2's complement arithmetic.
   (a) 11011 — 10100 (b) 11100 — 00100 [Ans. (a) 00111, (b) 11000]