Example 4.9.

A computer uses ASCII for its internal representation of characters. In which order will this computer sort the strings 23, A1, 1A, a2, 2a, aA, and Aa?

Solution:

In ASCII, alphabetic characters have values greater than numeric characters and lowercase letter have values greater than uppercase letters (refer to Figure 4.6). Hence, in the said computer, numeric characters will be placed before uppercase letters and lowercase letters will be placed after uppercase letters causing the given strings to be sorted as:

1A < 23 < 2a < A1 < Aa < a2 < aA

Therefore, the sorted sequence will be: 1A, 23, 2a, A1, Aa, a2, and aA

Points to Remember

1. Computers use binary coding schemes to represent data internally. In binary coding, a group of bits represents every symbol that appears in the data. The group of bits used to represent a symbol is called a *byte*. Most modern computers use 8 bits to represent a symbol. Hence, the term *byte* commonly means a group of 8 bits.
2. BCD, EBCDIC, ASCII, and Unicode are the commonly used computer codes for internal data representation.
3. *BCD* (*Binary Coded Decimal*) is a 6-bit code that can represent 64 different characters.
4. *EBCDIC* (*Extended Binary-Coded Decimal Interchange Code*) is an 8-bit code that can represent 256 different characters.
5. *ASCII* (*American Standard Code for Information Interchange*) is one of the most popular and widely supported computer codes. It is of two types – ASCII-7 and ASCII-8. ASCII-7 is a 7-bit code that can represent 128 different characters and ASCII-8 is an 8-bit code that can represent 256 different characters.
6. Unicode provides a consistent way of encoding multilingual plain text. It has the capacity to encode as many as a million characters. Unicode has three types of encoding forms – UTF-8, UTF-16, and UTF-32.
7. Collating sequence is ordering of symbols in an encoding standard.

Questions

1. Define the term 'byte'. What is the difference between a bit and a byte?
2. Write 4-bit BCD code for following numbers:
   (a) 25_{10} (c) 128_{10}
   (b) 64_{10} (d) 1024_{10}
3. Write BCD code for following words using binary notation:
   (a) BIT (c) CODE
   (b) BYTE (d) ZERO
4. Write BCD code for following words using octal notation:
   (c) COMPUTER (c) VIDEO
   (d) INPUT (d) OUTPUT
5. Why BCD code was extended to EBCDIC?
6. How many different characters are possible in the following encoding schemes:
   (e) BCD (c) ASCII-7
   (f) EBCDIC (d) ASCII-8

7. Why are octal and hexadecimal shortcut notations used? Identify shortcut notations used for each of the following computer codes:
   (g) BCD (c) ASCII-7
   (h) EBCDIC (d) ASCII-8

8. Why do we have a packed decimal format? How does it differ from a zoned decimal format?

9. A newly designed computer code uses 9 bits. How many different characters are possible in this code?

10. Write EBCDIC code for following words in both binary and hexadecimal notations:
   (i) SUN (c) CAT
   (j) MOON (d) DOG
   How many bytes are required to store each of these words using this coding?

11. Write EBCDIC code for following words in hexadecimal notation:
   (k) PROGRAM (c) BYTE
   (l) OUTPUT (d) OCTAL
   How many bytes are required to store each of these words using this coding?

12. Write EBCDIC code for following words in binary notation:
   (m) ZONE (c) SKY
   (n) BOY (d) TAPE
   How many bytes are required to store each of these words using this coding?

13. Using hexadecimal notation, write the zoned-decimal coding for following numbers:
   (o) 1256 (c) -63
   (p) +439 (d) -786
   How many bytes are required for each of these representations?

14. Using hexadecimal notation, write the packed-decimal coding for following numbers:
   (q) 12915 (c) 872
   (r) +9876 (d) -256
   How many bytes are required for each of these representations?

15. List the similarities and differences between 7-bit and 8-bit ASCII.

16. Write ASCII-7 and ASCII-8 codes for follow words using binary and hexadecimal notations:
   (s) DRY (c) DAMP
   (t) WET (d) TERM
   How many bytes are required to store each of thes words in ASCII?

17. Write ASCII-7 and ASCII-8 codes for followin words using binary notation:
   (u) PRINT (c) RUB
   (v) TYPE (d) GIVE
   How many bytes are required to store each of the words in ASCII?

18. What is Unicode? What motivated its development

19. Why it was difficult to exchange text file internationally before Unicode was developed How did Unicode help in overcoming this problem?

20. Why was Unicode developed?

21. List some key features of Unicode.

22. Describe the three encoding forms defined by Unicode standard and the environment for which each of them is most suitable.

23. Explain the meaning of the term "collating sequence".

24. A computer uses EBCDIC as its internal representation of characters. In which order w this computer sort following strings?
   (w) ABC (c) 245
   (x) 123 (d) ADD

25. A computer uses ASCII. In which order will this computer sort following strings?
   (y) BED (d) 2good
   (z) 512 (e) BaD
   (aa) ADD (f) a1B2

26. Write full form of following abbreviations:
   (bb) BCD (c) ASCII
   (cc) EBCDIC (d) UTF

Chapter 5

Computer Arithmetic

In Chapter 4, we saw that computers store numbers, letters, and other characters in coded form in binary number system. In this chapter, you will learn why computers use binary numbers instead of decimal numbers and how basic arithmetic operations are performed inside a computer by using binary numbers.

WHY BINARY?

In Chapter 3, we saw that use of a smaller base often requires more positions to represent a given value (recall the reason for using octal and hexadecimal notations), for example, $9_{10} = 1001_2$. Here, four positions are required instead of one to represent decimal number 9 in binary form. In spite of this, almost all computers use binary numbers. Hence, the obvious question is 'Why are binary numbers used instead of decimal numbers?' The reasons are as follows:

1. Information is handled by computer's electronic/electrical components such as transistors, semiconductors, wires, etc., and all these can indicate only two states or conditions – ON (state 1) or OFF (state 0). Transistors are either conducting (state 1) or non-conducting (state 0), magnetic materials are either magnetized (state 1) in one or opposite direction or non-magnetized (state 0), a pulse or voltage is present (state 1) or absent (state 0) in wire. Information is represented within a computer by the presence or absence of these types of signals. Binary number system having only two digits (0 and 1) is most suitable for expressing the two possible states. Figure 5.1 illustrates the concept of binary components.
2. By using binary numbers, computer circuits only have to handle two binary digits rather than ten decimal digits. This greatly simplifies the internal circuit design of c[illegible]ers [illegible] in [illegible]

Binary State	ON (1)	OFF (0)
Bulb		
Switch		
Circuit Pulse		

Figure 5.1. Examples of devices that work in binary mode. These devices can only represent two states – ON (state 1) or OFF (state 0).

BINARY ARITHMETIC

In this section, you will see how the four basic arithmetic operations are performed inside a computer by using binary numbers. Actually, binary arithmetic is simpler to learn because binary number system deals with only two digits – 0 and 1. Since all binary numbers are made up of only 0s and 1s, when arithmetic operations are performed on these numbers the results are also in 0s and 1s.

Addition

Binary addition is performed in the same manner as decimal addition. However, since binary number system has only two digits, the addition table for binary arithmetic is very simple and consists of only four entries. The complete table for binary addition is as follows:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 plus a carry of 1 to next higher column

Carry-overs are performed in the same manner as in decimal arithmetic. Since 1 is the largest digit in binary number system, any sum greater than 1 requires a digit to be carried over. For instance, 10 plus 10 binary requires addition of two 1s in the second position. Since 1 + 1 = 0 plus a carry-over of 1, the sum of 10 + 10 is 100 in binary.

Example 5.1.

Add binary numbers 101 and 10 in both decimal and binary forms.

Solution:

Binary	Decimal
101	5
+10	+2
111	7

Example 5.2.

Add binary numbers 10011 and 1001 in both decimal and binary forms.

Solution:

	Binary		Decimal
carry	11	carry	1
	10011		19
	+1001		+9
	11100		28

In this example of binary addition, a carry is generated for the first and second columns.

Example 5.3.

Add binary numbers 100111 and 11011 in both decimal and binary forms.

Solution:

	Binary		Decimal
carry	11111	carry	1
	100111		39
	+11011		+27
	1000010		66

In this example, we face a new situation (1 + 1 + 1) brought about by the carry-over of 1 in the second column. This can also be handled using the same four rules for binary addition. The addition of three 1s can be broken up into two steps. First, we add only two 1s giving 10 (1 + 1 = 10). The third 1 is now added to this result to obtain 11 (a 1 sum with a 1 carry). Hence, 1 + 1 + 1 = 1, plus a carry of 1 to next higher column.

Subtraction

The principles of decimal subtraction can as well be applied to subtraction of numbers in other number systems. It consists of two steps that are repeated for each column of the numbers. The first step is to determine if it is necessary to borrow. If the subtrahend (the lower digit) is larger than the minuend (the upper digit), it is necessary to borrow from the column to the left. It is important to note here that the value borrowed depends upon the base of the number system and is always the decimal equivalent of the base. Hence, in decimal 10 is borrowed, in binary 2 is borrowed, in octal 8 is borrowed, and in hexadecimal 16 is borrowed. The second step is simply to subtract the lower value from the upper value. The [illegible]

Observe that the only case in which it is necessary to borrow is when 1 is subtracted from 0. The examples g below illustrate the exact procedure.

Example 5.4.

Subtract 01110_2 from 10101_2.

Solution:

```
Borrow  { 12
        { 0202
          10101
         -01110
         ------
          00111
         ------
```

In the first column (from right to left), 0 is subtracted from 1. No borrow is required in this case and the result is 1. In the second column, we have to subtract 1 from 0. A borrow is necessary to perform this subtraction. Hence, a 1 is borrowed from the third column that becomes 2 (binary 10) in the second column because the base is 2. The third column now becomes 0. Now in the second column, we subtract 1 from 2 giving 1. Since the third column is 0 due to earlier borrow, we have to subtract 1 from 0 for which borrow is required. The fourth column contains a 0 and hence, has nothing to borrow. Therefore, we have to borrow from the fifth column. Borrowing 1 from the fifth column gives 2 in the fourth column and the fifth column becomes 0. Now the fourth column has something to borrow. When 1 of the 2 in the fourth column is borrowed, it becomes 2 in the third column and 1 remains in the fourth column. Now in the third column, we subtract 1 from 2, giving 1. Subtraction of the fourth column is now 1 from 1, giving 0 and in the fifth column, subtraction is 0 from 0, giving 0. Hence, the result of subtraction is 00111_2. The result may be verified by subtracting 14_{10} (= 01110_2) from 21_{10} (=10101_2), which gives 7_{10} (= 00111_2).

Example 5.5.

Subtract 0111000_2 from 1011100_2.

Solution:

```
Borrow   2
        1011100
       -0111000
       --------
        0100100
       --------
```

Verify the result by subtracting 56_{10} (= 0111000_2) from 92_{10} (= 1011100_2), which gives 36_{10} (= 0100100_2).

Additive Method of Subtraction

Direct method of subtraction by using the borrow concept seems to be easiest when we perform subtraction wit paper and pencil. However, when subtraction is implemented by means of digital components, this method is found to be less efficient than additive method of subtraction. It may sound surprising that even subtraction performed by using an additive method. Additive method of subtraction is known as *complementary subtraction.*

Example 5.6.

Find the complement of 37_{10}.

Solution:

Since the number has 2 digits and the value of base is 10,
$(Base)^n - 1 = 10^2 - 1 = 99$
Now, $99 - 37 = 62$
Hence, the complement of $37_{10} = 62_{10}$.

Example 5.7.

Find the complement of 6_8.

Solution:

Since the number has 1 digit and the value of base is 8,
$(Base)^n - 1 = 8^1 - 1 = 7_{10} = 7_8$
Now, $7_8 - 6_8 = 1_8$
Hence, the complement of $6_8 = 1_8$.

Example 5.8.

Find the complement of 10101_2.

Solution:

Since the number has 5 digits and the value of base is 2,
$(Base)^n - 1 = 2^5 - 1 = 31_{10} = 11111_2$
Now, $11111_2 - 10101_2 = 01010_2$
Hence, the complement of $10101_2 = 01010_2$.

In case of binary numbers, it is not necessary to go through the usual process of obtaining complement. A quicker way to obtain the complement of a binary number is to transform all its 0s to 1s and all its 1s to 0s. For example, the complement of 1011010 is 0100101. Observe this in case of Example 5.8. Hence, the circuit for obtaining the complement of a number in binary number system can be easily designed at very less expense.

Let us now see how subtraction is performed by using complementary method. It involves following steps:

Step 1: Find the complement of the number you are subtracting (subtrahend)

Step 2: Add this to the number from which you are taking away (minuend)

Step 3: If there is a carry of 1, add it to obtain the result; if there is no carry, recomplement the sum and attach a negative sign to obtain the result.

To illustrate the procedure, let us first consider few examples for decimal numbers.

Example 5.9.

Subtract 56_{10} from 92_{10} using complementary method.

Solution:

Step 1: Complement of 56_{10}
$= 10^2 - 1 - 56$
$= 99 - 56$
$= 43_{10}$

(Continued on next page ...)

Example 5.10.

Subtract 35_{10} from 18_{10} using complementary method.

Solution:

Step 1: Complement of 35_{10}
$= 10^2 - 1 - 35$
$= 99 - 35$
$= 64_{10}$

(Continued on next page ...)

(... Continued from previous page)

Step 2: 92
+43 (complement of 56)
135

Step 3: ↳1 (add the carry of 1)

Result = 36

Verify the result by using the method of normal subtraction:

92 - 56 = 36.

(... Continued from previous page)

Step 2: 18
+64 (complement of 35)
82

Step 3: Since there is no carry, recomplement the sum and attach a negative sign to obtain the result.

Result = -(99 - 82)
= -17

Verify the result by using the method of normal subtraction:

18 - 35 = -17.

Let us re-work these examples by using binary numbers.

Example 5.11.

Subtract 0111000_2 (56_{10}) from 1011100_2 (92_{10}) using complementary method.

Solution:

1011100
+1000111 (complement of 0111000)
10100011

↳ 1 (add the carry of 1)

0100100

Result = $0100100_2 = 36_{10}$

Verify the result with the results of Example 5.5 and Example 5.9.

Example 5.12.

Subtract 100011_2 (35_{10}) from 010010_2 (18_{10}) using complementary method.

Solution:

010010
+011100 (complement of 100011)
101110

Since there is no carry, we have to complement the sum and attach a negative sign to it. Hence,

Result = -010001_2 (complement of 101110_2)
$= -17_{10}$

Verify the result with the result of Example 5.10.

Example 5.13.

Subtract 01110_2 from 10101_2 using complementary method.

Solution:

10101
+10001 (complement of 01110)
100110

Multiplication

Multiplication in binary number system also follows the same general rules as multiplication in decimal number system. However, learning binary multiplication is a trivial task because the table for binary multiplication is very short, with only four entries, instead of 100 entries necessary for decimal multiplication. The complete table for binary multiplication is as follows:

0 x 0 = 0
0 x 1 = 0
1 x 0 = 0
1 x 1 = 1

The example given below illustrates the method of binary multiplication. It is only necessary to copy the multiplicand, if the digit in the multiplier is 1 and to copy all 0s, if the digit in the multiplier is 0. The ease with which each step of the operation is performed is apparent.

Example 5.14.

Multiply binary numbers 1010 and 1001.

Solution:

1010	Multiplicand
x1001	Multiplier
1010	Partial Product
0000	Partial Product
0000	Partial Product
1010	Partial Product
1011010	Final Product

Note that a partial product is obtained by simply copying the multiplicand when multiplier digit is 1 and by copying all 0s when the multiplier digit is 0. As in decimal multiplication, each partial product is shifted one place to the left from the previous partial product. The final product is obtained by adding all the partial products according to binary addition rules.

In actual practice, whenever a 0 appears in the multiplier a separate partial product consisting of a string of 0s need not be generated. Instead, only a left shift will do. As a result, Example 5.14 may be reduced to

```
   1010
  x1001
  ------
   1010
1010SS  (S = left shift)
  -------
1011010
  -------
```

A computer would also follow this procedure in performing multiplication. Verify the result of this multiplication by multiplying 10_{10} (1010_2) by 9_{10} (1001_2), which produces a result of 90_{10} (1011010_2).

Example 5.15.

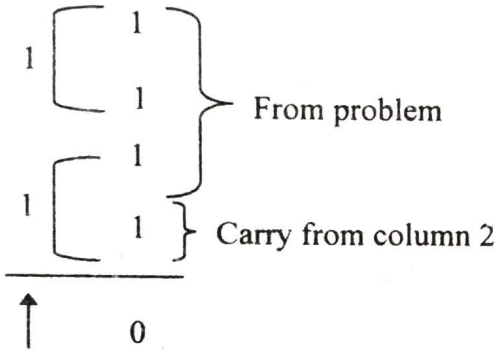
Multiply binary numbers 1111 and 111.

Solution:

			1	1	1	1
			x	1	1	1
			1	1	1	1
		1	1	1	1	
	1	1	1	1		
1	1	0	1	0	0	1

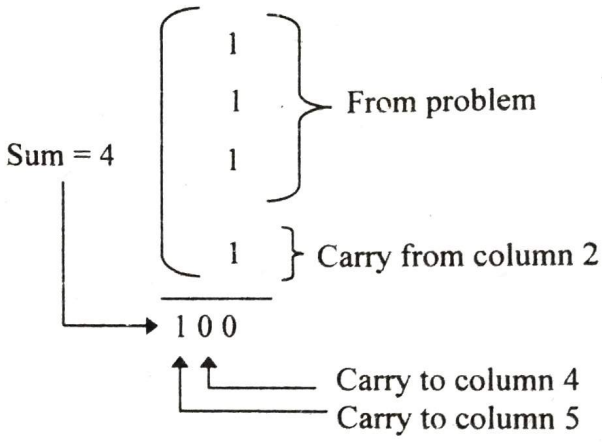
The addition in column 3 above can be handled in any one of the following two ways:

Addition handled as pairs (Column 3)



Add two carries to column 4

Addition handled as single carry (Column 3)



Additive Method of Multiplication

Most computers perform multiplication operation by the way of addition only. For example, the result of 4 x 8 can be obtained simply by adding the digit 8 four times (8 + 8 + 8 + 8). The same method may be used to perform multiplication operations in binary number system.

The idea of repeated addition may seem to be a longer way of doing things but remember that a computer is well suited to carry out addition operations at great speed. Internal circuit design of computer systems is also simplified to a great extent by using this method of multiplication.

Division

$0 \div 0$ = Divide by zero error
$0 \div 1 = 0$
$1 \div 0$ = Divide by zero error
$1 \div 1 = 1$

Binary division is performed in a manner similar to decimal division. The rules for binary division are:

1. Start from the left of the dividend.
2. Perform a series of subtractions, in which the divisor is subtracted from the dividend.
3. If subtraction is possible, put a 1 in the quotient and subtract the divisor from the corresponding digits of dividend.
4. If subtraction is not possible (divisor greater than remainder), record a 0 in the quotient.
5. Bring down the next digit to add to the remainder digits. Proceed as before in a manner similar to long division.

The example below illustrates this method.

Example 5.16

Divide 100001_2 by 110_2

Solution:

```
          0101     (Quotient)
110 ) 100001       (Dividend)
      110          1 <------ Divisor greater than 100, hence, put 0 in quotient
      1000         2 <------ Add digit from dividend to group used above
       110         3 <------ Subtraction possible, hence, put 1 in quotient
       100         4 <------ Remainder from subtraction plus digit from dividend
       110         5 <------ Divisor greater, hence, put 0 in quotient
       1001        6 <------ Add digit from dividend to group used above
        110        7 <------ Subtraction possible, hence, put 1 in quotient
         11        (Remainder)
```

Verify the result by dividing 33_{10} (100001_2) by 6_{10} (110_2), which gives a quotient of 5_{10} (101_2) and a remainder of 3_{10} (11_2).

Additive Method of Division

Ev[illegible] division ope[illegible]ion is [illegible] form[illegible] inside [illegible]

35 - 5 = 30
30 - 5 = 25
25 - 5 = 20
20 - 5 = 15
15 - 5 = 10
10 - 5 = 5
5 - 5 = 0

That is, the divisor is subtracted repeatedly from the dividend until the result of subtraction becomes less than or equal to zero. The total number of times subtraction was performed gives the value of the quotient. In this example, the value of quotient is 7 because the divisor (5) was subtracted 7 times from the dividend (35) until the result of subtraction became zero. If the result of last subtraction is zero, then there is no remainder for the division. However, if it is less than zero, then the last subtraction is ignored and the result of the previous subtraction is taken as the value of remainder. In this case, the last subtraction operation is not counted for evaluating the value of quotient. The example below illustrates the process.

Example 5.17.

Divide 33_{10} by 6_{10} using the method of addition.

Solution:

33 - 6 = 27
27 - 6 = 21
21 - 6 = 15
15 - 6 = 9
9 - 6 = 3
3 - 6 = -3

Total number of subtractions = 6.

Since the result of the last subtraction is less than zero,

Quotient = 6 - 1 (ignore last subtraction) = 5
Remainder = 3 (result of previous subtraction)

Hence, 33 ÷ 6 = 5 with a remainder 3.

It is assumed here that all subtraction operations are carried out by using complementary subtraction meth (additive method).

Once again, performing division inside a computer by the way of addition is desirable because addition and complementation operations are easily performed in a computer and usually save the labour and expense of designing complicated circuits.

We have demonstrated how computer arithmetic is based on addition. Exactly how this simplifies matter can only be understood in the context of binary (not in decimal). The number of individual steps may indeed be increased because all computer arithmetic is reduced to addition and complementation operations. However, a computer can carry out binary addition and complementation operations at such great speed that this is not a disadvantage.

Points to Remember

1. Almost all computers use binary numbers for internal computations because electronic circuits for arithmetic operations in binary mode are

2. The rules for *binary addition* are as follows:
   0 + 0 = 0

3. The rules for *binary subtraction* are as follows:
   0 - 0 = 0
   0 - 1 = 1 with a borrow from the next column
   1 - 0 = 1
   1 - 1 = 0

4. For a number having *n* digits, its *complement* is defined as the difference between the number and the base raised to the nth power minus one.

5. A quick way to obtain the *complement of a binary number* is to transform all its 0s to 1s, and all its 1s to 0s.

6. *Complementary subtraction* is an additive approach of subtraction.

7. The rules for *binary multiplication* are as follows:
   0 x 0 = 0
   0 x 1 = 0
   1 x 0 = 0
   1 x 1 = 1

8. The rules for *binary division* are as follows:
   0 ÷ 0 = Divide by zero error
   0 ÷ 1 = 0
   1 ÷ 0 = Divide by zero error
   1 ÷ 1 = 1

9. 'Divide by zero' is an error condition raised by the computer when any number is divided by zero.

10. Most computers use the *additive approach* for performing subtraction, multiplication, and division operations.

Questions

1. Why computers use the binary number system?
2. Add binary numbers 1011 and 101 in both decimal and binary forms.
3. Add binary numbers 1010110 and 1011010.
4. Add binary numbers 10111 and 1011.
5. Find the complement of following numbers:
   (a) 495_{10} (d) C_{16}
   (b) 29_{10} (e) 2_5
   (c) 4_8 (f) 32_4
6. Find the complement of following binary numbers:
   (a) 10 (d) 011011
   (b) 101 (e) 10110001
   (c) 101101 (f) 001101001110
7. Subtract 0110111_2 from 1101110_2.
8. Subtract 01010_2 from 10000_2.
9. Subtract 011011_2 from 110111_2.
10. Subtract 25_{10} from 50_{10} using complementary method.
11. Subtract 25_{10} from 20_{10} using complementary method.
12. Subtract 234_{10} from 588_{10} using complementary method.
13. Subtract 216_{10} from 172_{10} using complementary method.
14. Subtract 01010_2 from 10000_2 using complementary method.
15. Subtract 110111_2 from 101110_2 using complementary method.
16. Subtract 011011_2 from 110111_2 using complementary method.
17. Subtract 1111_2 from 1100_2 using complementary method.
18. Multiply binary numbers 1100 and 1010.
19. Multiply binary numbers 01101 and 1001.
20. Multiply binary numbers 101111 and 111.
21. Divide 11001_2 by 101_2.
22. Divide 0110111_2 by 0111_2.
23. What happens when a number is divided by zero in a computer?
24. Explain how multiplication and division operations are performed in a computer by using additive approach.
25. What is the primary advantage of performing subtraction by complementary method in digital computers?
26. Discuss the advantages and disadvantages of performing various arithmetic operations by additive method in a digital computer.