Rajmani Kumar,
Lecturer, Dept. of BCA
S.U.College, Hilsa (Nalanda)
Patliputra University, Patna

BCA-1st Year **Paper-I**

Arithmetic Operations of Binary Numbers

Binary Arithmetic:

Binary arithmetic includes the basic arithmetic operations of addition, subtraction, multiplication and division. The following sections present the rules that apply to these operations when they are performed on binary numbers.

Binary Addition:

Binary addition is performed in the same way as addition in the decimal-system and is, in fact, much easier to master. Binary addition obeys the following four basic rules:

$$\begin{array}{r} 0 \\ +0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \\ +1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +1 \\ \hline 10 \end{array}$$

The results of the last rule may seem some what strange, remember that these are binary numbers. Put into words, the last rule states that

binary one + binary one = binary two = binary "one zero"

When adding more than single-digit binary number, carry into, higher order columns as is done when adding decimal numbers. For example 11 and 10 are added as follows:

$$\begin{array}{r} 11 \\ +\ 10 \\ \hline 101 \end{array}$$

In the first column (L S C or 2°) '1 plus 0 equal 1. In the second column (2^1) 1 plus 1 equals 0 with a carry of 1 into the third column (2^2).

When we add 1 + 1.+ 1 (carry) produces 11, recorded as 1 with a carry to the next column.

Example 12: Add (a) 111 and 101 (b) 1010, 1001 and 1101.
Solution:

(a)

$$\begin{array}{r} (1)\,(1) \\ 1\,1\,1 \\ 1\,0\,1 \\ \hline 1\,1\,0\,0 \end{array}$$

(B)
$$\begin{array}{r} {\scriptstyle (2)(1)(1)(1)} \\ 1010 \\ 1001 \\ \underline{1101} \\ 10000 \end{array}$$

Binary Subtraction:

Binary subtraction is just as simple as addition subtraction of one bit from another obey the following four basic rules

$0 - 0 = 0$

$1 - 1 = 0$

$1 - 0 = 1$

$10 - 1 = 1$ with a transfer (borrow) of 1.

When doing subtracting, it is sometimes necessary to borrow from the next higher-order column. The only it will be necessary to borrow is when we try to subtract a 1 from a 0. In this case a 1 is borrowed from the next higher-order column, which leaves a 0 in that column and creates a 10 i.e., 2 in the column being subtracted. The following examples illustrate binary subtraction.

Example 13: Perform the following subtractions.

(a) 11 - 01 , (b) 11-10 (c) 100 - 011

Solution:

$$\begin{array}{rr} & 11 \\ - & \underline{01} \\ & 10 \end{array} \qquad \begin{array}{rr} & 11 \\ - & \underline{10} \\ & 01 \end{array} \qquad \begin{array}{rr} & 100 \\ - & \underline{011} \\ & 001 \end{array}$$

(a) (b (c)

Part (c) involves to borrows, which handled as follows. Since a 1 is to be subtracted from a 0 in the first column, a borrow is required from the next higher-order column. However, it also contains a 0; therefore, the second column must borrow the 1 in the third column. This leaves a 0 in the third column and place a 10 in the second column. Borrowing a 1 from 10 leaves a 1 in the second column and places a 10 i.e, 2 in the first column:

When subtracting a larger number from a smaller number, the results will be negative. To perform this subtraction, one must subtract the smaller number from the larger and prefix the results with the sign of the larger number.

Example 14: Perform the following subtraction 101 – 111.

Solution:

Subtract the smaller number from the larger.

$$\begin{array}{rr} & 111 \\ - & \underline{101} \\ & 010 \end{array}$$

Thus $1\ 0\ 1 - 1\ 1\ 1 = -\ 010 = -\ 10$

Binary multiplication:

Binary multiplication is performed in the same manner as decimal multiplication. It is much easier, since there are only two possible results of multiplying two bits. The Binary multiplication obeys the four basic rules.

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

Example 15: Multiply the following binary numbers.
(a) 101 x **1 1**
(b) 1101 x 10
(c) 1010 x **l 0 1**
(d) 1 0 1 l x 1 0 1 0
Solution

: (a)
```
  101
x 1 1
 101
101
1111
```

(b)
```
 11101
  x1 0
 0000
1101
11010
```

(c)
```
 1010
 x101
1010
0000
1010
110010
```

(d)
```
 1011
x1010
 0000
1011
0000
1011
1101110
```

Multiplication of fractional number is performed in the same way as with fractional numbers in the decimal numbers.

Example 16: Perform the binary multiplication 0.01 x **11.**
Solution:

```
  0.01
 1 1 x
  01
 01x
0.11
```

Binary Division:

Division in the binary number system employees the same procedure as division in the decimal system, as will be seen in the following examples.

Example 17: Perform the following binary division.

(a) 110 ÷ 11

1100÷1 1

Solution:

(a)

```
   10
11 110
   11
    0 0
    0 0
    0 0
```

(b)

```
   100
11 11000
   11
    0 0
    0 0
      0 0
      0 0
        0 0
```

Binary division problems with remainders are also treated the same as in the decimal system, as illustrates the following example.

Example 18: Perform the following binary division:

(a) 1111 ÷ 110 (b) 1100 ÷ 101

Solution: (a)

```
    10. 1
110 1 111.00
    1 10
      1 10
      1 10
      0 0 0
```

(b)

```
     10. 011
110 100.00
    1 01
    100
    000
     1000
      101
      110
      101
        1
```

(reminder)

Two's Complement Addition

Add the values and discard any carry-out bit.

Examples: using 8-bit two's complement numbers.

1. Add −8 to +3
2. (+3) 0000 0011
3. +(−8) 1111 1000
4. ----------
5. (−5) 1111 1011
6. Add −5 to −2
7. (−2) 1111 1110
8. +(−5) 1111 1011
9. ----------
10. (−7) 1 1111 1001 : discard carry-out

Overflow Rule for addition

If 2 Two's Complement numbers are added, and they both have the same sign (both positive or both negative), then overflow occurs if and only if the result has the opposite sign. Overflow never occurs when adding operands with different signs.

i.e. Adding two positive numbers must give a positive result Adding two negative numbers must give a negative result

Overflow occurs if

$(+A) + (+B) = -C$
$(-A) + (-B) = +C$

Example: Using 4-bit Two's Complement numbers ($-8 \leq x \leq +7$)

(−7) 1001
+(−6) 1010

(−13) 1 0011 = 3 : Overflow (largest −ve number is −8)

A couple of definitions: Subtrahend: what is being subtracted

Minuhend: what it is being subtracted from

Two's Complement Subtraction

Normally accomplished by negating the subtrahend and adding it to the minuhend. Any carry-out is discarded.

Example: Using 8-bit Two's Complement Numbers ($-128 \leq x \leq +127$)

```
(+8) 0000 1000                    0000 1000
−(+5) 0000 0101 -> Negate -> +1111 1011
-----                             -----------
(+3)                          1 0000 0011 : discard carry-out
```

Overflow Rule for Subtraction

If 2 Two's Complement numbers are subtracted, and their signs are different, then overflow occurs if and only if the result has the same sign as the subtrahend.

Overflow occurs if

$(+A) - (-B) = -C$
$(-A) - (+B) = +C$

Example: Using 4-bit Two's Complement numbers ($-8 \leq x \leq +7$)

Subtract −6 from +7

```
  (+7) 0111                0111
−(−6) 1010 -> Negate -> +0110
  ----------               -----
    13                     1101 = −8 + 5 = −3 : Overflow
```

BCD CODE:

Binary Coded Decimal – In this code each digit of a decimal number system is converted into its Binary equivalent rather than converting the entire decimal value into a pure Binary form.

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

We use group of 4 bits to represent a digit in BCD. 4 bits can represent only digits because 4 bits are insufficient to represent various characters used by the computer. In 6 bit BCD code 2 additional zone bits are added & we can represent 64 (2^6) different characters. This is sufficient number to code the decimal digits (10), alphabetic characters (26) & other special symbols (28).

EBCDIC CODE:

Extended Binary Coded Decimal Interchange Code- In this code it is possible to represent 256 (2^8). It also allows a large variety of printable characters & non printable control characters.

EBCDIC can be easily divided into 2 4 bit groups. Each of these 4 bit groups can be represented by 1 hexa digit. Thus Hexadecimal Number system is used as a shortcut notation for memory dump by computers that use EBCDIC for internal representation of characters.

Two types of formats :

1. Zoned Decimal Format
2. Packed Decimal Format.

When a numeric value is represented in EBCDIC, to represent whether number is positive, negative or unsigned sign indicator is used in the zone position of the rightmost digit.

Printers print only those numeric charaters that are in a Zoned Decimal Format so this format is useful while printing the Data.

Most computers cannot perform arithmetic operations on Zoned Decimal Data. To perform arithmetic calculation it had to be converted to Packed Decimal Format.

ASCII:

American Standard Code for Information Interchange - It is accepted by several computer manufacturers as their computer's internal code. This code is popular in data communications, is used almost exclusively to represent data internally in micro-computers.

ASCII is of two types:-

ASCII – 7 $\longrightarrow$ 7 bit code $2^7 = 128$

ASCII – 8 $\longrightarrow$ 8 bit code $2^8 = 256$

Additional bit is added to the Zone bit.