Learning objectives

After studying this lesson, student should be able to:

- Represent binary number using the convention Sign magnitude and 2’s complement
- Perform binary subtraction using 2’s complement.
- Represent floating point binary numbers.
- Describe and use the BCD, EBCDIC, ASCII code scheme.

Contents

I. REPRESENTATION OF NEGATIVE BINARY NUMBER .......................................................... 2
II. ADDITION—SUBTRACTION OF SIGNED NUMBERS USING 2S COMPLEMENT
ADDITION.................................................................................................................................. 6
III. BINARY CODING SCHEME................................................................................................. 7
EXERCISES.................................................................................................................................... 10
I. REPRESENTATION OF NEGATIVE BINARY NUMBER

We have just seen how to carry out arithmetic operations on positive numbers. We are going to see here different ways of representing negative numbers in binary. Usually a given computer uses a fixed number of bits for storing integers. So we use terms such as 8-bits integer, 16-bits integers, ... Whatever the principle remains the same, the only difference is that with more bits, we can store wider range of number. In general, with n bits, one can store $2^n$ numbers.

I.1 Signed and unsigned numbers

A binary number may be positive or negative. Generally, we use the symbol “+” and “-” to represent positive and negative numbers, respectively. The sign of a binary number has to be represented using 0 and 1, in the computer. An n-bit signed binary number consists of two parts – sign bit and magnitude. The left most bit, also called the Most Significant Bit (MSB) is the sign bit. The remaining n-1 bits denote the magnitude of the number.

![MSB and Magnitude](image)

The rules for signed and unsigned binary numbers are simple and are explained as follows:

- In an unsigned number, the MSB is a weighted position bit.
- In a signed number, the MSB (the sign bit) is 0 for a positive number.
- In a signed number, the MSB (the sign bit) is 1 for a negative number.

For example, 01100011 is a positive number since its sign bit is 0, and, 11001011 is a negative number since its sign bit is 1. An 8-bit signed number can represent data in the range -128 to +127 (-2^7 to +2^7-1). In general, using n bits, the range of integer we can store using two’s complement is $-2^{n-1}$ to $2^{n-1}$

I.2 Complements

The complement of a number is the number, which when added to the original will make it equal to a multiple of the base number system.

The complement of a number can be used as a representation of that number as a negative and as a positive number that represents a negative. It is a method, which can be used to make the subtraction easier for machines. Consequently, complements are used in the digital computers for simplifying the subtraction operation and for the logical operation.

For every base r system, there are two types of complements: rs complement and $(r - 1)s$ complement.
• For decimal \( r = 10 \), we have 9s and 10s complement.
• For binary \( r = 2 \), we have 1s and 2s complement.
• For octal \( r = 8 \), we have 7s and 8s complement.
• For hexadecimal \( r = 16 \), we have 15s and 16s complement.

There are two types of complements for the binary number system – 1’s complement and 2’s complement.

### I.2.1 The 1s Complement

To form the negative of any number, first complement all the bits of that number. This result is known as the 1s complement of the original number. This requires changing every logic 1 bit in a number to logic 0, and every logic 0 bit to logic 1.

For instance

- 1’s complement of 110 is 001
- 1’s complement of 1011 is 0100
- 1’s complement of 1101111 is 0010000

### I.2.2 The 2s Complement

We do not just place 1 in the MSB of a binary number to make it negative. We must take the 2s complement of the number. Taking the 2s complement of the number will cause the MSB to become 1. Using \( n \) bits, the range of integer we can store using two’s complement is \(-2^{n-1}\) to \(2^{n-1}-1\).

To obtain the 2s complement of a number, there is a two-step process:

1. Take the 1s complement of the number by changing every logic 1 bit in the number to logic 0 bit, and change every logic 0 bit to logic 1 bit.
2. Add 1 to the 1s complement of the binary number. Now, we have the 2s complement of the original number. Here, we can notice that the MSB has become 1.

For instance,

- 2’s complement of 110 is 001 + 1 = 010
- 2’s complement of 1011 is 0100 + 1 = 0101
- 2’s complement of 1101111 is 0010000 + 1 = 0010001

### I.3 Binary Data Representation

A binary number may also have a binary point, in addition to the sign. The binary point is used for representing fractions, integers and integer-fraction numbers. There are two ways of representing the position of the binary point in the register - fixed point number representation and floating point number representation.
• The **fixed point number representation** assumes that the binary point is fixed at one position. The binary point is not actually present in the register, but its presence is assumed based on whether the number which is stored is a fraction or an integer.

• The **floating point number representation** uses two registers. The first register stores the number without the binary point. The second register stores a number that indicates the position of the binary point in the first register.

We shall now discuss representation of data in the fixed point number representation and floating point number representation.

### I.3.1 Fixed Point Number Representation

The integer binary signed number is represented as follows –

- For a **positive integer binary number**, the sign bit is 0 and the magnitude is a positive binary number.
- For a **negative integer binary number**, the sign bit is 1. The magnitude is represented in any one of the three ways—
  - **Signed Magnitude representation** - The magnitude is the positive binary number itself.
  - **Signed 1’s complement representation** - The magnitude is the 1’s complement of the positive binary number.
  - **Signed 2’s complement representation** - The magnitude is the 2’s complement of the positive binary number.

The table below shows the representation of the signed number 18.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>+18</td>
<td>00010010</td>
<td>Sign bit is 0. 0010010 is binary equivalent of +18</td>
</tr>
<tr>
<td>-18</td>
<td>Signed magnitudes</td>
<td>10010010 Sign bit is 1. 0010010 is binary equivalent of +18</td>
</tr>
<tr>
<td></td>
<td>Signed 1’s complement</td>
<td>11101101 Sign bit is 1. 1101101 is 1’s complement of +18</td>
</tr>
<tr>
<td></td>
<td>Signed 2’s complement</td>
<td>11101110 Sign bit is 1. 1101110 is 2’s complement of +18</td>
</tr>
</tbody>
</table>

Signed magnitude and signed 1’s complement representation are seldom used in computer arithmetic.

### Representation Comparison for 8-bit Binary Numbers

<table>
<thead>
<tr>
<th>Representation</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Number of integers represented</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsigned</td>
<td>0</td>
<td>255</td>
<td>256</td>
</tr>
<tr>
<td>2’s Complement</td>
<td>-128</td>
<td>127</td>
<td>256</td>
</tr>
<tr>
<td>Signed Magnitude</td>
<td>-127</td>
<td>127</td>
<td>255</td>
</tr>
</tbody>
</table>
I.3.2 Floating Point Number Representation

We are familiar with the fact that any number can be written as a fraction multiplied by 10 to some power. For instance

123 can be written as 0.123 \times 10^3
0.0187 can be written as 0.187 \times 10^{-1}
-57.3 can be written as -0.573 \times 10^2

In the first example we can identify
(a) A Sign (not shown explicitly since it is positive)
(b) A fractional part 0.123 (called the mantissa)
(c) The base 10
(d) An exponent,

We can identify similar components for the other two examples. Note that all the examples have the common base, 10, thus if the base is understood, we can represent any number by specifying its sign, mantissa and exponent. (-, 0.573, 2) refers to -0.573 \times 10^2

The similar method is used for representing real numbers on a computer. The base varies from one machine to another. But the common bases are 2, 10 and 16. In order to keep things simple, we assume that 8 bits are to be used to store a floating point number, and that the base 2 is understood. Let’s then assume the following:

- 1 bit for the sign (0 for positive, 1 for negative)
- 4 for the mantissa
- 3 for the exponent

Our floating number representation should have the following format

\[
\begin{array}{c}
S \\
E \\
M \\
\end{array}
\]

We will note the following

- The sign bit is 0 for a positive number and 1 for a negative number
- The mantissa is to be treated as a binary fraction, where the binary point is assumed to be placed immediately in front of the leftmost bit. Thus the mantissa 1001 is taken to mean the binary fraction 0.1001
- The exponent is stored using ‘sign and magnitude’. With 3 bits the range of exponent is -3 to +3. -3 is stored as 111 and +3 is stored as 011

Examples

\(\text{(1) What floating number is represented by 001011010}\)
- The sign is positive
- The mantissa is 1010 represented the number 0.1010
- The exponent is 010 which represent 2
Therefore the number represented is \(0.1010 \times 2^2 = (10.10)_2 = (2.5)_{10}\)

**2) What is the binary representation of \((-2.75)_{10}\)**

\((2.75)_{10} = (10.11)_2 = (0.1011)_2 \times 2^2\)

- The sign is negative, hence 1
- The exponent is two, 010
- The mantissa is 1011

The representation of the number is then: **10101011**

**Features of the floating point representation**

(a) A zero is represented by 0 000 0000

(b) The smallest positive number is represented by 0 111 0001. The value represented is \((0.0001)_2 \times 2^{-3} = (0.0078125)_{10}\)

(c) The largest positive number is represented by 0 111 1111. The value represented is \((0.1111)_2 \times 2^3 = (111.1)_{10} = (7.5)_{10}\)

(d) The range of numbers which can be represented is -7.5 to +7.5

**II. ADDITION—SUBTRACTION OF SIGNED NUMBERS USING 2S COMPLEMENT ADDITION**

The addition of signed binary numbers represented in the radix complement form is similar to the unsigned case. However, when the 2s complement of a number is added to any other binary number, it will be equivalent to its subtraction from that number. As a result, subtraction of the signed numbers by 2s complement method is performed by using the following steps:

1. Convert both the numbers into the binary equivalent form.
2. Find the 2s complement form of the number, which is subtracting, that is, subtrahend.
3. Add this 2s complement number to the minuend.
4. If there is carry of 1, ignore it from the result to obtain the correct result.
5. If there is no carry, re-complement the result and attach the negative sign to the obtained result.

Please note that the negative output is automatically in the 2’s complement form. We get the decimal equivalent of the negative output number, by finding its 2’s complement, and attaching a negative sign to the obtained result.

Let’s understand the addition of two signed binary numbers with the help of some examples.

**Example 1:** Add \((27)_{10}\) and \((-11)_{10}\) using complementary representation for the negative value.

Binary form of \((27)_{10} = (011011)_2\) and of \((11)_{10} = (001011)_2\)
Get the 2s complement of (001011) = 1s complement of (001011) + 1

= 110100 + 1

= 110101

Add (011011)\(_2\) and (110101)\(_2\)

\[\begin{array}{c}
011011 \\
+110101 \\
\hline
101000
\end{array}\]

Hence, the result is (010000)\(_2\) or (16)\(_{10}\).

**Binary Data Representation**

**Example 2.** Add (-5)\(_{10}\) and (-10)\(_{10}\) using 2’s complement.

- We represent -5 in 2’s complement form, i.e., 1111 1011.
- We represent -10 in 2’s complement form, i.e., 1111 0110.
- Add the two numbers. The result is 1111 0001. The result is in 2’s complement form.

To find the decimal equivalent of the result 1111 0001 - Find the 2’s complement of 1111 0001, i.e., 0001 1110 + 1 = 0001 1111. This is binary equivalent of +15. Attaching a negative sign to the obtained result gives us -15.

**III. BINARY CODING SCHEME**

The alphabetic data, numeric data, alphanumeric data, symbols, sound data and video data, all are represented as combination of bits in the computer. The bits are grouped in a fixed size, such as 8 bits, 6 bits or 4 bits. A code is made by combining bits of definite size. Binary Coding schemes represent the data such as alphabets, digits 0-9, and symbols in a standard code. A combination of bits represents a unique symbol in the data. The standard code enables any programmer to use the same combination of bits to represent a symbol in the data.

The binary coding schemes that are most commonly used are

- Binary Coded Decimal (BCD)
- American Standard Code for Information Interchange (ASCII)
II.1 Binary Coded Decimal (BCD)

**Binary Coded Decimal** (BCD) is a method of using binary digits to represent the decimal digits 0–9. A decimal digit is represented by four binary digits. The BCD coding is the binary equivalent of the decimal digit. BCD system was developed by the IBM (International Business Machines) corporation. In this system, each digit of a number is converted into its binary equivalent rather than converting the entire decimal number to its binary form. Similarly, letters and special characters can be coded in the binary form.

Let us determine the BCD value for the decimal number 5319. Since there are four digits in the decimal number, there are four bytes in the BCD number. They are:

<table>
<thead>
<tr>
<th>Thousands—Hundreds</th>
<th>Tens—Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>53</td>
<td>19</td>
</tr>
<tr>
<td>0101 0011</td>
<td>0001 1001</td>
</tr>
</tbody>
</table>

Binary code decimal digits (0–9) are represented by using four bits. The valid combinations of bits and their respective values are shown in the table below.

<table>
<thead>
<tr>
<th>Decimal Code</th>
<th>BCD Digit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
</tr>
</tbody>
</table>

**Table. Binary-coded Decimal**

To represent the signs + and -, any of the remaining 4 bits patterns can be used. One convention uses 1010 for + and 1011 for -. You can see that in one byte we can have either 2 digits or a sign and one digit. Here we must use the full 4-bits representation of any digit (including the leading 0s). Thus the BCD representation of 5319 is 0101001100011001 and not 101001100011001

II.2 Extended Binary-coded Decimal Interchange Code (EBCDIC)

- The Extended Binary Coded Decimal Interchange Code (EBCDIC) uses 8 bits (4 bits for zone, 4 bits for digit) to represent a symbol in the data.
• EBCDIC allows $2^8 = 256$ combinations of bits.  
• 256 unique symbols are represented using EBCDIC code. It represents decimal numbers (0-9), lower case letters (a-z), uppercase letters (A-Z), Special characters, and Control characters (printable and non-printable e.g. for cursor movement, printer vertical spacing etc.).  
• EBCDIC codes are used, mainly, in the mainframe computers.

II.3 American Standard Code for Information Interchange (ASCII)

The name ASCII is an acronym for: American Standard Code for Information Interchange. It is a character encoding standard developed several decades ago to provide a standard way for digital machines to encode characters.

Some important things to note about ASCII code:

• ASCII codes are of two types – ASCII-7 and ASCII-8.  
• ASCII-7is a 7-bit standard ASCII code. In ASCII-7, the first 3 bits are the zone bits and the next 4 bits are for the digits. ASCII-7 allows $2^7 = 128$ combinations. 128 unique symbols are represented using ASCII-7. ASCII-7 has been modified by IBM to ASCII-8.  
• ASCII-8is an extended version of ASCII-7. ASCII-8 is an 8-bit code having 4 bits for zone and 4 bits for the digit. ASCII-8 allows $2^8 = 256$ combinations. ASCII-8 represents 256 unique symbols. ASCII is used widely to represent data in computers.  
• The ASCII-8 code represents 256 symbols.  
  o Codes 0 to 31 represent control characters (non-printable), because they are used for actions like, Carriage return (CR), Bell (BEL) etc.  
  o Codes 48 to 57 stand for numeric 0-9.  
  o Codes 65 to 90 stand for uppercase letters A-Z.  
  o Codes 97 to 122 stand for lowercase letters a-z.  
  o Codes 128-255 are the extended ASCII codes.

II.4 Unicode

• Unicode is a universal character encoding standard for the representation of text which includes letters, numbers and symbols in multi-lingual environments. The Unicode Consortium based in California develops the Unicode standard.  
• Unicode uses 32 bits to represent a symbol in the data.  
• Unicode allows $2^{32} = 4164895296$ (~ 4 billion) combinations.  
• Unicode codes can uniquely represent any character or symbol present in any language like Chinese, Japanese etc. In addition to the letters; mathematical and scientific symbols are also represented in Unicode codes.  
• An advantage of Unicode is that it is compatible with the ASCII-8 codes. The first 256 codes in Unicode are identical to the ASCII-8 codes.
• Unicode is implemented by different character encodings. UTF-8 is the most commonly used encoding scheme. UTF stands for Unicode Transformation Format. UTF-8 uses 8 bits to 32 bits per code.

EXERCISES

Ex 1

1) Using Sign and magnitude, what range of integer can be stored using 6 bits, 8 bits, 16 bits.
2) Using 6 bits, give the sign and magnitude representation of 14, -15, 31, -17
3) Using 6 bits, give the two complement representation of: 15, -28, 31, -1

Ex 2

1) What is the BCD representation of
   (a) 215
   (b) -215
   (c) 2001

2) What numbers are represented by the following bit strings assume that when there is a sign, it is represented by the first 4 bits with the convention 1010 for + and 1011 for -.
   (a) 10101100100110111
   (b) 10110011011110010100
   (c) 10011000000000010
   (d) 1010010101011011

Ex 3

1) What is the smallest and the largest floating point value which may be represented using the representation described in the note?

2) What floating point numbers are represented by
   (a) 1 001 1100
   (b) 0 101 1100
   (c) 0 000 1000
   (d) 1 100 0101
   (e) 1 111 1111
   (f) 0 010 1110
   (g) 1 100101010101101

3) Using the representation described in the note, give the floating point representation of each of the following. State whether the representation is exact or not. If inexact calculate the error in the representation.
   (a) 5.25
   (b) 2.1875
   (c) -4.5
   (d) 7.35
   (e) -3.3
   (f) 6.256
   (g) -7.7
   (h) 1.115
   (i) -3.75

Ex 4: Numbers are stored in an 8-bits byte of a computer in a normalized floating point representation as follow:

\[
\begin{array}{c}
S & M & E \\
\end{array}
\]

- S is the sign (0 for positive and 1 for negative)
- M is the mantissa with binary point assumed to be just before the leftmost bit
- E is the exponent (a power of two) stored as a two’s complement integer
(a) What are the largest and the smallest positive value which may be represented?
(b) Give, in binary form, the best possible representation of the values 0.875 and -1.4, indicating the actual value stored in each case.
(c) What is the smallest positive number which may be represented if the mantissa is allowed to be unnormalized

**Ex 5** A certain computer uses 11-bits words for storing a floating point number:
- 1 bit represent the sign, S; 0 for positive and 1 for negative
- The exponent, E, is a 4-bits value, stored in 2’s complement form
- The mantissa, M, is a 6-bits normalized value, with the binary point assumed to be just in front of the leftmost bit.

Give the representation of: 0.3, 7.3 and 6.375, Stating the error in each representation.

**Ex 5**
(a) What is the lowest possible value for an 8-bit signed magnitude binary number?
(b) What is the highest possible value for a 10-bit 2’s complement binary number?

**Ex 6**
1) Convert each of the following decimal values to 8-bit 2’s complement binary.
   a) 54_{10}  
b) -49_{10}  
c) -128_{10}  
d) -66_{10}  
e) -98_{10}

2) Convert each of the following 8-bit 2’s complement binary numbers to decimal.
   a) 1001101_2  
b) 00010101_2  
c) 11100110_2  
d) 01101001_2

**Ex 7**
1) Convert each of the following decimal values to 8-bit signed magnitude binary.
   a) 54_{10}  
b) -49_{10}  
c) -127_{10}  
d) -66_{10}  
e) -98_{10}

2) Convert each of the following 8-bit signed magnitude binary numbers to decimal.
   a) 1001101_2  
b) 00010101_2  
c) 11100110_2  
d) 01101001_2

**Ex 8:** Using 1’s and 2’s complements perform the following subtractions
   a. 100110 – 11011 
b. 1101010 – 110100 
c. 10011.1101 – 101.11 
d. 1010 - 11011

**Ex 9:** Find the following differences using 2’s complement arithmetic. First convert decimal values to corresponding binary values. Next, find the two’s complement representation of the subtrahend. Add the minuend and 2’s complemented subtrahend. Check your answers.
   a) 12 – 6  
b) 4 – 6  
c) 3.125 – 6.5  
d) 67.25 – 83.125