• To represent signed integers, computer systems allocate the high-order bit to indicate the sign of a number.
  o The high-order bit is the leftmost bit. It is also called the most significant bit.
  o 0 is used to indicate a positive number;
  o 1 indicates a negative number.

• The remaining bits contain the value of the number
• There are three ways in which signed binary integers may be expressed:
  o Signed magnitude
  o One’s complement
  o Two’s complement

2.4.1 Signed Magnitude

• In an 8-bit word, signed magnitude representation places the absolute value of the number in the 7 bits to the right of the sign bit.

• For example, in 8-bit signed magnitude representation:
  +3 is: 00000011
  -3 is: 10000011

• Binary addition is as easy as it gets. You need to know only four rules:
  0 + 0 = 0  0 + 1 = 1
  1 + 0 = 1  1 + 1 = 10
• The simplicity of this system makes it possible for digital circuits to carry out arithmetic operations.

• Example:
  o Using signed magnitude binary arithmetic, find the sum of 75 and 46.
    1. First, convert 75 and 46 to hexadecimal.
       Divisor  Dividend  Quotient  Remainder  Foreign Digit
       16       75        4         11         B
       16       4         0         4          4
       75       =         4B

       Divisor  Dividend  Quotient  Remainder  Foreign Digit
       16       46        2         14         E
       16       2         0         2          2
       46       =         2E

    2. Next convert the hexadecimal representations of 75 and 46 to binary.
       75 = 4B16 = 0100 10112
       46 = 2E16 = 0010 11102

    3. Next, select a field width for the unsigned numbers. Select 7 bits
4. Next, put the signs and the unsigned binary values in a form suitable for addition.

\[
\begin{array}{cccccc}
\text{Sign} & 1 & 1 \\
0 & 1 & 0 & 0 \quad 1 & 0 & 1 & 1 \\
+ & 0 & 0 & 1 & 0 \quad 1 & 1 & 1 & 0 \\
\hline \\
\text{Sum} & 1 & 1 & 1 & 1 \quad 1 & 0 & 0 & 1 \\
\end{array}
\]

5. Next, find the sum.

\[
\begin{array}{cccccc}
\text{Sign} & 1 & 1 & 1 \\
0 & 1 & 0 & 0 \quad 1 & 0 & 1 & 1 \\
+ & 0 & 0 & 1 & 0 \quad 1 & 1 & 1 & 0 \\
\hline \\
\text{Sum} & 0 & 1 & 1 & 1 \quad 1 & 0 & 0 & 1 \\
\end{array}
\]

6. Next, convert the binary value to hex.

\[1111001_2 = 79_{16}\]

7. Next, convert the hexadecimal value to decimal.

<table>
<thead>
<tr>
<th>Foreign Digit</th>
<th>Decimal</th>
<th>Position</th>
<th>Foreign Base</th>
<th>Exponentiated Base</th>
<th>Subtotal</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
<td>1</td>
<td>16</td>
<td>16</td>
<td>112</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>0</td>
<td>16</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>16</td>
<td>1</td>
<td>121</td>
</tr>
</tbody>
</table>

8. Next, perform the addition in decimal.

\[
\begin{array}{c}
75 \\
+ \quad 46 \\
\hline \\
121
\end{array}
\]

9. Finally, check to see that both sums are equal.

- Example:
  - Using signed magnitude binary arithmetic, find the difference of 13 minus 19.
    1. First, convert 75 and 46 to hexadecimal.

\[
\begin{array}{cccc}
\text{Divisor} & \text{Dividend} & \text{Quotient} & \text{Remainder} & \text{Foreign Digit} \\
16 & 19 & 1 & 3 & 3 \\
16 & 1 & 0 & 1 & 1 \\
\text{19} & = & \text{13} \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{Divisor} & \text{Dividend} & \text{Quotient} & \text{Remainder} & \text{Foreign Digit} \\
16 & 13 & 0 & 13 & D \\
\text{13} & = & \text{D} \\
\end{array}
\]
2. Next convert the hexadecimal representations of 75 and 46 to binary.

\[ 19 = 13_{16} = 0001\ 0011_2 \]
\[ 13 = 0D_{16} = 0000\ 1101_2 \]

3. Next, select a field width for the unsigned numbers. Select 7 bits

4. Next, put the signs and the unsigned binary values in a form suitable for addition.

\[
\begin{array}{ccccccc}
\text{Sign} & 1 & 0 & 0 & 0 & 1 & 0 \\
\text{Borrow} & 0 & 0 & 0 & 0 & 1 & 1 \\
+ & 0 & 0 & 0 & 0 & 1 & 1 \\
\hline
\text{Sum} & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{array}
\]

5. Next, find the sum.

\[
\begin{array}{ccccccc}
\text{Sign} & 1 & 0 & 0 & 0 & 1 & 2 \\
\text{Borrow} & 0 & 0 & 0 & 1 & 0 & 1 \\
+ & 0 & 0 & 0 & 0 & 1 & 1 \\
\hline
\text{Sum} & 0 & 0 & 0 & 0 & 1 & 1 \\
\end{array}
\]

2.4.2 Complement Systems

- Employ a fixed field to represent an integer.
- Use one-half of the range to represent negative values.

2.4.2.1 One’s Complement

- Invert all bits \(1 \rightarrow 0, 0 \rightarrow 1\)

Example: consider

\[ \begin{align*}
10011110, \text{ the one’s complement is} \\
01100001
\end{align*} \]

Think of the one’s complement as the difference between the initial operand and a number of equal length having a one in every position.

Example:

\[ \begin{align*}
11111111 \\
-10011110 \\
01100001
\end{align*} \]

- With one’s complement addition, the carry bit is “carried around” and added to the sum.
2.4.2.2 Two’s Complement

Two’s complement is one more than the one’s complement

Example: find the two’s complement of 10011110

1. Find the one’s complement
   01100001

2. Add one to find the two’s complement
   01100010

1. Choose a field width. Common field widths are 4, 8, 16, 32, and 64 bits.
2. A binary number is positive if the most significant digit is zero (0), otherwise it is negative.

Example: Find the decimal equivalent of the following 16-bit two’s complement number

1 001 1100 0000 0101

1. Make the two’s complement number positive. Find the magnitude of the two’s complement.
   1.1. First, find the one’s complement by inverting all the bits.
   001 1100 0000 0101
   0110 0011 1111 1010

   1.2. Next, add one (1) to find the two’s complement.
0110 0011 1111 1010
    +1
0110 0011 1111 1011

2. Convert to decimal.
   2.1. First convert to hexadecimal.

   0110 0011 1111 1011
   6 3 F B

   2.2. Next, convert to decimal.

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec.</th>
<th>16^3</th>
<th>16^2</th>
<th>16^1</th>
<th>16^0</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>24576</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>768</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>15</td>
<td>240</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>11</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>25595</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5
2.4.2.2 r’s Complement

\[(N)_{r,c} = r^n - N \text{ if } N \neq 0, 0 \text{ if } N = 0 \]

where

\[n = \text{number of digits in the integer portion of } N\]
\[r = \text{radix}\]

Example: Find the ten’s complement of 147

\[(147)_{10,c} = 3, r = 10, (147)_{10,c} = 10^3 - 147 = 853\]

Example: Find the ten’s complement of 0.53

\[(0.53)_{10,c} = 0, r = 10, (0.53)_{10,c} = 10^0 - 0.53 = 0.47\]

Example: Find the two’s complement of 1010

\[(1010)_{2,c} = 4, r = 2, (1010)_{2,c} = 2^4 - 1010_2 = 0110\]

\[2^4 = \begin{array}{c}
10000 \\
- 1010 \\
\hline
0110
\end{array}\]

Example: Find the two’s complement of 1010.101

\[(1010.101)_{2,c} = 4, r = 2\]

\[2^4 = \begin{array}{c}
10000.000 \\
- 1010.101 \\
\hline
0101.011
\end{array}\]
Subtraction with r’s complement

$M$ and $S$ are two positive base $r$ numbers were $r$ must be evenly divisible by 2. Find the difference $D = M - S$

1. Define a field width and add $M$ to the $r$’s complement of $S$. Discard any digits that carry into positions more significant than those defined by the field width. Discard any carry out digits.

Example Find the difference $(1010 - 0111)$
Find the 2’s complement of $S = 0111$

$$
\begin{array}{c}
1\text{’s complement of } S \\
1000 \\
\text{Add 1 to find 2’s complement of } S \\
1001
\end{array}
$$

Add $M$ to the $r$’s complement of $S$.

$$
\begin{array}{c|c}
M & 1010 \\
2\text{’s complement of } S & 1001 \\
\hline
D = M - S & 10011 \\
\text{Final result} & 0011
\end{array}
$$

2. Determining the sign.

2.1. If the most significant digit, $d \geq \frac{r}{2}$, then the number is negative, otherwise it is positive.

2.2. To find the magnitude of a negative number represented in $r$’s complement form, complement the number.

Example Find the difference $(0111 - 1010)$

Find the 2’s complement of $S = 0111$

$$
\begin{array}{c}
1\text{’s complement of } S \\
1010 \\
\text{Add 1 to find 2’s complement of } S \\
0110
\end{array}
$$

Add $M$ to the $r$’s complement of $S$.

$$
\begin{array}{c|c}
M & 0111 \\
2\text{’s complement of } S & 0110 \\
\hline
D = M - S & 1101 \\
1\text{’s complement} & 0010 \\
2\text{’s complement} & 0011
\end{array}
$$
3. Determining if overflow occurred.
   3.1. Overflow occurs when adding two positive numbers or when adding two
       negative numbers. When the magnitude of the sum exceeds the range of
       values that can be represented in the field an overflow has occurred.

       Example: Consider 10’s complement representation and a 2-digit field. A
       number, $n$, ranges over the interval, $-50 \leq n \leq 49$. When the sum is
       greater than 49 or less than -50 an overflow has occurred.

   3.2. Special case: Overflow in 2’s complement numbers. Overflow occurs when
       the carry into the sign bit is unequal to the carry out.

       Example: consider a 4-bit 2’s complement number. A number, $n$, ranges
       over the interval, $-8 \leq n \leq 7$. We consider two possibilities.

       | Decimal | Binary | 2’s Complement |
       |---------|--------|---------------|
       | -5      | 0101   | 1011          |
       | -4      | 0100   | 1100          |
       | -9      | 0100   | 10111         |

       | Decimal | Binary | 2’s Complement |
       |---------|--------|---------------|
       | 5       | 0101   | 0101          |
       | +4      | 0100   | 0100          |
       | 9       | 0100   | 01001         |

2.4.3 Excess-M Representation for Signed Numbers

   • Excess-M representation (also called offset binary representation) is another way for
     unsigned binary values to represent signed integers.
     - Excess-M representation is intuitive because the binary string with all 0s
       represents the smallest number, whereas the binary string with all 1s
       represents the largest value.
   • An unsigned binary integer $M$ (called the bias) represents the value 0, whereas all
     zeroes in the bit pattern represents the integer $2M$.
   • The integer is interpreted as positive or negative depending on where it falls in the
     range.
   • If $n$ bits are used for the binary representation, we select the bias in such a manner
     that we split the range equally.
• Typically we choose a bias of $2^{n-1} - 1$.
  – For example, if we were using 4-bit representation, the bias should be $2^3 - 1 = 7$.
• Just as with signed magnitude, one’s complement, and two’s complement, there is a specific range of values that can be expressed in $n$ bits.

• The unsigned binary value for a signed integer using excess-$M$ representation is determined simply by adding $M$ to that integer.
  – For example, assuming that we are using excess-7 representation, the integer $0_{10}$ is represented as $0 + 7 = 7_{10} = 0111_2$.
  – The integer $3_{10}$ is represented as $3 + 7 = 10_{10} = 1010_2$.
  – The integer -7 is represented as $-7 + 7 = 0_{10} = 0000_2$.
  – To find the decimal value of the excess-7 binary number $1111_2$ subtract 7: $1111_2 = 15_{10}$ and $15 - 7 = 8$; thus $1111_2$, in excess-7 is $+8_{10}$.

• Let’s compare our representations:

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary (for absolute value)</th>
<th>Signed Magnitude</th>
<th>One’s Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>00000010</td>
<td>00000010</td>
<td>00000010</td>
</tr>
<tr>
<td>-2</td>
<td>00000010</td>
<td>10000010</td>
<td>11111101</td>
</tr>
<tr>
<td>100</td>
<td>01100100</td>
<td>01100100</td>
<td>01100100</td>
</tr>
<tr>
<td>-100</td>
<td>01100100</td>
<td>11100100</td>
<td>10011011</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary (for absolute value)</th>
<th>Two’s Complement</th>
<th>Excess-127</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>00000010</td>
<td>00000010</td>
<td>1000001</td>
</tr>
<tr>
<td>-2</td>
<td>00000010</td>
<td>11111111</td>
<td>01111101</td>
</tr>
<tr>
<td>100</td>
<td>01100100</td>
<td>01100100</td>
<td>1110011</td>
</tr>
<tr>
<td>-100</td>
<td>01100100</td>
<td>10011100</td>
<td>00011011</td>
</tr>
</tbody>
</table>

2.4.4 Unsigned Versus Signed Numbers

$1101_2 = 13 \text{ (unsigned)} = 3\text{ (two’s complement)}$

2.4.5 Computers, Arithmetic, and Booth’s Algorithm

Read for yourself.
2.4.6 Carry Versus Overflow

- Overflow is used only with signed numbers (usually in complement representation).
- Carry meaning carry-out of the leftmost bit can occur in both signed and unsigned numbers.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
<th>Carry?</th>
<th>Overflow?</th>
<th>Correct Result?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100 + 0010</td>
<td>0110</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>0100 + 0110</td>
<td>1010</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>1100 + 1110</td>
<td>1010</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>1100 + 1010</td>
<td>0110</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 2.2 Examples of Carry and Overflow in Signed Numbers

2.4.7 Binary Multiplication and Division Using Shifting

- We can do binary multiplication and division by 2 very easily using an arithmetic shift operation.
- A left arithmetic shift inserts a 0 in for the rightmost bit and shifts everything else left one bit; in effect, it multiplies by 2.
- A right arithmetic shift shifts everything one bit to the right, but copies the sign bit; it divides by 2.
- Let’s look at some examples.

Example: Multiplication

Multiply the value 11 (expressed using 8-bit signed two’s complement representation) by 2.

We start with the binary value for 11:

00001011 (+11)

We shift left one place, resulting in:

00010110 (+22)

The sign bit has not changed, so the value is valid.

Example: Division

Divide the value 12 (expressed using 8-bit signed two’s complement representation) by 2.

We start with the binary value for 12:

00001100 (+12)

We shift left one place, resulting in:

00000110 (+6)

(Remember, we carry the sign bit to the left as we shift.)