Number Representation

Part I
Basics of Number System

- We are accustomed to using the so-called decimal number system
  - Ten digits :: 0,1,2,3,4,5,6,7,8,9
  - Every digit position has a weight which is a power of 10
  - **Base** or **radix** is 10

Example:

\[ 234 = 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0 \]
\[ 250.67 = 2 \times 10^2 + 5 \times 10^1 + 0 \times 10^0 + 6 \times 10^{-1} + 7 \times 10^{-2} \]
Binary Number System

- Two digits:
  - 0 and 1
  - Every digit position has a weight which is a power of 2
  - Base or radix is 2

- Example:
  \[110 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0\]
  \[101.01 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}\]
Positional Number Systems (General)

**Decimal Numbers:**
- 10 Symbols \{0,1,2,3,4,5,6,7,8,9\}, Base or Radix is 10
- \(136.25 = 1 \times 10^2 + 3 \times 10^1 + 6 \times 10^0 + 2 \times 10^{-1} + 3 \times 10^{-2}\)
Positional Number Systems (General)

<table>
<thead>
<tr>
<th>Decimal Numbers:</th>
</tr>
</thead>
<tbody>
<tr>
<td>✤ 10 Symbols {0,1,2,3,4,5,6,7,8,9}, Base or Radix is 10</td>
</tr>
<tr>
<td>✤ 136.25 = 1 \times 10^2 + 3 \times 10^1 + 6 \times 10^0 + 2 \times 10^{-1} + 3 \times 10^{-2}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Binary Numbers:</th>
</tr>
</thead>
<tbody>
<tr>
<td>✤ 2 Symbols {0,1}, Base or Radix is 2</td>
</tr>
<tr>
<td>✤ 101.01 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}</td>
</tr>
</tbody>
</table>
# Positional Number Systems (General)

## Decimal Numbers:
- 10 Symbols \{0,1,2,3,4,5,6,7,8,9\}, Base or Radix is 10
- $136.25 = 1 \times 10^2 + 3 \times 10^1 + 6 \times 10^0 + 2 \times 10^{-1} + 5 \times 10^{-2}$

## Binary Numbers:
- 2 Symbols \{0,1\}, Base or Radix is 2
- $101.01 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$

## Octal Numbers:
- 8 Symbols \{0,1,2,3,4,5,6,7\}, Base or Radix is 8
- $621.03 = 6 \times 8^2 + 2 \times 8^1 + 1 \times 8^0 + 0 \times 8^{-1} + 3 \times 8^{-2}$
Positional Number Systems (General)

Decimal Numbers:
- 10 Symbols {0,1,2,3,4,5,6,7,8,9}, Base or Radix is 10
- $136.25 = 1 \times 10^2 + 3 \times 10^1 + 6 \times 10^0 + 2 \times 10^{-1} + 3 \times 10^{-2}$

Binary Numbers:
- 2 Symbols {0,1}, Base or Radix is 2
- $101.01 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$

Octal Numbers:
- 8 Symbols {0,1,2,3,4,5,6,7}, Base or Radix is 8
- $621.03 = 6 \times 8^2 + 2 \times 8^1 + 1 \times 8^0 + 0 \times 8^{-1} + 3 \times 8^{-2}$

Hexadecimal Numbers:
- 16 Symbols {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}, Base is 16
- $6AF.3C = 6 \times 16^2 + 10 \times 16^1 + 15 \times 16^0 + 3 \times 16^{-1} + 12 \times 16^{-2}$
Binary-to-Decimal Conversion

- Each digit position of a binary number has a weight
  - Some power of 2
- A binary number:
  \[ B = b_{n-1} b_{n-2} \ldots b_1 b_0 \cdot b_{-1} b_{-2} \ldots b_{-m} \]

  Corresponding value in decimal:

  \[ D = \sum_{i = -m}^{n-1} b_i 2^i \]
Examples

101011 \( \Rightarrow \) \( 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \)
\[ = 43 \]
\( (101011)_2 = (43)_{10} \)

.0101 \( \Rightarrow \) \( 0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} \)
\[ = .3125 \]
\( (.0101)_2 = (.3125)_{10} \)

101.11 \( \Rightarrow \) \( 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} \)
\[ = 5.75 \]
\( (101.11)_2 = (5.75)_{10} \)
Decimal to Binary: Integer Part

Consider the integer and fractional parts separately.

For the integer part:
- Repeatedly divide the given number by 2, and go on accumulating the remainders, until the number becomes zero.
- Arrange the remainders in reverse order.

\[
\begin{array}{|c|c|c|}
\hline
\text{Base} & \text{Numb} & \text{Rem} \\
\hline
2 & 89 & \\
2 & 44 & 1 \\
2 & 22 & 0 \\
2 & 11 & 0 \\
2 & 5 & 1 \\
2 & 2 & 1 \\
2 & 1 & 0 \\
2 & 1 & 0 \\
\hline
\end{array}
\]

(89)\text{ten} = (1011001)\text{two}

(89)_{10} = (1011001)_{2}
Decimal to Binary: Integer Part

- Consider the integer and fractional parts separately.
- For the integer part:
  - Repeatedly divide the given number by 2, and go on accumulating the remainders, until the number becomes zero.
  - Arrange the remainders in reverse order.

<table>
<thead>
<tr>
<th>Base</th>
<th>Numb</th>
<th>Rem</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>89</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>44</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

\[(89)_{10} = (1011001)_2\]

<table>
<thead>
<tr>
<th>Base</th>
<th>Numb</th>
<th>Rem</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

\[(66)_{10} = (1000010)_2\]
Decimal to Binary: Integer Part

- Consider the integer and fractional parts separately.
- For the integer part:
  - Repeatedly divide the given number by 2, and go on accumulating the remainders, until the number becomes zero.
  - Arrange the remainders in reverse order.

### Examples:

<table>
<thead>
<tr>
<th>Base</th>
<th>Numb</th>
<th>Rem</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>89</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>44</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[(89)_{10} = (1011001)_2\]

<table>
<thead>
<tr>
<th>Base</th>
<th>Numb</th>
<th>Rem</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[(66)_{10} = (1000010)_2\]

<table>
<thead>
<tr>
<th>Base</th>
<th>Numb</th>
<th>Rem</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>239</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>119</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>59</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>29</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[(239)_{10} = (11101111)_2\]
Decimal to Binary: Fraction Part

- Repeatedly multiply the given fraction by 2.
  - Accumulate the integer part (0 or 1).
  - If the integer part is 1, chop it off.
- Arrange the integer parts in the order they are obtained.

**Example: 0.634**

\[
\begin{align*}
.634 \times 2 &= 1.268 \\
.268 \times 2 &= 0.536 \\
.536 \times 2 &= 1.072 \\
.072 \times 2 &= 0.144 \\
.144 \times 2 &= 0.288 \\
\vdots \\
\end{align*}
\]

\[
(.634)_{10} = (.10100\ldots)_{2}
\]
Decimal to Binary: Fraction Part

- Repeatedly multiply the given fraction by 2.
  - Accumulate the integer part (0 or 1).
  - If the integer part is 1, chop it off.
- Arrange the integer parts in the order they are obtained.

**Example: 0.634**

\[
\begin{align*}
.634 \times 2 &= 1.268 \\
.268 \times 2 &= 0.536 \\
.536 \times 2 &= 1.072 \\
.072 \times 2 &= 0.144 \\
.144 \times 2 &= 0.288 \\
& \vdots \\
(.634)_{10} &= (.10100\ldots)_2
\end{align*}
\]

**Example: 0.0625**

\[
\begin{align*}
.0625 \times 2 &= 0.125 \\
.1250 \times 2 &= 0.250 \\
.2500 \times 2 &= 0.500 \\
.5000 \times 2 &= 1.000 \\
(.0625)_{10} &= (.0001)_{2}
\end{align*}
\]
Decimal to Binary: Fraction Part

- Repeatedly multiply the given fraction by 2.
  - Accumulate the integer part (0 or 1).
  - If the integer part is 1, chop it off.
- Arrange the integer parts in the order they are obtained.

**Example: 0.634**

\[
\begin{align*}
.634 \times 2 &= 1.268 \\
.268 \times 2 &= 0.536 \\
.536 \times 2 &= 1.072 \\
.072 \times 2 &= 0.144 \\
.144 \times 2 &= 0.288 \\
\vdots \\
(.634)_{10} &= (.10100\ldots)_{2}
\end{align*}
\]

**Example: 0.0625**

\[
\begin{align*}
.0625 \times 2 &= 0.125 \\
.125 \times 2 &= 0.250 \\
.250 \times 2 &= 0.500 \\
.500 \times 2 &= 1.000 \\
(.0625)_{10} &= (.0001)_{2}
\end{align*}
\]

\[
(37)_{10} = (100101)_{2}
\]

\[
(.0625)_{10} = (.0001)_{2}
\]

\[
(37.0625)_{10} = (100101.0001)_{2}
\]
Hexadecimal Number System

- A compact way of representing binary numbers
- 16 different symbols (radix = 16)

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>Binary 4-bit</th>
<th>Binary 8-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0000</td>
<td>0000 0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>0001</td>
<td>0000 0001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>0010</td>
<td>0000 0010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>0011</td>
<td>0000 0011</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>0100</td>
<td>0000 0100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>0101</td>
<td>0000 0101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>0110</td>
<td>0000 0110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>0111</td>
<td>0000 0111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>1000</td>
<td>0001 0000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>1001</td>
<td>0001 0001</td>
</tr>
<tr>
<td>A</td>
<td>1010</td>
<td>1010</td>
<td>0001 0010</td>
</tr>
<tr>
<td>B</td>
<td>1011</td>
<td>1011</td>
<td>0001 0011</td>
</tr>
<tr>
<td>C</td>
<td>1100</td>
<td>1100</td>
<td>0001 0100</td>
</tr>
<tr>
<td>D</td>
<td>1101</td>
<td>1101</td>
<td>0001 0101</td>
</tr>
<tr>
<td>E</td>
<td>1110</td>
<td>1110</td>
<td>0001 0110</td>
</tr>
<tr>
<td>F</td>
<td>1111</td>
<td>1111</td>
<td>0001 0111</td>
</tr>
</tbody>
</table>
Binary-to-Hexadecimal Conversion

- For the integer part,
  - Scan the binary number from right to left
  - Translate each group of four bits into the corresponding hexadecimal digit
    - Add *leading* zeros if necessary

- For the fractional part,
  - Scan the binary number from left to right
  - Translate each group of four bits into the corresponding hexadecimal digit
    - Add *trailing* zeros if necessary
Example

1. \((1011\ 0100\ 0011)_2 = (B43)_{16}\)
2. \((10\ 1010\ 0001)_2 = (2A1)_{16}\)
3. \((.1000\ 010)_2 = (.84)_{16}\)
4. \((101\ .\ 0101\ 111)_2 = (5.5E)_{16}\)
Hexadecimal-to-Binary Conversion

- Translate every hexadecimal digit into its 4-bit binary equivalent

- Examples:
  
  \[(3A5)_{16} = (0011 1010 0101)_2\]
  
  \[(12.3D)_{16} = (0001 0010 . 0011 1101)_2\]
  
  \[(1.8)_{16} = (0001 . 1000)_2\]
Number Representation

Part II
Unsigned Binary Numbers

- An n-bit binary number
  \[ B = b_{n-1}b_{n-2} \ldots b_2b_1b_0 \]
  - \(2^n\) distinct combinations are possible, 0 to \(2^n-1\).
- For example, for \(n = 3\), there are 8 distinct combinations
  - 000, 001, 010, 011, 100, 101, 110, 111
- Range of numbers that can be represented
  - \(n=8\)  \(\rightarrow\)  0 to \(2^8-1\) (255)
  - \(n=16\) \(\rightarrow\)  0 to \(2^{16}-1\) (65535)
  - \(n=32\) \(\rightarrow\)  0 to \(2^{32}-1\) (4294967295)
Signed Integer Representation

- Many of the numerical data items that are used in a program are signed (positive or negative)
  - Question:: How to represent sign?

- Three possible approaches:
  - Sign-magnitude representation
  - One’s complement representation
  - Two’s complement representation
Sign-magnitude Representation

- For an n-bit number representation
  - The most significant bit (MSB) indicates sign
    - 0 → positive
    - 1 → negative
  - The remaining n-1 bits represent magnitude
Contd.

- Range of numbers that can be represented:
  - Maximum :: + \((2^{n-1} - 1)\)
  - Minimum :: \(- (2^{n-1} - 1)\)

- A problem:
  - Two different representations of zero
    - +0 \rightarrow 0 000\ldots0
    - -0 \rightarrow 1 000\ldots0
One’s Complement Representation

- Basic idea:
  - Positive numbers are represented exactly as in sign-magnitude form
  - Negative numbers are represented in 1’s complement form
- How to compute the 1’s complement of a number?
  - Complement every bit of the number (1→0 and 0→1)
  - MSB will indicate the sign of the number
    - 0 → positive
    - 1 → negative
Example: n=4

| 0000 | → | +0   | 1000 | → | -7   |
| 0001 | → | +1   | 1001 | → | -6   |
| 0010 | → | +2   | 1010 | → | -5   |
| 0011 | → | +3   | 1011 | → | -4   |
| 0100 | → | +4   | 1100 | → | -3   |
| 0101 | → | +5   | 1101 | → | -2   |
| 0110 | → | +6   | 1110 | → | -1   |
| 0111 | → | +7   | 1111 | → | -0   |

To find the representation of, say, -4, first note that

\[ +4 = 0100 \]

\[ -4 = \text{1’s complement of } 0100 = 1011 \]
Range of numbers that can be represented:

- Maximum :: $+ (2^{n-1} - 1)$
- Minimum :: $- (2^{n-1} - 1)$

A problem:

- Two different representations of zero.
  - $+0 \rightarrow 0000\ldots.0$
  - $-0 \rightarrow 1111\ldots.1$

Advantage of 1’s complement representation

- Subtraction can be done using addition
- Leads to substantial saving in circuitry
Two’s Complement Representation

- Basic idea:
  - Positive numbers are represented exactly as in sign-magnitude form
  - Negative numbers are represented in 2’s complement form

- How to compute the 2’s complement of a number?
  - Complement every bit of the number (1→0 and 0→1), and then add one to the resulting number
  - MSB will indicate the sign of the number
    - 0 → positive
    - 1 → negative
Example: n=4

<table>
<thead>
<tr>
<th>Binary</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>+0</td>
</tr>
<tr>
<td>0001</td>
<td>+1</td>
</tr>
<tr>
<td>0010</td>
<td>+2</td>
</tr>
<tr>
<td>0011</td>
<td>+3</td>
</tr>
<tr>
<td>0100</td>
<td>+4</td>
</tr>
<tr>
<td>0101</td>
<td>+5</td>
</tr>
<tr>
<td>0110</td>
<td>+6</td>
</tr>
<tr>
<td>0111</td>
<td>+7</td>
</tr>
<tr>
<td>1000</td>
<td>-8</td>
</tr>
<tr>
<td>1001</td>
<td>-7</td>
</tr>
<tr>
<td>1010</td>
<td>-6</td>
</tr>
<tr>
<td>1011</td>
<td>-5</td>
</tr>
<tr>
<td>1100</td>
<td>-4</td>
</tr>
<tr>
<td>1101</td>
<td>-3</td>
</tr>
<tr>
<td>1110</td>
<td>-2</td>
</tr>
<tr>
<td>1111</td>
<td>-1</td>
</tr>
</tbody>
</table>

To find the representation of, say, -4, first note that

\[ +4 = 0100 \]
\[ -4 = 2\text{'s complement of } 0100 = 1011+1 = 1100 \]

Rule: Value = \(-\) msb\(\times2^{(n-1)} \)+ [unsigned value of rest]

Example: \(0110 = 0 + 6 = 6 \quad 1110 = -8 + 6 = -2 \)
Contd.

- Range of numbers that can be represented:
  - Maximum :: $+ (2^{n-1} - 1)$
  - Minimum :: $- 2^{n-1}$

- Advantage:
  - Unique representation of zero
  - Subtraction can be done using addition
  - Leads to substantial saving in circuitry

- Almost all computers today use the 2’s complement representation for storing negative numbers
Adding Binary Numbers

- **Basic Rules:**
  - 0+0=0
  - 0+1=1
  - 1+0=1
  - 1+1=0 (carry 1)

- **Example:**
  
  \[
  \begin{array}{c}
  01101001 \\
  +00110100 \\
  \hline
  10011101
  \end{array}
  \]
Subtraction Using Addition: 1’s Complement

How to compute \( A - B \) ?

- Compute the 1’s complement of \( B \) (say, \( B_1 \)).
- Compute \( R = A + B_1 \)
- If the carry obtained after addition is ‘1’
  - Add the carry back to \( R \) (called end-around carry)
  - That is, \( R = R + 1 \)
  - The result is a positive number
- Else
  - The result is negative, and is in 1’s complement form
Example 1: $6 - 2$

1’s complement of 2 = 1101

6 :: 0110
-2 :: 1101

\[
\begin{array}{c}
10011 \\
1 \\
\hline
0100
\end{array}
\]

A
B_1
R

$\Rightarrow +4$

Assume 4-bit representations
Since there is a carry, it is added back to the result
The result is positive

End-around carry
Example 2: 3 − 5

1’s complement of 5 = 1010

3 :: 0011
-5 :: 1010

1101

Assume 4-bit representations
Since there is no carry, the result is negative
1101 is the 1’s complement of 0010, that is, it represents −2
Subtraction Using Addition: 2’s Complement

How to compute $A - B$?

- Compute the 2’s complement of $B$ (say, $B_2$)
- Compute $R = A + B_2$
- If the carry obtained after addition is ‘1’
  - Ignore the carry
  - The result is a positive number

Else
- The result is negative, and is in 2’s complement form
Example 1: $6 - 2$

2’s complement of 2 = $1101 + 1 = 1110$

6 :: 0110  
-2 :: 1110  

Assume 4-bit representations

Presence of carry indicates that the result is positive

No need to add the end-around carry like in 1’s complement

Ignore carry

+4
Example 2: 3 – 5

2’s complement of 5 = 1010 + 1 = 1011

A

B₂

R

3 :: 0011

-5 :: 1011

1110

Assume 4-bit representations
Since there is no carry, the result is negative
1110 is the 2’s complement of 0010, that is, it represents –2

-2
2’s complement arithmetic: More Examples

- Example 1: 18 - 11 = ?
- 18 is represented as 00010010
- 11 is represented as 00001011
  - 1’s complement of 11 is 11110100
  - 2’s complement of 11 is 11110101
- Add 18 to 2’s complement of 11

\[
\begin{array}{c}
00010010 \\
+ 11110101 \\
\hline
00000111 \text{ (with a carry of 1 which is ignored)}
\end{array}
\]

00000111 is 7
Example 2: 7 - 9 = ?

7 is represented as 00000111

9 is represented as 00001001

- 1’s complement of 9 is 11110110
- 2’s complement of 9 is 11110111
- Add 7 to 2’s complement of 9

\[
\begin{array}{c}
00000111 \\
+ 11110111 \\
\hline
11111110 \text{ (with a carry of 0 which is ignored)}
\end{array}
\]
Number Representation

Part III
Overflow/Underflow:

Adding two +ve (-ve) numbers should not produce a –ve (+ve) number. If it does, overflow (underflow) occurs
Overflow/Underflow:

Adding two +ve (-ve) numbers should not produce a –ve (+ve) number. If it does, overflow (underflow) occurs.

Another equivalent condition: carry in and carry out from Most Significant Bit (MSB) differ.
Overflow/Underflow:

Adding two +ve (-ve) numbers should not produce a –ve (+ve) number. If it does, overflow (underflow) occurs.

Another equivalent condition: carry in and carry out from Most Significant Bit (MSB) differ.

\[
\begin{array}{c}
(64) \ 01000000 \\
(4) \ 00000100 \\
\hline
\hline
(68) \ 01000100 \\
\end{array}
\]

carry (out)(in) 0 0
**Overflow/Underflow:**

Adding two +ve (-ve) numbers should not produce a –ve (+ve) number. If it does, overflow (underflow) occurs.

Another equivalent condition: carry in and carry out from Most Significant Bit (MSB) differ.

<table>
<thead>
<tr>
<th>(64) 01000000</th>
<th>(64) 01000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 4) 00000100</td>
<td>(96) 01100000</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------</td>
</tr>
<tr>
<td>(68) 01000100</td>
<td>(-96) 10100000</td>
</tr>
</tbody>
</table>

- carry (out)(in) 0 0
- carry out in 0 1
- differ: overflow
Floating-point Numbers

- The representations discussed so far applies only to integers
  - Cannot represent numbers with fractional parts
- We can assume a decimal point before a signed number
  - In that case, pure fractions (without integer parts) can be represented
- We can also assume the decimal point somewhere in between
  - This lacks flexibility
  - Very large and very small numbers cannot be represented
Representation of Floating-Point Numbers

- A floating-point number $F$ is represented by a doublet $<M,E>$:
  \[ F = M \times B^E \]
  - $B \rightarrow$ exponent base (usually 2)
  - $M \rightarrow$ mantissa
  - $E \rightarrow$ exponent

  - $M$ is usually represented in 2’s complement form, with an implied binary point before it

- For example,
  - In decimal, $0.235 \times 10^6$
  - In binary, $0.101011 \times 2^{0110}$
Example :: 32-bit representation

- M represents a 2’s complement fraction
  \[ 1 > M > -1 \]
- E represents the exponent (in 2’s complement form)
  \[ 127 > E > -128 \]

Points to note:
- The number of significant digits depends on the number of bits in M
  - 6 significant digits for 24-bit mantissa
- The range of the number depends on the number of bits in E
  - 10^{38} to 10^{-38} for 8-bit exponent.
- Sign bit is added in front to represent both +ve and –ve numbers
- The representation shown for floating-point numbers as shown is just for illustration
- The actual representation is a little more complex, we will not do here
  - Example: IEEE 754 Floating Point format