Number Representation

CS10001: Programming & Data Structures

Pallab Dasgupta
Professor, Dept. of Computer Sc. & Engg.,
Indian Institute of Technology Kharagpur
Topics to be Discussed

• How are numeric data items actually stored in computer memory?
• How much space (memory locations) is allocated for each type of data?
  – int, float, char, etc.
• How are characters and strings stored in memory?
Number System :: The Basics

- 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:**
  \[
  110 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\
  101.01 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}
  \]
Binary-to-Decimal Conversion

• Each digit position of a binary number has a weight.
  - Some power of 2.

• A binary number:

\[ B = b_{n-1} b_{n-2} \ldots b_1 b_0 \cdot b_4 b_2 \ldots b_m \]

**Corresponding value in decimal:**

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

1. $101011 \rightarrow 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
   
   \[= 43\]

   \[(101011)_2 = (43)_{10}\]

2. $.0101 \rightarrow 0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4}$

   \[= .3125\]

   \[(.0101)_2 = (.3125)_{10}\]

3. $101.11 \rightarrow 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$

   \[5.75\]

   \[(101.11)_2 = (5.75)_{10}\]
Decimal-to-Binary Conversion

- 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*.
- For the fractional 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 1 :: 239

\[(239)_{10} = (11101111)_2\]
Example 2 :: 64

\[ (64)_{10} = (1000000)_{2} \]
Example 3 :: .634

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

\[(.634)_{10} = (.10100\ldots)_{2}\]
Example 4 :: 37.0625

(37)_{10} = (100101)_{2} \\
(.0625)_{10} = (.0001)_{2} \\
\therefore (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>
</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>
<tr>
<td>A</td>
<td>1010</td>
</tr>
<tr>
<td>B</td>
<td>1011</td>
</tr>
<tr>
<td>C</td>
<td>1100</td>
</tr>
<tr>
<td>D</td>
<td>1101</td>
</tr>
<tr>
<td>E</td>
<td>1110</td>
</tr>
<tr>
<td>F</td>
<td>1111</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:

\[(3A5)_{16} = (0011 1010 0101)_{2}\]
\[(12.3D)_{16} = (0001 0010 . 0011 1101)_{2}\]
\[(1.8)_{16} = (0001 . 1000)_{2}\]
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.
• **Range of numbers that can be represented:**
  
  Maximum :: $+ (2^{n-1} - 1)$
  
  Minimum :: $- (2^{n-1} - 1)$

• **A problem:**
  
  Two different representations of zero.
  
  $+0 \rightarrow 0 \, 0 \, 0 \, 0 \, ... \, 0$
  
  $-0 \rightarrow 1 \, 0 \, 0 \, 0 \, ... \, 0$
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

<table>
<thead>
<tr>
<th>Binary</th>
<th>Integer</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>-7</td>
</tr>
<tr>
<td>1001</td>
<td>-6</td>
</tr>
<tr>
<td>1010</td>
<td>-5</td>
</tr>
<tr>
<td>1011</td>
<td>-4</td>
</tr>
<tr>
<td>1100</td>
<td>-3</td>
</tr>
<tr>
<td>1101</td>
<td>-2</td>
</tr>
<tr>
<td>1110</td>
<td>-1</td>
</tr>
<tr>
<td>1111</td>
<td>-0</td>
</tr>
</tbody>
</table>

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 → 0 000….0

  −0 → 1 111….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>Decimal</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

\[ +4 = 0100 \]

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

- **short int**
  - 16 bits $\Rightarrow$ $+ (2^{15}-1)$ to $-2^{15}$

- **int**
  - 32 bits $\Rightarrow$ $+ (2^{31}-1)$ to $-2^{31}$

- **long int**
  - 64 bits $\Rightarrow$ $+ (2^{63}-1)$ to $-2^{63}$
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

\[ \begin{array}{c}
6 \\
-2
\end{array} \]

A

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

B₁

\[ \begin{array}{c}
0100
\end{array} \]

R

End-around carry

Assume 4-bit representations.

Since there is a carry, it is added back to the result.

The result is positive.
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

\[ \underline{10100} \]

A
B_2
R

Ignore carry
+4

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.
Example 2 :: 3 − 5

2’s complement of 5 = 1010 + 1 = 1011

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.
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 2’s complement 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.
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 decimal 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.
A Warning

- The representation for floating-point numbers as shown is just for illustration.

- The actual representation is a little more complex.

- In C:
  - float :: 32-bit representation
  - double :: 64-bit representation
Many applications have to deal with non-numerical data.
- Characters and strings.
- There must be a standard mechanism to represent alphanumeric and other characters in memory.

Three standards in use:
- Extended Binary Coded Decimal Interchange Code (EBCDIC)
  - Used in older IBM machines.
- American Standard Code for Information Interchange (ASCII)
  - Most widely used today.
- UNICODE
  - Used to represent all international characters.
  - Used by Java.
ASCII Code

• Each individual character is numerically encoded into a unique 7-bit binary code.
  – A total of $2^7$ or 128 different characters.
  – A character is normally encoded in a byte (8 bits), with the MSB not been used.

• The binary encoding of the characters follow a regular ordering.
  – Digits are ordered consecutively in their proper numerical sequence (0 to 9).
  – Letters (uppercase and lowercase) are arranged consecutively in their proper alphabetic order.
### Some Common ASCII Codes

<table>
<thead>
<tr>
<th>Character</th>
<th>Decimal (D)</th>
<th>Hexadecimal (H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘A’</td>
<td>65</td>
<td>41</td>
</tr>
<tr>
<td>‘B’</td>
<td>66</td>
<td>42</td>
</tr>
<tr>
<td>‘Z’</td>
<td>90</td>
<td>5A</td>
</tr>
<tr>
<td>‘a’</td>
<td>97</td>
<td>61</td>
</tr>
<tr>
<td>‘b’</td>
<td>98</td>
<td>62</td>
</tr>
<tr>
<td>‘Z’</td>
<td>122</td>
<td>7A</td>
</tr>
<tr>
<td>‘0’</td>
<td>48</td>
<td>30</td>
</tr>
<tr>
<td>‘1’</td>
<td>49</td>
<td>31</td>
</tr>
<tr>
<td>‘9’</td>
<td>57</td>
<td>39</td>
</tr>
<tr>
<td>‘(’</td>
<td>40</td>
<td>28</td>
</tr>
<tr>
<td>‘+’</td>
<td>43</td>
<td>2B</td>
</tr>
<tr>
<td>‘?’</td>
<td>63</td>
<td>3F</td>
</tr>
<tr>
<td>‘\n’</td>
<td>10</td>
<td>0A</td>
</tr>
<tr>
<td>‘\0’</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>
Character Strings

- Two ways of representing a sequence of characters in memory.
  - The first location contains the number of characters in the string, followed by the actual characters.
    - The characters follow one another, and is terminated by a special delimiter.
String Representation in C

- In C, the second approach is used.
  - The ‘\0’ character is used as the string delimiter.

- Example:
  "Hello"  ➞  "Hello" \0

- A null string "" occupies one byte in memory.
  - Only the ‘\0’ character.