Data Representation: bits, bytes, integers

Yipeng Huang

Rutgers University

February 11, 2021
Table of contents

Announcements
   Programming assignments
   Looking ahead

Bits and bytes
   Why binary
   Decimal, binary, octal, and hexadecimal
   Bitwise operations
   Representing characters

Integers and basic arithmetic
   Representing negative and signed integers
Programming assignments

Programming assignment 1

▶ Due date extended to: 11:59pm Monday, February 15.
▶ Take good advantage of this opportunity.
▶ Familiarity with C is a vital foundation for this class and future classes.

Programming assignment 2

▶ Released later today Thursday, February 11.
▶ Due after two weeks: February 25.
▶ Same techniques in programming C.
▶ Review of graph algorithms.
Looking ahead

Lecture plan

1. Today, Thursday, 2/11: Data representation of integers.
2. Tuesday, 2/16: Data representation of floating point numbers.
3. Thursday, 2/18: Data representation of floating point numbers.

Reading assignment

Table of contents

Announcements
  Programming assignments
  Looking ahead

Bits and bytes
  Why binary
  Decimal, binary, octal, and hexadecimal
  Bitwise operations
  Representing characters

Integers and basic arithmetic
  Representing negative and signed integers
Why binary

Everything is bits

- Each bit is 0 or 1
- By encoding/interpreting sets of bits in various ways
  - Computers determine what to do (instructions)
  - ... and represent and manipulate numbers, sets, strings, etc...
- Why bits? Electronic Implementation
  - Easy to store with bistable elements
  - Reliably transmitted on noisy and inaccurate wires
Why binary

Figure: Rahul Sarpeshkar. Analog Versus Digital: Extrapolating from Electronics to Neurobiology. 1998.
Why binary

Digital encodings
Each doubling of either precision or range only needs one additional bit.

Analog encodings
Each doubling of either precision or range needs doubling of either area or power.
### Decimal, binary, octal, and hexadecimal

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Octal</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0b0000</td>
<td>0o0</td>
<td>0x0</td>
</tr>
<tr>
<td>1</td>
<td>0b0001</td>
<td>0o1</td>
<td>0x1</td>
</tr>
<tr>
<td>2</td>
<td>0b0010</td>
<td>0o2</td>
<td>0x2</td>
</tr>
<tr>
<td>3</td>
<td>0b0011</td>
<td>0o3</td>
<td>0x3</td>
</tr>
<tr>
<td>4</td>
<td>0b0100</td>
<td>0o4</td>
<td>0x4</td>
</tr>
<tr>
<td>5</td>
<td>0b0101</td>
<td>0o5</td>
<td>0x5</td>
</tr>
<tr>
<td>6</td>
<td>0b0110</td>
<td>0o6</td>
<td>0x6</td>
</tr>
<tr>
<td>7</td>
<td>0b0111</td>
<td>0o7</td>
<td>0x7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Octal</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0b1000</td>
<td>0o10</td>
<td>0x8</td>
</tr>
<tr>
<td>9</td>
<td>0b1001</td>
<td>0o11</td>
<td>0x9</td>
</tr>
<tr>
<td>10</td>
<td>0b1010</td>
<td>0o12</td>
<td>0xA</td>
</tr>
<tr>
<td>11</td>
<td>0b1011</td>
<td>0o13</td>
<td>0xB</td>
</tr>
<tr>
<td>12</td>
<td>0b1100</td>
<td>0o14</td>
<td>0xC</td>
</tr>
<tr>
<td>13</td>
<td>0b1101</td>
<td>0o15</td>
<td>0xD</td>
</tr>
<tr>
<td>14</td>
<td>0b1110</td>
<td>0o16</td>
<td>0xE</td>
</tr>
<tr>
<td>15</td>
<td>0b1111</td>
<td>0o17</td>
<td>0xF</td>
</tr>
</tbody>
</table>

In C, format specifiers for printf() and fscanf():

1. decimal: `'%d'`
2. binary: none
3. octal: `'%o'`
4. hexadecimal: `'%x'`
Decimal, binary, octal, and hexadecimal

How to represent the range of unsigned char in each?

Unsigned char is one byte, 8 bits.

1. decimal: 0 to 255
2. binary: 0b0 to 0b11111111
3. octal: 0 to 0o377 (group by 3 bits)
4. hexadecimal: 0x00 to 0xFF (group by 4 bits)
Bitwise operations

Why are bitwise operations important?

- Network and UNIX settings using bit masks (e.g., umask)
- Hardware and microcontroller programming (e.g., Arduinos)
- Instruction set architecture encodings (e.g., ARM, x86)
Bitwise operations

\[ \sim : \text{bitwise NOT} \]

unsigned char \( a \) = 128

\[
\begin{align*}
  a &= 0b1000_0000 \\
  \sim a &= \sim 0b1000_0000 \\
       &= 0b0111_1111 \\
       &= 127
\end{align*}
\]
Bitwise operations

\&: bitwise AND

\begin{align*}
3 \& 1 &= 0b11 \& 0b01 \\
&= 0b01 \\
&= 1
\end{align*}

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a &amp; b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Bitwise operations

|: bitwise OR

\[
\begin{array}{c}
3 \mid 1 = 0b11 \mid 0b01 \\
= 0b11 \\
= 3 \\
\end{array}
\]

\[
\begin{array}{cccc}
a & b & a \mid b \\
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{c}
2 \mid 1 = 0b10 \mid 0b01 \\
= 0b11 \\
= 3 \\
\end{array}
\]
^: bitwise XOR

\[ 3 \land 1 = 0b11 \land 0b01 \]
\[ = 0b10 \]
\[ = 2 \]
Don’t confuse bitwise operators with logical operators

Bitwise operators

▶ ~
▶ &
▶ |
▶ ^

Logical operators

▶ !
▶ &&
▶ ||
▶ != (for bool type)
Representing characters

**USASCII code chart**

<table>
<thead>
<tr>
<th>Column</th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>NUL</td>
<td>DLE</td>
<td>SP</td>
<td>@</td>
<td>P</td>
<td>`</td>
<td>p</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
<td>DC1</td>
<td>!</td>
<td>A</td>
<td>Q</td>
<td>a</td>
<td>q</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>DC2</td>
<td>&quot;</td>
<td>B</td>
<td>R</td>
<td>b</td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>DC3</td>
<td>#</td>
<td>C</td>
<td>S</td>
<td>c</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>EOT</td>
<td>DC4</td>
<td>$</td>
<td>D</td>
<td>T</td>
<td>d</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
<td>NAK</td>
<td>%</td>
<td>E</td>
<td>U</td>
<td>e</td>
<td>u</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>SYN</td>
<td>@</td>
<td>F</td>
<td>V</td>
<td>f</td>
<td>v</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
<td>ETB</td>
<td>'</td>
<td>G</td>
<td>W</td>
<td>g</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td>CAN</td>
<td>(</td>
<td>H</td>
<td>X</td>
<td>h</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>HT</td>
<td>EM</td>
<td>)</td>
<td>I</td>
<td>Y</td>
<td>i</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>LF</td>
<td>SUB</td>
<td>*</td>
<td>J</td>
<td>Z</td>
<td>j</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>VT</td>
<td>ESC</td>
<td>+</td>
<td>K</td>
<td>[</td>
<td>k</td>
<td>(</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>FF</td>
<td>FS</td>
<td>,</td>
<td>L</td>
<td>\</td>
<td>l</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>CR</td>
<td>GS</td>
<td>-</td>
<td>M</td>
<td>]</td>
<td>m</td>
<td>}</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>SO</td>
<td>RS</td>
<td>&gt;</td>
<td>N</td>
<td>~</td>
<td>n</td>
<td>~</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>SI</td>
<td>US</td>
<td>/</td>
<td>O</td>
<td>_</td>
<td>o</td>
<td>DEL</td>
<td></td>
</tr>
</tbody>
</table>

Figure: ASCII character set. Image credit Wikimedia
Table of contents

Announcements
  Programming assignments
  Looking ahead

Bits and bytes
  Why binary
  Decimal, binary, octal, and hexadecimal
  Bitwise operations
  Representing characters

Integers and basic arithmetic
  Representing negative and signed integers
Representing negative and signed integers

Ways to represent negative numbers

1. Sign magnitude
2. 1’s complement
3. 2’s complement
Representing negative and signed integers

Sign magnitude
Flip leading bit.
Representing negative and signed integers

1’s complement
► Flip all bits
► Addition in 1’s complement is sound
► In this encoding there are 2 encodings for 0
► -0: 0b1111
► +0: 0b0000
Representing negative and signed integers

2’s complement

<table>
<thead>
<tr>
<th>signed char</th>
<th>weight in decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000001</td>
<td>1</td>
</tr>
<tr>
<td>00000010</td>
<td>2</td>
</tr>
<tr>
<td>00000100</td>
<td>4</td>
</tr>
<tr>
<td>00001000</td>
<td>8</td>
</tr>
<tr>
<td>00010000</td>
<td>16</td>
</tr>
<tr>
<td>00100000</td>
<td>32</td>
</tr>
<tr>
<td>01000000</td>
<td>64</td>
</tr>
<tr>
<td>10000000</td>
<td>-128</td>
</tr>
</tbody>
</table>

Table: Weight of each bit in a signed char type

- what is the most positive value you can represent? 127
- what is the most negative value you can represent? -128
- how to represent -1? 11111111
- how to represent -2? 11111110
Representing negative and signed integers

2’s complement

<table>
<thead>
<tr>
<th>signed char</th>
<th>weight in decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000001</td>
<td>1</td>
</tr>
<tr>
<td>00000010</td>
<td>2</td>
</tr>
<tr>
<td>00000100</td>
<td>4</td>
</tr>
<tr>
<td>00001000</td>
<td>8</td>
</tr>
<tr>
<td>00010000</td>
<td>16</td>
</tr>
<tr>
<td>00100000</td>
<td>32</td>
</tr>
<tr>
<td>01000000</td>
<td>64</td>
</tr>
<tr>
<td>10000000</td>
<td>-128</td>
</tr>
</tbody>
</table>

Table: Weight of each bit in a signed char type

- MSB: 1 for negative
- Take the 1’s complement number + 1
- Most important; good properties for digital logic
Importance of paying attention to limits of encoding

Figure: Image credit: CS:APP
Importance of paying attention to limits of encoding

Figure: Image credit: CS:APP
Importance of paying attention to limits of encoding

Two's complement addition (4-bit word)

Figure: Image credit: CS:APP