Computer Systems Principles

Data Representation - Bits and Bytes
Today’s Class

1. Learn data representation in binary and hexadecimal
   – represent negative numbers.

2. Perform operations
   – addition, subtraction, multiplication

• Announcements: Moodle Quiz 2 & HW 1 are out.
Group Task

• Form partners

• Using only the three symbols @#& represent:
  – integers 0 – 10
How do we think about numbers?

• Representing and reasoning about numbers
  • Computers store variables (data).
  • Data is typically composed of numbers and characters.
  • Want a representation that is sensible.
Decimal Number Systems

- Digits 0-9
Decimal Number Systems

- Digits 0-9
- $171_{10} = 1 \times 100 + 7 \times 10 + 1 = 171$
Decimal Number Systems

- Digits 0-9
- \(171_{10} = 1 \times 100 + 7 \times 10 + 1 = 171\)
- Every time we move to the left we’re thinking in bundles of 10 of the space to the right
Decimal Number Systems

- Digits 0-9
- $171_{10} = 1 \times 100 + 7 \times 10 + 1 = 171$
- Every time we move to the left we’re thinking in bundles of 10 of the space to the right
- Power of the base 10 system

Given four positions: $[\text{X X X X}]_{10}$ what is the largest number you can represent?
  - $9999_{10}$
Binary Digits: (BITs)
Binary Digits: (BITS)

7 6 5 4 3 2 1 0

Most significant bit  →  10001111  ←  Least significant bit

• Sequence of eight bits: byte
Binary Digits: (BITS)

Most significant bit $\rightarrow$ 10001111 $\leftarrow$ Least significant bit

$b_7b_6b_5b_4b_3b_2b_1b_0$

$b_0$ 1s place
$b_1$ 2s place
$b_2$ 4s place
$b_3$ 8s place
$b_4$ 16s place
$b_5$ 32s place
$b_6$ 64s place
$b_7$ 128s place

* Sequence of eight bits: byte*
Binary Number Systems

- Transistors: On or Off
- Optical: Light or No light
- Magnetic: Positive or Negative
Conversion: Binary to Decimal

Method:
Multiply each of the binary digits by the appropriate power of 2:
Example:
\[ 11111111_2 = 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255_{10} \]
Conversion: Decimal to Binary

Method:
We have a decimal number \((x)\):
1. Highest power of two less than or equal to the decimal number \((y)\)
2. Subtract the power of two \((y)\) from the decimal number \((x)\) as \(x = x-y\).
3. If \(x = 0\), we have our result! Else: Repeat

Example:
\[19_{10} = 16 + 2 + 1 = 00010011_2\]
Hexadecimal Representation

- Hexadecimal numbers use 16 digits: \{0-9, A-F\}
  - does not distinguish between upper and lower case!

<table>
<thead>
<tr>
<th>DEC</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEX</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
</tbody>
</table>
Hexadecimal Representation

- Hexadecimal numbers use 16 digits: {0-9, A-F}
  - does not distinguish between upper and lower case!
- $AB_{16} = A \times 16 + B \times 1$

<table>
<thead>
<tr>
<th>DEC</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEX</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
</tbody>
</table>
Hexadecimal Representation

- Hexadecimal numbers use 16 digits: {0-9, A-F}
  - does not distinguish between upper and lower case!
- AB\textsubscript{16} = A \times 16 + B \times 1
  = 10 \times 16 + 11 \times 1
  = 171
Conversion from Binary to Hexadecimal

- Splitting into groups of 4 bits each.
- If not a multiple of 4:
  - pad the number with leading zeros.
- Example:

  \[3\text{CADB3}_{16} = \text{0011 1100 1010 1101 1011 0011}_2\]

<table>
<thead>
<tr>
<th>Bin</th>
<th>0011</th>
<th>1100</th>
<th>1010</th>
<th>1101</th>
<th>1011</th>
<th>0011</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hex</td>
<td>3</td>
<td>C</td>
<td>A</td>
<td>D</td>
<td>B</td>
<td>3</td>
</tr>
</tbody>
</table>
iClicker Activity

Convert the decimal number 231 to binary and hexadecimal equivalents.

a) 1110 1111, F7
b) 1110 0110, E6
c) 1110 0111, E7
d) 0110 0111, 57

<table>
<thead>
<tr>
<th>DEC</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEX</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
</tbody>
</table>
Binary Addition

Rules

• $1+0 = 1$
• $1+1 = 10$
• $1+1+1 = 11$
## Binary Addition

### Rules

- $1+0 = 1$
- $1+1 = 10$
- $1+1+1 = 11$

<table>
<thead>
<tr>
<th>Carry in</th>
<th>Augend (A)</th>
<th>Addend (B)</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 0 1 1 1</td>
<td>1 1 1 0</td>
<td>1 1 0 0 1</td>
</tr>
</tbody>
</table>

### Example:

- **Augend (A):** 1 0 1 1 1
- **Addend (B):** 1 1 1 0
- **Sum:** 1 1 0 0 1
Binary Addition

Rules

• $1+0 = 1$
• $1+1 = 10$
• $1+1+1 = 11$

\[
\begin{array}{cccc}
1 & 0 & 1 & 1 \\
+ & 1 & 1 & 1 & 0 \\
\hline
1 & 1 & 0 & 0 & 1
\end{array}
\]

$= 1 + 2 + 8 = 11$ (dec)

$= 2 + 4 + 8 = 14$ (dec)

$= 1 + 8 + 16 = 25$ (dec)
Binary Subtraction

Rules

- 1-0 = 1
- 1-1 = 0
- 0-1 = 1 (borrow 1)
- 1-1 = 0

\[
\begin{align*}
1 & \quad 0 & \quad 1 & \quad 1 & \quad \text{= 1 + 2 + 8 = 11 (dec)} \\
- & \quad 0 & \quad 1 & \quad 1 & \quad 0 & \quad \text{= 2 + 4 = 6 (dec)} \\
\hline
0 & \quad 1 & \quad 0 & \quad 1 & \quad \text{= 1 + 4 = 5 (dec)}
\end{align*}
\]
Binary Multiplication

Example:

\[
\begin{array}{c}
  & 1 & 1 \\
\times & 1 & 3 \\
\hline
  & 3 & 3 \\
\end{array}
\]

\[
\begin{array}{c}
  + & 1 & 1 \\
\hline
  1 & 4 & 3 \\
\end{array}
\]

- \(0 \times 0 = 0\)
- \(1 \times 0 = 0\)
- \(1 \times 1 = 1\)
Binary Multiplication

Example:

\[
\begin{array}{cccc}
1 & 1 \\
\times & 1 & 3 \\
\hline
 & 3 & 3 \\
+ & 1 & 1 & - \\
\hline
 & 1 & 4 & 3 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 0 & 1 & 1 \\
\times & 1 & 1 & 0 & 1 \\
\hline
 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & - \\
1 & 1 & 0 & 1 & 1 & - & - \\
+ & 1 & 1 & 0 & 1 & 1 & - & - & - \\
\hline
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

• \(0 \times 0 = 0\)
• \(1 \times 0 = 0\)
• \(1 \times 1 = 1\)
Binary Multiplication

Example:

\[
\begin{array}{c}
1 \\
1 \\
\times \\
1 \\
\hline
3 \\
3 \\
+ \\
1 \\
1 \\
\hline
1 \\
4 \\
3
\end{array}
\]

\[
\begin{array}{cccc}
1 & 0 & 1 & 1 \\
\times & 1 & 1 & 0 & 1 \\
\hline
1 & 0 & 1 & 1 \\
\downarrow & 0 & 0 & 0 & 0 & - \\
\downarrow & 1 & 1 & 0 & 1 & 1 & - & - \\
\downarrow + & 1 & 1 & 0 & 1 & 1 & - & - \\
\hline
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1
\end{array}
\]

\[
= 128 + 8 + 4 + 2 + 1 = 143
\]

- \(0 \times 0 = 0\)
- \(1 \times 0 = 0\)
- \(1 \times 1 = 1\)
Binary Multiplication

Example:

Since we always multiply by either 0 or 1, the partial products are always either 0000 or the multiplicand (in this example: 1011).

\[
\begin{array}{cccc}
1 & 0 & 1 & 1 \\
\times & 1 & 1 & 0 & 1 \\
\hline
1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & -
\end{array}
\]

\[
\begin{array}{cccc}
1 & 1 & 0 & 1 & 1 & - & - \\
+ & 1 & 1 & 0 & 1 & 1 & - & - \\
\hline
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1
\end{array}
\]

- \(0 \times 0 = 0\)
- \(1 \times 0 = 0\)
- \(1 \times 1 = 1\)

\[
\begin{align*}
1011 & = 8 + 2 + 1 = 11 \\
1101 & = 8 + 4 + 1 = 13 \\
1100000 & = 128 + 8 + 4 + 1 = 141
\end{align*}
\]
iClicker Activity

Multiply the two numbers: $1101 \times 1001$

a) 110 1001
b) 100 1111
c) 110 1101
d) 111 0101
iClicker Activity

- Largest binary integer that can be stored in 3 bits?

a) 001  
b) 100  
c) 111  
d) None of these

What about the smallest?
Unsigned Binary representation

• Unsigned binary:

Most significant bit $\rightarrow \underline{10001111}$ Least significant bit

Representation:

\[ 1 \times 2^7 + 0 \times 2^6 + \ldots + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \]
\[ 128 + 0 + 8 + 4 + 2 + 1 \]
\[ 10001111 = 143 \]

• We add/subtract/multiply using the normal rules that we use for decimal addition/subtraction/multiplication
Signed binary representation

• Sign bit
  – Left-most bit: $b_7b_6b_5b_4b_3b_2b_1b_0$
  – Also called the most significant bit
Negative Binary: Sign-magnitude

To get the negative number, set the sign bit to 1 and leave all the other bits unchanged.

<table>
<thead>
<tr>
<th>81</th>
<th>0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-81</td>
<td>1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1</td>
</tr>
</tbody>
</table>

Called sign-magnitude representation: sign bit + value

<table>
<thead>
<tr>
<th>0</th>
<th>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

Two zeroes! Not fun ....
More difficulties ...

- It is not easy to add, subtract, or compare numbers in sign-magnitude format

5 - 2 = 5 + (-2) = 0101 + 1010 = 1111 = -7
More difficulties ...

- It is not easy to add, subtract, or compare numbers in sign-magnitude format
  - $5 - 2 = 5 + (-2) = 0101 + 1010 = 1111 = -7$

- Values do not fall in a natural order as compared with unsigned numbers
An alternative: Ones Complement

To get the negative of a number, invert or complement all the bits.

Complement = (~)
- \(^1 = 0\) and \(^0 = 1\)

<table>
<thead>
<tr>
<th>81</th>
<th>0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-81</td>
<td>1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 0</td>
</tr>
</tbody>
</table>
An alternative: Ones Complement

• This solves the “unnatural order” problem
• But we still have two zeroes!
Two’s Complement

To get the negative of a number, invert all the bits and then add 1. If the addition causes a carry bit past the most significant bit, discard the high carry.

81 = \[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\]

\[\sim 81 = 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ +1\]

\[-81 = 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 1\]
Two’s Complement

Do we have a unique representation of the value zero? Yes!

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>~0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

+1

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Two’s Complement: Advantages

✓ Unique representation of zero
✓ Exactly same method for addition, multiplication, etc. as unsigned integers except throw away the high carry (high borrow for subtraction).
Two’s Complement: Addition, Subtraction

45 – 14 = 45 + (-14) = 31

45 = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1

-14 = 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

31 = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1

ignore high carry 1!
Two’s Complement: Advantages

• This solves the “unnatural order” problem
Conversion: Binary to Decimal

Method:
Multiply each of the binary digits by the appropriate power of 2:
Example:
$11111111_2 = 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255_{10}$
Two’s complement: Conversion from Binary to Decimal

Method:
Multiply each of the binary digits by the appropriate power of 2:
b-bit word \( x \) in Two’s complement
1) For bit \( 0 \leq i \leq b-2 \), multiply \( 2^i \)
2) For bit \( b-1 \), multiply \( -2^{b-1} \)

Example:
unsigned \( 11111111_2 = 128+64+32+16+8+4+2+1 = 255_{10} \)
signed \( 11111111_2 = -128+64+32+16+8+4+2+1 = -1_{10} \)
<table>
<thead>
<tr>
<th>Signed</th>
<th>Bits</th>
<th>Unsigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>-8</td>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>-7</td>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>-6</td>
<td>1010</td>
<td>10</td>
</tr>
<tr>
<td>-5</td>
<td>1011</td>
<td>11</td>
</tr>
<tr>
<td>-4</td>
<td>1100</td>
<td>12</td>
</tr>
<tr>
<td>-3</td>
<td>1101</td>
<td>13</td>
</tr>
<tr>
<td>-2</td>
<td>1110</td>
<td>14</td>
</tr>
<tr>
<td>-1</td>
<td>1111</td>
<td>15</td>
</tr>
</tbody>
</table>
Integer Value Range

• Representing negative and positive numbers in $b$ bits:
  – Unsigned: $0$ to $2^b-1 = 00...00$ to $11...11$
  – Signed: $-2^{(b-1)}$ to $2^{(b-1)} - 1 = 100..00$ to $011...11$
Integer Value Range

• Representing negative and positive numbers in \( b \) bits:
  – Unsigned: \( 0 \) to \( 2^b - 1 \) = 00…00 to 11…11
  – Signed: \( -2^{(b-1)} \) to \( 2^{(b-1)} - 1 \) = 100..00 to 011…11

• Example: range for 8 bits is:
  – Unsigned: \( 0 \) to \( 2^8 - 1 \) = 0 .... 255
  – Signed: \( -2^7 \) to \( 2^7 - 1 \) = -128 .... 127
i-clicker question

• What is the range of a signed 3-bit number?
  A. $-2^1$ to $+2^1-1$
  B. $-2^2$ to $+2^2-1$
  C. $-2^3$ to $+2^3-1$
  D. $-2^4$ to $+2^4-1$
Two’s Complement Overflow & Underflow

- **Overflow** is caused by a value near the **upper limit** of the range, while **an underflow** is caused by values near the **lower limit** of the range.
Overflow: Example

Consider the 8-bit two’s complement addition:

\[
\begin{array}{ccccccccccc}
1 & 2 & 7 & & & & & & & & \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & & \\
+1 & & & & & & & & & & \\
\hline
1 & 2 & 8 & & & & & & & & \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

• The result should be +128, but the leftmost bit is 1, therefore the result is -128!
• This is an overflow: an arithmetic operation that should be positive gives a negative result.
Consider the 8-bit two’s complement addition:

\[
\begin{array}{cccccccccccc}
-1 & 2 & 8 & & & & & & & & & & \\
-1 & & & & & & & & & & & & 1 \\
-1 & 2 & 9 & & & & & & & & & & \\
\end{array}
\]

• The result should be -129, but the leftmost bit is 0, therefore the result is +127!
• This is an underflow: as an arithmetic operation that should be negative gives a positive result.
iClicker Activity

• What is the result of the following 8-bit two’s complement addition 1000 0000 - 1?

a) 0111 1111
b) 1111 1111
c) 0000 0000
d) 0000 0001

Hint: Try converting it back to decimal and compare!
Next Class

• Lets represent Binary operations in C!
• Overflow conditions
  – How do we represent overflow?
  – What’s the impact of an overflow?
  – How do we mitigate it?
• Readings posted on website