Integer Representation

Bits, binary numbers, and bytes
Fixed-width representation of integers: unsigned and signed
Modular arithmetic and overflow

Positional number representation

- Base determines:
  - Each position holds a digit.
  - Represented value =

binary = base 2

<table>
<thead>
<tr>
<th>2</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

(100, 10^2), (10, 10^1), (1, 10^0)

= 2 \times 10^2 + 4 \times 10^1 + 0 \times 10^0

Powers of 2:
Memorize up to $\geq 2^{10}$ (in base ten)

When ambiguous, subscript with base:

101_{10} Dalmatians (movie) 101_{ten}

101_2 - Second Rule (folk wisdom for food safety) 101_{two}
Show powers, strategies.

**conversion and arithmetic**

\[ 19_{10} = ?_2 \]
\[ 1001_2 = ?_{10} \]

\[ 240_{10} = ?_2 \]
\[ 11010011_2 = ?_{10} \]

\[ 101_2 + 1011_2 = ?_2 \]
\[ 1001011_2 \times 2_{10} = ?_2 \]

---

**byte = 8 bits**

*a.k.a. octet*

**Smallest unit of data**  
*used by a typical modern computer*

**Binary**  
\[ 00000000_2 \rightarrow 11111111_2 \]

**Decimal**  
\[ 00_{10} \rightarrow 255_{10} \]

**Hexadecimal**  
\[ 00_{16} \rightarrow FF_{16} \]

Programmer’s hex notation (C, etc.):  
\[ 0xB4 = B4_{16} = B4_{\text{hex}} \]

Octal (base 8) also useful.

**Why do 240 students often confuse Halloween and Christmas?**

---

**word**  
\[ /\text{wərd}/, \text{n.} \]

Natural (fixed size) unit of data used by processor.

– Word size determines:

- **MSB**: most significant bit
- **LSB**: least significant bit

---

**fixed-size data representations**

<table>
<thead>
<tr>
<th>Java Data Type</th>
<th>C Data Type</th>
<th>(size in bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>char</td>
<td>1</td>
</tr>
<tr>
<td>byte</td>
<td>char</td>
<td>1</td>
</tr>
<tr>
<td>char</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>short</td>
<td>short int</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>float</td>
<td>float</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>long int</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>long</td>
<td>long long</td>
<td>8</td>
</tr>
<tr>
<td>long</td>
<td>long double</td>
<td>8</td>
</tr>
<tr>
<td>long</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Fixed-width integer encodings

**Unsigned** $\subseteq \mathbb{N}$, non-negative integers only

**Signed** $\subseteq \mathbb{Z}$, both negative and non-negative integers

$n$ bits offer only $2^n$ distinct values.

Terminology:
- “Most-significant” bit(s) or “high-order” bit(s)
- “Least-significant” bit(s) or “low-order” bit(s)

Unsigned addition overflows if and only if

(4-bit) unsigned integer representation

$$1011 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

$n$-bit unsigned integers:

- **Minimum** =
- **Maximum** =

modular arithmetic, overflow

$$\begin{array}{cccc}
11 & 1011 & + 2 & 0010 \\
+ & 13 & 1101 & + 5 & 0101 \\
\hline
13 & 1111 & + & 5 & 0101 \\
\end{array}$$

**Sign-magnitude**

Most-significant bit (MSB) is **sign bit**
- 0 means non-negative
- 1 means negative

Remaining bits are an unsigned magnitude

8-bit sign-magnitude:
- $00000000$ represents _____
- $01111111$ represents _____
- $1000101$ represents _____
- $10000000$ represents _____

Arithmetic?

Example:

$$4 - 3 \neq 4 + (-3)$$

Zero?
(4-bit) two's complement signed integer representation

\[
\begin{array}{cccc}
1 & 0 & 1 & 1 \\
\hline
-2^3 & 2^2 & 2^1 & 2^0
\end{array}
\]

= \(1 \times -2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0\)

4-bit two's complement integers:

minimum =

maximum =

two's complement vs. unsigned

\[
\begin{array}{cccccccc}
\_ & \_ & \cdots & \_ & \_ & \_ & \_ & \_ \\
\hline
2^{n-1} & 2^{n-2} & \cdots & 2^2 & 2^1 & 2^0 \\
-2^{n-1} & 2^{n-2} & \cdots & 2^2 & 2^1 & 2^0
\end{array}
\]

What's the difference?

\[
n\text{-bit minimum} = \quad n\text{-bit maximum} =
\]

8-bit representations

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}
\]

\[
\begin{array}{cccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \quad 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}
\]

n-bit two's complement numbers:

minimum =

maximum =

4-bit unsigned vs. 4-bit two's complement

\[
\begin{array}{cccc}
1 & 0 & 1 & 1 \\
\hline
1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 & 1 \times -2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0
\end{array}
\]

\[
11 \leftarrow \quad \text{difference} = \_\_ = 2 \rightarrow -5
\]
two’s complement addition

\[
\begin{array}{cccc}
2 & 0010 & -2 & 1110 \\
+3 & 0011 & +3 & 1101 \\
\end{array}
\]

-2 1110 2 0010
+3 0011 +3 1101

Modular Arithmetic

A few reasons two’s complement is awesome

Addition, subtraction, hardware

Sign

Negative one

Complement rules

two’s complement overflow

Addition overflows
if and only if
if and only if

\[
\begin{array}{cccc}
-1 & 1111 & \text{+2} & +0010 \\
+2 & +0010 & \text{+} & +0010 \\
6 & 0110 & +3 & +0011 \\
\end{array}
\]

Modular Arithmetic

Some CPUs/languages raise exceptions on overflow.

C and Java cruise along silently... Feature? Oops?

Another derivation

How should we represent 8-bit negatives?

- For all positive integers \( x \), \( x \) and \(-x\) should sum to zero.
- Use the standard addition algorithm.

\[
\begin{array}{cccc}
00000001 & 00000010 & 00000011 \\
+ & + & + \\
00000000 & 00000000 & 00000000 \\
\end{array}
\]

- Find a rule to represent \(-x\) where that works...