representing data with bits

bits, bytes, numbers, and notation

bit = binary digit = 0 or 1

Electronically: high voltage vs. low voltage

- 3.3V __ 0
- 2.8V __ 1
- 0.5V __
- 0.0V __

Basis of all digital representations
- ints, floats, chars, strings, booleans, etc.
- machine instructions

Modern Digital Computer
“von Neumann” model

1930s 1940s 1950s 1960s 1970s 1980s 1990s 2000s 2010s

How are data and instructions represented?

positional number representation

fancy name, familiar concept

<table>
<thead>
<tr>
<th>2</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>10^2</td>
<td>10^1</td>
<td>10^0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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

- Base determines:
  - Maximum digit (base – 1). Minimum digit is 0.
  - Weight of each position.
- Each position holds a digit.
- Represented value = sum of all position values
  - Position value = digit value \times base^{position \ index}
When ambiguous, subscript with base:

101₀ Dalmatians (movie)
101₂-Second Rule (folk wisdom about dropped food)

numbers and wires

- One wire per bit
- How many wires do we need...
- Fixed-size encodings to build hardware.

base 2 (binary)

\[
\begin{array}{cccc}
1 & 0 & 1 & 1 \\
8 & 4 & 2 & 1 \\
2^3 & 2^2 & 2^1 & 2^0 \\
3 & 2 & 1 & 0
\end{array}
\]

\[= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0\]

conversion and arithmetic

\[19_{10} = ?_2\]
One strategy:
Subtract largest power of 2 that is <= 19 from 19, and add it to 0₂.
Repeat with remaining, and sum, until 0₁₀ remains.

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

\[101₂ + 1011_2 = ?_2\]
\[1001011_2 \times 2_{10} = ?_2\]

byte = 8 bits
(a.k.a. octet)

Smallest unit of data
used by a typical modern computer

<table>
<thead>
<tr>
<th>Hex</th>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1000</td>
<td>0000000₀</td>
</tr>
<tr>
<td>1</td>
<td>1001</td>
<td>0000001₀</td>
</tr>
<tr>
<td>2</td>
<td>1010</td>
<td>0000010₀</td>
</tr>
<tr>
<td>3</td>
<td>1011</td>
<td>0000011₀</td>
</tr>
<tr>
<td>4</td>
<td>1100</td>
<td>0000100₀</td>
</tr>
<tr>
<td>5</td>
<td>1101</td>
<td>0000101₀</td>
</tr>
<tr>
<td>6</td>
<td>1110</td>
<td>0000110₀</td>
</tr>
<tr>
<td>7</td>
<td>1111</td>
<td>0000111₀</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>0010000₁₀</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>0010001₁₀</td>
</tr>
<tr>
<td>A</td>
<td>1010</td>
<td>0010010₁₀</td>
</tr>
<tr>
<td>B</td>
<td>1011</td>
<td>0010011₁₀</td>
</tr>
<tr>
<td>C</td>
<td>1100</td>
<td>0010100₁₀</td>
</tr>
<tr>
<td>D</td>
<td>1101</td>
<td>0010101₁₀</td>
</tr>
<tr>
<td>E</td>
<td>1110</td>
<td>0010110₁₀</td>
</tr>
<tr>
<td>F</td>
<td>1111</td>
<td>0010111₁₀</td>
</tr>
</tbody>
</table>

Programmer’s hex notation (C, etc.):

\[0xB4 = B₄₁₆\]

Octal (base 8) also useful.

What do you call 4 bits?

Byte = 2 hex digits!
**word** |ward|, n.

Natural unit of data used by processor.
- Fixed size (e.g. 32 bits, 64 bits)
  - depends on ISA: Instruction Set Architecture
- machine instructions usually operate on words
- word size = register size = address size

Java/C int = 4 bytes: 11,501,584

---

**C programming language**

- Invented in 1970s to help build UNIX operating system
  - OS manages hardware, C close to machine model
- Simple pieces look like Java:
  - if, while, for, local variables, assignment, etc.
- Other pieces do not:
  - no objects, no methods, no array bounds checks
  - addresses, pointers, structs, functions, weak type system
- Important language, still widely used, but many good PL ideas have come along since.

---

**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 1</td>
</tr>
<tr>
<td>byte</td>
<td>int</td>
<td>1 1</td>
</tr>
<tr>
<td>short</td>
<td>short int</td>
<td>2 2</td>
</tr>
<tr>
<td>int</td>
<td>int</td>
<td>4 4</td>
</tr>
<tr>
<td>float</td>
<td>float</td>
<td>4 4</td>
</tr>
<tr>
<td>double</td>
<td>long int</td>
<td>4 8</td>
</tr>
<tr>
<td>long</td>
<td>double</td>
<td>8 8</td>
</tr>
<tr>
<td></td>
<td>long double</td>
<td>8 16</td>
</tr>
</tbody>
</table>

*Depends on word size*

---

**boolean algebra, C notation**

<table>
<thead>
<tr>
<th>Operation</th>
<th>C Notation</th>
<th>32-bit</th>
<th>64-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND: A&amp;B</td>
<td>&amp;</td>
<td>0 0 0</td>
<td>1 0 1</td>
</tr>
<tr>
<td>OR: A</td>
<td>B</td>
<td></td>
<td>0 0 1</td>
</tr>
<tr>
<td>XOR: A^B</td>
<td>^</td>
<td>0 0 1</td>
<td>1 1 0</td>
</tr>
<tr>
<td>NOT: ~A</td>
<td>~</td>
<td>0 1</td>
<td>1 0</td>
</tr>
</tbody>
</table>
general boolean algebras
Operate on bit vectors with bitwise operators.

\[
\begin{array}{ll}
01101001 & 01101001 \\
\& 01010101 & \lor 01010101 \\
01000001 & \lnot 01010101 \\
\end{array}
\]

Laws of boolean algebra apply.
\[\text{e.g., DeMorgan's Law: } \lnot(A \lor B) = \lnot A \land \lnot B\]

How does this relate to set operations?

sets as bit vectors

Representation
w-bit vector represents subsets of \{0, \ldots, w-1\}.
a_i = 1 \equiv i \in A

\[
\begin{array}{ll}
01101001 & \{0, 3, 5, 6\} \\
\lnot 01010101 & \{0, 2, 4, 6\} \\
\end{array}
\]

Operations
\[
\begin{array}{ll}
\& & \text{Intersection} \\
\lor & \text{Union} \\
\lnot & \text{Symmetric difference} \\
\lnot & \text{Complement} \\
\end{array}
\]

\[
\begin{array}{ll}
01000001 & \{0, 6\} \\
01111101 & \{0, 2, 3, 4, 5, 6\} \\
00111100 & \{2, 3, 4, 5\} \\
10101010 & \{1, 3, 5, 7\} \\
\end{array}
\]

bitwise operations in C
\& | ^ ~ apply to any “integral” data type
long, int, short, char, unsigned

Examples (char)
\[
\begin{array}{ll}
\lnot 0x41 & = \ 0x69 \\
\lnot 0x00 & = \ 0x69 \\
0x69 & \land 0x55 & = \ 0x69 \\
0x69 & \lor 0x55 & = \ 0x69 \\
\end{array}
\]
Many bit-twiddling puzzles in first assignment

logical operations in C
\&\& || ! apply to any “integral” data type
long, int, short, char, unsigned

0 is false nonzero is true
result always 0 or 1
Early termination a.k.a. short-circuit evaluation

Examples (char)
\[
\begin{array}{ll}
! 0x41 & = \ 0x00 \\
!! 0x41 & = \ 0x41 \\
0x69 & \&\& 0x55 & = \ 0x69 \\
0x69 & || 0x55 & = \ 0x69 \\
\end{array}
\]