Overview

- Endianness
- Floating Point
Endianness
Endianness

- Refers to how individual bytes are distributed across words
  - How big is a word on MIPS?
Endianness

• Refers to how individual bytes are distributed across words
  • How big is a word on MIPS?
    • 32 bits (4 bytes)
Example

- Consider the number: 0x0A0B0C0D
- What are some different ways to store this in memory?
Example

- Consider the number: `0x0A0B0C0D`
- What are some different ways to store this in memory?

<table>
<thead>
<tr>
<th></th>
<th>Big-endian</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0A</td>
<td>0B</td>
<td>0C</td>
</tr>
<tr>
<td>Little-endian</td>
<td></td>
<td>0D</td>
<td>0C</td>
<td>0B</td>
</tr>
</tbody>
</table>

Others possible, and actually used!
Big-Endian

- Most significant bit is the leftmost bit, which has the largest value
- What might be some advantages to this?

Original value: \(0xA0B0C0D\)

| 0A | 0B | 0C | 0D |
Little-Endian

- *Least significant byte* is the leftmost byte, which has the smallest value
  - Why isn’t this the least significant *bit*?

Original value: \(0xA0B0C0D\)

| 0D | 0C | 0B | 0A |
Little-Endian

- Least significant byte is the leftmost byte, which has the smallest value
- Why isn’t this the least significant bit?
- Least significant bit is the rightmost bit of the leftmost byte

Original value: \(0x0A0B0C0D\)

| 0D | 0C | 0B | 0A |
Little-Endian

• *Least significant byte* is the leftmost byte, which has the smallest value

• What are some possible advantages of this format?

Original value: \(0x0A0B0C0D\)

| 0D | 0C | 0B | 0A |
Take-Home Point

- We need to pick an endianness, but it tends to not be that important of a choice
- The name “endianness” originally refers to which end of an egg should be cracked...pretty arbitrary
Floating Point
Floating Point

• Up until this point, you have dealt only with integers

• What about values like 2.572?
IEEE-754 Floating Point

• Idea: represent floating-point numbers in three parts:
  • A sign bit ($S$)
  • An exponent (fixed number of bits) ($E$)
  • A fraction (fixed number of bits) ($F$)

• Value of a number is the following:

$$(-1)^S \times (1 + F) \times 2^E$$
Exponent and Fraction Bits

- The number of bits is fixed in the standard
  - For a 32-bit float: 8 bits and 23 bits
  - For a 64-bit double: 11 bits and 52 bits
Issues

• Operations are much more complex than corresponding integer operations (slow)

• Not all values representable precisely (approximations)
  • Precision loss
  • Loss of mathematical properties
    • \((x + y) + z \neq x + (y + z)\)

• Errors can propagate easily
More Complexity

• The full standard is 70 pages long
• From my own research, it took someone nearly two months to implement a substantially simplified version of the standard
  • This was ahead of schedule
• Here be dragons