Word Size & Endianness
Word size

- Any given computer architecture has a “word size”.

- Word size determines the number of bits used to store a memory address (a pointer in C).

- Therefore you can $2^{\text{wordsize}}$ number of memory addresses.

- Until recently, most machines used 32 bits (4 bytes) as word size
  - Limits addresses to 4GB of total RAM

- These days, machines have 64-bit word size, actually only uses 48 bits of it for addresses
  - Potentially, could have $2^{48}$ addresses, thats a lot of memory.
  - Theoretically up to 65,000 times amount of RAM of 32-bit systems. (~260TB)
Word-oriented memory organization

- Address of a word in memory is the address of the first byte in that word.

- Consecutive word addresses differ by 4 (32-bit) or 8 (64-bit).
Byte ordering in a word

- There are two different conventions of byte ordering in a word

- **Big Endian**
  - Examples: Sun, PowerPC Mac, Internet
  - Most significant byte has lowest address

- **Little Endian**
  - Examples: x86, ARM processors running Android, iOS, Windows
  - Most significant byte has highest address

- In other words, if you have a multi-byte word, what order do the bytes appear? What “end” of the word does the MSB live at?
Variable $x$ has 4-byte value of $0x01234567$

Address given by dereferencing $x$ is $0x100$

- We can test this programmatically. See `memory/ endian.c`
Byte ordering representation in C

- Casting any pointer to unsigned char* allows is to treat the memory as a byte array.

- Using printf format specifiers
  - %p - print pointer
  - %x - print value in hexadecimal

- See memory/byte_ordering.c
Floating Point
Fractional binary numbers

- How can we represent fractional binary numbers?
- One idea: use same approach as with decimal numbers, except use powers of 2 (as opposed to 10).
- So what is $1011.101_2$?
Fractional binary numbers

- How can we represent fractional binary numbers?

- One idea: use same approach as with decimal numbers, except use powers of 2 (as opposed to 10).

- So what is \(1011.101_2\)?

\[
(1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3})
\]

\[
8 + 2 + 1 + \frac{1}{2} + \frac{1}{8}
\]

\[11.625_{10}\]
Fractional binary numbers

- How can we represent fractional binary numbers?

- One idea: use same approach as with decimal numbers, except use powers of 2 (as opposed to 10).

- So what is $1011.101_2$?

$$
(1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3})
$$

$$
8 + 2 + 1 + \frac{1}{2} + \frac{1}{8}
$$

$$
11.625_{10}
$$

- Going the other direction

  - $5 \frac{3}{4} \rightarrow 101.11_2$
  - $2 \frac{7}{8} \rightarrow 10.111_2$
Insufficient representation

- That way of representing floating point numbers is simple, but has two significant limitations.
  - Only numbers that can be written as the sum of powers of 2 can be represented exactly.
    - Example
      - $1/3 \quad 0.0101010101[01]…_2$
      - $1/5 \quad 0.001100110011[0011]…_2$
      - $1/10 \quad 0.0001100110011[0011]…_2$
    - Just one possible location for the binary point.
      - This limits how many bits can be used for the fractional part and the whole number part.
      - We can either represent very large numbers well or very small numbers well, but not both.
IEEE Floating Point

- **IEEE Standard 754**
  - Established in 1985 as uniform standard for floating point arithmetic
  - Before that, many idiosyncratic formats
  - Supported by all major CPUs

- **Driven by numerical concerns**
  - Nice standards for rounding, overflow, underflow
  - Numerical analysts predominated over hardware designers in defining standard
  - Therefore, hard to make fast in hardware (i.e. its slow!)
Floating Point Representation

- **Numerical form**
  \[ (-1)^s \times M_2 \times 2^E \]
  - Sign bit \( s \) determines whether number is negative or positive
  - Mantissa \( M \) *normally* a fractional value, range \([1.0, 2.0)\)
  - Exponent \( E \) weights value by power of two

- **Encoding**
  - Most significant bit is sign bit \( s \)
  - \texttt{exp} field *encodes* \( E \) *(but is not equal to \( E \))*
  - \texttt{frac} field *encodes* \( M \) *(but is not equal to \( M \))*
IEEE precision options

Single precision: 32 bits

Double precision: 64 bits
Interpreting IEEE Values

- Three possible methods by which we evaluate a given bit vector representing a floating point type.
  - ‘Normalized’ values
  - ‘Denormalized’ values
  - ‘Special’ values

- Normalized is the most common case.

- Denormalized is for representing numbers very close to zero.

- Special, is well, special.

- The value of $\text{exp}$ determines what kind of value it is, and therefore how it is encoded and interpreted.
Normalized values

- **Precondition:** \( \text{exp} \neq 000\ldots0 \) and \( \text{exp} \neq 111\ldots1 \)
- For some bit pattern: \( \text{value}_{10} = (-1)^s * M_2 * 2^E \)
- \( s = \) sign bit \( s \)
- \( E = [\text{exp}] - \text{bias} \)
  - \( \text{bias} = 2^{k-1} - 1 \)
  - \( k = \) number of bits in \( \text{exp} \)
- \( M = 1.\frac{\text{frac}}{} \)
  - By assuming the leading bit is 1, we get an extra bit for “free”
  - Smallest value when all bits are zero: 000\ldots0, \( M = 1.0 \)
  - Largest value when all bits are one: 111\ldots1, \( M = 2.0 - \varepsilon \)
Normalized encoding example

- float f = 15213.0;
  
  $15213_{10} = 11101101101101_2$
  
  $= 1.1101101101101_2 \times 2^{13}$

\[ \text{value}_{10} = (-1)^s \times M_2 \times 2^E \]
Normalized encoding example, con’t

- float $f = 15213.0$;
  
  \[ 15213_{10} = 11101101101101_2 \]
  
  \[ = 1.1101101101101_2 \times 2^{13} \]

- Mantissa
  
  \[ M = 1.1101101101101_2 \times 2^{13} \]
  
  \[ frac = 11011011011010000000000000_2 \]

\[ value_{10} = (-1)^s \times M_2 \times 2^E \]
Normalized encoding example, con’t

- float f = 15213.0;
  
  $$15213_{10} = 11101101101101_2$$
  $$= 1.1101101101101_2 \times 2^{13}$$

- Mantissa
  
  $$M = 1.1101101101101_2 \times 2^{13}$$
  $$frac = 1101101101101000000000000_2$$

- Exponent
  
  $$E = 13$$
  $$bias = 127 = (2^{8-1} - 1)$$
  $$exp = 140 = 10001100_2$$

$$s \quad exp \quad frac$$

<table>
<thead>
<tr>
<th></th>
<th>10001100</th>
<th>110110110110100000000000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

$$value_{10} = (-1)^s \times M_2 \times 2^E$$
Why?

- For normalized 32-bit single precision…
  - The value of $exp$ is in the range $0 < exp < 255$
  - The value of $E$ is in the range $-127 < E <= 127$
  - Fairly large numbers; $<= 2^{127}$
  - Fairly small numbers; $>= 2^{-126}$

- For normalized 64-bit double precision, obviously this range is greater.

- Note that there is always a leading 1 in the value of mantissa $M$ for ‘normalized values’, so we cannot represent numbers that are very small.

- Next, we will observe what happens when $exp$ is either 00…0 or 11…1
Denormalized values

- **Precondition:** \( exp = 000\ldots0 \)

- For some bit pattern: \( \text{value}_{10} = (-1)^s \times M_2 \times 2^E \)

- \( M = 0.\text{frac} \)
  - No implicit 1 prefix.
  - Allows for representation of numbers much closer to 0

- \( E = 1 - \text{bias} \)
  - \( \text{bias} = 2^{k-1} - 1 \)
  - \( k = \) number of bits in \( exp \)
  - Differs from ‘normalized’, as \( exp \) obviously 0

- If \( exp = 000\ldots0, \text{frac} = 000\ldots0 \) represents 0.0

- If \( exp = 000\ldots0, \text{frac} \neq 000\ldots0 \) represent numbers very close to 0.0
Special values

- Precondition: $exp = 111\ldots1$

- If $exp = 111\ldots1$, $frac = 000\ldots0$
  - Represents positive or negative infinity, a result of overflow
  - Examples:
    - $-1.0/-0.0 = +\infty$
    - $1.0/-0.0 = -\infty$

- If $exp = 111\ldots1$, $frac \neq 000\ldots0$
  - Not-a-Number (NaN)
  - A case when no numeric value can be determined
  - Examples:
    - $\sqrt{-1}$
    - $\infty-\infty$
    - $\infty*0$
Floating point encoding number line

-∞ \rightarrow -\text{Normalized} \rightarrow -\text{Denorm} \rightarrow +\text{Denorm} \rightarrow +\text{Normalized} \rightarrow +∞

NaN \rightarrow -0 \rightarrow +0 \rightarrow NaN
## Tiny Floating Point

A 6-bit Floating Point Representation:

- **Sign bit** \( s \) is in the most significant bit.
- The next three bits are the **exp**, with a **bias** of 3.
  - Note that the **bias** is the same for all 6-bit precision numbers!
- The last two bits are the **frac**.

### IEEE Format

- Normalized, denormalized and special values.
Tiny Normalized Example 1

\[ \text{value}_{10} = (-1)^s \times M_2 \times 2^E \]

\[ E = \text{exp} - \text{bias} \]

- **000100_2** (smallest positive value)
  - \( s = (-1)^0 = 1 \)
  - \( M = 1.00_2 \)
  - \( \text{bias} = 2^{3-1} - 1 = 3_{10} \)
  - \( E = 001_2 - 3_{10} = 1_{10} - 3_{10} = -2_{10} \)
  - \( 1 \times 1.00_2 \times 2^{-2} = .01_2 = 0.25_{10} \)

<table>
<thead>
<tr>
<th>s</th>
<th>exp</th>
<th>frac</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3-bits</td>
<td>2-bits</td>
</tr>
<tr>
<td>000000</td>
<td>010000</td>
<td>100000</td>
</tr>
<tr>
<td>000001</td>
<td>010001</td>
<td>100001</td>
</tr>
<tr>
<td>000010</td>
<td>010010</td>
<td>100100</td>
</tr>
<tr>
<td>000011</td>
<td>010101</td>
<td>100101</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

All possible 6-bit sequences
Tiny Normalized Example 2

\[
\text{value}_{10} = (-1)^s \times M_2 \times 2^E \\
E = \exp - \text{bias}
\]

- 011011₂ (largest positive value)
  - \( s = (-1)^0 = 1 \)
  - \( M = 1.11₂ \)
  - \( \text{bias} = 2^3 - 1 = 3_{10} \)
  - \( E = 110₂ - 3_{10} = 6_{10} - 3_{10} = 3_{10} \)
  - \( 1 \times 1.11₂ \times 2^3 = 1110₂ = 14.0_{10} \)
Tiny Denormalized Example 1

\[ \text{value}_{10} = (-1)^s \times M_2 \times 2^E \]

\[ E = 1 - \text{bias} \]

- **100011_2** (smallest negative value)
  - \( s = (-1)^1 = -1 \)
  - \( M = 0.11_2 \)
  - \( \text{bias} = 2^3 - 1 = 3_{10} \)
  - \( E = 1_{10} - 3_{10} = -2_{10} \)
  - \( -1 \times 0.11_2 \times 2^{-2} = -0.0011_2 = -0.1875_{10} \)
Tiny Denormalized Example 2

\[
\text{value}_{10} = (-1)^s \times M_2 \times 2^E
\]
\[
E = 1 - \text{bias}
\]

- **000001_2** (smallest positive less than 1)
  - \( s = (-1)^0 = 1 \)
  - \( M = 0.01_2 \)
  - \( \text{bias} = 2^{3-1} - 1 = 3_{10} \)
  - \( E = 1_{10} - 3_{10} = -2_{10} \)
  - \( 1 \times 0.01_2 \times 2^{-2} = 0.0001_2 = 0.0625_{10} \)
## Tiny special values

### Result of overflow or infeasibility

- \( \text{exp} = 111, \quad \text{frac} = 00 \)
  - 011100, 111100
  - Positive or negative infinity
- \( \text{exp} = 111, \quad \text{frac} \neq 00 \)
  - 011101, 011110, 011111, 111101, 111110, 111111
  - Not-a-Number (NaN)

### Table of all possible 6-bit sequences

<table>
<thead>
<tr>
<th>s</th>
<th>exp</th>
<th>frac</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3-bits</td>
<td>2-bits</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>000000</td>
<td>010000</td>
<td>100000</td>
</tr>
<tr>
<td>000001</td>
<td>010001</td>
<td>100001</td>
</tr>
<tr>
<td>000010</td>
<td>010010</td>
<td>100010</td>
</tr>
<tr>
<td>000011</td>
<td>010011</td>
<td>100011</td>
</tr>
<tr>
<td>000100</td>
<td>010100</td>
<td>100100</td>
</tr>
<tr>
<td>000101</td>
<td>010101</td>
<td>100101</td>
</tr>
<tr>
<td>000110</td>
<td>010110</td>
<td>100110</td>
</tr>
<tr>
<td>000111</td>
<td>010111</td>
<td>100111</td>
</tr>
<tr>
<td>001000</td>
<td>011000</td>
<td>101000</td>
</tr>
<tr>
<td>001001</td>
<td>011001</td>
<td>101001</td>
</tr>
<tr>
<td>001010</td>
<td>011010</td>
<td>101010</td>
</tr>
<tr>
<td>001011</td>
<td>011011</td>
<td>101011</td>
</tr>
<tr>
<td>001100</td>
<td>011100</td>
<td>101100</td>
</tr>
<tr>
<td>001101</td>
<td>011101</td>
<td>101101</td>
</tr>
<tr>
<td>001110</td>
<td>011110</td>
<td>101110</td>
</tr>
<tr>
<td>001111</td>
<td>011111</td>
<td>101111</td>
</tr>
</tbody>
</table>

#### Diagram:
- • Normalized
- ▲ Denormalized
- ◯ Special
Exercises
Exercise 1

value_{10} = (-1)^s \times M_2 \times 2^E
E = ? - bias

- 100111_2
  - s = (-1)^s = ?
  - M = ?_2
  - bias = ?_{10}
  - E = ?_{10}
  - value_{10} = ? = -0.4375_{10}
Exercise 2

\[
\text{value}_{10} = (-1)^s * M_2 * 2^E
\]

E = ? - bias

- \text{100001}_2
  - s = (-1)^s = ?
  - M = ?_2
  - bias = ?_{10}
  - E = ?_{10}
  - value_{10} = ? = -0.0625_{10}