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 $2^{\text{word size}}$ 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, that’s 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).

<table>
<thead>
<tr>
<th>32-bit Word</th>
<th>64-bit Word</th>
<th>Byte</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addr = 0000</td>
<td>Addr = 0000</td>
<td></td>
<td>0000</td>
</tr>
<tr>
<td>Addr = 0004</td>
<td></td>
<td></td>
<td>0001</td>
</tr>
<tr>
<td>Addr = 0008</td>
<td></td>
<td></td>
<td>0002</td>
</tr>
<tr>
<td>Addr = 0012</td>
<td></td>
<td></td>
<td>0003</td>
</tr>
<tr>
<td>Addr = 0004</td>
<td></td>
<td></td>
<td>0004</td>
</tr>
<tr>
<td>Addr = 0008</td>
<td></td>
<td></td>
<td>0005</td>
</tr>
<tr>
<td>Addr = 0012</td>
<td></td>
<td></td>
<td>0006</td>
</tr>
<tr>
<td>Addr = 0004</td>
<td></td>
<td></td>
<td>0007</td>
</tr>
<tr>
<td>Addr = 0008</td>
<td></td>
<td></td>
<td>0008</td>
</tr>
<tr>
<td>Addr = 0012</td>
<td></td>
<td></td>
<td>0009</td>
</tr>
<tr>
<td>Addr = 0004</td>
<td></td>
<td></td>
<td>0010</td>
</tr>
<tr>
<td>Addr = 0008</td>
<td></td>
<td></td>
<td>0011</td>
</tr>
<tr>
<td>Addr = 0012</td>
<td></td>
<td></td>
<td>0012</td>
</tr>
<tr>
<td>Addr = 0004</td>
<td></td>
<td></td>
<td>0013</td>
</tr>
<tr>
<td>Addr = 0008</td>
<td></td>
<td></td>
<td>0014</td>
</tr>
<tr>
<td>Addr = 0012</td>
<td></td>
<td></td>
<td>0015</td>
</tr>
</tbody>
</table>
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 to the bytes appear? What “end” of the word does the MSB live at?
Byte ordering in a word con’t

- Variable $x$ has 4-byte value of $0x01234567$
- Address given by dereferencing $x$ is $0x100$

![Byte Ordering Diagram](image)

- 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 $1011.101_2$?

$$\begin{align*}
(1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3}) \\
8 + 2 + 1 + \frac{1}{2} + \frac{1}{8} \\
11.625_{10}
\end{align*}$$

- Going the other direction

- $5 \frac{3}{4} \quad \rightarrow \quad 101.11_2$

- $2 \frac{7}{8} \quad \rightarrow \quad 10.111_2$
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
    - \(1/3\) 0.0101010101[01]…\(2\)
    - \(1/5\) 0.001100110011[0011]…\(2\)
    - \(1/10\) 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!)
IEEE precision options

Single precision: 32 bits

Double precision: 64 bits
Floating Point Representation

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

- **Encoding** (3 bit vectors that encode a floating point number)
  - Most significant bit is the sign bit \( s \)
  - \( \text{exp} \) field encodes \( E \) (but is not equal to \( E \))
  - \( \text{frac} \) field encodes \( M \) (but is not equal to \( M \))
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 $\exp$ determines “type” and therefore how it is encoded and interpreted.
Floating point encoding number line
Normalized values

- Precondition: $exp \neq 000\ldots0$ and $exp \neq 111\ldots1$
- For some bit pattern: $value_{10} = (-1)^s \times M_2 \times 2^E$
- $s = \text{sign bit } s$
- $E = [exp] - \text{bias}$
  - $\text{bias} = 2^{k-1} - 1$
  - $k = \text{number of bits in } exp$
- $M = 1.[frac]$
  - 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: $111\ldots1$, $M = 2.0 - \epsilon$
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;
  
  \[ 15213_{10} = 11101101101101_{2} \]
  \[ = 1.1101101101101_{2} \times 2^{13} \]

- Significand (aka Mantissa)
  
  \[ M = 1.1101101101101_{2} \times 2^{13} \]
  \[ frac = 11011011011010000000000000_{2} \]

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

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

- Significand (aka Mantissa)
  \[
  M = 1.1101101101101_{2} \times 2^{13} \\
  frac = 1101101101101000000000000_{2}
  \]

- Exponent
  \[
  E = 13 \\
  bias = (2^{8} - 1) = 127_{10} \\
  exp = E + bias = 140_{10} = 10001100_{2}
  \]

- Value
  \[
  value_{10} = (-1)^{s} \times M \times 2^{E}
  \]
Range of Expression

- For normalized 32-bit single precision…
  - The value if exp 1…254
  - The value of $E -126…127$
  - Fairly large numbers; $\leq 2^{127}$
  - Fairly small numbers; $\geq 2^{-126}$

- 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: $\text{exp} = 000\ldots0$
- 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
- $E = 1 - \text{bias}$
  - $\text{bias} = 2^{k-1} - 1$
  - $k = \text{number of bits in exp}$
  - Differs from ‘normalized’, as $\text{exp}$ is 0, so use 1
- If $\text{exp} = 000\ldots0, \text{frac} = 000\ldots0$ represents 0.0
- If $\text{exp} = 000\ldots0, \text{frac} \neq 000\ldots0$ 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:
    - \(\sqrt{-1}\)
    - \(\infty-\infty\)
    - \(\infty*0\)
Tiny Floating Point
Tiny Floating Point

- 6-bit Floating Point Representation
  - The 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

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

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

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

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

- **011011_2** (largest positive value)
  - \(s = (-1)^0 = 1\)
  - \(M = 1.11_2\)
  - \(\text{bias} = 2^{3-1} - 1 = 3_{10}\)
  - \(E = 110_2 - 3_{10} = 6_{10} - 3_{10} = 3_{10}\)
  - \(1 \times 1.11_2 \times 2^3 = 1110_2 = 14.0_{10}\)
**Tiny Denormalized Example 1**

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

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

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

<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>
</tbody>
</table>

All possible 6-bit sequences

- **Normalized**
- **Denormalized**
- **Special**

---

### All possible 6-bit sequences

<table>
<thead>
<tr>
<th>000000</th>
<th>010000</th>
<th>100000</th>
<th>110000</th>
</tr>
</thead>
<tbody>
<tr>
<td>000001</td>
<td>010001</td>
<td>100001</td>
<td>110001</td>
</tr>
<tr>
<td>000010</td>
<td>010010</td>
<td>100010</td>
<td>110010</td>
</tr>
<tr>
<td>000011</td>
<td>010011</td>
<td>100011</td>
<td>110011</td>
</tr>
<tr>
<td>000100</td>
<td>010100</td>
<td>100100</td>
<td>110100</td>
</tr>
<tr>
<td>000101</td>
<td>010101</td>
<td>100101</td>
<td>110101</td>
</tr>
<tr>
<td>000110</td>
<td>010110</td>
<td>100110</td>
<td>110110</td>
</tr>
<tr>
<td>000111</td>
<td>010111</td>
<td>100111</td>
<td>110111</td>
</tr>
<tr>
<td>001000</td>
<td>011000</td>
<td>101000</td>
<td>111000</td>
</tr>
<tr>
<td>001001</td>
<td>011001</td>
<td>101001</td>
<td>111001</td>
</tr>
<tr>
<td>001010</td>
<td>011010</td>
<td>101010</td>
<td>111010</td>
</tr>
<tr>
<td>001011</td>
<td>011011</td>
<td>101011</td>
<td>111011</td>
</tr>
<tr>
<td>001100</td>
<td>011100</td>
<td>101100</td>
<td>111100</td>
</tr>
<tr>
<td>001101</td>
<td>011101</td>
<td>101101</td>
<td>111101</td>
</tr>
<tr>
<td>001110</td>
<td>011110</td>
<td>101110</td>
<td>111110</td>
</tr>
<tr>
<td>001111</td>
<td>011111</td>
<td>101111</td>
<td>111111</td>
</tr>
</tbody>
</table>
Tiny Denormalized Example 2

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

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

---

<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>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>
<tr>
<td>010000</td>
<td>100000</td>
<td>101000</td>
</tr>
<tr>
<td>010001</td>
<td>100001</td>
<td>101001</td>
</tr>
<tr>
<td>010010</td>
<td>100010</td>
<td>101010</td>
</tr>
<tr>
<td>010011</td>
<td>100011</td>
<td>101011</td>
</tr>
<tr>
<td>010100</td>
<td>100100</td>
<td>101100</td>
</tr>
<tr>
<td>010101</td>
<td>100101</td>
<td>101101</td>
</tr>
<tr>
<td>010110</td>
<td>100110</td>
<td>101110</td>
</tr>
<tr>
<td>010111</td>
<td>100111</td>
<td>101111</td>
</tr>
<tr>
<td>011000</td>
<td>101000</td>
<td>110000</td>
</tr>
<tr>
<td>011001</td>
<td>101001</td>
<td>110001</td>
</tr>
<tr>
<td>011010</td>
<td>101010</td>
<td>110010</td>
</tr>
<tr>
<td>011011</td>
<td>101011</td>
<td>110011</td>
</tr>
<tr>
<td>011100</td>
<td>101100</td>
<td>110100</td>
</tr>
<tr>
<td>011101</td>
<td>101101</td>
<td>110101</td>
</tr>
<tr>
<td>011110</td>
<td>101110</td>
<td>110110</td>
</tr>
<tr>
<td>011111</td>
<td>101111</td>
<td>110111</td>
</tr>
</tbody>
</table>

All possible 6-bit sequences
Tiny special values

- **Result of overflow or infeasibility**
  - \( exp = 111, \ frac = 00 \)
    - 011100, 111100
  - Positive or negative infinity
  - \( exp = 111, \ frac \neq 00 \)
    - 011101, 011110, 011111, 111101, 111110, 111111
  - Not-a-Number (NaN)
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

value\textsubscript{10} = (-1)^s \times M_2 \times 2^E

E = ? - bias

- 100001\textsubscript{2}
  - s = (-1)^s = ?
  - M = ?\textsubscript{2}
  - bias = ?\textsubscript{10}
  - E = ?\textsubscript{10}
  - value\textsubscript{10} = ? = -0.0625\textsubscript{10}