Review of Numbers

• Computers are made to deal with numbers

• What can we represent in \( N \) bits?
  - \( 2^N \) things, and no more! They could be...
  - Unsigned integers:
    
    \[
    0 \text{ to } 2^N - 1
    \]
    (for \( N=32 \), \( 2^{31} = 4,294,967,295 \))
  - Signed Integers (Two's Complement)
    
    \[
    -2^{(N-1)} \text{ to } 2^{(N-1)} - 1
    \]
    (for \( N=32 \), \( 2^{31} = 2,147,483,648 \))

Representation of Fractions

“Binary Point” like decimal point signifies boundary between integer and fractional parts:

Example 6-bit representation:

\[
\begin{array}{cccccc}
XX & . & yyyy \\
2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
0 & 0 & 1 & 1 & 1 & 0
\end{array}
\]

\[
10.1010_2 = 1 \times 2^1 + 1 \times 2^{-1} + 1 \times 2^{-3} = 2.625_{10}
\]

If we assume “fixed binary point”, range of 6-bit representations with this format:

0 to 3.9375 (almost 4)

Fractional Powers of 2

<table>
<thead>
<tr>
<th>( i )</th>
<th>( 2^i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
</tr>
<tr>
<td>3</td>
<td>0.125</td>
</tr>
<tr>
<td>4</td>
<td>0.0625</td>
</tr>
<tr>
<td>5</td>
<td>0.03125</td>
</tr>
<tr>
<td>6</td>
<td>0.015625</td>
</tr>
<tr>
<td>7</td>
<td>0.0078125</td>
</tr>
<tr>
<td>8</td>
<td>0.00390625</td>
</tr>
<tr>
<td>9</td>
<td>0.001953125</td>
</tr>
<tr>
<td>10</td>
<td>0.0009765625</td>
</tr>
<tr>
<td>11</td>
<td>0.00048828125</td>
</tr>
<tr>
<td>12</td>
<td>0.000244140625</td>
</tr>
<tr>
<td>13</td>
<td>0.0001220703125</td>
</tr>
<tr>
<td>14</td>
<td>0.00006103515625</td>
</tr>
<tr>
<td>15</td>
<td>0.000030517578125</td>
</tr>
</tbody>
</table>

What about other numbers?

1. Very large numbers? (seconds/millennium)

\[
31,556,926,000,000 = (3.1556926 \times 10^{10})
\]

2. Very small numbers? (Bohr radius)

\[
6.0000000000529177 \times 10^{-11}
\]

3. Numbers with both integer & fractional parts?

\[
1.5
\]

First consider #3.

…our solution will also help with 1 and 2.

“Doomsday” Seed Vault Opens

“The seed bank on a remote island near the Arctic Ocean is considered the ultimate safety net for the world’s seed collections, protecting them from a wide range of threats including war, natural disasters, lack of funding or simply poor agricultural management.”

James Gosling
Sun Fellow
Java Inventor
1998-02-28

“95% of the folks out there are completely clueless about floating-point.”

James Gosling
Sun Fellow
Java Inventor
1998-02-28
**Representation of Fractions with Fixed Pt.**

What about addition and multiplication?

Addition is straightforward:

\[
\begin{align*}
01.100 & \quad 1.5 \\
00.100 & \quad 0.5 \\
\hline
10.000 & \quad 2.0 \\
\end{align*}
\]

Multiplication a bit more complex:

\[
\begin{align*}
01.100 & \quad 1.5 \\
00.100 & \quad 0.5 \\
0000 & \quad 0 \\
0110 & \quad 0 \\
0000 & \quad 0 \\
0001100 & \quad \text{HI} \quad \text{LOW}
\end{align*}
\]

Where’s the answer, 0.11? (need to remember where point is)

---

**Scientific Notation (in Decimal)**

- mantissa: 6.02\_10 \times 10^{23}
- exponent
- decimal point
- radix (base)

- Normalized form: no leading 0s (exactly one digit to left of decimal point)
- Alternatives to representing 1/1,000,000,000
  - Normalized: 1.0 \times 10^{-9}
  - Not normalized: 0.1 \times 10^{-8}, 10.0 \times 10^{-10}

---

**Scientific Notation (in Binary)**

- mantissa: 1.0\_two \times 2^{-1}
- exponent
- “binary point”
- radix (base)

- Computer arithmetic that supports it called floating point, because it represents numbers where the binary point is not fixed, as it is for integers
- Declare such variable in C as float

---

**Floating Point Representation (1/2)**

- Normal format: +1.xxxxxxxxx\_two \times 2^{yyy}\_two
- Multiple of Word Size (32 bits)

\[
\begin{array}{c|c|c}
31 & 30 & 0 \\
\hline
S & \text{Exponent} & \text{Significand} \\
1 \text{ bit} & 8 \text{ bits} & 23 \text{ bits}
\end{array}
\]

- S represents Sign
- Exponent represents y’s
- Significand represents x’s
- Represent numbers as small as 2.0 \times 10^{-38} to as large as 2.0 \times 10^{38}

---

**Floating Point Representation (2/2)**

- What if result too large? (> 2.0 \times 10^{38}, < -2.0 \times 10^{38})
  - Overflow! ⇒ Exponent larger than represented in 8-bit Exponent field
- What if result too small? (>0 \& < 2.0 \times 10^{-38}, <0 \& > - 2.0 \times 10^{-38})
  - Underflow! ⇒ Negative exponent larger than represented in 8-bit Exponent field

\[
\begin{array}{c|c|c|c|c|c|c}
\text{overflow} & \text{underflow} & \text{overflow} & \text{overflow} & \text{overflow} & \text{overflow} \\
-2 \times 10^{38} & -1 \times 2 \times 10^{38} & 0 & 2 \times 10^{38} & 1 & 2 \times 10^{36}
\end{array}
\]

- What would help reduce chances of overflow and/or underflow?
Double Precision Fl. Pt. Representation

- Next Multiple of Word Size (64 bits)
  31 30  20 19  0
  S| Exponent | Significand
  1 bit 11 bits  20 bits
  Significand (cont’d)
  32 bits
  - Double Precision (vs. Single Precision)
    - C variable declared as double
    - Represent numbers almost as small as $2.0 \times 10^{-308}$ to almost as large as $2.0 \times 10^{308}$
    - But primary advantage is greater accuracy due to larger significand

QUAD Precision Fl. Pt. Representation

- Next Multiple of Word Size (128 bits)
- Unbelievable range of numbers
- Unbelievable precision (accuracy)
- Currently being worked on (IEEE 754r)
  - Current version has 15 exponent bits and 112 significand bits (113 precision bits)
  - Oct-Precision?
    - Some have tried, no real traction so far
  - Half-Precision?

IEEE 754 Floating Point Standard (1/3)

Single Precision (DP similar):
  31 30  23 22  0
  S| Exponent | Significand
  1 bit  8 bits  23 bits
  - Sign bit: 1 means negative
 0 means positive
  - Significand:
    - To pack more bits, leading 1 implicit for normalized numbers
    - $1 + 23$ bits single, $1 + 52$ bits double
    - always true: $0 < \text{Significand} < 1$
      (for normalized numbers)
  - Note: 0 has no leading 1, so reserve exponent value 0 just for number 0

IEEE 754 Floating Point Standard (2/3)

- IEEE 754 uses “biased exponent” representation.
- Designers wanted FP numbers to be used even if no FP hardware; e.g., sort records with FP numbers using integer compares.
- Wanted bigger (integer) exponent field to represent bigger numbers.
- 2’s complement poses a problem (because negative numbers look bigger)
- We’re going to see that the numbers are ordered EXACTLY as in sign-magnitude
  - i.e., counting from binary odometer 00…00 up to 11…11 goes from 0 to +MAX to -0 to -MAX to 0

IEEE 754 Floating Point Standard (3/3)

-Called Biased Notation, where bias is number subtracted to get real number
- IEEE 754 uses bias of 127 for single prec.
- Subtract 127 from Exponent field to get actual value for exponent
- 1023 is bias for double precision

Summary (single precision):
  31 30  23 22  0
  S| Exponent | Significand
  1 bit  8 bits  23 bits
  \((-1)^s \times (1 + \text{Significand}) \times 2^{(\text{Exponent} - 127)}\)
  - Double precision identical, except with exponent bias of 1023 (half, quad similar)
Father of the Floating point standard

IEEE Standard 754 for Binary Floating-Point Arithmetic.

Prof. Kahan

1989 ACM Turing Award Winner!

www.cs.berkeley.edu/~wkahan/.../ieee754status/754story.html

Example: Converting Binary FP to Decimal

0 0110 1000 101 0101 0100 0011 0100 0010

- Sign: 0 => positive
- Exponent:
  - 0110 100010 = 104_{ten}
  - Bias adjustment: 104 - 127 = -23
- Significand:
  1 + 1x2^{-1} + 0x2^{-2} + 1x2^{-3} + 0x2^{-4} + 1x2^{-5} +...
  = 1.0 + 0.666115
- Represents: 1.666115 \times 2^{-23} \approx 1.986 \times 10^{-7}
  (about 2/10,000,000)

Example: Converting Decimal to FP

1. Denormalize: -23.40625
2. Convert integer part:
   23 = 16 + (7 = 4 + (3 = 2 + (1))) = 10111_{2}
3. Convert fractional part:
   .40625 = .25 + (.15625 = .125 + (.03125)) = .01101_{2}
4. Put parts together and normalize:
   10111.01101 = 1.011101101 \times 2^{4}
5. Convert exponent: 127 + 4 = 1000011_{2}

Understanding the Significand (1/2)

- Method 1 (Fractions):
  - In decimal: 0.340_{10} \xrightarrow{} 340_{10}/1000_{10} \xrightarrow{} 34_{10}/100_{10}
  - In binary: 0.110_{2} \xrightarrow{} 110/1000_{2} = 6_{10}/8_{10}
  - Advantage: less purely numerical, more thought oriented; this method usually helps people understand the meaning of the significand better

Understanding the Significand (2/2)

- Method 2 (Place Values):
  - Convert from scientific notation
  - In decimal: 1.6732 = (1 \times 10^{0}) + (6 \times 10^{-1}) + (7 \times 10^{-2}) + (3 \times 10^{-3}) + (2 \times 10^{-4})
  - In binary: 1.1001 = (1 \times 2^{0}) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (0 \times 2^{-3}) + (1 \times 2^{-4})
  - Interpretation of value in each position extends beyond the decimal/binary point
  - Advantage: good for quickly calculating significand value; use this method for translating FP numbers

Peer Instruction

What is the decimal equivalent of the floating pt # above?

1: -1.75
2: -3.5
3: -7.5
4: -7
5: -7.5
6: -15
7: -7 \times 2^{-129}
8: -129 \times 2^{-7}
Peer Instruction Answer

What is the decimal equivalent of:

<table>
<thead>
<tr>
<th>S</th>
<th>Exponent</th>
<th>Significand</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>111 0000 0000 0000 0000 0000</td>
<td>1000 0001</td>
</tr>
</tbody>
</table>

\[
(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}
\]

1. \((-1)^1 \times (1 + .111) \times 2^{(129-127)}
   = -1.111 \times 2^2
   = -7.5

2. \((-1)^1 \times (1 + .111) \times 2^{(129-127)}
   = -1.111 \times 2^2
   = -7.5

3. \((-1)^1 \times (1 + .111) \times 2^{(129-127)}
   = -1.111 \times 2^2
   = -7.5

4. \((-1)^1 \times (1 + .111) \times 2^{(129-127)}
   = -1.111 \times 2^2
   = -7.5

5. \((-1)^1 \times (1 + .111) \times 2^{(129-127)}
   = -1.111 \times 2^2
   = -7.5

6. \((-1)^1 \times (1 + .111) \times 2^{(129-127)}
   = -1.111 \times 2^2
   = -7.5

7. \((-1)^1 \times (1 + .111) \times 2^{(129-127)}
   = -1.111 \times 2^2
   = -7.5

8. \((-1)^1 \times (1 + .111) \times 2^{(129-127)}
   = -1.111 \times 2^2
   = -7.5

---

And in conclusion...

- Floating Point lets us:
  - Represent numbers containing both integer and fractional parts; makes efficient use of available bits.
  - Store approximate values for very large and very small numbers.

- IEEE 754 Floating Point Standard is most widely accepted attempt to standardize interpretation of such numbers (Every desktop or server computer sold since ~1997 follows these conventions)

<table>
<thead>
<tr>
<th>Summary (single precision):</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponent</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>1 bit</td>
</tr>
</tbody>
</table>

\[
(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}
\]

- Double precision identical, except with exponent bias of 1023 (half, quad similar)