CMPT 295
Data Representation
Lecture 6 – Representing real numbers - IEEE floating point representation
Last Lecture

- Demo (code and results) posted!

- Addition:
  - Unsigned/signed:
    - Behave the same way at the bit level
    - Interpretation of resulting bit vector (sum) may differ
  - Unsigned: \((x + y) \mod 2^w\) (sum in proper range)
    - Integer addition with possible subtraction of \(2^w\)
  - Signed: \(U2T_w [(x + y) \mod 2^w]\) (sum in proper range)
    - Integer addition with possible addition or subtraction of \(2^w\)

- Subtraction
  - Becomes an addition where negative operands are transformed into their additive inverse

- Multiplication:
  - Unsigned: \((x \times y) \mod 2^w\) (product in proper range)
  - Signed: \(U2T_w [(x \times y) \mod 2^w]\) (product in proper range)
  - Can be replaced by additions and shifts
Today’s Menu

- Representing information as bits
  - “Under the Hood” - A look at memory
  - Bit manipulation

- Representing integers in memory
  - Unsigned and signed
  - Converting, expanding and truncating
  - Arithmetic operations

- Representing real numbers in memory
  - IEEE floating point representation
  - Floating point in C – casting, rounding, addition, …
Review: Real numbers

- Positional notation:

\[
\begin{align*}
&d_i \quad d_{i-1} \quad \cdots \quad d_2 \quad d_1 \quad d_0 \quad d_{-1} \quad d_{-2} \quad d_{-3} \quad \cdots \quad d_j \\
&10^j \\
&10^{i-1} \\
&100 \\
&10 \\
&1 \\
&\frac{1}{10} \\
&\frac{1}{100} \\
&\frac{1}{1000} \\
&\cdots \\
&10^0 \\
&10^{-1} \\
&10^{-2} \\
&10^{-3}
\end{align*}
\]

Example:
\[
2.345 = 2 \times 10^0 + 3 \times 10^{-1} + 4 \times 10^{-2} + 5 \times 10^{-3}
\]
Representing real numbers in memory

- How would 346.625 (= 346 5/8) be represented in memory?
- Here is a possible encoding scheme:
  Step 1 – 346.625 -> 346 .625

Binary representation is:
Representing real numbers

- Positional notation: can this be a possible encoding scheme?
Positional notation as encoding scheme?

-> Conversion

Using the positional notation as an encoding scheme

- Reals <-> binary

- What is $1011.101_2$ as a real number?

<table>
<thead>
<tr>
<th>Negative Powers of 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^{-1} = 0.5$</td>
</tr>
<tr>
<td>$2^{-2} = 0.25$</td>
</tr>
<tr>
<td>$2^{-3} = 0.125$</td>
</tr>
<tr>
<td>$2^{-4} = 0.0625$</td>
</tr>
<tr>
<td>$2^{-5} = 0.03125$</td>
</tr>
<tr>
<td>$2^{-6} = 0.015625$</td>
</tr>
<tr>
<td>$2^{-7} = 0.0078125$</td>
</tr>
<tr>
<td>$2^{-8} = 0.00390625$</td>
</tr>
</tbody>
</table>
Positional notation as encoding scheme?

-> Arithmetic operations

Using the positional notation as an encoding scheme

- Resulting binary encodings can be
  - added
  - multiplied by 2 by shifting left
  - divided by 2 by shifting right (unsigned)

- Example:

  Divide by 2: >> \[1011.101_2 = 11 \frac{5}{8} = 8 + 2 + 1 + \frac{1}{2} + \frac{1}{8}\]
  Divide by 2: >> \[101.1101_2 = 5 \frac{13}{16} = 4 + 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{16}\]
  Divide by 2: >> \[10.11101_2 = 2 \frac{29}{32} = 2 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{32}\]
Positional notation as encoding scheme?

- **Advantage:**
  - Straightforward arithmetic: can shift to multiply and divide, convert

- **Disadvantage:**
  - Cannot encode all reals: can only exactly represent numbers of the form $x/2^k$ (within a range)
  - Limited range: Only one setting of binary point within the $w$ bits

- **What is the range?**
  - Example -> $w = 32$ bits and binary point is located at 16th bit: $[31..16].[15..0]
    \begin{array}{c}
    1111111111111111.1111111111111111
    \end{array}$
  - Range:
Representing real numbers in memory

- Here is another possible encoding scheme: IEEE floating point representation

- Overview:
  - Numerical Form: \( V = (-1)^s \cdot M \cdot 2^E \)
    - \( s \) - Sign bit -> determines whether number is negative or positive
    - \( M \) - Significand (or Mantissa) -> fractional part of number
    - \( E \) - Exponent
  - Encoding
    - MSbit \( s \) (similar to sign-magnitude encoding)
    - \( E \) - Exponent
      - exp field encodes \( E \) (but is not equal to \( E \))
    - \( M \) - Significand
      - frac field encodes \( M \) (but is not equal to \( M \))
IEEE Floating Point Representation

Precision options

- **Single precision**: 32 bits ≈ 7 decimal digits, $10^{±38}$
  - $s$ | exp | frac
  - 1 | 8-bits | 23-bits

- **Double precision**: 64 bits ≈ 16 decimal digits, $10^{±308}$
  - S | exp | Frac
  - 1 | 11-bits | 52-bits
IEEE Floating Point Representation
Three “kinds” of values

<table>
<thead>
<tr>
<th>s</th>
<th>exp</th>
<th>frac</th>
</tr>
</thead>
<tbody>
<tr>
<td>k bits</td>
<td>n bits</td>
<td></td>
</tr>
</tbody>
</table>

- **denormalized**: $00...00$
- **normalized**: $\text{exp} \neq 0$ and $\text{exp} \neq 11...11$
- **special cases**: $11...11$

$E = \text{exp} - \text{bias}$
and $\text{bias} = 2^{k-1} - 1$

$M = 1 + \text{frac}$
How would 346.625 (= 346 5/8) be represented in memory?

a) Normalize the binary number

b) Determine $s$:

- $M$:
- $\text{frac}$:
- $E$:
- $\text{bias}$:
- $\text{exp}$:

Result:

$\boxed{\begin{array}{c} s \\ \text{exp} \\ \text{frac} \end{array}}$
Review: Scientific Notation

- From Wikipedia: a way of expressing numbers that are too big or too small to be conveniently written in decimal form

- Examples:
  - A proton’s mass is 0.0000000000000000000000000016726 kg -> $1.6726 \times 10^{-27}$ kg
  - Speed of light is 299,792,458 m/s -> $2.99792,458 \times 10^8$ m/s

Syntax

$$\pm \text{sign} \quad d_0 \cdot d_{-1} d_{-2} d_{-3} \ldots d_{-n} \times b^\text{exp}$$

sign, significand, base, exponent
Let’s practice!

- How would 346.62 be represented in memory?
Summary

- Integer representation: encode small range of values exactly
- Representing real numbers in memory
  - Positional notation (advantages and disadvantages)
  - IEEE floating point representation: wider range, mostly approximately
- Overview of IEEE Floating Point representation
  - $V = (-1)^s \times M \times 2^E$
  - Precision options
  - **Normalized**, denormalized and special values
Next Lecture

- Representing information as bits
- Representing integers in memory
  - Unsigned and signed
  - Converting, expanding and truncating
  - Arithmetic operations
- Representing real numbers in memory
  - IEEE floating point representation
  - Floating point in C – casting, rounding, addition, …