Computational Linear Algebra: Floating Point Representation and Rounding Error

David Shuman

February 1, 2022

IEEE Standard Floating Point Representation

IEEE Standard Floating Point Representation

- Normalized numbers have the form

$$[\pm]1.[mantissa] * 2^{[exponent]}$$

IEEE Standard Floating Point Representation

- Normalized numbers have the form

$$[\pm]1.[mantissa] * 2^{[exponent]}$$

- Example:

$$(23.171875)_{10} = (10111.001011)_2 = +1.0111001011 * 2^4$$

IEEE Standard Floating Point Representation

- Normalized numbers have the form

$$[\pm]1.[\mathit{mantissa}] * 2^{[\mathit{exponent}]}$$

- Example:

$$(23.171875)_{10} = (10111.001011)_2 = +1.0111001011 * 2^4$$

- The leading 1 is not stored, it is implied

IEEE Standard Floating Point Representation (cont.)

- Different types of numbers have different types of bits for each part:

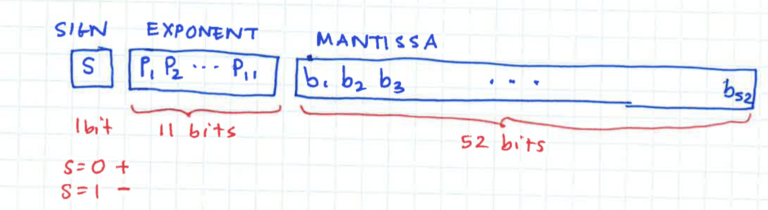
Precision	Sign	Exponent	Mantissa	Total
single	1	8	23	32
double	1	11	52	64
long double	1	15	64	80

IEEE Standard Floating Point Representation (cont.)

- Different types of numbers have different types of bits for each part:

Precision	Sign	Exponent	Mantissa	Total
single	1	8	23	32
double	1	11	52	64
long double	1	15	64	80

- We will almost always be using the 64-bit "double" precision:



- The sign is 0 for positive numbers and 1 for negative numbers.

IEEE Standard Floating Point Representation (cont.)

- $[\pm]1.[\mathit{mantissa}] * 2^{[\mathit{exponent}]}$
- Back to our example:

$$23.171875 = 10111.001011 = +1.0111001011 * 2^4$$

IEEE Standard Floating Point Representation (cont.)

- $[\pm]1.[\mathit{mantissa}] * 2^{[\mathit{exponent}]}$
- Back to our example:

$$23.171875 = 10111.001011 = +1.0111001011 * 2^4$$

- The sign is 0

IEEE Standard Floating Point Representation (cont.)

- $[\pm]1.[mantissa] * 2^{[exponent]}$
- Back to our example:

$$23.171875 = 10111.001011 = +1.0111001011 * 2^4$$

- The sign is 0
- The mantissa is

0111001011000000000000000000000000000000000000000000

or separated into four bit chunks:

0111 0010 1100 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

Exponents

- 11 bits allow us to represent $2^{11} = 2048$ different integers

Exponents

- 11 bits allow us to represent $2^{11} = 2048$ different integers
- We need both positive and negative exponents, and we keep two integers for special cases, so we have 2046 left to cover integers from -1022 to 1023:

binary	00000000000	00000000001	00000000010	$\cdots$	11111111110	11111111111
decimal	0	1	2	$\cdots$	2046	2047
exponent	special	-1022	-1021	$\cdots$	1023	special

Exponents

- 11 bits allow us to represent $2^{11} = 2048$ different integers
- We need both positive and negative exponents, and we keep two integers for special cases, so we have 2046 left to cover integers from -1022 to 1023:

binary	00000000000	00000000001	00000000010	$\cdots$	11111111110	11111111111
decimal	0	1	2	$\cdots$	2046	2047
exponent	special	-1022	-1021	$\cdots$	1023	special

- 1023 is the exponent bias. We add this amount to the actual exponent, and then store that number (note: this is the same as adding 1024 and subtracting 1)
- Subtract 1023 after storing (or subtract 1024 and add 1)
- Thus $+1.0111001011 * 2^4$ is

0	10000000011	$011100101100 \cdots 0$

Special Cases and More Examples

Special cases:

- Exponent of 2047 is equal to ∞ if the mantissa is all zeros, and equal to NaN (Not a Number) otherwise
- Exponent of 0 used to represent subnormal (very small) numbers. See the book as we will not harp on these

Special Cases and More Examples

Special cases:

- Exponent of 2047 is equal to ∞ if the mantissa is all zeros, and equal to NaN (Not a Number) otherwise
- Exponent of 0 used to represent subnormal (very small) numbers. See the book as we will not harp on these

More examples:

1. $(-101101.0011101)_2$ is represented as

1	10000000100	$011010011101000 \cdots 0$

Special Cases and More Examples

Special cases:

- Exponent of 2047 is equal to ∞ if the mantissa is all zeros, and equal to NaN (Not a Number) otherwise
- Exponent of 0 used to represent subnormal (very small) numbers. See the book as we will not harp on these

More examples:

1. $(-101101.0011101)_2$ is represented as

1	10000000100	$01101001110100 \cdots 0$

Special Cases and More Examples

Special cases:

- Exponent of 2047 is equal to ∞ if the mantissa is all zeros, and equal to NaN (Not a Number) otherwise
- Exponent of 0 used to represent subnormal (very small) numbers. See the book as we will not harp on these

More examples:

1. $(-101101.0011101)_2$ is represented as

1	10000000100	$01101001110100 \cdots 0$

2. What do these IEEE numbers represent in binary?

1	10000110001	$0111000101100 \cdots 0$

0	01111011110	$0111000101100 \cdots 0$

Special Cases and More Examples

Special cases:

- Exponent of 2047 is equal to ∞ if the mantissa is all zeros, and equal to NaN (Not a Number) otherwise
- Exponent of 0 used to represent subnormal (very small) numbers. See the book as we will not harp on these

More examples:

1. $(-101101.0011101)_2$ is represented as

1	10000000100	$011010011101 00 \cdots 0$

2. What do these IEEE numbers represent in binary?

1	10000110001	$0111000101100 \cdots 0$

0	01111011110	$0111000101100 \cdots 0$

$$-1.0111000101100 * 2^{50} \quad \text{and} \quad +1.0111000101100 * 2^{-33}$$

Rounding

Rounding

If we always chop (drop/ignore/truncate) the bits after b_{52} then we will systematically be rounding down. This can accumulate, so we use rounding:

- If what is left over is $< 1/2$:

  $$1.\boxed{b_1 b_2 b_3 \cdots b_{51} b_{52}}\ \underbrace{0\cdots\cdots\cdots\cdots}_{\text{round down by truncating}}$$

  just drop the bits beyond position 52

- If what is left over is $> 1/2$:

  $$1.\boxed{b_1 b_2 b_3 \cdots b_{51} b_{52}}\ \underbrace{1\cdots\cdots\cdots\cdots}_{\text{round up by adding 1 to } b_{52}}$$

  add 1 to bit b_{52} carrying as necessary

- If what is left over is $= 1/2$:

  $$1.\boxed{b_1 b_2 b_3 \cdots b_{51} b_{52}}1\ \underbrace{0\cdots\cdots\cdots\cdots}_{\text{all 0's}}$$

  if $b_{52} = 1$: (round up)
  add 1 to b_{52}
  if $b_{52} = 0$: (round down)
  do nothing

  - in this way we always end up with $b_{52} = 0$
  - these two possibilities are equally likely

Machine ε

- The number machine epsilon, denoted $\varepsilon_{\text{mach}}$ or ε_M, is the distance between 1 and the next smallest representable number

Machine ε

- The number machine epsilon, denoted $\varepsilon_{\text{mach}}$ or ε_M, is the distance between 1 and the next smallest representable number
- In our IEEE double precision representation, this next smallest number is

$$1 + \varepsilon_M = +1.\boxed{\,000\cdots\cdots\cdots 001\,} = 1 + 2^{-52},$$

Machine ε

- The number machine epsilon, denoted $\varepsilon_{\text{mach}}$ or ε_M, is the distance between 1 and the next smallest representable number

- In our IEEE double precision representation, this next smallest number is

$$1 + \varepsilon_M = +1.\boxed{000 \cdots\cdots\cdots 001} = 1 + 2^{-52},$$

- So $\varepsilon_M = 2^{-52}$

Machine ε

- The number machine epsilon, denoted $\varepsilon_{\text{mach}}$ or ε_M, is the distance between 1 and the next smallest representable number

- In our IEEE double precision representation, this next smallest number is

$$1 + \varepsilon_M = +1.\boxed{\;000\cdots\cdots\cdots001\;} = 1 + 2^{-52},$$

- So $\varepsilon_M = 2^{-52}$

- Note: $2^{-52} = (2^{10})^{-5.2} \approx (10^3)^{-5.2} = 10^{-15.6} \approx 10^{-16}$

Machine ε

- The number machine epsilon, denoted $\varepsilon_{\text{mach}}$ or ε_M, is the distance between 1 and the next smallest representable number
- In our IEEE double precision representation, this next smallest number is

$$1 + \varepsilon_M = +1.\boxed{000 \cdots\cdots\cdots 001} = 1 + 2^{-52},$$

- So $\varepsilon_M = 2^{-52}$
- Note: $2^{-52} = (2^{10})^{-5.2} \approx (10^3)^{-5.2} = 10^{-15.6} \approx 10^{-16}$
- `.Machine$double.eps`

Machine ε

- The number machine epsilon, denoted $\varepsilon_{\text{mach}}$ or ε_M, is the distance between 1 and the next smallest representable number

- In our IEEE double precision representation, this next smallest number is

$$1 + \varepsilon_M = +1.\boxed{000\cdots\cdots\cdots001} = 1 + 2^{-52},$$

- So $\varepsilon_M = 2^{-52}$

- Note: $2^{-52} = (2^{10})^{-5.2} \approx (10^3)^{-5.2} = 10^{-15.6} \approx 10^{-16}$

- `.Machine$double.eps`

- Find gap between 2^{20} and next smallest representable number

Rounding Error

Example:

- $$(9.4)_{10} = (1001.\overline{0110})_2$$

Rounding Error

Example:

- $$(9.4)_{10} = (1001.\overline{0110})_2$$
- $$\text{fl}(9.4) = +1.\boxed{0010\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1101} * 2^3$$
- So we truncated the tail by $.\overline{1100} * 2^{-52} * 2^3 = .\overline{0110} * 2^{-51} * 2^3 = 0.4 * 2^{-48} = 0.8 * 2^{-49}$
- And by rounding up, we added $2^{-52} * 2^3 = 1.0 * 2^{-49}$
- Therefore $\text{fl}(9.4) = 9.4 + 0.2 * 2^{-49}$

Absolute Rounding Error:

- $|x_c - x| = |\text{fl}(9.4) - 9.4| = 0.2 * 2^{-49}$

Relative Rounding Error:

- $\frac{|x_c - x|}{|x|} = \frac{|\text{fl}(9.4) - 9.4|}{9.4} = \frac{0.2 * 2^{-49}}{9.4} = \frac{8}{47} * 2^{-52}$
- The relative rounding error is always less than or equal to $\frac{\epsilon_{mach}}{2}$

Addition and Subtraction of Floating Point Numbers

- There is a register dedicated to doing computations so that the additions can be done with higher precision
- To add/subtract two numbers:
  1. Line up decimals
  2. Do the addition/subtraction (in higher precision, so you don't need to worry about rounding yet)
  3. Once the arithmetic is complete, round to convert back to floating point form