Computer Architecture
Floating Points 2

Cain Susko
January 31, 2022

Queen's University
School of Computing

Floating Point Operations

there are 2 operations with floating points:

$$x +_f y = round(x + y)$$
$$x \times_f y = round\,(x + y)\,.$$

We first compute the exact result and then make it fit into the precision of a given w-bit numbers. When rounding we default to rounding to the closest **whole** number. If the number is equally distant from either whole number, round to the **even** number.

$$2.50 = 2$$

When dealing with binary numbers, if the value to in the middle of 2 possible values then round so that the least significant digit is even.

$$2.\frac{2}{32} = 10.00011_2 \to 10.00_2 = 2$$
$$2.\frac{3}{16} = 10.00110_2 \to 10.01 = 2.\frac{1}{4}$$

note the decimal numbers are in the form int.frac where frac is a representation of what the number past the decimal is.

Addition

the exact result of adding 2 floating point numbers in the form of IEEE float is as follows:

$$(-1)^{s_1}M_1 * 2^{E_1} + (-1)^{s_2}M_2 * 2^{E_2} = (-1)^{s}M * 2^{E}$$

such that s and M are the results of signed align and add as well as E is equal to E_1. We must also fix some things about his sum:

1. if $M \geq 2$, shift M right, increment E
2. if $M < 0$ shift M left k positions, decrement E by k.
3. overflow if E is out of range
4. round M to fit $frac$ precision.

This operation is **Closed, Commutative, not Associative, almost Invertible, almost Monotonous** the almost is because of infinity and NaN.

Multiplication

multiplying 2 IEEE floating point numbers is as follows:

$$(-1)^{s_1} M_1 * 2^{E_1} \times (-1)^{s_2} M_2 * 2^{E_2} = (-1)^{s} M * 2^{E}$$

Such that the sign value $s = s_1 * s_2$, the significand $M = M_1 \times M_2$, and $E = E_1 + E_2$. The conditions where we need to fix things with the product are as follows:

1. if $M \geq 2$, shift M right, increment E
2. if E is out of range, overflow
3. round M to fit $frac$ precision.

When implementing floating point multiplication the biggest job is multiplying significands.
This operation is **Closed, Commutative, not Associative, Invertible, not Distributive over addition, almost Monotonous**. The almost is because of infinities and NaN.

Float in C

a float in C Guarantees 2 levels of precision:

1. float-single precision
2. double-double presision

the conversions between these types and the int type are as follows:

- double$\vee$float $\rightarrow$ int
  truncated fractional part and rounds towards 0

- int $\rightarrow$double
  is an exact conversion as long as int has word size.

- int $\rightarrow$float
  will round according to rounding mode

Summary

casting signed to unsigned integers results in their bit patterns being maintained but reinterpereted. This can have the unexpected effect of adding or subtracting 2^w
When **Expanding** the general rules are:

- Unsigned: Zeros added
- Signed: sign extension (add s bit)
- Both yeild expected result

Similarly, when **Truncating** the general rules are:

- Unsigned$\vee$Signed: bits are truncated. Result reinterpreted
- Unsigned: mod operation
- Signed: similar to mod operation
- for small numbers: yields expected behaviour

Rules for **addition**

- Unsigned/Signed: normal addition followed by truncate, same operation on bit level.
- Unsigned: addition mod 2^w
- Signed: modified addition mod 2^w such that the result is in the proper range

Rules for **multiplication**

- Unsigned/Signed: normal multiplication followed by truncate, same operation at bit level.
- Unsigned: multiplication mod 2^w.
- Signed: multiplication mod 2^w within proper range.