COMP 264 Fall 2015 Lecture Notes

R. I. Greenberg

September 21, 2015

1 Floating Point

1.1 Basic Notions

- Used to represent rational values (fractions) across a wide numeric range.
- We've seen basic idea of fractional binary values with a *binary point* analogous to decimal point in base 10. (Simple decimal values may not be exactly representable, e.g., closest approx. to $.1_{10}$ in binary using 8 bits is $.00011010_2$, but converting this back to decimal would give us $.1015625_{10}$.)
- Note $.11\ldots1_2$ is just below 1; e.g., in 8-bits, we get 255/256. Shorthand notation: $1-\epsilon$.
- Next step: Use scientific notation, with three key elements to an FP number:
  - Sign (`0` for `+`; `1` for `-`)
  - Significand (also often called "mantissa").
  - Exponent

  E.g., a close 8-bit approximation to $.1_{10}$ would be $.00011001_2$ (not quite the best). Could say sign positive, significand 1.1001, and exponent -4, since

  $$.00011001_2 \text{ is } 1.1001_2 \times 2^{-4} .$$

- A concern: Many ways to represent same number e.g., change above ex. to significand .11001 and exponent is -3, or significand 11001 and exponent -8. Undesirable, e.g., makes equality tests more difficult. Can fix by requiring *normalized* representations, e.g., that significand should always begin with ".1".
- Key characteristics of representation: *precision* and *range*. Range is indicated by smallest and largest possible values. Precision is no. of bits in significand; many computers have two FP representations: *single precision* and *double precision*.

1.2 IEEE Floating-Point Standard

IEEE 754 floating point standard used in almost all modern computers. Ensures same computations on different computers (assuming proper implementation) should yield same results.

1.2.1 IEEE FP Representation

- Three fields with number of bits as indicated below (32 total for SP and 64 for DP); just look at top 2 rows of table for now.

Kind of value	Sign	Exponent	Significand
single precision	1 bit	8 bits (with bias 127)	23 bits
double precision	1 bit	11 bits (with bias 1023)	52 bits
$\pm\infty$	1 bit	$1\ldots1$	$0\ldots0$
NaN	X	$1\ldots1$	$X\ldots X \neq 0$
denormalized	1 bit	$0\ldots0$	23 or 52 bits

- Really 3 types of FP nos. (within each of SP and DP): normalized, denormalized, and special values.

Normalized Values

- Significand always begins with "1."; store only the rest.
- Exponent in non-negative notation but must subtract the *bias* as per top 2 lines of table above to get the true exponent.
- Exponents represented by all 0s and all 1s not allowed, since reserved for other kinds of numbers.

Special Values : Next two rows of table.

- Significand all 0s with exponent all 1s for $+\infty$ and $-\infty$. Can use for overflow.
- Any significand$\neq 0$ with exponent all 1s for NaN ("not a number").

Denormalized values : Last row of table above; exponent of all zeros.

- Entire significand stored; only implicit binary point to the left.

- Exponent interpreted as -126 for single precision or -1022 for double precision. Smooth transition: e.g., smallest normalized SP is 1×2^{-126}; largest denormalized SP is $(1 - \epsilon) \times 2^{-126}$.

- Two benefits to having denormalized as well as normalized:

  - Can represent 0. Actually have a $+0$ and a -0; same in an additive context, different in multiplicative.
  - Can represent numbers with smaller nonzero magnitude. In SP, get down to 2^{-149}; in DP, 2^{-1074}. Reduces the occurrence of *underflow* (too small a magnitude to represent).

1.2.2 Example Numbers

See text; also *03-float slides* 4–22 summarize material covered so far. Note interesting property that positive floating point numbers, viewed as an unsigned integer (0, then exponent, then stored significand), can be sorted by an integer sort. Same is true for negative FP numbers, and Bryant & O'Hallaran, 2nd ed., Exercise 2.83 on page 127 shows how to put these two pieces together.

1.2.3 Rounding

Whatever precision one uses, there will be real values within the range of the representation that cannot be represented. We've seen re underflow; i.e, can fall into gap between 0 and the nearest representable values. But can happen throughout the range, when fall into gap between two adjacent representable values.

For example, consider representation with 4-bit significand (implicit binary point at left) and 3-bit exponent. If multiply $.1111 \times 2^2$ by $.1111 \times 2^{-2}$, get $.11100001 \times 2^0$. (Decimal $\frac{15}{16} \times \frac{15}{16} = \frac{225}{256}$.) Result within range, but significand has too many bits. Must *round* the result to produce representable approximation. Nearest representable values are $.1110 \times 2^0$ and $.1111 \times 2^0$.

IEEE standard defines four rounding modes, with "round to nearest" as default:

- *Round to nearest* or *round to even* or *unbiased rounding*: Round to the nearest representable value. $.1110 \times 2^0$ in example above. (In case of tie, make choice with rightmost bit a 0.) Error of at most half the value of the LSB (least significant bit) of the rounded result; all the upcoming methods can be off by up to 1 LSB.

- *Round toward 0*: Easiest; just throw away extra bits of the significand. $.1110 \times 2^0$ in example above.

- *Round toward* $+\infty$: Called the *ceiling* function. $.1111 \times 2^0$ in example above.

- *Round toward* $-\infty$: Called the *floor* function. $.1110 \times 2^0$ in example above.

Can see *03-float slides* 25–27 for a summary of rounding.

All these methods, except round toward 0, require that arithmetic operations produce some extra bits beyond what will appear in the final representation:

- Most significant: the *round bit*.
- Next: the *guard bit*.
- A third bit, the *sticky bit*, may be used in "shift-add" algorithms for multiplication; we won't go that deep.

1.2.4 FP Operations

- For "ordinary" FP numbers, return same result as if did true computation in reals and then rounded.
- The rest of FP arithmetic can be specified with just two rules:
  - Any operation involving a NaN results in NaN.
  - For any operation involving $\pm\infty$, replace $\pm\infty$ with a variable and take the limit as the variable goes to $\pm\infty$; if result not well-defined, then NaN.
- A concrete implementation of above is only a little more extensive. e.g., for addition and subtraction:
  - Any operation involving NaN returns NaN.
  - $x + 0$, $0 + x$, and $x - 0$ all return x, even if x is $\pm\infty$.
  - $0 - x$ just changes the sign bit of x.
  - An addition of two ∞'s of the same sign is ∞ of the same sign; with two ∞'s of opposite sign, the result is NaN. Similar rules for subtractions.
- Commutativity holds for floating-point addition and multiplication as it does for the integer operations.
- Associativity: yes for integers; no for floating point. Some single precision examples:

```
3.14f+1e8f-1e8f  =>  0
3.14f+(1e8f-1e8f)  =>  3.14
1e20f*1e20f*1e-20f  =>  INF
1e20f*(1e20f*1e-20f)  =>  1e20
```

- Distributivity yes for integers; no for floating point. A single precision example

```
1e20f*(1e20f-1e20f)  =>  0
1e20f*1e20f-1e20f*1e20f  =>  NaN
```

- We won't cover all the low-level details of floating-point arithmetic. But let us just note that one major change in comparison to integer arithmetic is that, since operands may have different exponents, we may need to align their significands. For example (still using 4-bit significands following an implicit binary point and 3-bit exponents for illustration), if we want to do $.1010 \times 2^5 + .1100 \times 2^3$, the significands are just stored as 1010 and 1100. Adding the two significands as they are is completely useless. We need to shift one over, i.e., make the exponents match before adding, e.g., to reexpress as $10.10 \times 2^3 + .1100 \times 2^3$.
- A good (albeit lengthy) reference on floating point is available at `http://cr.yp.to/2005-590/goldberg.pdf`.

1.3 Floating Point in C

- Declare variables as `float` or `double` to get SP or DP.
- C standard does not require IEEE FP standard, so no standard method for choosing rounding method or setting variables to $\pm\infty$ or NaN.
- If you're interested, see details of conversions between int, float, and double in text.
- If you're interested, see how certain machines can lead to anomalous results because of use of extended precision for values held in registers.

2 Exam 1 9/28

End of Material for Exam 1. Closed book. Chapters 1–2; really all or almost all from Chapter 2. Sample exam forthcoming.