IEEE Floating Point Standard

CS 471/Math 471

Brian T. Smith, UNM

Andrew C. Pineda, UNM

What Is The IEEE FP Standard?

- There are actually two standards
  - binary standard (IEEE-754)
  - base-independent standard (IEEE-854)
- work started on the binary standard in 1976 and was published as a binary standard in early 80s
- work on the non-binary standard completed in 1987 (W.J. Cody led it as chairman)
- binary standard essentially is a special case of the base-independent standard

History of Floating Point Standards

- Research by W. Kahan at University of Toronto starting in early 60s on good floating point arithmetics
- IBM 360 hexadecimal base chopped arithmetic in 32 bits demonstrated to have poor accuracy characteristics in many paper in 70s
- Effort begun by Kahan to develop a standard for floating point CPUs (chips) and gain support for it through a standards body (IEEE)

Essence of the IEEE Standard

- Recommended formats for representing floating point binary numbers:
  - 2 storage formats
    - 32 bits with sign, exponent, fraction
    - 64 bits with sign, larger exponent, larger fraction
  - a CPU register format
    - at least 80 bits, sign, larger exponent than 64 bits, and a larger fraction (with no implied bit)

Essence Continued

- Specified the behavior of the arithmetic (accuracy, overflow, underflow, not numbers)
- Specified 4 rounding modes
  - unbiased rounding
  - chopping towards zero
  - rounding up towards positive infinity
  - rounding down towards negative infinity

32 Bit Format

- Representations
  - three cases
    - positive zero, all zero bits
    - negative zero, all but the first (sign) bit is zero
    - non-zero values

32 Bit Format Continued

- Non-zero values, say $x = s \times b^e \times 1.f$
  - sign bit is s
  - exponent is 127 + e
  - fraction is f
- The number 1.0 is the binary value
  - 0 11111111 00000000000000000000000
  - that is,
    - S=0, exponent=127, fraction = 0

Exercise

- Use Fortran 90 to print out the binary values of several different floating point numbers. Try 1.0, 1.625, 1625, -0.0001625. Print the binary representations of huge(1.0) and tiny(1.0). Do the results make sense?
  - This can be done with the following program fragment

```
real X; integer I; equivalence (X,I)
X = -1.0; print '(B33.32)', I
```


Special Numbers

- There are several special values (reserved exponents) that IEEE defines. They are:
  - zero -- 32 zero bits
  - -zero -- a 1 bit followed by 31 zero bits
  - infinity -- a 0 bit followed by eight 1 bits followed by 23 zero bits
  - -infinity -- a 1 bit followed by eight 1 bits followed by 23 zero bits
  - NaN (Not a Number) -- non-zero fraction with all 1 bits in the exponent, either positive or negative
  - denormalized numbers -- zero exponent, non-zero fraction

Range of Exponent Values

- Exponents that are:
  - all zero bits
  - all one bits
- are reserved for zeros, infinities, and NaNs
- The maximum IEEE exponent is thus 127
  - maximum number is nearly 2^{128}
- The minimum exponent is thus -126
  - minimum magnitude is 2^{-126}

Numerical Precision

- A measure of numerical precision is the smallest positive number which when added to 1.0 changes it.
- This value depends on the rounding mode.
- The term "machine relative precision", also called epsilon or eps, is this measure of precision when the rounding mode is chopping towards zero.

Numerical Precision Cont'd

- Epsilon for 32 bit format IEEE arithmetic is 2^{-23}
- If the definition used rounding, it would be 2^{-24}
- Epsilon is also the maximum relative spacing between representable numbers when chopping mode is used.

64 Bit IEEE Format

- 3 fields
  - sign, 1 bit, 0 for positive sign, 1 for negative
  - biased exponent, 11 bits, bias is 1023
  - fraction, 52 bits, with an implied bit
- Similar special values
- Non-zero values, $x = s \times b^e \times 1.f$
  - sign bit is s
  - exponent is 1023 + e
  - fraction is f

80 Bit Register Format

- 3 fields
  - sign, 1 bit, 0 for positive sign, 1 for negative
  - biased exponent, 15 bits, bias is 1023
  - fraction, 64 bits, with *no* implied bit
- No special values
- Provides a larger precision and larger exponent range than the 64 bit format
  - guard bits for improved accuracy in calculations
  - supports proper rounding to be performed with sticky bit

Infinities

- Results that are too large are represented as these special values.
  - Permit the processor to produce a value without trap or stopping to handle the special case of overflow
  - Produces by such operations as:
    - large + large
    - large * large
    - large / small
    - nonzero / zero
  - All of these results have a sign

Why NaNs?

- There are floating point operators that result in no proper value
  - zero / zero
  - infinity - infinity
  - NaN as an operand
- All of these operations produce a NaN

Floating Point Error Models

- With the IEEE arithmetic and some other machines, all floating point operations satisfy the following model
  - $fl(a \oplus b) = a \oplus b\,(1 + \eta)$ where
    - $\oplus$ is any basic operation (+, -, *, /) and
    - η is a value less than epsilon in magnitude
- Using this model, the propagation of rounding errors in algorithms can be analyzed.
  - LU factorization, QR factorization, etc.