Floating Point Arithmetic

Randy Bryant
CS 347

Jan. 20, 1998

Topics

- IEEE Floating Point Standard
- Rounding
- Floating Point Operations
- Mathematical properties

IEEE Floating Point

IEEE Standard 754

- **Estabilished in 1985 as uniform standard for floating point arithmetic**
  - Before that, many idiosyncratic formats
- **Supported by all major CPUs**

Driven by Numerical Concerns

- **Nice standards for rounding, overflow, underflow**
- **Hard to make go fast**
  - Numercial analysts predominated over hardware types in defining standard

Representation

Numerical Form

- $-1^s \; m \; 2^E$
  - Sign bit *s* determines whether number is negative or positive
  - Mantissa *m* normally a value in range [1.0,2.0).
  - Exponent *E* weights value by power of two

Encoding

- **MSB is sign bit**
- **Exp field encodes *E***
- **Significand field encodes *m***

Sizes

- **Single precision: 8 exp bits, 23 significand bits**
- **Double precision: 11 exp bits, 52 significand bits**

"Normalized" Numeric Values

Condition

- **exp ≠ `000…0` and exp ≠ `111…1`**

Exponent coded as *biased* value

E = Exp – Bias

- *Exp* : unsigned value denoted by exp
- *Bias* : Bias value
  - » Single precision: 127
  - » Double precision: 1023

Mantissa coded with implied leading 1

***m* = `1.xxx…x`$_2$**

- xxx…x: bits of significand
- Minimum when `000…0` (*m* = 1.0)
- Maximum when `111…1` (*m* = 2.0 – ε)

Denormalized Values

Condition

- **exp = `000…0`**

Value

- **Exponent value *E* = –*Bias* + 1**
- **Mantissa value *m* = `0.xxx…x`$_2$**
  - xxx…x: bits of significand

Cases

- **exp = `000…0`, `significand` = `000…0`**
  - Represents value 0
  - Note that have distinct values +0 and –0
- **exp = `000…0`, `significand` ≠ `000…0`**
  - Numbers very close to 0.0
  - Lose precision as get smaller
  - “Gradual underflow”

Interesting Numbers

Description	Exp	Significand	Numeric Value
Zero	00…00	00…00	0.0
Smallest Pos. Denorm. • Single ≈ 1.4×10^{-45} • Double ≈ 4.9×10^{-324}	00…00	00…01	$2^{-\{23,52\}} \times 2^{-\{126,1022\}}$
Largest Denormalized • Single ≈ 1.18×10^{-38} • Double ≈ 2.2×10^{-308}	00…00	11…11	$(1.0 - \varepsilon) \times 2^{-\{126,1022\}}$
Smallest Pos. Normalized	00…01	00…00	$1.0 \times 2^{-\{126,1022\}}$
One	01…11	00…00	1.0
Largest Normalized • Single ≈ 3.4×10^{38} • Double ≈ 1.8×10^{308}	11…10	11…11	$(2.0 - \varepsilon) \times 2^{\{127,1023\}}$

Special Values

Condition

- **exp = 111…1**

Cases

- **exp = 111…1, significand = 000…0**
  - Represents value ∞ (infinity)
  - Operation that overflows
  - Both positive and negative
- **exp = 111…1, significand ≠ 000…0**
  - Not-a-Number (NaN)
  - Represents case when no numeric value can be determined
  - E.g., 1.0/0.0, sqrt(–1), ∞ − ∞
  - No fixed meaning assigned to significand bits

Special Properties of Encoding

FP Zero Same as Integer Zero

- **All bits = 0**

Can (Almost) Use Unsigned Integer Comparison

- **Must first compare sign bits**
- **NaNs problematic**
  - Will be greater than any other values
  - What should comparison yield?
- **Otherwise OK**
  - Denorm vs. normalized
  - Normalized vs. infinity

Floating Point Operations

Conceptual View

- **First compute exact result**
- **Make it fit into desired precision**
  - Possibly overflow if exponent too large
  - Possibly round to fit into significand

Actual Implementation

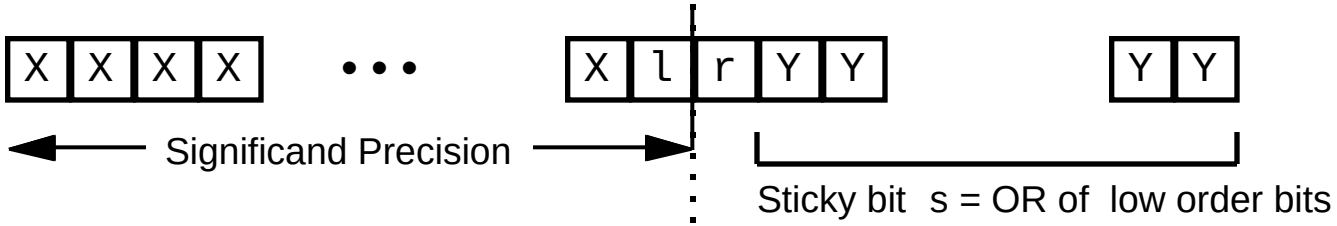
- **Tricks to avoid computing all low order bits**

Rounding Modes (illustrate with $ rounding)

	$1.40	**$1.60**	**$1.50**	**$2.50**	**–$1.50**
• **Zero**	$1.00	$2.00	$1.00	$2.00	–$1.00
• $-\infty$	$1.00	$2.00	$1.00	$2.00	–$2.00
• $+\infty$	$1.00	$2.00	$2.00	$3.00	–$1.00
• **Nearest Even (default)**	$1.00	$2.00	$2.00	$2.00	–$2.00

Implementing Rounding: Nonnegative

Format of Exact Result Significand

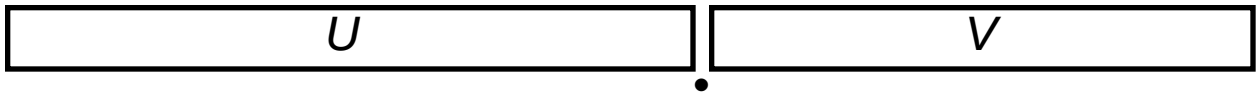


Numeric Value ($U \geq 0$, $V \geq 0$)

r	s	l	*U*	*V*
0	–	–	–	< 0.5
1	1	–	–	> 0.5
1	0	0	even	= 0.5
1	0	1	odd	= 0.5

Shifted Binary Pont

Rounding Rules: Nonnegative

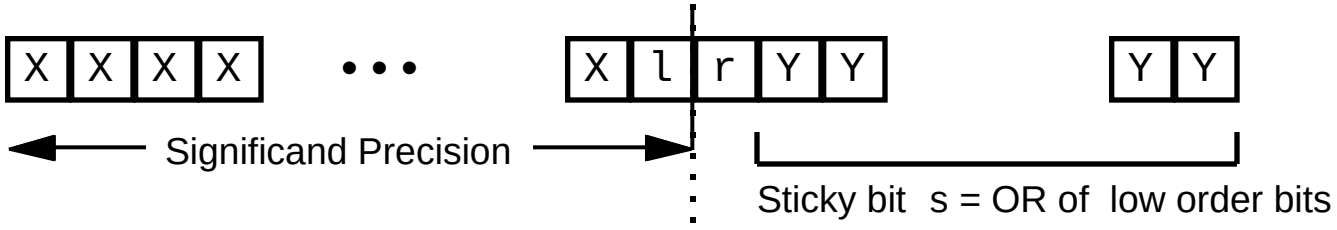


Rounded Significand (*U* ≥ 0)

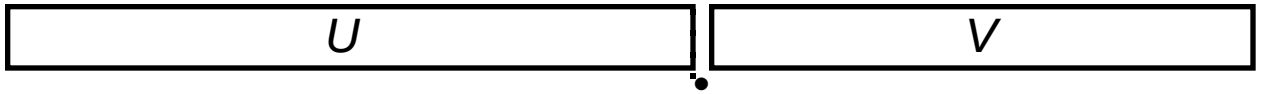
r	s	l	*U*	*V*	RT0	RT+Inf	RT–Inf	RTE
0	–	–	–	< 0.5	*U*	*U*	*U*	*U*
1	1	–	–	> 0.5	*U*+1	*U*+1	*U*+1	*U*+1
1	0	0	even	= 0.5	*U*	*U*+1	*U*	*U*
1	0	1	odd	= 0.5	*U*	*U*+1	*U*	*U*+1

Implementing Rounding: Negative

Format of Exact Result Significand

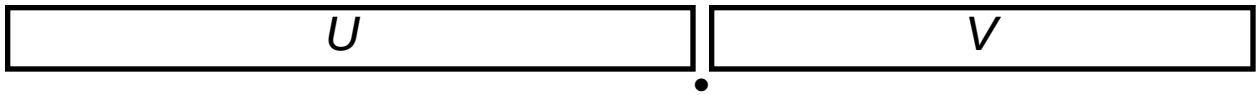


Numeric Value ($U < 0$, $V \leq 0$)



r	s	l	*U*	*V*
0	–	–	–	> –0.5
1	1	–	–	< –0.5
1	0	0	even	= –0.5
1	0	1	odd	= –0.5

Rounding Rules: Negative



Rounded Significand (*U* < 0, *V* < 0)

r	s	l	*U*	*V*	RT0	RT+Inf	RT–Inf	RTE
0	–	–	–	> –0.5	*U*	*U*	*U*	*U*
1	1	–	–	< –0.5	*U*–1	*U*–1	*U*–1	*U*–1
1	0	0	even	= –0.5	*U*	*U*	*U*–1	*U*
1	0	1	odd	= –0.5	*U*	*U*	*U*–1	*U*–1

FP Multiplication

Operands

$(-1)^{s1}\ m1\ \ 2^{E1}$

$(-1)^{s2}\ m2\ \ 2^{E2}$

Exact Result

$(-1)^{s}\ m\ \ 2^{E}$

- **Sign** s**:** $s1 \neq s2$
- **Mantissa** m**:** $m1 * m2$
- **Exponent** E**:** $E1 + E2$

Fixing

- **Overflow if** E **out of range**
- **Round** m **to fit significand precision**

Implementation

- **Biggest chore is multiplying mantissas**

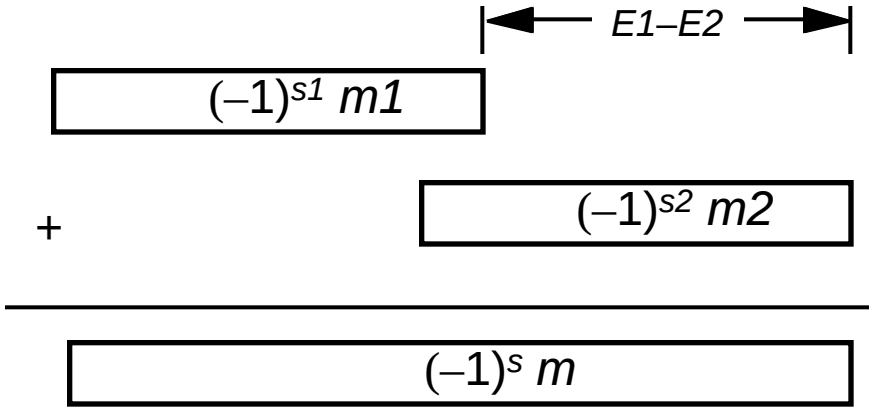
FP Addition

Operands

$(-1)^{s1}\ m1\ \ 2^{E1}$

$(-1)^{s2}\ m2\ \ 2^{E2}$

- **Assume** $E1 \geq E2$



Exact Result

$(-1)^{s}\ m\ \ 2^{E}$

- **Sign** s**, mantissa** m**:**
  - Result of signed align & add
- **Exponent** E**:** $E1 - E2$

Fixing

- **Shift** m **right, increment** E **if** $m \geq 2$
- **Shift** m **left** k **positions, decrement** E **by** k **if** $m < 1$
- **Overflow if** E **out of range**
- **Round** m **to fit significand precision**

Mathematical Properties of FP Add

Compare to those of Abelian Group

- **Closed under addition?** **YES**
  - But may generate infinity or NaN
- **Commutative?** **YES**
- **Associative?** **NO**
  - Overflow and inexactness of rounding
- **0 is additive identity?** **YES**
- **Every element has additive inverse** **ALMOST**
  - Except for infinities & NaNs

Montonicity

- $a \leq b \Longrightarrow a+c \leq b+c$**?** **ALMOST**
  - Except for infinities & NaNs

Algebraic Properties of FP Mult

Compare to Commutative Ring

- **Closed under multiplication?** **YES**
  - But may generate infinity or NaN
- **Multiplication Commutative?** **YES**
- **Multiplication is Associative?** **NO**
  - Possibility of overflow, inexactness of rounding
- **1 is multiplicative identity?** **YES**
- **Multiplication distributes over addtion?** **NO**
  - Possibility of overflow, inexactness of rounding

Montonicity

- $a \leq b \;\&\; c \geq 0 \implies a*c \leq b*c$**?** **ALMOST**
  - Except for infinities & NaNs

Floating Point in C

C Supports Two Levels

`float`	**single precision**
`double`	**double precision**

Conversions

- **Casting between `int`, `float`, and `double` changes numeric values**
- **Double or float to int**
  - Truncates fractional part
  - Like rounding toward zero
  - Not defined when out of range
    - » Generally saturates to TMin or TMax
- **int to double**
  - Exact conversion, as long as int has ≤ 54 bit word size
- **int to float**
  - Will round according to rounding mode

Floating Point Puzzles

- **For each of the following C expressions, either:**
  - Argue that is true for all argument values
  - Give example where not true

```
int x = …;
float f = …;
double d = …;
```

Assume neither d nor f is NAN

- `x == (float) x`
- `x == (double) x`
- `f == (double) f`
- `d == (float) d`
- `f == -(-f);`
- `2/3 == 2/3.0`
- `d < 0.0` `((d*2) < 0.0)`
- `d > f` `-f < -d`
- `d * d >= 0.0`
- `x > 0 && y > 0` `x + y > 0`