CS429: Computer Organization and Architecture
Floating Point

Warren Hunt, Jr. and Bill Young
Department of Computer Sciences
University of Texas at Austin

Last updated: August 26, 2014 at 08:53

Topics of this Slideset

- IEEE Floating Point Standard
- Rounding
- Floating point operations
- Mathematical properties

Floating Point Puzzles

For each of the following C expressions, either:

- argue that it is true for all argument values, or
- explain why it is not true.

```c
int x = ...;
float f = ...;
double d = ...;
```

Assume neither d nor f is NaN.

```
x == (int)(float) x
x == (int)(double) x
f == (float)(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
(d+f)-d == f
```

IEEE Floating Point Standard

IEEE Standard 754

- Established in 1985 as a uniform standard for floating point arithmetic
- It is supported by all major CPUs.
- Before 1985 there were many idiosyncratic formats.

Driven by Numerical Concerns

- Nice standards for rounding, overflow, underflow
- Hard to make go fast: numerical analysts predominated over hardware types in defining the standard
- Now all (add, subtract, multiply) operations are fast except divide.

Fractional Binary Numbers

The binary number $b_i b_{i-1} b_2 b_1 \ldots b_0 . b_{-1} b_{-2} b_{-3} \ldots b_{-j}$ represents a particular sum. Each digit is multiplied by a power of two according to the following chart:

Bit:	b_i	b_{i-1}	$\ldots$	b_2	b_1	b_0	$\cdot$	b_{-1}	b_{-2}	b_{-3}	$\ldots$	b_{-j}
Weight:	2^i	2^{i-1}	$\ldots$	4	2	1	$\cdot$	$1/2$	$1/4$	$1/8$	$\ldots$	2^{-j}

Representation:

- Bits to the right of the *binary point* represent fractional powers of 2.
- This represents the rational number:

$$\sum_{k=-j}^{i} b_k \times 2^k$$

Fractional Binary Numbers: Examples

Value	**Representation**
$5 + 3/4$	101.11_2
$2 + 7/8$	10.111_2
$63/64$	0.111111_2

Observations

- Divide by 2 by shifting right
- Multiply by 2 by shifting left
- Numbers of the form $0.11111\ldots_2$ are just below 1.0
  - $1/2 + 1/4 + 1/8 + \ldots + 1/2^i \rightarrow 1.0$
  - We use the notation $1.0 - \epsilon$.

Representable Numbers

Limitation

- You can only represent numbers of the form $y + x/2^i$.
- Other fractions have repeating bit representations

Value	**Representation**
$1/3$	$0.0101010101[01]\ldots_2$
$1/5$	$0.001100110011[0011]\ldots_2$
$1/10$	$0.0001100110011[0011]\ldots_2$

Numerical Form

$$-1^s \times M \times 2^E$$

- Sign bit s determines whether number is negative or positive.
- Significand M is normally a fractional value in the range $[1.0 \ldots 2.0)$
- Exponent E weights value by power of two.

Encoding

s	exp	frac

- The most significant bit is the sign bit.
- The `exp` field encodes E.
- The `frac` field encodes M.

Floating Point Precisions

Encoding

s	exp	frac

- The most significant bit is the sign bit.
- The `exp` field encodes E.
- The `frac` field encodes M.

Sizes

- Single precision: 8 exp bits, 23 frac bits, for 32 bits total
- Double precision: 11 exp bits, 52 frac bits, for 64 bits total
- Extended precision: 15 exp bits, 63 frac bits
  - Only found in Intel-compatible machines
  - Stored in 80 bits: an explicit "1" bit appears in the format, except when exp is 0.

Normalized Numeric Values

Condition: exp $\neq 000\ldots0$ and exp $\neq 111\ldots1$

Exponent is coded as a biased value

$E = Exp - Bias$

- Exp: unsigned value denoted by `exp`.
- $Bias$: Bias value
  - Single precision: 127 (Exp: $1\ldots254$, $E: -126\ldots127$)
  - Double precision: 1023 (Exp: $1\ldots2046$, $E: -1022\ldots1023$)
  - In general: $Bias = 2^{e-1} - 1$, where e is the number of exponent bits

Significand coded with implied leading 1

$M = 1.xxx\ldots x_2$

- $xxx\ldots x$: bits of `frac`
- Minimum when $000\ldots0$ ($M = 1.0$)
- Maximum when $111\ldots1$ ($M = 2.0 - \epsilon$)
- We get the extra leading bit "for free."

Normalized Encoding Example

Value:

`float F = 15213.0;`

$$15231_{10} = 11101101101101_2 = 1.1101101101101_2 \times 2^{13}$$

Significand

$\mathrm{M} = 1.1101101101101_2$

$\texttt{frac} = 11011011011010000000000$

Exponent

$\mathrm{E} = 13$

$\mathrm{Bias} = 127$

$\mathrm{Exp} = 140 = 10001100$

Normalized Encoding Example

Floating Point Representation

Hex: 466DB400

Binary: 0100 0110 0110 1101 1011 0100 0000 0000

140:	100 0110 0
15213:	1110 1101 1011 01

Denormalized Values

Condition: `exp = 000...0`

Value

- Exponent values: E = -Bias + 1 Why this value?
- Significand value: M = $0.xxx\ldots x_2$, where $xxx\ldots x$ are the bits of frac.

Cases

- exp = $000\ldots0$ and frac = $000\ldots0$
  - represents values of 0
  - notice that we have distinct +0 and -0
- exp = $000\ldots0$ and frac $\neq 000\ldots0$
  - These are numbers very close to 0.0
  - Lose precision as they get smaller
  - Experience “gradual underflow”

Special Values

Condition: `exp = 111...1`

Cases

- exp $= 111\ldots1$ and frac $= 000\ldots0$
  - Represents value of infinity (∞)
  - Result returned for operations that overflow
  - Sign indicates positive or negative
  - E.g., $1.0/0.0 = -1.0/-0.0 = +\infty$, $1.0/-0.0 = -\infty$
- exp $= 111\ldots1$ and frac $\neq 000\ldots0$
  - Not-a-Number (NaN)
  - Represents the case when no numeric value can be determined
  - E.g., sqrt(-1), $\infty - \infty$

Tiny Floating Point Example

8-bit Floating Point Representation

- The sign bit is in the most significant bit.
- The next four bits are the exponent with a bias of 7.
- The last three bits are the frac.

This has the general form of the IEEE Format

- Has both normalized and denormalized values.
- Has representations of 0, NaN, infinity.

7	6 … 3	2 … 0
s	exp	frac

Values Related to the Exponent

Exp	exp	E	2^E	comment
0	0000	-6	1/64	(denorms)
1	0001	-6	1/64	
2	0010	-5	1/32	
3	0011	-4	1/16	
4	0100	-3	1/8	
5	0101	-2	1/4	
6	0110	-1	1/2	
7	0111	0	1	
8	1000	+1	2	
9	1001	+2	4	
10	1010	+3	8	
11	1011	+4	16	
12	1100	+5	32	
13	1101	+6	64	
14	1110	+7	128	
15	1111	n/a		(inf, NaN)

Dynamic Range

	s	exp	frac	E	Value	
Denormalized numbers	0	0000	000	-6	0	
	0	0000	001	-6	$1/8 \times 1/64 = 1/512$	closest to zero
	0	0000	010	-6	$2/8 \times 1/64 = 2/512$	
				...		
	0	0000	110	-6	$6/8 \times 1/64 = 6/512$	
	0	0000	111	-6	$7/8 \times 1/64 = 7/512$	largest denorm
Normalized numbers	0	0001	000	-6	$8/8 \times 1/64 = 8/512$	smallest norm
	0	0001	001	-6	$9/8 \times 1/64 = 9/512$	
				...		
	0	0110	110	-1	$14/8 \times 1/2 = 14/16$	
	0	0110	111	-1	$15/8 \times 1/2 = 15/16$	closest to 1 below
	0	0111	000	0	$8/8 \times 1 = 1$	
	0	0111	001	0	$9/8 \times 1 = 9/8$	closest to 1 above
	0	0111	010	0	$10/8 \times 1 = 10/8$	
				...		
	0	1110	110	7	$14/8 \times 128 = 224$	
	0	1110	111	7	$15/8 \times 128 = 240$	largest norm
	0	1111	000	n/a	∞	

Interesting FP Numbers

Description	exp	frac	Numeric value
Zero	$00 \ldots 00$	$00 \ldots 00$	0.0
Smallest Pos. Denorm	$00 \ldots 00$	$00 \ldots 01$	$2^{\{-23,-52\}} \times 2^{\{-126,-1022\}}$

- Single $\approx 1.4 \times 10^{-45}$
- Double $\approx 4.9 \times 10^{-324}$

Largest Denorm.	$00 \ldots 00$	$11 \ldots 11$	$(1.0 - \epsilon) \times 2^{\{-126,-1022\}}$

- Single $\approx 1.18 \times 10^{-38}$
- Double $\approx 2.2 \times 10^{-308}$

Smallest Pos. Norm.	$00 \ldots 01$	$00 \ldots 01$	$1.0 \times 2^{\{-126,-1022\}}$

- Just larger than the largest denomalized.

One	$01 \ldots 11$	$00 \ldots 00$	1.0
Largest Norm.	$11 \ldots 11$	$11 \ldots 11$	$(2.0 - \epsilon) \times 2^{\{127,1023\}}$

- Single $\approx 3.4 \times 10^{38}$
- Double $\approx 1.8 \times 10^{308}$

Special Properties of Encoding

FP Zero is the Same as Integer Zero: All bits are 0.

Can (Almost) Use Unsigned Integer Comparison

- Must first compare sign bits.
- Must consider $-0 = 0$.
- NaNs are problematic:
  - Will be greater than any other values.
  - What should the comparison yield?
- Otherwise, it's OK.
  - Denorm vs. normalized works.
  - Normalized vs. infinity works.

Floating Point Operations

Conceptual View

- First compute the exact result.
- Make it fit into the desired precision.
  - Possibly overflows if exponent is too large.
  - Possibly round to fit into frac.

Rounding Modes (illustrated with $ rounding)

	$1.40	**$1.60**	**$1.50**	**$2.50**	**-$1.50**
Zero	$1	$1	$1	$2	-$1
Round down ($-\infty$)	$1	$1	$1	$2	-$2
Round up ($+\infty$)	$2	$2	$2	$3	-$1
Nearest even (default)	$1	$2	$2	$2	-$2

1. Round down: rounded result is close to but no greater than true result.
2. Round up: rounded result is close to but no less than true result.

Closer Look at Round to Even

Default Rounding Mode

- Hard to get any other kind without dropping into assembly.
- All others are statistically biased; the sum of a set of integers will consistently be under- or over-estimated.

Applying to Other Decimal Places / Bit Positions

When exactly halfway between two possible values, round so that the least significant digit is even.

E.g., round to the nearest hundredth:

1.2349999	1.23	Less than half way
1.2350001	1.24	Greater than half way
1.2350000	1.24	Half way, round up
1.2450000	1.24	Half way, round down

Rounding Binary Numbers

Binary Fractional Numbers

- "Even" when least significant bit is 0.
- Half way when bits to the right of rounding position = $100\ldots_2$.

Examples

E.g., Round to nearest 1/4 (2 bits to right of binary point).

Value	Binary	Rounded	Action	Rounded Value
2 2/32	10.00011_2	10.00	(< 1/2: down)	2
2 3/16	10.00110_2	10.01	(> 1/2: down)	2 1/4
2 7/8	10.11100_2	11.00	(1/2: up)	3
2 5/8	10.10100_2	10.10	(1/2: down)	2 1/2

FP Multiplication

Operands: $(-1)^{S_1} \times M_1 \times 2^{E_1}, (-1)^{S_2} \times M_2 \times 2^{E_2}$

Exact Result: $(-1)^S \times M \times 2^E$

- Sign S: S_1 xor S_2
- Significant M: $M_1 \times M_2$
- Exponent E: $E_1 + E_2$

Fixing

- If $M \geq 2$, shift M right, increment E
- E is out of range, overflow
- Round M to fit frac precision

Implementation

Biggest chore is multiplying significands.

FP Addition

Operands: $(-1)^{S_1} \times M_1 \times 2^{E_1}, (-1)^{S_2} \times M_2 \times 2^{E_2}$
Assume $E_1 > E_2$

Exact Result: $(-1)^S \times M \times 2^E$

- Sign S, Significant M; result of signed align and add.
- Exponent E: E_1

Fixing

- If $M \geq 2$, shift M right, increment E
- If $M < 1$, shift M left k positions, decrement E by k
- if E is out of range, overflow
- Round M to fit frac 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, because of overflow and inexactness of rounding.
- 0 is additive identity? Yes.
- Every element has additive inverse? Almost, except for infinities and NaNs.

Monotonicity

- $a \geq b \Longrightarrow a + c \geq b + c$? Almost, except for infinities and NaNs.

Mathematical Properties of FP Mult

Compare to those of Commutative Ring

- Closed under multiplication? Yes, but may generate infinity or NaN.
- Multiplication Commutative? Yes.
- Multiplication is Associative? No, because of possible overflow and inexactness of rounding.
- 1 is multiplicative identity? Yes.
- Multiplication distributes over addition? No, because of possible overflow and inexactness of rounding.

Monotonicity

- $a \geq b \,\&\, c \geq 0 \Longrightarrow a \times c \geq b \times c$? Almost, except for infinities and NaNs.

Floating Point in C

C guarantees two levels

- float: single precision
- double: double precision

Conversions

- Casting among 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 $\leq$ 53-bit word size
- int to float: will round according to rounding mode.

Answers to FP Puzzles

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

Assume neither d nor f is NaN.

`x == (int)(float) x`		No: 24 bit significand
`x == (int)(double) x`		Yes: 53 bit significand
`f == (float)(double) f`		Yes: increases precision
`d == (float) d`		No: loses precision
`f == -(-f)`		Yes: just change sign bit
`2/3 == 2/3.0`		No: $2/3 == 0$
`d < 0.0`	→ `((d*2) < 0.0)`	Yes
`d > f`	→ `-f < -d`	Yes
`d*d >= 0.0`		Yes
`(d+f)-d == f`		No: not associative

Ariane 5

On June 4, 1996 an unmanned Ariane 5 rocket launched by the European Space Agency exploded just forty seconds after its lift-off from Kourou, French Guiana. The rocket was on its first voyage, after a decade of development costing $7 billion. The destroyed rocket and its cargo were valued at $500 million. The cause of the failure was a software error in the inertial reference system. Specifically a 64-bit floating point number relating to the horizontal velocity of the rocket with respect to the platform was converted to a 16-bit signed integer. The number was larger than 32,767, the largest integer storeable in a 16-bit signed integer, and thus the conversion failed.

IEEE Floating Point has Clear Mathematical Properties

- Represents numbers of the form $M \times 2^E$.
- Can reason about operations independent of implementation: as if computed with perfect precision and then rounded.
- Not the same as real arithmetic.
  - Violates associativity and distributivity.
  - Makes life difficult for compilers and serious numerical application programmers.