Floating Point Puzzles

Solution

Terminology & Theory

Systems@ETH zürich

- Numerical Form: $f = (-1)^s * M * 2^E$
  - s: **sign bit**, set if number is negative
  - M: **significand** or **mantissa**
  - E: **exponent**

- Encoding:
  - Most significant bit is the sign bit **s**
  - **exp** field encodes E **(but is not equal to E)**
  - **frac** field encodes M **(but is not equal to M)**

s	exp	frac
1	8	23

Terminology & Theory

- x: **int** (4 bytes)
- f: **float** (4 bytes)
- d: **double** (8 bytes)

Q1: x == (int)(float) x

Systems@ETH Zürich

0) Assume x = INT_MAX = 0x7fffffff = 2147483647

1) Binary representation of this integer is
0111'1111'1111'1111'1111'1111'1111'1111

2) Bring binary representation into the format 1.XX....X by right-shifting:
1.11111111111111111111111111111 * 2^30

3) E = 30, Mantissa = 1.11111111111111111111111111111

4) Mantissa has 30 bits after the comma, but frac is only 23 bits long, so we need to round:
- Guard bit G (23rd bit) is 1
- Round bit R (24th bit) is 1
- OR of remaining bits (after the 24th bit) is 1
- Thus: > 0.5, round up

Q1: x == (int)(float) x

Systems@ETH Zürich

4) Mantissa has 30 bits after the comma, but frac is only 23 bits long, so we need to round:
   - Guard bit G (23rd bit) is 1
   - Round bit R (24th bit) is 1
   - OR of remaining bits (after the 24th bit) is 1
   - Thus: > 0.5, round up

5) New Mantissa (rounded up): 10.00000000000000000000000

6) Bring to format 1.XX..X again by right-shifting once:
   1.00000000000000000000000

7) E = 31, Mantissa = 1.00000000000000000000000

8) exp = E + Bias = 31 + 127 = 158 = 10011110 binary

9) frac (mantissa with implicit 1 at start) = 00000000000000000000000 binary

Q1: x == (int)(float) x

Systems@ETH zürich

9) frac (mantissa with implicit 1 at start) = 00000000000000000000000 binary

10) sign bit is 0 (value is not negative)

Yields:

0	10011110	00000000000000000000000

11) Cast back to integer, E = 158 - 127 = 31, Significand = 1.0, sign = 0:

$$\text{Value} = 1.0 * 2^{31} = 0x80000000$$

Conclusion: 0x7fffffff != 0x80000000, this equation does not hold for all values of x.

Q1: x == (int)(float) x

Or we can use the pigeonhole principle: Both integers and floats contain 32 bits of information. Thus we can at most represent 2^{32} different numbers with these types.

If we could represent every integer with floats, we would already need all 2^{32} possible binary representations of floating point values. Since we can also represent the value 0.5 using floats, we have a contradiction. This means that at least 1 integer can't be represented, and thus the equation does not hold for all integers.

Q2: x == (int)(double) x

Similar as Q1, however: the frac part of a double is 52 bits long. It's impossible to fill the whole frac part with an integer (max 30 bits), which means we never have to round.

Without rounding, there is no information loss, and casting back does not cause any information loss either.

This equation is true for all values of x.

Q3: f == (float)(double) f

By casting the double to a float, we simply gain more frac and exp bits; we're not losing any information.

Casting back to a float removes the unused frac and exp bits, again without loss of information.

This equation is true for all values of f.

Q4: d == (float) d

1) Assume every frac bit of the double is set to 1.

2) When we cast the value to a float, we lose 52-23 = 29 bits of information due to rounding.

3) When we cast back to a double, the 29 new frac bits we get will be filled with zeros.

This equation does not hold for all values of d.

Q5: f == -(-f)

1) The first negation changes the sign flag.

2) The second negation changes the sign flag again.

This equation holds for all values of f.

Note: Also true for doubles, as the same concept applies.

Q6: 2/3 == 2/3.0

1) The left side of the equation is an integer division (rounded down), so we get 2/3 = 0

2) The right side of the equation contains a double in the division. The integer value 2 will be promoted / casted to a double.
   The result of 2.0 / 3.0 is not 0.0

The equation is always false.

Q7: d < 0.0 => ((d*2) < 0.0)

1) d < 0.0 implies sign bit is set.

2) The value 2 will be promoted / casted to a double.

3) resulting sign bit = sign_bit_of(d) XOR sign_bit_of(2.0) = 1 XOR 0 = 1

4) The resulting value z = d*2 will have its sign bit set, which implies that z < 0.0

This implication is true for all values of d.

Q8: d > f => -f > -d

1) Comparison will promote the float to a double, so this implication is equivalent to: d1 > d2 => -d2 > -d1 for doubles d1 and d2

2) Multiply both sides of -d2 > -d1 with -1.0, yields d1 > d2, which is equivalent to the left side.

This implication is true for all values of f and d.

Q9: d * d >= 0.0

1) Let z := d * d

2) sign_bit_of(z) = sign_bit_of(d) XOR sign_bit_of(d) = 0

3) The resulting value z has its sign bit not set, which implies z >= 0.0

This equation holds for all values of d.

Q10: (d+f)-d == f

Systems@ETH zürich

1) Assume d = 0x7fefffffffffffff: sign(d) = 0, exp(d) = 11111111110,
   E(d) = 2046 - 1023 = 1023, and mantissa(d) = 1.1111....1

2) Assume f = 0x00800000: sign(f) = 0, exp(f) = 00000001,
   E(f) = 1 - 127 = -126, mantissa(f) = 1.0000....0

3) d+f promotes f to a double, which won't change the value of f

4) E(d) - E(f) = 1023 - (-126) = 1149. We have to add this value to the exponent of f so that both values have the same exponent. Adding this huge value to E(f) means lots and lots of right-shifts. The result:
   0.000000000..lots of zeros..10000..0 * 2^1023

5) Now we add the mantissa to the mantissa of d. Since nearly 1023 bits of the new mantissa of f are 0, the first 52 bits are 0 for sure, so we're essentially adding nothing to the mantissa of d!

Q10: (d+f)-d == f

Systems@ETH zürich

5) Now we add the mantissa to the mantissa of d. Since nearly 1023 bits of the new mantissa of f are 0, the first 52 bits are 0 for sure, so we're essentially adding nothing (zero) to the mantissa of d!

6) So we actually get: d+f == d

7) Now we subtract d from this value, so we get: d-d = 0

8) The equation now looks like this: 0 == f

Since f is not 0, this equation does not hold for all values of f and d.