ECE 271 – Microcomputer Architecture and Applications Lecture 18

Vince Weaver

http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

29 March 2022

Announcements

- Read Chapter 12
- Midterm approaching again, April 12th?
- Office hours might be irregular due to faculty interviews

Limits of Integer Math

- Signed 32-bit numbers:
  Max: 2,147,483,647 Min:-2,147,483,648
- What if you want larger numbers?
- What if you want fractions or decimals?

Place Value

- How does it work in base 10?
  $1234.56 = 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0 + 5 \times 10^{-1} + 6 \times 10^{-2}$
- You can do the same thing in binary (base2)
  $1010.10 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2}$
  This is 10.5 in decimal
- You can do this for arbitrary bases.
  You have to keep track of the decimal or “radix” point

Floating Point

- Tradeoff between range and precision
- Can express very large or very small numbers, but there are gaps with numbers that can't be represented

IEEE 754 Floating Point

- Standard from 1985
- Before this computers often had custom, incompatible floating point
- Prior to the 1990s was often an optional feature, possibly a separate FPU chip
- What can you do if you lack floating point unit? Emulate in software?

Floating Point Layout

- $(-1)^{sign}(1.fraction) \times 2^{exponent-bias}$
- bias offsets so that the value is always positive (makes for faster comparisons, no messing with twos complement)

Floating Point Sizes

- Single precision (32 bit) (float in C)
  Sign=1, Exponent=8, Fraction=23, Bias=127
- Double precision (64 bit) (double in C)
  Sign=1, Exponent=11, Fraction=52, Bias=1023
- Half precision (16 bit)
  Sign=1, Exponent=5, fraction=10, Bias=15
- Intel x86 (80 bits)
- Why have smaller sizes? They take up less room and are faster.

Floating Point Range

- Any integer with absolute value less than 24 bits can be expressed losslessly in float
- Any integer with absolute value less than 53 bits can be expressed losslessly in double

Floating Point Conversion Examples

There are some in the textbook too

Binary FP to Decimal

- You have the value `0xc1ff0000`. If it's a 32-bit floating point value, what is the decimal equivalent?
- `1100 0001 1111 1111 0000 0000 0000 0000`
- Sign bit is 1 (negative)
- Exponent is 8 bits, `1000 0011` which is 131
- Fraction is `1111 1110 0000 0000 0000 000`
  so 0.1111 111 $1/2 + 1/4 + 1/8 + 1/16 + 1/32 + 1/64 + 1/128 = 0.9921875$
- $f = (-1)^S \times (1 + fraction) \times 2^{exponent-127}$

- $f = (-1)^{-1} \times (1 + 0.9921875) \times 2^{131-127}$
- $f = -1 \times 1.9921875 \times 2^{4}$
- $f = -31.875$

Convert decimal FP to binary

- Want to convert 6.125 to binary
- **Sign=0**
- Need to divide or multiply by 2 until it is greater than one but less than 2
- $6.125/2 = 3.0625$
- $3.0625/2 = 1.53125$
- so $6.125 = 1.53125 * 2^2$
- so **exponent = 2+BIAS = 2+127=129 = 1000 0001**

- convert fraction (.53125) to binary
- repeatedly multiply fraction by 2
  - 0.53125*2 = 1.0625 **1**
  - 0.625*2= 0.125 **0**
  - 0.125*2= 0.25 **0**
  - 0.25*2 = 0.5 **0**
  - 0.5*2 = 1.0 **1**
  - so **fraction = 100 0100 0000 0000 0000 0000**
- so 32-bit float = 0100 0000 1100 0100 0000 0000 0000 0000
- 0x40C4 0000

- What would it look like in 64 bit?

Floating Point Conversion Examples

How to do this? Memcpy? unions? Flirting with undefined behavior?

```
int main(int argc, char **argv) {

        float f;
        unsigned int x;

        x=0x40c40000;
        memcpy(&f,&x,sizeof(float));
        printf("%x␣is␣%g\n",x,f);
        f=;
        memcpy(&x,&f,sizeof(float));
        printf("%x␣is␣%g\n",x,f);
        return 0;
}
```

40c40000 is 6.125 c1ff0000 is -31.875

Special Values

- Zero ... cannot represent in standard form (why? because it has to be 1.x in the mantissa). Special value, all zeros. Positive and Negative zeros.
- Positive and negative infinity: all exponent bits are 1s, mantissa all zeros
- NaN (not a number).
  Exponent all 1s, mantissa non-zero
  Things like 0/0, sqrt(-1), log(-1). Two types: QNaN (quiet) which does not cause an exception, and SNaN

(signaling) that cause an exception that needs o be handled.

Overflow and Underflow

- What's the smallest number you can represent? Smallest exponent 0b00000001, fraction 0000, so $(-1)^S \times (1+0) \times 2^{1-127} = \pm 1.18 \times 10^{-38}$
- What's the largest number? Maximum exponent 0b11111110 and mantissa all 1s $(-1)^S \times (1+1-2^{-23}) \times 2^{254-127} = \pm 3.40 \times 10^{38}$
- What is underflow? Too small but not zero?
- Overflow, too large to be represented

Subnormal Numbers

- Numbers between smallest and zero
- If exponent is 0, treat leading digit in mantissa as 0 instead of 1
- Can get down to 1.45×10^{-45}

Floating Point Comparison

- `float f = (5.0 - 1.0/7.0) + (1.0/7.0);`
- `if (f==5.0) printf("Five␣exactly.\n")`
- May work may not. Best way is to have some error (epsilon) and compare if absolute value less than a number.

Special Comparison

- +/- 0 compare equal
- Every NaN not equal to anything, not even itself
- All numbers are greater than -inf but smaller than inf

Floating Point Rounding Rules

- Complex mess, can cause interesting issues
- Especially as in floating point there are extra bits usually kept around for accuracy, but they are rounded off when written out to memory and you have to fit exactly in 32 or 64 bits
- IEEE-754
  - Round to nearest

    What do we do if *exactly* in between?

    round to even

guard bit, round bit, sticky bits

- Round toward zero (truncate)
- Round to $+$inf (round up)
- Round toward -inf (round down)

Floating Point Addition

- Shift smaller fraction to match larger one
- add or subtract based on sign bits
- normalize the sum
- round to appropriate bits
- detect overflow and underflow
- do example
- decimal, $5.2 * 10^0 + 9.5 * 10^1$
  - 0.52*10**1
  - 9.50*10**1

- 10.02
- $1.002 * 10^2$

Floating Point Multiplication

- Identify the sign
- add the exponents
- multiply the fractions (including leading hidden one)
- normalize the results

Transcendentals?

- sin(), cos(), etc
- Lookup tables?
- Newton's approximation
- Taylor series expansion
  For example, $cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - ...$
- cordic – (for when multiplies are slow) lookups, shift, add

Quake Square Root Trick

- Fast inverse square root, popularized by use in Quake source code

```c
float InvSqrt(float x) {
    float xhalf=0.5f*x;
    int i=*(int *)&x;
    i=0x5f3759cff - (i>>1);
    x=*(float)&i;
    x=x*(1.5f-xhalf+x*x);
    return x;
}
```