

October 7

- Read 5.1 through 5.3
- Register!
- Questions?
- Chapter 4 – Floating Point

10/7/2004

Comp 120 Fall 2004

1

What is the problem?

- Many numeric applications require numbers over a VERY large range. (e.g. nanoseconds to centuries)
- Most scientific applications require fractions (e.g. π)

But so far we only have integers.

We *COULD* implement the fractions explicitly (e.g. $\frac{1}{2}$, 1023/102934)

We *COULD* use bigger integers

Floating point is a better answer for most applications.

10/7/2004

Comp 120 Fall 2004

2

Floating Point

- Just **Scientific Notation** for computers (e.g. -1.345×10^{12})
 - Representation in IEEE 754 floating point standard
- 32 bits

sign 1	exponent 8	significand or mantissa 23
--------	------------	----------------------------

 single
- 64 bits

sign 1	exponent 11	significand or mantissa 52
--------	-------------	----------------------------

 double
- value = $(1-2^{\text{sign}}) \times \text{significand} \times 2^{\text{exponent}}$
 - more bits for significand gives more precision
 - more bits for exponent increases range

Remember, these are JUST BITS. It is up to the instruction to interpret them.

10/7/2004

Comp 120 Fall 2004

3

Normalization

- In Scientific Notation:
 $1234000 = 1234 \times 10^3 = 1.234 \times 10^6$
- Likewise in Floating Point
 $1011000 = 1011 \times 2^3 = 1.011 \times 2^6$ ← Normalized
- The standard says we should always keep them normalized so that the first digit is 1 and the binary point comes immediately after.
- But wait! There's more! If we know the first bit is 1 why keep it?

10/7/2004

Comp 120 Fall 2004

4

IEEE 754 floating-point standard

- Leading "1" bit of significand is implicit
- We want both positive and negative exponents for big and small numbers.
- Exponent is "biased" to make comparison easier
 - all 0s is smallest exponent all 1s is largest
 - bias of 127 for single precision and 1023 for double precision
 - summary: $(1-2^{\text{sign}}) \times (1+\text{significand}) \times 2^{\text{exponent} - \text{bias}}$
- Example:
 - decimal: $-0.75 = -3/4 = -3/2^2$
 - binary: $-0.11 = -1.1 \times 2^{-1}$
 - floating point: exponent = 126 = 01111110
 - IEEE single precision: 10111111010000000000000000000000
- What about zero?

10/7/2004

Comp 120 Fall 2004

5

Arithmetic in Floating Point

- In Scientific Notation we learned that to add to numbers you must first get a common exponent:
 - $1.23 \times 10^3 + 4.56 \times 10^6 ==$
 - $0.00123 \times 10^6 + 4.56 \times 10^6 ==$
 - 4.56123×10^6
- In Scientific Notation, we can multiply numbers by multiplying the significands and adding the exponents
 - $1.23 \times 10^3 \times 4.56 \times 10^6 ==$
 - $(1.23 \times 4.56) \times 10^{(3+6)} ==$
 - 5.609×10^9
- We use exactly these same rules in Floating point PLUS we add a step at the end to keep the result normalized.

10/7/2004

Comp 120 Fall 2004

6

Floating point AIN'T NATURAL

- It is CRUCIAL for computer scientists to know that Floating Point arithmetic is NOT the arithmetic you learned since childhood
- 1.0 is NOT EQUAL to $10^0 \cdot 1$ (Why?)
 - $1.0 * 10.0 == 10.0$
 - $0.1 * 10.0 != 1.0$
 - 0.1 decimal $= 1/16 + 1/32 + 1/256 + 1/512 + 1/4096 + \dots == 0.00110011001100110011\dots$
 - In decimal $1/3$ is a repeating fraction $0.333333\dots$
 - If you quit at some fixed number of digits, then $3 * 1/3 != 1$
- Floating Point arithmetic IS NOT associative
 $x + (y + z)$ is not necessarily equal to $(x + y) + z$
- Addition may not even result in a change
 $(x + 1) \text{ MAY } == x$

10/7/2004

Comp 120 Fall 2004

7

Floating Point Complexities

- In addition to overflow we can have "underflow"
- Accuracy can be a big problem
 - IEEE 754 keeps two extra bits, guard and round
 - four rounding modes
- Non-zero divided by zero yields "infinity" INF
- Zero divided by zero yields "not a number" NAN
- Implementing the standard can be tricky
- Not using the standard can be worse

10/7/2004

Comp 120 Fall 2004

8

MIPS Floating Point

- Floating point "Co-processor"
- 32 Floating point registers
 - separate from 32 general purpose registers
 - 32 bits wide each.
 - use an even-odd pair for double precision
- `add.d fd, fs, ft` # `fd = fs + ft` in double precision
- `add.s fd, fs, ft` # `fd = fs + ft` in single precision
- `sub.d, sub.s, mul.d, mul.s, div.d, div.s, abs.d, abs.s`
- `l.d fd, address` # load a double from address
- `l.s, s.d, s.s`
- Conversion instructions
- Compare instructions
- Branch (`bc1t, bc1f`)

10/7/2004

Comp 120 Fall 2004

9

Chapter Four Summary

- Computer arithmetic is constrained by limited precision
- Bit patterns have no inherent meaning but standards do exist
 - two's complement
 - IEEE 754 floating point
- Computer instructions determine "meaning" of the bit patterns
- Performance and accuracy are important so there are many complexities in real machines (i.e., algorithms and implementation).
- Accurate numerical computing requires methods quite different from those of the math you learned in grade school.

10/7/2004

Comp 120 Fall 2004

10