Slide 1

Administrivia

- Reminder: Midterm Thursday. Review sheet on the Web.
- Reminder: Homework 2 due today.
- Homework 1 solution will be available in hardcopy outside my office by end of today. Homework 2 solution tomorrow.

Slide 2

Representing Real (Non-Integer) Numbers, Review

- Representation based on binary version of scientific notation.
- So we need to store a sign, significant digits ("significand"), and an exponent.

Slide 3

Representing Real Numbers, Continued

- "IEEE 754 standard" defines formats and operations.
- Exponent is actually stored "biased" — actual exponent plus bias (so we only have to store non-negative exponents — simplifies comparisons).
- Significand doesn't include leading 1. (Why not?)
- But then how to represent 0? Agree that exponent of all 0s will represent 0 if significand is 0, else "de-normalized number".
- Also, agree that exponent of all 1s will represent +/- "infinity" if significand is 0, else NaN ("not a number" — result of indeterminate or invalid operations such as 0/0).
- "Single-precision" format has 8 bits for exponent, biased by 127, 23 bits for significand. (Double precision is 8, 1023, 52 respectively.)
- Work through an example . . .

Slide 4

Floating-Point Addition

- How to add two floating-point numbers? Approach is similar to how you'd do this with decimal numbers in scientific notation:
  1. Shift number with smaller exponent to right until exponents match.
  2. Add fractions.
  3. Normalize (get significand back in proper range).
  4. Check for overflow (exponent too big) or underflow (exponent too small).
  5. Round, and renormalize if necessary.
- Examples in textbook (worth looking through but not necessary to master details).

Slide 5

Floating-Point Multiplication

- How to multiply two floating-point numbers? Approach is also similar to how you'd do this with decimal numbers in scientific notation:
  1. Add exponents and subtract bias.
  2. Multiply fractions.
  3. Normalize (get significand back in proper range).
  4. Check for overflow (exponent too big) or underflow (exponent too small).
  5. Round, and renormalize if necessary.
  6. Set sign bit.

Slide 6

Floating-Point Arithmetic Can Be Strange, Part 1

- Consider the following loop:

```
for (f = 0.0; f != 1.0; f += 0.1)
    printf("f = %g\n", f);
```

  What do you think it does?

  Why?

Slide 7

Floating-Point Arithmetic Can Be Strange, Part 2

- Consider the following code:

```
double fsmall = 1e-10;
double fbig = 1e10;
double temp1 = fbig;
for (int i = 0; i < 10000; ++i)
    temp1 += fsmall;
double temp2 = 0.0;
for (int i = 0; i < 10000; ++i)
    temp2 += fsmall;
temp2 += fbig;
```

  After it runs, is `temp1` equal to `temp2`?

  (This has implications for parallel computing, as described in section 3.6.)

Slide 8

Floating Point in MIPS Architecture

- Architecture defines 32 floating-point registers (`$f0` through `$f31`), used singly for single-precision, in pairs for double-precision.
- Instruction set includes:
  - Arithmetic instructions:

    `add.s`, `sub.s`, `mul.s`, `div.s`; `add.d`, `sub.d`, `mul.d`, `div.d`
  - Load/store instructions (single-precision):

    `lwc1`; `swc1`
  - Comparisons:

    `c.eq.s`, `c.lt.s`, etc.; `c.eq.d`, `c.lt.d`, etc.

    These set a bit true/false, which can be used by `bc1t`, `bc1f`.

Slide 9

Review for Midterm

- (One more example MIPS program — factorial, as an example of a recursive function.)
- (Questions as time permits.)

Slide 10

Minute Essay

- None — quiz.