Slide 1

Administrivia

- Reminder: Quiz 3 Wednesday. Likely topics involve how numbers are represented in binary and some consequences of that (as in the “floating point is strange” examples). I might also ask you to do some simple base conversions (decimal to binary/hexadecimal or vice versa).
- Reminder: Homework 3 programming problems due today at 11:59pm. If you can’t quite finish by then, turn in what you have and then submit an improved version as soon as you can.
- Homework 4 on the Web; due next Monday.
- Early reminder: Midterm next Wednesday. I’ll post a short review sheet by this Wednesday.

Slide 2

Minute Essay From Last Lecture

- Many people had trouble with that last problem. I may have more to say after I grade it.
- Several people said one problem or another was interesting or helpful in understanding.
- Many people found at least some of this homework more difficult than the previous one, but one person found it easier(!).

Slide 3

Integer Addition/Subtraction — Recap

- We talked about these operations last time.
- Textbook comments that `slt` could also be implemented using the same logic — to check for `a < b`, compute `a-b` and check for negative result (high-order bit on). Clever!(?)

Slide 4

Integer Addition/Subtraction — Signed Versus Unsigned

- You may know/remember that since integers are of fixed size, addition and subtraction can "overflow". (Textbook goes into some details.)
- Some higher-level languages (C for example) don't require detecting such overflow. Others (Ada?) do.
- Thus, MIPS, like many architectures, offers "signed" versions of arithmetic operations, which check for overflow and raise a (hardware) exception if found, and "unsigned" versions (e.g., `addu`), which don't.

Integer Multiplication — Recap

Slide 5

- Simple algorithm based on same idea as multiplying in base 10. Details in textbook. A little intimidating unless/until you work through an example.
- MIPS instruction set includes one multiply instruction `mult` with two operands. Results go in pair of special-purpose registers; use `mflo` to retrieve low-order 32 bits.
- Basic algorithm works for signed quantities as well, though you have to be a bit careful. Details in textbook.

Division

Slide 6

- As with other arithmetic, first think through how we do this "by hand" in base 10. (Review terminology: We divide "dividend" a by "divisor" b to produce quotient q and remainder r, where $a = bq + r$ and $0 \leq |r| < b$.) Example?

  We can do the same thing in base 2; this gives the algorithm shown in textbook figures 3.8 through 3.10. (Work through example?)

  (Here too we can do better — later maybe).
- What about signs? Simplest solution is (they say!) to perform division on non-negative numbers and then fix up signs of the result if need be.

Slide 7

Division, Continued

- In MIPS architecture, 64-bit work area for quotient and remainder is kept in same two special-purpose registers used for multiplication (`lo` and `hi`). After division, quotient is in `lo` and remainder is in `hi`. Two (or more) instructions needed to do a division and get the result:

```
div rs1, rs2
mflo rq
mfhi rr
```

  Assembler provides a "pseudoinstruction":

```
div rdest, rs1, rs2
```

- Notice, however, that a "smart" compiler might turn some divisions into shifts. (Which ones?)

Slide 8

Integer Multiplication and Division, Recap

- Algorithms for both operations are based on how you do things "by hand", with some modifications to permit simpler hardware. It's not critical to understand the details, but probably useful to work through an example to believe that it works.
- Required hardware is something that can add two 32-bit numbers, a 64-bit "work area", something to do right and left shifts of the 64-bit area, and some control logic.
- MIPS architecture uses "special registers" `lo` and `hi` for the 64-bit work area. This is where the results end up. There are instructions to multiply, to divide, and to move from the special registers. ("Move from" explains the names of the instructions.)

Floating-Point Revisited

Slide 9

- We talked somewhat generally about floating-point formats. May be useful to work through an example or two to see how this works.
- Arithmetic on floating-point values is, no surprise, a bit complicated. Textbook shows algorithms (in flowchart form). Probably useful to skim.

Floating Point in MIPS Architecture

Slide 10

- 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`.

Minute Essay

- That's it for Chapter 3. Questions?
- Anything noteworthy about the programming problems for Homework 3? For most of you this was probably your first try at producing complete-for-SPIM MIPS programs; was it interesting, tedious, educational, . . . ?

Slide 11