Slide 1

Administrivia

- Reminder: Homework 1 due Wednesday.

Slide 2

Minute Essay From Last Lecture

- A few people got something that was more or less right. Many others were in the ballpark.
- Keep in mind that syntax for assembler-language instructions is pretty constrained — `add` has exactly three operands, which must be registers (usually referenced with $ and a symbolic name).

Slide 3

A Little About the Simulator

- Your code goes in a file with extension `.s`. (Sample starter code on “Sample programs” page. Contains many things we haven’t talked about yet but could still be useful for trying things out.)
- Start the simulator with command `xspim` (`spim` for command-line version). (Short demo.)

Slide 4

Representing Instructions in Binary — Review/Recap

- Objective here is to represent in binary (ones and zeros) the instructions we’re defining (`add`, etc.)
- Representation must indicate which instruction it is and its operands.
- Somewhat tricky in that different (sets of) instructions have different kinds of operands (contrast `add` and `lw`) of possibly-different sizes. Several ways to deal with that; MIPS designers chose to make all instructions the same length and different “instruction formats”.

Slide 5

R Format

- Meant for instructions such as `add`.
- Fields:
  - `op` — op code, 6 bits
  - `rs` — first source operand, 5 bits
  - `rt` — second source operand, 5 bits
  - `rd` — destination operand, 5 bits
  - `shamt` — “shift amount” (not used for `add`), 5 bits
  - `funct` — “function field”, 6 bits
- Example — find binary representation of

```
        add     $t0, $s1, $s2
```

Slide 6

I Format

- Meant for instructions such as `lw`.
- Fields:
  - `op` — op code, 6 bits
  - `rs` — first source operand, 5 bits
  - `rt` — destination operand, 5 bits
  - `disp` — displacement, 16 bits
- Example — find binary representation of

```
        lw      $t0, 1200($t1)
```

  Look up `op` and registers in tables on “green card”.
- How can we tell which format is being used? determined by value for `op`.

Slide 7

Logical Operations

- Sometimes useful to be able to work with individual bits — e.g., to implement a compact array of boolean values.
- Thus, MIPS instruction set provides “logical operations”. Hard to say whether these exist to support C bit-manipulation operations, or C bit-manipulation operations exist because most ISAs provide such instructions!

Slide 8

“Shift” Instructions

- C `<<` and `>>` (on unsigned numbers) are translated into `sll` (“shift left logical”) and `srl` (“shift right logical”).
- `sll` and `srl` do what the names imply — bits “fall off” one side, and we add zeros at the other side. These are R-format instructions, and they use that “shift amount” field.
- When shifting left, filling with zeros makes sense. But when shifting right, we might want to extend the sign bit instead. `sra` (“shift right arithmetic”) does that.
- Examples?

Bitwise And and Or

Slide 9

- C & is translated into `and` or `andi`. C | is translated into `or` or `ori`. Format/operands are analogous to `add` and `addi`.

  (Notice/recall that C has two sets of and/or operators — logical and bitwise. These are the bitwise ones.)

- We could use these to test/set particular bits. Examples? Could we use them to, e.g., compute remainder when dividing by power of 2?

Other Logical Operations

Slide 10

- “Exclusive or” implements — what the name suggests (see textbook).
- “Nor” likewise. Can be used to implement “not” (see textbook).

Slide 11

Flow of Control

- So far we know how to do (some) arithmetic, move data into and out of memory. What about if/then/else, loops? (See sidebar on p. 105 for early commentary on conditional execution.)
- We need instructions that allow us to "make a decision" — `beq` ("branch if equal"), `bne` ("branch if not equal").
- Illustrate with an example . . .

Slide 12

Flow of Control Example

- Suppose we have this in C

```
        if (i == j) goto L1:
        f = g + h;
L1:     f = f - i;
```

- What instructions should compiler produce? Assume we're using `$s0` through `$s4` for for `f`, `g`, `h`, `i`, `j`.
- (For now, punt on how to represent `L1`.)

Slide 13

Another Flow of Control Example

- Of course, we don't usually have `go to` in C. More likely is this:

```
if (i == j)
        f = g + h
else
        f = g - h
```

- What to do with this? Rewrite using `go to` . . .

Slide 14

Loops

- Do we have enough to do (some kinds of) loops? Yes — example:

```
Loop:   g = g + A[i];
        i = i + j;
        if (i != h) goto Loop:
```

  assuming we're using `$s1` through `$s4` for `g`, `h`, `i`, `j`, and `$s5` for the address of A.

- Or how about something that looks more like normal C?

```
while (A[i] == k) {
        i = i + j;
```

- (To be continued . . .)

Slide 15

Minute Essay

- Is this making sense? Is the pace of the class too fast, too slow, about right?