Slide 1

Administrivia

- Reminder: Homework 2 due by 5pm today.
- Quiz solutions will be available on the Web, usually shortly after class.

Slide 2

Flow of Control, Review

- Instructions we have so far for testing conditions and changing "flow of control":
  - `beq r1, r2, label` — compares contents of registers `r1` and `r2` and "branches" to `label` if equal. (Punt for now on what `label` means.)
  - `bne r1, r2, label` — similar but branches if not equal.
  - `j label` — unconditionally "jumps" to `label`.
- Do we have enough to do loops?

Slide 3

Loop Examples

- "Compile" the following C:

```
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;
```

Slide 4

More Flow of Control

- We can do if/then/else and loops, but only if condition being tested is equals / not equals.
- So, we need instructions such as `blt`, `ble`, right?
- But those are difficult to implement well, so instead MIPS has "set on less than":

```
slt   r1, r2, r3
```

  which compares the contents of registers `r2` and `r3` and sets `r1` — 1 if `r2` is smaller, else 0.

- Also define that register 0 (`$zero`) always contains 0.
- Example — compile the following C:

```
        if (a < b) go to Less:
```

  assuming we're using `$s0`, `$s1` for `a`, `b`

Slide 5

More Flow of Control, Continued

- Do we have enough now? for all six possible C comparisons of integers? Yes . . .
- One more C flow-of-control construct we could talk about — `switch` — but defer for now.
- But we do want to talk about one more HLL feature . . .

Slide 6

Procedure Calls

- How do we call procedures (a.k.a. functions, methods)? Consider an example:

```
a = a + a;
x = foo(a);
b = b + b;
y = foo(b);
```

- If we've compiled this code (and function `foo`), what do we have in memory when it's running? What's supposed to happen when we get to a call to `foo`?

Slide 7

Procedure Calls, Continued

- So, what we have to do to call a procedure is:
  1. Put parameters where procedure can find them.
  2. Transfer control to procedure.
  3. Acquire storage resources for procedure (recall that every time you call a C function you get a "new copy" of all its local variables).
  4. Run procedure.
  5. Put results where caller can find them.
  6. Return control to caller.
- How to do all this? To be continued . . .

Slide 8

A Little About the Simulator

- Your code goes in a file with extension `.s`.
- Start the simulator with command `xspim`. Need a copy of `/usr/local/spim-6.3/trap.handler` in the current directory.

  (Short demo.)

Slide 9

Minute Essay

- Write MIPS assembler for the following C code fragment:

```
while (i < h) {
        A[i] = i;
        i = i + j;
}
```

assuming we're using `$s1` through `$s3` for `h`, `i`, `j`, and `$s4` for the address of `A`.