Slide 1

Administrivia

- Quiz solutions will be online, usually shortly after the quiz.

Slide 2

Procedure Calls, Review/Recap

- 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.
  4. Run procedure.
  5. Put results where caller can find them.
  6. Return control to caller.
- We discussed last time details of how to do that.

Slide 3

A Little About Assembly Language and Assemblers

- We've done example(s) of translating assembly language into machine language.
- Normally this is done programmatically, by an "assembler". Accepts symbolic representations of instructions. Also uses some directives to help keep track of instructions, define character strings, etc. Details for MIPS assembler in Appendix B.

Slide 4

Example

- How to compile the following?

```
int main() {
        a = 5; b = 6; c = 7;
        x = addproc(a, b, c);
        return 0;
}
int addproc(int a, int b, int c) {
        int x;
        x = a + b + c;
        return x;
}
```

(Sample program `call-addproc.s`.)

Data Formats, Revisited

- Recall, inside the computer "it's all ones and zeros" — so we must encode anything we want to represent.
- Integers — binary numbers, often 32 bits for MIPS, but could be other sizes too. Several choices for signed numbers; two's complement is the most common.
- Real numbers — floating-point format, later.
- Text — ASCII (8 bits per character) or Unicode (16 bits). Strings represented with explicit length field (Java) or terminating character (C).
- Many, many more complex formats (`.doc`, `MP3`, `GIF`, etc.).

More Load/Store Instructions

- MIPS architecture defines `lw` and `sw` for loading/storing data in 32-bit chunks; also defines `lb` ("load byte") and `sb` ("store byte") for loading/storing data in 8-bit chunks, plus instructions to load/store data in 16-bit chunks.

  All must align on appropriate boundaries.

Slide 7

Working with Constants, Revisited

- Recall `addi` instruction. Exists because often we need to use a small constant in a program.
- Uses same format ("I format") as `lw` and `sw`, which allows 16 bits for constant.
- What if we need more than 16 bits? "Load upper immediate" instruction:

  ```
  lui register, constant
  ```

  Puts (16-bit) constant in "upper" 16 bits of register. Follow with `addi` (or, better, `ori`) to load a full 32-bit constant.

Slide 8

Addressing Modes

- We've been unspecific about how to specify addresses of a lot of things.
- So, now look at various "addressing modes" — ways to specify where to find an operand.
- Which is used? Defined by instruction format (R, I, J).

Slide 9

Addressing Modes, Continued

- Register addressing: Value is in one of the general-purpose registers. Assembler defines symbolic names for them (e.g., `$t0`).
- Immediate addressing: Value is in instruction itself.
- Base-displacement addressing: Value is in memory, with address calculated by adding a displacement to what's in a register. Example is memory-address operand of `lw`, `sw`.
- PC-relative addressing (more shortly).
- Pseudo-direct addressing (more shortly).

Slide 10

PC-Relative Addressing

- Address is formed by adding offset in instruction (16 bits) and contents of the program counter (special register).

  (Actually, address is offset times 4, plus the updated program counter.)
- Example is conditional branches (`beq`, `bne`).
- Does this limit what we can do with `beq` and `bne`? If so, how often will it matter? What could we do to work around it?

Slide 11

Pseudo-Direct Addressing

- Address is formed by combining address in instruction (26 bits) and upper bits of program counter.

  (Actually, address is address in instruction times 4, or'd with upper bits of program counter.)
- Example is unconditional branch (`j`).
- Does this limit what we can do with `j`? If so, will that be a problem? Can we work around it?

Slide 12

Minute Essay

- Write MIPS assembler code for the following procedure, saving/restoring the return address at least:

```
int foo(int a, int b) {
        return a + b;
}
```

Minute Essay Answer

- Here is one answer:

```
foo:    addi    $sp, $sp, -4
        sw      $ra, 0($sp)
        add     $v0, $a0, $a1
        lw      $ra, 0($sp)
        addi    $sp, $sp, 4
        jr      $ra
```