Slide 1

Administrivia

- (Review minute essay from last time.)
- Homework 1 revised slightly to ask you to show at least some work. Could use minute essay from last time as a model of how much to show.
- Quiz 1 next Friday. In class, about ten minutes, open book/notes. More about possible topics next week.

Slide 2

High-Level Languages Versus Assembly Language

- In a high-level language you work with "variables" — conceptually, names for memory locations. You can do arithmetic on them, copy them, etc.
- In machine/assembly language, what you can do may be more restricted — e.g., in MIPS architecture, you must load data into a register before doing arithmetic).
- The compiler's job is to translate from the somewhat abstract HLL view to machine language. To do this, normally associate variables with registers — load data from memory into registers, calculate, store it back. A "good" compiler tries to minimize loads/stores.

Load/Store Example

Slide 3

- Suppose we have this in C

```
a[12] = h + a[8];
```

- What instructions should compiler produce? Assume we're using `$s3` for starting ("base") address of `a`, `$s2` for `h`.

Addition Using Constant

Slide 4

- "Add immediate"

```
addi r1, r2, c
```

adds constant `c` (16-bit signed integer, can be negative) to contents of `r2`, puts result in `r1`.

- Exists because often we need to use a small constant in a program.

"Design Principle 3: Make the common case fast."

Slide 5

Representing (Integer) Data in Binary

- Remember that to the hardware "it's all ones and zero" — any data you're working with.
- As an example — representation of signed integers using two's complement notation. Should have been covered in CSCI 1320, but read/skim 2.4 if you don't remember.

Slide 6

Representing Instructions in Binary

- "It's all ones and zeros" applies not only to data but also to programs — "stored program" idea. (Some very early computers didn't work that way — programming was by rewiring(!).)
- So we need a way to represent instructions in binary.

Slide 7

Representing Instructions in Binary, Continued

- First consider what we have to represent:
  - For all instructions, which instruction it is.
  - For `add` and `sub`, three operands (all register numbers).
  - For `lw` and `sw`, three operands (two register numbers and a "displacement").
  - And so forth ...
- So, each instruction will have "fields" — consistent format for storing pieces of data, a little like a C `struct`.

Slide 8

Representing Instructions in Binary, Continued

- So, can we use the same format for all instructions? Some data ("which instruction") is common to all, but operands may need to be different.
- Can we / should we make all instructions the same length? For MIPS, yes (other architectures differ), and then define different ways of dividing up the length — "formats".

  "Design Principle 4: Good design involves good compromises."

Slide 9

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 10

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)
```

- How can we tell which format is being used? determined by value for `op`.

Minute Essay

Slide 11

- Write MIPS assembly code for the following C program fragment:

```
a = b + c + d + e
```

  Assume we have b, c, d, e in $s1 through $s4 and want to have a in $s0

  Optional: Can you think of more than one way to do it? If you can, does one seem better than the other, and why?

  OR

- Write MIPS assembler code to exchange the values of a[0] and a[1]. Assume register $s0 contains the address of a (start of the array), and a is an array of integers.

Minute Essay Answer

Slide 12

- One way:

```
add    $s0, $s1, $s2
add    $s0, $s0, $s3
add    $s0, $s0, $s4
```

  Another way (not as good since uses more registers?):

```
add    $t0, $s1, $s2
add    $t1, $s3, $s4
add    $s0, $t0, $t1
```

- One way:

```
lw    $t0, 0($s0)
lw    $t1, 4($s0)
sw    $t0, 4($s0)
sw    $t1, 0($s0)
```