Slide 1

Administrivia

- Notice (if you haven't already) that textbook includes "check yourself" questions — answers to which appear after end-of-chapter problems. Useful!
- Homework 2 will be on the Web later today (I will send mail). Due in a week.

Slide 2

Minute Essay From Last Lecture

- No one really came close! See notes for one answer.

Slide 3

Assembly Language to Machine Language

- We talked earlier about how to turn (some parts of) assembly instructions into machine language (ones and zeros). We punted on labels. Translating them into machine code done by assembler.
- Look at an example — machine language for this C:

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

Assume we're using `$s3` through `$s6` for `i`, `j`, `k`, address of `a`, and that code is in memory at (decimal) location 80000.

What does the machine code look like?

Slide 4

Decoding Machine Language

- As a check on whether what we have is sensible — try going from machine language back to assembly language, using same example.
- Notice that there are limits to how reversible this process is. (What?)

Slide 5

Sidebar: Parallel Execution and Synchronization

- A lot of commodity hardware these days features multiple processing units ("cores") sharing access to memory. One reason for this is that in theory we can make individual applications faster by splitting computation up among processing elements.

  (Shameless self-promotion: We plan to try again to offer the parallel-programming elective in the fall.)

- Having processing elements share memory makes parallel programming easier in some ways but has risks ("race conditions"). Avoiding the risks requires some way to control access to shared variables (e.g., to implement notion of "lock").

- (To be continued.)

Slide 6

Minute Essay

- Is the material in chapter 2 making sense to you? If not, can you say what the trouble is?