Slide 1

Administrivia

- None?

Slide 2

Minute Essay From Last Lecture

- Question: Sketch a finite state machine for a slightly improved traffic signal:
  - Inputs are sensors as described plus a “flash both directions” input.
  - Outputs are “N/S green” and “E/W green” as described, plus “N/S flash” and “E/W flash”.
  - Operation should be as before, except that whenever the “flash both directions” input is on, the two “flash” outputs should be 1 and the two “green” outputs should be 0.
- Answer?

Slide 3

Generating Control Signals with a FSM, Recap

- First we draw a FSM that represents our overall strategy (5 steps, different steps for different instructions) — figure 5.42.
- Then we implement this as in figure B.29 / figure 5.43.
- Then we use this as the “Control” CL block in figure 5.33. (Actually it’s not completely a CL block any more, right?)

Slide 4

Generating Control Signals with Microprogramming

- The FSM approach works okay if the number of states is reasonable. If we had more instructions, though, the graphical representation could get out of control, no?
- Another approach is to treat generating control signals as a programming problem, sort of — “microprogramming”.

  Can think of this as text representation of FSM, or as an “assembly language below assembly language”.
- Basic idea is to define “microinstructions” (groups of control signals for the datapath) and use them to build a text representation of the FSM.

Slide 5

Microinstructions

- Format for microinstructions (more details in figure 5.45):
  - Label.
  - ALU control (`ALUOp`). Possible values: `Add`, `Sub`, `Func code`.
  - Source 1, Source 2 — ALU inputs. One possible value for each input to ALU-input multiplexors.
  - Register control — what to do with register file. Possible values `Read`, `Write ALU`, `Write MDR`. Allows us to generate "do we write" flag and input to multiplexor.
  - Memory — what to do with memory. Possible values `Read PC`, `Read ALU`, `Write ALU`. Allows us to generate control signals for memory and IR.
  - Continued on next slide . . .

Slide 6

Microinstructions, Continued

- Format for microinstructions, continued:
  - PC Write control — whether/how to change PC. Possible values `ALU`, `ALUOut-cond`, `Jump address`. Allows us to generate control signals for changing PC.
  - Sequencing — how to choose next microinstruction. Possible values `Seq`, `Fetch`, `Dispatch` n.

Slide 7

Microinstructions — Sequencing

- The choices we want are:
  - Go to next microinstruction.
  - Go back to start (i.e., start execution of next MIPS instruction).
  - Go somewhere else. For this, the idea is to have a “dispatch table” that maps control-unit inputs to labels. Can have more than one of these.

Slide 8

Microprogramming for Our Multiple-Cycle Implementation

- Microprogram for our subset implementation has just 10 instructions (one for each state of FSM). See figure 5.46.
- How to turn this into hardware? several ways . . .

Hardware for Microprogramming

- One approach is to treat the microprogram as a text representation of a FSM and implement the same way. (Translating truth tables into gates or other hardware is completely mechanical, so there are tools to automate it.)
- Another approach — “ROM” (read-only memory) and sequencer. Idea is to:
  - “Assemble” microinstructions into binary form (e.g., into needed control signals) and store in a small read-only memory.
  - Use counter to store address (in this ROM) of next instruction.
  - Update counter using adder and some control logic (driven by output of main control unit).

  See figure 5.47.

  Probably a better choice if many states, especially if next state is often `Seq`. (E.g., consider multiply, divide, floating-point instructions.)

Minute Essay

- Was Homework 6 helpful, tedious, both, neither, . . . ?
- Reminder: Homework 6 due today.