Slide 1

Administrivia

- Homework 7 on Web. Due next Monday.
- Reminder: Quiz 6 Wednesday.

Slide 2

Generating Control Signals for a Multi-Cycle Implementation, Recap

- One approach — draw finite state machine, turn into tables mapping inputs (six bits of opcode) and current state to outputs (control signals) and next state. Then convert to big CL block. Reasonable if not many states.
- Another approach — "microprogramming", basically a text representation of the FSM. Represent each microinstruction in binary (encoding control signals and representation of next state), store in ROM. Better if many states.
- More details, for those who are interested, in Appendix C. (In particular, the explanation of dispatch tables and "sequencers" seems clearer than in chapter 5.)

Slide 3

Exceptions and Interrupts

- Up to now we've talked only about running a single program in which nothing ever "goes wrong". But real systems must also cope with errors, e.g.:
  - Arithmetic overflow.
  - Undefined instruction. (How could this happen?)
  - Hardware problems.
- A useful approach — transfer control to some part of operating system and let it take action (try to recover, terminate program, etc.).
- Same approach can be used for other events whose timing is unpredictable, e.g.:
  - I/O device requests service/attention.
  - User program requests operating-system service.
- Can classify such events as internal/external, exceptions/interrupts, but basic idea is the same.

Slide 4

Exceptions and Our Multi-Cycle Implementation

- As examples, consider two kinds of exceptions possible with our implementation — arithmetic overflow, undefined instruction.
- What do we want to happen?
  - Save information about what happened — address of instruction that caused error, which kind of exception. Save each in a special-purpose register.
  - Transfer control to fixed address (presumably containing part of operating system), which can then take appropriate action. (Could that include continuing the program that caused the exception?)

Slide 5

Datapath Additions for Exceptions

- What do we need to add to make it possible to deal with exceptions as described?
  - Someplace to save address of problem instruction, type of exception — `EPC` and `Cause` registers. Note that we don't want to write to them every cycle. Note also that value for `EPC` isn't current value of PC!
  - Some way to set PC to address of "interrupt handler".
- Result is figure 5.48 (which leaves out routing `Overflow` output of ALU to main "control" block).

Slide 6

Control Signals for Exceptions

- New control signals `EPCWrite`, `CauseWrite`. Everything else can be done with existing signals.

Slide 7

Exceptions and our FSM

- Two new states for FSM, one for each type of exception. How/where to add these? See figure 5.50.

  (Actually, figure 5.50 is not quite right — we need a 1-bit register to save `Overflow`.)

Slide 8

Minute Essay

- Write a sequence of MIPS instructions that would be guaranteed to cause an arithmetic overflow exception.
- Write a sequence of MIPS instructions that would be guaranteed to cause an "undefined instruction" exception.