

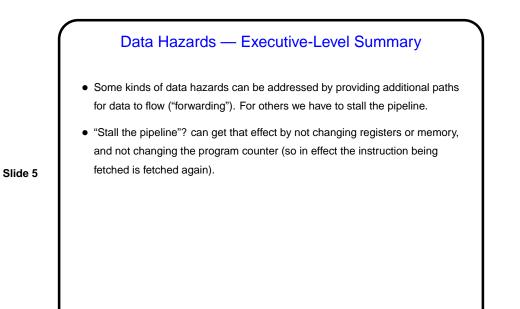


- Idea is to break up processing of each instruction into several stages, and overlap processing. Textbook compares to laundry room; another widely-used analogy is an assembly line.
- For MIPS architecture (subset), five stages makes sense. To make this all work, logically separate datapath into five pieces and add "registers" (place to save data) between stages as needed, as shown in Figure 4.35. Next few figures show execution of 1w and may help this make sense.

Slide 3

Pipelined Implementation — What's Left Need to be explicit about exactly what's needed for those "registers" between stages, but should be common sense(?). Need to generate control signals, as in single-cycle implementation — and here, need to also add (some of) them to those interstage registers. Figure 4.51 shows result. Need to deal with data and control hazards. (Structural hazards don't exist for MIPS ISA — well, assuming we have separate instruction/data memories, as in the single-cycle implementation.)

Slide 4

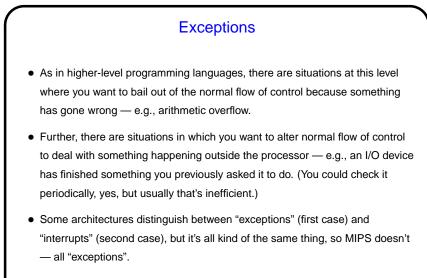


Control Hazards — Executive-Level Summary

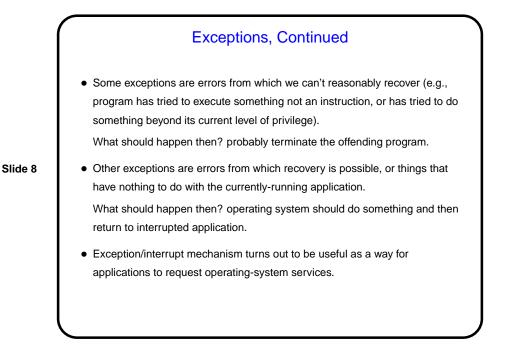
- Several ways to deal with control hazards.
- One would be to stall the pipeline (though apparently that isn't done).
- Others involve "flushing" in-progress instructions (before they change anything!).

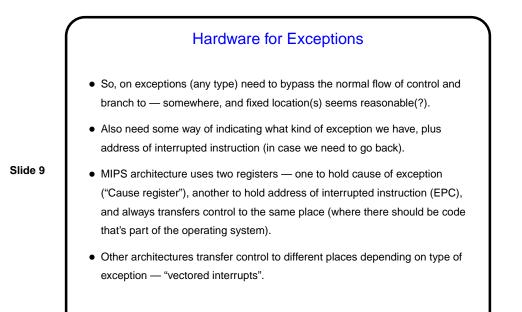
Slide 6

Slide 7



• What should happen on exception? Several possibilities ...





Exceptions — Hardware Versus Software

- Hardware must save current PC (with a caveat) and transfer control to fixed location(s) with an indication of cause of exception.
- Code at fixed location(s) must "do the right thing" for the exception, as described previously. Normally this code is part of operating system.
- Caveat: Pipelining complicates exception processing must allow instructions prior to the interrupted one complete, complete or flush the interrupted one, etc. Textbook has (some of) details.

Slide 10

