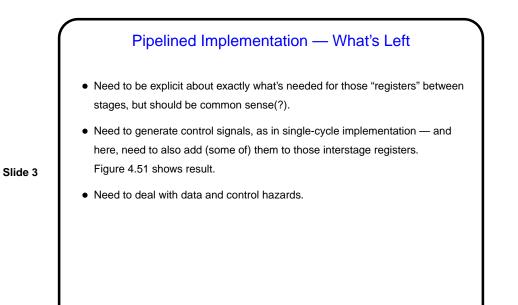
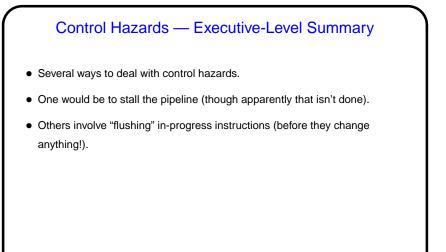


Pipelined Implementation — Review/Recap
Idea is to break up processing of each instruction into several stages, and overlap processing. Textbook compares to laundry room; another widely-used analogy is an assembly line.
For MIPS architecture (subset), five stages makes sense. To make this all work, logically 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.



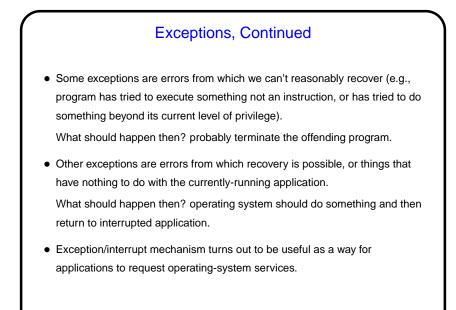
Data Hazards — Executive-Level Summary
Some kinds of data hazards can be addressed by providing additional paths for data to flow ("forwarding"). For others we have to stall the pipeline.
"Stall the pipeline"? can get that effect by not changing registers or memory, and not changing the program counter (so in effect the instruction being fetched is fetched again).



Slide 5

Slide 6

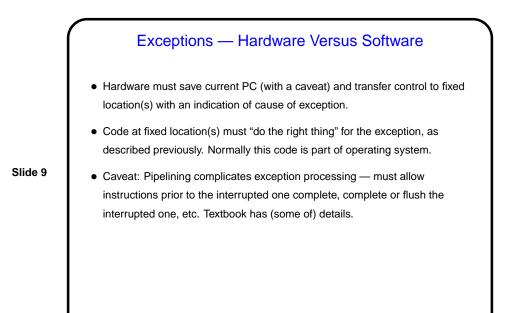
Exceptions As in higher-level programming languages, there are situations at this level where you want to bail out of the normal flow of control because something has gone wrong — e.g., arithmetic overflow. Further, there are situations in which you want to alter normal flow of control to deal with something happening outside the processor — e.g., an I/O device has finished something you previously asked it to do. (You could check it periodically, yes, but usually that's inefficient.) Some architectures distinguish between "exceptions" (first case) and "interrupts" (second case), but it's all kind of the same thing, so MIPS doesn't — all "exceptions". What should happen on exception? Several possibilities …

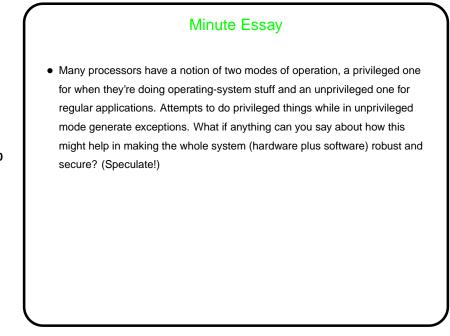


Slide 7

Slide 8

Hardware for Exceptions So, on exceptions (any type) need to bypass the normal flow of control and branch to — somewhere, and fixed location(s) seems reasonable(?). Also need some way of indicating what kind of exception we have, plus address of interrupted instruction (in case we need to go back). MIPS architecture uses two registers — one to hold cause of exception ("Cause register"), another to hold address of interrupted instruction (EPC), and always transfers control to the same place (where there should be code that's part of the operating system). Other architectures transfer control to different places depending on type of exception — "vectored interrupts".





Minute Essay Answer

 If regular applications execute in unprivileged mode, the hardware can enforce some restrictions on what they can do (e.g., only request I/O by going through the operating system). How do you get from unprivileged mode to privileged mode then? As part of exception processing — hardware transfers control to fixed location(s) and switches to privileged mode.