CSE 490/590 Computer Architecture

Pipelining II

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Last Time…
- MIPS instructions
  - Reg-Reg: ADD R1,R2,R3
  - Reg-Imm: LD R1,30(R2), BEQZ R4, name
  - Jump/Call: J name, JAL name
- MIPS single cycle implementation
  - Fetch → Decode & Reg fetch → execute → mem → WB
    - All in one cycle
- MIPS pipelining
  - 5 stages
  - Iron law: performance benefit analysis
    » Pipelining reduces time/cycle
    » Also try to maintain CPI == 1

Harvard-Style Datapath for MIPS

5-Stage Pipelined Execution

5-Stage Pipelined Execution
Resource Usage Diagram

Pipelined Execution: ALU Instructions

Not quite correct!
We need an Instruction Reg (IR) for each stage
Instructions interact with each other in pipeline

- An instruction in the pipeline may need a resource being used by another instruction in the pipeline → structural hazard
- An instruction may depend on something produced by an earlier instruction
  - Dependence may be for a data value → data hazard
  - Dependence may be for the next instruction’s address → control hazard (branches, exceptions)

Resolving Structural Hazards

- Structural hazards occurs when two instructions need same hardware resource at same time
  - Can resolve in hardware by stalling newer instruction till older instruction finished with resource
- A structural hazard can always be avoided by adding more hardware to design
  - E.g., if two instructions both need a port to memory at same time, could avoid hazard by adding second port to memory
- Our 5-stage pipe has no structural hazards by design
  - Thanks to MIPS ISA, which was designed for pipelining

Resolving Data Hazards (1)

Strategy 1:

Wait for the result to be available by freezing earlier pipeline stages → interlocks

Feedback to Resolve Hazards

- Later stages provide dependence information to earlier stages which can stall (or kill) instructions
Interlocks to resolve Data Hazards

Interlock Control Logic

Hazards due to Loads & Stores
Load & Store Hazards

However, the hazard is avoided because our memory system completes writes in one cycle!

Load/Store hazards are sometimes resolved in the pipeline and sometimes in the memory system itself.

More on this later in the course.

Resolving Data Hazards (2)

Strategy 2:

Route data as soon as possible after it is calculated to the earlier pipeline stage \( \rightarrow \text{bypass} \)

Bypassing

Each stall or kill introduces a bubble in the pipeline \( \Rightarrow CPI > 1 \)

A new datapath, i.e., a bypass, can get the data from the output of the ALU to its input

Adding a Bypass

When does this bypass help?

Acknowledgements

These slides heavily contain material developed and copyright by:
- Krste Asanovic (MIT/UCB)
- David Patterson (UCB)
- Arvind (MIT)
- Joel Emer (Intel/MIT)
- James Hoe (CMU)
- John Kubiatowicz (UCB)
- MIT material derived from course 6.823
- UCB material derived from course CS252