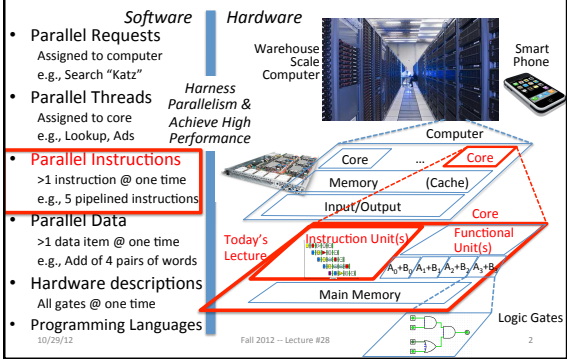
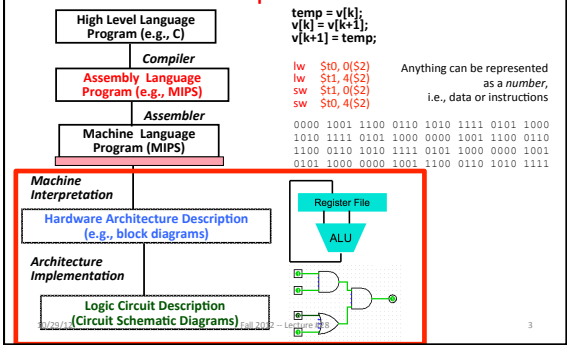
CS 61C:
Great Ideas in Computer Architecture
Control and Pipelining, Part II

Instructors:
Krste Asanovic, Randy H. Katz
http://inst.eecs.Berkeley.edu/~cs61c/fa12

You Are Here!



Levels of Representation/ Interpretation



Agenda

- Structural and Data Hazards
- Administrivia
- Control Hazards
- And in Conclusion, …

Agenda

- Structural and Data Hazards
- Administrivia
- Control Hazards
- And in Conclusion, …

Hazards

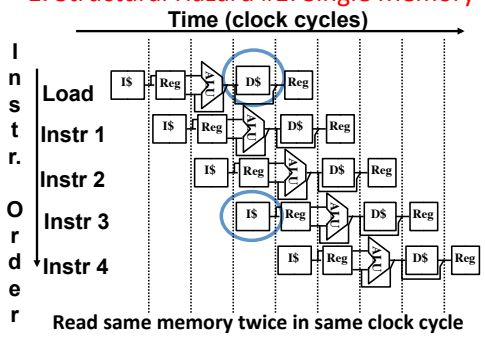
Situations that prevent starting the next logical instruction in the next clock cycle

1. Structural hazards
   – Required resource is busy (e.g., stasher is studying)
2. Data hazard
   – Need to wait for previous instruction to complete its data read/write (e.g., pair of socks in different loads)
3. Control hazard
   – Deciding on control action depends on previous instruction (e.g., how much detergent based on how clean prior load turns out)

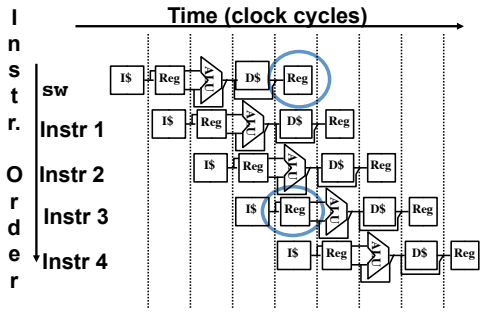
1. Structural Hazards

- Conflict for use of a resource
- In MIPS pipeline with a single memory
  - Load/Store requires memory access for data
  - Instruction fetch would have to *stall* for that cycle
    - Causes a pipeline "*bubble*"
- Hence, pipelined datapaths require separate instruction/data memories
  - In reality, provide separate L1 instruction cache and L1 data cache

1. Structural Hazard #1: Single Memory



1. Structural Hazard #2: Registers (1/2)



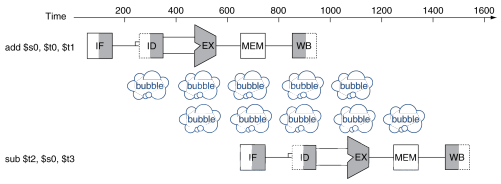
1. Structural Hazard #2: Registers (2/2)

- Two different solutions have been used:
  1) RegFile access is *VERY* fast: takes less than half the time of ALU stage
     - Write to Registers during first half of each clock cycle
     - Read from Registers during second half of each clock cycle
  2) Build RegFile with independent read and write ports
- Result: can perform Read and Write during same clock cycle

2. Data Hazards

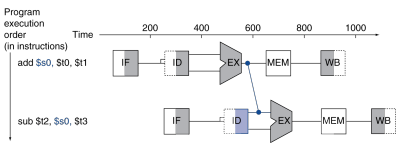
- An instruction depends on completion of data access by a previous instruction

```
add  $s0, $t0, $t1
sub  $t2, $s0, $t3
```

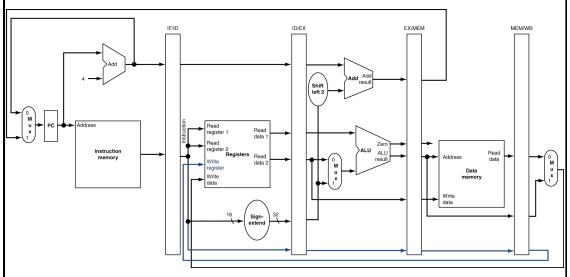


Forwarding (aka Bypassing)

- Use result when it is computed
  - Don't wait for it to be stored in a register
  - Requires extra connections in the datapath

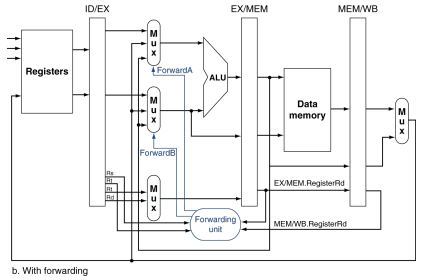


Corrected Datapath for Forwarding?



Student Roulette?

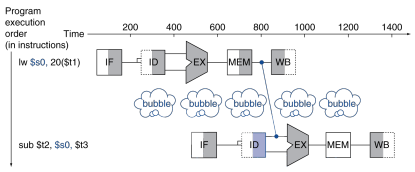
Forwarding Paths



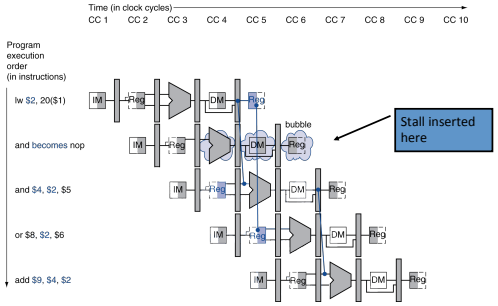
b. With forwarding

Load-Use Data Hazard

- Can't always avoid stalls by forwarding
  - If value not computed when needed
  - Can't forward backward in time!



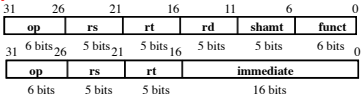
Stall/Bubble in the Pipeline



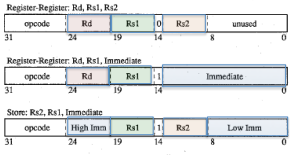
Pipelining and ISA Design

- MIPS Instruction Set designed for pipelining
- All instructions are 32-bits
  - Easier to fetch and decode in one cycle
  - x86: 1- to 17-byte instructions
    (x86 HW actually translates to internal RISC instructions!)
- Few and regular instruction formats, 2 source register fields always in same place
  - Can decode and read registers in one step
- Memory operands only in Loads and Stores
  - Can calculate address 3rd stage, access memory 4th stage
- Alignment of memory operands
  - Memory access takes only one cycle

Why Isn't the Destination Register Always in the Same Field in MIPS ISA?



- Need to have 2 part immediate if 2 sources and 1 destination always in same place



SPUR processor (A project Dave and Randy Katz worked on together)

Agenda

- Structural and Data Hazards
- Administrivia
- Control Hazards
- And in Conclusion, …

Administrivia

- Project #4, Labs #10 and #11, (Last) HW #6 posted
  - Project due 11/11 (Sunday after next)
  - Project is not difficult, but is long: don't wait for the weekend when it is due! Start early!
  - Look at Logisim labs before you start on Project #4
    - Lots of useful tip and tricks for using Logisim in those labs
- TA Sung Roa will hold extended office hours on the project this coming weekend

Administrivia

- Reweighting of Projects
  - Still 40% of grade overall
    - Project 1: 5%
    - Project 2: 12.5%
    - Project 3: 10%
    - Project 4: 12.5%
    - Optional Extra Credit Project 5: up to 5% extra
      - Gold, Silver, and Bronze medals to the three fastest projects
      - Code size and aesthetics also a criteria for recognition
      - Winning projects as selected by the TAs will be highlighted in class

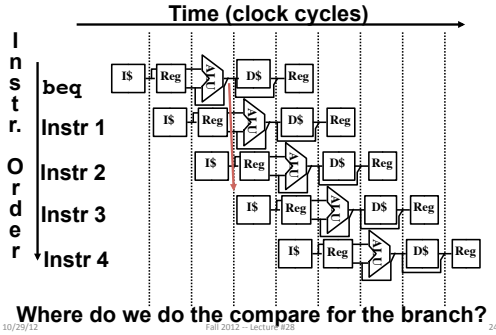
Agenda

- Structural and Data Hazards
- Administrivia
- Control Hazards
- And in Conclusion, …

3. Control Hazards

- Branch determines flow of control
  - Fetching next instruction depends on branch outcome
  - Pipeline can't always fetch correct instruction
    - Still working on ID stage of branch
- BEQ, BNE in MIPS pipeline
- Simple solution Option 1: *Stall* on every branch until have new PC value
  - Would add 2 bubbles/clock cycles for every Branch! (~ 20% of instructions executed)

Stall => 2 Bubbles/Clocks

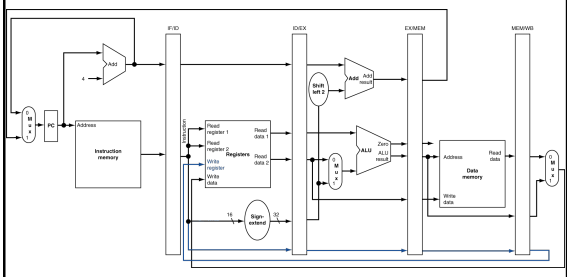


Where do we do the compare for the branch?

3. Control Hazard: Branching

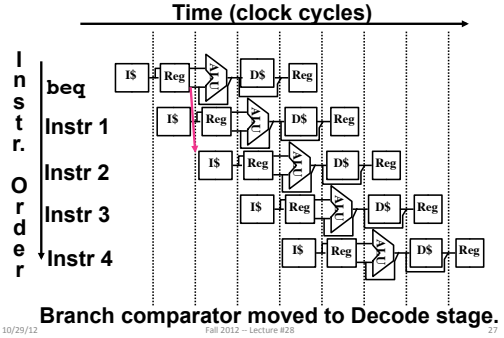
- Optimization #1:
  - Insert special branch comparator in Stage 2
  - As soon as instruction is decoded (Opcode identifies it as a branch), immediately make a decision and set the new value of the PC
  - Benefit: since branch is complete in Stage 2, only one unnecessary instruction is fetched, so only one no-op is needed
  - Side Note: means that branches are idle in Stages 3, 4 and 5

Corrected Datapath for BEQ/BNE?



Student Roulette?

One Clock Cycle Stall



3. Control Hazards

- Option 2: *Predict* outcome of a branch, fix up if guess wrong
  - Must cancel all instructions in pipeline that depended on guess that was wrong
- Simplest hardware if we predict that all branches are NOT taken
  - Why?

Student Roulette?

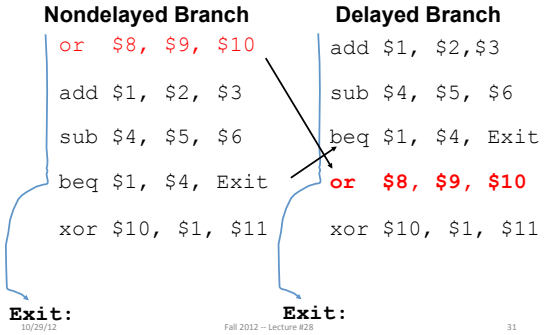
3. Control Hazard: Branching

- Option #3: Redefine branches
  - Old definition: if we take the branch, none of the instructions after the branch get executed by accident
  - New definition: whether or not we take the branch, the single instruction immediately following the branch gets executed (the *branch-delay slot*)
- *Delayed Branch* means *we always execute inst after branch*
- This optimization is used with MIPS

3. Control Hazard: Branching

- Notes on Branch-Delay Slot
  - Worst-Case Scenario: put a no-op in the branch-delay slot
  - Better Case: place some instruction preceding the branch in the branch-delay slot—as long as the changed doesn't affect the logic of program
    - Re-ordering instructions is common way to speed up programs
    - Compiler usually finds such an instruction 50% of time
    - Jumps also have a delay slot …

Example: Nondelayed vs. Delayed Branch



Delayed Branch/Jump and MIPS ISA?

- Why does JAL put PC+8 in register 31?

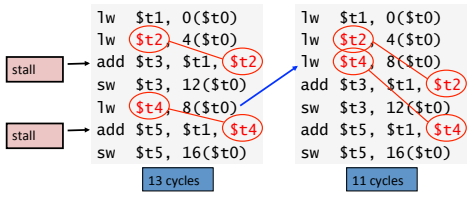
Student Roulette?

Delayed Branch/Jump and MIPS ISA?

- Why does JAL put PC+8 in register 31?
- *JAL executes following instruction (PC+4) so should return to PC+8*

Code Scheduling to Avoid Stalls

- Reorder code to avoid use of load result in the next instruction
- C code for A = B + E; C = B + F;



Peer Instruction

I. Thanks to pipelining, I have reduced the time it took me to wash my one shirt.

II. Longer pipelines are always a win (since less work per stage & a faster clock).

A)(orange) I is True and II is True
B)(green) I is False and II is True
C)(pink) I is True and II is False
D)(yellow) I is False and II is False

Peer Instruction Answer

1) Throughput better, not latency!

2) "…longer pipelines do usually mean faster clock rate, but hazards can cause problems!"

1) Thanks to pipelining, I have reduced the time it took me to wash my one shirt. FALSE

2) Longer pipelines are always a win (since less work per stage & a faster clock). FALSE

	12
a)	FF
b)	FT
c)	TF
d)	TT

And in Conclusion, …

The BIG Picture

- Pipelining improves performance by increasing instruction throughput: exploits ILP
  - Executes multiple instructions in parallel
  - Each instruction has the same latency
- Subject to hazards
  - Structure, data, control
- Stalls reduce performance
  - But are required to get correct results
- Compiler can arrange code to avoid hazards and stalls
  - Requires knowledge of the pipeline structure