CMSC 411
Computer Systems Architecture
Lecture 5
Basic Pipelining (cont.)

Alan Sussman
als@cs.umd.edu

Administrivia

- Homework problems for Unit 1 due Thursday
- Continue reading Appendix C

Speed Up Equation for Pipelining

$$CPI_{pipelined} = \text{Ideal CPI} + \text{Average Stall cycles per Inst}$$

$$\text{Speedup} = \frac{\text{Ideal CPI} \times \text{Pipeline depth}}{\text{Ideal CPI} + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{unpipelined}}{\text{Cycle Time}_{pipelined}}$$

For simple RISC pipeline, ideal CPI = 1:

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{unpipelined}}{\text{Cycle Time}_{pipelined}}$$

Example: Dual-port vs. Single-port

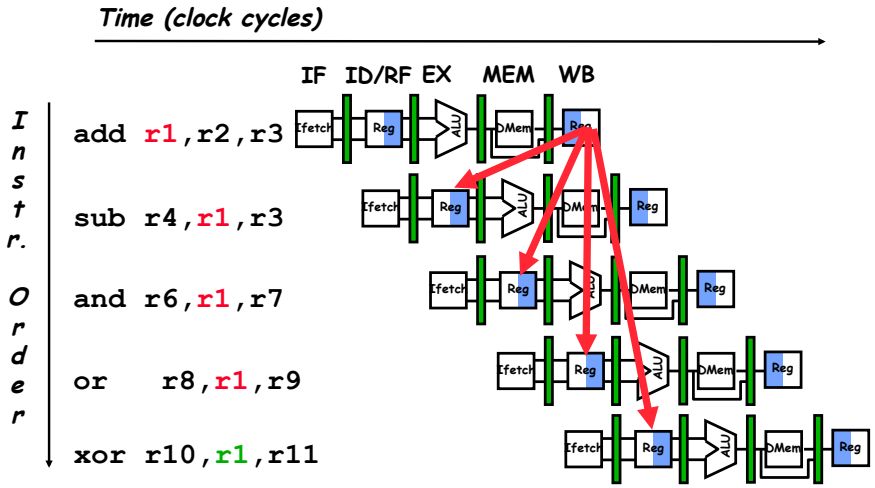
- Machine A: Dual ported memory ("Harvard Architecture")
- Machine B: Single ported memory, but its pipelined implementation has a 1.05 times faster clock rate
- Ideal CPI = 1 for both
- Loads are 40% of instructions executed

  $SpeedUp_A$ = Pipeline Depth/(1 + 0) x ($clock_{unpipe}$/$clock_{pipe}$)
  = Pipeline Depth

  $SpeedUp_B$ = Pipeline Depth/(1 + 0.4 x 1) x ($clock_{unpipe}$/($clock_{unpipe}$ / 1.05)
  = (Pipeline Depth/1.4) x 1.05
  = 0.75 x Pipeline Depth

  $SpeedUp_A$ / $SpeedUp_B$ = Pipeline Depth/(0.75 x Pipeline Depth) = 1.33
- Machine A is 1.33 times faster

Data Hazard on R1



Three Generic Data Hazards

- **Read After Write (RAW)**
  $Instr_J$ tries to read operand before $Instr_I$ writes it

```
I: add r1,r2,r3
J: sub r4,r1,r3
```

- **Caused by a "Dependence" (in compiler nomenclature). This hazard results from an actual need for communication.**

Three Generic Data Hazards

- **Write After Read (WAR)**
  $Instr_J$ writes operand *before* $Instr_I$ reads it

```
I: sub r4,r1,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- **Called an "anti-dependence" by compiler writers. This results from reuse of the name "r1".**
- **Can't happen in MIPS 5 stage pipeline because:**
  - **All instructions take 5 stages, and**
  - **Reads are always in stage 2, and**
  - **Writes are always in stage 5**

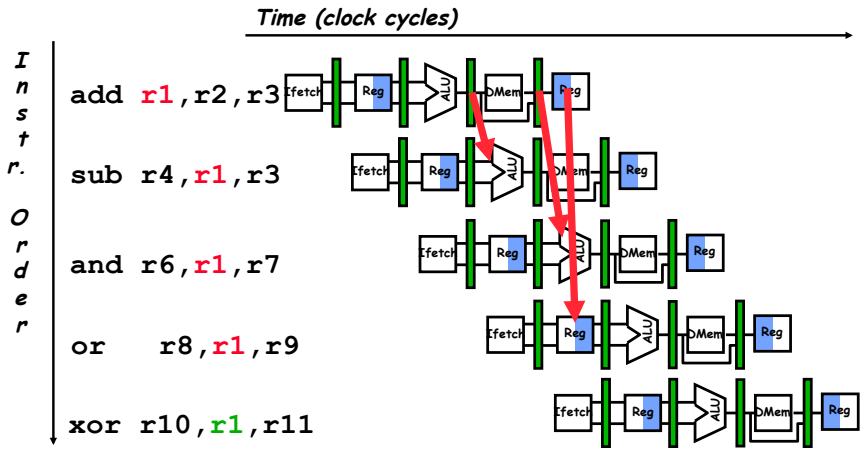
Three Generic Data Hazards

- **Write After Write (WAW)**
  $Instr_J$ writes operand *before* $Instr_I$ writes it.

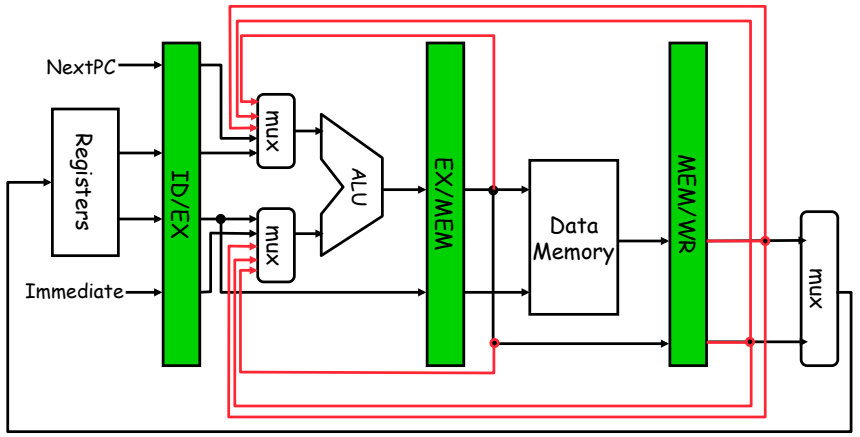
```
I: sub r1,r4,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- **Called an "output dependence" by compiler writers This also results from the reuse of name "r1".**
- **Can't happen in MIPS 5 stage pipeline because:**
  - **All instructions take 5 stages, and**
  - **Writes are always in stage 5**
- **Will see WAR and WAW in more complicated pipes**

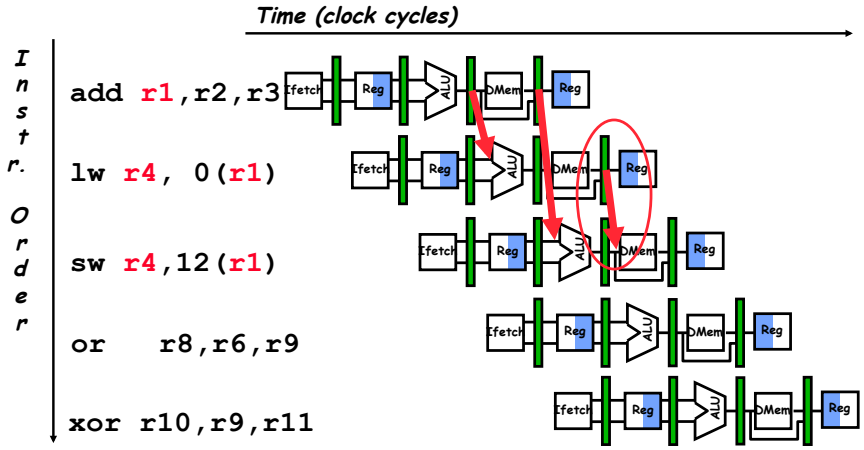
Forwarding to Avoid Data Hazard



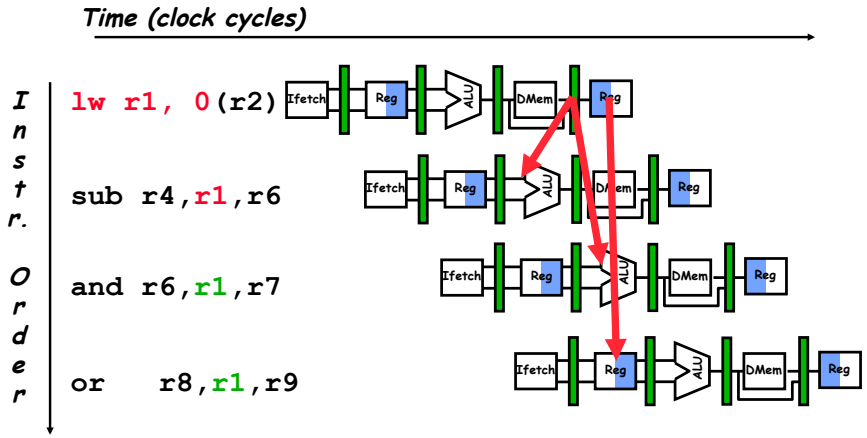
HW Change for Forwarding



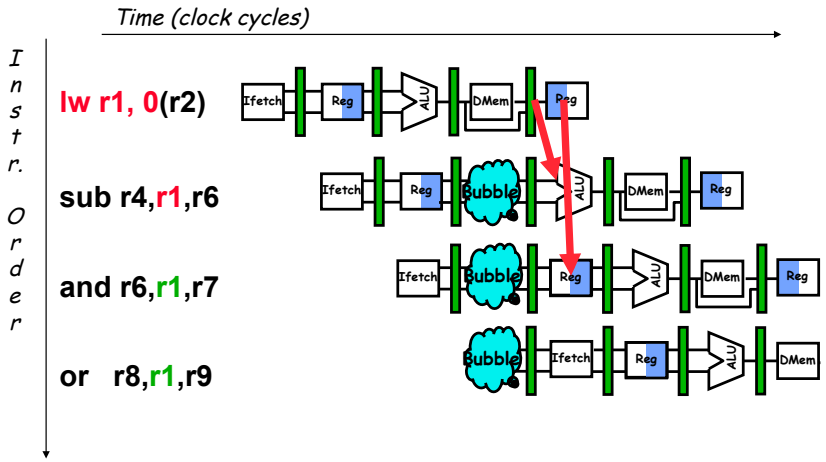
Forwarding to Avoid LW-SW Data Hazard



Data Hazard Even with Forwarding



Data Hazard Even with Forwarding



Software Scheduling Instead

Try producing fast code for

a = b + c;

d = e – f;

assuming a, b, c, d ,e, and f in memory.

Slow code:		Fast code:	
LW	Rb,b	LW	Rb,b
LW	Rc,c	LW	Rc,c
ADD	Ra,Rb,Rc	LW	Re,e
SW	a,Ra	ADD	Ra,Rb,Rc
LW	Re,e	LW	Rf,f
LW	Rf,f	SW	a,Ra
SUB	Rd,Re,Rf	SUB	Rd,Re,Rf
SW	d,Rd	SW	d,Rd

Compiler optimizes for performance. Hardware checks for safety.

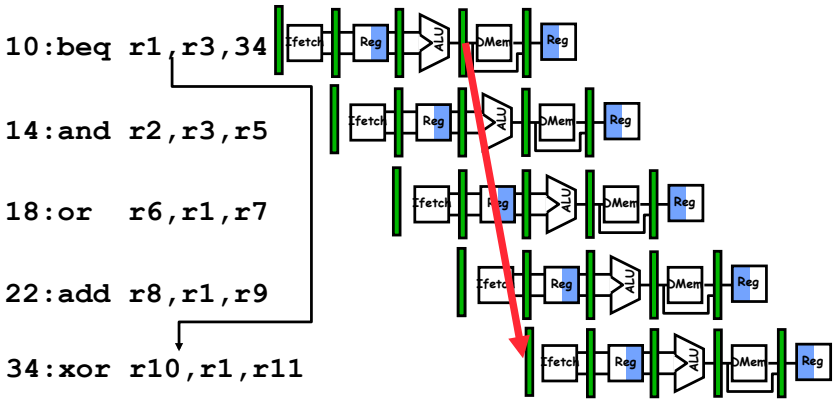
Control hazards

- Question: **When do we find out that the PC needs to be modified?**
  - Answer: **In pipeline stage ID of a branch instruction**
  - **So, if a branch is not-taken (i.e., if the PC is not modified), need a one-cycle delay**
- **Question: When is a taken branch's address known?**
  - **ALU used to compute, so EX stage**
  - **Need two (or three) cycle delay**

Example

- **If branch in 30% of instructions, then instead of executing 1 instruction per cycle, have 70% of instructions executing in 1 cycle and 30% of instructions executing in 2 cycles**
- **An average of .7 + .6 = 1.3 cycles per instruction**
  - **Worse by 30%**

Control Hazard on Branches Three Stage Stall



What do you do with the 3 instructions in between?

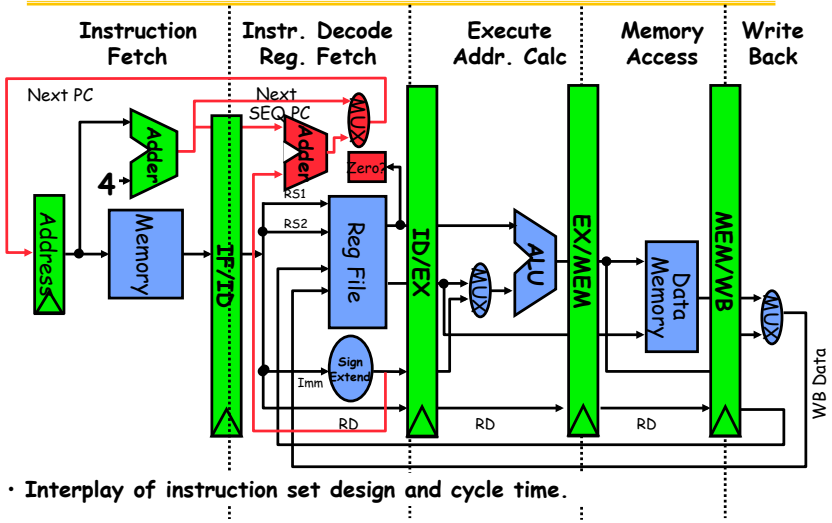
How do you do it?

Where is the "commit"?

Branch Stall Impact

- If CPI = 1, 30% branch,
  Stall 3 cycles => new CPI = 1.9!
- Two part solution:
  - Determine branch taken or not taken sooner, AND
  - Compute taken branch address earlier
- MIPS branch tests if register = 0 or ≠ 0
- MIPS Solution:
  - Move Zero test to ID/RF stage
  - Adder to calculate new PC in ID/RF stage
  - 1 clock cycle penalty for branch versus 3

Pipelined MIPS Datapath



- Interplay of instruction set design and cycle time.

Four Branch Hazard Alternatives

#1: Stall until branch direction is clear

#2: Predict Branch Not Taken
- Execute successor instructions in sequence
- "Squash" instructions in pipeline if branch actually taken
- Advantage of late pipeline state update
- 47% MIPS branches not taken on average
- PC+4 already calculated, so use it to get next instruction

#3: Predict Branch Taken
- 53% MIPS branches taken on average
- But haven't calculated branch target address in MIPS
  - » MIPS still incurs 1 cycle branch penalty
  - » Other machines: branch target known before outcome

Four Branch Hazard Alternatives

#4: Delayed Branch

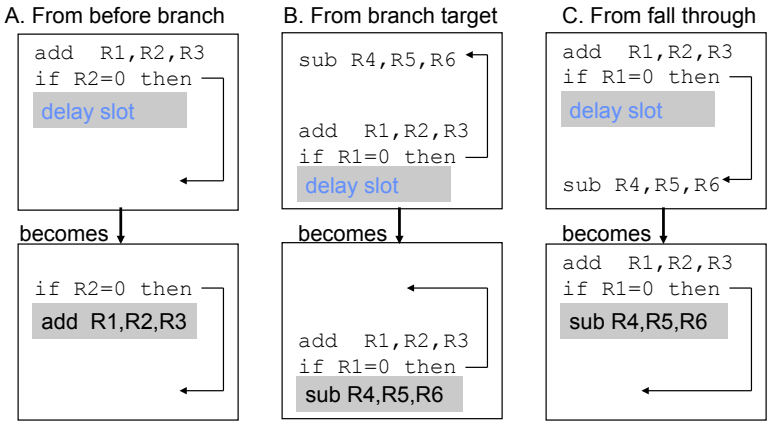
- Define branch to take place AFTER a following instruction

```
branch instruction
  sequential successor1
  sequential successor2
  ........
  sequential successorn
branch target if taken
```

Branch delay of length *n*

- 1 slot delay allows proper decision and branch target address in 5 stage pipeline
- MIPS uses this

Scheduling Branch Delay Slots



- A is the best choice, fills delay slot & reduces instruction count (IC)
- In B, the `sub` instruction may need to be copied, increasing IC
- In B and C, must be okay to execute `sub` when branch fails