

Data Hazards

```

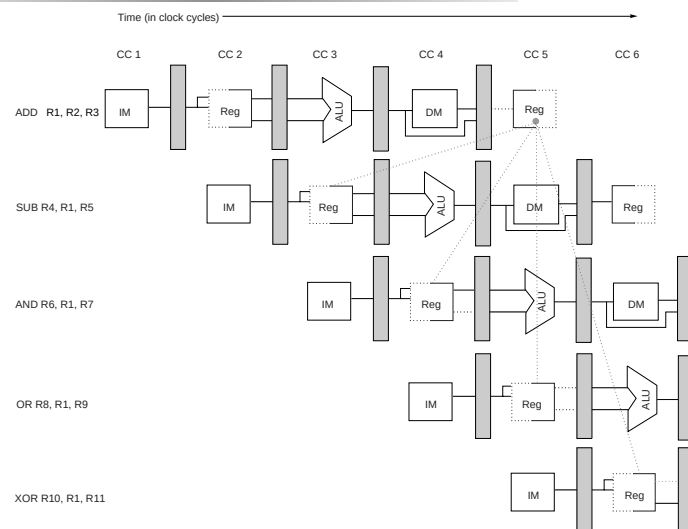
ADD  R1, R2, R3
SUB  R4, R5, R1
AND  R6, R1, R7
OR   R8, R1, R9
XOR  R10, R1, R11
    
```

Pipeline Hazards

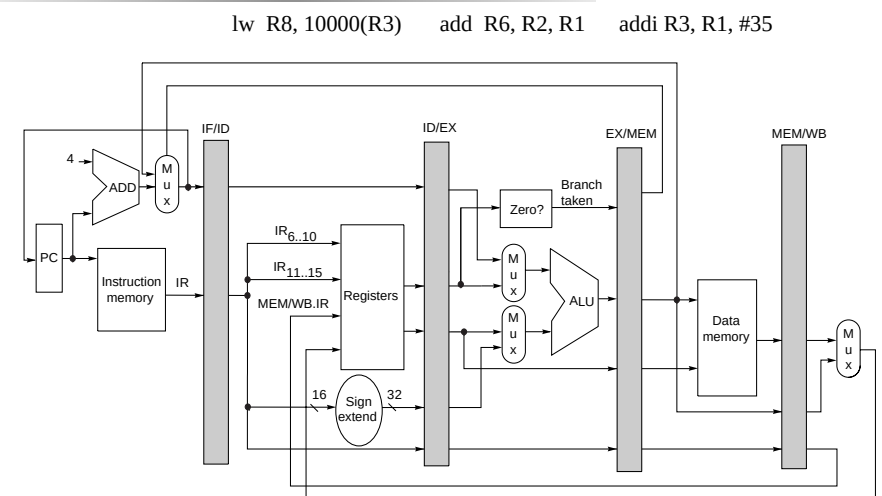
or

Danger! Danger! Danger!

Data Hazards



Data Hazard



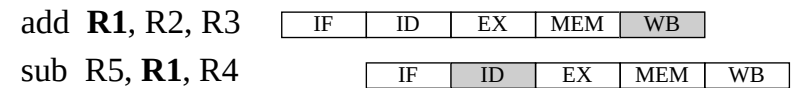
Data Dependence

- Data hazards are caused by data dependences
- Data dependences, and thus data hazards, come in 3 flavors
 - RAW (read-after-write)
 - WAW (write-after-write)
 - WAR (write-after-read)

RAW Hazard

- later instruction tries to read an operand before earlier instruction writes it
- The dependence
 - add **R1**, R2, R3
 - sub R5, **R1**, R4

- The hazard

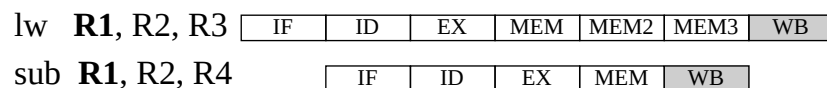


- RAW hazard is extremely common

WAW Hazard

- later instruction tries to write an operand before earlier instruction writes it
- The dependence
 - add **R1**, R2, R3
 - sub **R1**, R2, R4

- The hazard



- WAW hazard is possible in a reasonable pipeline

WAR Hazard

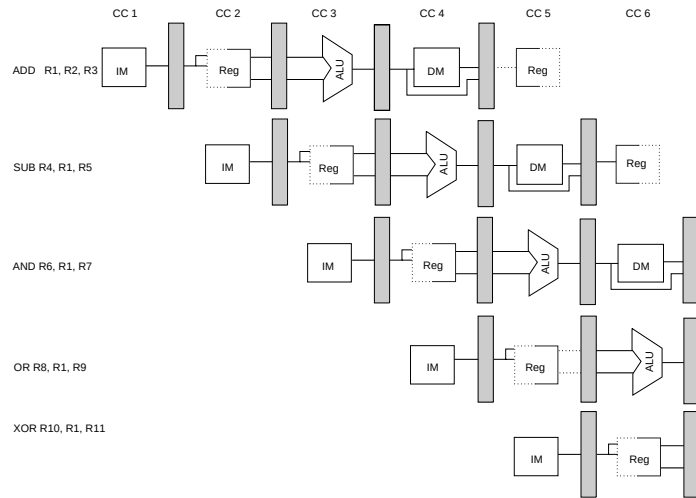
- later instruction tries to write an operand before earlier instruction reads it
- The dependence
 - add R1, **R2**, R3
 - sub **R2**, R5, R4

- The hazard?



- WAR hazard is uncommon/impossible in a reasonable (in-order) pipeline

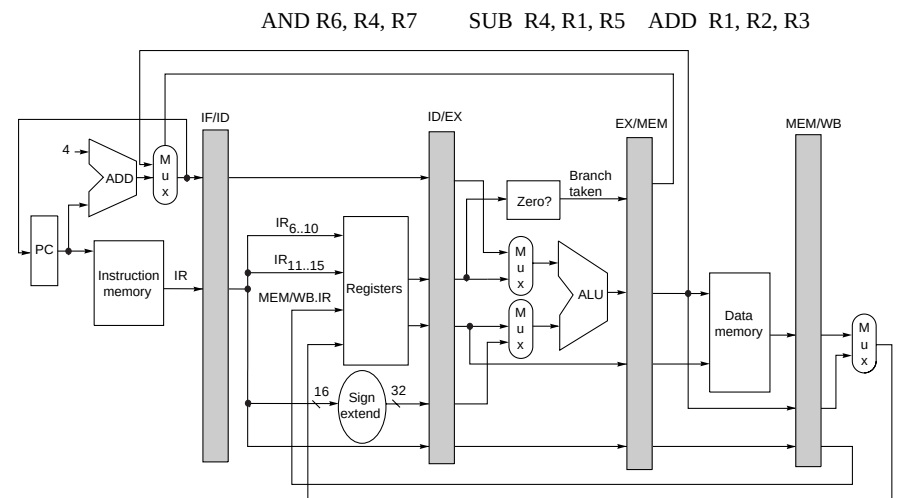
Dealing with Data Hazards through Forwarding



CSE 240

Dean Tullsen

Dealing with Data Hazards through Forwarding



CSE 240

Dean Tullsen

Forwarding Options

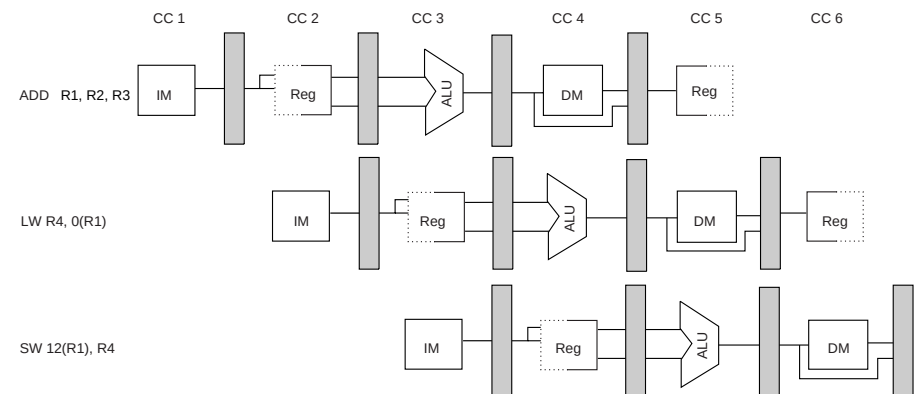
- ADD -> ADD
- ADD -> LW
- ADD -> SW (2 operands)
- LW -> ADD
- LW -> LW
- LW -> SW (2 operands)

(I'm letting ADD stand in for all ALU operations)

CSE 240

Dean Tullsen

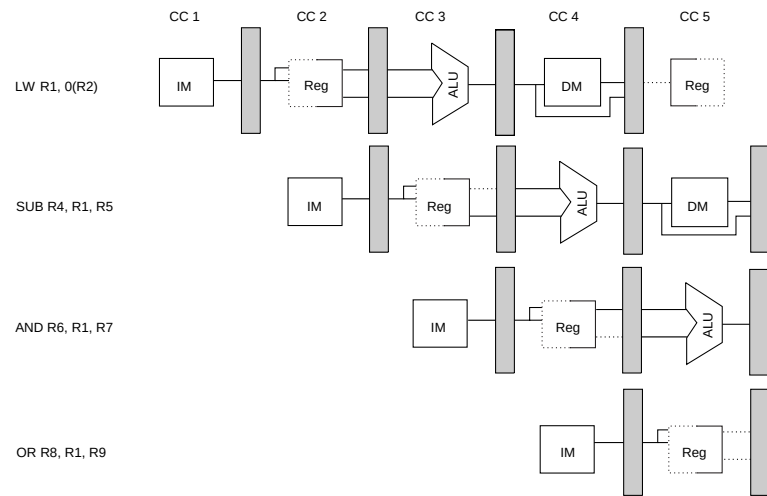
More Forwarding



CSE 240

Dean Tullsen

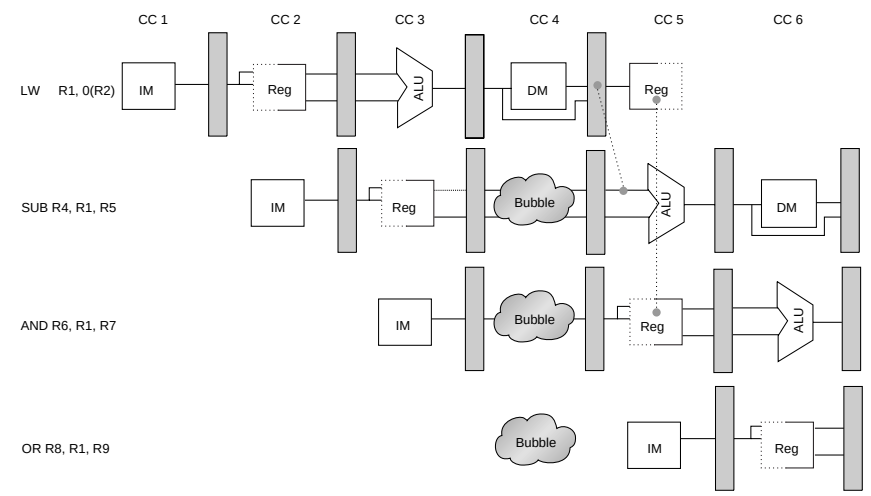
More Forwarding



CSE 240

Dean Tullsen

Forwarding and Stalling



CSE 240

Dean Tullsen

Example

ADD R1, R2, R3	IF	ID	EX	MEM	WB	
SW R1, 1000(R2)		IF	ID	EX	MEM	WB
LW R7, 2000(R2)						
ADD R5, R7, R1						
LW R8, 2004(R2)						
SW R7, 2008(R8)						
ADD R8, R8, R2						
LW R9, 1012(R8)						
SW R9, 1016(R8)						

CSE 240

Dean Tullsen

Avoiding Pipeline Stalls

lw R1, 1000(R2)
 lw R3, 2000(R2)
 add R4, R1, R3
 lw R1, 3000(R2)
 add R6, R4, R1
 sw R6, 1000(R2)

- this is a compiler technique called *instruction scheduling*.

CSE 240

Dean Tullsen

How big a problem are these pipeline stalls?

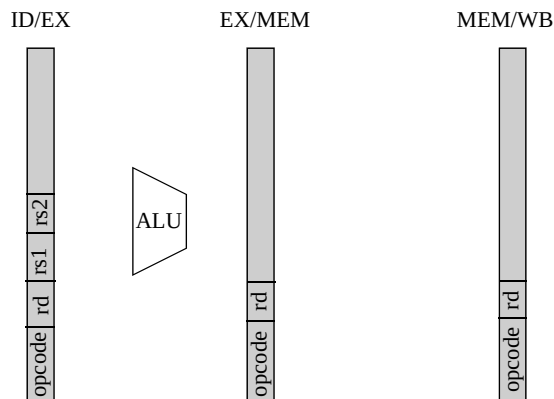
- 13% of the loads in FP programs
- 25% of the loads in integer programs

Detecting ALU Input Hazards

ALUop	R?, R?, R?	
ALUop	R?, R?, R?	
ALUop	R?, R?, R?	lw R?, disp(R?)
		lw R?, disp(R?)
		ALUop R?, R?, R?

ALUop	R?, R?, R?
ALUop	R?, R?, R?
lw/sw	R?, disp(R?)

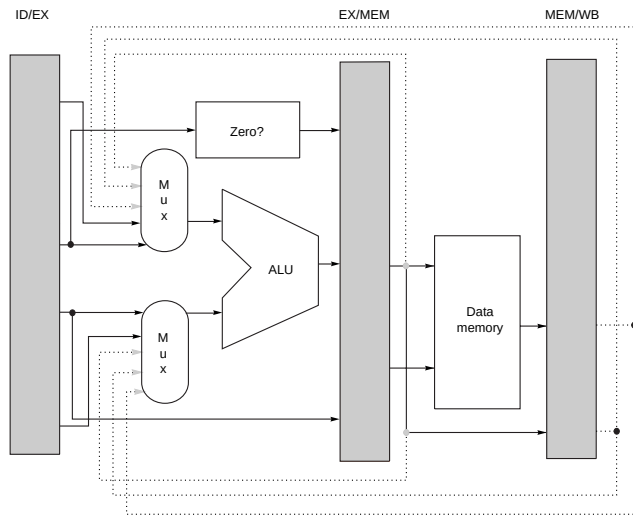
Detecting ALU Input Hazards



Inserting Bubbles

- Set all control values in the EX/MEM register to zero (inserts a no-op instruction)
- Keep same values in the ID/EX register and IF/ID register
- Keep PC from incrementing

Adding Datapaths

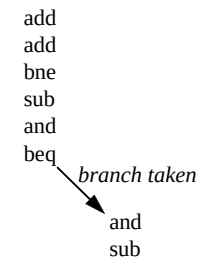


CSE 240

Dean Tullsen

Control Hazards

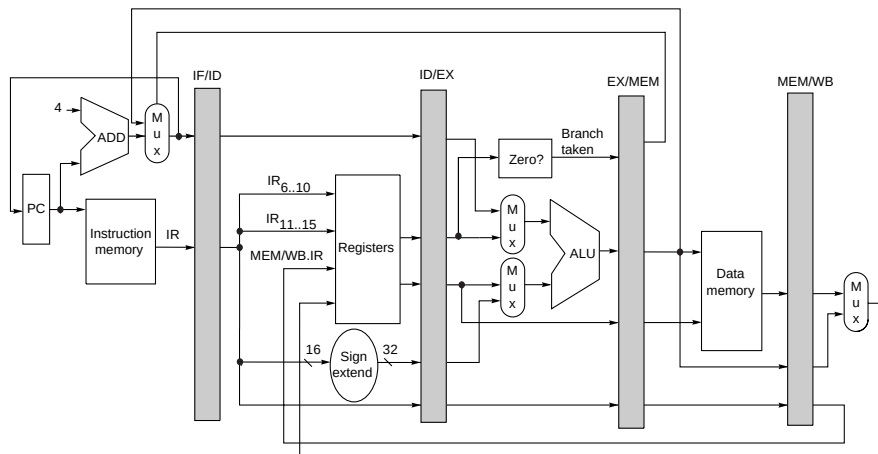
- Instructions are not only dependent on instructions that produce their operands, but also on all previous control flow (branch, jump) instructions that lead to that instruction.



CSE 240

Dean Tullsen

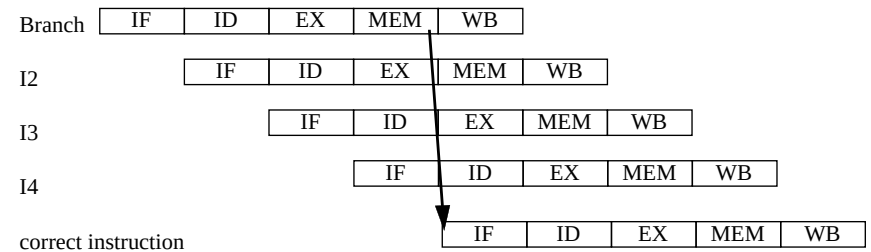
Old Datapath



CSE 240

Dean Tullsen

Branch Hazards



CSE 240

Dean Tullsen

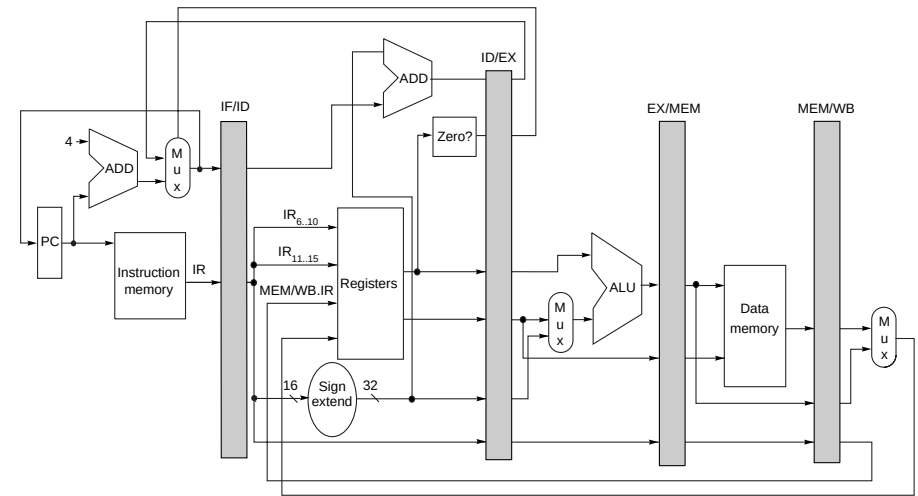
Branch Stall Impact

- If CPI = 1, 30% branch, Stall 3 cycles => new CPI = 1.9!
- Two part solution:
 - Determine branch taken or not sooner, AND
 - Compute taken branch address earlier
- DLX branch tests if register = 0 or $\neq 0$
- DLX 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

CSE 240

Dean Tullsen

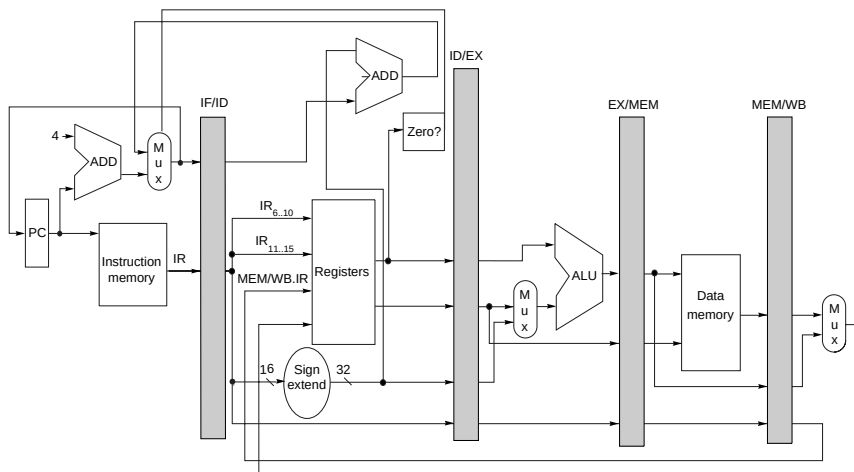
New Datapath?



CSE 240

Dean Tullsen

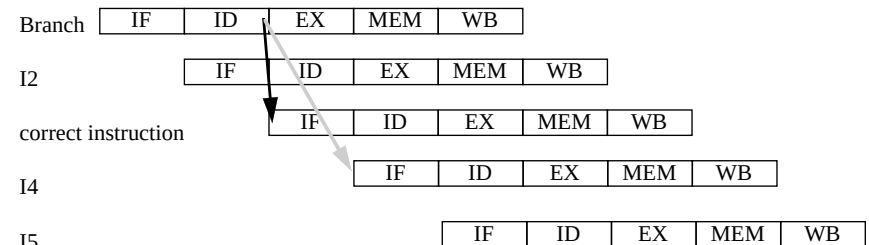
New Datapath



CSE 240

Dean Tullsen

Branch Hazards



CSE 240

Dean Tullsen

What We Know About Branches

- more conditional branches than unconditional
- more forward than backward
- 67% of branches taken
- backward branches taken 80%

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
- 33% DLX branches not taken on average
- PC+4 already calculated, so use it to get next instruction

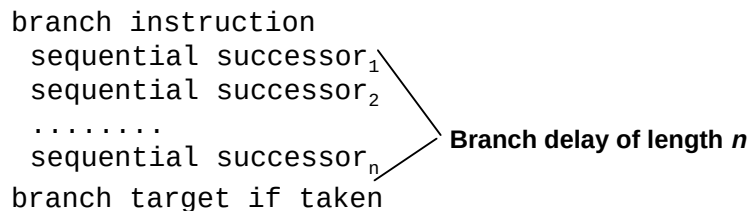
#3: Predict Branch Taken

- 67% DLX branches taken on average
- *But haven’t calculated branch target address in DLX*
 - DLX 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



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

Delayed Branch

● Where to get instructions to fill branch delay slot?

- Before branch instruction
- From the target address: only valuable when branch taken
- From fall through: only valuable when branch not taken
- Cancelling branches allow more slots to be filled

● Compiler effectiveness for single branch delay slot:

- Fills about 60% of branch delay slots
- About 80% of instructions executed in branch delay slots useful in computation
- About 50% (60% x 80%) of slots usefully filled