

UNIVERSITY OF MASSACHUSETTS
Dept. of Electrical & Computer Engineering

Advanced Computer Architecture
ECE 568

Part 4

Control Hazards

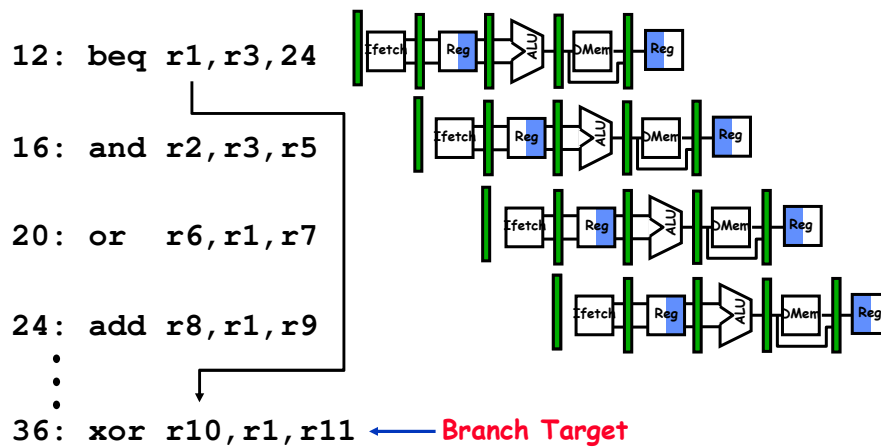
Israel Koren
Fall 2011

ECE568/Koren Part.4.1

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Control Hazard on Branches



ECE568/Koren Part.4.2

Adapted from UCB and other sources

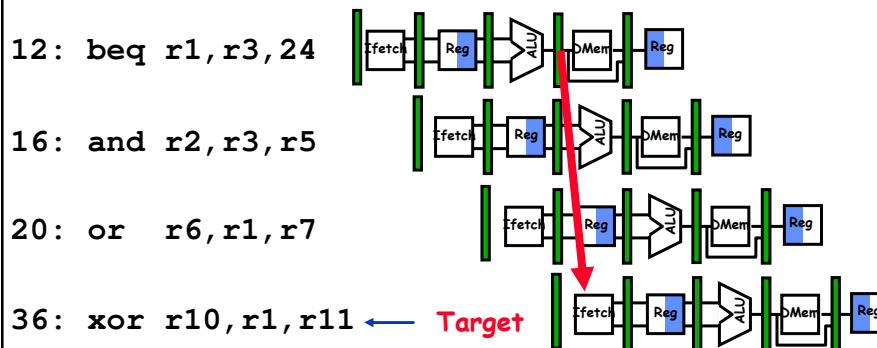
Copyright UCB & Morgan Kaufmann

MIPS pipeline (Fig.A.19)

Stage	Any instruction						
IF	IF/ID.IR \leftarrow Mem[PC]; IF/ID.NPC,PC \leftarrow (if ((EX/MEM.opcode == branch) & EX/MEM.cond){EX/MEM.ALUOutput} else {PC+4});						
ID	ID/EX.A \leftarrow Regs[IF/ID.IR[rs]]; ID/EX.B \leftarrow Regs[IF/ID.IR[rt]]; ID/EX.NPC \leftarrow IF/ID.NPC; ID/EX.IR \leftarrow IF/ID.IR; ID/EX.Imm \leftarrow sign-extend(IF/ID.IR[immediate field]);						
	<table border="0" style="width: 100%;"> <tr> <td style="width: 33%;">ALU instruction</td> <td style="width: 33%;">Load or store instruction</td> <td style="width: 33%;">Branch instruction</td> </tr> <tr> <td>EX/MEM.IR \leftarrow ID/EX.IR; EX/MEM.ALUOutput \leftarrow ID/EX.A <i>func</i> ID/EX.B; or EX/MEM.ALUOutput \leftarrow ID/EX.A <i>op</i> ID/EX.Imm;</td> <td>EX/MEM.IR \leftarrow ID/EX.IR EX/MEM.ALUOutput \leftarrow ID/EX.A + ID/EX.Imm; EX/MEM.B \leftarrow ID/EX.B;</td> <td>EX/MEM.ALUOutput \leftarrow ID/EX.NPC + (ID/EX.Imm \ll 2); EX/MEM.cond \leftarrow (ID/EX.A == 0);</td> </tr> </table>	ALU instruction	Load or store instruction	Branch instruction	EX/MEM.IR \leftarrow ID/EX.IR; EX/MEM.ALUOutput \leftarrow ID/EX.A <i>func</i> ID/EX.B; or EX/MEM.ALUOutput \leftarrow ID/EX.A <i>op</i> ID/EX.Imm;	EX/MEM.IR \leftarrow ID/EX.IR EX/MEM.ALUOutput \leftarrow ID/EX.A + ID/EX.Imm; EX/MEM.B \leftarrow ID/EX.B;	EX/MEM.ALUOutput \leftarrow ID/EX.NPC + (ID/EX.Imm \ll 2); EX/MEM.cond \leftarrow (ID/EX.A == 0);
ALU instruction	Load or store instruction	Branch instruction					
EX/MEM.IR \leftarrow ID/EX.IR; EX/MEM.ALUOutput \leftarrow ID/EX.A <i>func</i> ID/EX.B; or EX/MEM.ALUOutput \leftarrow ID/EX.A <i>op</i> ID/EX.Imm;	EX/MEM.IR \leftarrow ID/EX.IR EX/MEM.ALUOutput \leftarrow ID/EX.A + ID/EX.Imm; EX/MEM.B \leftarrow ID/EX.B;	EX/MEM.ALUOutput \leftarrow ID/EX.NPC + (ID/EX.Imm \ll 2); EX/MEM.cond \leftarrow (ID/EX.A == 0);					
MEM	MEM/WB.IR \leftarrow EX/MEM.IR; MEM/WB.ALUOutput \leftarrow EX/MEM.ALUOutput; MEM/WB.IR \leftarrow EX/MEM.IR; MEM/WB.LMD \leftarrow Mem[EX/MEM.ALUOutput]; or MEM/WB.LMD \leftarrow Mem[EX/MEM.ALUOutput] \leftarrow EX/MEM.B;						
WB	Regs[MEM/WB.IR[rd]] \leftarrow MEM/WB.ALUOutput; or Regs[MEM/WB.IR[rt]] \leftarrow MEM/WB.ALUOutput; For load only: Regs[MEM/WB.IR[rt]] \leftarrow MEM/WB.LMD;						

ECE568/Koren Part.4.3 Adapted from UCB and other sources Copyright UCB & Morgan Kaufmann

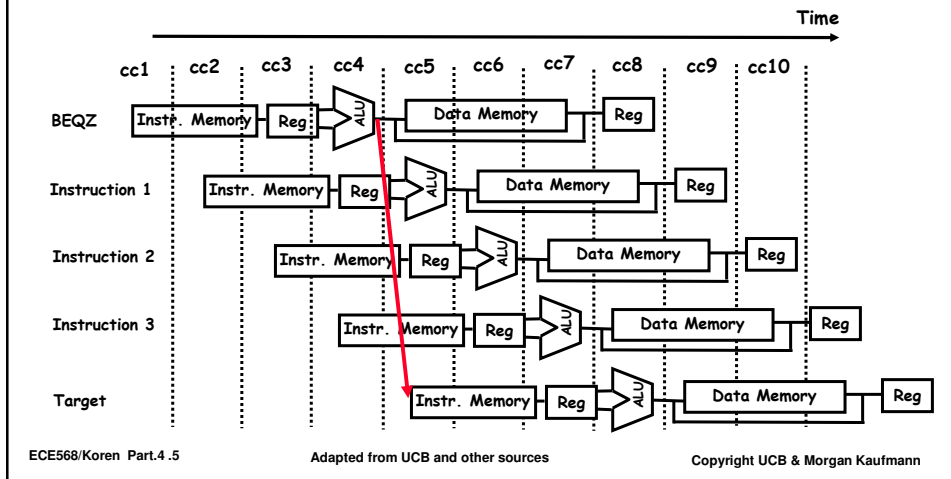
Control Hazard on Branches Two Stage Stall



Freeze or Flush (higher performance but must undo side effects)

Branch Stall Impact - MIPS R4000

- ◆ Deeper pipeline -> worse penalty



Branch Stall Impact - MIPS R4000

- ◆ Assume CPI = 1.0 ignoring branches & data hazards
- ◆ Assume solution was stalling for 3 cycles for every branch
- ◆ If 20% branch, Stall 3 cycles

Op	Freq	Cycles	CPI(i)
Other	80%	1	0.8
Branch	20%	4	0.8

⇒ new CPI = 1.6 or 60% slower

Conditional and unconditional Branch

- ◆ Unconditional branches incur lower penalty
 - Assume 2 cycles vs 3 for unconditional branch
- ◆ Avg. Stall cycles = $\sum \text{Branch_frequency} \times \text{Branch_penalty}$

Branch_type	Freq.	Penalty	Freq × Penalty
Unconditional	0.04	2	0.08
Conditional	0.16	3	0.48
			0.56

$$\text{Pipeline_speedup} = \frac{\text{Pipeline_depth}}{1 + \sum \text{Branch_frequency} \times \text{Branch_penalty}}$$

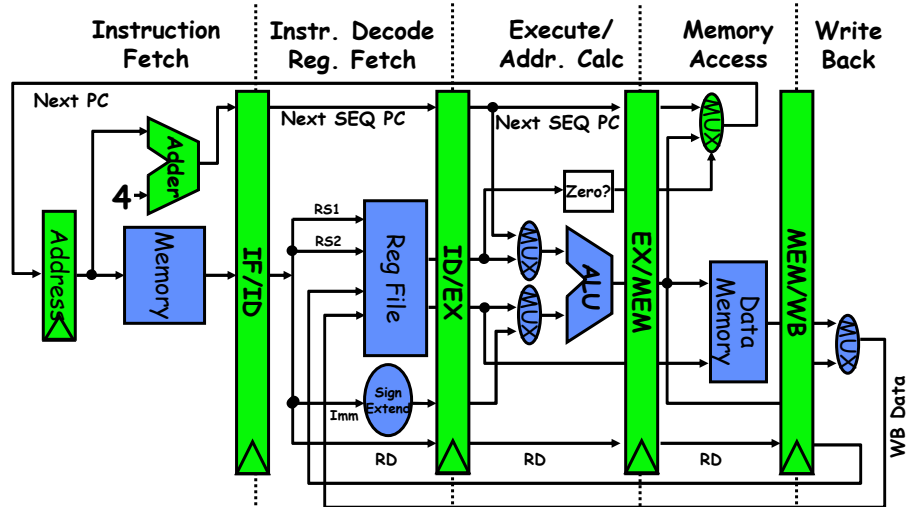
$$\text{Pipeline_speedup} = 8 / 1.56 = 5.13$$

(ignoring all other stalls)

Dealing with Branch in MIPS

- ◆ Branch penalty is significant
- ◆ Two part solution:
 - Determine branch taken or not sooner, AND
 - Compute taken branch address earlier
- ◆ MIPS branch tests if register = 0 or $\neq 0$
- ◆ 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 2

Original MIPS 5-stage pipeline

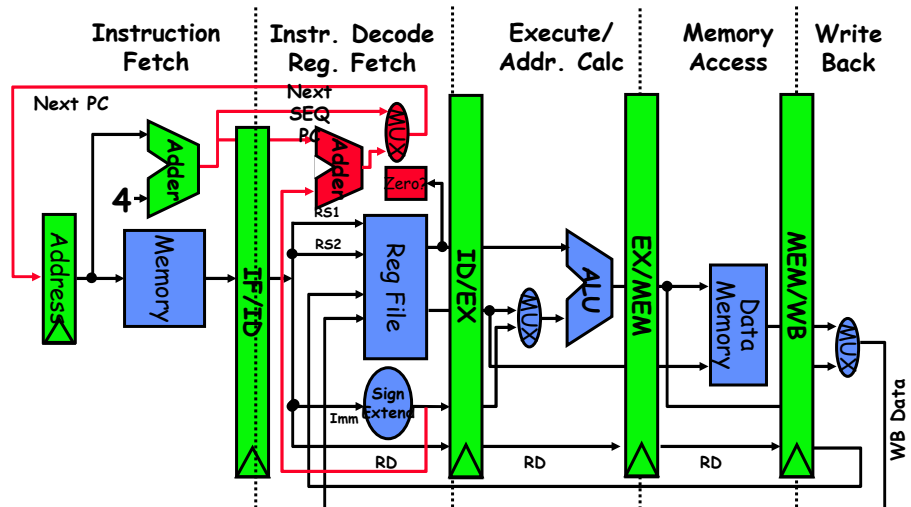


ECE568/Koren Part.4.9

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Modified MIPS Pipeline



ECE568/Koren Part.4.10

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Four Branch Hazard Alternatives - 1 & 2

#1: Stall until branch direction is clear

#2: Static Prediction:

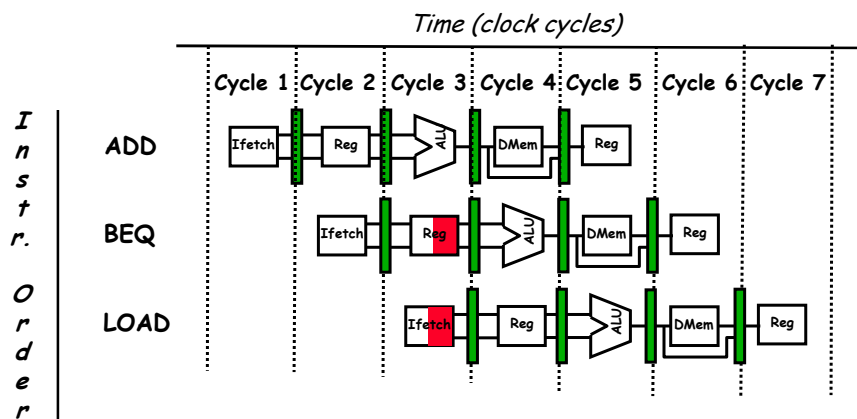
(a) Predict Branch Not Taken

- Execute successor instructions in sequence
- "Squash" instructions in pipeline if branch actually taken
- 47% MIPS branches not taken on average

(b): Predict Branch Taken

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

Static Prediction - Not Taken



- ♦ **Predict:** guess Not Taken then back up if wrong
- ♦ **Impact:** 0 lost cycles per branch instruction if right, 1 if wrong (right about 50% of time)
 - Need to "Squash" following instruction (LOAD) if wrong

Predict Not Taken

Untaken branch instruction	IF	ID	EX	MEM	WB				
Instruction $i + 1$		IF	ID	EX	MEM	WB			
Instruction $i + 2$			IF	ID	EX	MEM	WB		
Instruction $i + 3$				IF	ID	EX	MEM	WB	
Instruction $i + 4$					IF	ID	EX	MEM	WB
Taken branch instruction	IF	ID	EX	MEM	WB				
Instruction $i + 1$		IF	idle	idle	idle	idle			
Branch target			IF	ID	EX	MEM	WB		
Branch target + 1				IF	ID	EX	MEM	WB	
Branch target + 2					IF	ID	EX	MEM	WB

- ◆ CPI for branch: $(1 * .47 + 2 * .53) = 1.53$
- ◆ Total CPI might be: $1.53 * .2 + 1 * .8 = 1.106$
(20% branch)
- ◆ Compare to Always Stall (Freeze)

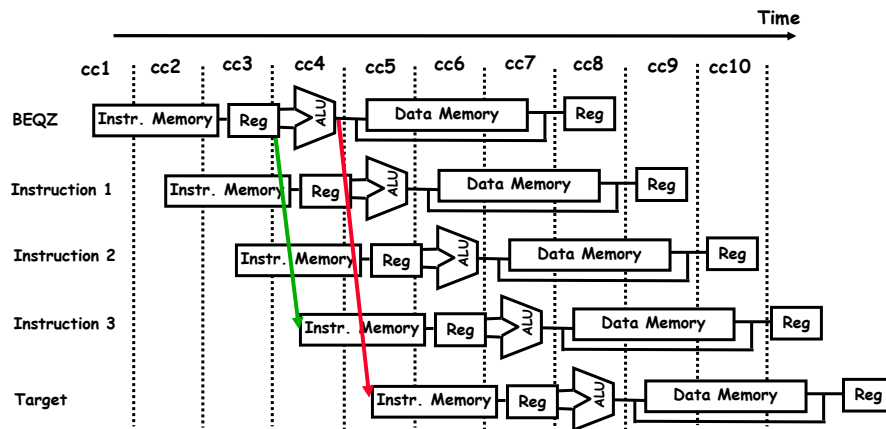
ECE568/Koren Part.4 .13

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

MIPS R4000

Predict Taken - 2 (vs. 3) stall Cycles



ECE568/Koren Part.4 .14

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

CPI Penalty - MIPS R4000

Branch scheme	Penalty unconditional	Penalty untaken	Penalty taken
Stall pipeline	2	3	3
Predicted taken	2	3	2
Predicted untaken	2	0	3

Additions to the CPI from branch costs				
Branch scheme	Unconditional branches	Untaken conditional branches	Taken conditional branches	All branches
Frequency of event	4%	6%	10%	20%
Stall pipeline	0.08	0.18	0.30	0.56
Predicted taken	0.08	0.18	0.20	0.46
Predicted untaken	0.08	0.00	0.30	0.38

$$CPI_{Stall} = 1.56, \quad CPI_{P/T} = 1.46, \quad CPI_{P/UT} = 1.38$$

$$Speedup_{(P/UT \text{ vs. Stall})} = 1.56/1.38 = 1.13$$

ECE568/Koren Part.4.15

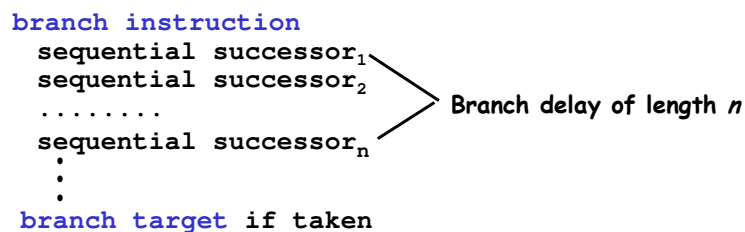
Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Four Branch Hazard Alternatives - 3 & 4

#3: Delayed Branch

- Define branch to take place **AFTER** a following instruction(s)



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

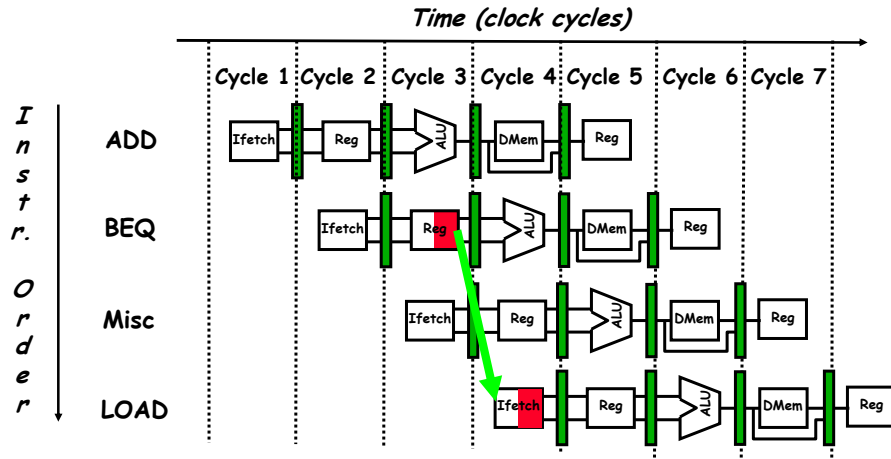
#4: Dynamic Branch Prediction

ECE568/Koren Part.4.16

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

MIPS - One Delayed Branch Slot



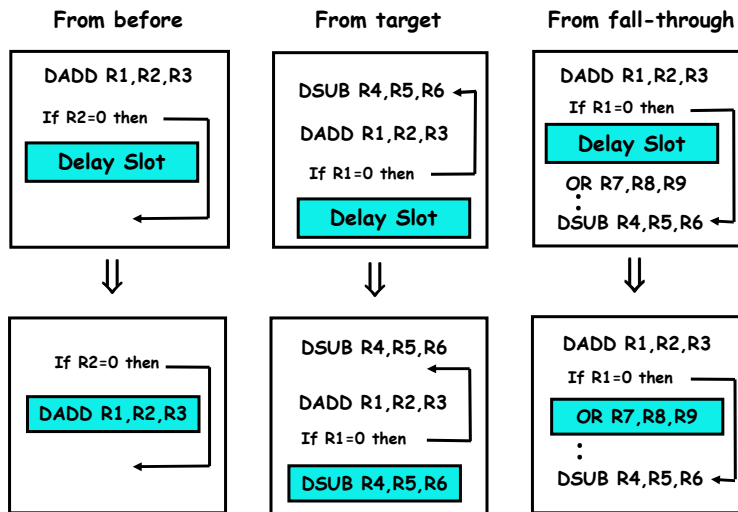
◆ Impact: 0 clock cycles per branch instruction if can find useful instruction to put in "slot" ($\approx 50\%$ of time)

ECE568/Koren Part.4.17

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Scheduling the branch delay slot



ECE568/Koren Part.4.18

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

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
 - Canceling 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 48% (60% × 80%) of slots usefully filled
- ◆ $CPI = 1 + Prob\{Branch\} * Prob\{Un-usefull_fill\} = 1 + .2 * .52 = 1.104$
- ◆ Delayed Branch downside: 8-10 stage pipelines, multiple instructions issued per clock (superscalar)

Evaluating Branch Alternatives for MIPS

UnCond: 4%, NotTaken_Cond: 6%, Taken_Cond: 10%

Slot filled usefully: 48%

$CPI \{Delayed_Branch\} = 1 + .2 * .52 = 1.104$

<i>Scheduling scheme</i>	<i>Branch penalty</i>	<i>CPI</i>	<i>speedup v. stall</i>
Stall pipeline	1	1.2	1.0
Predict taken	1	1.2	1.0
Predict not taken	0/1	1.14	1.05
Delayed branch	0.52	1.104	1.087

MIPS R4000 - Delayed Branch and Static Prediction (Not-Taken)

(2) Taken

	Clock number								
Instruction number	1	2	3	4	5	6	7	8	9
Branch instruction	IF	IS	RF	EX	DF	DS	TC	WB	
Delay slot		IF	IS	RF	EX	DF	DS	TC	WB
Stall			stall	stall	stall	stall	stall	stall	stall
Stall				stall	stall	stall	stall	stall	stall
Branch target					IF	IS	RF	EX	DF

(1) Un-Taken

	Clock number								
Instruction number	1	2	3	4	5	6	7	8	9
Branch instruction	IF	IS	RF	EX	DF	DS	TC	WB	
Delay slot		IF	IS	RF	EX	DF	DS	TC	WB
Branch instruction + 2			IF	IS	RF	EX	DF	DS	TC
Branch instruction + 3				IF	IS	RF	EX	DF	DS