18-447 Lecture 12: Pipelined Implementations: Control Hazards and Resolutions

James C. Hoe
Dept of ECE, CMU
March 2, 2009

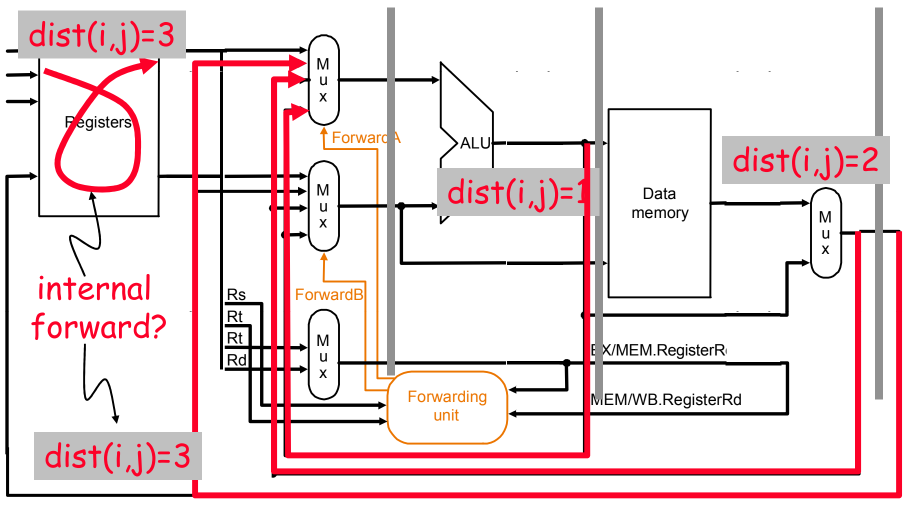
Announcements:	Spring break next week!!
	Project 2 due the week after spring break
	HW3 due Monday after spring break
	(no more homework until week 12)
Handouts:	Handout #10 Project 2 (On Blackboard)

Terminology

- Dependencies
  - ordering requirement between instructions
- Pipeline Hazards:
  - (potential) violations of dependencies
- Hazard Resolution:
  - static $\Rightarrow$ schedule instructions at compile time to avoid hazards
  - dynamic $\Rightarrow$ detect hazard and adjust pipeline operation

    Stall, Forward/byapss, anything else?
- Pipeline Interlock:
  - hardware mechanisms for dynamic hazard resolution
  - detect and enforce dependences at run time

Forwarding Paths (v1)



[Based on figures from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

Data Hazard Analysis (with Forwarding)

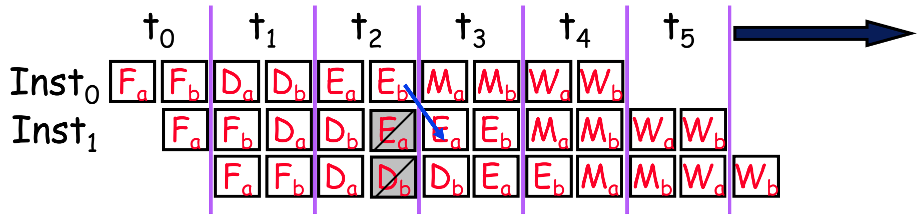
	R/I-Type	LW	SW	Br	J	Jr
IF						
ID						use
EX	use produce	use	use	use		
MEM		produce	(use)			
WB						

- Even with data-forwarding, RAW dependence on an immediate preceding LW instruction produces a hazard

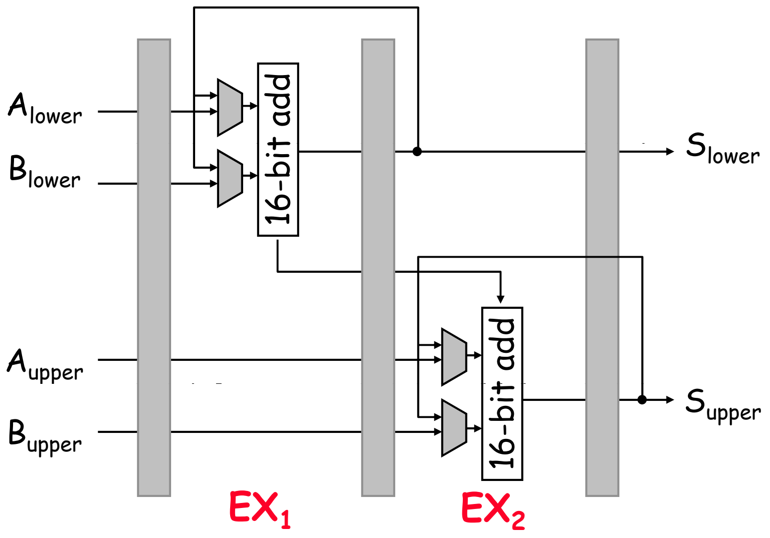
Why not very deep pipelines?

- 5-stage pipeline still has plenty of combinational delay between registers
- "Superpipelining" ⇒ increase pipelining such that even intrinsic operations (e.g. ALU, RF access, memory access) require multiple stages
- What's the problem?

Inst_0: r1 ← r2 + r3
Inst_1: r4 ← r1 + 2

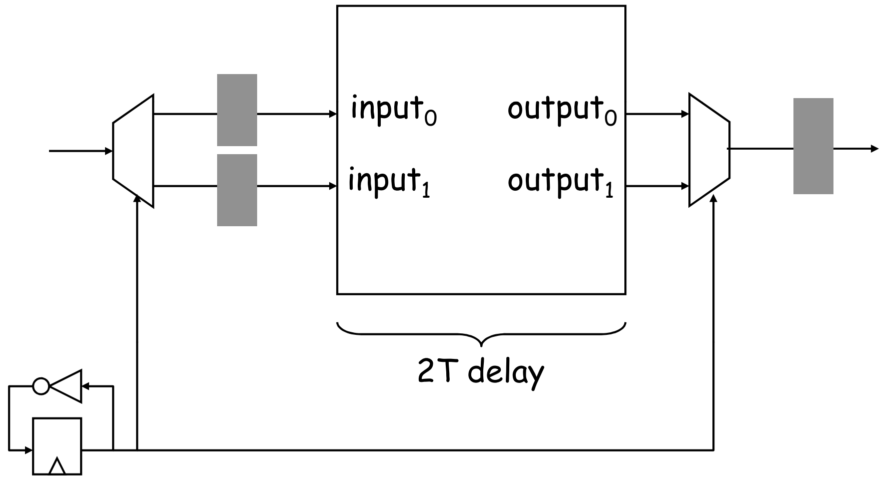

t_0
t_1
t_2
t_3
t_4
t_5
Inst_0
Inst_1
F_a F_b D_a D_b E_a E_b M_a M_b W_a W_b
F_a F_b D_a D_b E_a E_a E_b M_a M_b W_a W_b
F_a F_b D_a D_b D_b E_a E_b M_a M_b W_a W_b

Intel P4's Superpipelined Integer ALU



32-bit addition pipelined over 2 stages, BW=1/latency$_{16\text{-bit-add}}$
No stall between back-to-back dependencies

What if you really can't superpipeline?



If you can't double the bandwidth by pipelining, doubling the resource also doubles the bandwidth

Instruction Ordering/Dependencies

- Data Dependence
  - True dependence or Read after Write (RAW)
    Instruction must wait for all required input operands
  - Anti-Dependence or Write after Read (WAR)
    Later write must not clobber a still-pending earlier read
  - Output dependence or Write after Write (WAW)
    Earlier write must not clobber an already-finished later write

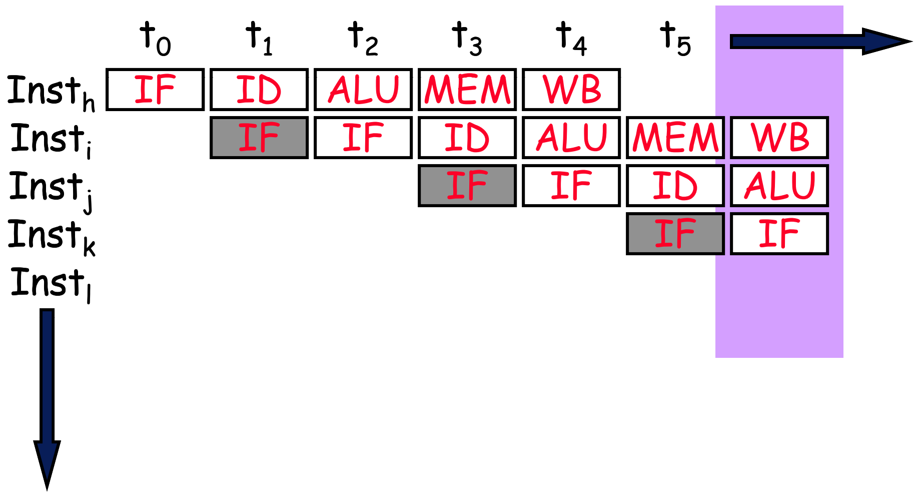
- Control Dependence (or Procedural Dependence)
  - all instructions are dependent by control flow
  - every instruction use and set the PC
    Control dependence is data dependence on the PC

PC Data Hazard Analysis

	R/I-Type	LW	SW	Br	J	Jr
IF	use	use	use	use	use	use
ID	produce	produce	produce		produce	produce
EX				produce		
MEM						
WB						

- PC hazard distance is at least 1
- Does that mean we must stall after every instruction?
  - IF stage can't know which PC to fetch next until the current PC is fetched and decoded

Control Hazard by Stalling



	t_0	t_1	t_2	t_3	t_4	t_5	
$Inst_h$	IF	ID	ALU	MEM	WB		
$Inst_i$		IF	IF	ID	ALU	MEM	WB
$Inst_j$				IF	IF	ID	ALU
$Inst_k$						IF	IF
$Inst_l$							

For non-control-flow instructions

Control Speculation for Dummies

- Rather than waiting for true-dependence on PC to resolve, just guess nextPC = PC+4 to keep fetching every cycle

  Is this a good guess?

  What do you lose if you guessed incorrectly?

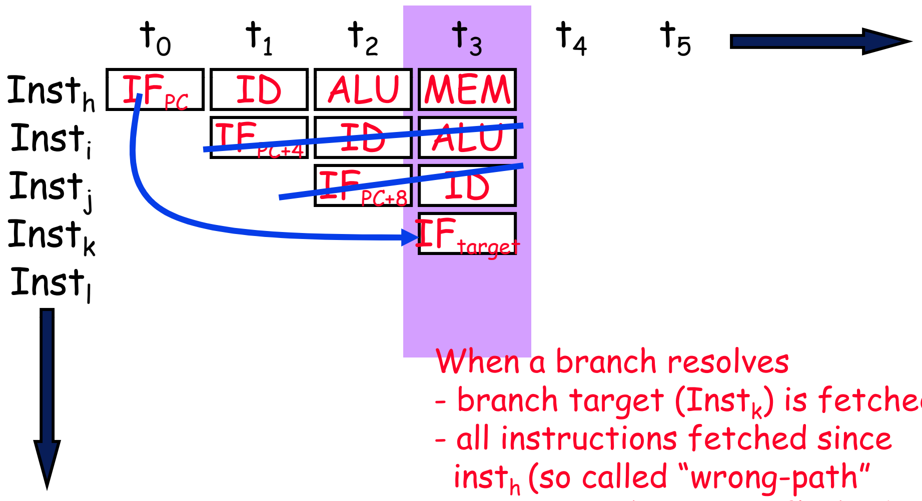
- Only ~20% of the instruction mix is control flow
  - ~50 % of "forward" control flow (i.e., if-then-else) is taken
  - ~90% of "backward" control flow (i.e., loop back) is taken

  Over all, typically ~70% taken and ~30% not taken

  [Lee and Smith, 1984]

- Expect "nextPC = PC+4" ~86% of the time, but what about the remaining 14%?

Control Speculation: PC+4



When a branch resolves
- branch target ($Inst_k$) is fetched
- all instructions fetched since $inst_h$ (so called "wrong-path" instructions) must be flushed

$Inst_h$ is a branch

Pipeline Flush on Misprediction

	t_0	t_1	t_2	t_3	t_4	t_5	
$Inst_h$	IF_{PC}	ID	ALU	MEM	WB		
$Inst_i$		IF_{PC+4}	ID	killed			
$Inst_j$			IF_{PC+8}	killed			
$Inst_k$				IF_{target}	ID	ALU	WB
$Inst_l$					IF	ID	ALU
						IF	ID
							IF

$Inst_h$ is a branch

Pipeline Flush on Misprediction

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
IF	h	i	j	k	l	m	n				
ID		h	i	bub	k	l	m	n			
EX			h	bub	bub	k	l	m	n		
MEM				h	bub	bub	k	l	m	n	
WB					h	bub	bub	k	l	m	n

branch resolved

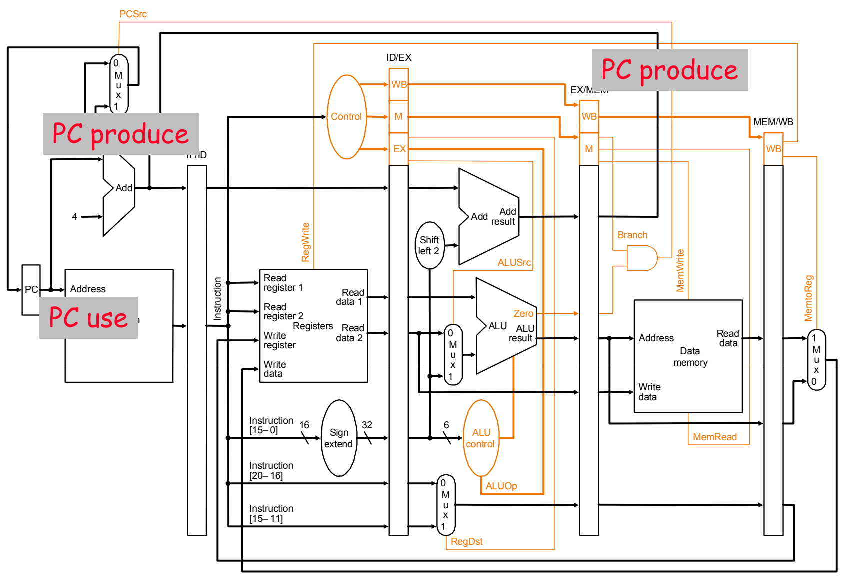
Performance Impact

- correct guess ⇒ no penalty ~86% of the time
- incorrect guess ⇒ 2 bubbles
- Assume
  - no data hazards
  - 20% control flow instructions
  - 70% of control flow instructions are taken
  - IPC = 1 / [1 + (0.20*0.7) * 2] =
    = 1 / [1 + 0.14 * 2] = 1 / 1.28 = 0.78

probability of a wrong guess   penalty for a wrong guess

Can we reduce either of the two penalty terms?

Reducing Mispredict Penalty



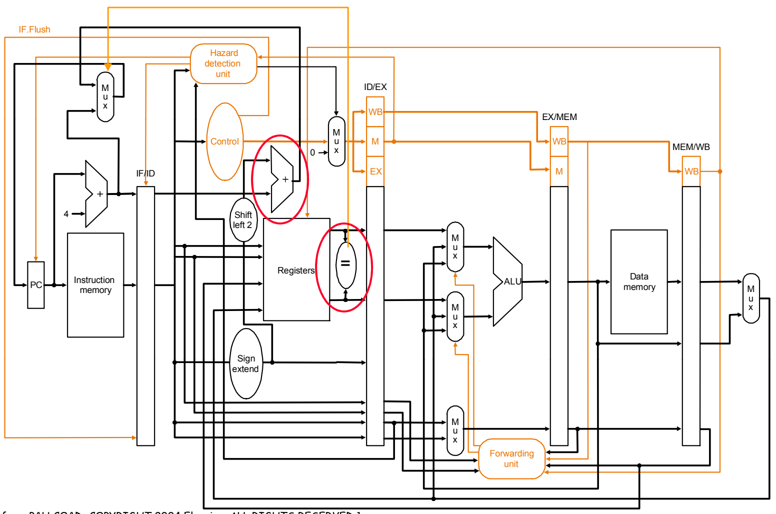
[Based on figures from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

MIPS R2000 Control Flow Design

- Simple address calculation based on instruction only
  - Branch PC-offset: 16-bit full-addition + 14-bit half-addition
  - Jump PC-offset: concatenation only
- Simple branch condition based on RF
  - One register relative (>, <, =) to 0
  - Equality between 2 registers

  No addition/subtraction necessary!
- An explicit ISA design choice to enable branch resolution in ID of a 5-stage pipeline

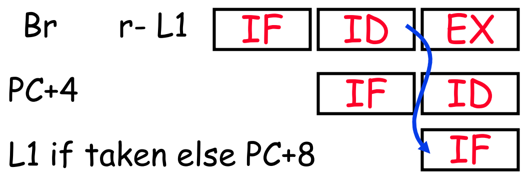
Branch Resolved in ID



[Based on figures from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

IPC = 1 / [1 + (0.2*$\underline{0.7}$) * 1] = 0.88

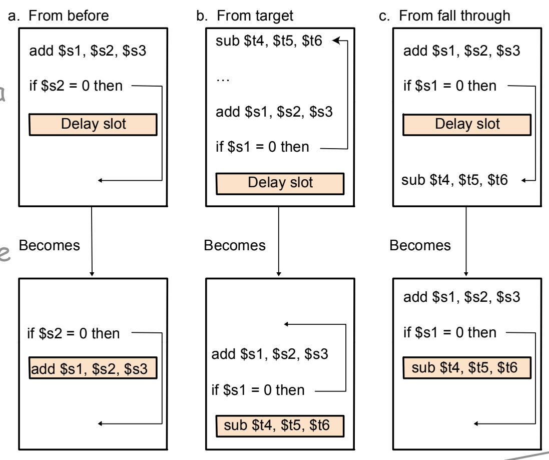
Branch Delay Slots



- PC+4 is already in the pipeline
  - throwing PC+4 away cost 1 bubble
  - letting PC+4 finish to the end won't hurt performance
- R2000 defined branches to have an architectural latency of 1 instruction
  - the instruction immediately after a branch is always executed
  - branch target takes effect on the 2nd instruction
  - if delay slot can always do useful work, effective IPC=1 without BTB or even a pipeline flush logic
  - ~80% of delay slots can be filled automatically by compilers

Filling Delay Slots by Static Reordering Transformation

reordering data independent (RAW, WAW, WAR) instructions does not change program



within same basic block

a new instruction added to not-taken path??

a new instruction added to taken??

Safe?

Final Data Hazard Analysis

	R/I-Type	LW	SW	Br	J	Jr
IF						
ID				use		use
EX	use produce	use	use	$\uparrow$		
MEM		produce	(use)			
WB						

- With forwarding, hazard distance is 0 except for RAW dependence on LW where it is 1
- Load delay slot semantics ensures a dependent instruction to be at least distance 2

Final PC Hazard Analysis

	R/I-Type	LW	SW	Br	J	Jr
IF	use (produce)	use (produce)	use (produce)	use	use	use
ID				produce	produce	produce
EX						
MEM						
WB						

- Hazard distance on a taken branch is 1
- Again, branch delay slot semantics ensures a dependent instruction to be at least distance 2

  MIPS R2000 can be interlock-free!!
- Hazard distance is greater in your project, why?

Making a Better Guess

- For ALU instructions
  - can't do better than guessing nextPC=PC+4
  - still tricky since must guess nextPC before the current instruction is fetched
- For Branch/Jump instructions
  - why not always guess in the taken direction since 70% correct
  - again, must guess nextPC before the branch instruction is fetched (but branch target is encoded in the instruction)

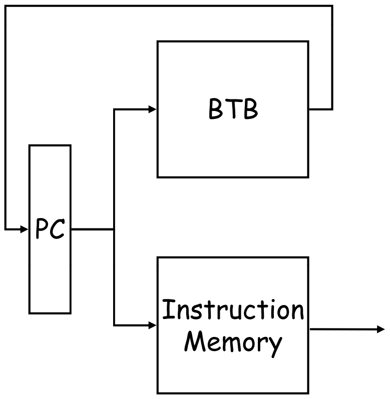
⇒ Must make a guess based only on the current fetch PC !!!

⇒ Fortunately,
- PC-offset branch/jump target is static
- We are allowed to be wrong some of the time

The Locality Principle

- One's recent past is a very good predictor of his/her near future.
- Temporal Locality: If you just did something, it is very likely that you will do the same thing again soon
  - since you are here today, there is a good chance you will be here again and again regularly
  - inverse is also true
- Spatial Locality: If you just did something, it is very likely you will do something similar/related
  - every time I find you in this room, you are probably sitting in the same seat
  - you are probably sitting near the same people

Branch Target Buffer (Oracle)



- BTB (Oracle)
  - a giant table indexed by PC
  - returns the guess for nextPC
- When encountering a PC for the first time, store in BTB

- PC + 4	if ALU/LD/ST
- PC+offset	if Branch or Jump
- ??	if JR

- Effectively guessing branches are always taken

  IPC = 1 / [1 + (0.20*<u>0.3</u>) * 2]

  = 0.89