18-447 Lecture 8: Data Hazard and Resolution

James C. Hoe
Department of ECE
Carnegie Mellon University
Housekeeping

• Your goal today
  – detect and resolve data hazards in in-order pipelines

• Notices
  – Lab 2, status check next week, due wk of 2/26
  – HW 2, due 2/21
  – **Office Hours: M 11~12 and F 1:30~2:30**

• Readings
  – P&H Ch 4
Instruction Pipeline Reality

• Not identical tasks
  – coalescing instruction types into one “multi-function” pipe
  – external fragmentation (some idle stages)

• Not uniform suboperations
  – group or sub-divide steps into stages to minimize variance
  – internal fragmentation (some too-fast stages)

• Not independent tasks
  – dependency detection and resolution
  – next lecture(s)

Even more messy if not RISC
Data Dependence

Data dependence

\[ r_3 \leftarrow r_1 \text{ op } r_2 \]
\[ \ldots \]
\[ r_5 \leftarrow r_3 \text{ op } r_4 \]

Read-after-Write (RAW)

Anti-dependence

\[ r_3 \leftarrow r_1 \text{ op } r_2 \]
\[ \ldots \]
\[ r_1 \leftarrow r_4 \text{ op } r_5 \]

Write-after-Read (WAR)

Output-dependence

\[ r_3 \leftarrow r_1 \text{ op } r_2 \]
\[ \ldots \]
\[ r_3 \leftarrow r_6 \text{ op } r_7 \]

Write-after-Write (WAW)

Don’t forget memory instructions
RAW Dependency and Hazard

<table>
<thead>
<tr>
<th></th>
<th>t0</th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addi</td>
<td>ra r- -</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>Addi</td>
<td>r- ra -</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>Addi</td>
<td>r- ra -</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td></td>
</tr>
<tr>
<td>Addi</td>
<td>r- ra -</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Addi</td>
<td>r- ra -</td>
<td>IF</td>
<td>ID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Addi</td>
<td>r- ra -</td>
<td>IF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Register Data Hazard Analysis

<table>
<thead>
<tr>
<th>R/I-Type</th>
<th>LW</th>
<th>SW</th>
<th>Bxx</th>
<th>Jal</th>
<th>Jalr</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>read RF</td>
<td>read RF</td>
<td>read RF</td>
<td>read RF</td>
<td>read RF</td>
</tr>
<tr>
<td>EX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WB</td>
<td>write RF</td>
<td>write RF</td>
<td></td>
<td>write RF</td>
<td>write RF</td>
</tr>
</tbody>
</table>

- For a given pipeline, when is there a register data hazard between 2 dependent instructions?
  - dependence type: RAW, WAR, WAW?
  - instruction types involved?
  - distance between the two instructions?
Hazard in In-order Pipeline

RAW Hazard

\[ \text{dist}_{\text{dependence}}(i, j) \leq \text{dist}_{\text{hazard}}(X, Y) \Rightarrow \text{Hazard!!} \]

\[ \text{dist}_{\text{dependence}}(i, j) > \text{dist}_{\text{hazard}}(X, Y) \Rightarrow \text{Safe} \]
RAW Hazard Analysis Example

<table>
<thead>
<tr>
<th></th>
<th>R/I-Type</th>
<th>LW</th>
<th>SW</th>
<th>Bxx</th>
<th>Jal</th>
<th>Jalr</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>read RF</td>
<td>read RF</td>
<td>read RF</td>
<td>read RF</td>
<td>read RF</td>
<td></td>
</tr>
<tr>
<td>EX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WB</td>
<td>write RF</td>
<td>write RF</td>
<td></td>
<td>write RF</td>
<td>write RF</td>
<td></td>
</tr>
</tbody>
</table>

- Older $I_A$ and younger $I_B$ have RAW hazard iff
  - $I_B$ (R/I, LW, SW, Bxx or JALR) reads a register written by $I_A$ (R/I, LW, or JAL/R)
  - $\text{dist}(I_A, I_B) \leq \text{dist(ID, WB)} = 3$

What about WAW and WAR hazard?
What about memory data hazard?
Pipeline Stall:
universal hazard resolution

Stall==make younger instruction wait until hazard passes
1. stop all up-stream stages
2. drain all down-stream stages
What should happen in this case?

Inst\textsubscript{h}

\begin{tabular}{|c|c|c|c|c|c|}
\hline
&t\_0&t\_1&t\_2&t\_3&t\_4&t\_5 \\
\hline
\text{IF} & \text{ID} & \text{ALU} & \text{MEM} & \text{WB} \\
\hline
\text{Inst} \_i & i & \text{IF} & \text{ID} & \text{ALU} & \text{MEM} & \text{WB} \\
\hline
\text{Inst} \_j & j & \text{IF} & \text{ID} & \text{ALU} & \text{MEM} & \text{WB} \\
\hline
\text{Inst} \_k & k & \text{IF} & \text{ID} & \text{ALU} & \text{MEM} & \text{WB} \\
\hline
\text{Inst} \_l & & & & & & \\
\hline
\end{tabular}

i: \text{ } r_x \leftarrow _

j: \text{ } r_y \leftarrow r_z

k: _ \leftarrow r_x \quad \text{dist}(i,k)=2

18-447-S18-L08-S10, James C. Hoe, CMU/ECE/CALCM, ©2018
# Pipeline Stall

<table>
<thead>
<tr>
<th>IF</th>
<th>t₀</th>
<th>t₁</th>
<th>t₂</th>
<th>t₃</th>
<th>t₄</th>
<th>t₅</th>
<th>t₆</th>
<th>t₇</th>
<th>t₈</th>
<th>t₉</th>
<th>t₁₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>j</td>
<td>k</td>
<td>k</td>
<td>k</td>
<td>k</td>
<td>k</td>
<td>l</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>t₀</th>
<th>t₁</th>
<th>t₂</th>
<th>t₃</th>
<th>t₄</th>
<th>t₅</th>
<th>t₆</th>
<th>t₇</th>
<th>t₈</th>
<th>t₉</th>
<th>t₁₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>i</td>
<td>j</td>
<td>j</td>
<td>j</td>
<td>j</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EX</th>
<th>t₀</th>
<th>t₁</th>
<th>t₂</th>
<th>t₃</th>
<th>t₄</th>
<th>t₅</th>
<th>t₆</th>
<th>t₇</th>
<th>t₈</th>
<th>t₉</th>
<th>t₁₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>i</td>
<td>bub</td>
<td>bub</td>
<td>bub</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MEM</th>
<th>t₀</th>
<th>t₁</th>
<th>t₂</th>
<th>t₃</th>
<th>t₄</th>
<th>t₅</th>
<th>t₆</th>
<th>t₇</th>
<th>t₈</th>
<th>t₉</th>
<th>t₁₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>i</td>
<td>bub</td>
<td>bub</td>
<td>bub</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WB</th>
<th>t₀</th>
<th>t₁</th>
<th>t₂</th>
<th>t₃</th>
<th>t₄</th>
<th>t₅</th>
<th>t₆</th>
<th>t₇</th>
<th>t₈</th>
<th>t₉</th>
<th>t₁₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>i</td>
<td>bub</td>
<td>bub</td>
<td>bub</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**i:** rx $\leftarrow _-

**j:** _ $\leftarrow$ rx
• Stall
  – disable **PC** and **IR** latching
  – set \( \text{RegWrite}_{\text{ID}} = 0 \) and \( \text{MemWrite}_{\text{ID}} = 0 \)
Stall Condition

• Older $I_A$ and younger $I_B$ have RAW hazard iff
  – $I_B$ (R/I, LW, SW, Bxx or JALR) reads a register written by $I_A$ (R/I, LW, or JAL/R)
  – $\text{dist}(I_A, I_B) \leq \text{dist}(ID, WB) = 3$

• More plainly, before $I_B$ in ID reads a register, $I_B$ needs to check if any $I_A$ in EX, MEM or WB is going to update it (if so, value in RF is “stale”)

Watch out for $x0$!!
Stall Condition

• Helper functions
  – $use_{rs1}(l)$ returns true if $l$ uses $rs1$ && $rs1!\neq x0$

• Stall IF and ID when
  – $(rs1_{ID} == rd_{EX})$ && $use_{rs1}(IR_{ID})$ && $RegWrite_{EX}$ or
  – $(rs1_{ID} == rd_{MEM})$ && $use_{rs1}(IR_{ID})$ && $RegWrite_{MEM}$ or
  – $(rs1_{ID} == rd_{WB})$ && $use_{rs1}(IR_{ID})$ && $RegWrite_{WB}$ or
  – $(rs2_{ID} == rd_{EX})$ && $use_{rs2}(IR_{ID})$ && $RegWrite_{EX}$ or
  – $(rs2_{ID} == rd_{MEM})$ && $use_{rs2}(IR_{ID})$ && $RegWrite_{MEM}$ or
  – $(rs2_{ID} == rd_{WB})$ && $use_{rs2}(IR_{ID})$ && $RegWrite_{WB}$

It is crucial that EX, MEM and WB continue to advance during stall
Impact of Stall on Performance

- Each stall cycle corresponds to 1 lost ALU cycle
- A program with $N$ instructions and $S$ stall cycles:
  \[
  \text{average IPC} = \frac{N}{N+S}
  \]
- $S$ depends on
  - frequency of hazard-causing dependencies
  - distance between hazard-causing instruction pairs
  - distance between hazard-causing dependencies

  (suppose $i_1, i_2$ and $i_3$ all depend on $i_0$, once $i_1$’s hazard is resolved by stalling, $i_2$ and $i_3$ do not stall)
Sample Assembly [P&H]

for (j=i-1; j>=0 && v[j] > v[j+1]; j--) { ...... }

addi $s1, $s0, -1  
for2tst: slti $t0, $s1, 0  
        bne $t0, $zero, exit2  
sll $t1, $s1, 2  
add $t2, $a0, $t1  
lw $t3, 0($t2)  
lw $t4, 4($t2)  
slt $t0, $t4, $t3  
beq $t0, $zero, exit2  
.........
addi $s1, $s1, -1  
exit2: j for2tst

3 stalls
3 stalls
3 stalls
3 stalls
3 stalls
3 stalls
3 stalls
3 stalls
Data Forwarding (or Register Bypassing)

• What does “ADD rx ry rz” mean? Get inputs from RF[ry] and RF[rz] and put result in RF[rx]?

• But, RF is just a part of an abstraction – a way to connect dataflow between instructions
  – “inputs to ADD are resulting values of the last instructions to assign to RF[ry] and RF[rz]”
  – RF doesn’t have to exist as an literal object

• If only dataflow matters, don’t wait for WB . . .
Resolving RAW Hazard by Forwarding

• Older $I_A$ and younger $I_B$ have RAW hazard iff
  - $I_B$ ($R/I$, $LW$, $SW$, $Bxx$ or $JALR$) reads a register written by $I_A$ ($R/I$, $LW$, or $JAL/R$)
  - $\text{dist}(I_A, I_B) \leq \text{dist}(ID, WB) = 3$

• More plainly, before $I_B$ in ID reads a register, $I_B$ needs to check if any $I_A$ in EX, MEM or WB is going to update it (if so, value in RF is “stale”)

• Before: $I_B$ need to stall for RF to update

• Now: $I_B$ need to stall for $I_A$ to produce result
  - retrieve $I_A$ result from datapath when ready
  - must retrieve from youngest if multiple hazards
Forwarding Paths (v1)

For the given diagram, the text description includes:

- Registers
- ID/EX
- EX/MEM
- MEM/WB

Forwarding Paths with labels:

- dist(i,j)=1
- dist(i,j)=2
- dist(i,j)=3

Forwarding unit and paths for Rd, Rs, Rt, and internal forwarding.

[Based on original figure from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
Forwarding Paths (v2)

better if EX is the fastest stage
Forwarding Logic (for v1)

if \( \text{rs1}_{\text{ID}} \neq 0 \) \&\& \( \text{rs1}_{\text{ID}} = \text{rd}_{\text{EX}} \) \&\& \( \text{RegWrite}_{\text{EX}} \) then
   forward writeback value from EX \hspace{1cm} // dist=1
else if \( \text{rs1}_{\text{ID}} \neq 0 \) \&\& \( \text{rs1}_{\text{ID}} = \text{rd}_{\text{MEM}} \) \&\& \( \text{RegWrite}_{\text{MEM}} \) then
   forward writeback value from MEM \hspace{1cm} // dist=2
else if \( \text{rs1}_{\text{ID}} \neq 0 \) \&\& \( \text{rs1}_{\text{ID}} = \text{rd}_{\text{WB}} \) \&\& \( \text{RegWrite}_{\text{WB}} \) then
   forward writeback value from WB \hspace{1cm} // dist=3
else
   use \( A_{\text{ID}} \) \hspace{1cm} // dist > 3

Must check in right order
Why doesn’t \textit{use}_rs1( ) appear?
Isn’t it bad to forward from LW in EX?
### Data Hazard Analysis (with Forwarding)

<table>
<thead>
<tr>
<th>R/I-Type</th>
<th>LW</th>
<th>SW</th>
<th>Bxx</th>
<th>Jal</th>
<th>Jalr</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EX</td>
<td>use</td>
<td>use</td>
<td>use</td>
<td>use</td>
<td>produce</td>
</tr>
<tr>
<td>MEM</td>
<td>produce</td>
<td>(use)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Even with forwarding, RAW dependence on immediate preceding LW results in hazard
- **Stall** = \( \{ ((rs_{1\text{ID}} == rd_{EX}) \&\& \text{use}_{-rs1}(IR_{ID})) \| ((rs_{2\text{ID}} == rd_{EX}) \&\& \text{use}_{-rs2}(IR_{ID})) \} \&\& \text{MemRead}_{EX} \)  
  i.e., \( op_{EX} = Lx \)
MIPS Load “Delay Slot” Feature

R2000 defined LW with arch. latency of 1 inst
  - invalid for I2 (in LW’s delay slot) to ask for LW’s result
  - any dependence on LW at least distance 2

Delay slot vs dynamic stalling
  - fill with an independent instruction (no difference)
  - if not, fill with a NOP (no difference)

Can’t lose on 5-stage . . . good idea?

Hint: 1. non-atomic instruction; 2. μarch influence
Sample Assembly [P&H]

for (j=i-1; j>=0 && v[j] > v[j+1]; j-=1) { ...... }

addi $s1, $s0, -1

for2tst: slti $t0, $s1, 0
          bne $t0, $zero, exit2
          sll $t1, $s1, 2
          add $t2, $a0, $t1
          lw $t3, 0($t2)
          lw $t4, 4($t2)
          slt $t0, $t4, $t3
          beq $t0, $zero, exit2
          ........

addi $s1, $s1, -1

exit2: j for2tst

1 stall or 1 nop (MIPS)
Terminology

• Dependency
  – ordering requirement between instructions

• Pipeline Hazard:
  – (potential) violation of dependencies

• Hazard Resolution:
  – static ⇒ schedule instructions at compile time to avoid hazards
  – dynamic ⇒ detect hazard and adjust pipeline operation

• Pipeline Interlock (i.e., stall)
Dividing into Stages

Is this the correct partitioning? Why not 4 or 6 stages? Why not different boundaries

Based on original figure from [P&H CO&D, COPYRIGHT 2004, ALL RIGHTS RESERVED.]
Why not very deep pipelines?

- With only 5 stages, still plenty of combinational logic between registers
- “Superpipelining” ⇒ increase pipelining such that even intrinsic operations (e.g. ALU, RF access, memory access) require multiple stages
- What’s the problem?

Inst0: $r1 \leftarrow r2 + r3$

Inst1: $r4 \leftarrow r1 + 2$
Intel P4’s Superpipelined Adder Hack

32-bit addition pipelined over 2 stages, $BW=1/\text{latency}_{16\text{-bit-add}}$
No stall between back-to-back dependencies
When you can’t split a stage . . .

I @ (rate=2/T) → d → e → A (rate=1/T) → O @ (rate=2/T)

B (rate=1/T) → 0.5T clock

T delay
Dependencies and Pipelining  
(architecture vs. microarchitecture)

Sequential and atomic instruction semantics

True dependence between two instructions may only require ordering of certain sub-operations

- \( i_1 \)
- \( i_2 \)
- \( i_3 \)

Defines what is correct; doesn’t say do it this way