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 instruction pipelines

• Notices
  – Lab 2, status check next week, due week of 2/24
  – HW 2, due 2/19 before class

• 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

1. \text{addi ra r- -}
2. \text{addi r- ra -}
3. \text{addi r- ra -}
4. \text{addi r- ra -}
5. \text{addi r- ra -}

\begin{tabular}{cccccc}
\text{t}_0 & \text{t}_1 & \text{t}_2 & \text{t}_3 & \text{t}_4 & \text{t}_5 \\
\text{IF} & \text{ID} & \text{EX} & \text{MEM} & \text{WB} & \\
\text{IF} & \text{ID} & \text{EX} & \text{MEM} & \text{WB} & \\
\text{IF} & \text{ID} & \text{EX} & \text{MEM} & \\
\text{IF} & \text{ID} & \text{EX} & \\
\text{IF} & \\
\end{tabular}
### 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

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

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

Pipeline Stall resolution:

1. Stop all up-stream stages
2. Drain all down-stream stages

Stall==make younger instruction wait until hazard passes:

1. stop all up-stream stages
2. drain all down-stream stages

Inst_h

Inst_i

Inst_j

Inst_k

Inst_l

i: \( r_x \leftarrow \_ \)
bubble
bubble
bubble

j: \( \_ \leftarrow r_x \)

\( \text{dist}(i,j) = 4 \)
# Pipeline Stall

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
<th>$t_4$</th>
<th>$t_5$</th>
<th>$t_6$</th>
<th>$t_7$</th>
<th>$t_8$</th>
<th>$t_9$</th>
<th>$t_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<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>
</tr>
<tr>
<td>ID</td>
<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>
</tr>
<tr>
<td>EX</td>
<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>
</tr>
<tr>
<td>MEM</td>
<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>
</tr>
<tr>
<td>WB</td>
<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>
</tr>
</tbody>
</table>

\[ i: \text{rx} \leftarrow \_ \]

\[ j: \_ \leftarrow \text{rx} \]
Pop Quiz: What happens in this case?

Inst_h

Inst_i

Inst_j

Inst_k

Inst_l

i: \( r_x \leftarrow \_ \)  
j: \( r_y \leftarrow r_z \)  
k: \( \_ \leftarrow r_x \)  \( \text{dist}(i,k)=2 \)
• 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)
  - $\text{dist}(I_A, I_B) \leq \text{dist}(\text{ID}, \text{WB}) = 3$

- 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**
  - $use_{\text{rs1}}(I)$ returns true if $I$ uses $rs1$ \&\& $rs1!=x0$

- **Stall IF and ID when**
  - $(rs1_{ID}==rd_{EX}) \&\& use_{\text{rs1}}(IR_{ID}) \&\& RegWrite_{EX}$ \hspace{1cm} or
  - $(rs1_{ID}==rd_{MEM}) \&\& use_{\text{rs1}}(IR_{ID}) \&\& RegWrite_{MEM}$ \hspace{1cm} or
  - $(rs1_{ID}==rd_{WB}) \&\& use_{\text{rs1}}(IR_{ID}) \&\& RegWrite_{WB}$ \hspace{1cm} or
  - $(rs2_{ID}==rd_{EX}) \&\& use_{\text{rs2}}(IR_{ID}) \&\& RegWrite_{EX}$ \hspace{1cm} or
  - $(rs2_{ID}==rd_{MEM}) \&\& use_{\text{rs2}}(IR_{ID}) \&\& RegWrite_{MEM}$ \hspace{1cm} or
  - $(rs2_{ID}==rd_{WB}) \&\& use_{\text{rs2}}(IR_{ID}) \&\& RegWrite_{WB}$

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-=1) { ...... }

for2tst:
  addi $s1, $s0, -1
  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
j for2tst

exit2:
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 a literal object
- If only dataflow matters, don’t wait for WB . . .

```
add    ra r- r-
addi   r- ra r-
```

```
IF  ID  EX  MEM  WB
  IF  ID  EX  MEM  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)

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 \((rs1_{\text{ID}} \neq 0) \land (rs1_{\text{ID}} = rd_{\text{EX}}) \land \text{RegWrite}_{\text{EX}}\) then
forward writeback value from EX // dist=1
else if \((rs1_{\text{ID}} \neq 0) \land (rs1_{\text{ID}} = rd_{\text{MEM}}) \land \text{RegWrite}_{\text{MEM}}\) then
forward writeback value from MEM // dist=2
else if \((rs1_{\text{ID}} \neq 0) \land (rs1_{\text{ID}} = rd_{\text{WB}}) \land \text{RegWrite}_{\text{WB}}\) then
forward writeback value from WB // dist=3
else
use \(A_{\text{ID}}\) // dist > 3

Must check in right order
Why doesn’t \textit{use}\_\textit{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 produce</td>
<td>use</td>
<td>use</td>
<td>use produce</td>
<td>use 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
- \[\text{Stall} = \{ ((\text{rs1}_{ID} \equiv \text{rd}_{EX}) \land \text{use}_{rs1}(\text{IR}_{ID})) \lor ((\text{rs2}_{ID} \equiv \text{rd}_{EX}) \land \text{use}_{rs2}(\text{IR}_{ID})) \} \land \text{MemRead}_{EX} \]
  \[\text{i.e., } \text{op}_{EX} = \text{Lx}\]
Historical: MIPS Load “Delay Slot”

- 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]

```assembly
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
    j for2tst

exit2:
```

1 stall or 1 nop (MIPS)
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?
  
  \[
  \text{Inst}_0: r1 \leftarrow r2 + r3 \\
  \text{Inst}_1: r4 \leftarrow r1 + 2
  \]
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)
Dependencies and Pipelining
(architecture vs. microarchitecture)

Sequential and atomic
instruction semantics

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

Defines what is correct; doesn’t say do it this way
Lab 2 with 6 Stages

200ps
IF: Instruction fetch

100ps
ID: Instruction decode/ register file read

200ps
EX: Execute/ address calculation

200ps
MEM: Memory access

100ps
WB: Write back

RF
write

ignore
for today

200ps
100ps
200ps
200ps
100ps

16
Sign extend
32

6 Mux
1

6 Mux

Zero
ALU result

Address
Data memory

Write memory

Write data

Read data

Read register 2
Write register

Read data 2

Read register 1
Write register

Read data 1

Read data

Add

Add result

Shift left 2

Add

Instruction
memory

Instruction

Address

PC

Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
Lab 2: Data Hazard Analysis with Stalling

<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></td>
<td>read RF</td>
</tr>
<tr>
<td>EX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEM1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEM2</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></td>
<td>write RF</td>
<td>write RF</td>
</tr>
</tbody>
</table>
Lab 2: Data Hazard Analysis with Forwarding

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EX</td>
<td></td>
<td>use</td>
<td>use</td>
<td>use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEM1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(use)</td>
<td></td>
</tr>
<tr>
<td>MEM2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>produce</td>
</tr>
</tbody>
</table>

Note: "use" indicates a use hazard, "produce" indicates a produce hazard.