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
  – control dependence next time

• Notices
  – Lab 1, Part B, due Friday midnight
  – Handout #7: Lab 2 out on Friday
  – Midterm 1, Wednesday, March 10

• 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
\[ x_3 \leftarrow x_1 \text{ op } x_2 \quad \text{Read-after-Write (RAW)} \]
\[ \ldots \]
\[ x_5 \leftarrow x_3 \text{ op } x_4 \]

Anti-dependence
\[ x_3 \leftarrow x_1 \text{ op } x_2 \quad \text{Write-after-Read (WAR)} \]
\[ \ldots \]
\[ x_1 \leftarrow x_4 \text{ op } x_5 \]

Output-dependence
\[ x_3 \leftarrow x_1 \text{ op } x_2 \quad \text{Write-after-Write (WAW)} \]
\[ \ldots \]
\[ x_3 \leftarrow x_6 \text{ op } x_7 \]

Don’t forget memory instructions
Dependency and Hazard: e.g. RAW

- addi x1, x0, 0
- addi x2, x1, 0
- addi x3, x1, 0
- addi x4, x1, 0
- addi x5, x1, 0
- addi x6, x1, 0

```
t_0   t_1   t_2   t_3   t_4   t_5
IF    ID    EX    MEM   WB
IF    ID    EX    MEM   WB
IF    ID    EX    MEM   WB
IF    ID    EX    MEM   WB
IF    ID    EX    MEM   WB
IF    ID    EX    MEM   WB
IF    ID    EX    MEM   WB
```
## 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>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>

- 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
## Pipeline Stall

<table>
<thead>
<tr>
<th></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>
<th>$t_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IF</strong></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><strong>ID</strong></td>
<td>h</td>
<td>i</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>
<tr>
<td><strong>EX</strong></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><strong>MEM</strong></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><strong>WB</strong></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>

$i$: $x_1 \leftarrow _$

$j$: _$ $\leftarrow x_1$
Pop Quiz: What happens in this case?

Inst_h

Inst_i

Inst_j

Inst_k

Inst_l

i: x1 ← _

j: x3 ← x2

k: _ ← x1    dist(i,k)=2
Stall

- disable PC and IR latching
- set $\text{RegWrite}_{\text{ID}}=0$ and $\text{MemWrite}_{\text{ID}}=0$
When to Stall

- 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)
  - \( \text{dist}(I_A, I_B) \leq \text{dist}(I_D, WB) = 3 \)

  Above is about existence of hazard

- Operationally, to detect hazard in time to prevent:
  - 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
  - $need_{rs1}(I)$ returns true if $I$ uses $rs1$ && $rs1! = x0$

- Stall IF and ID when
  - $(rs1_{ID} == rd_{EX})$ && $need_{rs1}(IR_{ID})$ && RegWrite$_{EX}$ or
  - $(rs1_{ID} == rd_{MEM})$ && $need_{rs1}(IR_{ID})$ && RegWrite$_{MEM}$ or
  - $(rs1_{ID} == rd_{WB})$ && $need_{rs1}(IR_{ID})$ && RegWrite$_{WB}$ or
  - $(rs2_{ID} == rd_{EX})$ && $need_{rs2}(IR_{ID})$ && RegWrite$_{EX}$ or
  - $(rs2_{ID} == rd_{MEM})$ && $need_{rs2}(IR_{ID})$ && RegWrite$_{MEM}$ or
  - $(rs2_{ID} == rd_{WB})$ && $need_{rs2}(IR_{ID})$ && 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) {
    .......
    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:
3 stalls
3 stalls
3 stalls
3 stalls
3 stalls
3 stalls
3 stalls
3 stalls
3 stalls
3 stalls
3 stalls
3 stalls
3 stalls
Data Forwarding (or Register Bypassing)

• What does “ADD \( r_x r_y r_z \)” mean? Get inputs from RF[\( r_y \)] and RF[\( r_z \)] and put result in RF[\( r_x \)]?

• But, RF is just a part of an abstraction
  – a way to connect dataflow between instructions
    “operands to ADD are resulting values of the last instructions to assign to RF[\( r_y \)] and RF[\( r_z \)]”
    – RF doesn’t have to exist/behave as a literal object!!!

• If only dataflow matters, don’t wait for WB . . .

```
addi x1, x0, 0
addi x2, x1, 0
```

18-447-S21-L08-S17, James C. Hoe, CMU/ECE/CALCM, ©2021
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)
  - $\text{dist}(I_A, I_B) \leq \text{dist}(ID, WB) = 3$

• To detect hazard in time to prevent, 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

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

- **dist(i,j)=3**
  - Internal forward?
  - dist(i,j)=3

- **dist(i,j)=2**
  - ID/EX/RegisterRD
  - EX/MEM/RegisterRD

- **dist(i,j)=1**
  - ForwardA
  - ForwardB
  - ALU

- **dist(i,j)=3**
  - Registers

- 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

Based on original figure from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.
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} // \hspace{0.5cm} \text{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} // \hspace{0.5cm} \text{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} // \hspace{0.5cm} \text{dist}=3
else
use \( A_{\text{ID}} \)  \hspace{1cm} // \hspace{0.5cm} \text{dist} > 3

Must prioritize young-to-old
Why doesn’t \textit{need\_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>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.

- \(\text{Stall} = \{ ((rs_{1ID} == rd_{EX}) && \text{need}_rs1(IR_{ID})) || ((rs_{2ID} == rd_{EX}) && \text{need}_rs2(IR_{ID})) \} && \text{MemRead}_{EX} \)
  
i.e., \(\text{op}_{EX}=\text{Lx}\)
Historical: MIPS Load “Delay Slot”

- **R2000 defined LW with arch. latency of 1 inst**
  - invalid for \( l_2 \) (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. \( \mu \)arch influence
Sample Assembly [P&H]

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

```assembly
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)
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?  
  
  $\text{Inst}_0$: addi $x_1$, $x_0$, 0
  $\text{Inst}_1$: addi $x_2$, $x_1$, 0
Terminology

- **Dependency**
  - property of program
  - ordering requirement between instructions

- **Pipeline Hazard:**
  - property of uarch when interacting with program
  - (potential) violation of dependencies in program

- **Hazard Resolution:**
  - static $\Rightarrow$ schedule instructions at compile time to avoid hazards
  - dynamic $\Rightarrow$ detect hazard and adjust pipeline operation

Stall, Flush or Forward
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
**2021 Lab 2 with Synchronous Memory**

The diagram illustrates a synchronous memory-based system with the following stages:

- **IF**: Instruction fetch (200ps)
- **ID**: Instruction decode/register file read (100ps)
- **EX**: Execute/address calculation (200ps)
- **MEM**: Memory access (200ps)
- **WB**: Write back (100ps)

The diagram shows the flow of data and instructions through these stages, including operations such as reading from instruction memory, writing to register files, and accessing data from memory. The timing is indicated by 200ps and 100ps intervals, reflecting the duration of each stage.

The diagram also includes a section marked as "ignore for today," indicating a part of the system that might not be covered in the current lab session.

**Note:** The diagram is based on the original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]