EEC 483 Computer Organization

Chapter 4.7 Data Hazards, Forwarding and Stalling

Chansu Yu

Cleveland State University

5-Stage Pipeline

Instructions and data move generally from left to right through the five stages as they complete execution “except two cases”.

- WB stage
- PC selection at MEM stage
Pipeline Complexities

Assume $2=2222$, $3=3333$, …

- add $5, 2, 3$
- sub $6, 5, 2$
- beq $7, label$
- add $1, 2, 3$
- add $4, 5, 6$
- label: sub $7, 8, 9$

Hazard = when an instruction’s stage is unable to execute during the current cycle. Forced to stall (delay) the pipeline:

Instruction #2 stage 3 unable to continue.
Hazard Types

- **Structural hazards**: Necessary functional unit is busy
  - suppose we had only one memory
  - we already remove this type of hazards

- **Control hazards**: Next instruction address unknown
  - need to worry about branch instructions

- **Data hazards**: Source data not yet available
  - an instruction depends on a previous instruction

---

Structural Hazards

A needed functional unit is busy executing a previous instruction.

Example:
- Our sample MIPS pipeline has none.
- What if PC+4 computation used main ALU instead of separate adder?

![Structural Hazards Diagram](image-url)
Structural Hazards: Avoiding

- Add or replicate functional units.
  - Usually easy in modern processors (lots of hardware real estate).
  - Our sample pipeline avoids all structural hazards.

- Load/store architecture
  - Avoids structural hazards by limiting memory access only to load and store instructions
  - “sub $1, 200($10)”
    - One more stage (EX) between MEM/WB with additional ALU is not a big problem
    - But all other instructions should be wasting a clock since they need to execute the same number of stages

*Graphically Representing Pipelines

- Can help with answering questions like:
  - how many cycles does it take to execute this code?
  - what is the ALU doing during cycle 4?
  - use this representation to help understand datapaths
Data Hazards

- Needed data still being computed by previous instruction

\[
\text{sub $2$, $1$, $3$} \\
\text{and $12$, $2$, $5$} \\
\text{or $13$, $6$, $2$} \\
\text{add $14$, $2$, $2$} \\
\text{sw $15$, 100($2$)}
\]

Assume $I=10$, $S=10$, $S_3=30$

Data Hazards: Dependencies

- Problem with starting next instruction before first is finished
- Dependencies that “go backward in time” are data hazards

<table>
<thead>
<tr>
<th>Time (in clock cycles)</th>
<th>Value of register $2$</th>
<th>CC 1</th>
<th>CC 2</th>
<th>CC 3</th>
<th>CC 4</th>
<th>CC 5</th>
<th>CC 6</th>
<th>CC 7</th>
<th>CC 8</th>
<th>CC 9</th>
</tr>
</thead>
</table>

Program execution (in instructions)

- sub $2$, $1$, $3$
- and $12$, $2$, $5$
- or $13$, $6$, $2$
- add $14$, $2$, $2$
- sw $15$, 100($2$)

“and” has a problem
“or” has a problem
“add” ???
“sw” is OK
Data Hazards: Software Solution

- Have compiler guarantee no hazards
- Where do we insert the “nops”?
  - sub $2, $1, $3
  - and $12, $2, $5
  - or $13, $6, $2
  - add $14, $2, $2
  - sw $15, 100($2)

- Or, detect and “stall” the pipeline
- Problem: this really slows us down!

Data Hazards: Forwarding

While result not written back until WB:
- sub $2, $1, $3
- and $12, $2, $5

It is calculated earlier – in EX:
- sub $2, $1, $3
- and $12, $2, $5

Actually available after EX stage (not WB)
Actually needed at EX stage (not ID)

Add forwarding hardware to allow, e.g., EX’s output (located in EX/MEM pipeline register) to be EX’s input.

c.yu91@csuohio.edu
Forwarding: All 2 Cases

- **Time (in clock cycles):**
  - CC 1: 10
  - CC 2: 10
  - CC 3: 10
  - CC 4: 10
  - CC 5: 10
  - CC 6: 10
  - CC 7: 10
  - CC 8: 10
  - CC 9: 10

- **Value of register $2:**
  - CC 1: 10
  - CC 2: 10
  - CC 3: 10
  - CC 4: 10
  - CC 5: 10
  - CC 6: 10
  - CC 7: 10
  - CC 8: 10
  - CC 9: 10

- **Value of MEM/WD:**
  - CC 1: X
  - CC 2: X
  - CC 3: X
  - CC 4: X
  - CC 5: X
  - CC 6: X
  - CC 7: X
  - CC 8: X

- Program execution order:
  - sub $11, $1, $3
  - and $12, $2, $5
  - or $13, $6, $2
  - add $14, $2, $2
  - sub $15, 1 $2

- “and” has a problem
  - => fixed
- “or” has a problem
  - => fixed
- “add” ??? -> OK
- “sw” is OK

Forwarding: Implementation

- **Read registers rs($1) and rt($3), but just pass rd($2)**
  - sub $2, $1, $3
    - *Calculate ALU output: $1+$3*
    - *Write to register rd($3) with ALU output*

- and $12, $2, $5
  - *Read registers rs($2) and rt($5)*
  - *Calculate ALU output: $2+$5*
  - *Write to register rd($12) with ALU output*

*If they match, we need to feed the result directly.*
- *How to detect?*
  - => control
- *How to feed?*
  - => datapath
Forwarding: Implementation

Data Hazards (again)

- Needed data still being computed by previous instruction

\[
\begin{align*}
\text{sub} & \quad [\text{R1}], \quad \$3, \quad \$2 \\
\text{and} & \quad [\text{R1}], \quad \$11, \quad \$4 \\
\text{or} & \quad [\text{R1}], \quad \$6, \quad \$11 \\
\text{add} & \quad \$14, \quad \$8, \quad \$9 \\
\text{sw} & \quad \$15, \quad 100(\$2)
\end{align*}
\]
sub $11, $3, $2

and $12, $11, $4
sub $11, $3, $2
Forwarding: Implementation

Additional datapath for forwarding?

How to control the forwarding datapath?
Forwarding : Forwarding Unit

Forwarding unit: 6-input, 2-output combinational circuit

Forwarding : Other Case

- Use temporary results, don’t wait for them to be written
  - Register file forwarding to handle read/write to same register
  - ALU forwarding

- In the previous example,
  - EX/MEM.Rd = ID/EX.Rs
  - How about “EX/MEM.Rd = ID/EX.Rt”?
  - They are the case: EX/MEM → EX: Forwarding to the next instruction

- Anything more?
  - In MIPS pipeline, one more forwarding case:
    - MEM/WB → EX: Forwarding to the instruction after the next.
      - sub $2, $1, $3
      - and $12, $2, $5
      - or $13, $6, $2
    - MEM/WB.Rd = ID/EX.Rs and MEM/WB.Rd = ID/EX.Rt

Cleveland State University

HW1, (5)
c.yu91@csuohio.edu
Forwarding Control

- **Control logic**
  - **ForwardA** =
    - 10 if (EX/MEM.Rd = ID/EX.Rs) <- get operand from EX/MEM
    - 01 if (MEM/WB.Rd = ID/EX.Rs) <- get operand from MEM/WB
    - 00, otherwise <- get operand from ID/EX

  - **ForwardB** =
    - 10 if (EX/MEM.Rd = ID/EX.Rt) <- get operand from EX/MEM
    - 01 if (MEM/WB.Rd = ID/EX.Rt) <- get operand from MEM/WB
    - 00, otherwise <- get operand from ID/EX
Forwarding Control

- RegWrite must be active & dest. reg. is not $0$
- Control logic
  - ForwardA =
    - 10 if ((EX/MEM.Rd = ID/EX.Rs) & & EX/MEM.RegWrite & & (EX/MEM.Rd ≠ 0))
    - 01 if ((MEM/WB.Rd = ID/EX.Rs) & & MEM/WB.RegWrite & & (MEM/WB.Rd ≠ 0))
    - 00, otherwise
  - ForwardB =
    - 10 if ((EX/MEM.Rd = ID/EX.Rt) & & EX/MEM.RegWrite & & (EX/MEM.Rd ≠ 0))
    - 01 if ((MEM/WB.Rd = ID/EX.Rt) & & MEM/WB.RegWrite & & (MEM/WB.Rd ≠ 0))
    - 00, otherwise

Forwarding Control

- Control logic
  - ForwardA =
    - 10 if ((EX/MEM.Rd = ID/EX.Rs) & & EX/MEM.RegWrite & & (EX/MEM.Rd ≠ 0))
    - 01 if ((MEM/WB.Rd = ID/EX.Rs) & & MEM/WB.RegWrite & & (MEM/WB.Rd ≠ 0))
    - 00, otherwise
  - If a list of codes is
    - Add $1$, $2$, $3$ : at WB -> MEM/WB.Rd=1
    - Add $1$, $1$, $4$ : at MEM -> EX/MEM.Rd=1
    - Add $1$, $1$, $5$ : at EX -> ID/EX.Rs=1
    - When the third inst. is at EX stage, the two conditions are all met. Then, ForwardA=10 or 01???
    - Should be forwarded from the most recent result, which is "10"
    - Therefore, ForwardA=01
  - if (((MEM/WB.Rd = ID/EX.Rs) & & MEM/WB.RegWrite & & (MEM/WB.Rd ≠ 0)) & & (EX/MEM.Rd ≠ ID/EX.Rs))
Forwarding Control

- RegWrite must be active & dest. reg. is not $0$
- Control logic
  - ForwardA =
    - 10 if (($EX/MEM.Rd = ID/EX.Rs) && EX/MEM.RegWrite && (EX/MEM.Rd ≠ 0))
    - 01 if (($MEM/WB.Rd = ID/EX.Rs) && MEM/WB.RegWrite && (MEM/WB.Rd ≠ 0) && (EX/MEM.Rd ≠ ID/EX.Rs))
    - 00, otherwise
  - ForwardB =
    - 10 if (($EX/MEM.Rd = ID/EX.Rt) && EX/MEM.RegWrite && (EX/MEM.Rd ≠ 0))
    - 01 if (($MEM/WB.Rd = ID/EX.Rt) && MEM/WB.RegWrite && (MEM/WB.Rd ≠ 0) && (EX/MEM.Rd ≠ ID/EX.Rt))
    - 00, otherwise

Forwarding : Forwarding Unit

Forwarding unit: 6-input, 2-output combinational circuit
Where does ALUSrc go?

Data Hazards: All Considered ???

...but it doesn’t eliminate all data hazards:

1w $s5, 0($s4)  
add $s7, $s5, $s6

...especially when we remember that memory access is really often much longer than a single cycle:

1w $s5, 0($s4)  
add $s7, $s5, $s6
Data Hazards: Stalling

- Stall the pipeline by keeping an instruction in the same stage

<table>
<thead>
<tr>
<th>Program execution order (in instructions)</th>
<th>Time (in clock cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lw $1, 30($1)</code></td>
<td>CC 1</td>
</tr>
<tr>
<td><code>and $4, $5, $6</code></td>
<td>CC 2</td>
</tr>
<tr>
<td><code>or $8, $7, $6</code></td>
<td>CC 3</td>
</tr>
<tr>
<td><code>add $9, $4, $2</code></td>
<td>CC 4</td>
</tr>
<tr>
<td><code>at $1, $6, $7</code></td>
<td>CC 5</td>
</tr>
</tbody>
</table>

In-and
In-or
At CC5, MEM stage is empty !!!

Stalling detection and control

- Detects during the ID stage when “lw” instruction is in EX stage
  - The following two instructions are in ID (“and”) and IF (“or”) stages, respectively
- If detected,
  - Stall the following instruction (in ID stage, “and”) so that it repeats the ID stage again => IF/ID pipeline register should not be changed
  - Stall the second instruction (in IF stage, “or”) so that it repeats the IF stage again => PC should not be changed
Data Hazards: Stalling

- Hazard detection
  - If (ID/EX.MemRead and (ID/EX.Rt = IF/ID.Rs) or (ID/EX.Rt = IF/ID.Rt)) stall the pipeline

- Control signals generated from hazard detection unit
  - IF/IDWrite to prevent IF/ID register from changing
  - PCWrite to prevent PC from changing
  - MUX control to delay forwarding control signals (pass “null” signals)

Stalling: Detection Unit

- Stall by letting an instruction that won’t write anything go forward
Stalling: What happen in the pipleine?

- What does the pipeline do?
  - What does EX stage do at CC7?
    - simple add (ALUop1/ALUop0=00)
  - What does MEM stage do at CC8?
    - No MemRead/MemWrite (zero control signals)
  - What does WB stage do at CC9?
    - No RegWrite (zero control signals)
Data Hazards: Stalling

- Data forwarding for stalled instructions
  - They are needed; otherwise, we need one more stall cycle
    - “and” – forwarding to EX stage from WB stage (MEM/WB pipeline register)
    - “or” – no forwarding is required
  - Then, is this forwarding already covered?
    - It seems not because there is no “Rd” for lw instruction
    - But, it is covered because “Rd” actually means destination register among Rd and Rt, selected at EX stage

Example 1
- lw $1, 200($1)
- Or $1, $1, $6
- Sub $1, $1, $3

Example 2
- lw $1, 100($2)
- lw $1, 200($1)
- Sub $1, $1, $4

Any problems with
- Example 1
- Example 2
Data hazards: Stalling & Forwarding

Hazard detection (at CC6):
If (ID/EX.MemRead and ((ID/EX.Rt = IF/ID.Rs) or (ID/EX.Rt = IF/ID.Rt))) stall the pipeline

Forwarding (at CC8):
ForwardA = 01 if ((MEM/WB.Rd = ID/EX.Rs) && … (EX/MEM.Rd ≠ ID/EX.Rs))

Cleveland State University

e.yu91@csuohio.edu
Data hazards: Stalling & Forwarding

Cleveland State University

c.yu91@csuohio.edu
Data hazards: Stalling & Forwarding

Hazard detection (at CC6):
If (ID/EX.MemRead and
((ID/EX.Rt = IF/ID.Rs) or
(ID/EX.Rt = IF/ID.Ri))
stall the pipeline

Forwarding (at CC8):
ForwardA = 01
if ((MEM/WB.Rd = ID/EX.Rs)
& & … (EX/MEM.Rd ≠ ID/EX.Rs))

What else?