CIS 429/529
Hazard Resolution

Prof. Michel A. Kinsky
Resolving Data Hazards

- **Strategy 1:** Wait for the result to be available by freezing earlier pipeline stages
  - **interlocks**

- **Strategy 2:** Route data as soon as possible after it is calculated to the earlier pipeline stage
  - **bypass**
Interlocks to resolve Data Hazards

Stall Condition

... r1 ← r0 + 10
r4 ← r1 + 17
...

Computer Architecture and Embedded Systems Laboratory (CAES Lab)
Source & Destination Registers

**R-type:**

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>func</th>
</tr>
</thead>
</table>

**I-type:**

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate16</th>
</tr>
</thead>
</table>

**J-type:**

<table>
<thead>
<tr>
<th>op</th>
<th>immediate26</th>
</tr>
</thead>
</table>

**source(s) destination**

- **ALU**: \( rd \leftarrow (rs) \) func (rt)
  - \( rs, rt \rightarrow rd \)
- **ALUi**: \( rt \leftarrow (rs) \) op imm
  - \( rs \rightarrow rt \)
- **LW**: \( rt \leftarrow M [(rs) + \) imm\]
  - \( rs \rightarrow rt \)
- **SW**: \( M [(rs) + \) imm\] \( \leftarrow (rt) \)
  - \( rs, rt \)
- **BZ**: \( \text{cond} (rs) \)
  - \( true: \) PC \( \leftarrow (PC) + \) imm
    - \( rs \)
  - \( false: \) PC \( \leftarrow (PC) + 4 \)
    - \( rs \)
- **J**: PC \( \leftarrow (PC) + \) imm
- **JAL**: r31 \( \leftarrow (PC) \), PC \( \leftarrow (PC) + \) imm
  - r31
- **JR**: PC \( \leftarrow (rs) \)
  - rs
- **JALR**: r31 \( \leftarrow (PC) \), PC \( \leftarrow (rs) \)
  - rs
  - r31
### Deriving the Stall Signal

**C\_\text{dest}**

\[
\begin{align*}
ws &= \text{Case opcode} \\
    \text{ALU} &\rightarrow \text{rd} \\
    \text{ALUi, LW} &\rightarrow \text{rt} \\
    \text{JAL, JALR} &\rightarrow \text{R31}
\end{align*}
\]

\[
\begin{align*}
\text{we} &= \text{Case opcode} \\
    \text{ALU, ALUi, LW} &\rightarrow (ws \neq 0) \\
    \text{JAL, JALR} &\rightarrow \text{on} \\
    \ldots &\rightarrow \text{off}
\end{align*}
\]

**C\_\text{re}**

\[
\begin{align*}
\text{re1} &= \text{Case opcode} \\
    \text{ALU, ALUi, LW, SW, BZ, JR, JALR} &\rightarrow \text{on} \\
    \text{J, JAL} &\rightarrow \text{off}
\end{align*}
\]

\[
\begin{align*}
\text{re2} &= \text{Case opcode} \\
    \text{ALU, SW} &\rightarrow \text{on} \\
    \ldots &\rightarrow \text{off}
\end{align*}
\]

**C\_\text{stall}**

\[
\begin{align*}
stall &= ((rs\_D = ws\_E).we\_E + \\
    (rs\_D = ws\_M).we\_M + \\
    (rs\_D = ws\_W).we\_W).re\_D + \\
    ((rt\_D = ws\_E).we\_E + \\
    (rt\_D = ws\_M).we\_M + \\
    (rt\_D = ws\_W).we\_W).re\_2D
\end{align*}
\]
Resolving Data Hazards

- **Strategy 1**: Wait for the result to be available by freezing earlier pipeline stages
  - interlocks

- **Strategy 2**: Route data as soon as possible after it is calculated to the earlier pipeline stage
  - bypass
**Bypassing**

- Each stall or kill introduces a bubble in the pipeline \( \rightarrow \) CPI > 1

<table>
<thead>
<tr>
<th>time</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>
</tr>
</thead>
<tbody>
<tr>
<td>(I₁) r1 ( \leftarrow ) r₀ + 10</td>
<td>IF₁</td>
<td>ID₁</td>
<td>EX₁</td>
<td>MA₁</td>
<td>WB₁</td>
<td>ID₂</td>
<td>EX₂</td>
<td>MA₂</td>
</tr>
<tr>
<td>(I₂) r₄ ( \leftarrow ) r₁ + 17</td>
<td>IF₂</td>
<td>ID₂</td>
<td>EX₂</td>
<td>MA₂</td>
<td>WB₂</td>
<td>ID₃</td>
<td>EX₃</td>
<td>MA₃</td>
</tr>
<tr>
<td>(I₃)</td>
<td>IF₃</td>
<td>ID₃</td>
<td>EX₃</td>
<td>MA₃</td>
<td>WB₃</td>
<td>ID₄</td>
<td>EX₄</td>
<td>MA₄</td>
</tr>
<tr>
<td>(I₄)</td>
<td>IF₄</td>
<td>ID₄</td>
<td>EX₄</td>
<td>MA₄</td>
<td>WB₄</td>
<td>ID₅</td>
<td>EX₅</td>
<td>MA₅</td>
</tr>
<tr>
<td>(I₅)</td>
<td>IF₅</td>
<td>ID₅</td>
<td>EX₅</td>
<td>MA₅</td>
<td>WB₅</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**stalled stages**
Adding a Bypass

When does this bypass help?

(I_1) r1 ← r0 + 10  
(I_2) r4 ← r1 + 17

r1 ← M[r0 + 10]  
r4 ← r1 + 17

JAL 500  
r4 ← r31 + 17
The Bypass Signal

\[ C_{\text{dest}} \]
\[ ws = \text{Case opcode} \]
\[ \text{ALU} \rightarrow \text{rd} \]
\[ \text{ALUi, LW} \rightarrow \text{rt} \]
\[ \text{JAL, JALR} \rightarrow \text{R31} \]

\[ we = \text{Case opcode} \]
\[ \text{ALU, ALUi, LW} \rightarrow (ws \neq 0) \]
\[ \text{JAL, JALR} \rightarrow \text{on} \]
\[ \ldots \rightarrow \text{off} \]

\[ C_{\text{re}} \]
\[ \text{re1} = \text{Case opcode} \]
\[ \text{ALU, ALUi, LW, SW, BZ, JR, JALR} \rightarrow \text{on} \]
\[ \text{J, JAL} \rightarrow \text{off} \]

\[ \text{re2} = \text{Case opcode} \]
\[ \text{ALU, SW} \rightarrow \text{on} \]
\[ \ldots \rightarrow \text{off} \]

\[ C_{\text{stall}} \]
\[ \text{stall} = ((rs_D = ws_E).we_E + (rs_D = ws_M).we_M + (rs_D = ws_W).we_W).re1_D \]
\[ + ((rt_D = ws_E).we_E + (rt_D = ws_M).we_M + (rt_D = ws_W).we_W).re2_D \]

\[ \text{ASrc} = (rs_D = ws_E).we_E.\text{re1}_D \]

Is this correct?
Bypass and Stall Signals

• Split $w_{E}$ into two components: $w_{e}$-bypass, $w_{e}$-stall

\[
\begin{align*}
\text{we-bypass}_{E} & = \text{Case opcode}_{E} \\
& \text{ALU, ALU} \rightarrow (w_{s} \neq 0) \\
& \ldots \rightarrow \text{off}
\end{align*}
\]

\[
\begin{align*}
\text{we-stall}_{E} & = \text{Case opcode}_{E} \\
& \text{LW} \rightarrow (w_{s} \neq 0) \\
& \text{JAL, JALR} \rightarrow \text{on} \\
& \ldots \rightarrow \text{off}
\end{align*}
\]

\[\text{ASrc} = (r_{s_{D}} = w_{s_{E}}).\text{we-bypass}_{E} \cdot \text{re}_{1_{D}}\]

\[\text{stall} = ((r_{s_{D}} = w_{s_{E}}).\text{we-stall}_{E} + (r_{s_{D}} = w_{s_{M}}).\text{we}_{M} + \\
(r_{s_{D}} = w_{s_{W}}).\text{we}_{W}). \text{re}_{1_{D}} + ((r_{t_{D}} = w_{s_{E}}).\text{we}_{E} + \\
(r_{t_{D}} = w_{s_{M}}).\text{we}_{M} + (r_{t_{D}} = w_{s_{W}}).\text{we}_{W}). \text{re}_{2_{D}}\]
Fully Bypassed Datapath

PC for JAL, ...

0x4 Add

addr Inst Memory

nop

ASrc

ALU

Imm Ext

BSrc

MD1

MD2

W

R

A

B

Y

E

M

IR

IR

IR

IR

Addr Data Memory

Addr Data Memory

Add

inst

stall

D

we rs1 rs2

rd1

ws wrd rd2

GPRs

wdata

rdata

wdata

addr

inst
Is there still a need for the stall signal?

\[
\text{stall} = (\text{rs}_D = \text{ws}_E) \cdot (\text{opcode}_E = \text{LW}_E) \cdot (\text{ws}_E \neq 0) \cdot \text{re}_1_D + (\text{rt}_D = \text{ws}_E) \cdot (\text{opcode}_E = \text{LW}_E) \cdot (\text{ws}_E \neq 0) \cdot \text{re}_2_D
\]
Why a program may have CPI > 1

- Why an Instruction may not be dispatched every cycle (CPI>1)?
  - Full bypassing may be too expensive to implement
    - typically all frequently used paths are provided
    - some infrequently used bypass paths may increase cycle time and counteract the benefit of reducing CPI
  - Loads have two cycle latency
    - Instruction after load cannot use load result
    - MIPS-I ISA defined load delay slots, a software-visible pipeline hazard (compiler schedules independent instruction or inserts NOP to avoid hazard) - Removed in MIPS-II
Why a program may have CPI > 1

• Why an Instruction may not be dispatched every cycle (CPI>1)?
  ‣ Conditional branches may cause bubbles
    • kill following instruction(s) if no delay slots
Branch or Load Delay Slots

• Expose control hazard to software:
  ‣ Change the ISA semantics so that the instruction that follows a jump or branch is always executed
  • Gives compiler the flexibility to put in a useful instruction where normally a pipeline bubble would have resulted

| I₁  | 096 | ADD       |
| I₂  | 100 | BEQZ r1 200 |
| I₃  | 104 | ADD       |
| I₄  | 304 | ADD       |

Delay slot instruction executed regardless of branch outcome

• Other techniques include branch prediction, which can dramatically reduce the branch penalty... to come later
Critical paths and clocks

- Hazard resolution, bypassing and other complex speculation strategies can make the control circuitry determine the length of the critical path because of communication between various pipeline stages.

- We will discuss techniques to address this problem in future …