Two special cases can arise on the same clock cycle.

**Combination A:**
- Not-taken branch
- `ret` instruction at branch target

**Combination B:**
- Instruction that reads from memory to `%esp`
- Followed by `ret` instruction
### Control Combination A

![Diagram showing Control Combination A with Load/use, Mispredict, and ret 1 stages]

#### Condition Table

<table>
<thead>
<tr>
<th>Condition</th>
<th>F</th>
<th>D</th>
<th>E</th>
<th>M</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing ret</td>
<td>stall</td>
<td>bubble</td>
<td>normal</td>
<td>normal</td>
<td>normal</td>
</tr>
<tr>
<td>Mispredicted Branch</td>
<td>normal</td>
<td>bubble</td>
<td>bubble</td>
<td>normal</td>
<td>normal</td>
</tr>
<tr>
<td>Combination</td>
<td>stall</td>
<td>bubble</td>
<td>bubble</td>
<td>normal</td>
<td>normal</td>
</tr>
</tbody>
</table>

- Should handle as mispredicted branch.
- Stalls F pipeline register.
- But PC selection logic will be using M_{val}M anyway.
Control Combination B: First Attempt

Would attempt to bubble and stall pipeline register D.

Signalled by processor as pipeline error.
Control Combination B: Correct Approach

- Load/use hazard should get priority.
- ret instruction should be held in decode stage for additional cycle.
## Corrected Pipeline Control Logic

<table>
<thead>
<tr>
<th>Condition</th>
<th>F</th>
<th>D</th>
<th>E</th>
<th>M</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing ret</td>
<td>stall</td>
<td>bubble</td>
<td>normal</td>
<td>normal</td>
<td>normal</td>
</tr>
<tr>
<td>Load/Use Hazard</td>
<td>stall</td>
<td>stall</td>
<td>bubble</td>
<td>normal</td>
<td>normal</td>
</tr>
<tr>
<td>Combination</td>
<td>stall</td>
<td>stall</td>
<td>bubble</td>
<td>normal</td>
<td>normal</td>
</tr>
</tbody>
</table>

- **Load / use hazard should get priority.**
- **`ret` instruction should be held in decode stage for additional cycle.**
Pipeline Summary

Data Hazards
- Most handled by forwarding with no performance penalty
- Load / use hazard requires one cycle stall

Control Hazards
- Cancel instructions when detect mispredicted branch; two cycles wasted
- Stall fetch stage while ret pass through pipeline; three cycles wasted.

Control Combinations
- Must analyze carefully
- First version had a subtle bug
- Only arises with unusual instruction combination
Performance Analysis with Pipelining

\[
\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}
\]

- Ideal pipelined machine: CPI = 1
  - One instruction completed per cycle.
  - But much faster cycle time than unpipelined machine.
- However, hazards work against the ideal
  - Hazards resolved using forwarding are fine.
  - Stalling degrades performance and instruction completion rate is interrupted.
- CPI is a measure of the “architectural efficiency” of the design.
Computing CPI

CPI is a function of useful instructions and bubbles:

\[
CPI = \frac{C_i + C_b}{C_i} = 1.0 + \frac{C_b}{C_i}
\]

You can reformulate this to account for:

- load penalties \((lp)\)
- branch misprediction penalties \((mp)\)
- return penalties \((rp)\)

\[
CPI = 1.0 + lp + mp + rp
\]
So, how do we determine the penalties?

- Depends on how often each situation occurs on average.
- How often does a load occur and how often does that load cause a stall?
- How often does a branch occur and how often is it mispredicted?
- How often does a return occur?

We can measure these using:

- a simulator, or
- hardware performance counters.

We can also estimate them through historical averages.

- Then use estimates to make early design tradeoffs for the architecture.
### Computing CPI (3)

<table>
<thead>
<tr>
<th>Cause</th>
<th>Name</th>
<th>Instruction Frequency</th>
<th>Condition Frequency</th>
<th>Stalls</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load/use</td>
<td>lp</td>
<td>0.30</td>
<td>0.3</td>
<td>1</td>
<td>0.09</td>
</tr>
<tr>
<td>Mispredict</td>
<td>mp</td>
<td>0.20</td>
<td>0.4</td>
<td>2</td>
<td>0.16</td>
</tr>
<tr>
<td>Return</td>
<td>rp</td>
<td>0.02</td>
<td>1.0</td>
<td>3</td>
<td>0.06</td>
</tr>
<tr>
<td><strong>Total penalty</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>0.31</strong></td>
</tr>
</tbody>
</table>

\[
\text{CPI} = 1 + 0.31 = 1.31 \approx 31\%
\]

This is not ideal.

This gets worse when:
- you also account for non-ideal memory access latency;
- deeper pipeline (where stalls per hazard increase).