Systems I

Pipelining IV

Topics

- Implementing pipeline control
- Pipelining and performance analysis
Implementing Pipeline Control

- Combinational logic generates pipeline control signals
- Action occurs at start of following cycle
Initial Version of Pipeline Control

bool F_stall =
    # Conditions for a load/use hazard
    E_icode in { IMRMOVl, IPOPL } && E_dstM in { d_srcA, d_srcB } ||
    # Stalling at fetch while ret passes through pipeline
    IRET in { D_icode, E_icode, M_icode };

bool D_stall =
    # Conditions for a load/use hazard
    E_icode in { IMRMOVl, IPOPL } && E_dstM in { d_srcA, d_srcB };

bool D_bubble =
    # Mispredicted branch
    (E_icode == IJXX && !e_Bch) ||
    # Bubble for ret
    IRET in { D_icode, E_icode, M_icode };

bool E_bubble =
    # Mispredicted branch
    (E_icode == IJXX && !e_Bch) ||
    # Load/use hazard
    E_icode in { IMRMOVl, IPOPL } && E_dstM in { d_srcA, d_srcB};
Control Combinations

- Special cases that can arise on 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

- Should handle as mispredicted branch
- Stalls F pipeline register
- But PC selection logic will be using M_valM anyhow

<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>
## Control Combination B

- Would attempt to bubble and stall pipeline register D
- Signaled by processor as pipeline error

### Table: Control Combination B Conditions

<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>bubble + stall</td>
<td>bubble</td>
<td>normal</td>
<td>normal</td>
</tr>
</tbody>
</table>
Handling Control Combination B

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

### Table: Handling Control Combination B

<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>

- **Combination B**

![Diagram showing the handling of control combinations](image-url)
Corrected Pipeline Control Logic

```cpp
bool D_bubble =
    # Mispredicted branch
    (E_icode == IJXX && !e_Bch) ||
    # Stalling at fetch while ret passes through pipeline
    IRET in { D_icode, E_icode, M_icode }
    # but not condition for a load/use hazard
    && !(E_icode in { IMRMOVL, IPOPL }
         && E_dstM in { d_srcA, d_srcB });
```

<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
  - No performance penalty
- Load/use hazard requires one cycle stall

Control Hazards

- Cancel instructions when detect mispredicted branch
  - Two clock cycles wasted
- Stall fetch stage while \( ret \) passes through pipeline
  - Three clock cycles wasted

Control Combinations

- Must analyze carefully
- First version had 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 are working against the ideal**

- Hazards resolved using forwarding are fine
- Stalling degrades performance and instruction completion rate is interrupted

**CPI is measure of “architectural efficiency” of design**
CPI for PIPE

CPI \approx 1.0

- Fetch instruction each clock cycle
- Effectively process new instruction almost every cycle
  - Although each individual instruction has latency of 5 cycles

CPI > 1.0

- Sometimes must stall or cancel branches

Computing CPI

- C clock cycles
- I instructions executed to completion
- B bubbles injected (C = I + B)

\[
\text{CPI} = \frac{C}{I} = \frac{(I+B)}{I} = 1.0 + \frac{B}{I}
\]

- Factor B/I represents average penalty due to bubbles
Computing CPI

CPI

- Function of useful instruction and bubbles
  \[ CPI = \frac{C_i + C_b}{C_i} = 1.0 + \frac{C_b}{C_i} \]
- \( C_b/C_i \) represents the pipeline penalty due to stalls

Can reformulate 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

- simulator
- hardware performance counters

We can estimate through historical averages

- Then use to make early design tradeoffs for architecture
Computing CPI - III

<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>Total penalty</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.31</td>
</tr>
</tbody>
</table>

CPI = 1 + 0.31 = 1.31 == 31% worse than ideal

This gets worse when:
- Account for non-ideal memory access latency
- Deeper pipelines (where stalls per hazard increase)
CPI for PIPE (Cont.)

\[ \text{B/I} = \text{LP} + \text{MP} + \text{RP} \]

- **LP:** Penalty due to load/use hazard stalling
  - Fraction of instructions that are loads: 0.25
  - Fraction of load instructions requiring stall: 0.20
  - Number of bubbles injected each time: 1
  \[ \Rightarrow \text{LP} = 0.25 \times 0.20 \times 1 = 0.05 \]

- **MP:** Penalty due to mispredicted branches
  - Fraction of instructions that are cond. jumps: 0.20
  - Fraction of cond. jumps mispredicted: 0.40
  - Number of bubbles injected each time: 2
  \[ \Rightarrow \text{MP} = 0.20 \times 0.40 \times 2 = 0.16 \]

- **RP:** Penalty due to return instructions
  - Fraction of instructions that are returns: 0.02
  - Number of bubbles injected each time: 3
  \[ \Rightarrow \text{RP} = 0.02 \times 3 = 0.06 \]

- **Net effect of penalties:**
  \[ 0.05 + 0.16 + 0.06 = 0.27 \]
  \[ \Rightarrow \text{CPI} = 1.27 \quad \text{(Not bad!)} \]

**Typical Values**

- Fraction of instructions that are loads: 0.25
- Fraction of load instructions requiring stall: 0.20
- Number of bubbles injected each time: 1
- Fraction of instructions that are cond. jumps: 0.20
- Fraction of cond. jumps mispredicted: 0.40
- Number of bubbles injected each time: 2
- Fraction of instructions that are returns: 0.02
Summary

Today

- Pipeline control logic
- Effect on CPI and performance

Next Time

- Further mitigation of branch mispredictions
- State machine design