RISC Architecture: Pipeline Hazard

Virendra Singh
Associate Professor
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in
Ways to Handle Branch

- Stall or bubble
- Delayed branch
- Branch prediction:
  - Heuristics
    - Next instruction
    - Prediction based on statistics (dynamic)
    - Hardware decision (dynamic)
  - Prediction error: pipeline flush
Branch Prediction

• Useful for program loops.
• A one-bit prediction scheme: a one-bit buffer carries a **history bit** that tells what happened on the last branch instruction
  • History bit = 1, branch was taken
  • History bit = 0, branch was not taken
Branch Prediction

Address of recent branch instructions | Target addresses | History bit(s)

Low-order bits used as index

PC

PC+4

Next PC

Prediction Logic

0

1
Branch Prediction for a Loop

1. \( I = 0 \)
2. \( I = I + 1 \)
3. \( X = X + R(I) \)
4. \( I - 10 = 0? \)
5. Store \( X \) in memory

Execution of Instruction 4

<table>
<thead>
<tr>
<th>Exec. seq.</th>
<th>Old hist. bit</th>
<th>Next instr.</th>
<th>New hist. bit</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>2</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>2</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>2</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

h.bit = 0 branch not taken, h.bit = 1 branch taken.
Two-Bit Prediction Buffer

• Can improve correct prediction statistics.

Predict branch taken 11

Predict branch not taken 01

Predict branch taken 10

Predict branch not taken 00

Not taken

Not taken

Not taken

Not taken

Not taken

Not taken

Not taken

Not taken

Not taken

Not taken

Not taken

Not taken

Not taken
Branch Prediction for a Loop

1. \( I = 0 \)

2. \( I = I + 1 \)

3. \( X = X + R(I) \)

4. \( I - 10 = 0? \)

5. Store X in memory

 Execution of Instruction 4

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Pred.</td>
<td>I</td>
<td>Act.</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>2</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>2</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
<td>2</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>2</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>2</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>
Summary: Hazards

• Structural hazards
  – Cause: resource conflict
  – Remedies: (i) hardware resources, (ii) stall (bubble)

• Data hazards
  – Cause: data unavailability
  – Remedies: (i) forwarding, (ii) stall (bubble), (iii) code reordering

• Control hazards
  – Cause: out-of-sequence execution (branch or jump)
  – Remedies: (i) stall (bubble), (ii) branch prediction/pipeline flush, (iii) delayed branch/pipeline flush
Single Lane Traffic
Wish List: Expresway
Limits of Pipelining

• IBM RISC Experience
  – Control and data dependences add 15%
  – Best case CPI of 1.15, IPC of 0.87
  – Deeper pipelines (higher frequency) magnify dependence penalties

• This analysis assumes 100% cache hit rates
  – Hit rates approach 100% for some programs
  – Many important programs have much worse hit rates
Processor Performance

Processor Performance = \frac{\text{Time}}{\text{Program}}

= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}

\text{(code size)} \quad \text{(CPI)} \quad \text{(cycle time)}

- In the 1980’s (decade of pipelining):
  - CPI: 5.0 => 1.15
- In the 1990’s (decade of superscalar):
  - CPI: 1.15 => 0.5 (best case)
- In the 2000’s (decade of multicore):
  - Marginal CPI improvement
Thank You