RISC Design: Pipeline Hazards

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

CP-226: Computer Architecture
Lecture 14 (13 March 2013)
Branch Hazard

• Consider heuristic – branch not taken.
• Continue fetching instructions in sequence following the branch instructions.
• If branch is taken (indicated by zero output of ALU):
  – Control generates $branch$ signal in ID cycle.
  – $branch$ activates $PCSource$ signal in the MEM cycle to load PC with new branch address.
  – Three instructions in the pipeline must be flushed if branch is taken – can this penalty be reduced?
Branch Not Taken

Branch to Z
A
B
C
D
Z

cycle b | cycle b+1 | cycle b+2 | cycle b+3 | cycle b+4
---|---|---|---|---
Branch fetched | Branch decoded | Branch decision | PC keeps D (br. not taken) | A continues
A fetched | A decoded | B fetched | B decoded | B executed
C fetched | C decoded | D fetched
Branch Taken

Branch to Z
A
B
C
D
Z

cycle b  cycle b+1  cycle b+2  cycle b+3  cycle b+4

Branch fetched  Branch decoded  Branch decision  PC gets Z (br. taken)
A fetched  A decoded  A executed
B fetched  B decoded
C fetched

Three instructions are flushed if branch is taken

Nop
Nop
Z fetched
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

= prediction logic

PC+4

0

1

Next PC

Address of target branch

Instructions

History bit(s)
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 - tion 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>1</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>1</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>1</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.
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
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{Time}{Program}

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

(code size) \times (CPI) \times (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
Pipelined Performance Model

- $g$ = fraction of time pipeline is filled
- $1-g$ = fraction of time pipeline is not filled (stalled)
g = fraction of time pipeline is filled
1-\(g\) = fraction of time pipeline is not filled (stalled)
• Tyranny of Amdahl’s Law [Bob Colwell]
  – When $g$ is even slightly below 100%, a big performance hit will result
  – Stalled cycles are the key adversary and must be minimized as much as possible
## Limits on Instruction Level Parallelism (ILP)

<table>
<thead>
<tr>
<th>Reference</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weiss and Smith [1984]</td>
<td>1.58</td>
</tr>
<tr>
<td>Sohi and Vajapeyam [1987]</td>
<td>1.81</td>
</tr>
<tr>
<td>Tjaden and Flynn [1970]</td>
<td>1.86 (Flynn’s bottleneck)</td>
</tr>
<tr>
<td>Tjaden and Flynn [1973]</td>
<td>1.96</td>
</tr>
<tr>
<td>Uht [1986]</td>
<td>2.00</td>
</tr>
<tr>
<td>Smith et al. [1989]</td>
<td>2.00</td>
</tr>
<tr>
<td>Jouppi and Wall [1988]</td>
<td>2.40</td>
</tr>
<tr>
<td>Johnson [1991]</td>
<td>2.50</td>
</tr>
<tr>
<td>Acosta et al. [1986]</td>
<td>2.79</td>
</tr>
<tr>
<td>Wedig [1982]</td>
<td>3.00</td>
</tr>
<tr>
<td>Butler et al. [1991]</td>
<td>5.8</td>
</tr>
<tr>
<td>Melvin and Patt [1991]</td>
<td>6</td>
</tr>
<tr>
<td>Wall [1991]</td>
<td>7 (Jouppi disagreed)</td>
</tr>
<tr>
<td>Kuck et al. [1972]</td>
<td>8</td>
</tr>
<tr>
<td>Riseman and Foster [1972]</td>
<td>51 (no control dependences)</td>
</tr>
<tr>
<td>Nicolau and Fisher [1984]</td>
<td>90 (Fisher’s optimism)</td>
</tr>
</tbody>
</table>
Superscalar Proposal

- Go beyond single instruction pipeline, achieve IPC $> 1$
- Dispatch multiple instructions per cycle
- Provide more generally applicable form of concurrency (not just vectors)
- Geared for sequential code that is hard to parallelize otherwise
- Exploit fine-grained or instruction-level parallelism (ILP)
Thank You