Chapter 3 – Instruction-Level Parallelism and its Exploitation (Part 2)

ILP vs. Parallel Computers
Dynamic Scheduling (Section 3.4, 3.5)
Dynamic Branch Prediction (Section 3.3, 3.9, and Appendix C)
Hardware Speculation and Precise Interrupts (Section 3.6)
Multiple Issue (Section 3.7)
Static Techniques (Section 3.2, Appendix H)
Limitations of ILP
Multithreading (Section 3.11)
Putting it Together (Mini-projects)
Reducing penalties from control dependences

Basic idea

- Hardware guesses
  * Whether branch will be taken/not taken
  * Where the branch will go

Especially important for multiple issue processors

Desirable properties

- Good prediction rate
- Make correct prediction fast
- Don't slow too much on misprediction
Maintain a buffer with prediction bits

Index buffer with LSBs of branch instruction PC

Predict based on indexed bit, change bit on misprediction

Accessed in ID stage (not useful for simple 5-stage pipeline)

Limitation of 1-bit predictor?
Variations on Branch Prediction Buffer

Variations

- n-bit predictor
- Correlating predictors
- Tournament predictors
N-bit Predictor

Contains n-bit saturating counter
Count up if taken, down if not taken
Predict taken if $\geq 2^{*(n-1)}$; predict not taken if $< 2^{*(n-1)}$
2-bit good for loops
Use outcome of previous m branches and n-bit predictors

For each branch, the prediction buffer contains

An entry for each possible history of previous m branches
Each entry is an n-bit predictor

* This figure has been taken from Computer Architecture, A Quantitative Approach, 3rd Edition Copyright 2003 by Elsevier Inc. All rights reserved. It has been used with permission by Elsevier Inc.
(1,1) predictor
   Prediction based on 1 previous branch,
   1 bit predictor

Number of prediction entries per branch = ??

Number of bits per prediction entry = ??
Correlating Predictors Example

Loop:
  If a == 1 /* b1 */
    a = 0
  If a == 0 /* b2 */
    …
  
Let a = 1, 3, 1, 3, 1, 3, …

Notation: N=not taken; T=taken

Initialize (1,1) prediction buffer entries of b2 to NT
  (1ˢᵗ entry for previous branch taken, 2ⁿᵈ for not taken)

Direction of b1:

Direction of b2:

History at b2:

Prediction entries of b2:

Prediction for b2:
Tournament Predictor

Combine multiple predictors with a selector

Often combine a global predictor and a local predictor

Selector typically two bit saturating counter

Increment when predicted predictor correct, other incorrect
Tournament Predictor Example - Alpha 21264

Uses 4K 2-bit counters to choose from global and local predictor

Global predictor

4K entries of 2-bit predictors
Indexed by history of last 12 branches

Local predictor is a two-level predictor

History table with 1K 10-bit entries (for that branch)
- Each entry gives 10 most recent branch outcomes
  Indexes table of 1K entries with 3-bit counters

Total of 29K bits

Misprediction rate
- SPECfp95 – 1 per 1000
- SPECint95 – 11.5 per 1000
More Predictors

Lots of work on branch prediction
International Branch Prediction Competition!
Limitations

- May use bit from wrong PC
- Target must be known when branch resolved
Store target PC along with prediction

Accessed in IF stage

Next IF stage uses target PC

No bubbles on correctly predicted taken branch

Must store tag

More state

Can remove not-taken branches?
Branch Target Cache With Target Instruction

Store target instruction along with prediction
Send target instruction instead of branch into ID
Zero cycle branch - branch folding
Used for unconditional jumps
E.g., ARM Cortex A-53
**Return Address Stack (Section 3.9)**

Hardware stack for addresses for returns

Call pushes return address in stack

Return pops the address

Perfect prediction if stack length $\geq$ call depth
Speculative Execution

How far can we go with branch prediction?
Speculative fetch?
Speculative issue?
Speculative execution?
Speculative write?
Speculative Execution

Allows instructions after branch to execute before knowing if branch will be taken

Must be able to undo if branch is not taken

Often try to combine with dynamic scheduling

Key insight: Split Write stage into Complete and Commit
  Complete out of order
    No state update
  Commit in order
    State updated (instruction no longer speculative)

Use reorder buffer
Overview

Instructions complete out-of-order
Reorder buffer reorganizes instructions
Modify state in-order

<table>
<thead>
<tr>
<th>Entry</th>
<th>Busy</th>
<th>Type</th>
<th>Dest</th>
<th>Result</th>
<th>State</th>
<th>Excep</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>LD</td>
<td>4</td>
<td>Exec</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>BR</td>
<td></td>
<td>Exec</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>ADD</td>
<td>6</td>
<td>75</td>
<td>Compl</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Instruction tag now is reorder buffer entry
Re-order Buffer Pipeline

Issue:

Execute:

Complete:

Commit:
Precise Interrupts Again

Precise interrupts hard with dynamic scheduling

Consider our canonical code fragment:

\[
\begin{align*}
LF & \ F6,34 \ (R2) \\
LF & \ F2,45 \ (R3) \\
MULTF & \ F0,F2,F4 \\
SUBF & \ F8,F6,F2 \\
DIVF & \ F10,F0,F6 \\
ADDF & \ F6,F8,F2
\end{align*}
\]

What happens if \textbf{DIVF} causes an interrupt?

- ADDF has already completed

Out-of-order completion makes interrupts hard

But reorder buffer can help!
Reorder Buffer for Precise Interrupts
Re-order Buffer Drawback

Operands need to be read from reorder buffer or registers

Alternative: Rename registers
Many current machines

- More physical registers than logical registers
- Reorder buffer does not have values
- Read all values from registers

Rename mechanism

- Rename map stores mapping from logical to physical registers
  - (Logical register Ri mapped to physical register Rp)
- On issue, Ri mapped to Rp-new
- On completion, write to Rp-new
- On commit, old mapping of Ri discarded (free Rp-old)
- On misprediction, new mapping of Ri discarded (free Rp-new)