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

- 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)

Branch Prediction Buffer Strategies: Limitations

- May use bit from wrong PC
- Target must be known when branch resolved

Branch Target Buffer or Cache (Section 3.9)

- 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

**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>5</td>
<td>Compl</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</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:

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

What happens if 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
Rename Registers + Reorder Buffer

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)