Review Tomasulo

- Reservations stations: *implicit register renaming* to larger set of registers + buffering source operands
  - Prevents registers as bottleneck
  - Avoids WAR, WAW hazards of Scoreboard
  - Allows loop unrolling in HW
- Not limited to basic blocks (integer units gets ahead, beyond branches)
- Today, helps cache misses as well
  - Don’t stall for L1 Data cache miss (insufficient ILP for L2 miss?)
- Lasting Contributions
  - Dynamic scheduling
  - Register renaming
  - Load/store disambiguation
- 360/91 descendants are Pentium III; PowerPC 604; MIPS R10000; HP-PA 8000; Alpha 21264

Tomasulo Algorithm and Branch Prediction

- 360/91 predicted branches, but did not speculate: pipeline stopped until the branch was resolved
  - No speculation; only instructions that can complete
- Speculation with Reorder Buffer allows execution past branch, and then discard if branch fails
  - Just need to hold instructions in buffer until branch can commit

Case for Branch Prediction when Issue N instructions per clock cycle

1. Branches will arrive up to \( n \) times faster in an \( n \)-issue processor
2. Amdahl’s Law \( \Rightarrow \) relative impact of the control stalls will be larger with the lower potential CPI in an \( n \)-issue processor
7 Branch Prediction Schemes

1. 1-bit Branch-Prediction Buffer
2. 2-bit Branch-Prediction Buffer
3. Correlating Branch Prediction Buffer
4. Tournament Branch Predictor
5. Branch Target Buffer
6. Integrated Instruction Fetch Units
7. Return Address Predictors

Dynamic Branch Prediction

- Performance = f(accuracy, cost of misprediction)
- Branch History Table: Lower bits of PC address index table of 1-bit values
  - Says whether or not branch taken last time
  - No address check (saves HW, but may not be right branch)
- Problem: in a loop, 1-bit BHT will cause 2 mispredictions (avg is 9 iterations before exit):
  - End of loop case, when it exits instead of looping as before
  - First time through loop on next time through code, when it predicts exit instead of looping
  - Only 80% accuracy even if loop 90% of the time

Dynamic Branch Prediction
(Jim Smith, 1981)

- Solution: 2-bit scheme where change prediction only if get misprediction twice: (Figure 3.7, p. 198)
  - Red: stop, not taken
  - Green: go, taken
  - Adds hysteresis to decision making process

Prediction accuracy: 4K-entry 2-bit table vs infinite table size
Correlating Predictors

- 2-bit prediction uses a small amount of (hopefully) local information to predict behaviour
- Sometimes behaviour is correlated, and we can do better by keeping track of direction of related branches, for example consider the following code:

```plaintext
if (d==0)
    d = 1;
if (d==1) {
```
- If the first branch is not taken, neither is the second. Predictors that use the behaviour of other branches to make a prediction are called correlating predictors or two-level predictors.

Correlating Branches

Idea: taken/not taken of recently executed branches is related to behavior of next branch (as well as the history of that branch behavior)
- Then behavior of recent branches selects between, say, 4 predictions of next branch, updating just that prediction
- (2,2) predictor: 2-bit global, 2-bit local

Accuracy of Different Schemes

(Figure 3.15, p. 206)

- 4096 Entries 2-bit BHT
- Unlimited Entries 2-bit BHT
- 1024 Entries (2,2) BHT

Re-evaluating Correlation

- Several of the SPEC benchmarks have less than a dozen branches responsible for 90% of taken branches:
  
<table>
<thead>
<tr>
<th>program</th>
<th>branch %</th>
<th>static</th>
<th># = 90%</th>
</tr>
</thead>
<tbody>
<tr>
<td>compress</td>
<td>14%</td>
<td>236</td>
<td>13</td>
</tr>
<tr>
<td>eqntott</td>
<td>25%</td>
<td>494</td>
<td>5</td>
</tr>
<tr>
<td>gcc</td>
<td>15%</td>
<td>9531</td>
<td>2020</td>
</tr>
<tr>
<td>mpeg</td>
<td>10%</td>
<td>5598</td>
<td>532</td>
</tr>
<tr>
<td>real gcc</td>
<td>13%</td>
<td>17361</td>
<td>3214</td>
</tr>
</tbody>
</table>

- Real programs + OS more like gcc
- Small benefits beyond benchmarks for correlation? problems with branch aliases?
BHT Accuracy

- Mispredict because either:
  - Wrong guess for that branch
  - Got branch history of wrong branch when index the table
- 4096 entry table programs vary from 1% misprediction (nasa7, tomcatv) to 18% (eqntott), with spice at 9% and gcc at 12%
- For SPEC92, 4096 about as good as infinite table

Tournament Predictors

- Motivation for correlating branch predictors is 2-bit predictor failed on important branches; by adding global information, performance improved
- Tournament predictors: use 2 predictors, 1 based on global information and 1 based on local information, and combine with a selector
- Hopes to select right predictor for right branch

Tournament Predictor in Alpha 21264

- 4K 2-bit counters to choose from among a global predictor and a local predictor
- Global predictor also has 4K entries and is indexed by the history of the last 12 branches; each entry in the global predictor is a standard 2-bit predictor
  - 12-bit pattern: ith bit 0 => ith prior branch not taken; ith bit 1 => ith prior branch taken;
- Local predictor consists of a 2-level predictor:
  - Top level a local history table consisting of 1024 10-bit entries; each 10-bit entry corresponds to the most recent 10 branch outcomes for the entry. 10-bit history allows patterns 10 branches to be discovered and predicted.
  - Next level Selected entry from the local history table is used to index a table of 1K entries consisting a 3-bit saturating counters, which provide the local prediction
- Total size: $4K \times 2 + 4K \times 2 + 1K \times 10 + 1K \times 3 = 29K$ bits! (~180,000 transistors)

% of predictions from local predictor in Tournament Prediction Scheme

<table>
<thead>
<tr>
<th>Program</th>
<th>0%</th>
<th>20%</th>
<th>40%</th>
<th>60%</th>
<th>80%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>nasa7</td>
<td>100%</td>
<td>98%</td>
<td>94%</td>
<td>90%</td>
<td>98%</td>
<td></td>
</tr>
<tr>
<td>matrix300</td>
<td>100%</td>
<td>100%</td>
<td>94%</td>
<td>90%</td>
<td>96%</td>
<td></td>
</tr>
<tr>
<td>tomcatv</td>
<td>100%</td>
<td>98%</td>
<td>94%</td>
<td>90%</td>
<td>96%</td>
<td></td>
</tr>
<tr>
<td>doduc</td>
<td>100%</td>
<td>98%</td>
<td>94%</td>
<td>90%</td>
<td>96%</td>
<td></td>
</tr>
<tr>
<td>spice</td>
<td>100%</td>
<td>98%</td>
<td>94%</td>
<td>90%</td>
<td>96%</td>
<td></td>
</tr>
<tr>
<td>fppp</td>
<td>100%</td>
<td>98%</td>
<td>94%</td>
<td>90%</td>
<td>96%</td>
<td></td>
</tr>
<tr>
<td>gcc</td>
<td>100%</td>
<td>98%</td>
<td>94%</td>
<td>90%</td>
<td>96%</td>
<td></td>
</tr>
<tr>
<td>espresso</td>
<td>100%</td>
<td>98%</td>
<td>94%</td>
<td>90%</td>
<td>96%</td>
<td></td>
</tr>
<tr>
<td>eqntott</td>
<td>100%</td>
<td>98%</td>
<td>94%</td>
<td>90%</td>
<td>96%</td>
<td></td>
</tr>
<tr>
<td>li</td>
<td>100%</td>
<td>98%</td>
<td>94%</td>
<td>90%</td>
<td>96%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>55%</td>
<td>55%</td>
<td>55%</td>
<td>55%</td>
<td>55%</td>
<td>55%</td>
</tr>
</tbody>
</table>
**Accuracy v. Size (SPEC89)**

<table>
<thead>
<tr>
<th>Total predictor size (Kbits)</th>
<th>Local</th>
<th>Correlating</th>
<th>Tournament</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Pitfall: Sometimes bigger and dumber is better**

- 21264 uses tournament predictor (29 Kbits)
- Earlier 21164 uses a simple 2-bit predictor with 2K entries (or a total of 4 Kbits)
- SPEC95 benchmarks, 21264 outperforms
  - 21264 avg 11.5 mispredictions per 1000 instructions
  - 21164 avg 16.5 mispredictions per 1000 instructions
- Reversed for transaction processing (TP)!
  - 21264 avg 17 mispredictions per 1000 instructions
  - 21164 avg 15 mispredictions per 1000 instructions
- TP code much larger & 21164 hold 2X branch predictions based on local behavior (2K vs. 1K local predictor in the 21264)

**Need Address at Same Time as Prediction**

- Branch Target Buffer (BTB): Address of branch index to get prediction AND branch address (if taken)
  - Note: must check for branch match now, since can't use wrong branch address (Figure 3.19, p. 210)

- | Branch PC | Predicted PC |
- | PC of instruction | FETCH |
- | No: branch not predicted, proceed normally (Next PC = PC+4) | Yes: instruction is branch and use predicted PC as next PC |
- | Extra prediction state bits | ? |

**Extra prediction state bits**

- Yes: instruction is branch and use predicted PC as next PC
**Predicated Execution**

- Avoid branch prediction by turning branches into conditionally executed instructions:
  
  \[
  \text{if (x) then } A = B \text{ op } C \text{ else NOP}
  \]
  
  - If false, then neither store result nor cause exception
  - Expanded ISA of Alpha, MIPS, PowerPC, SPARC have conditional move; PA-RISC can annul any following instr.
  - IA-64: 64 1-bit condition fields selected so conditional execution of any instruction
  - This transformation is called "if-conversion"

- Drawbacks to conditional instructions
  - Still takes a clock even if "annulled"
  - Stall if condition evaluated late
  - Complex conditions reduce effectiveness; condition becomes known late in pipeline

**Special Case Return Addresses**

- Register Indirect branch hard to predict address
- SPEC89 85% such branches for procedure return
- Since stack discipline for procedures, save return address in small buffer that acts like a stack: 8 to 16 entries has small miss rate

**Dynamic Branch Prediction Summary**

- Prediction becoming important part of scalar execution
- Branch History Table: 2 bits for loop accuracy
- Correlation: Recently executed branches correlated with next branch.
  - Either different branches
  - Or different executions of some branches
- Tournament Predictor: more resources to competitive solutions and pick between them
- Branch Target Buffer: include branch address & prediction
- Predicated Execution can reduce number of branches, number of mispredicted branches
- Return address stack for prediction of indirect jump