**Review: Tomasulo Organization**

- From Mem → Load Buffers → Load Queue → Load1, Load2, Load3, Load4, Load5, Load6 → Add1, Add2, Add3 → FP Adders → Reservation Stations → FP Multipliers → FP Registers → Store Buffers → To Mem

**Review: Tomasulo Architecture**

- Reservations stations: 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)
- Dynamic Scheduling:
  - Scoreboarding/Tomasulo
  - In-order issue, out-of-order execution, out-of-order commit
- Branch prediction/speculation
  - Regularities in program execution permit prediction of branch directions and data values
  - Necessary for wide superscalar issue

**Recall: Prediction: Branches, Dependencies, Data**

- Prediction has become essential to getting good performance from scalar instruction streams.
- We will discuss predicting branches. However, architects are now predicting everything: data dependencies, actual data, and results of groups of instructions:
  - At what point does computation become a probabilistic operation + verification?
  - We are pretty close with control hazards already...
- Why does prediction work?
  - Underlying algorithm has regularities.
  - Data that is being operated on has regularities.
  - Instruction sequence has redundancies that are artifacts of way that humans/compilers think about problems.
- Prediction ⇒ Compressible information streams?
Review: Independent “Fetch” unit

Stream of Instructions
To Execute

Instruction Fetch
with Branch Prediction

Out-Of-Order Execution Unit

Correctness Feedback
On Branch Results

- Instruction fetch decoupled from execution
- Often issue logic (+ rename) included with Fetch

Review: Branches must be resolved quickly

- In our loop-unrolling example, we relied on the fact that branches were under control of “fast” integer unit in order to get overlap!

```
Loop:
LD   F0  0   R1
MULTD F4 F0 F2
SD   F4  0   R1
SUBI R1 R1 #8
BNEZ R1 Loop
```

- What happens if branch depends on result of multd??
  - We completely lose all of our advantages!
  - Need to be able to “predict” branch outcome.
  - If we were to predict that branch was taken, this would be right most of the time.

- Problem much worse for superscalar machines!

Dynamic Branch Prediction

- Prediction could be “Static” (at compile time) or “Dynamic” (at runtime)
  - For our example, if we were to statically predict “taken”, we would only be wrong once each pass through loop

- Is dynamic branch prediction better than static branch prediction?
  - Seems to be. Still some debate to this effect
  - Today, lots of hardware being devoted to dynamic branch predictors.

- Does branch prediction make sense for 5-stage, in-order pipeline? What about 8-stage pipeline?
  - Perhaps: eliminate branch delay slots
  - Then predict branches

Simple dynamic prediction: Branch Target Buffer (BTB)

- Address of branch index to get prediction AND branch address (if taken)
  - Must check for branch match now, since can’t use wrong branch address
  - Grab predicted PC from table since may take several cycles to compute

- Update predicted PC when branch is actually resolved

- Return instruction addresses predicted with stack
Branch History Table

- BHT is a table of “Predictors”
  - Usually 2-bit, saturating counters
  - Indexed by PC address of Branch – without tags

- In Fetch state of branch:
  - BTB identifies branch
  - Predictor from BHT used to make prediction

- When branch completes
  - Update corresponding Predictor

Dynamic Branch Prediction

- Branch History Table: Lower bits of PC address prediction table
  - Branch Target Buffer (BTB): identify branches and hold *taken addresses*
    - Trick: identify branch before fetching instruction!
  - Branch History Table makes prediction
    - Simplest possibility – could be something much more complicated.
    - No address check: Can be good, can be bad….
  - Simple 1-bit BHT: keep last direction of branch
  - Problem: in a loop, 1-bit BHT will cause two 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
  - Performance = \( f(\text{accuracy, cost of misprediction}) \)
    - Misprediction \( \Rightarrow \) Flush Reorder Buffer

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%
- 4096 about as good as infinite table (in Alpha 211164)
Correlating Branches
° Hypothesis: behavior of recently executed branches affects prediction of current branch
° Two possibilities: Current branch depends on:
  • Last m most recently executed branches anywhere in program
    Produces a “GA” (for “global address”) in the Yeh and Patt classification (e.g. GAg)
  • Last m most recent outcomes of same branch.
    Produces a “PA” (for “per address”) in same classification (e.g. PAg)
° Idea: record m most recently executed branches as taken or not taken, and use that pattern to select the proper branch history table entry
  • A single history table shared by all branches (appends a “g” at end), indexed by history value.
  • Address is used along with history to select table entry (appends a “p” at end of classification)
  • If only portion of address used, often appends an “s” to indicate “set-indexed” tables (i.e. GAs)

(2,2) GAs predictor
Branch address
2-bits per branch predictors

For instance, consider global history, set-indexed BHT. That gives us a GAs history table.

2-bit global branch history register

Accuracy of Different Schemes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Frequency of Mispredictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>4096 Entries 2-bit BHT</td>
<td>11%</td>
</tr>
<tr>
<td>Unlimited Entries 2-bit BHT</td>
<td>13%</td>
</tr>
<tr>
<td>1024 Entries (2,2) BHT</td>
<td>16%</td>
</tr>
</tbody>
</table>

HW support for More ILP
° Avoid branch prediction by turning branches into conditionally executed instructions:
  if (x) then A = B op C 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.
    • EPIC: 64 1-bit condition fields selected so conditional execution
° 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
Inherent limitations of ILP

- 1 branch in 5: How to keep a 5-way superscalar busy?
- Latencies of units: many operations must be scheduled
- Need about Pipeline Depth x No. Functional Units of independent instructions to keep fully busy
- Increase ports to Register File
  - VLIW example needs 7 read and 3 write for Int. Reg. & 5 read and 3 write for FP reg
- Increase ports to memory
- Current state of the art: Many hardware structures (such as issue/rename logic) have delay proportional to square of number of instructions issued/cycle

Programs

<table>
<thead>
<tr>
<th>IPC</th>
<th>gcc</th>
<th>espresso</th>
<th>li</th>
<th>fpopp</th>
<th>doducd</th>
<th>tomcatv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer: 18 - 60</td>
<td>54.8</td>
<td>62.6</td>
<td>17.9</td>
<td>75.2</td>
<td>118.7</td>
<td>150.1</td>
</tr>
<tr>
<td>FP: 75 - 150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Conflicting studies of amount

- Benchmarks (vectorized Fortran FP vs. integer C programs)
- Hardware sophistication
- Compiler sophistication

Initial HW Model here; MIPS compilers.

Assumptions for ideal/perfect machine to start:

1. Register renaming—infinit virtual registers and all WAW & WAR hazards are avoided
2. Branch prediction—perfect; no mispredictions
3. Jump prediction—all jumps perfectly predicted => machine with perfect speculation & an unbounded buffer of instructions available
4. Memory-address alias analysis—addresses are known & a store can be moved before a load provided addresses not equal

1 cycle latency for all instructions; unlimited number of instructions issued per clock cycle

Change from infinite window to examine to 2000 and maximum issue of 64 instructions per clock cycle

Programs

<table>
<thead>
<tr>
<th>IPC</th>
<th>gcc</th>
<th>espresso</th>
<th>li</th>
<th>fpopp</th>
<th>doducd</th>
<th>tomcatv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfect</td>
<td>35</td>
<td>3</td>
<td>21</td>
<td>32</td>
<td>61</td>
<td>48</td>
</tr>
<tr>
<td>Pick Cor. or BHT</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>14</td>
</tr>
<tr>
<td>BHT (512)</td>
<td>20</td>
<td>5</td>
<td>20</td>
<td>50</td>
<td>78</td>
<td>54</td>
</tr>
<tr>
<td>Profile</td>
<td>20</td>
<td>5</td>
<td>20</td>
<td>50</td>
<td>78</td>
<td>54</td>
</tr>
<tr>
<td>No prediction</td>
<td>20</td>
<td>5</td>
<td>20</td>
<td>50</td>
<td>78</td>
<td>54</td>
</tr>
</tbody>
</table>

More Realistic HW: Branch Impact
Summary:

- Parallelism hard to get from real hardware.
- Modern computer architects predict everything:
  - Branches
  - Data Dependencies
  - Data!
- Fairly simple hardware structures can do a good job of predicting branches:
  - Branch Target Buffer (BTB) identifies branches and branch offsets
  - Branch History Table (BHT) does prediction