The Case for Branch Prediction

- Dynamic scheduling increases the amount of ILP => control dependence becomes the limiting factor
- Multiple issue processors
  - Branches will arrive up to N times faster in an n-issue processor
  - Amdahl’s Law => relative impact of the control stalls will be larger with the lower potential CPI in an n-issue processor
- What have we done?
  - Static schemes for dealing with branches – compiler optimizes the branch behavior by scheduling it at compile time

7 Branch Prediction Schemes

- 1-bit Branch-Prediction Buffer
- 2-bit Branch-Prediction Buffer
- Correlating Branch Prediction Buffer
- Tournament Branch Predictor
- Branch Target Buffer
- Integrated Instruction Fetch Units
- Return Address Predictors

Basic Branch Prediction (1)

- Performance = f(accuracy, cost of misprediction)
- Branch History Table: a small table of 1-bit values indexed by the lower bits of PC address
  - Says whether or not branch taken last time
  - Useful only to reduce branch delay when it is longer than the time to compute the possible target PC
- No address check – BHT has no address tags, so the prediction bit may have been put by another branch that has the same low-order bits
- Prediction is a hint, assumed to be correct – fetching begins in the predicted direction; if it turns out to be wrong, the prediction bit is inverted and stored back
Basic Branch Prediction (2)

- 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
  - Ideally for highly regular branches, the accuracy of predictor = taken branch frequency
- Solution: use two-bit prediction schemes

2-bit Scheme

- States in a two-bit prediction scheme

  - Red: stop, not taken
  - Green: go, taken
  - Adds hysteresis to decision making process

BHT Implementation

- 1) Small, special “cache” accessed with the instruction address during the IF pipe stage
- 2) Pair of bits attached to each block in the instruction cache and fetched with the instruction
  - How many branches per instruction?
  - Complexity?
- Instruction is decoded as branch, and branch is predicted as taken => fetch from the target as soon as the PC is known
- Note: Does this scheme help for simple MIPS?

BHT Performance

- Prediction accuracy of 2-bit predictor with 4096 entries is ranging from over 99% to 82% or misprediction rate of 1% to 18%
- Real impact on performance: prediction accuracy + branch cost + branch frequency
- How to improve prediction accuracy?
  - Increase the size of the buffer (number of entries)
  - Increase the accuracy for each prediction (increase the number of bits)
- Both have limited impact!
Case for Correlating Predictors

- Basic two-bit predictor schemes
  - use recent behavior of a branch to predict its future behavior
- Improve the prediction accuracy
  - look also at recent behavior of other branches

```c
if (aa == 2) aa = 0;
if (bb == 2) bb = 0;
if (aa != bb) {}  
```

Use correlating predictors or two-level predictors.

---

An Example

- Introduce one bit of correlation
  - Each branch has two separate prediction bits: one prediction assuming the last branch executed was not taken, and another prediction assuming it was taken.

**Behavior of one-bit predictor with one bit of correlation initialized to NT/NT; Assume last branch NT**

```
<table>
<thead>
<tr>
<th>d=7</th>
<th>b1 prediction</th>
<th>b1 act</th>
<th>New b1 prediction</th>
<th>b2 prediction</th>
<th>new b2 prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NT/NT</td>
<td>NT/NT</td>
<td>T/NT</td>
<td>T/NT</td>
<td>T/NT</td>
</tr>
<tr>
<td>1</td>
<td>NT/NT</td>
<td>NT/NT</td>
<td>T/NT</td>
<td>T/NT</td>
<td>T/NT</td>
</tr>
<tr>
<td>2</td>
<td>T/NT</td>
<td>NT/NT</td>
<td>T/NT</td>
<td>T/NT</td>
<td>T/NT</td>
</tr>
<tr>
<td>3</td>
<td>NT/NT</td>
<td>NT/NT</td>
<td>T/NT</td>
<td>T/NT</td>
<td>T/NT</td>
</tr>
<tr>
<td>4</td>
<td>NT/NT</td>
<td>NT/NT</td>
<td>T/NT</td>
<td>T/NT</td>
<td>T/NT</td>
</tr>
<tr>
<td>5</td>
<td>NT/NT</td>
<td>NT/NT</td>
<td>T/NT</td>
<td>T/NT</td>
<td>T/NT</td>
</tr>
<tr>
<td>6</td>
<td>NT/NT</td>
<td>NT/NT</td>
<td>T/NT</td>
<td>T/NT</td>
<td>T/NT</td>
</tr>
<tr>
<td>7</td>
<td>NT/NT</td>
<td>NT/NT</td>
<td>T/NT</td>
<td>T/NT</td>
<td>T/NT</td>
</tr>
</tbody>
</table>
```

Rightarrow: Only misprediction is on the first iteration.

(1, 1) Predictor

- (1, 1) predictor from the previous example
  - Uses the behavior of the last branch to choose from among a pair of one-bit branch predictors
- (m, n) predictor
  - Uses the behavior of the last m branches to choose from among 2^m predictors, each of which is a n-bit predictor for a single branch
  - Global history of the most recent branches can be recorded in an m-bit shift register (each bit records whether a branch is taken or not)
(2,2) Predictor

- 2-bit global history to choose from among 4 predictors for each branch address
- 2-bit local predictor

(2,2) predictor is implemented as a linear memory array that is 2 bits wide; the indexing is done by concatenating the global history bits and the number of required bits from the branch address.

Fair Predictor Comparison

- Compare predictors that use the same number of state bits
  - number of state bits for (m, n):
    \[2^m \times n \times \text{(number of prediction entries)}\]
  - number of state bits for (0, n):
    \[n \times \text{(number of prediction entries)}\]
- Example: How many branch selected entries are in a (2,2) predictor that has a total of 8K state bits
  \[\Rightarrow 2^2 \times 2 \times \text{(number of entries)} = 8K\]
  \[\Rightarrow \text{number of branch selected entries is 1K}\]

Accuracy of Different Schemes

- 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:
  - program branch % static # = 90%
  - compress 14% 236 13
  - eqntott 25% 494 5
  - gcc 15% 9531 2020
  - mpeg 10% 5598 532
  - real gcc 13% 17381 3214
- Real programs + OS more like gcc
- Small benefits beyond benchmarks for correlation? problems with branch aliases?
Predicated Execution

- Avoid branch prediction by turning branches into conditionally executed instructions:
  - if (x) then A = B op C else NOP
  - 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

Predicated Execution: An Example

if (R1 > R2) {
    R3 = R1 + R2;
    R4 = R2 + 1;
} else
    R3 = R1 - R2;

SGT R5, R1, R2
BZ L1
ADD R3, R1, R2
ADDI R4, R2, #1
J After

After: ...

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 several levels of branch prediction tables together with an algorithm for choosing among predictors
  - Hopes to select right predictor for right branch
Tournament Predictor in Alpha 21264

1. 4K 2-bit counters to choose from among a global predictor and a local predictor.

   Legend:
   - 0/0 – Prediction for L is incorrect,
   - 0/1 – Prediction for G is incorrect

   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*2 + 4K*2 + 1K*10 + 1K*3 = 29K bits! (~180,000 transistors)

2. % of predictions from local predictor in Tournament Prediction Scheme

   ![Graph showing accuracy of branch prediction]
Accuracy v. Size (SPEC89)

Branch Target Buffers

- Prediction in DLX
  - Need to know from what address to fetch at the end of IF
  - Need to know whether the as-yet-undecoded instruction is branch, and if so, what the next PC should be
- Branch prediction cache that stores the predicted address for the next instruction after a branch is called a **branch target buffer (BTB)**

Branch Target Buffers

- 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
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)

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 same 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