Static Branch Prediction

- To reorder code around branches, need to predict branch statically when compile
- Simplest scheme is to predict a branch as taken
  - Average misprediction = untaken branch frequency = 34% SPEC
- More accurate scheme predicts branches using profile information collected from earlier runs, and modify prediction based on last run:

![Graph showing misprediction rates for different programs]

Adapted from Patterson and Hennessey (Morgan Kaufman Pubs)

Dynamic Branch Prediction

- 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
- Is dynamic branch prediction better than static branch prediction?
  - Seems to be
  - There are a small number of important branches in programs which have dynamic behavior

Adapted from Patterson and Hennessey (Morgan Kaufman Pubs)
Dynamic Branch Prediction

• Performance = \( f(\text{accuracy}, \text{cost of misprediction}) \)
• 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
• 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

Dynamic Branch Prediction

• Solution: 2-bit scheme where change prediction only if get misprediction twice
BHT Accuracy

- Mispredict because either:
  - Wrong guess for that branch
  - Got branch history of wrong branch when index the table

- 4096 entry table:

![BHT Accuracy Chart]

Correlated Branch Prediction

- Idea: record $m$ most recently executed branches as taken or not taken, and use that pattern to select the proper $n$-bit branch history table

- In general, $(m,n)$ predictor means record last $m$ branches to select between $2^m$ history tables, each with $n$-bit counters
  - Thus, old 2-bit BHT is a $(0,2)$ predictor

- Global Branch History: $m$-bit shift register keeping T/NT status of last $m$ branches.

- Each entry in table has $m$ $n$-bit predictors.
**Correlating Branches**

(2,2) predictor
- Behavior of recent branches selects between four predictions of next branch, updating just that prediction.

![Diagram showing 2-bit global branch history and branch address prediction]

**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>0%</td>
</tr>
<tr>
<td>Unlimited Entries 2-bit BHT</td>
<td>1%</td>
</tr>
<tr>
<td>1024 Entries (2,2) BHT</td>
<td>5%</td>
</tr>
</tbody>
</table>

![Bar chart showing frequency of mispredictions for different schemes]
Tournament Predictors

- Multilevel branch predictor
- Use $n$-bit saturating counter to choose between predictors
- Usual choice between global and local predictors

Tournament predictor using, say, 4K 2-bit counters indexed by local branch address. Chooses between:

- Global predictor
  - 4K entries index by history of last 12 branches ($2^{12} = 4K$)
  - Each entry is a standard 2-bit predictor

- Local predictor
  - Local history table: 1024 10-bit entries recording last 10 branches, index by branch address
  - The pattern of the last 10 occurrences of that particular branch used to index table of 1K entries with 3-bit saturating counters
Comparing Predictors (Fig. 2.8)

- Advantage of tournament predictor is ability to select the right predictor for a particular branch
  - Particularly crucial for integer benchmarks.
  - A typical tournament predictor will select the global predictor almost 40% of the time for the SPEC integer benchmarks and less than 15% of the time for the SPEC FP benchmarks.

Pentium 4 Misprediction Rate
(per 1000 instructions, not per branch)

≈6% misprediction rate per branch SPECint
(19% of INT instructions are branch)
≈2% misprediction rate per branch SPECfp
(5% of FP instructions are branch)
Branch Target Buffers (BTB)

- Branch target calculation is costly and stalls the instruction fetch.
- BTB stores PCs the same way as caches
- The PC of a branch is sent to the BTB
- When a match is found the corresponding Predicted PC is returned
- If the branch was predicted taken, instruction fetch continues at the returned predicted PC

Adapted from Patterson and Hennessey
(Morgan Kaufman Pubs)
Dynamic Branch Prediction Summary

• Prediction becoming important part of execution
• Branch History Table: 2 bits for loop accuracy
• Correlation: Recently executed branches correlated with next branch
  – Either different branches (GA)
  – Or different executions of same branches (PA)
• Tournament predictors take insight to next level, by using multiple predictors
  – usually one based on global information and one based on local information, and combining them with a selector
  – In 2006, tournament predictors using \( \approx 30K \) bits are in processors like the Power5 and Pentium 4
• Branch Target Buffer: include branch address & prediction