CMSC 611: Advanced Computer Architecture

Branch Prediction
Recall Branch Penalties

- CPI = (1 - branch%) * non-branch CPI + branch% * branch CPI
- CPI = (1 - branch%) * 1 + branch% * (1 + penalty)
- CPI = 1 + (branch% * penalty)
- penalty = not taken% * not taken cost + taken% * taken cost
Branching Dilemma

- Instruction Level Parallelism increases throughput
  - Worse, the more advanced the method
    - Deep pipeline, multiple functional units, n-issue per clock, …

- Control dependence rapidly becomes the limiting factor to the amount of ILP

- Compiler-based techniques can only rely on static program properties to handle control hazards

- Hardware-based techniques refer to the dynamic behavior of the program to predict the outcome of a branch
Assume
- 20% of instructions are branches
- 53% of branches are taken

Predict not taken
- CPI = 1 + 20% * (53%*1 + 47%*0) = 1.106

Predict taken
- CPI = 1 + 20% * (53%*1 + 47%*1) = 1.2

Penalty for being wrong
Penalty for not having the address ready in time
Branch Target Cache

• Predict not-taken: still stalls to wait for branch target computation
• If address could be guessed, the branch penalty becomes zero
• Cache predicted address based on branch address
• Complications for complex predictors: do we know in time?
Branch Target Cache

- PC of instruction to fetch
  - Look up
  - Predicted PC

Number of entries in branch-target buffer

- No: instruction is not predicted to be branch. Proceed normally
- Yes: then instruction is branch and predicted PC should be used as the next PC

Branch predicted taken or not taken
Handling Branch Target Cache

- No branch delay if the a branch prediction entry is found and is correct
- A penalty of two cycle is imposed for a wrong prediction or a cache miss
- Cache update on misprediction and misses can extend the time penalty
- Dealing with misses or misprediction is expensive and should be optimized
Return Address Cache

- Branch target caching can be applied to expedite unconditional jumps (branch folding) and returns for procedure calls.
- For calls from multiple sites, not clustered in time, a stack implementation of the branch target cache can be useful.

![Graph showing misprediction rate versus number of entries in the return stack.](image)
Basic Branch Prediction

- Simplest dynamic branch-prediction scheme
  - Use a branch history table to track when the branch was taken and not taken
  - Branch history table is a small 1-bit buffer indexed by lower bits of PC address with the bit is set to reflect the whether or not branch taken last time

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

- Problem: in a nested loop, 1-bit branch history table will cause two mispredictions:
  - End of loop case, when it exits instead of looping
  - First time through loop on next time through code, when it predicts exit instead of looping
2-bit Branch History Table

- A two-bit buffer better captures the history of the branch instruction
- A prediction must miss twice to change
N-bit Predictors

• 2-bit is a special case of n-bit counter
  – For every entry in the prediction buffer
  – Increment/decrement if branch taken/not
  – If the counter value is one half of the maximum value \((2n-1)\), predict taken

• Slow to change prediction, but can change
• **Prediction accuracy of a 4096-entry prediction buffer ranges from 82% to 99% for the SPEC89 benchmarks**

• **The performance impact depends on frequency of branching instructions and the penalty of misprediction**
**Optimal Size for 2-bit Branch Buffers**

- **Buffer size has little impact beyond a certain size**
- **Misprediction is because either:**
  - Wrong guess for that branch
  - Got branch history of wrong branch (different branches with same low-bits of PC)

---

**SPEC89 benchmarks**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Frequency of mispredictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>nasa7</td>
<td>1% 0%</td>
</tr>
<tr>
<td>matrix300</td>
<td>0% 0%</td>
</tr>
<tr>
<td>tomcatv</td>
<td>1% 0%</td>
</tr>
<tr>
<td>doduc</td>
<td>5% 5%</td>
</tr>
<tr>
<td>spice</td>
<td>9% 9%</td>
</tr>
<tr>
<td>fpppp</td>
<td>9% 9%</td>
</tr>
<tr>
<td>gcc</td>
<td>12% 11%</td>
</tr>
<tr>
<td>espresso</td>
<td>5% 5%</td>
</tr>
<tr>
<td>eqntott</td>
<td>18% 18%</td>
</tr>
<tr>
<td>li</td>
<td>10% 10%</td>
</tr>
</tbody>
</table>

- **4096 entries (2 bits/entry)**
- **Unlimited entries (2 bits/entry)**