Static Branch Prediction

♦ Simplest
  • Predict Taken
  • average misprediction rate = untaken branch frequency, which for the SPEC programs is 34%
  • Unfortunately, the correct prediction rate ranges from not very accurate (41%) to highly accurate (91%)

♦ Predict on the basis of branch direction?
  • choosing backward-going branches to be taken (loop)
  • forward-going branches to be not taken (if)
  • SPEC programs, however, most forward-going branches are taken => predict taken is better

♦ Predict branches on the basis of profile information collected from earlier runs
  • Misprediction varies from 5% to 22%
Seven Branch Prediction/Handling Schemes

- 1-bit Branch-Prediction
- 2-bit Branch-Prediction
- Correlating Branch Prediction
- Tournament Branch Predictor
- Branch Target Buffer
- Conditionally Executed Instructions
- Return Address Predictors

- Branch Prediction even more important when N instructions per cycle are issued
- Amdahl’s Law => relative impact of the control stalls will be larger with the lower potential CPI in an n-issue processor

Dynamic Branch Prediction

- Memorize behavior of each branch in last few executions. History can change dynamically. Simplest scheme -- use a single history bit. Very long and mostly empty storage array addressed by the PC
  - Indicates: was the branch taken last time (T-Taken, N)
- Branch History Table (BHT): Lower bits of PC address index table of 1-bit/entry
  - No full address check (saves HW, but may be wrong)
- 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 77.8% accuracy if 9 iterations per loop on average
2-bit Branch Prediction - Scheme 1

- 2-bit scheme where change prediction only if get misprediction twice but return in one step:

- **Red:** stop, not taken
- **Green:** go, taken

(Jim Smith, 1981)

Branch History Table (BHT)

- **BHT** is a table of “Predictors”
  - 2-bit, saturating counters indexed by PC address of Branch
- **In Fetch phase of branch:**
  - Predictor from BHT used to make prediction
- **When branch completes:**
  - Update corresponding Predictor
2-bit Branch Prediction - Scheme 2

- Another Solution: 2-bit scheme where change prediction (in either direction) only if get misprediction twice:

- Red: stop, not taken
- Green: go, taken

Lee & A. Smith, IEEE Computer, Jan 1984

---

Comparison

Scheme 1

Actual: N N T T N N N T T T
State: N* N N N N N* T* N* N T T
Predicted: N N N N N N N N N

Scheme 2

Actual: N N T T N N N T T T
State: N* N N N N N* T* N* N T T
Predicted: N N N N N N N N N

For both schemes
Further Comparison

- Alternating taken / not-taken
  
  Actual: \( N \ T \ N \ T \ N \ T \ N \)  
  State: \( N^* \ N^* \ N^* \ N^* \ N^* \ N^* \ N^* \ N^* \)  
  Predicted: \( N \ N \ N \ N \ N \ N \ N \ N \)  

- On average, both schemes achieve 80-95% accuracy with only a small difference in behavior  
- 3-bit predictors do not offer higher accuracy to justify their higher complexity

Correlating Branches

Idea: taken/not taken of recently executed branches is related to behavior of present branch (as well as the history of that branch behavior)  

Example: if \( x<1 \) then ...  
if \( x>1 \) then ...
(k,2) predictor: k-bit global, 2-bit local

Branch address (4 bits)

2-bits per branch local predictors

2-bit recent global branch history (01 = not taken then taken)

Global BHR – Global Branch History shift Register – k bits

Accuracy of Different Schemes
(SPEC benchmark)

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

Frequency of Mispredictions

4096 entries: 2-bits per entry
Unlimited entries: 2-bits/entry
1,024 entries (2,2)
**GShare Predictor**

Global BHR - Branch History shift Register  
PHT - Pattern History Table  

A particular branch can have several entries in the PHT for different global histories.

---

**Tournament Predictors**

- Some branches need only local information while others can benefit from global information.
- Tournament predictors: use two predictors, 1 based on global information and 1 based on local information, and combine with a selector.
- Hopes to select the right predictor for the right branch (or right context of the branch).
Tournament Predictor in Alpha 21264

- **Selector**: 4K 2-bit counters to choose among a global predictor and a local predictor (predictors “1” & “2”)
- **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 is 0 ⇒ ith prior branch not taken;
  - ith bit is 1 ⇒ ith prior branch taken;

![Diagram of Tournament Predictor](image-url)

Tournament Predictor in Alpha 21264

- **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! (~180K 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>98%</td>
<td>98%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>matrix300</td>
<td>100%</td>
<td>100%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tomcatv</td>
<td>94%</td>
<td>94%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doduc</td>
<td>90%</td>
<td>90%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>spice</td>
<td>55%</td>
<td>55%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fpppp</td>
<td>76%</td>
<td>76%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gcc</td>
<td>72%</td>
<td>72%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>espresso</td>
<td>63%</td>
<td>63%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eqntott</td>
<td>37%</td>
<td>37%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>li</td>
<td>69%</td>
<td>69%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Accuracy of Branch Prediction

- Profile: branch profile from last execution (static in that is encoded in instruction, but profile)

- Profile-based
- 2-bit counter
- Tournament
Accuracy v. Size (SPEC)

- Local - 2 bit counters
- Correlating - (2,2) scheme
- Tournament

Need Address at Same Time as Prediction

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

- Branch PC
- Predicted PC

Yes: instruction is branch; use predicted PC as next PC (if predict Taken)

No: branch not predicted; proceed normally (PC+4)
**Branch Target "Cache"**

- Branch Target cache - Only predicted taken branches
- "Cache" - Content Addressable Memory (CAM) or Associative Memory

![Diagram of Branch Target "Cache" Array](image)

- Standard RAM
- Content addressable memory

- n
- $2^n$
- Key_i, Data_i
- External KEY
- Yes: output data_i
- No: not found

---

**Branch Target "Cache"**

- Use a big Branch History Table & a small Branch Target Cache

![Diagram of Branch Target "Cache" History Table](image)

- Branch PC
- Predicted PC
- 30 bits
- PC
- Yes: predicted taken branch found
- Prediction state bits (optional)
- No: not found
Steps with Branch target Buffer for the 5-stage MIPS

Branch CPI Penalty = [BTB_hit_rate \times P(\text{Incorrect\_prediction}|\text{is in BTB}) \times \text{Penalty\_Cycles} + \left(1 - \text{BTB\_hit\_rate}\right) \times P(\text{Branch\_taken}|\text{not in BTB}) \times \text{Penalty\_Cycles}] = .11 \times .1 \times 2 + .89 \times .06 \times 2 = .127

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 interference
  - Expanded ISA of Alpha, MIPS, PowerPC, SPARC have conditional move.

- Drawbacks to conditional instructions
  - Still takes a clock even if “annulled”
  - Stall if condition evaluated late: Complex conditions reduce effectiveness since condition becomes known late in pipeline
Special Case: Return Addresses

♦ Register Indirect branch (JR r31) - hard to predict address
♦ An entry in the BTB would be useless
♦ SPEC 85% such branches for procedure return
♦ Use stack discipline for procedures, save return address in small buffer that acts like a stack for nested subroutines: 8 to 16 entries has small miss rate

Pitfall: Sometimes dumber is better

♦ Alpha 21264 uses tournament predictor (29 Kbits)
♦ Earlier 21164 uses a simple 2-bit local predictor with 2K entries (or a total of 4 Kbits)
♦ SPEC 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)