Branch and Target Predictions

Frontend organization, 1-bit BHT, 2-bit BHT, branch target buffer, return address stack
Instruction Flow

Instruction flow must be **continuous**

**Target prediction**: What is the target PC

**Branch prediction**: What direction does a branch take

**Return Address Prediction**: Special target prediction for return instructions

* If fast recovery is allowed

Flush signal, correct PC and feedback
Instruction Flow

Design questions:
- When is inst type known?
- How to avoid fetch stage bubbles?

Many alternatives, e.g. pre-decode instructions and move branch and RA predictors to fetch stage

Flush signal, correct PC and feedback
Mis-prediction Recovery

Pipeline flushing
- Mis-prediction is detected when a branch is resolved
- May wait until the branch is to be committed, and then flush the pipeline
- Fast recovery: Immediately and selectively flush misfetched instructions

Fetch stage flushing: Special cases, e.g.
- Branch predictor, if located at the dispatch stage, may not agree with the target predictor on branch direction
- Unconditional branches (jumps) were predicted as not-taken
Target Prediction: Branch Target Buffer

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

Example: BTB combined with BHT

<table>
<thead>
<tr>
<th>Branch PC</th>
<th>Predicted PC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- PC of instruction
- FETCH

=.

No: branch not predicted, proceed normally (Next PC = PC+4)
Yes: instruction is branch and use predicted PC as next PC
Extra prediction state Bits (see later)
Branch Prediction

- Predict branch direction: taken or not taken (T/NT)
  
  ![Branch Prediction Diagram]

- Static prediction: compilers decide the direction
- Dynamic prediction: hardware decides the direction using dynamic information
  
  1. 1-bit Branch-Prediction Buffer
  2. 2-bit Branch-Prediction Buffer
  3. Correlating Branch Prediction Buffer
  4. Tournament Branch Predictor
  5. and more ...

BNE R1, R2, L1

... L1: ...
...
Predictor for a Single Branch

General Form

1. Access
   PC

2. Predict
   Output T/NT

3. Feedback T/NT

1-bit prediction

Predict Taken

Feedback

Predict Taken

Not
Branch History Table of 1-bit Predictor

BHT also Called Branch Prediction Buffer in textbook

- Can use only one 1-bit predictor, but accuracy is low
- BHT: use a table of simple predictors, indexed by bits from PC
- Similar to direct mapped cache
- More entries, more cost, but less conflicts, higher accuracy
- BHT can contain complex predictors
1-bit BHT Weakness

Example: in a loop, 1-bit BHT will cause 2 mispredictions

Consider a loop of 9 iterations before exit:

```c
for (...){
    for (i=0; i<9; i++)
        a[i] = a[i] * 2.0;
}
```

- 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
2-bit Saturating Counter

Solution: 2-bit scheme where change prediction only if get misprediction *twice*: (Figure 3.7, p. 249)

- **Blue**: stop, not taken
- **Gray**: go, taken
- Adds *hysteresis* to decision making process
Correlating Branches

Code example showing the potential

If (d==0)
    d=1;
If (d==1)
    ...

Assemble code

BNEZ R1, L1
DADDIU R1,R0,#1
L1: DADDIU R3,R1,#-1
    BNEZ R3, L2
L2:
    ...

Observation: if BNEZ1 is not taken, then BNEZ2 is not taken
Correlating Branch Predictor

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

- Then behavior of recent branches selects between, say, 2 predictions of next branch, updating just that prediction
- (1,1) predictor: 1-bit global, 1-bit local
Correlating Branch Predictor

General form: \((m, n)\) predictor

- \(m\) bits for global history, \(n\) bits for local history
- Records correlation between \(m+1\) branches
- Simple implementation: global history can be store in a shift register
- Example: \((2, 2)\) predictor, 2-bit global, 2-bit local

Branch address (4 bits)

2-bits per branch local predictors

2-bit global branch history

(01 = not taken then taken)
(2, 2) Predictor

Branch address xxx101xx

2-bit per-branch predictors

000 001 010 011 100 101 110 111

2-bit global branch history

0 1

prediction
(2, 2) Predictor Update

Branch address: xxx101xx

2-bit per-branch predictors:

- 00
- 01
- 10
- 11

2-bit global branch history:

- 11
Accuracy of Different Schemes
(Figure 3.15, p. 206)

4096 Entries 2-bit BHT
Unlimited Entries 2-bit BHT
1024 Entries (2,2) BHT
Estimate Branch Penalty

EX: BHT correct rate is 95%, BTB hit rate is 95%

Average miss penalty is 6 cycles

How much is the branch penalty?
Return Addresses Prediction

- Register indirect branch hard to predict address
  - Many callers, one callee
  - Jump to multiple return addresses from a single address (no PC-target correlation)
- 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
Return Address Stack Design

Push address

- Branch address
- Size of instruction
- Target prediction
- Function call

Pop address

- Branch address
- BTB
- Is this a return?
- Target prediction
- Function return
Accuracy of Return Address Predictor
Tournament Branch Predictor

- Used in Alpha 21264: Track both “local” and global history
- Intended for mixed types of applications
- Global history: T/NT history of past k branches, e.g. 0 1 0 1 0 1 (NT T NT T NT T)

Diagram:

- PC
- Local Predictor
- Global Predictor
- Choice Predictor
- mux
- NT/T
- Global history
Tournament Branch Predictor

Local predictor: use 10-bit local history, shared 3-bit counters

Global and choice predictors:
Tournament Predictor

- Two component predictors $P_0$ and $P_1$
- One meta-predictor $M$
  - Meta-prediction = 0, branch prediction = $P_0$
  - Meta-prediction = 1, branch prediction = $P_1$
Tournament Meta-predictor Update

Rules

<table>
<thead>
<tr>
<th>CO (P0 correct?)</th>
<th>C1 (P1 correct?)</th>
<th>Modification to M</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Do nothing</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Saturating increment</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Saturating decrement</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Do nothing</td>
</tr>
</tbody>
</table>
Branch Predictors

- Smith (bimodal) predictor
- Pattern-based predictors
  - Two-level, gshare, bi-mode, gskewed, Agree, ...
- Predictors based on alternative contexts
  - Alloyed history, path history, loop counting, ...
- Hybrid predictors
  - Multiple component predictors + selection/fusion
  - Tournament, multihybrid, prediction fusion, ...
Branch Prediction With n-way Issue

1. Branches will arrive up to $n$ times faster in an $n$-issue processor.

2. Amdahl’s Law $\Rightarrow$ relative impact of the control stalls will be larger with the lower potential CPI in an $n$-issue processor.
Modern Design: Frontend and Backend

Frontend: Instruction fetch and dispatch
- To supply high-quality instructions to the backend
- Instruction flows in program order

Backend: Schedule/execute, Writeback and Commit
- Instructions are processed out-of-order

Frontend Enhancements
- Instruction prefetch: fetch ahead to deliver multiple instructions per cycle
- To handle multiple branches: may access multiple cache lines in one cycle, use prefetch to hide the cost
- Target and branch predictions may be integrated with instruction cache: e.g. Intel P4 trace cache
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
- Return address stack for prediction of indirect jump