Branch Prediction

Case for Branch Prediction when Issue N instructions per clock cycle

1. Branches will arrive up to n times faster in an n-issue processor
2. Amdahl's Law => relative impact of the control stalls will be larger with the lower potential CPI in an n-issue processor

conversely, need branch prediction to 'see' potential parallelism

Branch Prediction Schemes

1. 1-bit Branch-Prediction Buffer
2. 2-bit Branch-Prediction Buffer
3. Correlating Branch Prediction
4. Tournament Branch Predictor
5. Branch Target Buffer
6. Integrated Instruction Fetch Units
7. Return Address Predictors

Parts of the predictor

- Direction Predictor
  - For conditional branches
  - Predicts whether the branch will be taken
  - Examples:
    » Always taken; backwards taken
- Address Predictor
  - Predicts the target address (use if predicted taken)
  - Examples:
    » BTB; Return Address Stack; Precomputed Branch
- Recovery logic
Example gzip:

- gzip: loop branch A@ 0x1200098d8
  - Executed: 1359575 times
  - Taken: 1359565 times
  - Not-taken: 10 times
  - % time taken: 99% - 100%

Easy to predict (direction and address)

Example gzip:

- gzip: if branch B@ 0x12000fa04
  - Executed: 151409 times
  - Taken: 71480 times
  - Not-taken: 79929 times
  - % time taken: ~49%

Easy to predict? (maybe not/ maybe dynamically)

Branch Backwards

- Most backward branches are heavily TAKEN
- Forward branches slightly more likely to be NOT-TAKEN

Dynamic Branch Prediction

- Performance = \( f(\text{accuracy, cost of prediction and misprediction}) \)
- 1-bit branch prediction scheme - simplest.
- Branch History Table (BHT): Lower bits of PC address index table of 1-bit values
  - Says whether or not branch taken last time
  - No address check (saves HW, but may not be right branch)
- 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

Ref: The Effects of Predicated Execution on Branch Prediction
1-bit predictor

- 1-bit history (direction predictor)
  - Remember the last direction for a branch

Branch History Table

Dynamic Branch Prediction (Jim Smith, 1981)

- Better Solution: 2-bit scheme where change prediction only if get misprediction *twice:

![Diagram of 2-bit predictor]

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

2-bit predictor

- 2-bit history (direction predictor)

Branch History Table

Correlating Branches (2-level Branch Predictors)

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, 4 predictions of next branch, updating just that prediction

- (2,2) predictor: 2-bit global, 2-bit local

![Diagram of branch predictors and behavior]

Branch address (4 bits)

2-bits per branch

Local predictors

2-bit recent global branch history

(01 = not taken then taken)
Correlating Branches (2-level Branch Predictors)

(m, n) branch predictor
m-bit global history
n-bit local predictor

The number of bits in an (m, n) predictor is:

$$2^m \times n \times 2^k$$

A 16 entry (k=4) (2, 2) predictor has:

$$2^2 \times 2 \times 2^4 = 128 \text{ bits.}$$

Example

If (d == 0)
\[ d = 1; \]
if (d==1)
\[ \text{L1: BNEZ R1, L1 ; branch b1 (d!=0)} \]
\[ \text{ADDI R1, R0, #1 ; d==0, so d=1} \]
\[ \text{if (d==1)} \]
\[ \text{L1: BNEZ R3, L2 ;branch b2 (d!=1)} \]

Local history only: Behavior of a 1-bit predictor initialized to NT (not taken)

<table>
<thead>
<tr>
<th>Initial value of d</th>
<th>Pred b1</th>
<th>real b1</th>
<th>Pred b2</th>
<th>real b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>NT/NT</td>
<td>T</td>
<td>NT/NT</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>T</td>
<td>NT</td>
<td>NT/T</td>
<td>NT</td>
</tr>
<tr>
<td>2</td>
<td>NT</td>
<td>T</td>
<td>NT/T</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>T/NT</td>
<td>NT</td>
<td>NT/T</td>
<td>NT</td>
</tr>
</tbody>
</table>

The action of the 1-bit predictor with 1-bit correlation -- (1,1) predictor

<table>
<thead>
<tr>
<th>Initial value of d</th>
<th>Pred b1</th>
<th>real b1</th>
<th>Pred b2</th>
<th>real b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>T/T</td>
<td>T</td>
<td>T/T</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>T/T</td>
<td>NT</td>
<td>T/T</td>
<td>NT</td>
</tr>
<tr>
<td>2</td>
<td>T/NT</td>
<td>T</td>
<td>NT/T</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>T/NT</td>
<td>NT</td>
<td>NT/T</td>
<td>NT</td>
</tr>
</tbody>
</table>

Example

The action of the 1-bit predictor with 1-bit correlation -- (1,1) predictor

<table>
<thead>
<tr>
<th>Initial value of d</th>
<th>Pred b1</th>
<th>real b1</th>
<th>Pred b2</th>
<th>real b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>T/T</td>
<td>T</td>
<td>T/T</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>T/T</td>
<td>NT</td>
<td>T/T</td>
<td>NT</td>
</tr>
<tr>
<td>2</td>
<td>T/NT</td>
<td>T</td>
<td>NT/T</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>T/NT</td>
<td>NT</td>
<td>NT/T</td>
<td>NT</td>
</tr>
</tbody>
</table>
The action of the 2-bit predictor with 1-bit correlation -- (1,2) predictor
Initialized to NT(00)/NT(00)

<table>
<thead>
<tr>
<th>Initial d</th>
<th>Pred b1</th>
<th>real b1</th>
<th>Pred b2</th>
<th>real b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>NT(00)/NT(00)</td>
<td>T</td>
<td>NT(00)/NT(00)</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>NT(01)/NT(00)</td>
<td>NT</td>
<td>NT(00)/NT(01)</td>
<td>NT</td>
</tr>
<tr>
<td>2</td>
<td>NT(01)/NT(00)</td>
<td>T</td>
<td>NT(00)/NT(01)</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>T(11)/NT(00)</td>
<td>NT</td>
<td>NT(00)/T(11)</td>
<td>NT</td>
</tr>
</tbody>
</table>

The action of the 2-bit predictor with 2-bit correlation -- (2,2) predictor
Initialized to NT/NT/NT/NT

<table>
<thead>
<tr>
<th>Initial value of d</th>
<th>Pred b1</th>
<th>real b1</th>
<th>Pred b2</th>
<th>real b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>NT/NT/NT/NT</td>
<td>T</td>
<td>NT/NT/NT/NT</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>NT</td>
<td></td>
<td>NT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td></td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>NT</td>
<td></td>
<td>NT</td>
<td></td>
</tr>
</tbody>
</table>

Accuracy of Different Schemes

![Accuracy Graph]

- 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:
  
<table>
<thead>
<tr>
<th>Program</th>
<th>Branch %</th>
<th>Static</th>
<th># = 90%</th>
</tr>
</thead>
<tbody>
<tr>
<td>compress</td>
<td>14%</td>
<td>236</td>
<td>13</td>
</tr>
<tr>
<td>eqntott</td>
<td>25%</td>
<td>494</td>
<td>5</td>
</tr>
<tr>
<td>gcc</td>
<td>15%</td>
<td>9531</td>
<td>2020</td>
</tr>
<tr>
<td>mpeg</td>
<td>10%</td>
<td>5598</td>
<td>532</td>
</tr>
<tr>
<td>real gcc</td>
<td>13%</td>
<td>17361</td>
<td>3214</td>
</tr>
</tbody>
</table>

- Real programs + OS more like gcc
- Small benefits beyond benchmarks for correlation? problems with branch aliases?
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 2 predictors, 1 based on global information and 1 based on local information, and combine with a selector
• Hopes to select right predictor for right branch (or right context of branch)

7 Branch Prediction Schemes

1. 1-bit Branch-Prediction Buffer
2. 2-bit Branch-Prediction Buffer
3. Correlating Branch Prediction
4. Tournament Branch Predictor
5. Branch Target Buffer
6. Integrated Instruction Fetch Units
7. Return Address Predictors

Parts of the predictor

• Direction Predictor
  - For conditional branches
    » Predicts whether the branch will be taken
  - Examples:
    » Always taken; backwards taken
• Address Predictor
  - Predicts the target address (use if predicted token)
  - Examples:
    » BTB; Return Address Stack; Precomputed Branch
• Recovery logic
Example
BNEZ R1, L1 ; branch b1 (d!=0)
ADDI R1, R0, #1 ; d==0, so d=1
L1:
SUBI R3, R1, #1
BNEZ R3, L2 ; branch b2 (d!=1)
L2:

The action of the 1-bit predictor with 1-bit correlation -- (1,1) predictor
Initialized to T/T

<table>
<thead>
<tr>
<th>Initial value of d</th>
<th>Pred b1</th>
<th>real b1</th>
<th>Pred b2</th>
<th>real b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>T/T</td>
<td>T</td>
<td>T/T</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>T/T</td>
<td>NT</td>
<td>T/T</td>
<td>NT</td>
</tr>
<tr>
<td>2</td>
<td>T/NT</td>
<td>T</td>
<td>NT/T</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>T/NT</td>
<td>NT</td>
<td>NT/T</td>
<td>NT</td>
</tr>
</tbody>
</table>

The action of the 2-bit (per address (local) history) predictor with 2-bit (global history) correlation -- (2,2) predictor (GAp predictor (Yeh&Patt’s terminology))
Initialized to NT(00)/NT(00)/NT(00)/NT(00)

<table>
<thead>
<tr>
<th>Initial value of d</th>
<th>Pred b1</th>
<th>real b1</th>
<th>Pred b2</th>
<th>real b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>NT(00)/NT(00)/NT(00)/NT(00)</td>
<td>T</td>
<td>NT(00)/NT(00)/NT(00)/NT(00)</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>NT(01)/NT(00)/NT(00)/NT(00)</td>
<td>NT</td>
<td>NT(00)/NT(01)/NT(00)/NT(00)</td>
<td>NT</td>
</tr>
<tr>
<td>2</td>
<td>NT(01)/NT(00)/NT(00)/NT(00)</td>
<td>NT</td>
<td>NT(00)/NT(01)/NT(00)/NT(00)</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>T(11)/NT(00)/NT(00)/NT(00)</td>
<td>NT</td>
<td>NT(00)/T(11)/NT(00)/NT(00)</td>
<td>NT</td>
</tr>
</tbody>
</table>

The action of the 2-bit (per address (local) history) predictor with 2-bit (global history) correlation -- (2,2) predictor (GAp predictor (Yeh&Patt’s terminology))

<table>
<thead>
<tr>
<th>Pred b1</th>
<th>real b1</th>
<th>Pred b2</th>
<th>real b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>T(10)/NT(01)/T(10)/NT(00)</td>
<td>NT</td>
<td>NT(01)/NT(01)/NT(00)/T(11)</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>NT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>NT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NT</td>
<td>NT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>NT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NT</td>
<td>NT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Re-evaluating Correlation

- Several of the SPEC benchmarks have less than a dozen branches responsible for 90% of taken branches:
  
<table>
<thead>
<tr>
<th>Program</th>
<th>Branch %</th>
<th>Static</th>
<th># = 90%</th>
</tr>
</thead>
<tbody>
<tr>
<td>compress</td>
<td>14%</td>
<td>236</td>
<td>13</td>
</tr>
<tr>
<td>eqntott</td>
<td>25%</td>
<td>494</td>
<td>5</td>
</tr>
<tr>
<td>gcc</td>
<td>15%</td>
<td>9531</td>
<td>2020</td>
</tr>
<tr>
<td>mpeg</td>
<td>10%</td>
<td>5598</td>
<td>532</td>
</tr>
<tr>
<td>real gcc</td>
<td>13%</td>
<td>17361</td>
<td>3214</td>
</tr>
</tbody>
</table>

- Real programs + OS more like gcc
- Small benefits beyond benchmarks for correlation? problems with branch aliases?

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 2 predictors, 1 based on global information and 1 based on local information, and combine with a selector
- Hopes to select right predictor for right branch (or right context of branch)
Tournament (hybrid) predictors

Local predictor (e.g. 2-bit)

Global/gshare predictor (much more state)

Selection table (2-bit state machine)

How do you select which predictor to use?
How do you update the various predictor/selector?

Tournament Predictor in Alpha 21264

- 4K 2-bit counters to choose from among a global predictor and a local predictor
- 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: \( i \)th bit 0 \( \Rightarrow \) \( i \)th prior branch not taken;
  \( i \)th bit 1 \( \Rightarrow \) \( i \)th 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: \( 4K \times 2 + 4K \times 2 + 1K \times 10 + 1K \times 3 = 29K \) bits!
  (~180,000 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>100%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>matrix300</td>
<td>94%</td>
<td>100%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tomcatv</td>
<td>76%</td>
<td>55%</td>
<td>63%</td>
<td>69%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doduc</td>
<td>90%</td>
<td>62%</td>
<td>47%</td>
<td>69%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>spice</td>
<td>72%</td>
<td>63%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fpppp</td>
<td>76%</td>
<td>63%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gcc</td>
<td>69%</td>
<td>58%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>espresso</td>
<td>63%</td>
<td>47%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eqntott</td>
<td>37%</td>
<td>20%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>li</td>
<td>69%</td>
<td>58%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gcc</td>
<td>37%</td>
<td>20%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>espresso</td>
<td>63%</td>
<td>47%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>li</td>
<td>69%</td>
<td>58%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gcc</td>
<td>37%</td>
<td>20%</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 in encoded in instruction, but profile)
### Accuracy v. Size (SPEC89)

![Chart showing accuracy vs. size]

- **Local**
- **Correlating**
- **Tournament**

### Need Address at Same Time as Prediction

- **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

![Diagram illustrating need address at same time as prediction]

### Overriding Predictors

- Big predictors are slow, but more accurate
- Use a single cycle predictor in fetch
- Start the multi-cycle predictor
  - When it completes, compare it to the fast prediction.
    - If same, do nothing
    - If different, assume the slow predictor is right and flush pipeline.
- Advantage: reduced branch penalty for those branches mispredicted by the fast predictor and correctly predicted by the slow predictor

### Predicated Execution

- Avoid branch prediction by turning branches into conditionally executed instructions:
  - `if (x) then A = B op C else NOP`
    - If false, then neither store result nor cause exception
    - 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
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

General speculation

- Control speculation
  - "I think this branch will go to address 90004"
- Data speculation
  - "I'll guess the result of the load will be zero"
- Memory conflict speculation
  - "I don't think this load conflicts with any proceeding store."
- Error speculation
  - "I don't think there were any errors in this calculation"
Speculation in general

- Need to be 100% sure on final correctness!
  - So need a recovery mechanism
  - Must make forward progress!
- Want to speed up overall performance
  - So recovery cost should be low or expected rate of occurrence should be low.
  - There can be a real trade-off on accuracy, cost of recovery, and speedup when correct.
- Should keep the worst case in mind...

Control Flow Speculation

- Leading Speculation
  - Tag speculative instructions
  - Advance branch and following instructions
  - Buffer addresses of speculated branch instructions

Mis-speculation Recovery

- Eliminate Incorrect Path
  - Use branch tag(s) to deallocate completion buffer entries occupied by speculative instructions (now determined to be mis-speculated).
  - Invalidate all instructions in the decode and dispatch buffers, as well as those in reservation stations

- Start New Correct Path
  - Update PC with computed branch target (if it was predicted NT)
  - Update PC with sequential instruction address (if it was predicted T)
  - Can begin speculation once again when encounter a new branch

How expensive is a misprediction?

How soon can you restart?
Trailing Confirmation

- When branch is resolved, remove/deallocate speculation tag.
- Permit completion of branch and following instructions.

Fast Branch Rewind and Restart:

- Discard all ROB entries (and corresponding operations) younger than the mispredicted branches.
- Can restart immediately from the corrected branch target because the ROB has sufficient information (rename & value) to continue from where left off.
- Works with nested mispredictions!!

Rewinding/Flushing of Rename Table

- To reinitiate renaming:
  - Wait for all instructions older than the rewind point to drain clear of the pipeline and then reset register remapping to null.
  - Reorder buffer has to remember how to restored the map table to the point of the mispredicted branch.
  - Cache rename map after branch prediction.
  - Complicated multi-cycle logic.