Material from Section 4.3

This set under construction.

Outline

- Branch Prediction Overview
- One-Level Predictor
- Two-Level Correlating Predictor
- Other topics to be added.
- Sample Problems
Branch and Target Prediction

Motivation

Branches occur frequently in code.

At best, one cycle of branch delay; more with dependencies.

Therefore, impact on CPI is large.

Techniques

*Branch Prediction:*
Predict outcome of branch. (Taken or not taken.)

*Branch Target Prediction:*
Predict branch or other CTI’s target address.

*Branch Folding:*
Replace branch or other CTI with target instruction.
Branch Prediction

Methods Covered

- One-level predictor

- \((m, n)\) two-level correlating predictor or \((m, n)\) predictor for short.
Idea: Predict based on assumption that patterns hold.

Example:

```
LOOP:
   lw  r1, 0(r2)  ! Read random number, either 0 or 1.
   addi r2, r2, #4
   slt  r6, r2, r7
   beqz r1, SKIP
   addi r3, r3, #1
SKIP:
   bneq r6, LOOP  ! Loop executes 100 iterations.
   nop
```

Second branch, `bneq`, taken 99 out of 100 executions.

Pattern for `bneq`: T T T ... NT T T T

First branch shows no pattern.
SPEC89 benchmarks on IBM POWER (predecessor to PowerPC).

FIGURE 4.14 Prediction accuracy of a 4096-entry two-bit prediction buffer for the SPEC89 benchmarks.
FIGURE 4.15 Prediction accuracy of a 4096-entry two-bit prediction buffer versus an infinite buffer for the SPEC89 benchmarks.
Branch Prediction Terminology

*Outcome:* [of a branch instruction execution].
The outcome of the execution of a branch instruction.

*T:*
A taken branch.

*NT: or N*
A branch that is not taken.

*Prediction:* [made by branch prediction hardware].
The predicted outcome of a branch.

*Misprediction:*
An incorrectly predicted outcome.

*Prediction Accuracy:* [of a branch prediction scheme].
The number of correct predictions divided by the number of predictions.
**Speculative Execution:**
The execution of instructions following a predicted branch.

**Misprediction Recovery:**
Undoing the effect of speculatively executed instructions ... ... and re-starting instruction fetch at the correct address.
One-Level Branch Predictor

Idea: maintain a *branch history* for each branch instruction.

Branch histories stored in a *branch history table*.

Branch history can be an arbitrary finite state machine or a *counter*.

Branch outcome causes a change in branch history.

Branch prediction based on state of branch history.
Branch History Counter

If a counter used, branch history incremented when branch taken . . .
. . . and decremented when branch not taken.

Symbol $n$ denotes number of bits for branch history.

To save space and for performance reasons . . .
. . . branch history limited to a few bits, usually $n = 2$.

Branch history updated using a saturating counter.

A saturating counter is an arithmetic unit that can add or subtract one . . .
. . . in which $x + 1 \rightarrow x + 1$ for $x \in [0, 2^n - 2]$ . . .
. . . $x - 1 \rightarrow x - 1$ for $x \in [1, 2^n - 1]$ . . .
. . . $(2^n - 1) + 1 \rightarrow 2^n - 1$ . . .
. . . and $0 - 1 \rightarrow 0$.

For an $n$-bit counter, predict taken if counter $> 2^{n-1}$. 
One-Level Branch Predictor Hardware

Illustrated for Chapter-3 DLX implementation . . .
. . . even though prediction not very useful.

Branch Prediction Steps

1: Predict.

   Read branch history, available in ID.

2: Determine Branch Outcome

   Execute predicted branch in usual way.

3: Recover (If necessary.)

   Undo effect of speculatively executing instructions, start fetching from correct path.

4: Update Branch History
Branch History Table

Stores branch histories,

Implemented using a memory device.

Address (called index) is hash of branch address (PC).

For $2^m$-entry BHT, hash is $m$ lowest bits of branch PC **skipping alignment**.

<table>
<thead>
<tr>
<th>BHT Addr</th>
<th>Align.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch address:</td>
<td>0</td>
</tr>
<tr>
<td>31</td>
<td>m+2</td>
</tr>
</tbody>
</table>

Data input and output of BHT is branch history.
Sample Local History

Outcomes for individual branches, categorized by pattern, sorted by frequency.

Branches running \TeX\ text formatter compiled for SPARC (Solaris).

<table>
<thead>
<tr>
<th>Arbitrary, pat 60288, br732164, 0.7743 0.7170 0.7199 (0.19675)</th>
</tr>
</thead>
<tbody>
<tr>
<td>% Patterns</td>
</tr>
<tr>
<td>-------------</td>
</tr>
<tr>
<td>0: fe7f</td>
</tr>
<tr>
<td>1: ff3f</td>
</tr>
<tr>
<td>2: fcff</td>
</tr>
<tr>
<td>3: ff9f</td>
</tr>
<tr>
<td>4: f9ff</td>
</tr>
<tr>
<td>5: ffcf</td>
</tr>
<tr>
<td>6: 60</td>
</tr>
<tr>
<td>7: 180</td>
</tr>
<tr>
<td>8: 300</td>
</tr>
<tr>
<td>9: c0</td>
</tr>
</tbody>
</table>
### Short Loop, pat 124, br 137681, 0.8908 0.9055 0.7441 (0.03700)

<table>
<thead>
<tr>
<th>% Patterns</th>
<th># Branches</th>
<th>gshre</th>
<th>local</th>
<th>corr</th>
<th>Local History</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:</td>
<td>5555</td>
<td>0.0040</td>
<td>14753</td>
<td>0.987</td>
<td>0.981 0.912  TNTNTNTNTNTNTNTN</td>
</tr>
<tr>
<td>1:</td>
<td>aaaa</td>
<td>0.0040</td>
<td>14730</td>
<td>0.859</td>
<td>0.978 0.461  NTNTNTNTNTNTNTNT</td>
</tr>
<tr>
<td>2:</td>
<td>9249</td>
<td>0.0022</td>
<td>8062</td>
<td>0.997</td>
<td>0.992 0.988  TNNNTNNTNNTNTNTNT</td>
</tr>
<tr>
<td>3:</td>
<td>4924</td>
<td>0.0022</td>
<td>8055</td>
<td>0.997</td>
<td>0.998 0.998  NNTNNTNNTNNTNTNTN</td>
</tr>
<tr>
<td>4:</td>
<td>2492</td>
<td>0.0022</td>
<td>8047</td>
<td>0.993</td>
<td>0.991 0.009  NNTNNTNNTNNTNTNN</td>
</tr>
<tr>
<td>5:</td>
<td>db6d</td>
<td>0.0013</td>
<td>4864</td>
<td>0.713</td>
<td>0.915 0.065  TNNNTNNTNNTNTNTT</td>
</tr>
<tr>
<td>6:</td>
<td>b6db</td>
<td>0.0013</td>
<td>4713</td>
<td>0.862</td>
<td>0.903 0.926  TNNNTNNTNNTNTNTT</td>
</tr>
<tr>
<td>7:</td>
<td>6db6</td>
<td>0.0012</td>
<td>4640</td>
<td>0.991</td>
<td>0.978 0.970  NNTNNTNNTNNTNTTN</td>
</tr>
<tr>
<td>8:</td>
<td>bbbb</td>
<td>0.0008</td>
<td>3061</td>
<td>0.896</td>
<td>0.936 0.949  TNNNTTNTNNTNTNTN</td>
</tr>
</tbody>
</table>

### Long Loop?, pat 32, br 185795, 0.9170 0.9052 0.9096 (0.04993)

<table>
<thead>
<tr>
<th>% Patterns</th>
<th># Branches</th>
<th>gshre</th>
<th>local</th>
<th>corr</th>
<th>Local History</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:</td>
<td>fffe</td>
<td>0.0025</td>
<td>9204</td>
<td>0.902</td>
<td>0.930 0.913  NTTTTTTTTTTTTTTT</td>
</tr>
<tr>
<td>1:</td>
<td>8000</td>
<td>0.0025</td>
<td>9198</td>
<td>0.654</td>
<td>0.700 0.705  NNNNNNNNNNNNNNNT</td>
</tr>
<tr>
<td>2:</td>
<td>7fff</td>
<td>0.0022</td>
<td>8052</td>
<td>0.890</td>
<td>0.817 0.818  TTTTTTTTTTTTTTTT</td>
</tr>
<tr>
<td>3:</td>
<td>ffbf</td>
<td>0.0018</td>
<td>6800</td>
<td>0.933</td>
<td>0.908 0.920  TTTTTTTTTTTTTTTT</td>
</tr>
<tr>
<td>4:</td>
<td>feff</td>
<td>0.0018</td>
<td>6782</td>
<td>0.946</td>
<td>0.938 0.942  TTTTTTTTTTTTTTTT</td>
</tr>
<tr>
<td>5:</td>
<td>ff7f</td>
<td>0.0018</td>
<td>6778</td>
<td>0.949</td>
<td>0.946 0.950  TTTTTTTTTTTTTTTT</td>
</tr>
<tr>
<td>6:</td>
<td>fdff</td>
<td>0.0018</td>
<td>6738</td>
<td>0.947</td>
<td>0.941 0.946  TTTTTTTTTTTTTTTT</td>
</tr>
<tr>
<td>7:</td>
<td>1</td>
<td>0.0018</td>
<td>6690</td>
<td>0.955</td>
<td>0.945 0.942  TNNNNNNNNNNNNNNN</td>
</tr>
<tr>
<td>8:</td>
<td>fffd</td>
<td>0.0018</td>
<td>6667</td>
<td>0.968</td>
<td>0.966 0.967  TTTTTTTTTTTTTTTT</td>
</tr>
</tbody>
</table>
Phase Change, pat 26, br 48190, 0.8453 0.9040 0.8470 (0.01295)

<table>
<thead>
<tr>
<th>% Patterns</th>
<th># Branches</th>
<th>gshre</th>
<th>local</th>
<th>corr</th>
<th>Local History</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: c000</td>
<td>0.0012</td>
<td>4554</td>
<td>0.653</td>
<td>0.777</td>
<td>0.680</td>
</tr>
<tr>
<td>1: e000</td>
<td>0.0009</td>
<td>3420</td>
<td>0.714</td>
<td>0.859</td>
<td>0.758</td>
</tr>
<tr>
<td>2: f000</td>
<td>0.0008</td>
<td>2942</td>
<td>0.756</td>
<td>0.888</td>
<td>0.788</td>
</tr>
<tr>
<td>3: fffc</td>
<td>0.0008</td>
<td>2878</td>
<td>0.908</td>
<td>0.960</td>
<td>0.959</td>
</tr>
<tr>
<td>4: f800</td>
<td>0.0007</td>
<td>2642</td>
<td>0.786</td>
<td>0.917</td>
<td>0.827</td>
</tr>
<tr>
<td>5: 3</td>
<td>0.0007</td>
<td>2572</td>
<td>0.968</td>
<td>0.952</td>
<td>0.951</td>
</tr>
<tr>
<td>6: fc00</td>
<td>0.0007</td>
<td>2435</td>
<td>0.815</td>
<td>0.933</td>
<td>0.854</td>
</tr>
<tr>
<td>7: fe00</td>
<td>0.0006</td>
<td>2225</td>
<td>0.836</td>
<td>0.936</td>
<td>0.876</td>
</tr>
<tr>
<td>8: ff00</td>
<td>0.0006</td>
<td>2140</td>
<td>0.856</td>
<td>0.947</td>
<td>0.931</td>
</tr>
<tr>
<td>9: ff80</td>
<td>0.0006</td>
<td>2061</td>
<td>0.854</td>
<td>0.941</td>
<td>0.934</td>
</tr>
</tbody>
</table>

One Way, pat 2, br 2617433, 0.9917 0.9934 0.9897 (0.70337)

<table>
<thead>
<tr>
<th>% Patterns</th>
<th># Branches</th>
<th>gshre</th>
<th>local</th>
<th>corr</th>
<th>Local History</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: ffff</td>
<td>0.5151</td>
<td>1916950</td>
<td>0.993</td>
<td>0.996</td>
<td>0.993</td>
</tr>
<tr>
<td>1: 0</td>
<td>0.1882</td>
<td>700483</td>
<td>0.988</td>
<td>0.986</td>
<td>0.982</td>
</tr>
</tbody>
</table>
Steps in BHT Use on a Dynamically Scheduled Proc.

Register \( r1 \) not available until cycle ten\(^1\).

When branch in ID, read BHT and make prediction. (Cycle 1)

(Optional) Backup (checkpoint) register map (if present).

Execute branch in usual way and check prediction. (Cycle 10.)

If prediction correct, update BHT when branch commits (Cycle 11.).

If prediction wrong, start recovery process (does not occur here).

\[ \text{Cycle:} \quad 0 \quad 1 \quad 2 \quad 3 \quad 10 \quad 11 \quad 12 \quad 13 \]
\[ \text{bneq} \ r1, \text{TARGET} \quad \text{IF} \quad \text{ID} \quad 0:RS \quad 0:RS \quad \ldots \quad 0:B \quad 0:WC \]
\[ \text{xor} \ r2, \ r3, \ r4 \quad \text{IF} \quad \text{ID} \quad 5:EX \quad 5:WB \quad 5:C \quad 6:C \]

TARGET:
\[ \text{and} \ r5, \ r6, \ r7 \]

\(^1\) Perhaps due to a cache miss, or maybe it depended on a long-latency floating-point operation, the reason is not important.
BHT use when branch taken, correctly predicted.

Register \texttt{r1} not available until cycle 10.

When branch in ID, compute target, read BHT and make prediction. (Cycle 1).

Execute branch in usual way and check prediction. (Cycle 10.)

Commit branch after \texttt{div}. (Cycle 23.)

\begin{verbatim}
! Predict taken, taken.
Cycle: 0 1 2 3 10 11 \ldots 21 22 23
\texttt{div f0,f2, f4} ID DIV
\texttt{bneq r1, TARGET} IF ID 0:RS 0:RS \ldots 0:B 0:WB C
\texttt{xor r2, r3, r4} IFx

\ldots

\texttt{TARGET:}
\texttt{and r5, r6, r7} IF... C
\end{verbatim}
BHT use when branch taken, incorrectly predicted, register map not backed up.

Register r1 not available until cycle 10.

When branch in ID, compute target, read BHT and make prediction. (Cycle 1).

Cycle 10: ooops, misprediction. Because register map not backed up, recovery must wait until commit.

Cycle 23: Start recovery: Squash instructions in reorder buffer, start fetching correct path.

! Predict not taken, taken. Register map not backed up.

<table>
<thead>
<tr>
<th>Cycle:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>10</th>
<th>11</th>
<th>...</th>
<th>21</th>
<th>22</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>div f0, f2, f4</td>
<td>ID</td>
<td>DIV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bneq r1, TARGET</td>
<td>IF</td>
<td>ID</td>
<td>0:RS</td>
<td>0:RS</td>
<td>...</td>
<td>0:B</td>
<td>0:WB</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xor r2, r3, r4</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... TARGET:

and r5, r6, r7

IF ....
BHT use when branch taken, incorrectly predicted, register map backed up.

Register \texttt{r1} not available until cycle 10.

When branch in ID, backup (checkpoint) register map, compute target, read BHT and make prediction. (Cycle 1).

Cycle 10: ooops, misprediction. Squash reorder buffer past branch, switch to backed up register map, start fetching correct path.

Cycle 23: Branch commits.

\begin{verbatim}
! Predict not taken, taken. Register map backed up.
Cycle: 0 1 2 3 10 11 ... 21 22 23
div f0,f2, f4  ID  DIV  
bneq r1, TARGET IF  ID  0:RS 0:RS ... 0:B 0:WB  C
xor r2, r3, r4 IF  ID  EX

... TARGET:
and r5, r6, r7 IF ....
\end{verbatim}
(m, n) Two-Level Correlating Predictor

Idea: Base branch decision on . . .
. . . the identity of the branch instruction (as in the one-level scheme) . . .
. . . and the most recent branch outcomes (global history).

Global History:
The outcome of the most recent branches.

In an (m, n) predictor, interested in m most-recent branches.
Global History Example

! Loop always iterates 4 times.
! Branch below never taken.
  bneq r2, SKIP   N
  add f0, f0, f2
SKIP:
  addi r1, r0, #4
LOOPE:
  multd f0, f0, f2
  subi r1, r1, #1
  bneq r1, LOOP   T T T T N ... T T T T N ...
! Cycle 10 20 30 40 50 110 120 130 140 150
!
! Global History (m=4), X: depends on earlier branches.
! 10 XXXN Human would predict taken.
! 20 XXNT Human would predict taken.
! 30 XNTT Human would predict taken.
! 40 NTTT Human would predict not taken.
! 50 TTTN
BHT Indexing for \((m, n)\) Predictor

Two methods of generating address for BHT:

- **gselect**: Concatenate global history with branch address.

- **gshare**: Exclusive-or global history with branch address.

gselect is easier to understand, but gshare uses BHT more efficiently.
Global history must be accurate.

Why that’s a problem:

! First branch: Predict not taken, taken. Register map backed up.
Cycle: 0 1 2 3 10 11 12 13 ... 21 22 23
  div f0, f2, f4 ID DIV DIV WC
  bneq r1, TARGET IF ID 0:RS 0:RS ... 0:B 0:WB C
  beqz r2, SKIP IF ID 1:B ...
  xor r2, r3, r4 IF ID EX ...

...  
TARGET:
  and r5, r6, r7 IF ID EX ...
  beqz r4, LINE1 IF ID ...
Cycle: 0 1 2 3 10 11 12 13 ... 21 22 23

Cycle 2: beqz should see global history with bneq not taken.

Global history includes assumption that bneq not taken.
First branch: Predict not taken, taken. Register map backed up.

Cycle:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>...</th>
<th>21</th>
<th>22</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>div f0, f2, f4</td>
<td>ID</td>
<td>DIV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bneq r1, TARGET</td>
<td>IF</td>
<td>ID</td>
<td>0:RS</td>
<td>0:RS</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>beqz r2, SKIP</td>
<td>IF</td>
<td>ID</td>
<td>1:B</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xor r2, r3, r4</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TARGET:

| and r5, r6, r7 | IF | ID | EX | ... | | | | | | | |
| beqz r4, LINE1 | IF | ID | ... | | | | | | | | |

Cycle: 0 1 2 3 10 11 12 13 ... 21 22 23

Cycle 3: Now global history includes assumption that $\text{bneq}$ and first $\text{beqz}$ not taken.

Cycle 11: Ooops, $\text{bneq}$ misprediction discovered.

Global history has two incorrect assumptions . . .

. . . unless they’re fixed prediction for second $\text{beqz}$ won’t be accurate.

Cycle 12: $\text{beqz}$ should see global history with $\text{bneq}$ taken.
Global History and Dynamic Execution

Global History in Two-Level Predictor with Dynamic Execution

Global history backed up (*checkpointed*) at each branch.

*Predicted* outcome shifted into global history.

If misprediction discovered, global history restored from backup . . .

. . . just as the register map can be.
Target Prediction and Folding

Target Prediction:
Predicting the outcome and target of a branch.

Branch Target Buffer:
A table indexed by branch address holding a predicted target address.

Target Prediction

Put BTB in IF stage.

Use PC to read an entry from BTB.

If valid entry found, replace PC with predicted target.

With target correctly predicted, zero branch delay.
Target Prediction Example

Static scheduled system (for clarity).

<table>
<thead>
<tr>
<th>Cycle:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>bneq r1, TARGET</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>xor r2, r3, r4</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
</tbody>
</table>

TARGET:

and r5, r6, r7 | IF | ID | EX | MEM | WB | IF | X |

Cycle 0

BTB lookup and prediction. Predict taken.

Target from BTB will be clocked into PC.
Target Prediction Example, continued.

Static scheduled system (for clarity).

<table>
<thead>
<tr>
<th>Cycle:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>bneq r1, TARGET</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>xor r2, r3, r4</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
</tbody>
</table>

TARGET:
and r5, r6, r7 | IF | ID | EX | MEM | WB | IF | X

Cycle 1

Start fetching predicted target.

Execute branch instruction (in ID).

Check predicted outcome and predicted target.

Correct predictions, continue execution.
Target Prediction Example, continued.

<table>
<thead>
<tr>
<th>Cycle</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>bneq</td>
<td>r1, TARGET</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
</tr>
<tr>
<td>xor</td>
<td>r2, r3, r4</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td></td>
<td></td>
<td></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
</tr>
</tbody>
</table>

TARGET:

<table>
<thead>
<tr>
<th>and</th>
<th>r5, r6, r7</th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
<th>IF</th>
<th>X</th>
</tr>
</thead>
</table>

Cycle 10

BTB lookup and prediction. Predict taken.

Target from BTB will be clocked into PC.

Cycle 11

Start fetching predicted target.

Execute branch instruction (in ID).

Ooops, incorrect outcome prediction . . .

. . . replace target with nop . . .

. . . and clock correct target into PC.
What BTB predicts for branch instructions:

That instruction will be a CTI.

If CTI is a branch, that branch is taken.

CTI target.

For branches and non-indirect jumps (j, jal)
... predicting target is easy, since target always same.

bneq r1, LOOP     ! Target always PC + 4 + 4 * LOOP
j LINEJ           ! Target always PC + 4 + 4 * LINEJ

For register-indirect jumps (jr, jalr)
... prediction depends on predictable behavior.

jr r1             ! Target is in r1. Can be different each time.
jalr r1           ! Target is in r1. Can be different each time.
Behavior of Register-Indirect Jumps

Predictability depends on how jumps used.

Major Uses

- Procedure Passed as Parameter

  For example, function passed to the C library’s `qsort`.

  These rarely change so target is predictable.

- Case Statements

  These change, and so prediction more difficult.
Target Prediction

Two Methods

Keep a stack of (what appear to be) return addresses. Used for procedure return instructions.

Predict last target. Used for all other instructions.

Predict Last Target

Used for everything except return instructions.

Last time instruction executed target address stored in BTB.

If entry found and predicted taken (for a branch), last target address used.

Effectiveness:

Perfect for non-indirect jumps and branches (if taken).

Reasonably effective on indirect branches.
Predict Return Address

Used for return instruction. (An instruction used for a procedure return, which may not have the mnemonic `return`).

Hardware keeps a stack of return addresses.

BTB stores whether instruction is a return.

When a call instruction encountered push return address on stack.

When BTB identifies instruction as a return target address is popped off stack.

Effectiveness depends on whether call and return instructions can be identified.
Consider code for C switch statement:

! Possible code for a switch statement.
! switch( r2 ) { case 0: foo(); break; case 1: bar(); break; ... }
! Set r1 to base of switch address table.
  lhi   r1, #0x1234
  ori   r1, r1, #0x5670
! Multiply switch index by stride of table (4 bytes per address).
  slli  r3, r2, #2
! Get address of case code address.
  add   r1, r1, r3
! Get case code address.
  lw    r4, 0(r1)
! Jump to case code.
  jr     r4

If r2 rarely changes, jr predictable.
Possible BTB Contents

Target address.

History information (instead of in BHT).

Tag, to detect collisions.