Lecture 6
Branch Prediction

Why do we want to predict branches?

- MIPS based pipeline – 1 instruction issued per cycle, branch hazard of 1 cycle.
  - Delayed branch
- Modern processor and next generation – multiple instructions issued per cycle, more branch hazard cycles will incur.
  - Cost of branch misfetch goes up
  - Pentium Pro – 3 instructions issued per cycle, 12+ cycle misfetch penalty
    - HUGE penalty for a misfetched path following a branch
Branch Prediction

- **Static Prediction**
  - Always taken, always not taken
  - Opcode based
  - Displacement based (forward not taken, backward taken)
  - Compiler directed (branch likely, branch not likely)

- **Dynamic Prediction**
  - 1 bit predictor – remember last taken/not taken per branch
    - Use a branch-prediction buffer or branch-history table with 1 bit entry
    - Use part of the PC (low-order bits) to index buffer/table – Why?
      - Multiple branches may share the same bit
    - Invert the bit if the prediction is wrong
    - Backward branches for loops will be mispredicted twice
      -- once upon loop exit and then again on loop entry

2-bit Branch Prediction

- Has 4 states instead of 2
- A prediction must miss twice before it is changed
- Good for backward branches of loops
Branch History Table

- Has limited size
- 2 bits by N (e.g. 4K entries)
- Uses low-order bits of branch PC to choose entry
- Plot misprediction instead of prediction

![Branch History Table Diagram]

Observations

- Prediction Accuracy ranges from 99% to 82% or a misprediction rate of 1% to 18%
- Misprediction for integer programs (gcc, espresso, eqntott, li) is substantially higher than FP programs (nasa7, matrix300, tomcatv, doduc, spice, fppp)
- Branch penalty involves both misprediction rate and branch frequency, and is higher for integer benchmarks
- Prediction accuracy improves with buffer size, but does not improve beyond 4K entries
Correlating or Two-level Predictors

- Correlating branch predictors also look at other branches for clues. Consider the following example.

```c
if (aa==2) -- branch b1
    aa = 0;
if (bb==2) --- branch b2
    bb = 0;
if(aa!=bb) { ... --- branch b3 – Clearly depends on the results of b1 and b2

Prediction if the last branch is NT

Prediction if the last branch is T

(1,1) predictor – uses history of 1 branch and uses a 1-bit predictor
```

Another Example

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

Code Sequence assuming d is assigned to R1:

```c
BNEZ R1, L1 ; branch b1 (d=0)
DADDU R1,R0,#1 ; d==0, so d=1
L1: DADDIU R3,R1,#-1
       BNEZ  R3,L2 ; branch b2 (d=1)
BNEZ  R3,L2 ; branch b2 (d=1)
```

Possible Execution Sequence for the code fragment:

<table>
<thead>
<tr>
<th>Initial d</th>
<th>d==0?</th>
<th>B1</th>
<th>d before b2</th>
<th>d==1?</th>
<th>b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>yes</td>
<td>not taken</td>
<td>1</td>
<td>yes</td>
<td>not taken</td>
</tr>
<tr>
<td>1</td>
<td>no</td>
<td>taken</td>
<td>1</td>
<td>yes</td>
<td>not taken</td>
</tr>
<tr>
<td>2</td>
<td>no</td>
<td>taken</td>
<td>2</td>
<td>no</td>
<td>taken</td>
</tr>
</tbody>
</table>

Clearly, if b1 is not taken b2 will not be taken => correlation
Correlating Branch Predictor

- If we use 2 branches as histories, then there are 4 possibilities (T-T, NT-T, NT-NT, NT-T).
- For each possibility, we need to use a predictor (1-bit or 2-bit).
- And this repeats for every branch.

![Diagram of 2-bit per-branch predictors](image)

(2,2) branch prediction

Performance of Correlating Branch Prediction

- With same number of state bits, (2,2) performs better than non-correlating 2-bit predictor.
- Outperforms a 2-bit predictor with infinite number of entries
General (m,n) Branch Predictors

- The *global history register* is an *m*-bit shift register that records the last *m* branches encountered by the processor.
- Usually use both the PC address and the GHR (2-level)

```
PC  Combining function  m-bit ghr  01  n-bit predictors  00
```

Is Branch Predictor Enough?

- When is using branch prediction beneficial?
  - When the outcome is known later than the target

- If we predict the branch as taken, and suppose that is correct, what is the target address?
  - Need a mechanism to provide target address as well

- Can we eliminate the one cycle delay for the 5-stage pipeline?
  - Need to fetch from branch target immediately after branch
Branch Target Buffer (BTB)

BTB is a cache that contains the predicted PC value for taken branches.

Is the current instruction a branch?
• BTB provides the answer before the current instruction is decoded and therefore enables fetching to begin after IF-stage.

What is the branch target?
• BTB provides the branch target if the prediction is a taken direct branch (for not taken branches the target is simply PC+4).

BTB

- 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
BTB operations

- Two things can go wrong
  - BTB miss (misfetch)
  - Mispredicted a branch (mispredict)

- Suppose for branches, BTB hit rate of 85% and predict accuracy of 90%, misfetch penalty of 2 cycles and mispredict penalty of 5 cycles. What is the average branch penalty?
  \[ 2 \times (15\%) + 5 \times (85\% \times 10\%) \]

- BTB and BPT can be used together to perform better prediction – for example 2 bit predictor can be used for branch prediction.

BTB Performance
Integrated Instruction Fetch Unit

Separate out IF from the pipeline and integrate with the following components. So, the pipeline consists of Issue, Read, EX, and WB (scoreboarding); Or Issue, EX and WB stages (Tomasulo).

1. Integrate Branch Prediction – Branch predictor is part of the IFU.

2. Instruction Prefetch – Fetch instruction from instruction memory ahead of PC computation with the help of branch predictor & store in a prefetch buffer.

3. Instruction Memory Access and Buffering - Keep on filling the Instruction Queue independent of the execution => Decoupled Execution.

Branch Prediction Summary

- The better we predict, the lesser penalty we might incur
- 2-bit predictors capture branch behavior well
- Correlating predictors improve accuracy, particularly when combined with 2-bit predictors
- Accurate branch prediction does no good if we don’t know there was a branch to predict
- BTB identifies branches in IF stage
- BTB combined with branch prediction table