Lecture: Branch Prediction

- Topics: branch prediction, bimodal/global/local/tournament predictors, branch target buffer (Section 3.3, notes on class webpage)
Pipeline without Branch Predictor

In the 5-stage pipeline, a branch completes in two cycles →
If the branch went the wrong way, one incorrect instr is fetched →
One stall cycle per incorrect branch
In the 5-stage pipeline, a branch completes in two cycles → If the branch went the wrong way, one incorrect instr is fetched → One stall cycle per incorrect branch
1-Bit Bimodal Prediction

• For each branch, keep track of what happened last time and use that outcome as the prediction

• What are prediction accuracies for branches 1 and 2 below:

```c
while (1) {
    for (i=0; i<10; i++) {                     branch-1
        ...
    }
    for (j=0; j<20; j++) {                     branch-2
        ...
    }
}
```
2-Bit Bimodal Prediction

- For each branch, maintain a 2-bit saturating counter:
  if the branch is taken: counter = min(3, counter + 1)
  if the branch is not taken: counter = max(0, counter - 1)

- If (counter >= 2), predict taken, else predict not taken

- Advantage: a few atypical branches will not influence the prediction (a better measure of “the common case”)

- Especially useful when multiple branches share the same counter (some bits of the branch PC are used to index into the branch predictor)

- Can be easily extended to N-bits (in most processors, N=2)
Bimodal 1-Bit Predictor

The table keeps track of what the branch did last time
Bimodal 2-Bit Predictor

The table keeps track of the common-case outcome for the branch.
Correlating Predictors

• Basic branch prediction: maintain a 2-bit saturating counter for each entry (or use 10 branch PC bits to index into one of 1024 counters) – captures the recent “common case” for each branch

• Can we take advantage of additional information?
  - If a branch recently went 01111, expect 0; if it recently went 11101, expect 1; can we have a separate counter for each case?
  - If the previous branches went 01, expect 0; if the previous branches went 11, expect 1; can we have a separate counter for each case?

Hence, build correlating predictors
Global Predictor

The table keeps track of the common-case outcome for the branch/history combo.
Local Predictor

Branch PC

Use 6 bits of branch PC to index into local history table

Table of 64 entries of 14-bit histories for a single branch

10110111011001

14-bit history indexes into next level

Also a two-level predictor that only uses local histories at the first level

Table of 16K entries of 2-bit saturating counters
Local Predictor

The table keeps track of the common-case outcome for the branch/local-history combo.
Local/Global Predictors

• Instead of maintaining a counter for each branch to capture the common case,

→ Maintain a counter for each branch and surrounding pattern
→ If the surrounding pattern belongs to the branch being predicted, the predictor is referred to as a local predictor
→ If the surrounding pattern includes neighboring branches, the predictor is referred to as a global predictor
Tournament Predictors

- A local predictor might work well for some branches or programs, while a global predictor might work well for others.

- Provide one of each and maintain another predictor to identify which predictor is best for each branch.
Branch Target Prediction

• In addition to predicting the branch direction, we must also predict the branch target address

• Branch PC indexes into a predictor table; indirect branches might be problematic

• Most common indirect branch: return from a procedure – can be easily handled with a stack of return addresses
Problem 1

• What is the storage requirement for a global predictor that uses 3-bit saturating counters and that produces an index by XOR-ing 12 bits of branch PC with 12 bits of global history?
Problem 1

• What is the storage requirement for a global predictor that uses 3-bit saturating counters and that produces an index by XOR-ing 12 bits of branch PC with 12 bits of global history?

The index is 12 bits wide, so the table has $2^{12}$ saturating counters. Each counter is 3 bits wide. So total storage = $3 \times 4096 = 12$ Kb or 1.5 KB
Problem 2

• What is the storage requirement for a tournament predictor that uses the following structures:
  ▪ a “selector” that has 4K entries and 2-bit counters
  ▪ a “global” predictor that XORs 14 bits of branch PC with 14 bits of global history and uses 3-bit counters
  ▪ a “local” predictor that uses an 8-bit index into L1, and produces a 12-bit index into L2 by XOR-ing branch PC and local history. The L2 uses 2-bit counters.
Problem 2

- What is the storage requirement for a tournament predictor that uses the following structures:
  - a “selector” that has 4K entries and 2-bit counters
  - a “global” predictor that XORs 14 bits of branch PC with 14 bits of global history and uses 3-bit counters
  - a “local” predictor that uses an 8-bit index into L1, and produces a 12-bit index into L2 by XOR-ing branch PC and local history. The L2 uses 2-bit counters.

\[
\begin{align*}
\text{Selector} &= 4K \times 2^b = 8 \text{ Kb} \\
\text{Global} &= 3b \times 2^{14} = 48 \text{ Kb} \\
\text{Local} &= (12b \times 2^8) + (2b \times 2^{12}) = 3 \text{ Kb} + 8 \text{ Kb} = 11 \text{ Kb} \\
\text{Total} &= 67 \text{ Kb}
\end{align*}
\]
Problem 3

- For the code snippet below, estimate the steady-state \( b_{pred} \) accuracies for the default PC+4 prediction, the 1-bit bimodal, 2-bit bimodal, global, and local predictors. Assume that the global/local preds use 5-bit histories.

```c
do {
    for (i=0; i<4; i++) {
        increment something
    }
    for (j=0; j<8; j++) {
        increment something
    }
    k++;
} while (k < some large number)
```
Problem 3

• For the code snippet below, estimate the steady-state bpred accuracies for the default PC+4 prediction, the 1-bit bimodal, 2-bit bimodal, global, and local predictors. Assume that the global/local preds use 5-bit histories.

```c
    do {
        for (i=0; i<4; i++) {
            increment something
        }
        for (j=0; j<8; j++) {
            increment something
        }
        k++;
    } while (k < some large number)
```

PC+4: 2/13 = 15%
1b Bim: (2+6+1)/(4+8+1) = 9/13 = 69%
2b Bim: (3+7+1)/13 = 11/13 = 85%
Global: (4+7+1)/13 = 12/13 = 92%
(gets confused by 01111 unless you take branch-PC into account while indexing)
Local: (4+7+1)/13 = 12/13 = 92%
Title

• Bullet