# Programming Assignment #4
# Branch Prediction

CS 3220 / CS 5220 Spring 2024
20 points
due Tuesday, Apr. 16th, 11:59 pm

## 1 Branch Prediction

You'll write a problem to simulate branch prediction using execution traces of actual branch behavior from several programs.

You may work individually or with a partner.

### 1.1 Hardware analysis with Pin

Using a development tool called Pin from Intel, it's possible to analyze a program as it's running on an x86 CPU and to collect information on a wide variety of hardware activities, including branching during instruction execution.

I've created traces of several programs and put them in our class gitlab site. They contain lines having this format:

```
7f2ec86f6dd2 0
7f2ec86f6e48 1
7f2ec86f6ed1 0
7f2ec86f65b7 0
7f2ec86f28dc 1
7f2ec86f2925 1
7f2ec86f65b0 0
7f2ec86f64c7 1
7f2ec86f6e48 0
7f2ec86f64e8 0
7f2ec86f6560 0
7f2ec86f2a2d 0
```

Each line consists of the address of a branch instruction and the result of that branch: taken (1) or not taken (0). So from this output, you can see that the branch instruction at `PC = 7f2ec86f6e48` was taken, and then the next time that branch was hit (seven lines farther down in the output), the branch was not taken.

### 1.2 Simulating branch prediction

Write a program, in the language of your choice, that will read one of these trace files and make a prediction about each branch and then compare the prediction to the result of the actual branch instruction.

You program should have several command-line flags:

- to let the user specify the input trace file
- to let the user specify the number of bits to use: either 0, 1, 2, or 3
- to let the user specify the size of the branch-prediction buffer, in bits (see below)

In real hardware, the branch-prediction buffer (BPB; also called the branch-history table, or BHT) has a fixed number of entries. Each entry is a counter for all of the branch instructions that map to that counter. If $N$, the number of counters, is a power of 2, then the counter for a specific branch instruction is found by ANDing the address of the instruction with the value $N - 1$. With an infinitely large branch-prediction buffer, addresses would never collide (each branch instruction would have its own counter). In reality, the BHT is finite in size. So, there will be collisions, in the same way that there are collisions in the cache.

Instead of requiring $N$ to be a power of 2, We'll allow it to be an arbitrary value, and so the index of the counter for a branch instruction at address $A$ will be given by $A \% N$. And the number of entries in the table will be the total number of bits in the BHT divided by the size (in bits) of each counter.

Tally all of the results across all of the branches and report the total number of branches, the percentage of correct predictions, and the percentage of incorrect predictions.

### 1.2.1 Static BP

This is simple. Predict that a branch will always be not taken. In other words, predict that the taken/not-taken value will always be zero. We can also call this a zero-bit predictor, since it doesn't use any history information to make a prediction.

### 1.2.2 One-bit BP

Use the result of the previous evaluation for each branch. So for the branch at instruction $A$, set $index = A \% N$, and then look at the value `counters[index]`. If the value is 0, and the actual taken/not-taken value is 1, then this is an incorrect prediction. And vice versa. Then set `counters[index]` to the current value (i.e., to 1 if the branch was taken; or to 0 if the branch was not taken), so that you'll be able to make the next prediction.

### 1.2.3 Two-bit BP

Use a two-bit saturating counter. If the value at `counters[index]` is 2 or 3, then predict taken; otherwise predict not taken. If the current branch was taken, then set

```
counters[index] = min(counters[index]+1, 3)
```

If the current branch was not taken, then set

```
counters[index] = max(counters[index]-1, 0)
```

### 1.2.4 Three-bit BP

Use a three-bit saturating counter. Check its value to make a prediction, and then update it.

### 1.2.5 Correlating BP

Graduate students, and undergraduates for a little extra credit: Implement $(1, 1)$, $(1, 2)$, and $(2, 2)$ correlating branch predictors: the value of one or more previous branches (any branch—not necessarily this particular branch) chooses from one of two or more $n$-bit branch predictors for this branch. Since each entry in the BHT now consists of $2^m$ $n$-bit counters, the number of entries will be the size of the BHT (in bits) divided by $n \times 2^m$.

# 2 Developing and Testing

Get the trace files from gitlab. The overall accuracy of the branch prediction across the whole program should improve as you use more sophisticated branch prediction.

Here are my results, using $N = 512$ bits. This shows the percentage of correct predictions:

| test case | #branches | 0-bit | 1-bit | 2-bit | 3-bit |
|---|---|---|---|---|---|
| curl1m | 999986 | 57.71 | 84.47 | 87.50 | 86.99 |
| gcc | 387233 | 62.92 | 87.75 | 89.86 | 89.17 |
| java1m | 1000000 | 58.61 | 88.14 | 91.73 | 91.52 |

And results using $N = 256$ bits:

| test case | #branches | 0-bit | 1-bit | 2-bit | 3-bit |
|---|---|---|---|---|---|
| curl1m | 999986 | 57.71 | 83.91 | 80.92 | 84.48 |
| gcc | 387233 | 62.92 | 87.14 | 88.51 | 88.53 |
| java1m | 1000000 | 58.61 | 87.74 | 87.29 | 89.05 |

## 2.1 Results for graduate students

With a (2, 2) correlating branch predictor, I get these results using $N = 256$ bits:

| test case | #branches | (2, 2) CBP |
|---|---|---|
| curl1m | 999986 | 87.51 |
| gcc | 387233 | 88.08 |
| java1m | 1000000 | 89.23 |

A (2, 2) correlating branch predictor, using $N = 512$ bits:

| test case | #branches | (2, 2) CBP |
|---|---|---|
| curl1m | 999986 | 88.86 |
| gcc | 387233 | 89.66 |
| java1m | 1000000 | 90.92 |

And (2, 2) with $N = 2048$, just to try to drive up the accuracy:

| test case | #branches | (2, 2) CBP |
|---|---|---|
| curl1m | 999986 | 91.17 |
| gcc | 387233 | 91.72 |
| java1m | 1000000 | 93.62 |

# 3 What to Submit

Submit your code and some nicely formatted results. Make a bar chart showing the effectiveness of the various branch predictors. Vary the size of the BHT as well, and show the effect of the size on the prediction accuracy.