McGill University
Department of Electrical and Computer Engineering
Computer Organization and Architecture
ECSE 425 – Winter 2009
Project

Due date: <u>April 3, 2009, 3:35 PM</u>

DETAILED GUIDELINE

The goal of this project is to explore the effectiveness of branch direction prediction (taken vs not taken) on an actual program.
The input to your simulator will be a *trace file*. The trace file is the result of running a real program and reflects the information about the execution of each branch of the program. In the trace file a trace of branches and their outcomes have been presented. An example of a portion of a trace file is given below:

```
4257192 0
4257215 1
…
```

The first entry on each line is the address of a branch instruction and the next entry is a 0 if the branch was not taken and 1 if the branch was taken. The simulator takes as input a trace file, simulates different branch prediction schemes and different predictor sizes and as output produces statistics as to the performance. In all cases, the output should be stored in a file in human readable form.

1. For “Static” branch prediction policies presented below write a program to read in the trace and calculate the mis-prediction rate:
   a. “Always predict taken”
   b. “Always predict not taken”

To analyze the effectiveness of these two schemes you have to answer the following two questions:

- Which of these two policies is more accurate (has few mis-predictions)?
- Based on common programming idioms, what might explain the above result?

2. For dynamic branch prediction policies you have to write a software simulator to model:
   a. Branch-prediction buffer (n-bits predictor)
   b. Correlating predictor ((h,n) correlating predictor)
   c. Tournament predictor

Branch-prediction buffer
The branch-prediction buffer is an array of 2^m n-bit saturating counters. Each counter includes one of 2^n values. For example for 2- bit counter the values are: 11 - strongly taken (T), 10 - weakly taken (t), 01 - weakly not taken (n), and 00 - strongly not taken (N).

To make a prediction, the predictor selects a counter from the table using the lower-order m bits of the instruction's address (its program counter value). The direction prediction is made based on the value of the counter.

After each branch (correctly predicted or not), the hardware increments or decrements the corresponding counter to bias the counter toward the actual branch outcome (the outcome given in the trace file). As these are saturating counters, decrementing a minimum counter or incrementing a maxed out counter should have no impact. Initialize the predictor to “strongly not taken” (00).

To analyze the impact of predictor size on prediction accuracy generate data for 2 bits, 3 bits, 4 bits, 5 bits, ... 20 bits predictors with correspondently 2^2, 2^3, 2^4, 2^5 ... 2^{20} counters. Generate a line plot of the data using MS Excel or some other graphing program. On the y-axis, plot "percentage of branches mis-predicted". On the x-axis plot the log of the predictor size (basically, the number of index bits).

Answer the following questions base on the data you collected:

- What is the best mis-prediction rate obtainable by the n-bit predictor?
- How large must the predictor be to reduce the number of mis-predictions by approximately half as compared to the better of "always taken" and "always not taken"? Give the predictor size both in terms of number of counters as well as bytes.

Correlating predictor

A Correlating predictor is a more advanced dynamic branch predictor that uses the history of recently executed branches (the last h branches) to predict the next branch. It does this by shifting in the most recent conditional branch outcome into the low-order bits of the branch history shift register of h bits. It then hashes the branch history and the PC of the branch when making predictions. To index into the table, a correlating predictor uses the lowest m bits of the program counter and h bits the history shift register. The chosen counter used for prediction is trained just as in a n-bits predictor.

For correlating predictor you have to analyze the impact of predictor size on prediction accuracy as in previous question. Extend your program to simulate (h, n) correlating predictor.

Use your program to simulate a correlating predictor in 8 history bits of varying sizes (the same sizes as in the previous question). Add the received data to the graph you created in the previous question.

Answer the following questions base on the data you collected:

- What is the best mis-prediction rate obtainable by this predictor with 8 history bits?
- At what table size is this predictor generally better than the simple n-bits predictor?
- Explain why the correlating predictor is sometimes more accurate than simple n-bits predictor.

- Explain why n-bits predictor is sometimes more accurate than correlating predictor.

Tournament predictor

We will implement a tournament predictor in the following way: we will use three tables. The first and second tables are just normal n-bits predictor and correlating predictor. The third table is a "chooser" table that predicts whether the n-bits or correlating predictor will be more accurate. The chooser is a table of two-bit saturating counters indexed by the low-order bits of the PC that determines which of the other two table's prediction to return. For the chooser, the two-bit counter encodes: strongly prefer n-bits predictor, weakly prefer n-bits predictor, weakly prefer a correlating predictor, and strong prefer a correlating predictor.

Access the chooser table using the low-order bits of the branch's program counter address. Generate two predictions from both predictors. Based on the result of the lookup in the chooser table, chose the prediction. Initialize the chooser table to strongly prefer n-bits predictor.

Compare the tournament predictor's accuracy versus the data from its two constituent predictors (using the data from the previous question).