Computer Design and Organization

Assignment #2

Due: Wednesday October 23

In this assignment you will compare the performance of various branch predictors and reason about their performance. You will modify the Bliss simulator in order to perform the experiments. You will run experiments using the same Spec benchmark *perl* as in Assignment #1.

You can be in teams of two but choose a different partner from the one you were paired with in Assignment #1 (if any).

Turn in your Assignment following the Report guidelines for the following questions.

1. How is branch prediction implemented in the original Bliss i.e., describe in detail the prediction mechanism and how it is indexed. What is the organization of the Branch Target Buffer? (In the remainder of this assignment you won't have to deal with the BTB.). At this time you should also have the results of Experiments 2 and 3 from Assignment #1. What is the frequency of branches for the application you simulated (Note: Bliss distinguishes between conditional branches
–System. ... . RetiredOperations.branch
for which a prediction is made and indirect branches, including call and returns,
–System. ... . RetiredOperations.ibranch
for which only BPB or return stack predictions are made). Is this consistent with the "conventional wisdom" found, for example, in your textbook? If not, can you conjecture why not?

For simulating the static branch predictors defined below, use the set-up of Experiment 2 of Assignment #1.
For simulating the dynamic branch predictors defined below, use the set-up of Experiments 2 and 3 of Assignment #1.

Before answering the next question:

- Implement a *static* predictor with the policy "Always predict branch not taken" (BNT). Simulate and record the results.
- Implement a *static* predictor with the policy "Predict backward branches to be taken and predict forward branches not taken" (BTFNT). Simulate and record the results.
- Implement a *dynamic* predictor with a table of 1K 2-bit counters indexed by the PC (Note: Bliss instructions are 32-bits long and the 2 rightmost bits of the PC should not be part of the index). Simulate and record the results.
- Implement a *dynamic* predictor with a set of 1K 2-bit counters that are accessed via the PC but as in a direct-mapped cache structure (the default policy is NotTaken in case of a miss). Simulate and record the results.

2. Compare the accuracies of the various branch predictors. As a guideline, you should include at least the following comparisons:

- Compare the results from the two static branch predictors. Are the results consistent with what you expected?
- Compare the results from the two different structures for the 2-bit counters. Discuss. What is the influence of the warm-up?
- Compare the results from the best static branch predictor with the results of the two 2-bit counter schemes. Discuss.
- Compare the results of the two 2-bit counter schemes with that of the GShare. Discuss. In particular since GShare requires much less hardware than the cache-like structure, do you think that GShare is an efficient predictor (for this particular application; don't over generalize).

3. A criticism of the GShare predictor is that there can be too much aliasing, i.e., different branches with different history mapping to the same 2-bit counter. Without increasing the number of 2-bit counters, describe, implement and simulate a scheme that you think could reduce the amount of aliasing. Discuss your results. Of course there is no single solution to this problem and your scheme might not work well for this application but you should give reasons why you think it should work.