ECE4100/6100 Advanced Computer Architecture
Programming Assignment 2
Dynamic Branch Predictors

Assigned: Sept 10, 2010 **Due 12:35pm Oct 1 (Friday), 2010**

TA Signoff: _____

In this assignment, you will study and implement **several dynamic branch predictors** using Simplescalar. We will use **sim-outorder**, the performance simulator, which is much more sophisticated than the sim-safe you used in the first assignment.

Stress Level: Tracing through sim-outorder could cause anxiety and dizziness. If you haven't done so, start this project early.

Problem Statement

This assignment is designed to compare the performance of a few different dynamic branch predictors using Simplescalar by running selective SPEC2000 benchmark applications. For each predictor (to be enumerated below), you have to use the `default.2bc.cfg` provided but change the first few knobs in the file whenever necessary. This default specifies a generic 8-wide microprocessor that performs speculation and out-of-order execution. For each simulation, in addition to the default of `"-fetch:mplat 3"`, you also need to change this particular knob to `"-fetch:mplat 9"` to generate one more data point for each run to mimic the penalty from a deeper pipeline effect.

Two-bit counter (Bi-modal): Use the `default.2bc.cfg` in which a 16384-entry 2-bit counter based predictor is specified in this provided file.

Gshare: Use the `default.2bc.cfg` file and modify the configuration to simulate a Gshare predictor with the following configuration: a 12-bit first-level BHR, a 4096-entry PHT as the second level, along with a 4-way 4096-entry BTB.

Hybrid: You will modify the `default.2bc.cfg` file again to enable an Alpha 21264 style hybrid predictor. (There is no need to modify the source code, but you have to trace the code to get understandings of what to specify.) Due to the implementation used in Simplescalar, you cannot specify an exactly the same 21264 tournament predictor. Instead, you will configure your predictor based on the schematic depicted in Figure 1. The differences are the following: first, it uses 2 bit counter arrays for all counter tables, whereas 21264 used 3-bit counters in its local PHT. Second, the local predictor, global predictor, and meta predictor in this assignment use some PC bits to index, while a global history was used in 21264. If you trace the code, you will find that Simplescalar indexes the local table using [PC<<2] and both the global predictor and meta predictor with some kind of hash function. (The hash function is defined in a macro called `BIMOD_HASH` in `bpred.c`)

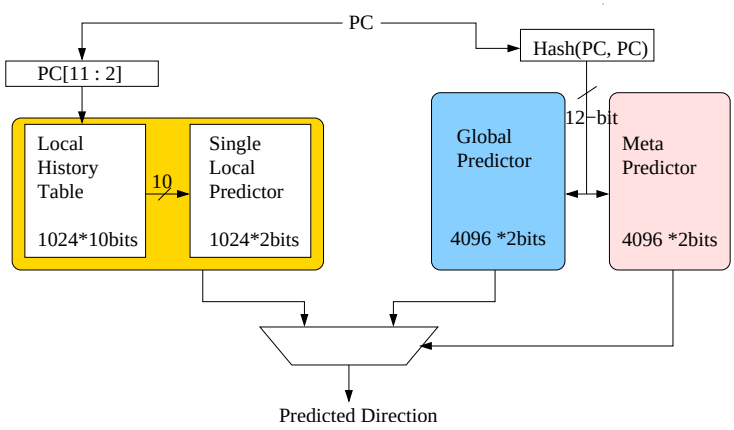

Figure 1. Revised Tournament Branch Predictor

Perceptron: In this part, you will first read the **paper** provided in T-square ─ *Dynamic Branch Prediction with Perceptrons* by Jimenez and Lin in HPCA-7 to understand the single-layer perceptron branch predictor. Then you have to implement this predictor in `sim-outorder.c` and `bpred.[c,h]`. Figure 2 illustrates the microarchitectural schematic. In this predictor, we follow the suggestion in the paper, using (1.93*BHR length +14) as the threshold value (θ in the paper). Initialize all the weights (w) to be 0 at the beginning. Confine the weight in between [-128, 127] during the runtime learning, i.e., each weight is represented by 8 bits. Assume a configuration **(K, n)** represents a predictor in which a **K-entry Weight Table** is used (using (PC mod K) to index) inside the perceptron predictor with an **n-bit Global BHR**. In your simulations, you need to simulate the following two configurations. To use similar hardware, the first configuration is set to (K, n) = (128, 27). Then, simulate another larger configuration of (K, n) = (256, 54).

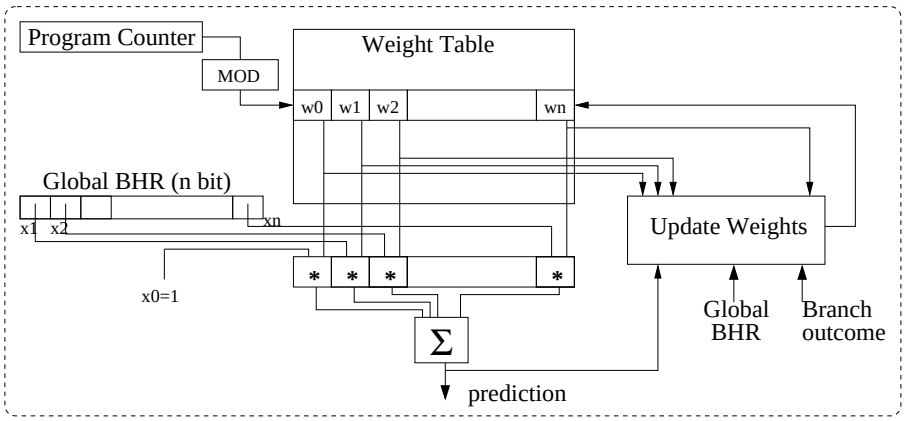

Figure 2. A Perceptron Predictor

Perfect: Use `default.2bc.cfg` provided but simply change the predictor type to be perfect. The results from this predictor are used upper bound cases. Notice that, the knob –`fetch:mplat` is insensitive in this case. Hence you do not need to run for two different penalty latencies.

Simulation you need to carry out.
The same SPEC benchmark programs provided in Programming Assignment 1 are to be simulated in this assignment. There are 6 integer benchmark programs and 5 floating point benchmark programs in the 2 gnu-zipped files. Please simulate **200 millions** instructions for each benchmark program. In plotting your result, please show the performance "**sim_IPC**" and the direction prediction rate ".**bpred_dir_rate**" and perform certain level of analysis on these results you collect. Note that there is no guarantee one branch predictor will always outperform the other for all benchmark programs.

Programming Tips.

- `sim-outorder` is much more complex than `sim-safe`. It models the microarchitecture in reasonable details to report the performance results of the benchmark program simulated, e.g., IPC, cache miss rates, branch misprediction rates, etc. For this project, it is not necessary for you to understand the entire simulator. Most of your modifications for the **Perceptron** predictor should be in `bpred.c`, `bpred.h` and `sim-outorder.c`, in particular, the `bpred_update` and `bpred_lookup` functions. You also need to register new knobs to add your perceptron predictor as **a new knob** for the simulator. Depending on how avid you are as a programmer, this project may take much more time than the first one.
- Please take the simulation hours into account when working on this project. Depending on the Linux machine you use, to execute 200 million instructions in `sim-outorder` may take 10 minutes to hours. Given the large number of simulations you have to launch to complete this project, you want to start as soon as you can. My suggestion is: run the **Perfect, Bi-modal**, and **Gshare** cases immediately, they do not need any code modification. The **Hybrid** (also requiring no code modification) can be launched once you figure out how to change and specify the configuration file, which is not difficult to do. The perceptron predictor requiring new code in the simulator can be run later once you finish implementing and have your code verified.
- To help you debug your code, you definitely need the **`alphaobjdump`** provided in the first assignment for you to correlate the PC and its corresponding branch instruction.
- The total number of simulation instances = 11 applications * 2 penalty cases * (1 Bimodal + 1 Gshare + 1 Hybrid + 2 Perceptrons) + 11 applications * 1 Perfect = 121 simulations needed. Remember, they take time to simulate.

How to receive credit? Similar to your first assignment, you need TA's check-off. Similar to the first assignment, you need to create a short report to summarize your results of all predictors (i.e., IPC and direction-prediction rate) and analyze your observation from your experiments.

Honor Code. This assignment must be done **individually**. For those who violate the rule, both the originator and the copier will receive zero credit and will be **immediately** reported to the Dean of Students' Affair for further action.