**CS-320 – Spring 2016: Project 1**

**Submission Due on Blackboard: Friday, March 18th at 9pm**

**Project demos: during lab sessions on Tuesday, March 22nd**

The goal of this project is to measure the effectiveness of various branch direction prediction ("taken" or "non-taken") schemes on several traces of conditional branch instructions. Each trace contains a variable number of branch instructions, and for each branch, the program counter (PC, word address is given) and the actual outcome of the branch are recorded in a trace line. Several trace files are provided for evaluating your predictor designs.

Your goal is to write a program in C or C++ that would use these traces to measure the accuracy of various branch predictors that we studied in class. The branch outcomes from the trace file should be used to train your predictors. Specifically, the following predictors have to be implemented:

1) **[5%] Always Taken.**

2) **[5%] Always Non-Taken**.

3) **[20%] Bimodal Predictor with a single bit of history** stored in each predictor entry. Determine the prediction accuracy of this predictor for the table size of 16, 32, 128, 256, 512, 1024 and 2048 entries. Present results for all traces separately, and the averages across all traces (details of what exactly needs to be submitted are provided below). Assume that the initial state of all prediction counters is "Taken" (T)

4) **[20%] Bimodal Predictor with 2-bit saturating counters** stored in each predictor entry. Repeat the same experiments as in part (3) above. Assume that the initial state of all prediction counters is "Strongly Taken" (TT)

5) **[25%] Gshare predictor**, where the PC is XOR-ed with the global history bits to generate the index into the predictor table. Fix the table size at 2048 entries and determine the prediction accuracy as a function of the number of bits in the global history register. Vary the history length from 3 bits to 11 bits in 1-bit increments. Assume that the initial state of all prediction counters is "Strongly Taken" (TT). The global history register should be initialized to contain all zeroes (where 0=NT and 1=T). The global history register should be maintained such that the least significant bit of the register represents the result of the most recent branch, and the most significant bit of the register represents the result of the least recent branch in the history.

6) **[25%] Tournament Predictor.** The tournament predictor selects between gshare and bimodal predictor for every branch. Configure gshare with 2048-entry table and 11 bits of global history, and configure bimodal predictor with 2048-entry table. Furthermore, configure the selector table with 2048 entries and use the same index as you use for bimodal predictor to index into the selector table (that is, the PC). For each entry in the selector, the two-bit counter encodes the following states: 00 – prefer Gshare, 01 – weakly prefer Gshare, 10 – weakly prefer Bimodal, 11 – prefer bimodal. If the two predictors provide the same prediction, then the corresponding selector counter remains the same. If one of the predictors is correct and the other one is wrong, then the selector's counter is decremented or incremented to move towards the predictor that was correct. Initialize all the component predictors to "Strongly Taken" and initialize the selector's counters to "Prefer Gshare".

**Materials on Blackboard:**

There is a tar/gzipped archive of materials on Blackboard that contains the following:

1.) A sample output file called sample_output.txt complete with comments

2.) A directory called examples, containing short snippets of code that show how to read the input

3.) A directory called traces, containing the following 6 trace files:

long_trace1.txt (23532921 branch instructions)

long_trace2.txt (27946011 branch instructions)

long_trace3.txt (14796021 branch instructions)

short_trace1.txt (3771697 branch instructions)

short_trace2.txt (2866495 branch instructions)

short_trace3.txt (2229289 branch instructions)

4.) A directory called correct_outputs, containing the correct outputs for the given traces. These can and should be used to check that your program works correctly and outputs the results in the required format.

To access these materials, download a copy from Blackboard, cd into the directory where you placed the tar/gzipped archive and issue the following command:

tar -xzvf project1.tar.gz

This will create a new directory (named project1) containing the files mentioned above.

**Submission requirements:**

**NOTE: Please carefully read and follow all directions while preparing your submissions. Submissions will be graded using a script and failure to follow these directions will likely end up crashing the script causing the TAs to have to grade your submission by hand. If this happens points will be deducted from your grade. Examples of things to watch out for: the directory inside your tar archive isn't named with your BU-ID, incorrectly named executable after 'make', using standard input and output for I/O, and not using command-line arguments for the names of the input and output files.**

You need to submit your source code, so that we can compile it and test for correctness. For checking your code, we will be using the same traces that you used for generating your results, plus some more traces that you will not have access to.

The code that you submit should compile into a single executable called **predictors** with a simple `make` command. This executable should run all of the predictors on the given trace, which will be specified via command line options as follows:

./predictors input_trace.txt output.txt

Where: -input_trace.txt – file containing branch trace

-output.txt – file to place output statistics

The output file should have the following format: (an example text file is on Blackboard too with comments, which should not be output by your program)

#,@;

#,@;

#,@; #,@; #,@; #,@; #,@; #,@; #,@;

#,@; #,@; #,@; #,@; #,@; #,@; #,@;

#,@; #,@; #,@; #,@; #,@; #,@; #,@; #,@; #,@;

#,@;

Where each # corresponds to the number of correct predictions made by each of the predictors and @ corresponds to the number of branches. First line provides the number of correct predictions for the always taken predictor, second line for always not taken, third line gives the correct predictions for all seven variations of the bimodal predictor with a single bit of history, fourth line repeats the third line for the two bit saturating counter based bimodal predictor, and the fifth line shows the number of correct predictions for the nine variations of Gshare predictor. Finally the last line is for the tournament predictor. The number of correct predictions and branches should be separated by a comma and every configuration of predictor should be separated by a semicolon(;) and a space.

Submissions will be checked using a script that will compare your output file to the correct output file using the UNIX `diff` tool, so if your output does not **EXACTLY** match the correct output the grading program will mark it as wrong. The TA will have to check such submissions by hand which will result in at least a few points being deducted.

**The blackboard submissions are due on Friday, March 18th at 9pm.**

Another requirement is to present your project to Jesse or Daniel either in the lab sessions on March 22nd, or during Jesse's office hours that week. A separate schedule for that will be created soon, so you can sign up for a slot.

**Submission Rules:**

You must submit all of the following:

1.) All source code

2.) A Makefile

3.) A README, which minimally contains your Name, BU-ID (everything before the @ in your Binghamton University e-mail), B Number, and whether or not you did the extra credit (explained below). Other things to include might be: what works/what doesn't, things you found interesting, etc.

These materials should be turned in as follows: (using Jesse's name and BU-ID as an example)

Jesse's e-mail is jelwell1@binghamton.edu so his BU-ID is jelwell1

1.) Create a new directory whose name is your BU-ID:

mkdir jelwell1/

2.) Copy all relevant files into this new directory (please do not include any .o files, executables, **or copies of the traces**)

3.) Create a tar/gzipped archive whose name is also your BU-ID from the directory as follows:

tar -czvf jelwell1.tar.gz jelwell1/

(This should output name of all archived files, **make sure there are no .o files, executables or trace files in this list before submission**)

4.) Submit tar/gzipped archive via Blackboard

**Extra Credit:**

**[up to 20%] Beating the tournament predictor.** The tournament predictor described above should provide the best accuracy compared to all other predictors that you implemented in this project. As specified in the question above, the tournament predictor requires 12144 bits to implement it (three 2048-entry tables, with each entry being two bits). To receive this extra credit, come up with a design of a branch predictor that beats the tournament predictor **without increasing the number of bits needed to implement the prediction logic**. Your solution should be general (i.e. you cannot somehow hard-code the optimal solution only for the traces given) and should provide benefits on at least some of the provided traces. Note that simply changing the initial state of the existing predictors (even if that increases accuracy slightly) will not give you the extra credit. You should describe your new design in detail in the project report document and explain the intuition behind your approach. Don't be afraid to innovate and test your ideas – we will account for both ideas and results in determining the extra credit grade. Feel free to read some related research papers to get inspired with ideas. **Note**: If you decide to do the extra credit, when submitting your code **do not print any output from this predictor**. However, be sure to note that you completed the extra credit in your README file. Additionally, your README should contain details of the design that you implemented including an analysis of how many bits the design requires to implement.

**Clarifications**

This is a list of things that may or may not be explicitly mentioned above, that you should watch out for:

- Both bimodal predictors do not require you to test with a table size of 64.

- Gshare's global history register should be maintained as follows: the least significant bit of the register should represent the result of the most recent branch, while the most significant bit represents the least recent branch. This has been added to its description above as well.

- Gshare predictor should use modulo operator before XOR when selecting an index from the table.

- The tournament predictor should use a 2-bit bimodal predictor and a Gshare predictor.