Advanced Computer Architecture CMSC 611
Homework 4

Due in class at 1.05pm, Nov 7th, 2012

(For Part B, you could submit an electronic file containing the output of your simulations. If you wish to **go green,** then you can submit the entire Homework electronically as well. Make sure you include the string "CMSC 611 Homework" in your subject line. Deadline remains the same) Please **DO NOT** email your homework to Dr. Olano!! **DO NOT** include him in the CC either!! There is a strong chance it won't be graded if you do!! **Send it only to <abhay1@umbc.edu>**

PART A

1) Branch Prediction! (35 points)
   We have a piece of code with three static branch instructions B1, B2 and B3 and a co-relating branch predictor. The execution of these branches which forms the global history is as shown in the table below.

Branch instruction	B1	B2	B1	B2	B1	B2	B3	B1	B2	B1	B2	B3	B1
Direction	T	N	T	N	T	T	T	T	N	T	T	N	N

T stands for branch Taken and N stands for branch Not Taken. This table indicates the sequence of branch instructions and their corresponding directions during the execution (The actual direction). You can assume that the predictor's initial state before execution, predicts not taken (NT) for all branches.

Fill the table below assuming a (2, 2) co-relating predictor uses only the **LOCAL** history to predict the direction for branch **B1**.

(2, 2) Predictor using **local** history for B1							
B1 invocation number	History used for prediction	Prediction for B1	Actual Direction for B1	Predictor State			
				N, N	N, T	T, N	T, T
1	N, N	N	T	N	N	N	N
2	N, T	N	T	N	N	N	N
3	T, T	N	T	N	N	N	N
4	T, T	N	T	N	N	N	T
5	T, T	T	T	N	N	N	T
6	T, T	T	N	N	N	N	T

Fill the table below assuming a (2, 1) co-relating predictor uses only the **LOCAL** history to predict the direction for branch **B1**.

(2, 1) Predictor using **local** history for B1							
B1 invocation number	History used for prediction	Prediction for B1	Actual Direction for B1	Predictor State			
				N, N	N, T	T, N	T, T
1	N, N	N	T	T	N	N	N
2	N, T	N	T	T	T	N	N
3	T, T	N	T	T	T	N	T
4	T, T	T	T	T	T	N	T
5	T, T	T	T	T	T	N	T
6	T, T	T	N	T	T	N	N

Fill the table below assuming the (2, 2) co-relating predictor uses only the **GLOBAL** history to predict the direction for branch **B1**.

(2, 2) Predictor using **global** history for B1							
B1 invocation number	History used for prediction	Prediction for B1	Actual Direction for B1	Predictor State			
				N, N	N, T	T, N	T, T
1	N, N	N	T	N	N	N	N
2	T, N	N	T	N	N	N	N
3	T, N	N	T	N	N	T	N
4	T, T	N	T	N	N	T	N
5	T, N	T	T	N	N	T	N
6	T, N	T	N	N	N	T	N

Fill the table below assuming the (2, 1) co-relating predictor uses only the **GLOBAL** history to predict the direction for branch **B1**.

(2, 1) Predictor using **global** history for B1							
B1 invocation number	History used for prediction	Prediction for B1	Actual Direction for B1	Predictor State			
				N, N	N, T	T, N	T, T
1	N, N	N	T	T	N	N	N
2	T, N	N	T	T	N	T	N
3	T, N	T	T	T	N	T	N
4	T, T	N	T	T	N	T	T
5	T, N	T	T	T	N	T	T
6	T, N	T	N	T	N	N	T

2) Given (15 points)

- Cache Size = 256KB
- Block Size = 32 Bytes
- Address size = 32 bits

Calculate the size of the tag, offset and index fields and **explain** very briefly how you computed your values, for a

a) Direct mapped cache –

Cache Size/Block size = total lines i.e 256KB/32 Bytes = 8192 lines which is 2^{13}. So Index is 13 to address 8192 lines since it is direct mapped. We have 32 byte block size, so we need 5 bits as offset since $32 = 2^5$. The remaining bits = 32-13-5 = 14 = tag.

b) 8-way associative cache

There are 8 blocks in each set. So total number of sets = 8192/8 = 1024 sets which needs index of 10 bits. Offset = 5 bits as before. Tag = 17 bits.

c) Fully associative cache

No bits for index needed since there is only one set. Offset = 5 and tag = 27

PART B – Introduction to SimpleScalar Simulator

1) Simulating a test program. (10 points)

   There are a few benchmarks available and for our purposes we will use a test program called `test-math` located at

   `~olano/simplesim-3.0/tests-pisa/bin.little/`

   This is the little endian PISA binary.

   Run this benchmark and answer the following questions.

   a) Total Number of instructions executed = 213703

   b) Total number of loads and stores executed = 56899

   c) Program text (code) size in bytes = 91774 bytes

   d) Total first level page table misses = 34

   e) Total page table accesses = 1546179

2) Branch Prediction can be simulated using `sim-bpred`. (40 points)

   Run the program without arguments. Read and understand the arguments they take.

   Answers vary.
   For the 1st benchmark, bimod is almost just as good as the 2-level predictor but not so much for the second benchmark.
   The increase in history size reduces the number of missed in 2-level predictor which is sharp at first and later it slowly plateaus.
   (Full points not given if you have simply said which is best based on the number of misses)