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							
3							
4							
5							
6							

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							
3							
4							
5							
6							

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							
3							
4							
5							
6							

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							
3							
4							
5							
6							

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
b) 8-way associative cache
c) Fully associative cache

PART B – Introduction to SimpleScalar Simulator

Simplescalar is a suite of programs that simulate the execution of programs compiled using a MIPS-like instruction set called PISA. You can simulate the execution of any program using Simplescalar by simply re-compiling the program using a version of gcc that knows how to generate PISA instructions as well as x86 instructions.

Please refer http://www.simplescalar.com/ to get a brief introduction. The simulator has the following modules (taken from the tutorial slides)
• sim-safe.c - minimal functional simulator
• sim-fast.c - faster (and twisted) version of sim-safe
• sim-eio.c - EIO trace and checkpoint generator
• sim-profile.c - profiling simulator
• sim-cache.c - two-level cache simulator (no timing)
• sim-cheetah.c - Cheetah single-pass multiple configuration cache simulator
• sim-bpred.c - branch predictor simulator (no timing)
• sim-outorder.c - detailed OoO issue performance simulator (with timing)

Running any exe without any arguments will document the list of arguments it can take. The SimpleScalar simulator is located at `~olano/simplesim-3.0/`

To start off, log into <linux.gl.umbc.edu> machine using your account information. You can navigate to the above directory by typing

```
cd ~olano/simplesim-3.0/
```

and type in

`./sim-`*`foo`* where foo can be *safe*, *fast*, *cache* etc.

This will show the arguments they take. (I have pasted the output of running sim-safe without any arguments

```
linux3[270]% ~olano/simplesim-3.0/sim-safe
sim-safe: SimpleScalar/PISA Tool Set version 3.0 of August, 2003.
Copyright (c) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar, LLC.
All Rights Reserved. This version of SimpleScalar is licensed for academic
non-commercial use.  No portion of this work may be used by any commercial
entity, or for any commercial purpose, without the prior written permission
of SimpleScalar, LLC (info@simplescalar.com).

Usage: /afs/umbc.edu/users/o/l/olano/home/simplesim-3.0/sim-safe {-options}
executable {arguments}

sim-safe: This simulator implements a functional simulator.  This
functional simulator is the simplest, most user-friendly simulator in the
```

```
simplescalar tool set.  Unlike sim-fast, this functional simulator checks
for all instruction errors, and the implementation is crafted for clarity
rather than speed.

#
# -option        <args>           #    <default> # description
#
-config          <string>         #       <null> # load configuration from a
file
-dumpconfig      <string>         #       <null> # dump configuration to a
file
-h               <true|false>     #        false # print help message
-v               <true|false>     #        false # verbose operation
-d               <true|false>     #        false # enable debug message
-i               <true|false>     #        false # start in Dlite debugger
-seed            <int>            #            1 # random number generator
seed (0 for timer seed)
-q               <true|false>     #        false # initialize and terminate
immediately
-chkpt           <string>         #       <null> # restore EIO trace
execution from <fname>
-redir:sim       <string>         #       <null> # redirect simulator output
to file (non-interactive only)
-redir:prog      <string>         #       <null> # redirect simulated program
output to file
-nice            <int>            #            0 # simulator scheduling
priority
-max:inst        <uint>           #            0 # maximum number of inst's
to execute
```

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

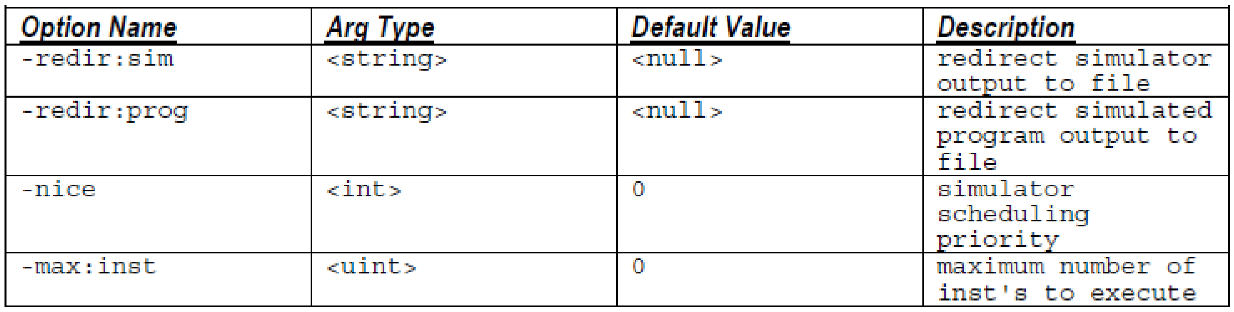
   `~olano/simplesim-3.0/tests-pisa/bin.little/`

   This is the little endian PISA binary.

   Run this benchmark and answer the following questions.

   a) Total Number of instructions executed
   b) Total number of loads and stores executed
   c) Program text (code) size in bytes
   d) Total first level page table misses
   e) Total page table accesses

(If you wish to redirect the output, you can use redir flags as shown below



Option Name	Arg Type	Default Value	Description
-redir:sim	<string>	<null>	redirect simulator output to file
-redir:prog	<string>	<null>	redirect simulated program output to file
-nice	<int>	0	simulator scheduling priority
-max:inst	<uint>	0	maximum number of inst's to execute

So a command would look like

```
sim-foo        -redir:sim        simOutputFilePath        -redir:prog
progOutputFilepath  benchmarkProgramPath
```

2) Branch Prediction can be simulated using `sim-bpred`. (40 points)
Run the program without arguments. Read and understand the arguments they take.

a) (20 points)
Now run the simulator with the `test-math` benchmark using the 4 branch predictors learnt in class namely, taken, not taken, bi modal FSM and the 2 level predictor. The 2-level predictor can further take arguments about the size of history it maintains.
   - Using the values the simulator dumps out as evidence, explain how each predictor performed and compare them.
   - Vary the different parameters the 2-level predictor takes and comment on how this affected the performance of the predictor.

b) (20 points)
Now **repeat** what you did for the above problem with a different benchmark `anagram`.
This benchmark program finds all the anagrams of a given input file. For this problem both the dictionary and the input file is provided. A sim-safe simulation command is shown below

```
cd       ~olano/simplesim-3.0
./sim-safe      tests-pisa/bin.little/anagram     tests-pisa/inputs/words     <     tests-
pisa/inputs/input.txt
```

Here the file tests-pisa/inputs/words is the dictionary and the input file is the latter.

NO ESSAY TYPE ANSWERS PLEASE!!
Be specific about the statistic you are using and what you infer from it. Generic and vague answers will not fetch you full points.