CSc 656 Project 1 Part b

(This document available on iLearn.)
due Monday 3/27/2017 3:00 pm (no grace period)
(4% of your grade)

This is an individual assignment. Work on your own!

Problem 4:

You are given two trace files (li1.trace and li2.trace) on unixlab.sfsu.edu in

~whsu/csc656/Traces/S17/P1

(These are in the unixlab file system. You can log on to unixlab.sfsu.edu, and access them through Unix file system commands. The directory is also linked to a web-accessible area: http://unixlab.sfsu.edu/~whsu/csc656/Traces/S17/P1)

Write two branch prediction simulators to work with these trace files. You may work in C/C++ or Java; other languages will require prior approval.

(There's also a third, very short trace file, test.trace, for preliminary debugging only.)

Each line of a trace file contains four hexadecimal integers, representing information from branches extracted from an instruction trace. The four (hexadecimal) integers on each line are:

the address (i.e., PC) of the branch
the *type* of the branch (1 means direct branch, meaning either a conditional or unconditional branch similar to a MIPS I-format branch; 2 means indirect branch, meaning a branch where the target address comes from a register, similar to the MIPS jr instruction)
the target address
whether the branch is taken (1) or not taken (0)

(These are SPARC traces, so a very small number of indirect branches are actually conditional.) For this project, your simulator will only work with conditional branches (type 1). **Discard all branches that are not of type 1.**

You will develop two branch prediction simulators, System 1 and System 2. **Your code must compile and run on unixlab.sfsu.edu.** Sample executables for these two systems, named sys1 and sys2, can be found on unixlab.sfsu.edu in ~whsu/csc656/CODE/S17/P1. Your programs will simulate the behavior of these branch predictors:

System 1 (static prediction):

Forward branches are predicted not taken, backward branches predicted taken. This is not a hardware branch predictor; you can get the misprediction statistics from scanning the list of conditional branches in each trace file. So if a forward branch is not taken, it is predicted correctly, otherwise it is mispredicted. Similarly, if a backward branch is taken, it is predicted correctly, otherwise it is mispredicted.

System 1 will take Unix command line arguments. The first argument is the name of the trace file. The second argument (optional) is "-v", which turns on verbose mode. See **Submissions** section for format.

System 1 must work correctly for System 2 code to be graded. If your System 1 code does not work, don't even bother working on System 2.

System 2 (basic 2-bit predictor with branch target buffer):

This is the N-entry 2-bit predictor/M-entry branch target buffer from Chapter 4a slides, in the section Branch Handling 2: dynamic branch prediction. Your code should work with any N and M, both powers of 2. Follow the definitions from the slides on how to index the array of 2-bit branch predictors, and how to update the states of each 2-bit predictor. Assume all predictors start in state 01. Remember that predictors are not tagged; different conditional branches can update the same predictor without reinitializing to state 01. The branch target buffer entries are tagged, of course.

System 2 operation in more detail:

```
For each branch $B_i$
    Get prediction for $B_i$ (predict taken or predict not taken)
    If $B_i$ is predicted taken
        Check BTB entry for $B_i$
        If valid bit of BTB entry == 1 && tag of BTB entry == tag of $B_i$
            We have a BTB hit
        Else
            We have a BTB miss
    Else if $B_i$ is predicted not taken
        Do nothing

    Check whether $B_i$ is actually taken/not taken
    If prediction == actual behavior, prediction is correct
    Else prediction is wrong

    update prediction for $B_i$
    If $B_i$ actually taken
        Tag of BTB entry = tag of $B_i$
        Valid bit of BTB entry = 1
```

System 2 will take Unix command line arguments. The first argument is the name of the trace file. The second argument is N (number of entries in predictor buffer, always a power of two), the third argument is M (number of entries in branch target buffer, always a power of two). The fourth argument (optional) is “-v”, which turns on verbose mode. See **Submissions** section for format.

Simulator output

Sys1 should print

- the total number of conditional branches
- the number of forward branches
- the number of backward branches
- the number of forward taken branches
- the number of backward taken branches
- the number of mispredicted branches
- the misprediction rate for all branches (# mispredictions / # branches)

Sys2 should print all of the above, plus:

- the number of BTB misses
- the BTB miss rate (# BTB misses / # BTB accesses)

Your simulator should implement a verbose mode for debugging. Verbose mode is turned off by default, and turned on by the –v flag (see Submissions section for format). When verbose mode is on, in addition to the output generated in non-verbose mode, your simulator prints out, for each branch in the trace file, a list of integers on a single line. These are:

- Order of type 1 branch in trace file (the first type 1 branch in the file is numbered 0)
- Index of prediction buffer accessed (in hexadecimal)
- Current state of prediction buffer
- New state of prediction buffer after update
- Index of BTB accessed (in hexadecimal)
- Tag of BTB entry accessed (in hexadecimal)
- Number of BTB accesses so far
- Number of BTB misses so far

For example, the test trace *test.trace* contains:

```
18244 1 18338 1
1838c 1 18338 1
1838c 1 18338 1
1838c 1 18338 0
1a204 2 18210 1
18244 1 18338 1
1838c 1 18338 0
```

```
1a204 2 17fe8 1
18004 2 18380 1
1838c 1 18338 1
1838c 1 18338 0
1a204 2 17fe8 1
18004 1 1808c 1
```

The output of sys1 should be:

```
Number of branches = 9
Number of forward branches = 3
Number of forward taken branches = 3
Number of backward branches = 6
Number of backward taken branches = 3
Number of mispredictions = 6 0.666667
```

The verbose output of sys2 with 16 prediction buffers and 4 target buffers should be:

```
unixlab: ./sys2 test.trace 16 4 -v
0 1 1 2 1 1824 0 0
1 3 1 2 3 1838 0 0
2 3 2 3 3 1838 1 0
3 3 3 2 3 1838 2 0
4 1 2 3 1 1824 3 0
5 3 2 1 3 1838 4 0
6 3 1 2 3 1838 4 0
7 3 2 1 3 1838 5 0
8 1 3 3 1 1800 6 1
Number of branches = 9
Number of forward branches = 3
Number of forward taken branches = 3
Number of backward branches = 6
Number of backward taken branches = 3
Number of mispredictions = 6 0.666667
Number of BTB misses = 1 0.166667
unixlab:
```

Measurements and results

You should make measurements for the two trace files, then fill out this table for each of the traces:

	# mispredictions	misprediction	# BTB	BTB
		rate	misses	miss rate
System 1			N/A	N/A
System 2 (N=256,M=64)				
System 2 (N=1024,M=256)				

# mispredictions is the number of times the prediction from the prediction buffer does not match the behavior of the branch. Misprediction rate is # mispredictions / # branches.

A BTB miss occurs when a branch is predicted taken, but its target address is not in the target buffer. Branches that are not taken do not access the BTB. Hence, BTB miss rate is # BTB misses / # branches that are predicted taken.

Submit the result table with your code.

Submission (source code + results table):

Submit a tar/zip file using the iLearn submission link. Your tar/zip file should expand into a single directory tagged with your name. The directory should contain your source files and the results table; each source file should have a header that with accurate instructions on compiling and running your code on the Unix command line. If your instructions don't work perfectly, you may get a zero on the project.

Your instructions should allow us to generate two executables, sys1 and sys2 (for System 1 and 2 respectively). We will then run each one with this command line (assuming we're on unixlab.sfsu.edu) for C/C++:

```
./sys1 ~whsu/csc656/Traces/S17/P1/li1.trace [-v]
./sys2 ~whsu/csc656/Traces/S17/P1/li2.trace 1024 256 [-v]
etc etc
```

Or for Java:

```
java sys1 ~whsu/csc656/Traces/S17/P1/li1.trace [-v]
java sys2 ~whsu/csc656/Traces/S17/P1/li2.trace 1024 256 [-v]
etc etc
```