ECE 2500: Spring 2018
Project 3 (100 points)

Due on May 1st 9:00 PM
Late policy on canvas

You are allowed to use one of the following languages: C/C++/MATLAB/Python. You are **not** allowed to use any existing code that is available online (except for standard data structure libraries that implement map, arrays, linked list, etc.). If you want to use any other library, please first check with the instructor or the TA. Failure to follow these guidelines will result in a penalty.

Cache Simulator (70 points):

Write a program to simulate caches of different configurations. Make the following assumptions about all the caches we simulate.

1. The replacement strategy is LRU (Least Recently Used), if needed.
2. The cache is initially empty.
3. In write-through, the processor stalls until the data is written to the memory before proceeding. That is, the memory and cache are always consistent.

You must simulate caches with the following configurations:

Cache Size (in bytes):

1. 1K
2. 4K
3. 64K
4. 128K

Block Size (in bytes):

1. 8
2. 16
3. 32
4. 128

Cache Placement Type:

1. Direct Mapped
2. 2-way set associative
3. 4-way set associative
4. Fully associative

Write Policy:

1. Write back
2. Write through

Therefore, you must simulate 4x4x4x2, i.e., 128 different cache configurations.

Note that you have to write only one program that simulates all 128 cache configurations, one after the other. Use the fact that a direct-mapped cache is also a 1-way set associative cache.

Input File Format: We will use trace files to simulate our cache configurations. Each trace file is a sequence of memory accesses, one on each line, with the format:

```
read addressHex   OR
write addressHex
```

where address is a 32-bit address given in hex. Example files are provided on canvas and can be viewed in any text editor.

Output File Format: Your program should summarize the result of simulation for each cache configuration for a given trace file. The output must be a text file called `test.result`.

Example files are provided under the assignment link. You can view this using wordpad or other text editors.

There are 128 lines in each output file. Each line represents result for one specific cache configuration and is formatted as follows:

```
1024 8 FA WB 0.91 1024 512
```

The above numbers represent the following:

1. Cache size (in bytes)
2. Block size (in bytes)
3. Mapping type (DM, 2W, 4W, and FA for direct-mapped, 2-way and 4-way set associative, and fully-associative caches respectively)
4. Write policy (WB, WT)
5. Hit rate (two decimal points)
6. Total bytes transferred from memory to the cache
7. Total bytes transferred from cache to memory

Execution Format: Your program should, by default, be able to read in the input trace file named ‘test.trace’ and produce at the end an output file named ‘test.result’. Name your main executable (in case of C/C++/Python) as “simulate”.

You must create a separate file named “README.txt” in the submission folder. In this file, you must give detailed instructions on how to compile and execute your

program. If these instructions are missing or incomplete, we will not be able to test your program and points will be deducted. Be as detailed as needed. In particular, include the name of the compiler and OS platform that you used to create your program. We recommend using Visual Studio on Windows and gcc on Linux and Mac for C/C++, and Python 2.7 for python.

Trace Files (15 points):

As part of this project, you will have to generate two traces to test your cache simulator. Here are the traces you need to create:

1. Create a trace that shows the effect of associativity. In other words, create a trace that results in higher miss rate in direct mapped cache and lower miss rate in set associative caches. Name this trace *associative.trace*. You can use any appropriate cache size for generating this plot.

2. Create a trace that shows the effect of increasing the block size. Specifically, your trace should show an improvement in hit rate when the block size is increased from 8 bytes to 16 bytes. However, you have to show that the improvement stops at some point and increasing the block size hurts hit rate beyond that point. Use the 1K byte direct-mapped cache as the basis for this example. Name this trace *block.trace*.

Report (15 points):

Write a short report that describes your approach in implementing the project. Also, include two graphs representing the two traces that you have created. The first one should show miss rate with respect to associativity and the second one should show miss rate with respect to block size. Describe each graph in a few sentences.

What to turn in: Create a directory called **project3_*PID*** where ***PID*** is your VT login. Put your programs, pdf file report (also named project3_*PID*), your created trace files, and the README.txt file in the directory. Do NOT put your compiled project and executable in the folder. Zip this directory up and submit the resulting zip file on canvas.