

**For full and partial credits, you must show your work and explain all steps.**

In this lab, you will write a cache simulator. Your grade for the project will be based on the code (60%), your documentation (20%), and your testing cases (20%). You should submit a report detailing your code, its functionality, testing process, and, optionally, what you learned. Your code should be included with the report and should be well-structured and well-documented. You should include the tests you used with your code. Please include all materials you are submitting for the project in a single zip file and upload it to Canvas.

You will develop your cache simulator to operate as with the cache memory we discussed in class. The cache will be configurable with command line options as follows (you may include default values and make these optional parameters for your program):

- Block size (number of bytes).
- Number of blocks in the cache.
- Associativity (1 for direct, 2 for two-way set associative, etc.)
- (Extra credit 20%) Implement both random and LRU policies when appropriate.

Your program will be given a series of byte addresses (from a text file containing hex addresses, one address per line). One such file is provided to you as part of this lab. Given this input and for the cache as parameterized at the command line, compute the hit/miss rate. You may choose reasonable values according to your needs.

Your cache only needs to support reads. We will test your code with the test cases you developed as well as with some additional ones I've developed. The TA may request a meeting to discuss your code, testing, and other relevant details.

It is recommended to use Python for this assignment. However, C or Java is also acceptable. None of the code for this project is to be developed by others (i.e., no finding or sharing of external resources or code). The software will be checked for any similarity with other codes available online. Violation of this will result in a zero for the project and may lead to a referral to Judicial Affairs.

### **Testing guide**

The testing procedure of this programming assignment makes sure that your program is working correctly. In the grading process, we will evaluate with the following two steps:

Step 1: We are going to test with the following example to verify that your code is working. Your solution should give the same result as below: Suppose that there are a series of address references given as word addresses 0, 3, 11, 16, 21, 11, 16, 48, 16.

- a) Show the hits, misses, and final cache contents for a direct-mapped cache with one-word blocks and a total size of 16 words.

- b) Show the hits and misses and final cache contents for a two-way set-associative cache and a total size of 16 words. Assume LRU replacement. (Hint: there is a total of eight sets in this case)

Solution:

- a) Here is the hit and miss history for the memory accesses:

(Summary: the second 11 and the second 16 are hits, all others are misses)

0 (miss), 3 (miss), 11 (miss), 16 (miss), 21 (miss), 11 (hit), 16 (hit), 48 (miss), 16 (miss)

Contents in the cache at the end, in the format of (address: content): 0:16, 3:3, 5:21, 11:11

- b) The second 11, the second and the third 16 are hits, all others are misses.

0 (miss), 3 (miss), 11 (miss), 16 (miss), 21 (miss), 11 (hit), 16 (hit), 48 (miss), 16 (hit)

Set contents in the end:

Contents of cache blocks in the end															
Set 0	Set 0	Set 1	Set 1	Set 2	Set 2	Set 3	Set 3	Set 4	Set 4	Set 5	Set 5	Set 6	Set 6	Set 7	Set 7
48	16					3	11			21					

Step 2: We will test your program's robustness with large inputs, such as the address.txt file we provided in the programming assignment. Depending on your cache configuration, you will obtain different results. For example, with the block size as 16 words and with 256 blocks, assume a direct-mapped approach, then a program may print out the following:

**Cache size: 640k**

**Reads: 10000**

**Hits: 61**

**Misses: 9939**

**Hit Rate: 0.61%**

**Miss Rate: 99.39%**

On the other hand, if we increase the block size to 256 words, with 256 blocks, the result becomes:

**Cache size: 640k**

**Reads: 10000**

**Hits: 978**

**Misses: 9022**

**Hit Rate: 9.78%**

**Miss Rate: 90.22%**

As observed, increasing the block size and cache size helps reduce the cache miss rate.

Finally, to make it easier for the testing and grading, please make sure your program takes the following parameters:

- Options for command line arguments are: block size, number of lines (blocks), associativity, hit time, miss time, LRU, and datafile name. Your program should ensure that associativity x number of sets equals the number of blocks.