

# CSE 238/2038/2138 Systems Programming Project 3

Due: 07.06.2023 23:59 PM

**IMPORTANT NOTE: Implement this project only if you do not implement the bonus project.**

## 1 Logistics

This is a group project. Project groups are of 4 people. You must run this project on a 64-bit x86-64 machine.

## 2 Downloading the assignment

You can download the handout for the project through Canvas. You will need the trace files in `traces.rar` and the RAM image in `RAM.dat`.

## 3 Description

In this project, you will write a cache simulator which takes an image of memory and a memory trace as input, simulates the hit/miss behavior of a cache memory on this trace, and outputs the total number of hits, misses, and evictions for the cache along with the content of the cache at the end.

Your simulator will take the following command-line arguments:

Usage: `./your_simulator -s <s> -E <E> -b <b> -t <tracefile>`

- `-s <s>`: Number of set index bits for the cache ( $S = 2^s$  is the number of sets)
- `-E <E>`: Associativity for the cache (number of lines per set)
- `-b <b>`: Number of block bits for the cache ( $B = 2^b$  is the block size)
- `-t <tracefile>`: Name of the trace file (see Reference Trace Files part below)

The command-line arguments are based on the notation ( $s$ ,  $E$ , and  $b$ ) from page 652 of the CS:APP3e textbook.

For example, if you want to simulate a 2-way set associative ( $s=1$ ) cache of 2 lines ( $E=2$ ) and 8 blocks ( $b=3$ ), and see the results for the trace file `test1.trace`, you will run your program with the arguments given in the following example. Also, the results should be in the given format.

```
linux> ./your_simulator -s 1 -E 2 -b 3 -t test1.trace
hits:0 misses:1 evictions:0
```

```
L 5, 3
Miss
Place in set 0
S 0, 1, ab
Hit
Store in cache and RAM
```

## Programming Rules

- You can use either Java or C.
- Name of your source code file should include your student IDs like `student1ID_student2ID_student3ID_student4ID.c`
- Your code must compile without warnings in order to receive credit.
- Your simulator must work correctly for different sets of  $s$ ,  $E$ , and  $b$  values for each cache type. This means that you will need to allocate storage for your simulator's data structures using the `malloc` function. Type "man malloc" for information about this function.

- Each of the caches will implement write-through and no write allocate mechanism for store and modify instructions.
- For the evictions, FIFO (first in first out) policy will be used.
- To receive credit, you must print the total number of hits, misses, and evictions, at the end of your program.
- For this project, you should assume that memory accesses are aligned properly, such that a single memory access never crosses block boundaries (Read and write requests are less than or equal to block size).
- We will hand you sample RAM image for testing. For each Miss in the caches you can fetch the data from the text file “RAM.dat”. This file is a binary file that includes the contents of the memory in the beginning of the execution.
- The contents of the cache at the end of the execution will be written to the corresponding txt file.

## Reference Trace Files

The `traces` subdirectory of the handout directory contains a collection of *reference trace files* that we will use to evaluate the correctness of the cache simulator you write. The memory trace files have the following form:

```
M 000ebe20, 3, 58a35a
L 000eaa30, 6
S 0003b020, 7, abb2cdc69bb454
```

Each line denotes one or two memory accesses. The format of each line for L:  
*operation address, size*

The format of each line for M and S:  
*operation address, size, data*

The *operation* field denotes the type of memory access:

- “L” a data load,
- “S” a data store, and
- “M” a data modify (i.e., a data load followed by a data store).

The *address* field specifies a 32-bit hexadecimal memory address.

The *size* field specifies the number of bytes accessed by the operation.

The *data* field specifies the data bytes stored in the given address.

**EXAMPLE RUN:**

test1.trace:

L 5, 3

S 0, 1, ab

Initial image of RAM and cache:

Cache				RAM	
tag	time	v	data	Address	Data
				0x00000000	1234567887654321
				0x00000008	8765432112345678
				0x00000010	0abcd1e22e1dcba0
				0x00000018	f1e2a3d44d3a2e1f
				0x00000020	c1a2b3e44e3b2a1c
				0x00000028	0123abcdcdba3210
				0x00000030	dcba12300321abcd
				0x00000038	02468aceeca86420

Your simulation:

linux&gt; ./your\_simulator -s 0 -E 2 -b 3 -t test1.trace

hits:0 misses:1 evictions:0

L 5, 3

Miss

Place in cache

S 0, 1, ab

Hit

Store in cache and RAM

L 5, 3  
 L1D miss, L2 miss  
 Place in L2 set 0, L1D

Cache				RAM	
tag	time	v	data	Address	Data
0000000	1	1	1234567887654321	0x00000000	1234567887654321
				0x00000008	8765432112345678
				0x00000010	0abcd1e22e1dcba0
				0x00000018	f1e2a3d44d3a2e1f
				0x00000020	c1a2b3e44e3b2a1c
				0x00000028	0123abcd dcba3210
				0x00000030	dcba12300321abcd
				0x00000038	02468aceeca86420

S 0, 1, ab  
 L1D hit, L2 hit  
 Store in L1D, L2, RAM

Cache				RAM	
tag	time	v	data	Address	Data
0000000	1	1	ab34567887654321	0x00000000	ab34567887654321
				0x00000008	8765432112345678
				0x00000010	0abcd1e22e1dcba0
				0x00000018	f1e2a3d44d3a2e1f
				0x00000020	c1a2b3e44e3b2a1c
				0x00000028	0123abcd dcba3210
				0x00000030	dcba12300321abcd
				0x00000038	02468aceeca86420

## 5 Working on the Project

Here are some hints and suggestions for working on the project:

- Each data load (L) or store (S) operation can cause at most one cache miss (Always aligned. If blocks are 4 bytes, there will be no request more than 4 bytes in test trace inputs). The data modify operation (M) is treated as a load

followed by a store to the same address. Thus, an M operation can result in two cache hits, or a miss and a hit plus a possible eviction.

## **6 Handing in Your Work**

You will submit your handin through Canvas. You will handin your source code.