CSE 238/2038/2138 Systems Programming Project 3
Due: 25.05.2017 23:59 PM

1 Logistics
This is an individual project. You must run this lab 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 in that 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 each cache type along with the content of each cache at the end.

Your simulator will take the following command-line arguments:

```
Usage: ./your_simulator -L1s <L1s> -L1E <L1E> -L1b <L1b>
                        -L2s <L2s> -L2E <L2E> -L2b <L2b>
                        -t <tracefile>
```

- `-L1Ds <L1Ds>`: Number of set index bits for L1 data/instruction cache ($S = 2^s$ is the number of sets)
- `-L1DE <L1DE>`: Associativity for L1 data/instruction cache (number of lines per set)
- `-L1Db <L1Db>`: Number of block bits for L1 data/instruction cache ($B = 2^b$ is the block size)
- `-L2s <L2s>`: Number of set index bits for L2 cache ($S = 2^s$ is the number of sets)
- `-L2E <L2E>`: Associativity for L2 cache (number of lines per set)
- `-L2b <L2b>`: Number of block bits for L2 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. The s, E, and b values will be the same for both L1 data and instruction caches.

For example, if you want to simulate a fully associative (s=0) L1 cache of 2 lines (E=2) and 8 blocks (b=3), and a 2-way set associative (E=2) L2 cache of 2 sets (s=1) 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  -L1s 0 -L1E 2 -L1b 3 -L2s 1 -L2E 2 -L2b 3 -t test1.trace
      L1I-hits:0 L1I-misses:1 L1I-evictions:0
      L1D-hits:1 L1D-misses:1 L1D-evictions:0
      L2-hits:1 L2-misses:2 L2-evictions:0

  L 5, 3
    L1D miss, L2 miss
    Place in L2 set 0, L1D
  I 10, 8
    L1I miss, L2 miss
    Place in L2 set 0, L1I
  S 0, 1, ab
    L1D hit, L2 hit
    Store in L1D, L2, RAM
```

Programming Rules

- Name of your `.c` file should include your name like *yournamesurname.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.

- For this lab, we are interested in L1 data cache, L1 instruction cache, and unified L2 cache performance.

- Each of the caches will implement <u>write-through and no-write allocate</u> mechanism for store and modify instructions.

- For the <u>evictions, LRU (least recently used)</u> line will be selected.

- To receive credit, you must call the function `printSummary` for each cache (L1D, L1I, L2), with the total number of hits, misses, and evictions, at the end of your `main` function:

        printSummary(hit_count, miss_count, eviction_count);

- For this lab, 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 L2 level cache you can fetch the data from the binary file "RAM.dat". This file is a byte file that includes the contents of the memory in the beginning of the execution.

- The contents of the caches at the end of the execution will be written to the corresponding byte file for each cache.

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
        I 00002010, 6

Each line denotes one or two memory accesses. The format of each line for I and 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:
- "I" denotes an instruction load,
- "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
    I 10, 8
    S 0, 1, ab
```

Inital image of RAM and caches:

L1I

tag	time	v	data

L1D

tag	time	v	data

RAM

Address	Data
0x00000000	1234567887654321
0x00000008	8765432112345678
0x00000010	0abcd1e22e1dcba0
0x00000018	f1e2a3d44d3a2e1f
0x00000020	c1a2b3e44e3b2a1c
0x00000028	0123abcddcba3210
0x00000030	dcba12300321abcd
0x00000038	02468aceeca86420

L2

	tag	time	v	data
Set 0				
Set 1				

Your simulation:
```
  linux> ./your_simulator  -L1s 0 -L1E 2 -L1b 3 -L2s 1 -L2E 2 -L2b 3 -t test1.trace

      L1I-hits:0 L1I-misses:1 L1I-evictions:0
      L1D-hits:1 L1D-misses:1 L1D-evictions:0
      L2-hits:1 L2-misses:2 L2-evictions:0

  L 5, 3
    L1D miss, L2 miss
    Place in L2 set 0, L1D
  I 10, 8
    L1I miss, L2 miss
    Place in L2 set 0, L1I
  S 0, 1, ab
    L1D hit, L2 hit
    Store in L1D, L2, RAM
```

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

L1I

tag	time	v	data

L1D

tag	time	v	data
0000000	1	1	1234567887654321

L2

	tag	time	v	data
Set 0	0000000	1	1	1234567887654321
Set 1				

RAM

Address	Data
0x00000000	1234567887654321
0x00000008	8765432112345678
0x00000010	0abcd1e22e1dcba0
0x00000018	f1e2a3d44d3a2e1f
0x00000020	c1a2b3e44e3b2a1c
0x00000028	0123abcddcba3210
0x00000030	dcba12300321abcd
0x00000038	02468aceeca86420

```
I 10, 8
L1I miss, L2 miss
Place in L2 set 0, L1I
```

L1I

tag	time	v	data
0000002	2	1	0abcd1e22e1dcba0

L1D

tag	time	v	data
0000000	1	1	1234567887654321

L2

	tag	time	v	data
Set 0	0000000	1	1	1234567887654321
	0000001	2	1	0abcd1e22e1dcba0
Set 1				

RAM

Address	Data
0x00000000	1234567887654321
0x00000008	8765432112345678
0x00000010	0abcd1e22e1dcba0
0x00000018	f1e2a3d44d3a2e1f
0x00000020	c1a2b3e44e3b2a1c
0x00000028	0123abcddcba3210
0x00000030	dcba12300321abcd
0x00000038	02468aceeca86420

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

L1I

tag	time	v	data
0000002	2	1	0abcd1e22e1dcba0

L1D

tag	time	v	data
0000000	3	1	ab34567887654321

L2

	tag	time	v	data
Set 0	0000000	3	1	ab34567887654321
	0000001	2	1	0abcd1e22e1dcba0

Set 1				

RAM

Address	Data
0x00000000	ab34567887654321
0x00000008	8765432112345678
0x00000010	0abcd1e22e1dcba0
0x00000018	f1e2a3d44d3a2e1f
0x00000020	c1a2b3e44e3b2a1c
0x00000028	0123abcddcba3210
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 code.