**CSCE 312: Computer Organization Fall 2021**

**Project**



## The Cache Simulator

The purpose of this project is to get you familiar with cache memory organization, cache configuration, and replacement policies.

You will build a command-line based cache simulator. The simulator uses two data structures: a configurable cache memory and a physical memory (RAM). The address width is **8 bits**.

Write your program in any of the following languages (Python, Java, C, or C++). No other languages will be accepted.

### 1. Initialize the physical memory

Your program should start by initializing the physical memory with bytes in hexadecimal provided in input.txt. The path of this file will be received as a command-line argument. The physical memory is **byte-addressable**. The cache memory will use the data generated in the physical memory; therefore, you should make sure that the RAM has some content as given file before starting the simulator

```
*** Welcome to the cache simulator ***
initialize the RAM:
init-ram 0x00 0xFF
RAM successfully initialized!
```

## 2. Configure the cache

After initializing the physical memory, your program should configure the cache. The program should allow the user to configure the cache using the following parameters.

| Parameter | Description |
|---|---|
| Cache size | The aggregate size of all the cache blocks. The unit for the cache size is in Bytes. The allowed cache size should be between **8 Bytes** and **256 Bytes**. |
| Data block size | The number of bytes that a block can contain. |
| Associativity | An n-way set associative cache holds n lines per set. The user can only choose from a 1-way, 2-way, or 4-way set-associative cache. |
| Replacement policy | Defines how to replace a cache entry following a cache miss. The user can only choose from the **Random replacement (RR) (1)** or the **Least Recently Used (LRU) (2)** policy. |
| Write hit policy | Defines where to write the data when an address is a hit. The user can only choose from a write-through or write-back policy.<br>**write-through (1):** write the data in both the block in the cache and the block in RAM.<br>**write-back (2):** write the data only in the block in the cache. |
| Write miss policy | Defines where to write the data when an address is a miss. The user can only choose from a write-allocate or no write-allocate policy.<br>**write-allocate (1):** load the block from RAM and write it in the cache.<br>**no write-allocate (2):** write the block in RAM and do not load it in the cache. |

Note: The cache configuration input order will be the same i.e.

Cache size->data block size->associativity->replacement policy->write hit policy->write miss policy

Every input for the configuration is an integer.

```
configure the cache:
cache size: 32
data block size: 8
associativity: 2
replacement policy: 1
write hit policy: 1
write miss policy: 1
cache successfully configured!
```

### 3. Simulate the cache

The program should show a menu to the user with a list of available options to perform. The options are **cache-read, cache-write, cache-flush, cache-view, memory-view, cache-dump, memory-dump, quit.**

Commands would be typed as keywords like **cache-read**, **cache-write**, etc., not numbers.

```
*** Cache simulator menu ***
type one command:
1. cache-read
2. cache-write
3. cache-flush
4. cache-view
5. memory-view
6. cache-dump
7. memory-dump
8. quit
**************************
```

**cache-read:** the cache-read command reads data from an address. The user should type the **read** command followed by an 8-bit address in hexadecimal.

The following example reads from address **0x18** = 000 11 000

Output format:

Line 1: set:<set number in decimal>

Line 2: tag:<tag number in hex>

Line 3: hit:<cache hit? yes|no>

Line 4: eviction_line: <line to evict in decimal. If cache hit, -1>

Line 5: ram_address:<ram address to read in hexadecimal. If cache hit, -1>

Line 6: data:<data in hexadecimal>

Sample of cache miss:

```
cache-read 0x18
set:3
tag:0
hit:no
eviction_line:0
ram_address:0x18
data:0x84
```

**cache-write:** the cache-write command writes data to an address in the cache. The user should type the write command followed by an 8-bit address and a byte of data in hexadecimal. In case of a tie following LRU or LFU replacement policies, evict one with the min line number.

The following example writes data **0xAB** to address **0x10** = 000 10 000

Output format:

Line 1: set:<set number in decimal>

Line 2: tag:<tag number in hex>

Line 3: write_hit:<cache hit? yes|no>

Line 4: eviction_line: <line to evict in decimal. If write hit, -1>

Line 5: ram_address:<ram address to read in hexadecimal. If write hit, -1>

Line 6: data:<data to write in hexadecimal>

Line 7: dirty_bit:<1|0>

Sample of cache write miss:

```
cache-write 0x10 0xAB
set:2
tag:0
write_hit:no
eviction_line:0
ram_address:0x10
data:0xAB
dirty_bit:1
```

**cache-flush:** the cache-flush command clears the cache.

```
cache-flush
cache_cleared
```

**cache-view:** the cache-view command displays the cache content and status.

The view should print the cache configuration and the cache's content.

Output format:

Line1: cache_size:<integer in bytes>

Line2: data_block_size:<integer in bytes>

Line3: associativity:<integer 1|2|4>

Line4: replacement_policy:<random_replacement|least_recently_used>

Line5: write_hit_policy:<write_through|write_back>

Line6: write_miss_policy:<write_allocate|no_write_allocate>

Line7: number_of_cache_hits:<integer>

Line8: number_of_cache_misses:<integer>

Line9: cache_content:

<valid bit 0|1> <dirty bit 0|1> <tag in 2 hexadecimal numbers> <hex data block with single-space-separated bytes>

Example: In `0 1 F0 F6 AB 01 22 A5 A6 44 DB`, the valid bit is 0, the dirty bit is 1, the tag is F0, and the data is F6 AB 01 22 A5 A6 44 DB.

Sample of cache view

```
cache-view
cache_size:32
data_block_size:8
associativity:2
replacement_policy:random_replacement
write_hit_policy:write_through
write_miss_policy:write_allocate
number_of_cache_hits:2
number_of_cache_misses:6
cache_content:
0 0 F0 F6 AB 01 22 A5 A6 44 DB
0 0 01 DA FF 23 11 A5 10 29 87
0 0 01 F6 AB CD 97 BB A6 72 DB
0 0 01 F6 AB CD 97 BB A6 72 DB
```

**memory-view:** the memory-view command displays the RAM content and status.

Line 1: memory_size:<integer in bytes>

Line 2: memory_content:

address:data

0x<address in hexadecimal>:<8 single-space-separated bytes of data in hexadecimal>

Example: In `0x00:F6 AB 01 22 A5 A6 44 DB`, the address is 00 and the data is F6 AB 01 22 A5 A6 44 DB

Sample of memory view

```
memory-view
memory_size:256
memory_content:
address:data
0x00:F6 AB 01 22 A5 A6 44 DB
0x08:DA FF 23 11 A5 10 29 87
0x10:F6 AB CD 97 BB A6 72 DB
```

**cache-dump:** the cache-dump command dumps the current state of cache in a file cache.txt.

The cache.txt file gets updated only when this command is called.

Output format in **cache.txt**

Line1:<hex data in set 0 line 0 with single-space-separated byte byte0 byte1 ...>

Line2:<hex data in set 0 line 1 with single-space-separated byte byte0 byte1 ...>

..

Line n:<hex data in the last set last line with single-space-separated byte byte0 byte1 ...>

Sample of cache dump

```
cache-dump
```

```
F6 AB 01 22 A5 A6 44 DB
DA FF 23 11 A5 10 29 87
F6 AB CD 97 BB A6 72 DB
F6 AB CD 97 BB A6 72 DB
```

**memory-dump:** the memory-dump command dumps the RAM content in a file **ram.txt**.
The ram.txt file gets updated only when this command is called.
Output format in **ram.txt**
Line 1: hex data at address 0
Line 2: hex data at address 1
Line 3: hex data at address 2
...
Line 256: hex data at address 255

```
memory-dump
F6
AB
01
22
A5
A6
44
DB
DA
FF
23
11
A5
10
29
87
DA
FF
23
11
A5
10
29
87
...
```

**quit:** the quit command exits the program. Otherwise, it always waits for the next command in an infinite loop.

**README.txt:**

All projects must include a README.txt file that mentions how to compile and run the program.

## Extra credit

Implement an additional cache line replacement policy for your simulator: **Least Frequently Used (LFU) (3)**. You must update your menu of commands with the additional command.

In the configuration, if the user enters **replacement policy: 3,** then the cache should use the LFU policy.

## Testing instructions

- The sample outputs provided above do not cover all the cases encountered during the program execution.
- **If your program does not compile or has runtime errors, you will not receive credit**. Use office hours and labs to resolve such issues before submission.
- If your program show warnings, points will be deducted.
- Do not add extra spaces, commas, semicolons, or extra words other than the provided output format.
- Do not change the input order for the cache config or the given syntax for the cache commands; otherwise, the grading tool will fail at testing your program.
- Your program will be tested with multiple test cases in which multiple cache command options are given.
- The project is partially auto-graded, so the output formats and input sequences are important. You will lose all marks for test cases in which the output format does not match the requirement or those for which the input order is incorrect.
- Dumps should be in filenames **cache.txt** and **ram.txt** only. No other file names are accepted.

## Notes

- The cache will always have a dirty bit irrespective of the policy.
- For random replacement, evict random lines in case of invalid lines too.
- Line numbers within a set start from 0.
- The eviction line number is relative to the set. e.g., in a 4-associativity, `eviction_line:3` evicts line 3 within set 1.
- Use the cache formulas C=S*E*B and m = t+s+b in your cache configuration. In our test cases, you can expect C to be divisible by E*B.
- The hexadecimal inputs are uppercase and two digits; e.g., 0xAA.
- Demo video: https://youtu.be/-1Gnlk3I3ZA

## Individual work, group work, and peer-review

You can work on this project individually or as a team of **2 members** at most. Every team member is required to participate. If working in a team, you will be evaluated partially by your teammate

using a **peer-review evaluation**. If you need additional help with your project, you should ask the TAs or the course instructor. The TAs are available during regular lab time and office hours. You can also post general questions on Piazza.

### Requirements for compilation

Your file should be **cachesimulator.c|cpp|py|java,** do not use a different file name, or the autograder will ignore it.

If you write the code in **C or C++**:

- Put the main function in a file called **cachesimulator.c/.cpp**
- The program should compile and generate an executable cachesimulator.out or cachesimulator.exe by using: g++ -std=c++11 *.cpp -o cachesimulator
- The program should accept one command line argument when you run it, so you should make sure the program is run with the command: ./cachesimulator.out <input.txt file> or ./cachesimulator.exe <input.txt file>.

If you write the code in **Java**:

- Put the main function in a file **cachesimulator.java** and make sure cachesimulator.class is generated after compiling using: javac *.java
- The program should accept one command line argument when running it, so you should get the output by running: java cachesimulator input.txt. Do not assume that the input.txt file is in the same directory as the .java files. The input.txt file can be substituted with a path to an input.txt file.

If you write the code in **Python**:

- You must use Python 3.
- Put the main function in a file called **cachesimulator.py**.
- Make sure your program accepts one command line argument when you run it, so you should get the input.txt file after running: python cachesimulator.py input.txt

### Requirements for teams

If you are working on the project in a team, you must do the following to get a grade.

- Enter the names of the team members in the teams' spreadsheet by 11/24/2021.
- Once you join a team, you cannot switch teams.
- Each team member must fill out the **required** peer evaluation form starting 12/06/2021.
- In Gradescope, you must add the two team members' names to the submission. If a member's name is missing from the submission, that member will not get a grade.

### Academic Dishonesty

**We will check your submission for plagiarism and authenticity. Copying work from another source (Github, Chegg, Stackoverflow, etc.) and submitting it as your work is plagiarism and violates the honor code. The minimum penalty for plagiarism is a grade of zero and will be reported to the Aggie honor system office.**

### Submitting Your Assignment

Once you have completed your project, submit all files on Gradescope. You can resubmit your files as many times as you need until the due date. Only the most recent submission is graded. You are required to include the following lines in the header of all your files:

```
// File: filename
// Author(s):
// Date: xx/xx/2021
// Section: Student section number
// E-mail(s): student_email@tamu.edu
// Description:
// e.g., The content of this file implements ...
```

Submit your files on gradescope.com

The last day to submit your project is **Wednesday, December 08th, 2021, at 11:59 PM CST**.

**No late submissions will be accepted.**

**The project submission should include:**

a. Your project file(s)

b. Your generated project documentation in HTML using Doxygen