Homework #3: Building a Cache Simulator

Due: Dec 15, 11:59 PM
Responsible TA: Dongyeon Yu (dy3199@unist.ac.kr)

- **Several updated notes:**
  - We will only consider for word addresses in the range of `0x10000000` to `0x10000700`. In other words, valid addresses are multiples of 4 within the range of `0x10000000` to `0x10000700`.

1 Homework Description

In this homework, you will build a 4-way set associative cache given memory access log as input. The primary goal of this assignment is to deepen your understanding of how a memory hierarchy works and how caches operate within it. You will explore key concepts such as cache organization, data transfer between cache and main memory, and optimization techniques like Least Recently Used (LRU) replacement policies.

2 Submission

- **Since the final exam period is approaching, it is recommended to complete this assignment as soon as possible.**
- Late submission will be assessed a penalty of 10% per day (We will only accept late submissions of up to 3 days).
- You should complete all sections of the provided skeleton code [1] and submit the completed files as a zip archive named in the format `Your_ID-hw3.zip`. For example, if your ID is 20231234, then you should submit a file named `20231234-hw3.zip`. Upload this zip file to Blackboard.
- You should include comments explaining each part of your implemented logic within the code. Comments can be written in either English or Korean.

3 Execution Environment

We will conduct grading in the same execution environment as homework 2. You are required to fulfill this homework using Python 3.9, utilizing only the Python standard library [3]. This means we do not assume any additional libraries downloaded via `pip`. Note that we will evaluate your code in a Python 3.9.20 on Ubuntu 20.04. If your code does not run correctly in our environment, you will receive zero points for this homework. No exceptions will be made.

To help you ensure that your code runs correctly in our grading environment, we provide a Docker image [2]. You can set up our grading environment by running the following two commands in your code directory.

```
$ docker pull cse261/hw
$ docker run --rm -it -v .:/app cse261/hw /bin/bash
```

Using the `-v` option, the working directory inside the grading environment will be linked to the working directory in your host environment. By running your code in this grading environment, you can ensure that it runs properly before submission.

4 Cache Details

Download the skeleton code from [1]. In this homework, your task is to complete the skeleton code. The main goal of this project is building a **4-way set associative** cache.

4.1 Simulator Execution

`main.py` reads the input file (§4.2), a memory access log, and displays the corresponding execution results (§4.4), including cache and memory dumps. Below is an example of its usage:

```
python3 main.py --access-log testcase_read_hit/memory_access_log.txt
```

We provide multiple `testcase_*` directories within the skeleton code directory. You can use these `testcase_*` directories to verify that your cache simulator code works correctly. During our evaluation, we will test your code not only with the `testcase_*` test cases but also with various other test cases to ensure it operates properly.

4.2 Input File: Memroy Access Log (`memory_access_log.txt`)

In this homework, a memory access log file will be provided as input. This file contains traces of memory read and write operations. Specifically, a memory access log is a sequence consisting of an [address] and an optional [value]. The address shows the memory location the CPU is accessing, and if a value is included, it means the CPU is writing data to that location. If no value is given, it means the CPU is reading from that address. All numbers are represented in hexadecimal format. The layout of the memory trace is as follows:

```
[Col 1: Address] [Col 2: Value]
```

Below is an example of the `memory_access_log.txt` file:

```
0x10000504 0xAAAAAAAA  # Memory write: write 0xAAAAAAAA to 0x10000504
0x10000504             # Memory read: read data from 0x10000504
```

In the example, hexadecimal values were listed with comments following the # symbol indicating the meaning of each memory access. Keep in mind that during evaluation, we will provide test input files where no comments are included.

4.3 Cache Layout

Your goal is to building a 4-way associative cache given memory access log file as input. Your cache simulator should satisfy the following specification:

- **Block size**: 32-byte
- **Number of sets**: 8
- **Write policy**: write back, allocation
- **Replacement scheme**: Least Recently Used (LRU)
- **Cache size**: (8 sets x 4 blocks/set x 32 bytes/block) = 1KB (except for valid bit and dirty bit)

index	v	tag	way 0	v	tag	way1	v	tag	way2	v	tag	way3
000	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..
001	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..
010	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..
011	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..
100	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..
101	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..
110	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..
111	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..	0	00000	..cache block data (32byte)..

- **Address translation layout**:

- Byte offset: log2(word size in bytes) = log2(4) = 2 bits
- Block offset: log2(block size in words) = log2(8) = 3 bits
- Index: log2(# sets) = log2(8) = 3 bits
- Tag: 32 - 3 - 3 - 2 = 24 bits

Tag (24 bits)	**Cache index** (3 bits)	**Block offset** (3 bits)	**Byte offset** (2 bits)

4.4 Output Format

When you run `main.py`, you must display (1) the current state of the cache and (2) the current state of the memory *after each memory access*, all in the standard output. Note that the `dump` functions for printing the cache and memory values are provided in the `cache.py` and `ram.py` files, respectively. We provide an `expected_result.txt` file in each `testcase_*` folder, which contains the expected output for the corresponding directory's input files.

4.5 Skeleton Code - `cache.py` and `ram.py`

The files `cache.py` and `ram.py` contain the implementation of the cache simulator's core functionality. Figure 1 illustrates the data transfer flow, which will be helpful when you are doing this homework. Your task is to complete the following functions to enable proper handling of read and write operations in the 4-way set associative cache:

- **Implement the `read` function in `cache.py`.** This function retrieves data from the cache using the given `address`. If the data is not found (read miss), the block is fetched from main memory, the cache is updated, and the data is returned. Use an LRU replacement policy to handle block replacements within the set. Dirty blocks must be written back to main memory when evicted. Finally, this function should return the data as an integer value corresponding to the requested memory address.

  Tip: You can leverage the `self.history` attribute in the `Set` class to implement the LRU policy, which tracks the access order of blocks within the set. The block at the first index of the `self.history` represents the least recently used (LRU) block. For example, if `self.history` is initialized as `[0, 1, 2, 3]`, this means that the first block (index 0) is considered the least recently used and has the highest priority for replacement. The fourth block (index 3) is treated as the most recently used and has the lowest priority for replacement.

- **Implement the `write` function in `cache.py`.** This function stores `data` in the cache at the specified `address`. If the block is present in the cache, it updates the data and marks the block as dirty. On a write miss, a block is allocated according to the cache's allocation policy, and the value is written. Dirty blocks must be written back to main memory when evicted.

- **Implement the `block_read` function in `ram.py`.** This function retrieves a 32-byte block of data from the main memory. It calculates the starting address of the block using the provided `address` parameter. Finally, this function should return the block data as a list of integers, where each integer represents a word in the 32-byte block.

- **Implement the `block_write` function in `ram.py`.** This function writes a 32-byte block of data from the cache back to the main memory. It uses the `address` parameter to determine the starting address in memory and updates the memory locations with the provided block data.

5 Test Your Code

As mentioned in §4.1, there are multiple `testcase_*` directories within the skeleton code directory. Each `testcase_*` directory includes input file, i.e., `memory_access_log.txt`, as well as an `expected_result.txt` file, which contains the expected output corresponding to this input file (the content that should appear in the standard output). You can use these to check if your code produces the expected output for the given input files. To facilitate comparison, we provide a tool called `test_cpu.py`. You can run `test_memory.py` with the following command:

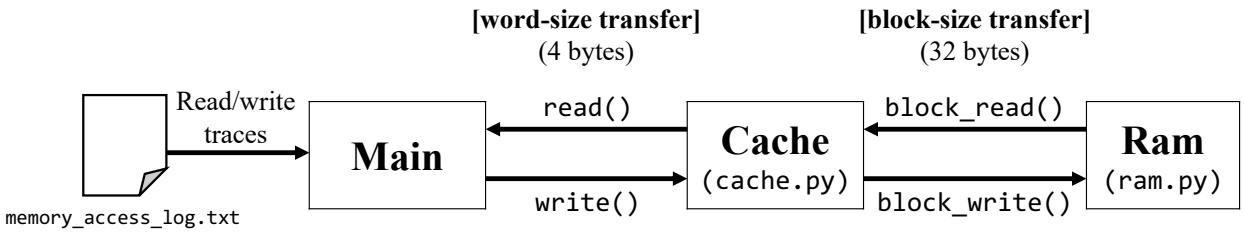

Figure 1: Data transfer call flow.

```
python3 test_memory.py <test_case_dir>
```

Using this command, you can verify whether your cache simulator works correctly for the specified `<test_case_dir>`. If it operates correctly, the standard output will display `"The output matches the expected output."` If it does not, it will show `"The output does NOT match the expected output."`, along with details of any discrepancies. Note that we will evaluate your submitted cache program by using `test_memory.py`.

6 Note

- **Initial value.** When the simulator starts, all cache and memory data are initialized to 0, ensuring a clean and predictable state for the simulation. Regarding the LRU initialization, the `self.history` attribute in the Set class is initialized as `[0, 1, 2, 3]`. This means that the first block (index 0) is considered the least recently used and has the highest priority for replacement. The fourth block (index 3) is treated as the most recently used and has the lowest priority for replacement.
- **Memory address range.** In this simulator, we only consider memory addresses within the range `0x10000000` to `0x10000700`. You do not need to handle exceptional cases such as invalid memory address access (e.g., attempting to write a value to `0x11111010`), as we will only consider inputs within the specified valid range during grading.
- **Be careful about plagiarism!** We will conduct strict cross-plagiarism detection for evaluation, including a series of answer generated by ChatGPT, code found online, and your submissions. Therefore, we do not recommend consulting ChatGPT or online solutions. Last semester, we identified several cases of plagiarism using an automated tool. If you are found to be engaged in "deep collaboration" with other students, both the provider and the recipient will receive penalties, including, in the worst case, receiving an F grade.
- **Grading policy.** We will grade strictly based on the accuracy of the output relative to the input. No excuses will be accepted for discrepancies. Unlike Homework 2, we will not test each function individually in this assignment. Therefore, if you do not want to use the skeleton code, you are allowed to write the code from scratch. Additionally, you are free to modify the parameters, return types, or other aspects of the functions. However, you are supposed to follow the execution command, input formant, and output format exactly.
- **Questions.** If you have any requests or questions (technical difficulties, late submission due to inevitable circumstances, etc.), please ask the TAs on Blackboard. We generally encourage the use of Blackboard for discussions. However, for urgent issues or secret issues, you can send an email to the responsible TA, Dongyeon Yu.

References

[1] 2024. CSE261 HW3 skeleton code. https://websec-lab.github.io/courses/2024f-cse261/hw/hw3-skeleton.zip.

[2] 2024. Docker Hub Image: cse261/hw. https://hub.docker.com/repository/docker/cse261/hw.

[3] 2024. The Python Standard Library. https://docs.python.org/3.9/library/.