# CSE26101 Project 4: Building a Cache Simulator

***Due 11:59 PM, Dec 19th***

(No late submission allowed)

## 1. Overview

In this project, you will build a **2 level 4-way associative cache** given memory traces as inputs.

## 2. Simulation Details

### 2.1 Cache Specification

Your simulator should satisfy the following specification:

- L1 cache
  1. Block (cacheline) size: **32-byte**
  2. Number of sets: **8**
  3. Write policy: **write back**
  4. Replacement scheme: **LRU**
  5. Cache size: **(8 sets *4 block/set * 32byte/block) = 1KB**
  6. Address translation layout:
     Offset: **log2(cacheline) = log2(32) = 5 bits**
     Index: **log2(# sets) = log2(8) = 3 bits**
     Tag: **32 - 5 - 3 = 24 bits**

- L2 cache
  1. Block (cacheline) size: **64-byte**
  2. Number of sets: **8**
  3. Write policy: **write back**
  4. Replacement scheme: **LRU**
  5. Cache size: **(8 sets *4 block/set * 64byte/block) = 2KB**
  6. Address translation layout:
     Offset: **log2(cacheline) = log2(64) = 6 bits**
     Index: **log2(# sets) = log2(8) = 3 bits**
     Tag: **32 - 6 - 3 = 23 bit**

### 2.2 Memory Traces as Inputs

In this project memory traces would be given as inputs, not binary code.

A memory trace is a sequence of an *address* and a possible *value* where, the *address* specifies the address location the CPU is trying to access, and the *value* specifying a *read* operation when none is given or a *write* operation when a number is given.

The layout of the memory trace is given as follows.
[Col 1: Address] [Col 2: Value]
Ex)
For memory write

```
    2147478952 2147483552
```

For memory read

```
    2147478968
```

Example trace

```
2147478872 2147483544
2147478888 2147479504
```

```
2147478892 1009
2147478892
2147478892
2147478888
```

### 2.3 Functions to implement.

You should implement the following functions in ***memory_system.py***.

**The following are the functions with the function header provided in class *cache*.**

**2.3.1 read(address)**

*read(address)* returns the value pointed by the address from the cache.

**2.3.2 write(address, value)**

*write(address,value)* stores the value in the cache pointed by the address.

**2.3.3 fetch(address)**

*fetch(address)* reads a value from the address in the lower layer of the memory hierarchy.

**2.3.4 write_back(address, value)**

*write_back(address, value)* writes a value into the address in the lower layer of the memory hierarchy.

**The following guidelines are for implementing the LRU policy. You can implement them freely anywhere in memory_system.py. We provide some hints for your convenience, but everything depends on your decision.**

**2.3.5 LRU()**

*LRU()* is the policy for selecting the block to evict.

**2.3.6 evict()**

*evict()* evicts a cache block into the lower hierarchy of the memory system.

**2.3.7 is_hit()**

*is_hit()* is used for checking cache hits.

**2.3.8 get_block()**

*get_block()* is used for fetching a block index from the memory

## 3. Testing and Submission via PA Submit System (PASS)

We will use ***PA Submit System***, developed by the Systems Software Lab of Ajou University, to manage the programming assignments. Please refer to the BlackBoard announcement for the PASS user manual.

### 3.1 Downloading the Skeleton Code

The skeleton code is available on BlackBoard under “Assignment” → “PA4: Building a Cache Simulator”. Please download and read the skeleton code carefully before you begin working on PA4. Some macros and helper functions are given, please do not modify them.

If you do not want to use the skeleton code, you are allowed to write the code from scratch. However, you are supposed to follow the input and output file format exactly, as the grading script works on the provided sample input and sample output files provided to you.

### 3.2 Submission

You should submit only *pa4.zip* on PASS, which only contains *memory_system.py*, under “PA4: Building a Cache Simulator”. **Please do not modify the file name or include unnecessary files in your submission.**

You can submit up to **20** times for PS4. We encourage you to use PASS only to verify your implementation, and not to use it for debugging purposes. You can choose which submission will be used for grading.

**You should remove or comment out unnecessary codes before you submit to PASS**, especially if it prints or generates log files. The automated grading program may fail to evaluate your assignment accurately.

### 3.3 Testing Locally

You can test your implementation locally with the following command:

```
$ python main.py input_trace_file
```

Your simulator should print the output with standard output. The output file should show cache contents and memory, as specified by the arguments above.

The functions for printing the memory and register values are provided in the *memory_system.py* file.

## 4. Grading Criteria

Your assignment will be graded with 8 testcases. Each testcase is worth 1 point, no partial points will be given.

For assignment4, due to the time limit of grading the final scores, no late submissions are allowed. Please submit within the time limit.

## 5. Updates/Announcements

We will post a notice on BlackBoard and Pizza if there are any updates to the assignment, please check BlackBoard and Piazza regularly.

Note that you are allowed to import and use external packages for this assignment. If you think it is necessary to use an external package, you must first get permission from the TAs by posting in Piazza. Your post must explain clearly why the package you intend to use is necessary. **Using an external package without permission will be regarded as plagiarism.**

## 7. Misc

Be careful about plagiarism! Last semester, we found a couple of plagiarism cases through an automated tool. If you are caught in “deep collaboration” with other students, you will split the score equally with your collaborators.
**Please do not upload your code when posting in Piazza as it is considered cheating.** If you think your question contains hints to other students, please make sure you post it as a “private post”.

If you have any requests or questions (technical difficulties, late submission due to inevitable circumstances, etc.), please ask the TAs on Piazza.

We generally encourage the use of Piazza for discussions. However, for urgent issues, you can send an email to the TAs (minseok1335@unist.ac.kr(Head TA) / dyryu@unist.ac.kr / xinyuema@unist.ac.kr / garvel@unist.ac.kr ).