# CS203 Fall '16 - Lab 3

Due: Thursday, November 17 midnight (11:59:59 pm)
All projects will be turned in to iLearn.

## 1. Objective

The goal of project 3 is to design and implement a simple cache, with support for direct-mapped, set-associative, and fully-associative mapping.
If you have any questions, please post to Piazza as others may be having the same issue.

## 2. Memory Access Traces

To facilitate the testing of your cache simulator, we will utilize two sets of traces collected from a run of gcc. The traces `gcc-10K.memtrace`, and `gcc-1M.memtrace` are available at the following git repository: https://bitbucket.org/danwong/cs203-labs-f16 under the Cachesim subdirectory. The traces contain ~10 thousand and ~1.5 million entries. A sample of the trace is given below:

```
L -200 7fffe7ff088
L 0 7fffe7fefd0
S 8 12ff228
S 8 12ff208
L 0 a295e8
```

Each line in the trace file contains the memory instruction type (L = load, S = store), the offset, and the memory address in hexadecimal. *Note that this trace was obtained from an x86 machine, and thus the memory address is 44-bits. For the purpose of this class, we can assume 32-bits and you can truncate the most significant 12 bits.*

## 3. Cachesim

Now that you have a better idea of how simulators are implemented, this project will be a bit more open ended. You will implement from scratch a C++/Java/Python based cache simulator (let's call it Cachesim). Cachesim is a performance simulator, with the main focus on collecting cache miss and cache hit rates. Your simulator must meet the following requirements.

a) Command line argument for input trace file name
b) Command line argument for cache configuration
    a. The cache configuration will take several parameters including:
        - Total cache size in bytes
        - Cache block size in bytes
        - Number of ways
c) Initialize all entries, tag, and LRU-bits of the cache to 0's.
d) Output the cache miss and hit rate, as well as the number of sets, ways, and number of address bits for the tag, index, and offset.

For sample code of how to read a file, refer back to the pipesim source code.

For simplicity, we can make the following assumptions:

a) Maximum of 4MB cache size
b) Assume Least Recently Used (LRU) cache replacement policy

Please include a printout of your simulation run when executing `gcc-10K.memtrace` and `gcc-1M.memtrace`. In addition, ***include documentation on how to compile and run your code***. Submit your source code, your printout, and answers to questions to iLearn in a single zip file.

## 4. Questions

1. Assuming a 512KB 4-way set associative cache with 16B block size, how many bits does the tag have? What is the total size, in bytes, of the cache including tag bits?

2. What is the cache miss rate of the given traces and cache configuration? Assume we have a 512KB cache and 16B block size.

| *Trace* | Direct | 2-way | 4-way | Fully Assoc. |
|---|---|---|---|---|
| *gcc-10K* | | | | |
| *gcc-1M* | | | | |

3. For the following configurations, how many bits are for tag, index, and offset fields? Assume we have a 256KB cache and 16B block size.

| *Trace* | Direct | 2-way | 4-way | Fully Assoc. |
|---|---|---|---|---|
| *gcc-10K* | | | | |
| *gcc-1M* | | | | |

4. Assuming a 256KB cache and 32B block size. How does increasing the number of ways affect cache miss rate? Plot the number of ways (1,2,4,8,16) vs miss rate for the two traces. What do you observe and why?

5. Assuming a 256KB 2-way set associative cache. How does varying the size of the block affect cache miss rate? Plot the block size (2B, 4B, 8B, 16B, 32B, 64B) vs miss rate for the two traces. What do you observe and why?

6. Assuming a 2-way set associative cache and 32B block size. How does varying the size of the cache affect cache miss rate? Plot the cache size (64K, 128K, 256K, 512K, 1M, 2M) vs miss rate for the two traces. What do you observe and why?