ECE/CS 472/572 – Computer Architecture
Instructor: Prof. Lizhong Chen
Lab Assignment

Part I. Instructions

1. Basic info

a. **Due: Wednesday, 6/5 at 8:30am, on paper.**
b. Up to 2 students per team are allowed.
c. If you didn't or couldn't redeemed Google Cloud credits, please team up with a student who redeemed successfully, or use your own computers. You don't have to use Google Cloud (e.g., Bash on Ubuntu on Windows also works).
d. Using university servers might not be a good option as "sudo" is not supported on those servers.
e. One final report per team, submitted in hardcopy.
f. Late penalty: 20% each 24 hours.
g. For any technical questions regarding this lab, please contact the TAs.

2. Setup Google Cloud

a. Claim the google credits as guided
   - Below is the URL you will need to access in order to request a Google Cloud Platform coupon. You will be asked to provide your school email address and name. An email will be sent to you to confirm these details before a coupon is sent to you.
   https://google.secure.force.com/GCPEDU?cid=nodpwuS0R4iG1wx42yWSqj7P%2FmAlk8KAMwp115epHI32SP6XgafwBxegAoLI9tVn
   - You can request a coupon from the URL and redeem it until 8/1/2019 and the Coupon is valid through 4/1/2020; You can only request ONE code per email
b. In top-left google cloud navigation menu, click "Compute Engine"
c. Click "create instance" after the button is ready
d. Enter "Change" under the "Boot Disk" Box
e. Select OS image "Ubuntu 14.04 LTS", then click select
f. Click create to create the instance.
g. After creating the instance, click "SSH" to log in this instance.

3. Setup environments for installation of Simplescalar

a. Enter "sudo bash", then press "enter". (If you are not familiar with Linux, you'd better enter this command each time when you log in)
b. Enter "apt-get install git", then press "enter".
c. Enter "git clone https://github.com/gyb1325/lab_setup"
d. cd lab_setup

e. ./Install-SimpleScalar.sh
f. After executing the script above, the simplescalar and its dependence will automatically be installed in the root path.
g. cd ~/SimpleScalar/simplesim-3.0
h. (Optional but helpful) Change configuration to PISA by typing the following three commands in a sequence (under the simplesim-3.0 path):
    i. make clean
    ii. make config-pisa
    iii. make
i. When you want to change configuration to Alpha, just execute the same command sequence except that in the second command replace config-pisa by config-alpha

4. Try to run test benchmarks with simplescalar

a. cd <path to simplesim-3.0>
b. ./sim-outorder -config config/default.cfg tests/bin.little/test-math

Part II. Tasks and Requirements

This lab will let you become familiar with the sim-cache component of the popular SimpleScaler simulator that emulates a system with multiple levels of instruction and data caches. Each cache can be configured for different sizes and organizations. You will use this simulator to do cache simulation with various configurations.

Get help: Under simplesim-3.0 directory, type the following to seek help about sim-cache:
./sim-cache –h

Note: Make sure that the simulator is configured as **PISA** Simulator, and perform the following exercises on **test-math** program, which is available in /simplesim-3.0/tests-pisa/bin.little/ directory. In Part II and Bonus part, we only use “sim-cache” rather than “sim-outorder”

1) Use sim-cache to simulate the performance of cache under the following conditions:

- least-recently-used (LRU) replacement policy
- 32 to 512 sets
- 1-way to 8-way associativity
- 16-byte cache lines (block size)

Do this twice, one for a data-only cache and another for an instruction-only cache.
Please download the configuration file **cache_1a.cfg** and **cache_1b.cfg** in “https://github.com/gyb1325/lab_config” and modify them to conduct the following experiments. Fill in the tables 1 and 2 below.

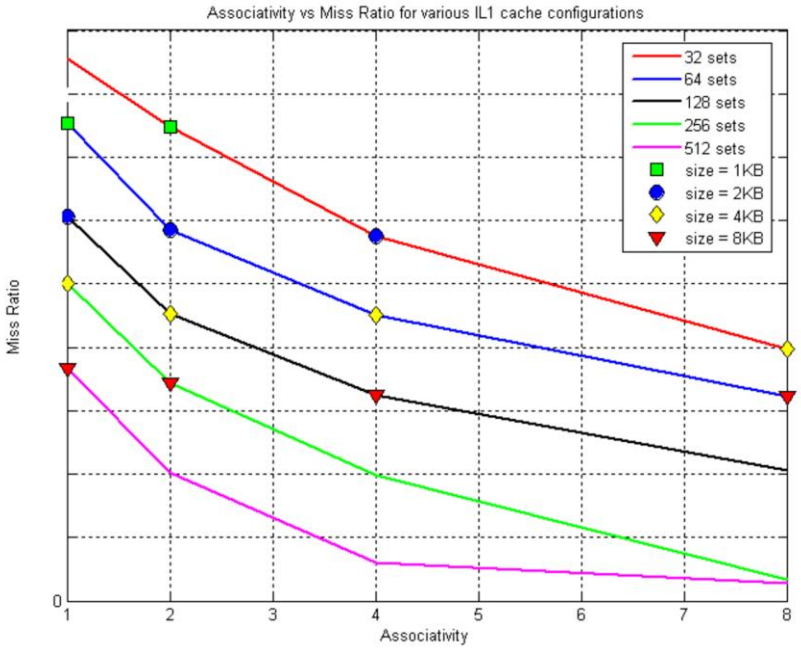
Table I

Miss Ratio (I-Cache)	1-way	2-way	4-way	8way
16 sets				
32 sets				
64 sets				
128 sets				
256 sets				
512 sets				

Table II

Miss Ratio (D-Cache)	1-way	2-way	4-way	8way
16 sets				
32 sets				
64 sets				
128 sets				
256 sets				
512 sets				

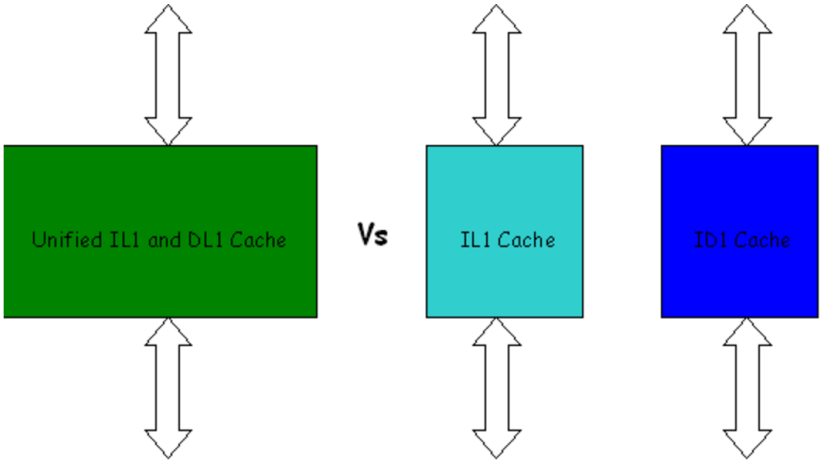
Now use MATLAB, EXCEL or python to plot the results of the simulations. For each of the simulations (data, instruction), plot the miss ratio versus associativity for each number of sets. Using markers, show the points on the curves which correspond to total cache sizes of 1 Kbytes, 2 Kbytes, 4 Kbytes and 8 K bytes (total cache size = sets * block size * associativity). For each simulation, you should produce something that resembles the plot below. In the plot results are shown for the instruction cache simulation.



Please answer the following questions based on the above results.

a) For a given number of sets, what effect does increasing associativity have on the miss ratio?
b) For a given associativity, what is the effect of increasing the number of sets?
c) For a given cache size, how does the miss ratio change when going from an associativity of one to two to four? Explain.
d) If you were to design an Instruction cache, limited to a total cache size of 4 Kbytes, which cache organization would you choose, based solely on performance?
e) If you were to design a data cache, limited to a total cache size of 4 Kbytes, which cache organization would you choose, based solely on performance?

2) Investigate the performance of cache with unified/separate instruction and data caches (see Figure 1 below) while executing the test-math program.



(Fig 1)

Use sim-cache to simulate the performance of the cache under the following conditions:

- least-recently-used (LRU) replacement policy
- 16-byte cache lines (block size)
- Sets/associativity combinations: 128/1, 128/2, 128/4, 2048/1, 2048/2, 2048/4
- Only L1 caches will be used in this problem; L2 caches will be assumed not present

To do this, **you need to specify a split cache configuration** that has instruction and data caches that are half the size of the unified cache, for each of the prescribed sets/associativity combinations (e.g. a 128-set unified configuration would be compared to a 64-set instruction/data caches in a split configuration).

To compare the performance of the split cache to the unified cache, one must compute the overall miss ratio of the split cache. To do this, one must take note of the total number of accesses to each of the instruction and data caches, and the number of misses. The simulator will give you the miss rates for the instruction and data caches, which should be **(instruction misses)/(instruction accesses)** and **(data misses)/(data accesses)** respectively. The effective combined cache miss ratio can be computed as: **(instruction misses + data misses)/ (instruction accesses + data accesses)**. The number of accesses to each cache should be the same for each configuration; you need only record this once. The number of misses must be recorded for each configuration. Fill Table 1 below.

Table I

Sets/Assoc	I-Cache Miss No (N/2 sets)	D-Cache Miss No (N/2 sets)	Effective Combined Cache Miss Rate	Unified Cache Miss Rate
128/1				
128/2				
128/4				
2048/1				
2048/2				
2048/4				

The number **N** is the number of sets in the unified cache, the split cache consists of two caches of N/2 sets.

3) Bonus Points

You need to modify the C/C++ source code in **sim-cache.c** file to add a third-level data cache and a third-level instruction cache. After you modify the code, recompile it using command:

```
make
```

you will get new simulation tool: "sim-cache" with extended functionality. You need to use the two provided configuration files (cache_3a.cfg and cache_3b.cfg: also in the previous github repository) on test-math to test if your implementation is correct. Use the command that is similar to the following command.

```
./sim-cache -config cache_3a.cfg -redir:sim cache_3a.out ./test-math
```

Hints:

some related functions in sim-cache.c:

```
sim_reg_options(struct opt_odb_t *odb): /* register simulator-specific options*/
sim_check_options(...): /* check simulator-specific option values */
sim_reg_stats(struct stat_sdb_t *sdb): /* register simulator-specific statistics */
dl1_access_fn(...): /* l1 data cache l1 block miss handler function */
dl2_access_fn(...): /* l2 data cache l2 block miss handler function */
il1_access_fn(...): /*l1 inst cache l1 block miss handler function */
il2_access_fn(...): /*l2 inst cache l2 block miss handler function */
sim_main(void):/*start simulation, program loaded, processor precise state initialized */
```

You should describe how to implement a level-3 instruction cache and a level-3 data cache in SimpleScalar. You should also attach the modified source code : sim-cache.c, and mark the place where you make modification. Run simulations using provided simulation configuration file(cache_3a.cfg and cache_3b.cfg), for each run you should submit one output file.