CS211 Fall 2016
Programming Assignment 4: Cache Simulator
Due: 11:55 PM, Tuesday Nov. 30

1- Overview

This assignment is designed to give us a better understanding about cache behavior. You will write a cache simulator using C programming language. **The programs have to run on iLab machines**. It's also very important to follow the exact format as it's explained in the description. Otherwise, you may lose up to **20%** if your submission doesn't follow the instructions.

2- Understanding the Memory Access Traces

We have provided you with **three** different memory *trace files* (mytrace.txt, trace1.txt and trace2.txt). These traces are the memory access (address) pattern of a program. By knowing the address of the memory locations during program execution you will be able to write a cache simulator. These two traces correspond to the sequence of memory accesses that are generated by the processor as it executes the stream of instructions from a particular program. For example, suppose the following:

Currently, %*esp* = 0*x*00*ffff*40, and the processor executes the instruction:

pushl %ebp

Remember that the *pushl* instruction will copy the content of some register (in this case %*ebp*) to the memory location whose address is currently contained in %*esp*. (It also decrements %*esp* by 4, but that is not relevant to the memory system). This will result in the processor generating a W to address 0*x*00*ffff*40. So these traces consist of 3 columns. First one is **ip** (instruction pointer), after that you can see **R** or **W** which means it is memory read or write and third column is the **memory address**.

Note: In this assignment, you only need the memory address and R or W information. You do **not** need to keep or use ip addresses.

3- Cache Simulator

You will implement a Cache Simulator to evaluate different configurations of Caches, running it in different traces files. The followings are the requirements for the simulation:

1. Simulate only **one level** cache: L1
2. The size of the cache, associativity and blocksize are parameterizable.

3. Replacement algorithm: ***FIFO*** https://en.wikipedia.org/wiki/Page_replacement_algorithm#First-in.2C_first-out or ***LRU*** https://en.wikipedia.org/wiki/Cache_replacement_policies#LRU

4. Implement both **write through and write_back** cache policies.

3-1 Invocation Interface

Implement a program **c-sim (the name of your binary file should be exactly this)** that will simulate the operation of a cache. Your program c-sim should support the following usage interface:

```
./c-sim  <cache size> <associativity> <block size> <replacement_policy> <write_policy> <trace file>
```

where:
A) $< cachesize >$ is the total size of the cache. This should be a **power of 2**. Also, it should always be true that
$< cachesize > =$ number of sets $\times < setsize > \times < blocksize >$.

For direct-mapped caches, $< setsize > = 1$. For $n - way$ associative caches, $< setsize > = n$.
Given the above formula, together with $< cachesize >$, $< setsize >$, and $< blocksize >$, you can always compute the number of sets in your cache.

B) $< associativity >$ is one of:
– direct - simulate a direct mapped cache.
– assoc - simulate a fully associative cache.
– assoc:n - simulate an $n - way$ associative cache. n should be a power of 2.

C) $< blocksize >$ is an power of 2 integer that specifies the size of the cache block.

D) <replacement_policy> is the *LRU* or *FIFO*.

E) <write policy> is “wt” for write-through cache and “wb” for write-back cache.

F*)* $< tracefile >$ is the name of a file that contains a memory access traces.

Your program should print out the number of memory reads, memory writes, cache hits, and cache misses. **You should follow** the format of the examples below (**order of outputs**, style and etc.).

Below is a small sample of inputs and their correct outputs. But, they are not enough to show that your code works perfectly for all test cases we’ll use for grading. So, we encourage all of you to come up with more tricky test case, share them with your friends and compare your results and make sure your program gives correct answer for all of them. (you can use Sakai chatroom for communication)

```
./c-sim 4  direct  1 FIFO  wt mytrace.txt
Memory reads: 5
Memory writes: 3
Cache hits: 1
Cache misses: 5

./c-sim 4  assoc:2  1  FIFO  wt mytrace.txt
Memory reads: 3
Memory writes: 3
Cache hits: 3
Cache misses: 3

./c-sim 4  assoc  1  FIFO wt mytrace.txt
Memory reads: 3
Memory writes: 3
Cache hits: 3
Cache misses: 3

./c-sim 4 assoc:2 1 LRU wt mytrace.txt
Memory reads: 4
Memory writes: 3
Cache hits: 2
Cache misses: 4

./c-sim 4 assoc:2 1 LRU wb mytrace.txt
Memory reads: 4
Memory writes: 2
Cache hits: 2
Cache misses: 4
```

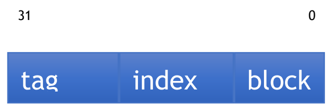
3-2 Simulation Details

1. (a) When your program first starts running, all entries in the cache should be **invalid**; (b) you can assume that the memory size is 2^{32} . Therefore, memory address length is **32 bit**. (c) the number of bits in the tag, cache address, and byte address are then determine by the cache size and block size; (d) Your simulator should simulate the operation of a cache according to the given parameters for the given trace; (e) at the end, it should print out the number of cache hits, cache misses, memory reads and memory writes.
2. For *write misses* , assume the data brought in to the cache from memory (one *read memory) before write take place*.

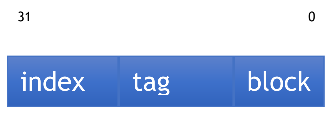
4- Understanding the locality of reference

To capture the locality of addresses, you always use bits at *most significant* position as a *tag bits.* Now, we want you to test how the caches behave in term of cache hit/miss rate when you tweak the order and report your observation. To do that, assume we have a 2-way associate cache of size 32, cache block size is 4. Report cache miss and hit ratio for following two scenarios. (report your results for all different cache policies LRU or FIFO and write through or write back):

1) *Tag* bit is in most significant position and *then you have index* bit and block bits, like below.



2) *Index* bit are in most significant position and then you have *tag* bits in between the block bits and index bit, like below.



Which one of above caches gives better hit rate and why? Explain your observation and thoughts on this in the **report.**

5- Submission

You have to e-submit the assignment using **Sakai**. Put all files (**source code + Makefile + report)** into a **directory** and name it **pa4**. Your report should clearly answer part 4 questions. You then need to create a tar file (follow the instructions in the previous assignments to create the tar file). Your submission should be **only** a **tar** file named pa4.tar. You must submit it then on Sakai before the deadline.