CS211 Fall 2015
Programming Assignment 5: Cache Simulator
Due: 5 PM, Friday Dec. 11

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**.

2- Understanding the Memory Access Traces

We have provided you with **two** different memory *trace files* (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 **memory address**. (Open up the trace file and check that out)

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
4. Implement **write through** cache.

3-1 Invocation Interface

Implement a program **c-sim (the name of your binary file should be exactly the same)** 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> <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 × < *setsize* > × < *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) < *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.).

Note: The result given for each test case are the "right" answers for that set of inputs. So, you can test your program with the same set of inputs and then try to get the same results.

```
./c-sim 4  direct  1  mytrace2.txt
Memory reads: 6
Memory writes: 3
Cache hits: 0
Cache misses: 6

./c-sim 4  assoc:2  1  mytrace2.txt
Memory reads: 4
Memory writes: 3
Cache hits: 2
Cache misses: 4

./c-sim 4  assoc  1  mytrace2.txt
Memory reads: 3
Memory writes: 3
Cache hits: 3
Cache misses: 3
```

./c-sim 32 assoc:2 4 trace2.txt
Memory reads: 3499
Memory writes: 2861
Cache hits: 6501
Cache misses: 3499

./c-sim 32 assoc 4 trace2.txt
Memory reads: 4204
Memory writes: 2861
Cache hits: 5796
Cache misses: 4204

./c-sim 32 direct 4 trace2.txt
Memory reads: 3503
Memory writes: 2861
Cache hits: 6497
Cache misses: 3503

./c-sim 16 assoc:4 4 trace1.txt
Memory reads: 501
Memory writes: 334
Cache hits: 499
Cache misses: 501

./c-sim 16 assoc 4 trace1.txt
Memory reads: 501
Memory writes: 334
Cache hits: 499
Cache misses: 501

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 a write-through cache, there is the question of what should happen if a write generated by the processor results in a cache miss. For this assignment, your cache simulator should bring the corresponding block in (*read memory)*, then execute the write (write to memory). Future reads or write to any location in the newly brought in block will hit in the cache until the block is evicted because of replacement.

4- Submission

You have to e-submit the assignment using **Sakai**. Put all files into a **directory** and name it **pa5**. Then, 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 pa5.tar. You must submit the following files in your submission.

• **Makefile**: there should be at least two rules in your Makefile:
– c-sim build your sim executable.
– clean prepare for rebuilding from scratch.

• **Source code**: all source code files necessary for building c-sim. You are recommended to put any global definitions and function declarations in the header file, while put function definitions in the source file.
We will compile and test your programs on the iLab machines so **you should make sure that your programs compile and run correctly on these machines**.