**IMPORTANT LAB 3 NOTES**

* **To compile** csim.c **type:**linux> make clean  
  linux> make
* **Command-Line Arguments** (to be written into csim.c)
  + -h: Optional help flag that prints usage info
  + -v: Optional verbose flag that displays trace info
  + -s <s>: Number of set index bits (S = 2­­­s is the number of sets)
  + -E <E>: Associativity (number of lines per set)
  + -b <b>: Number of block bits (B = 2b is the block size)
  + -t <tracefile>: Name of the valgrind trace to display
* **Command line arguments are based on the *s*, *E*, and *b* notations from p. 617 of the textbook**(Note: For whatever reason the lab instructions say p.597, which doesn’t seem to match my textbook at all. P.617 actually has the table summarizing the parameters)
* **MEMORY TRACE FORMAT**
  + [space]operation address,size
* **EXAMPLE INPUT:**  
  linux> ./csim-ref -s 4 -E 1 -b 4 -t traces/yi.trace  
  hits:4 misses:5 evictions:3
* **EXAMPLE INPUT (With verbose ‘-v’ flag)**linux> ./csim-ref -v -s 4 -E 1 -b 4 -t traces/yi.trace

L 10,1 miss

M 20,1 miss hit

L 22,1 hit

S 18,1 hit

L 110,1 miss eviction

L 210,1 miss eviction

M 12,1 miss eviction hit

hits:4 misses:5 evictions:3

* **LAB CRITERION**
  + Include your name, ID and user account (login ID) in the header comment for csim.c
  + Your csim.c file **must compile without warnings** in order to receive credit
  + Your simulator must work correctly for arbitrary *s*, *E*, and *b*
    - This means that you will need to **allocate storage for your simulator’s data structures** using the malloc function
    - Type “man malloc” for information about this function
  + For this lab, **we are interested only in data cache performance**, so your simulator should ignore all instruction cache accesses (lines starting with “I”)
    - Recall that valgrind always puts “I” in the **first column** (with no preceding space), and “M”, “L”, and “S” in the second column (with a preceding space). **This may help you parse the trace**.
  + For this lab, you should **assume that memory access are aligned properly**, such that a single memory access never crosses block boundaries. By making this assumption, **you can ignore the request sizes in the** valgrind **traces**.
* **LAB EVALUATION**
  + The reference simulator csim-ref can be used to obtain the correct answer for each of these test cases
  + For each test case, **outputting the correct number of cache hits, misses and evictions will give you full credit for that test case**.
    - **Each of your reported number of hits, misses, and evictions are worth 1/3 of the credit** for that test case. That is, if a particular test case is worth 3 points, and your simulator outputs the correct number of hits and misses, but reports the wrong number of evictions, then you will earn 2 points.
  + There are five test cases running for evaluation, and **each worth 3 points**. They are randomly chosen from below:  
    linux> ./csim -s 1 -E 1 -b 1 -t traces/yi2.trace

linux> ./csim -s 4 -E 2 -b 4 -t traces/yi.trace

linux> ./csim -s 2 -E 1 -b 4 -t traces/dave.tracei  
linux> ./csim -s 2 -E 1 -b 3 -t traces/trans.trace

linux> ./csim -s 2 -E 2 -b 3 -t traces/trans.trace

linux> ./csim -s 2 -E 4 -b 3 -t traces/trans.trace

linux> ./csim -s 5 -E 1 -b 5 -t traces/trans.trace

linux> ./csim -s 5 -E 1 -b 5 -t traces/long.trace

* **ADDITIONAL TIPS**
  + **Do initial debugging on small traces**, such as traces/dave.trace
  + The reference simulator takes an optional -v argument that enables verbose output, displaying the hits, misses, and evictions that occur as a result of each memory access. **You are not required to implement this feature in your** csim.c **code, but *we strongly recommend that you do so***. It will help you debug by allowing you to directly compare the behavior of your simulator with the reference simulator on the reference trace files.
  + We recommend using the getopt function to parse your command line arguments. **You’ll need the following header files**:  
    #include <getopt.h>

#include <stdlib.h>

#include <unistd.h>

* + **Each data load (L) or stores (S) operation can cause at most *one cache miss***. The data modify operation (M) is treated as a load followed by a store to the same address. **Thus, an M operation can result in the *two cache hits*, or a *miss and a hit* plus *a possible eviction***
  + If you would like to use C0-style contracts from 15-122, you can include contracts.h, which we have provided in the handout directory for your convenience (I have no idea what any of this means)
  + **When finished, submit the tarball to Blackboard** (see sec. 7 in the lab instructions)
* ‘IMLS’ – information for analyzing data from the trace
* L = load, s = store, m = modify…
* Check whether you have access for loading to that location
* Request size is *important*, cache size can be seen – you want to try to access cache.
* What about actual data in the cache? ***We don’t care!***
* How can you know that we’ve cached before? ***Valid tag!***
* Validity matches, tag is different? Cache information.
* Focus solely on ***address*** and ***size*** of the cache!
* Upload source files to Blackboard!