PA1
Darron Li
11836410

How I Implemented PA1

Data Structures: I utilized 3 different data structures, a struct for Line, a struct for Set, and a struct for the actual Cache.
Line: Represents a cache line with a validity bit, a tag, and an LRU counter.
Set: Contained a pointer of Lines
Cache: Contains the array of sets, configuration values, and statistic counters necessary for the assignment.
Each data structure is allocated dynamically using calloc, instead of malloc since it automatically initializes the entire block of memory to 0. This removed some redundancy and guaranteed that all sets and lines started out empty.

Understanding Caches: I then had to fall back onto my notes to understand addresses. Following the formula from lecture to figure out block offset and set index, Since we are modifying bits, we can utilize bit masking and bit shifting to calculate our set index and block offset. Everything else (everything above s + b) is set to our tag.

Simulation Process: For each trace, the simulator parses the operation, address, and size. I wrote a helper function so that my void run() is simpler and doesn't do all the work. It also helped me visualize what actually happens when we run. We ignore I operations (according to the document). For load and store, we perform one cache access, and check if we hit or miss. On a modify, we perform two accesses, and the second access is always a hit if the first brings the block into the cache.

Helper Function (access) logic: We increment the global time tick in order for our LRU replacement to work. It tracks whenever a line was last accessed and we increment it on the start of every access, and assign it if a line was successfully accessed. We also check whether it's in the target set, and if a valid line also matches the tag it is a hit. We increment our hit counter and update its LRU counter. Otherwise, no hits occur and we increment the miss counter. If there is an invalid line, we place the block there and make it valid. However, if all lines are valid, we perform an eviction and remove the smallest LRU counter, and replace it with the new block.

Conclusion: Putting all these pieces together, in my main I used the sample code given in order to parse out the input from the parameters. We then convert it from ASCII value into integer values, and create our cache after the loop.