Josiah Norton

I started my assignment by making sure that the given code worked. I did this by making and initializing global variables for hit count, misses, and evictions. After I made sure that the original code worked, I started by making structs for the elements of the dynamically nested array of structs I would need to build. The first struct is set up to represent the cache line with a value, tag and the last used tick for least recently used. I chose an int for the value and unsigned long's for the other two because the value just needs the one bit to be either 0 or 1, the tag out needs to hold lots of long address tags, and the last used tick could grow very large for larger time counters and a larger bit amount may prevent an overflow.

The next thing I did was construct my make_cache, and initialize the fields in it to their respective flags which we were given in the original code. c.nS = (1UL <<s); is how I calculate the number of sets, since s is the set index bits, 1UL is the number 1 set as an unsigned long, which gets shifted left by s positions. I believe this is the equivalent of multiplying it by 2^s. Storing Ns in the cache allowed me to not recalculate the 2^s later. Next, I used calloc to allocate the set array by using the stored nS to gie me nS zero initialized structs, which I believe is the safest default. Next, I allocate each set's line array, using calloc once again, setting the valid tag, and ticks to 0. I also check twice to make sure that calloc succeeded, throwing an error if it failed. I chose to use calloc over malloc, because using malloc would require me to loop through each line and manually set all of their fields to zero. I discovered this may work better than just malloc through the "man malloc" then googling what some of the functions in there were used for.

The next thing I did was make a destructor for the cache, this helps guard against nullpointers, and because of freeing inner-outter arrays, nulls the top level pointer last, avoiding dangling pointers. This was also something I was taught to do in my other CS classes, so it is something I made. After making both of these functions I added an instance of making a cache name and freeing a cache to my main, where I tested to make sure it would print out all zero's, I had to resolve a few bugs such as accidentally setting my unsigned long i to 1 instead of zero, but it worked after I fixed them.

To begin the next step I made my global which I would need for Least Recently Used, which is how I am tracking the evictions as requested. When making my modify cache I initialize evictions to 0. I then use my 1UL to shift by s positions removing the block offset bits, and make a group of bits which allows me to isolate the set index bits. After this I get the tag bits from the address, essentially this process allows me to do (tagbits = m -(s+b)). I then select the set I need from the array of sets and set the pointer to scan for a hit. After this process I use a for loop to scan E lines, which will check for a hit and make sure the tag matches. Each time I hit, I increment the ticks and set the field of the ticks, returning 1 if we get a hit.

If I do not get a hit then I increment my misses, and make some local variables to track my least recently used lines. Whenever we have a line with a smaller timestand stored in t, then I update these variables. My for loop records the first empty slot we come across and tracks the least recently used line. After the loop, if we found an empty line we fill it, otherwise if the set is full, we use the least recently used to evict the line and increment my evictions. Once this is finished I install the new line and update its time stamp, and return a 0 to show that we had a miss. After implementing these, I tested my mod_cache by installing one line, then trying to install the same line so I would have a hit and a miss without an eviction.

Next I made my do_trace function keeping in mind to ignore instruction command access. The function starts by opening a file to read, and printing an error if this fails. I found Clion very helpful with

making my error messages, since it has some defaults which save you typing. I then make two buffers to use with sscanf one to temporarily store the line, and one for each of the operation types. I also made an unsigned long for the hex memory address, and an int size for the decimal number of bytes for the access. My while loop reads the trace file line by line until I reach the end of file. Next is the if to skip instruction loads. Then I parse for L/S/M and get rid of the leading white space that may be in the line. Lines which don't match what we are looking for are ignored. I then make a local eviction flag and echo the input fields. Once we go into the if op==m, we use mod_cache and if it finds the line it is a hit, if it does not it is a miss. We will only set evictions to 1 if the set was full and we had to evict a line. We put whether we got a hit, a miss, or a miss and an eviction. After this runs, I close the file.

After I had finished all of these, I put my new functions into main, and added the files which the TA's would be testing and ran the test myself. Here are my results.