Name:\_\_Xiang, Xin\_\_ Date: \_\_\_\_ \_May 9, 2020\_ \_ \_

Section: \_ 005\_ \_ \_ Github Username: \_\_FreyaXiang\_ \_

**Lab 5**

Total in points (100 points total):

Professor’s Comments:

Honor Pledge: I have neither given nor received aid on this assignment.

Signature: Xin Xiang

Part A:

The goal is to build a cache using LRU rules. Since we can ignore the request sizes, each cache block only needs to contain 3 integers:

1. Valid bit. If a block is empty, the valid bit is 0. After being used, the valid bit is set to 1.

2. A number used to trace the using order for LRU. The least recently used block in each set is 1, and the most recently used block in each set is E (the associativity number).

3. Tag.

After retrieving E, b, s, S(since S=2^s, we can easily compute what S is) and the file name from the command line using getopt(), we use malloc() to allocate sizeof(int)\*S\*E\*3 bytes of memory in the heap. This is because we have S\*E cache blocks, and each cache block contains 3 integers.

Then we begin to read from the file according to the file name. We read one line at a time. For each line, we store the instruction (L, S or M) into the variable mode and convert addresses from hex string into int. Then we compute the tag and index number based on the address we get and s, b.

After we get the index and tag, the cache simulation begins.

First, we go to the specific set according to the index number and use a for loop to iterate through all the cache blocks in this set. If the valid bit of the cache block is 0, then we continue. If not, compare the two tags. If the tags are the same, then there is a hit! Update the LRU bits to E and break the loop.

If we iterate through the blocks in the specific set but do not hit, then there is a miss. First, check if there are any empty blocks in this set. That is, check if there are any cache blocks with valid bit 0. If we have one, then put the tag into that block, update the LRU bit to E and set valid bit to 1. If we cannot find one, that means we need to evict other blocks.

To find which block to evict, just check which cache block in this set has LRU bit value 1. Replace the tag to that block, update the LRU bit to E, and decrement other LRU bits.

If the mode is L or S, we only need to do the simulation once. If the mode is M, do twice.

Part B:

The technique I use for part B is blocking.

For 32 by 32: I set the block size to 8 and used 4 loops. The outer two are used to iterate across each block. The two inner loops are used to iterate through each element in the block. We need to be more careful when hitting diagonal elements (when i == j).

For 64 by 64: This case is special. Set block size to 4. And instead of using 4 loops, I used 2 loops to reduce misses. And then for each i, A[i][], A[i+1][], and A[i+2][] are set to the corresponding variables.

For 61 by 67: In this case, the block size is 16. Again, use similar code (4 loops) as we implemented the 32 by 32 matrix. The only difference is that because this is not a square matrix, block increment may exceed size of matrix A. Thus, we need to check if i < N and j < M.