Lab 2: Cache Memory

Dr. Qing Cao
Jerome Vaz, Sam Neyhart, Daniel Barry
ECE 451 – Fall 2017

I. PROBLEM STATEMENT

In this lab we were asked to generate a cache simulator with configurable command line options. To do so, we first had to understand the concepts of associative cache, cache blocks vs words, as well as the random and least recently used replacement algorithms. Additionally, we were asked to calculate the hit/miss rate and the AMAT of each address given in the instruction file and see how these values were affected when changing different command line arguments such as the block size or the number of total blocks.

II. OPERATION OF CODE

We decided to write C code to implement our cache simulator. It emulates the blocks of cache using an array of a struct that we created called a cblock. A cblock, or *cache-block,* stores the block address of the memory it contains as well as an integer value called history which is used by the LRU (least recently used) replacement algorithm. The other replacement method required by the project description was the random replacement method, where we can randomly choose a block within a set to replace. Our program takes six command line arguments: block size, number of blocks, associativity, hit time (cycles), miss time (cycles), replacement policy (LRU, random) and uses those parameters to configure our cache simulator to the proper size. To configure our cache, the program first creates an array with size equal to the number of cblocks. Next it reads in the simulated memory addresses from stdin and attempts to place them in the correct cache set for the associativity we are using. Here associativity is defined as the number of blocks within each set. The correct cache set for a given memory address can be determined by the formula:

$$\frac{Word\ address}{\#\ of\ sets\ in\ cache}$$

where the number of sets in the cache is the number of blocks in the cache divided by the associativity.

$$\#\ of\ sets\ in\ cache = \frac{\#\ of\ blocks\ in\ the\ cache}{associativity}$$

Using the definition of block address: word address/size of block in words,

$$block\ address\ =\ \frac{word\ address}{size\ of\ block\ in\ words}$$

our program uses the following logic for adding a memory block to the cache:

- If a block address in the necessary cache set is empty the memory will be placed in this block.
- If a block address is already located in the correct cache set, nothing

needs to be done and this operation will be counted as a hit. This hit is then recorded in a global integer variable called total_hits.

- If neither of the above is true, then the cache set is full and an element of the set needs to be replaced. Our program uses LRU or random replacement as specified by the user in the command line.

Our program keeps track of the total number of memory accesses, a counter which is incremented in all three of the above cases, as well as the total number of hits, a counter which is only incremented in the second of the above scenarios. For the LRU replacement policy, we also keep track of the history, a variable that keeps track of when a block was last accessed and is reset every time that block is accessed or replaced. This variable is then used to decide which of the blocks to replace if a replacement is required. The replacement policy is decided by the user when the program is first run as either random or LRU. Random performs as follows: in the event of a cache miss, when there are no empty blocks in a given set, an element of the cache set will be selected at random and replaced with the new memory block. LRU will replace an element under the same circumstances but chooses the element which has the highest history counter, thus being the least recently used block, to replace.

After all input memory addresses have been simulated, the program calculates the following parameters: total inputs, hits, misses, hit rate, miss rate, and AMAT or *average memory access time*. We have kept track of the total number of hits so we can get the hit rate by simply using the following equation:

$$hit\ rate\ =\ \frac{total\ hits}{total\ reads}$$

Similarly, the miss rate can be calculated as follows:

$$miss\ rate\ = 1 - hit\ rate$$

Finally, we can calculate the AMAT using the following formula:

$$AMAT = hit\ time + (miss\ time\ *\ miss\ rate)$$

III. TESTING

In order to test our code, a function was written called print_cache which prints out the current memory blocks stored in the each block of cache. It will print out the block address 0x0000 if a block of cache is empty. We used this function to test the cache simulator by providing it with various word addresses and seeing if they were getting stored correctly by the cache. This allowed us to ensure that our simulator handles block size, number of blocks, associativity, as well as the replacement policy correctly. Additionally, we used the example scenario provided in “Testing Guide.pdf” and made sure that the results were similar. In our case, the results were identical.

The following is an explanation with results of our test cases from “Testing Guide.pdf”. First we tested the case for direct mapped, one word blocks, with 16 total blocks. When given word addresses 0, 3, 11, 16, 21, 11, 16, 48, 16 (we entered these addresses in hexadecimal values) we get the results shown in Figure 1. We see that our solution matches the professor's solution exactly. Next we tested the same word addresses but with 2-way associativity. The solution for this test is shown in Figure 2 below.

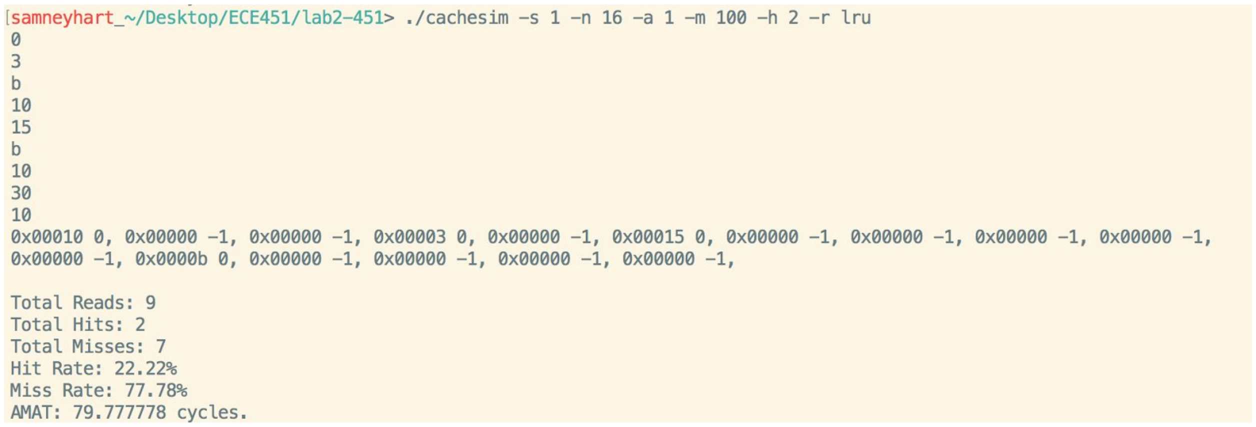
For step two of the testing process we will use a cache with 16 word blocks, with 256 total block, using the direct mapped method. We used the provided file "address.txt" to run this test. Next we changed the word blocks to 256 and ran it again with the inputs from "address.txt". Both of our results can be found below as Figure 3 and Figure 4 respectively.

Finally, to show that our cache simulator can handle a large hit count we have shown the results for an example containing 256 word blocks, with 256 total blocks, using full associativity and using the provided file, "addresses_high_hit.txt". The results of this example can be seen below as Figure 5. This is solely to show that our cache simulator is robust enough to handle a high hit count and a high associativity.

Please note that to input large files into our program we use redirection of stdin with the '<' character.

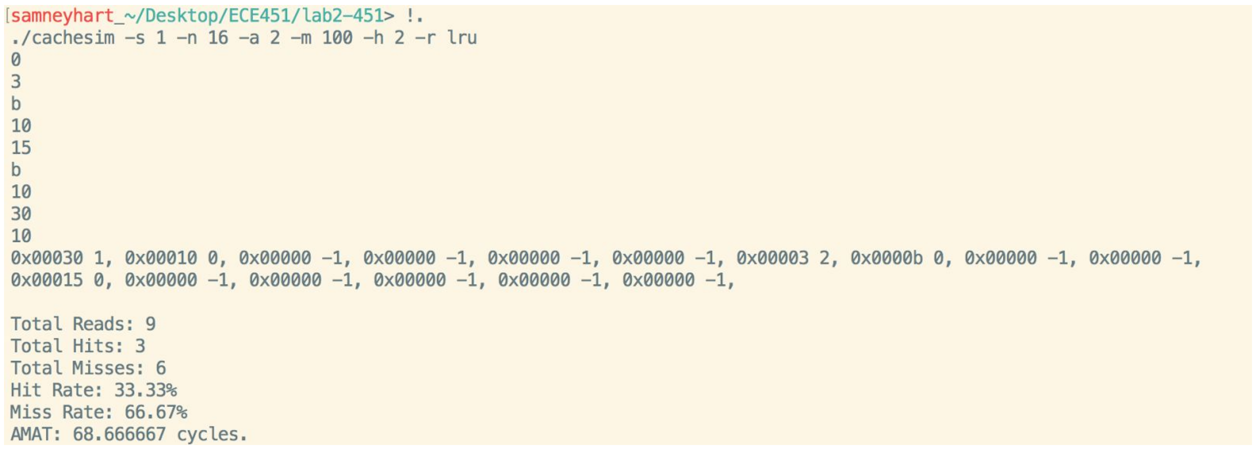
IV. CONCLUSION

In conclusion, this project has shown us the complexities of how a computer's cache operates. We see that the idea of word block compared to total blocks combined with the idea of associativity quickly makes keeping track of the cache very complex. However, once we can comfortably keep track of all of the different ideas involved, we see that the basic concepts of the cache are actually quite simple.



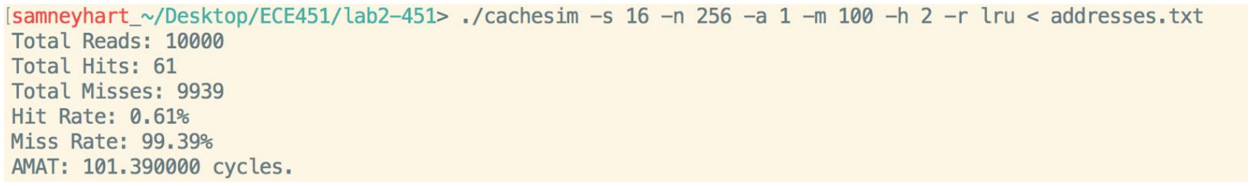
```
[samneyhart_~/Desktop/ECE451/lab2-451> ./cachesim -s 1 -n 16 -a 1 -m 100 -h 2 -r lru
0
3
b
10
15
b
10
30
10
0x00010 0, 0x00000 -1, 0x00000 -1, 0x00003 0, 0x00000 -1, 0x00015 0, 0x00000 -1, 0x00000 -1, 0x00000 -1, 0x00000 -1,
0x00000 -1, 0x0000b 0, 0x00000 -1, 0x00000 -1, 0x00000 -1, 0x00000 -1,

Total Reads: 9
Total Hits: 2
Total Misses: 7
Hit Rate: 22.22%
Miss Rate: 77.78%
AMAT: 79.777778 cycles.
```

Figure 1 - Step 1 Part a.



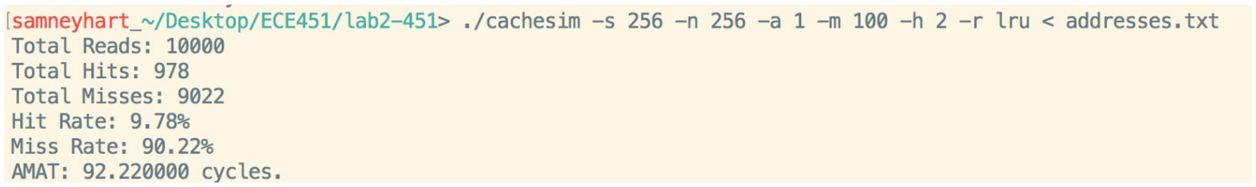
```
[samneyhart_~/Desktop/ECE451/lab2-451> !.
./cachesim -s 1 -n 16 -a 2 -m 100 -h 2 -r lru
0
3
b
10
15
b
10
30
10
0x00030 1, 0x00010 0, 0x00000 -1, 0x00000 -1, 0x00000 -1, 0x00000 -1, 0x00003 2, 0x0000b 0, 0x00000 -1, 0x00000 -1,
0x00015 0, 0x00000 -1, 0x00000 -1, 0x00000 -1, 0x00000 -1, 0x00000 -1,

Total Reads: 9
Total Hits: 3
Total Misses: 6
Hit Rate: 33.33%
Miss Rate: 66.67%
AMAT: 68.666667 cycles.
```

Figure 2 - Step 1 Part b.



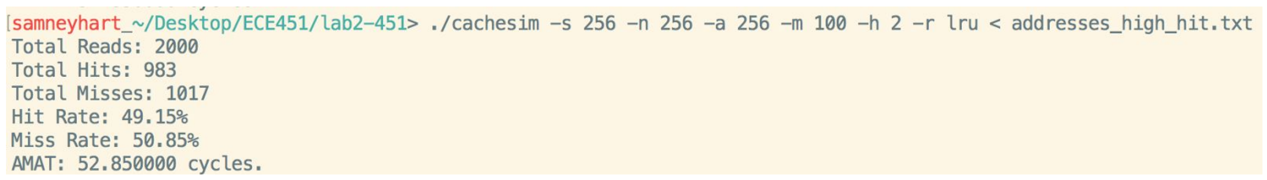
```
[samneyhart_~/Desktop/ECE451/lab2-451> ./cachesim -s 16 -n 256 -a 1 -m 100 -h 2 -r lru < addresses.txt
Total Reads: 10000
Total Hits: 61
Total Misses: 9939
Hit Rate: 0.61%
Miss Rate: 99.39%
AMAT: 101.390000 cycles.
```

Figure 3 - Step 2: 256 16 word blocks



```
[samneyhart_~/Desktop/ECE451/lab2-451> ./cachesim -s 256 -n 256 -a 1 -m 100 -h 2 -r lru < addresses.txt
Total Reads: 10000
Total Hits: 978
Total Misses: 9022
Hit Rate: 9.78%
Miss Rate: 90.22%
AMAT: 92.220000 cycles.
```

Figure 4 - Step 2: 256 256 word blocks



```
[samneyhart_~/Desktop/ECE451/lab2-451> ./cachesim -s 256 -n 256 -a 256 -m 100 -h 2 -r lru < addresses_high_hit.txt
Total Reads: 2000
Total Hits: 983
Total Misses: 1017
Hit Rate: 49.15%
Miss Rate: 50.85%
AMAT: 52.850000 cycles.
```

Figure 5 - 256 256 word blocks with full associativity using high hit input file