**Ayush Mahant**

**CSSS brach**

**2019353**

**Group 9**

**ENDSEM ASSIGNMENT**

**DOCUMENTATION- CACHE MEMORY**

INTRODUCTION-

I have done the bonus assignment in a different file but documented it here.

The file of the main assignment without the bonus assignment is named-

“AYUSHMAHANT\_2019353\_FINALASSIGNMENT.java”

And the bonus assignment is named “AYUSHMAHANT\_2019353\_BONUSASSIGNMENT.java”

Cache memory is used to decrease the time to access the main memory. So in this assignment, we have to design a cache implementation, such that storing values in the physical address takes place. We know about the main memory space, the cache space(or the number of cache lines) and the block size.

LANGUAGE USED-

Java is the language used in my cache implementation. It is easier in handling string manipulation and is currently being taught in our course.

ASSUMPTIONS-

1, All the inputs are in the power of 2 except the number of bytes for the main memory.

2. Everything is handled by the user where the input is on the terminal itself.

3. For writing it is supposed to be -

w

Physical address

Data

4. For reading it should be like-

r

Physical address

5. There is use of main memory just for the replacement part as I thought it would be better for the data part too.

6. I don’t put everything that is inside cache memory in the main memory as said by the TF.

7.The length of the physical address is equal to the number of bytes in the main memory space.

8. When the tag is found nowhere the output is 0.

EXPLANATION OF THE CODE-

We are implementing the cache such that there are three ways of mapping for it.

One way is better than the other.

First I am asking the user for the type of mapping the user wants.

1.**FULLY ASSOCIATIVE MAPPING-**

A *fully associative* cache contains a single set with *B* ways, where *B* is the number of blocks. A memory address can map to a block in any of these ways. A fully [associative cache](https://www.sciencedirect.com/topics/computer-science/associative-cache) is another name for a *B*-way [set associative cache](https://www.sciencedirect.com/topics/computer-science/set-associative-cache) with one set. First I am inputting the number of bytes (in n). Such that 2^n is the main memory space.

I am further asking for the block size(in power of 2). Now the number of cache lines is equal to the number of blocks .The number of blocks have been calculated using a mathematical formula on line number 19 of the code. Then I ask for the number of operations. Then write “w” and the physical address and the data to be stored or read “ r” and the physical address. If it is written, first I divide the physical address in two parts- the tag part and the offset part. For this I need the block size. Suppose the block size is 4 and 2^2 is 4 , so the number of bits given to the offset is 2.

Rest is the tag part. I make a tag array which is one dimensional and a block array which is 2d with number of blocks as row and number of offset as columns, like the offset is 2, so possible columns are 0,1,2,3.

The tag is checked for in the tag array . If a tag is present it is “CACHE HIT” and the corresponding value is written in the blockarray. If the tag array is not present it is “CACHE MISS” and the tag is stored in any space left in the tag array. If there is no space left in the tag array, then stores inside the main memory and replaces it with the first indexed tag in the tag array.

Then if the user writes “r”, the same division of physical address takes place and, now the tag is seen inside the tag array and if the tag is present , the corresponding value of the tag in the block array is displayed and “CACHE HIT” is printed. If the tag is not present it is searched inside the main memory and replaced in the tag array and then read “CACHE MISS”.If the tag is present nowhere “0” is printed on the screen. Atlast I print the whole cache memory.

2. **DIRECT MAPPING-**

In Direct mapping, assigned each memory block to a specific line in the cache. If a line is previously taken up by a memory block when a new block needs to be loaded, the old block is trashed. An address space is split into two parts: index field and a tag field. The cache is used to store the tag field whereas the rest is stored in the main memory.

First I am inputting the number of bytes (in n). Such that 2^n is the main memory space.

I am further asking for the block size(in power of 2). Now the number of cache lines is equal to the number of blocks . The user is asked to input the number of cache lines . Then I ask for the number of operations. Then write “w” and the physical address and the data to be stored or read “ r” and the physical address. If it is written, first I divide the physical address in two parts- the tag part and the offset part. For this I need the block size. Suppose the block size is 4 and 2^2 is 4 , so the number of bits given to the offset is 2.

Rest is the tag part. I make a tag array which is one dimensional and a block array which is 2d with number of blocks as row and number of offset as columns, like the offset is 2, so possible columns are 0,1,2,3.

The tag part is further broken in two parts - the tag1 part and the index part. The number of bits given to the index is determined using the number of cache lines .Suppose the number of cache lines is 8 which is 2^3. So the number of bits given to the index is 3.Rest is tag1.

The tag1 is checked for in the tag array at the specified index . If a tag1 is present it is “CACHE HIT” and the corresponding value is written in the blockarray. If the tag array is not present it is “CACHE MISS” and the tag is stored in any space left in the tag array. If there is no space left in the tag array, then stores inside the main memory and replaces it with the first indexed tag in the tag array.

Then if the user writes “r”, the same division of physical address takes place and, now the tag1 is seen inside the tag array and if the tag1 is present , the corresponding value of the tag in the block array is displayed and “CACHE HIT” is printed. If the tag is not present it is searched inside the main memory and replaced in the tag array and then read “CACHE MISS”.If the tag is present nowhere “0” is printed on the screen. Atlast I print the whole cache memory.

3. **Set Associative Mapping or k way mapping-**

This form of mapping is an enhanced form of direct mapping where the drawbacks of direct mapping are removed. Set associative addresses the problem of possible thrashing in the direct mapping method. It does this by saying that instead of having exactly one line that a block can map to in the cache, we will group a few lines together creating a ***set***. Then a block in memory can map to any one of the lines of a specific set.

First I am inputting the number of bytes (in n). Such that 2^n is the main memory space. I am further asking for the block size(in power of 2). Now the number of cache lines is equal to the number of blocks . The user is asked to input the number of cache lines and the kth way of mapping. Then I ask for the number of operations. Then write “w” and the physical address and the data to be stored or read “ r” and the physical address. If it is written, first I divide the physical address in two parts- the tag part and the offset part. For this I need the block size. Suppose the block size is 4 and 2^2 is 4 , so the number of bits given to the offset is 2.

Rest is the tag part. I make a tag array which is one dimensional and a block array which is 2d with number of blocks as row and number of offset as columns, like the offset is 2, so possible columns are 0,1,2,3.

The tag part is further broken in two parts - the tag1 part and the set index part. The number of bits given to the index is determined using the number of kth ways .Suppose the kth way is 4 which is 2^2. So the number of bits given to the set index is 2.Rest is tag1.

The tag1 is checked for in the tag array at the specified set index . If a tag1 is present it is “CACHE HIT” and the corresponding value is written in the blockarray. If the tag array is not present it is “CACHE MISS” and the tag is stored in any space left in the tag array. If there is no space left in the tag array, then stores inside the main memory and replaces it with the first indexed tag in the tag array. It is basically going inside the set index which contains k number of blocks or cache lines and all the k number of blocks are searched for.

Then if the user writes “r”, the same division of physical address takes place and, now the tag1 is seen inside the tag array and if the tag1 is present , the corresponding value of the tag in the block array is displayed and “CACHE HIT” is printed. If the tag is not present it is searched inside the main memory and replaced in the tag array and then read “CACHE MISS”.If the tag is present nowhere “0” is printed on the screen. Atlast I print the whole cache memory.

It is the most efficient way of mapping.

I**NPUT FORMAT-**

1. Asking for the type of mapping- “A” for fully associative , “D” for direct mapping and “S” for set associative mapping.
2. In all, asking for the number of bytes for the main memory(in power of 2), the number of cache lines, the block size.
3. In set associative , also asking the k way.

**BONUS ASSIGNMENT-**

There are two levels - level1 and level2.

The first level is half the number of cache lines in level 2.

First writing in the first level as well as second level takes place.

Inclusion property is followed in direct and set associative.Exclusive is followed in fully associative just to show the difference.

Writing in the cache takes place in two levels. First if it fits in the size of level 1 then it is fitted in level 1 and then if it fits in level2, it is fitted in level2. Same is for reading. Then if on reading time the one where level 2 is to be read from the block elements are travelled to level1.

All these things are followed in all the mappings which are enhancements of the level 1 made in initial assignment.

**OUTPUT FORMAT-**

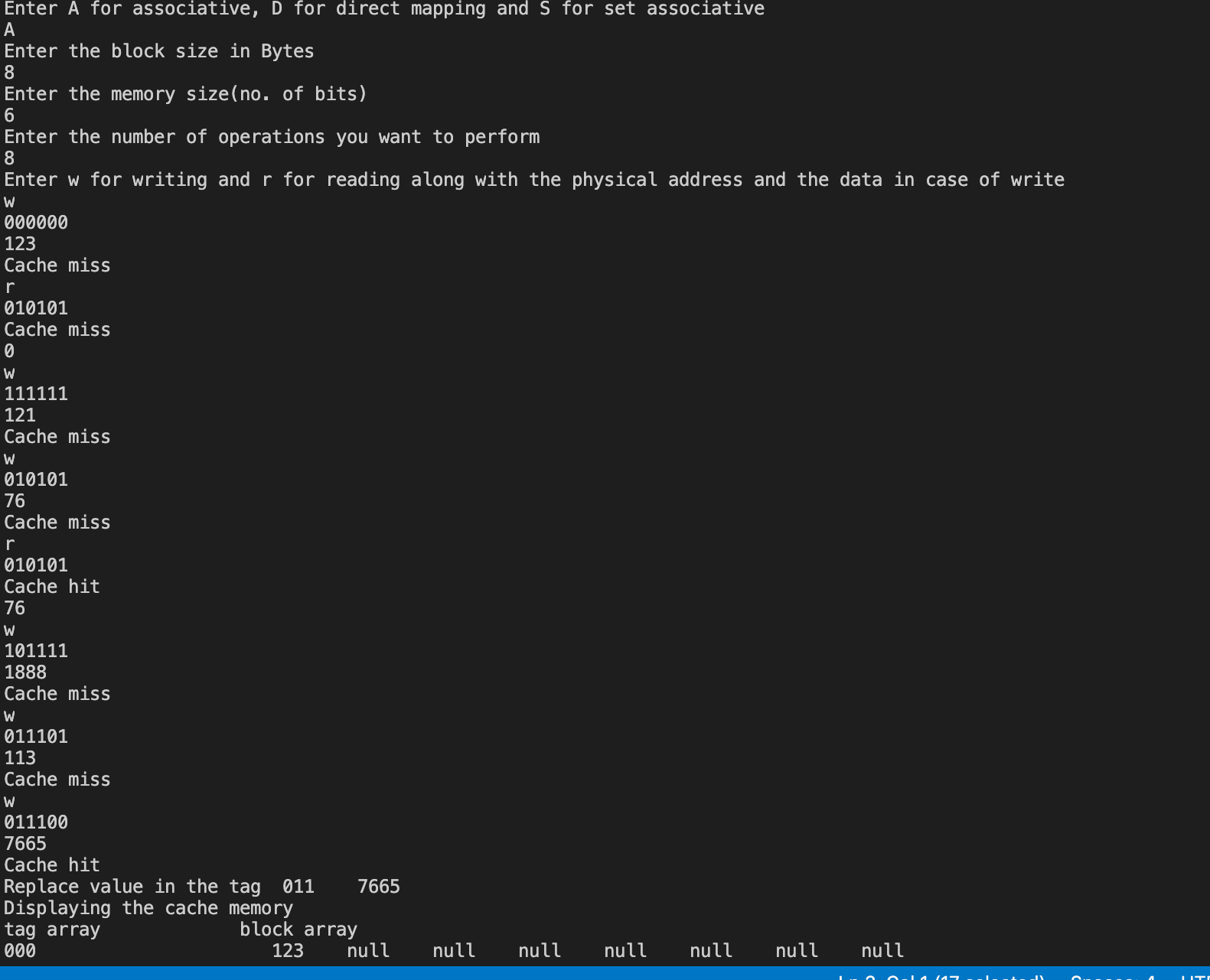
Printing “CACHE MISS” AND “ CACHE HIT” at respective occurrences.

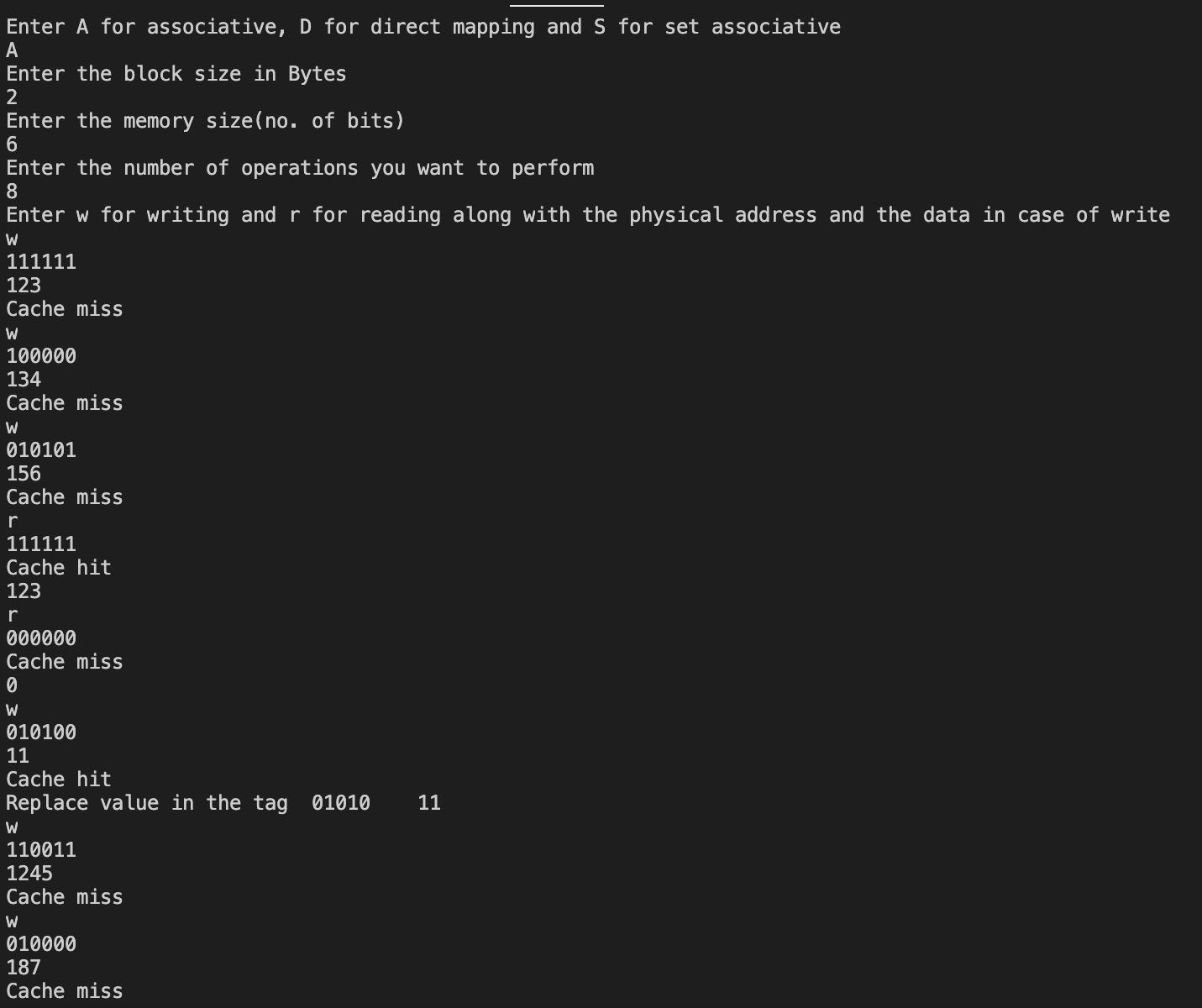
Also when reading is input , displaying the respective data. At the end displaying the full cache memory.

OUTPUT AND INPUT PICTURES EXAMPLES-

1.**FULLY ASSOCIATIVE MAPPING-**

A)

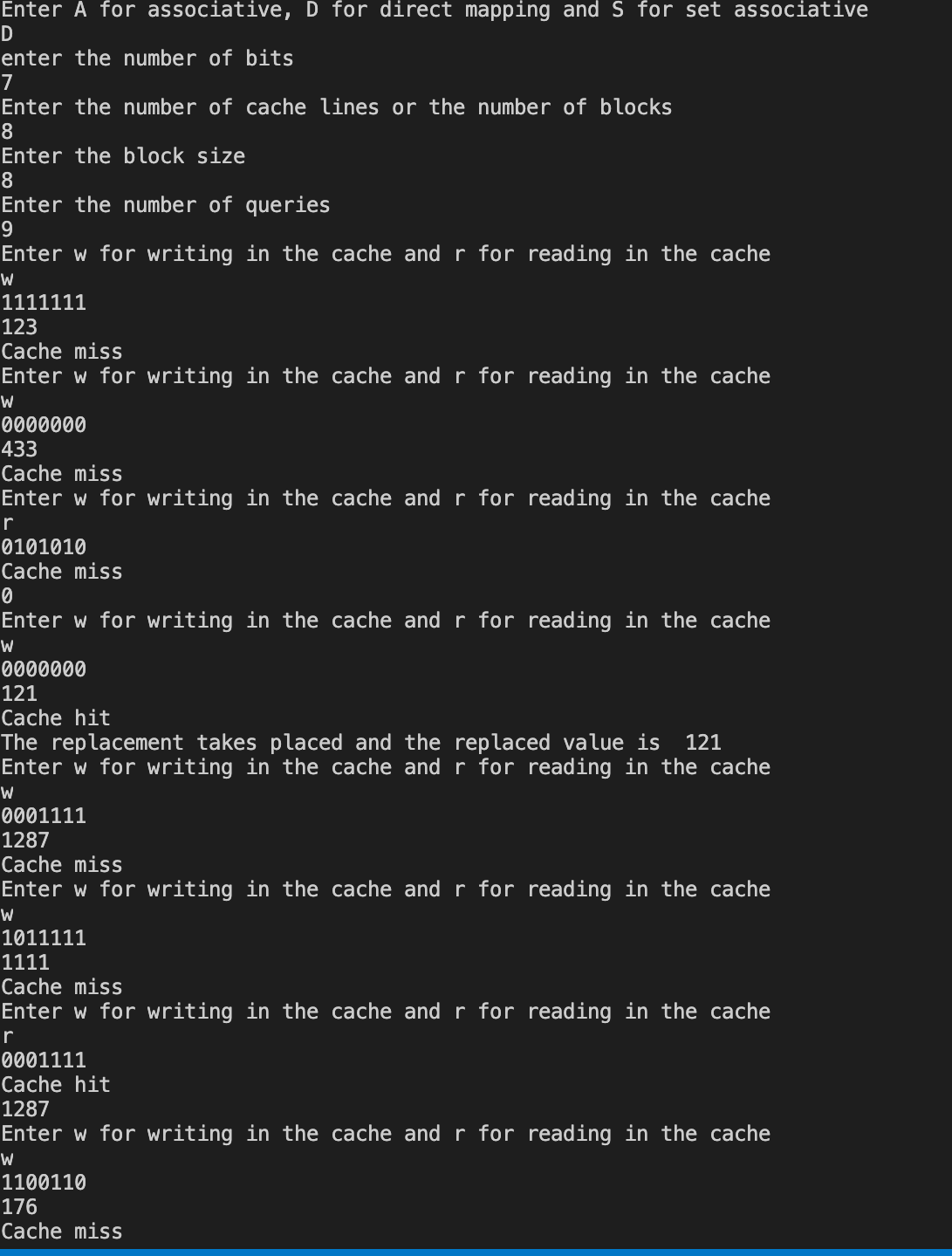


B)

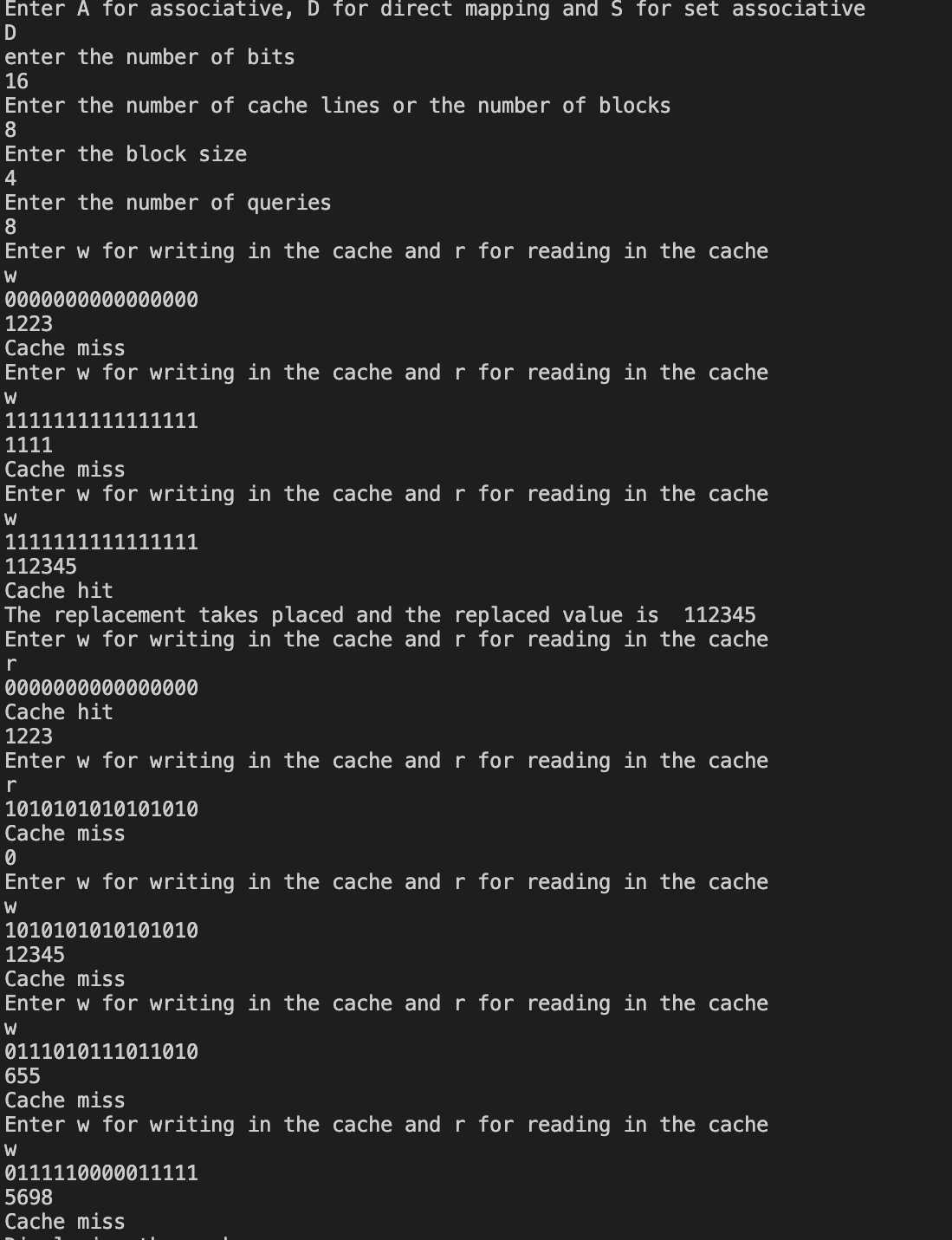
“Abc “ here means there is no tag present.”null” means no data present.

2. **DIRECT MAPPING-**

A)

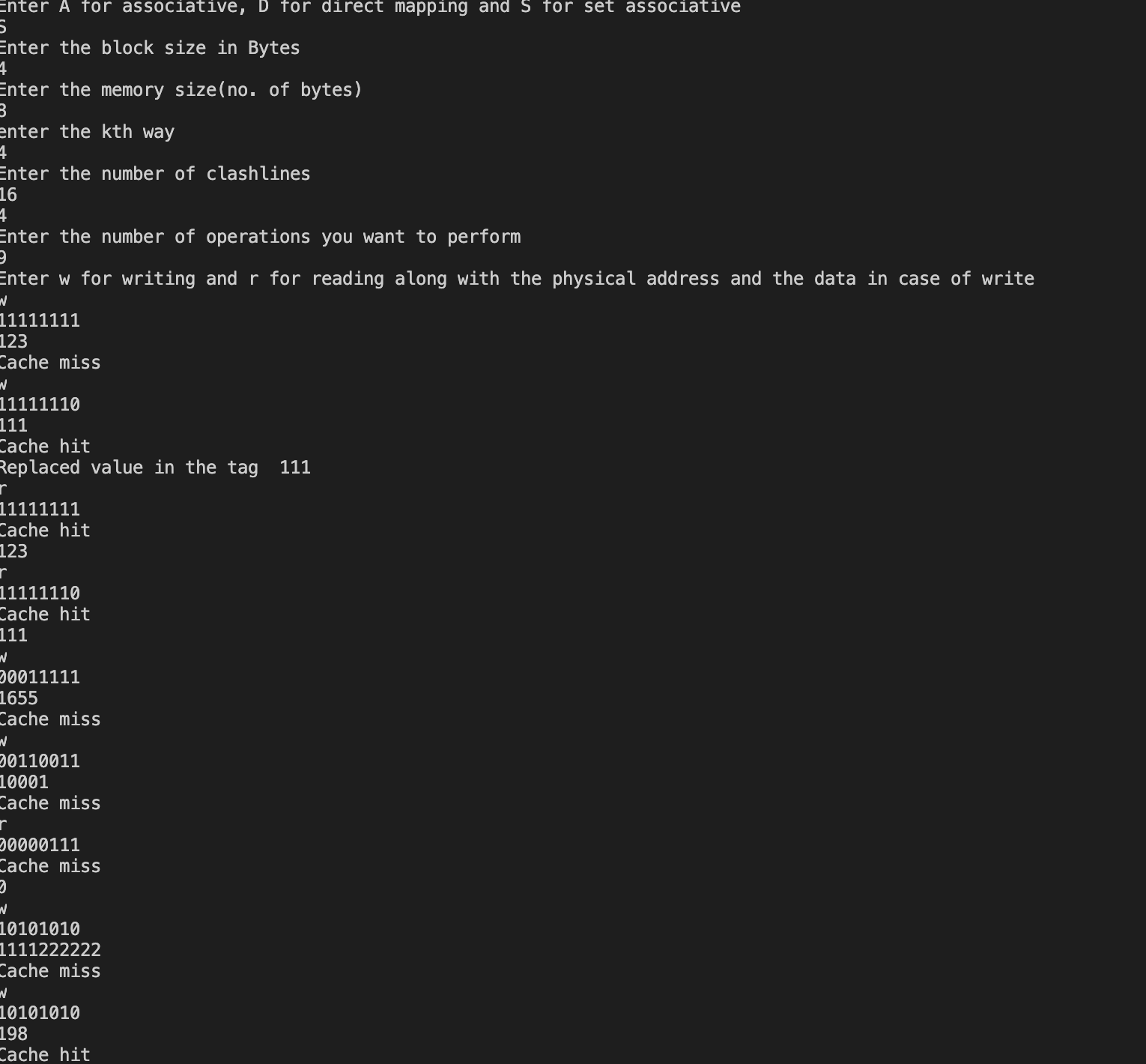


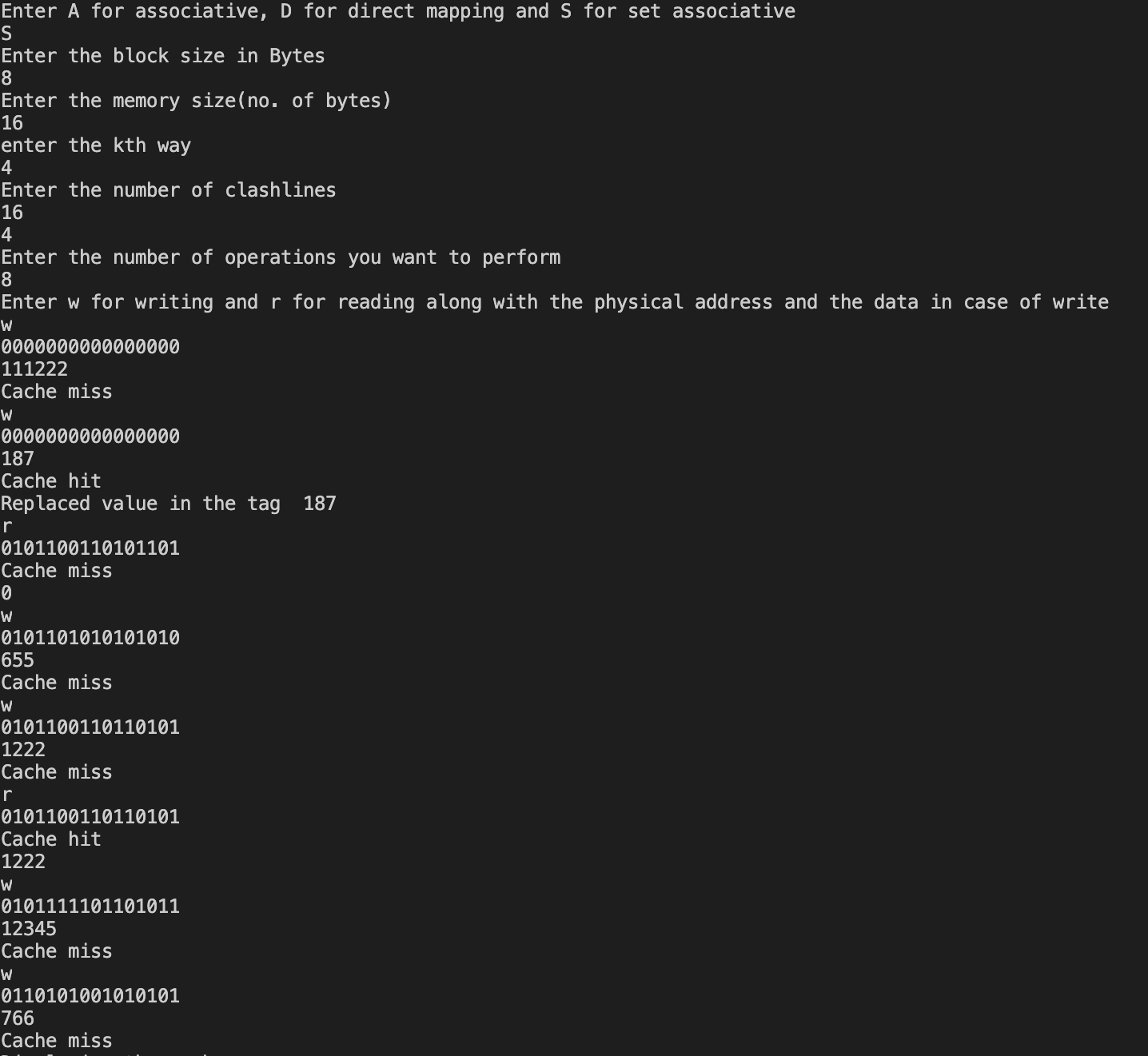
B)



3. **SET ASSOCIATIVE MAPPING-**

A)



B)

ALL THE SAMPLE OUTPUTS HAVE BEEN SHOWN.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

THANKYOU