CO END SEMESTER DOCUMENTATION

Name - SUYASHI SINGHAL

Section - A **Group - 2**

Roll number - 2019478

I am making a cache of size **S** which has a **CL** number of cache lines and **Block** is the size of the blocks. I am considering a 16 bit machine for the same. The addresses can be of 32 bits or 64 bits also if needed. The addresses will be in binary. CL and Block are in powers of 2.
Here **S = CL** * **Block .** Hence we only need to input **CL**(cache lines) and **Block**(block size).

INPUT TO SELECT THE TYPE OF CACHE

After entering CL and Block , the user has to (enter)choose the type of cache he wants to implement.
There are three options-

1. FA - Fully Associative Cache
2. DM - Direct Mapped Cache
3. SA - Set associative Cache

There are three functions in the class "cache" - each implement one type of cache.

BASIC IMPLEMENTATION OF EACH CACHE

In each type of cache the basic idea is to maintain two String arrays. One is the tag array which stores the tag part(block address) of the address in binary string format. The other is the 2D data array which stores the data inside the respective blocks. The number of columns in the data array is equal to the block size and number of rows is equal to the number of cache lines. Using the tag address input , we can determine the offset, block address and the index(in case of direct mapped and set associative cache). The block offset tells us the column in which the data may be found or entered.

PROGRAMMING LANGUAGE USED - JAVA

DATA STRUCTURES USED

1. 1D and 2D string arrays
2. Inbuilt linked list to make a fully associative LRU(least recently used) cache.

Assumptions

We have assumed the following things:

1. CL, Block(block size) and K (in set associative cache) are going to be in **powers of 2**.

2. We are not maintaining a main memory. Any block which is evicted from the cache gets lost permanently (though it's contents are displayed before the said eviction).
3. During a particular run of a program the user can use only one type of cache out of the given three. The user needs to run a new instance of the program to use a new type of cache.
4. The user can make as many queries as he likes consisting of read(R), write(W) and print(P). But as soon as the user chooses to Quit(Q) the program, the program terminates and the data is lost.
5. The tag address can consist of any number of bytes. The only constraint is that N>X i.e the number of bits in input address should be greater than the bits required to determine the offset in case of fully associative.
   And in case of direct mapped N>X+Y i.e the number of bits in input address should be greater than the sum of the number of offset bits(X) and the number of bits to denote the index(Y) in a direct mapped or K-way set associative cache.
6. **The input data is string type. It can be an integer, a character or any string.**

Fully Associative

We make a fully associative cache in case the user enters "**FA**" as the type of cache he wants.

WORKING OF A FULLY ASSOCIATIVE CACHE

Fully associative means that a given block can be present(associated) with any of the cache lines. If CL is the number of cache lines, then we can call it to be a CL-way set associative cache with one set i.e there is a single set with CL blocks in it.
We need to check each entry in the tag array for the presence of the block address to perform a read or a write.
In case of a read, if the block address is present in any of the CL entries in the tag array, and the data is present in the corresponding offset of the block, then we display the data and get a READ HIT. Otherwise, we get a READ MISS.
In case of a write, if the block address is present in any of the CL lines in the tag array, we write the data. If it is not present and we have an empty entry in the tag array, we enter the data and the block address in the first empty entry we encounter. Moreover, if the cache is full, then we evict the least recently used block and replace it with the new block we had to write. In order to evict the least recently used block we make the use of a linked list.

Let the number of bits in the input address be N and B be block size.
If $B = 2^x$ then, X is the number of bits in the offset and N-X is the number of bits in the address which uniquely identify the block. We can compute X by taking log of B with base 2 .

(N-X) bits - **block address**	X bits - **offset**

CODE

The fully associative cache is called by using the function “fully”. Let N be the number of bits in the address.
The parameters taken by the function are:

1. T - Tag array → It is a 1D string array that contains the block addresses. It has rows equal to the number of cache lines.
2. Data - Data array → It is a 2D string array which contains the data.
   No. of columns = Block size
   No of. Rows = no. of cache lines
3. C - It is the number of cache lines (CL).
4. X - It is the number of bits required in the offset
5. B - It is the block size i.e the number of columns in the data array.

Initially the user is prompted to enter the Query. He can either choose to “R” - read, “W” - write, “P” - print, or “Q” - quit.
In case of a write query i.e “W”, the user has to enter the memory address and the data that has to be written in it. The string off is the last X bits of the address. Offset is the decimal of “off” denoting the column number of the block. Tagad is the block address i.e the first N-X bits of the address. If the block address is present in the tag array and no data is present in the corresponding offset column of the block then we write the data in it. If some data is present in that entry we print it and replace it with the new data. If the address is not present in the tag array and one or more of the entries in the tag array is empty, we enter the tag address in the first empty entry we encounter and the corresponding data in the data array. If the cache is full then, the least recently used entry is removed and replaced by the new entry we want to enter. This is done by maintaining a string linked list named “**LRU**”which contains least recently used addresses at the beginning and most recently used ones at the end. Whenever we have to evict a block, we evict the one which is at the head of the linked list.

In case of a read i.e “R”, the user enters the memory address from which the data has to be read. If the block address is present in the tag array and there is data in the corresponding offset of the block, then we get a READ HIT and the data is displayed. Otherwise we get a READ MISS.
In case he chooses the print option - “P” the entire cache gets printed in the form of a table containing the addresses and the corresponding data present in them.

If the user chooses to quit - “Q”, then the program finishes and the user can run the program again if required for a new cache implementation.

Direct Mapping

We make a Direct Mapped cache in case the user enters “**DM**”.

WORKING OF A DIRECT MAPPED CACHE

In the case of a direct mapped cache a block can be stored in only one fixed entry/location in the tag array. We can assign y number of bits from the address as the index in which the block will be stored.
If 2^y = CL , then y bits will be required to denote all the indexes in the cache. For eg. if 128 entries are there in the tag array then 2^7 = 128 . Thus 7 bits will be required to denote all indexes in the array (in this case 0 - 127) .
Hence the last X bits, are used to denote the offset. The last y bits of the address denote the index and lastly the remaining bits are entered as the block address into the tag array. In this way, we divide the tag address into Tag(Block address) , Index, Offset.

(N-X-y)bits - **block address**	y bits - **Index**	X bits - **offset**

In case of a read, if the entered block address is present at its corresponding index and the data is present in the given offset, then we get a “READ HIT”. Otherwise we get a “READ MISS”.
In the case of a write query if the address is present at the corresponding index, then we write the data at it’s offset column. And if not, then we replace the existing address and data (if present) with the new one.
Hence, in a direct mapped cache we only need to check one index for the presence of the block. Thus it is much faster than fully associative. Though, the drawback is that it has the maximum number of cache misses as well.

CODE

The direct mapped cache is called by using the function “direct”. Let N be the number of bits in the address.
The parameters taken by the function are:

1.T - Tag array → It is a 1D string array that contains the block addresses. It has rows equal to the number of cache lines.
2. Data - Data array → It is a 2D string array which contains the data.

No. of columns = Block size
No of. Rows = no. of cache lines
3. C - It is the number of cache lines (CL).
4. X - It is the number of bits required in the offset
5. B - It is the block size i.e the number of columns in the data array.

Initially the program prompts the user to choose one out of the following queries. “W” - write, “R” - read , “P” - print and “Q”-quit.
If the user chooses R i.e to read, then he has to enter the memory address from which the data has to be read. The program determines the “offset” which is the number of bits to be used to denote the column number of the data. It also computes the index, i.e the index of the unique entry in which the block may be found. If the block address in that entry is equal to the block address entered by the user and if data is present in the corresponding offset of the block, then we get a “READ HIT” and the data is displayed. Otherwise , we get a “READ MISS”.
If the user chooses “W”, i.e to write, then he has to enter the memory address and the data that has to be written in it . We again compute the offset and the index. If the tag entry at that index is not equal to the block address entered by the user, then we display the old tag entry and it’s data and replace it with the new block. In this case we get a “WRITE MISS”. In case it is equal to the block address given by the user and there is some data at the corresponding column, then the data is replaced by the new data. On the other hand if the entry at that index is empty, then we add a new entry entered by the user. Here, we get a WRITE HIT.
If the user chooses “P”, i.e to print, then we print the entire cache in the form of a table containing the block addresses and the data inside the block.
After completing each query, the user is once again prompted to enter a new one until he selects to Quit(Q). In that case the program terminates.

SET ASSOCIATIVE CACHE

We make a Set associative Cache in case the user enters “SA” as the type of cache he wants to use.

WORKING OF SET ASSOCIATIVE CACHE

It has the advantages of both the fully associative cache and the direct mapped cache. The cache misses here are less than those in direct mapped and it is faster and less time consuming than the fully associative cache.
If K is the number of cache lines we want in each set then
S(number of sets) = CL(no.of cache lines)/K(blocks in each set)

And if $2^Y = S$ then Y is the number of bits required to index the sets.
The following is how an address of N bits is divided-

(N-Y-Offset)Bits - **BLOCK ADDRESS**	Y bits - **INDEX**	Offset bits - **OFFSET**

In case of a read, if the entered block address is present in its corresponding set and the data is present in the given offset of that block, then we get a “READ HIT”. Otherwise we get a “READ MISS”.
In the case of a write query if the address is present within the corresponding set then we write the data at it’s offset column. And if not, then we replace the existing address and data (if present) with the new one.

CODE

The set associative cache is called by using the function “Set”. Let N be the number of bits in the address.
The parameters taken by the function are:

1.T - Tag array → It is a 1D string array that contains the block addresses. It has rows equal to the number of cache lines.
2. Data - Data array → It is a 2D string array which contains the data.
No. of columns = Block size
No of. Rows = no. of cache lines
3. C - It is the number of cache lines (CL).
4. X - It is the number of bits required in the offset
5. B - It is the block size i.e the number of columns in the data array.
6. K - It is the number of blocks in each set i.e to implement a K- way set associative cache.

After choosing “SA” as the cache type, the user is prompted to enter **K**, in order to make a K-way set associative cache. The program calculates the total number of sets which is equal to no. of cache lines divided by K. It also calculates the number of bits required in order to index these sets.

After that the user is asked to select one of the queries - “R” for read, “W” for write, “P” for print and “Q” for quit.

If the user chooses to read i.e selects “R”, then he has to enter the memory address from which data has to be read. The offset and the index of the set is calculated. It next checks whether the block address entered by the user is present in that set or not. If **i** is the set number then the entries in the set are from **i*K to i*K+K-1**. If the block address and the data is present in the corresponding offset column then we get a “READ HIT” and the data is displayed. Otherwise, we get a “READ MISS”.
If the user chooses to write, i.e “W’, then he has to enter the memory address and the data he wants to write in it. The offset and the index of the set is calculated. It checks whether the block address entered by the user is present in that set or not. If i is the set number then the entries in the set are from i*K to i*K+K-1. If the block address is present in it’s set, then we write the data in the corresponding offset column. If there was some other data, then we display this data and replace it by the new one. In this case we get a “WRITE HIT”. In case the address is not present, but one of the entries in the set is empty, then we fill this entry with the one entered by the user. If the set is completely filled, then we display the evicted block and replace it with the new block. We always evict the first entry in the set(i.e the i*Kth entry) whenever it becomes completely filled. Here, we get a “WRITE MISS”.
If the user chooses “P”, i.e to print, then we print the entire cache in the form of a table containing the block addresses and the data inside the block.
After completing each query, the user is once again prompted to enter a new one until he selects to Quit(Q). In that case the program terminates.

INPUT FORMAT

1. Initially the user needs to input the number of cache lines and block size.
2. Then, he has to put in the type of cache he wants to use.
    - FA - fully associative
    - DM - direct mapped
    - SA - set associative
3. Then the user has to input the query
    - R- read
      Just enter the memory address.
    - W - write
      Enter the memory address and data (in the written order)
    - P - print
      Nothing has to be entered
    - Q - quit
      Nothing has to be entered. The program gets terminated

   After each query is finished the user needs to enter another one till he chooses to quit.
4. In the case of set associative cache K is also required as an input.

OUTPUT

1. We can output the data present in any memory address in case we get a READ HIT. Otherwise, print READ MISS when the data is not present.
2. We get a WRITE HIT whenever the block is already present in the cache. Otherwise we get a WRITE MISS.
3. We can output the data or the block that has been evicted/removed while writing in the cache.
4. We can print the entire cache with all it's block addresses and the data inside them by evoking the "P"- print query .

TRAVERSAL OF ARRAYS IN RESPECTIVE CACHES

1) FULLY ASSOCIATIVE - The entire tag array is traversed to find the block.
2) DIRECT MAPPING - Only one entry is checked depending on the index for the block.
3) SET ASSOCIATIVE - All the entries in the particular set are traversed in order to find the block.

   SPEED

   FA>SA>DM

   CACHE HITS

   FA>SA>DM

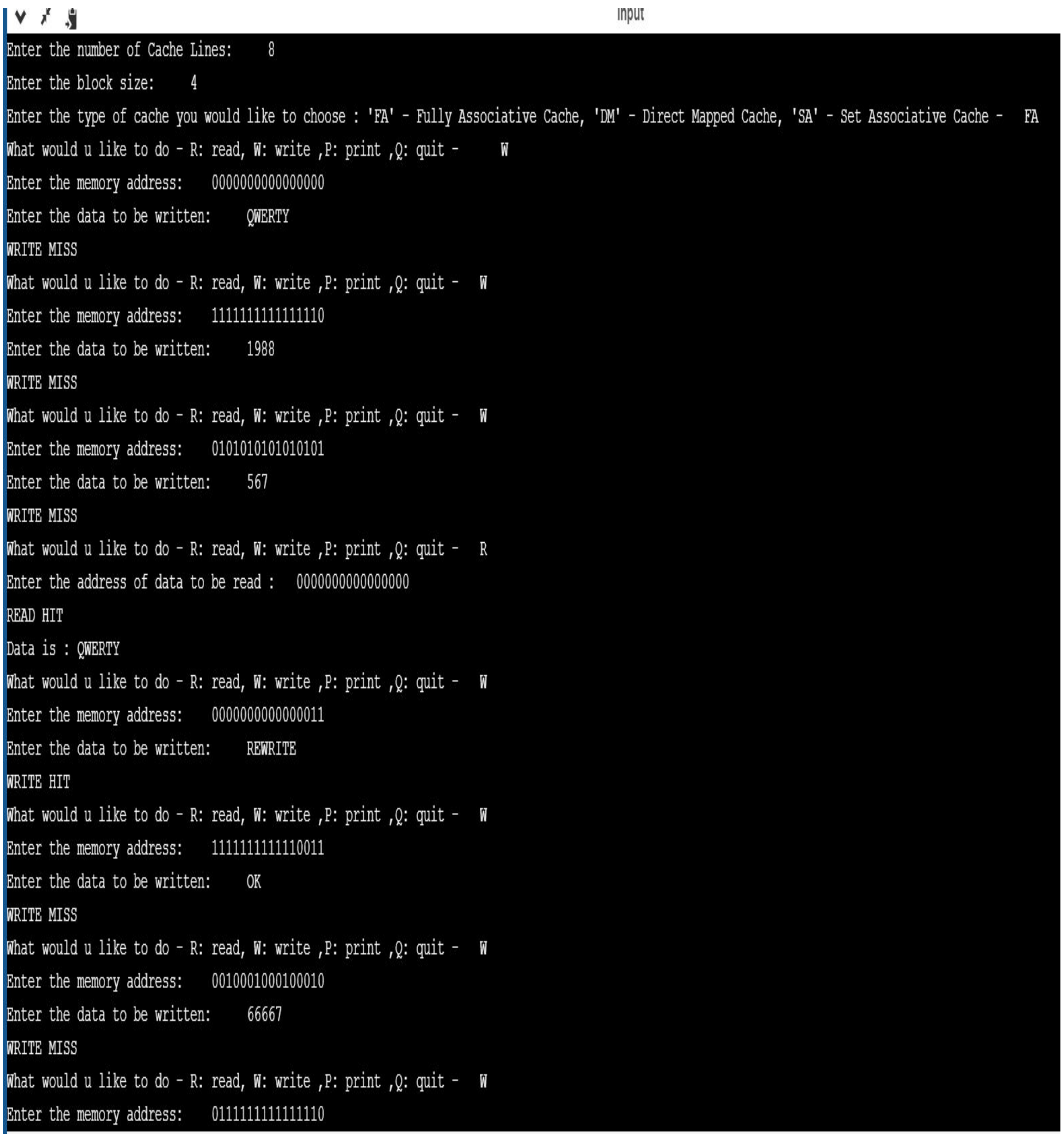
ACTUAL OUTPUT OF THE CODE ON INTERACTIVE CONSOLE

Following are the screenshots of working with each type of cache in my program -

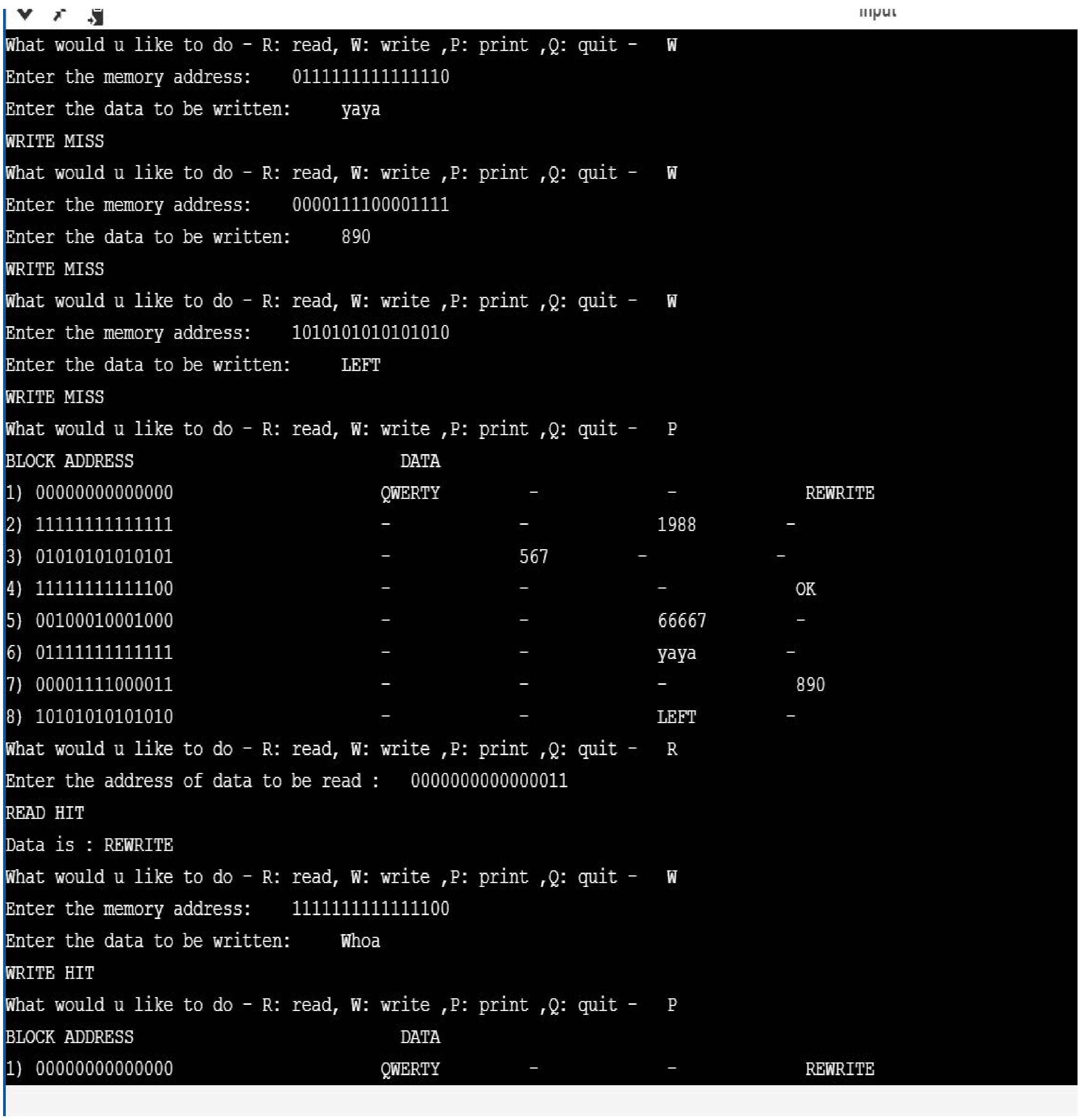
1) Fully associative

The major things to observe here are -

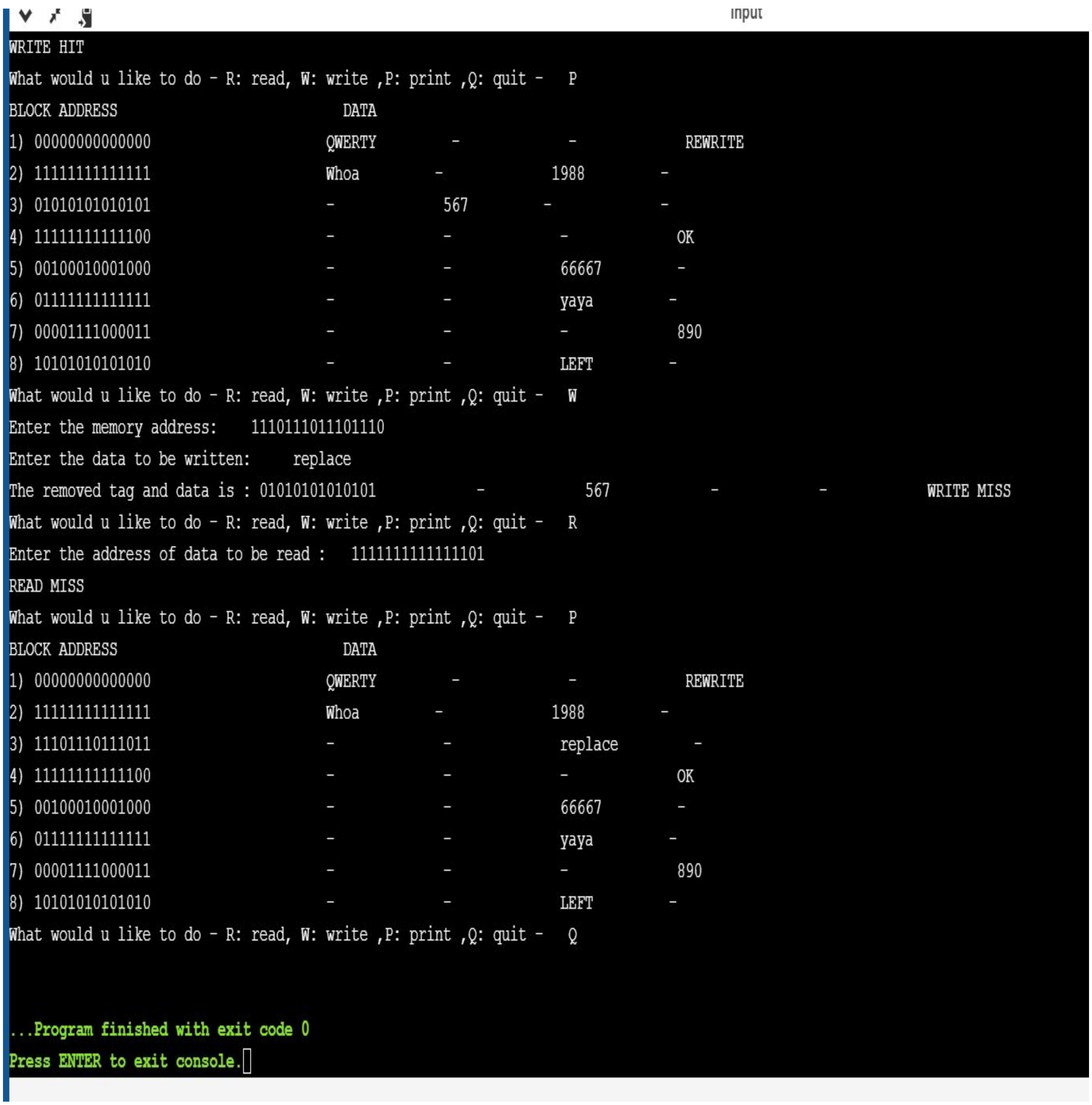
1) We have considered it to be 16 bit machine i.e memory addresses are of 16 bits.
2) The least recently used block is evicted when the cache gets filled up and we need to input a new block. The evicted block is displayed before removal.
3) In case the block is the same, but we input new data at a place where data is already present, the program shows us the removed data.
4) The program prints the entire cache whenever we invoke the "P" print query.


Input

```
Enter the number of Cache Lines:     8
Enter the block size:     4
Enter the type of cache you would like to choose : 'FA' - Fully Associative Cache, 'DM' - Direct Mapped Cache, 'SA' - Set Associative Cache -   FA
What would u like to do - R: read, W: write ,P: print ,Q: quit -      W
Enter the memory address:    0000000000000000
Enter the data to be written:     QWERTY
WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -   W
Enter the memory address:    1111111111111110
Enter the data to be written:     1988
WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -   W
Enter the memory address:    0101010101010101
Enter the data to be written:     567
WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -   R
Enter the address of data to be read :   0000000000000000
READ HIT
Data is : QWERTY
What would u like to do - R: read, W: write ,P: print ,Q: quit -   W
Enter the memory address:    0000000000000011
Enter the data to be written:     REWRITE
WRITE HIT
What would u like to do - R: read, W: write ,P: print ,Q: quit -   W
Enter the memory address:    1111111111110011
Enter the data to be written:     OK
WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -   W
Enter the memory address:    0010001000100010
Enter the data to be written:     66667
WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -   W
Enter the memory address:    0111111111111110
```



```
What would u like to do - R: read, W: write ,P: print ,Q: quit -   W
Enter the memory address:    0111111111111110
Enter the data to be written:     yaya
WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -   W
Enter the memory address:    0000111100001111
Enter the data to be written:     890
WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -   W
Enter the memory address:    1010101010101010
Enter the data to be written:     LEFT
WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -   P
BLOCK ADDRESS                        DATA
1) 00000000000000                  QWERTY        -             -             REWRITE
2) 11111111111111                  -             -             1988          -
3) 01010101010101                  -             567         -             -
4) 11111111111100                  -             -             -             OK
5) 00100010001000                  -             -             66667         -
6) 01111111111111                  -             -             yaya          -
7) 00001111000011                  -             -             -             890
8) 10101010101010                  -             -             LEFT          -
What would u like to do - R: read, W: write ,P: print ,Q: quit -   R
Enter the address of data to be read :   0000000000000011
READ HIT
Data is : REWRITE
What would u like to do - R: read, W: write ,P: print ,Q: quit -   W
Enter the memory address:    1111111111111100
Enter the data to be written:     Whoa
WRITE HIT
What would u like to do - R: read, W: write ,P: print ,Q: quit -   P
BLOCK ADDRESS                        DATA
1) 00000000000000                  QWERTY        -             -             REWRITE
```



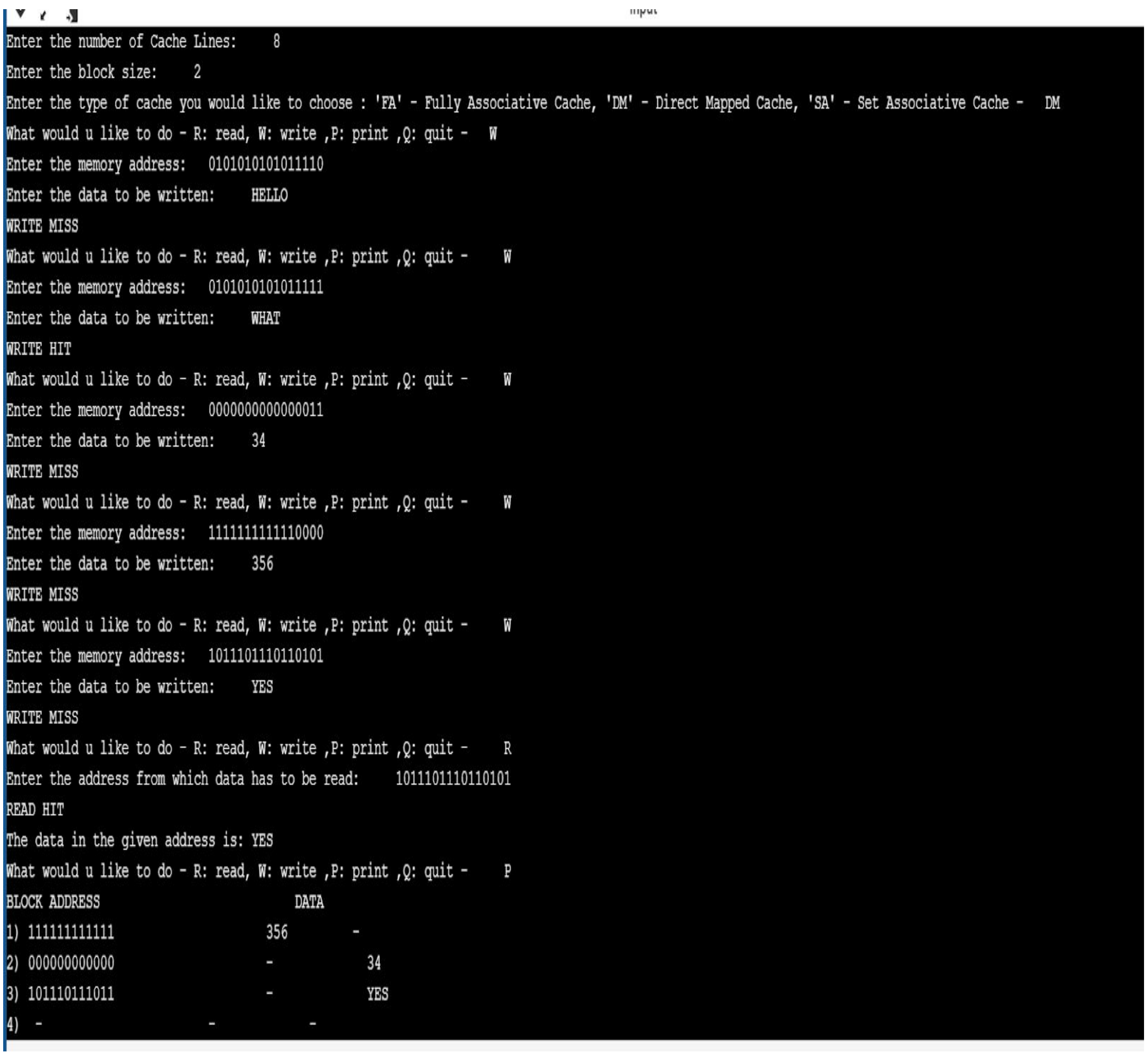
```
WRITE HIT
What would u like to do - R: read, W: write ,P: print ,Q: quit -   P
BLOCK ADDRESS                          DATA
1) 00000000000000                    QWERTY        -             -            REWRITE
2) 11111111111111                    Whoa        -             1988         -
3) 01010101010101                    -             567         -            -
4) 11111111111100                    -             -             -            OK
5) 00100010001000                    -             -             66667        -
6) 01111111111111                    -             -             yaya         -
7) 00001111000011                    -             -             -            890
8) 10101010101010                    -             -             LEFT         -
What would u like to do - R: read, W: write ,P: print ,Q: quit -   W
Enter the memory address:    1110111011101110
Enter the data to be written:     replace
The removed tag and data is : 01010101010101          -            567            -            -            WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -   R
Enter the address of data to be read :   1111111111111101
READ MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -   P
BLOCK ADDRESS                          DATA
1) 00000000000000                    QWERTY        -             -            REWRITE
2) 11111111111111                    Whoa        -             1988         -
3) 11101110111011                    -             -             replace        -
4) 11111111111100                    -             -             -            OK
5) 00100010001000                    -             -             66667        -
6) 01111111111111                    -             -             yaya         -
7) 00001111000011                    -             -             -            890
8) 10101010101010                    -             -             LEFT         -
What would u like to do - R: read, W: write ,P: print ,Q: quit -   Q


...Program finished with exit code 0
Press ENTER to exit console.
```

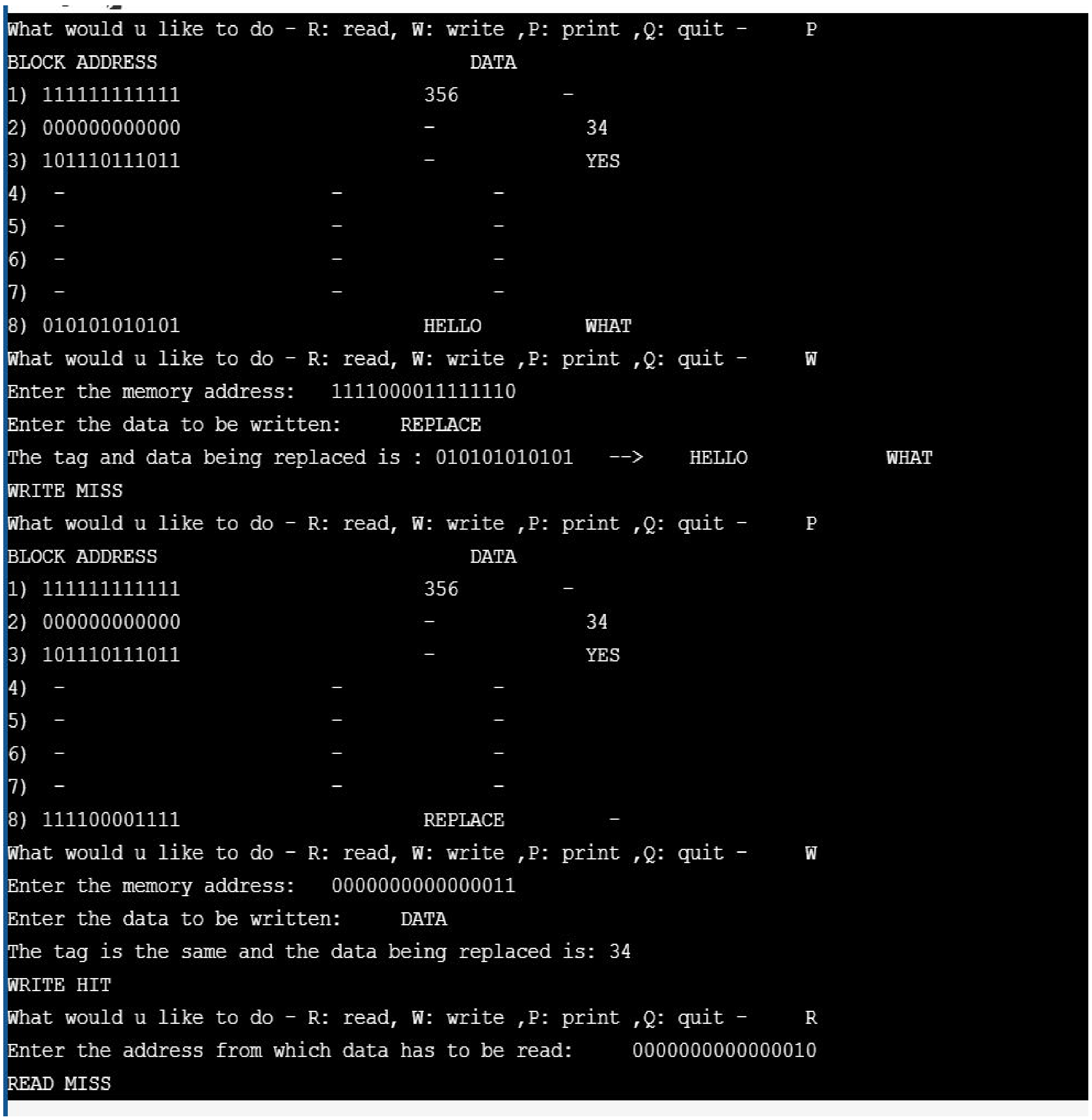
2) Direct Mapped

The major things to observe here are:

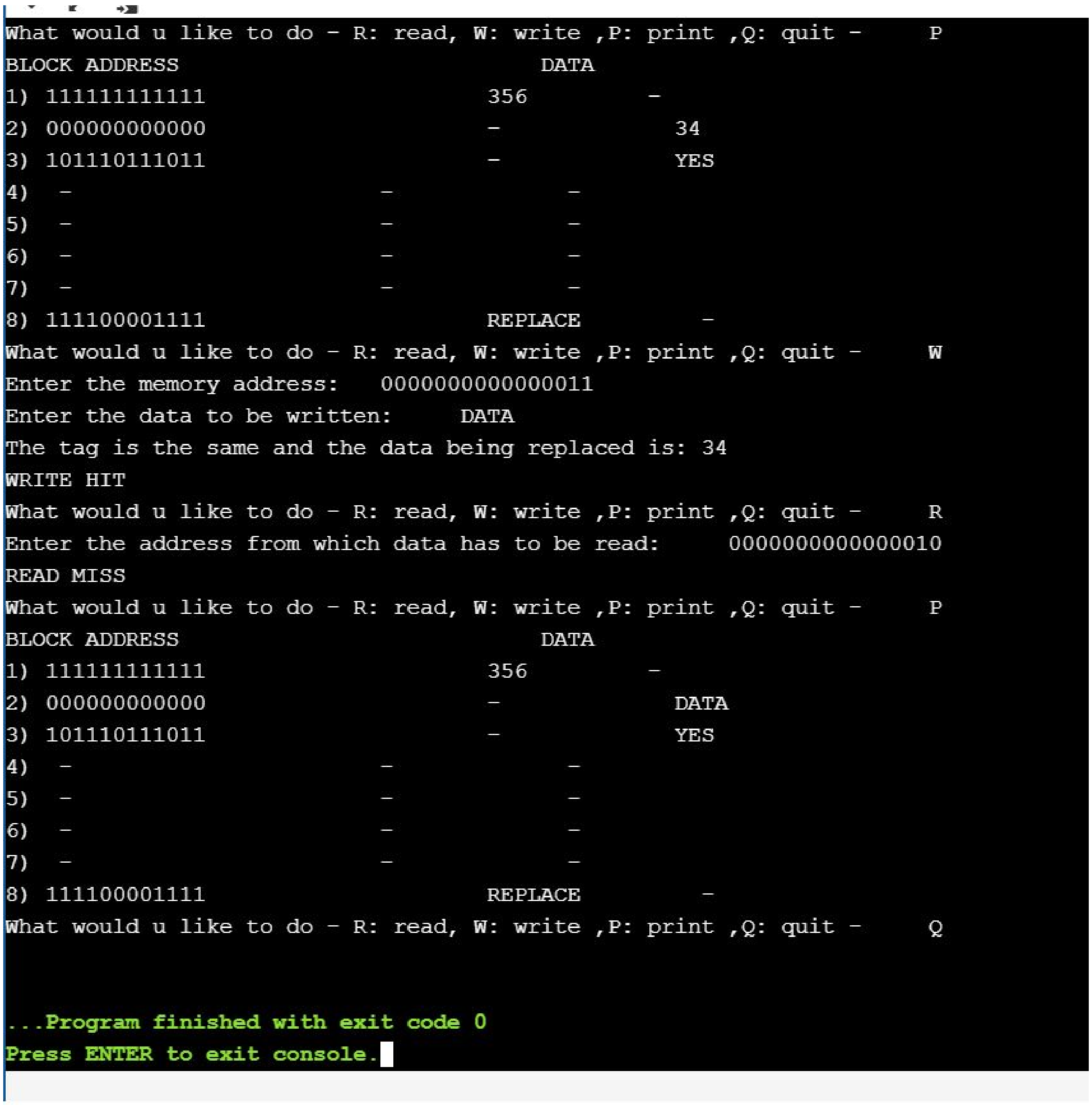
1. Again, the memory addresses are of 16 bits. They can be 32 or 64 bits even, if required.
2. A given block can be put into a unique entry only depending on the index, even if some part of the cache is empty.
3. We display the evicted block/ data before it’s removal.
4. There would be more number of misses as a block is allowed to occupy a unique index only.
5. We can print the entire cache in a tabular manner by invoking “P” query whenever we want.
6. Here, the block address in different entries need not be unique.



```
Enter the number of Cache Lines:     8
Enter the block size:     2
Enter the type of cache you would like to choose : 'FA' - Fully Associative Cache, 'DM' - Direct Mapped Cache, 'SA' - Set Associative Cache -   DM
What would u like to do - R: read, W: write ,P: print ,Q: quit -   W
Enter the memory address:   0101010101011110
Enter the data to be written:     HELLO
WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -     W
Enter the memory address:   0101010101011111
Enter the data to be written:     WHAT
WRITE HIT
What would u like to do - R: read, W: write ,P: print ,Q: quit -     W
Enter the memory address:   0000000000000011
Enter the data to be written:     34
WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -     W
Enter the memory address:   1111111111110000
Enter the data to be written:     356
WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -     W
Enter the memory address:   1011101110110101
Enter the data to be written:     YES
WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -     R
Enter the address from which data has to be read:     1011101110110101
READ HIT
The data in the given address is: YES
What would u like to do - R: read, W: write ,P: print ,Q: quit -     P
BLOCK ADDRESS                         DATA
1) 111111111111                    356        -
2) 000000000000                    -            34
3) 101110111011                    -            YES
4)  -                       -             -
```



```
What would u like to do - R: read, W: write ,P: print ,Q: quit -     P
BLOCK ADDRESS                               DATA
1) 111111111111                     356         -
2) 000000000000                     -             34
3) 101110111011                     -             YES
4)  -                       -             -
5)  -                       -             -
6)  -                       -             -
7)  -                       -             -
8) 010101010101                     HELLO         WHAT
What would u like to do - R: read, W: write ,P: print ,Q: quit -     W
Enter the memory address:   1111000011111110
Enter the data to be written:     REPLACE
The tag and data being replaced is : 010101010101   -->    HELLO            WHAT
WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -     P
BLOCK ADDRESS                               DATA
1) 111111111111                     356         -
2) 000000000000                     -             34
3) 101110111011                     -             YES
4)  -                       -             -
5)  -                       -             -
6)  -                       -             -
7)  -                       -             -
8) 111100001111                     REPLACE         -
What would u like to do - R: read, W: write ,P: print ,Q: quit -     W
Enter the memory address:   0000000000000011
Enter the data to be written:     DATA
The tag is the same and the data being replaced is: 34
WRITE HIT
What would u like to do - R: read, W: write ,P: print ,Q: quit -     R
Enter the address from which data has to be read:     0000000000000010
READ MISS
```



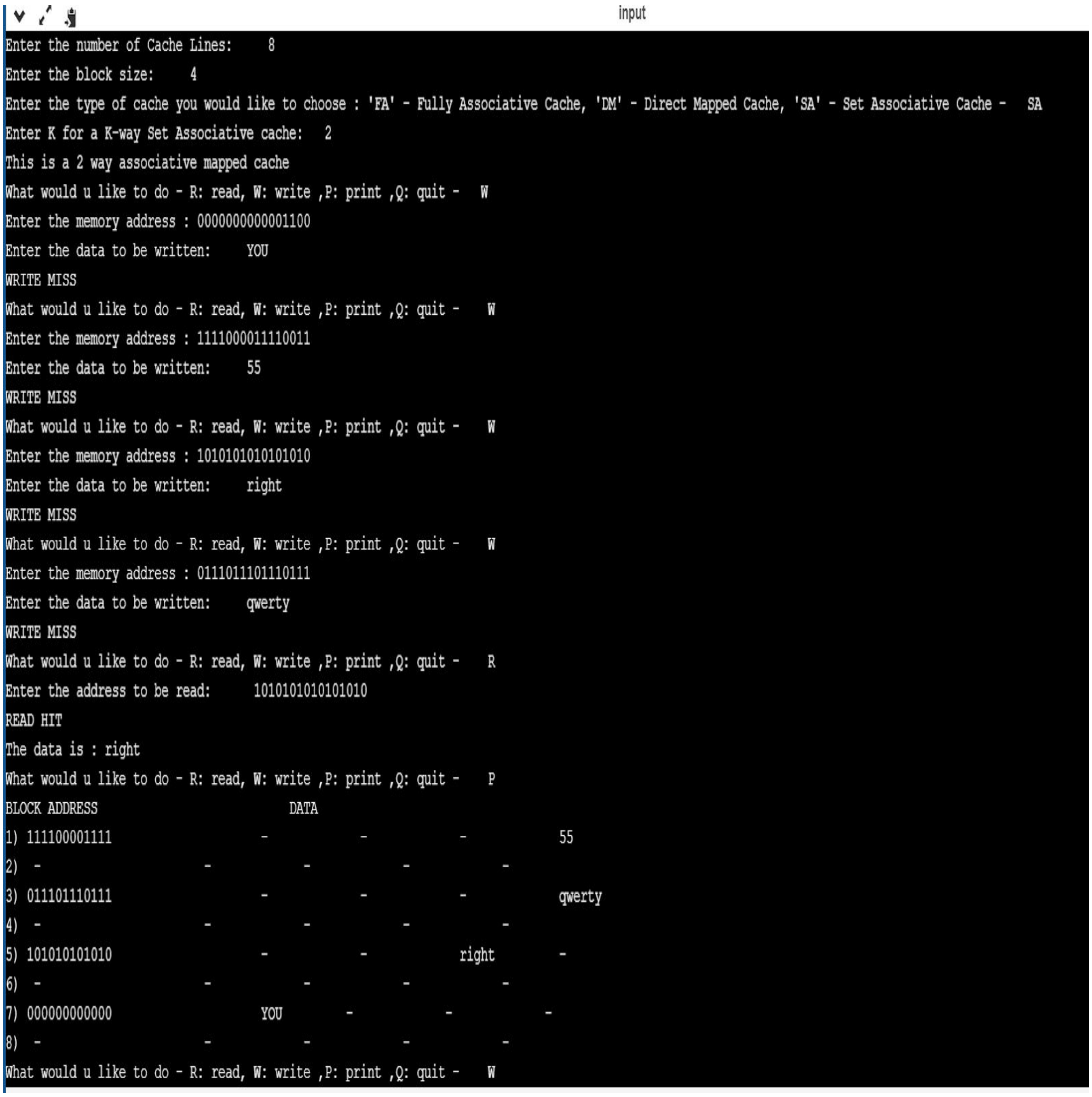
```
What would u like to do - R: read, W: write ,P: print ,Q: quit -     P
BLOCK ADDRESS                               DATA
1) 111111111111                         356          -
2) 000000000000                         -              34
3) 101110111011                         -              YES
4)  -                       -              -
5)  -                       -              -
6)  -                       -              -
7)  -                       -              -
8) 111100001111                         REPLACE          -
What would u like to do - R: read, W: write ,P: print ,Q: quit -     W
Enter the memory address:   0000000000000011
Enter the data to be written:     DATA
The tag is the same and the data being replaced is: 34
WRITE HIT
What would u like to do - R: read, W: write ,P: print ,Q: quit -     R
Enter the address from which data has to be read:     0000000000000010
READ MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -     P
BLOCK ADDRESS                               DATA
1) 111111111111                         356          -
2) 000000000000                         -              DATA
3) 101110111011                         -              YES
4)  -                       -              -
5)  -                       -              -
6)  -                       -              -
7)  -                       -              -
8) 111100001111                         REPLACE          -
What would u like to do - R: read, W: write ,P: print ,Q: quit -     Q


...Program finished with exit code 0
Press ENTER to exit console.
```

3) Set associative

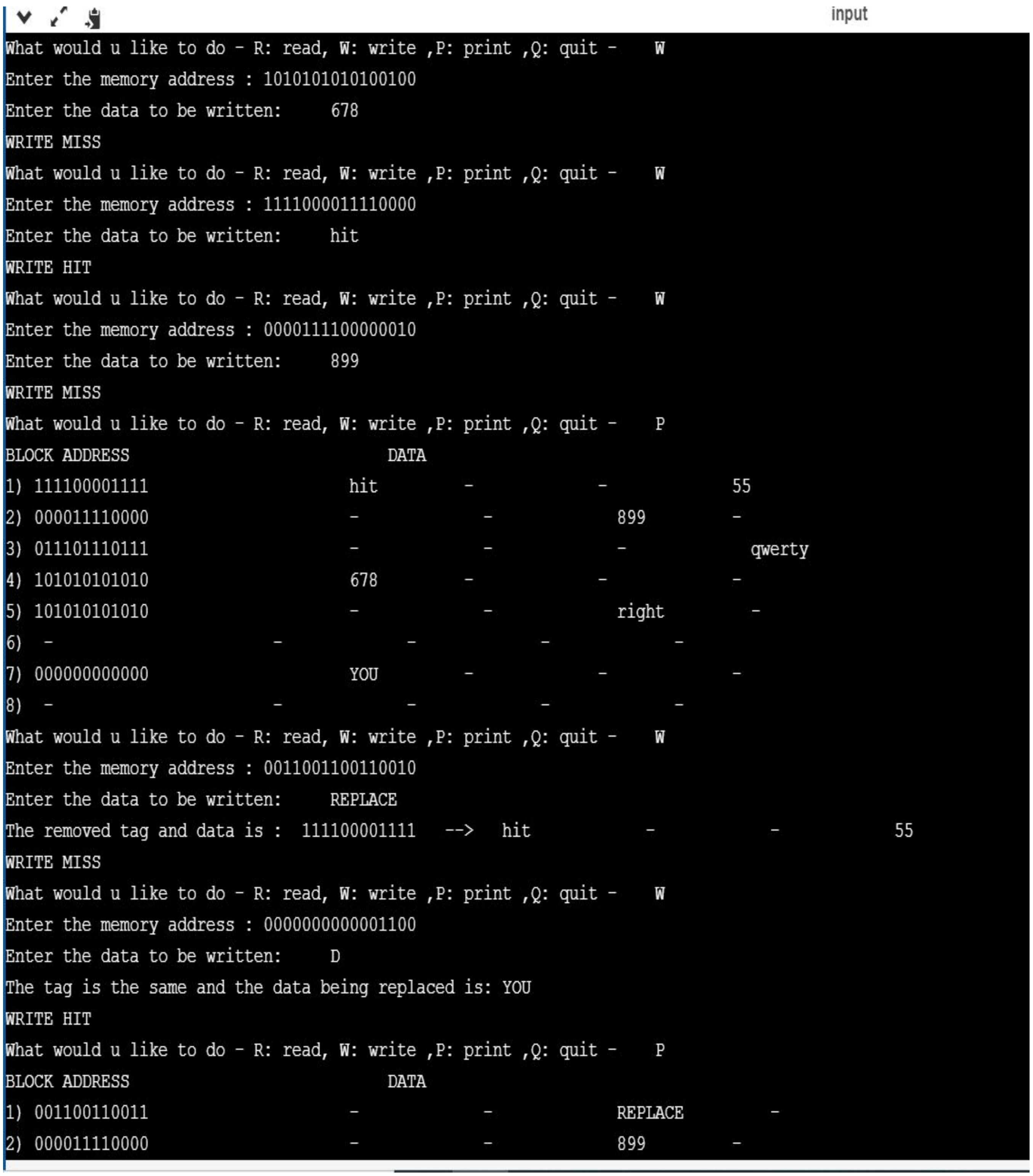
The major things to observe here are:

1. Each set contains K entries.
2. The block address has to be unique in a given set. But the same block address can be present in two or more different sets.
3. When the set gets filled up and we need to enter a new block into it, we remove the first element of the set and replace it with a new block.
4. We display the contents of the block being evicted or the data being replaced before the said removal.
5. We can print the entire cache in a tabular manner by invoking the “P” print query.
6. This type of cache balances the advantages and disadvantages of a fully associative cache and a direct mapped cache.

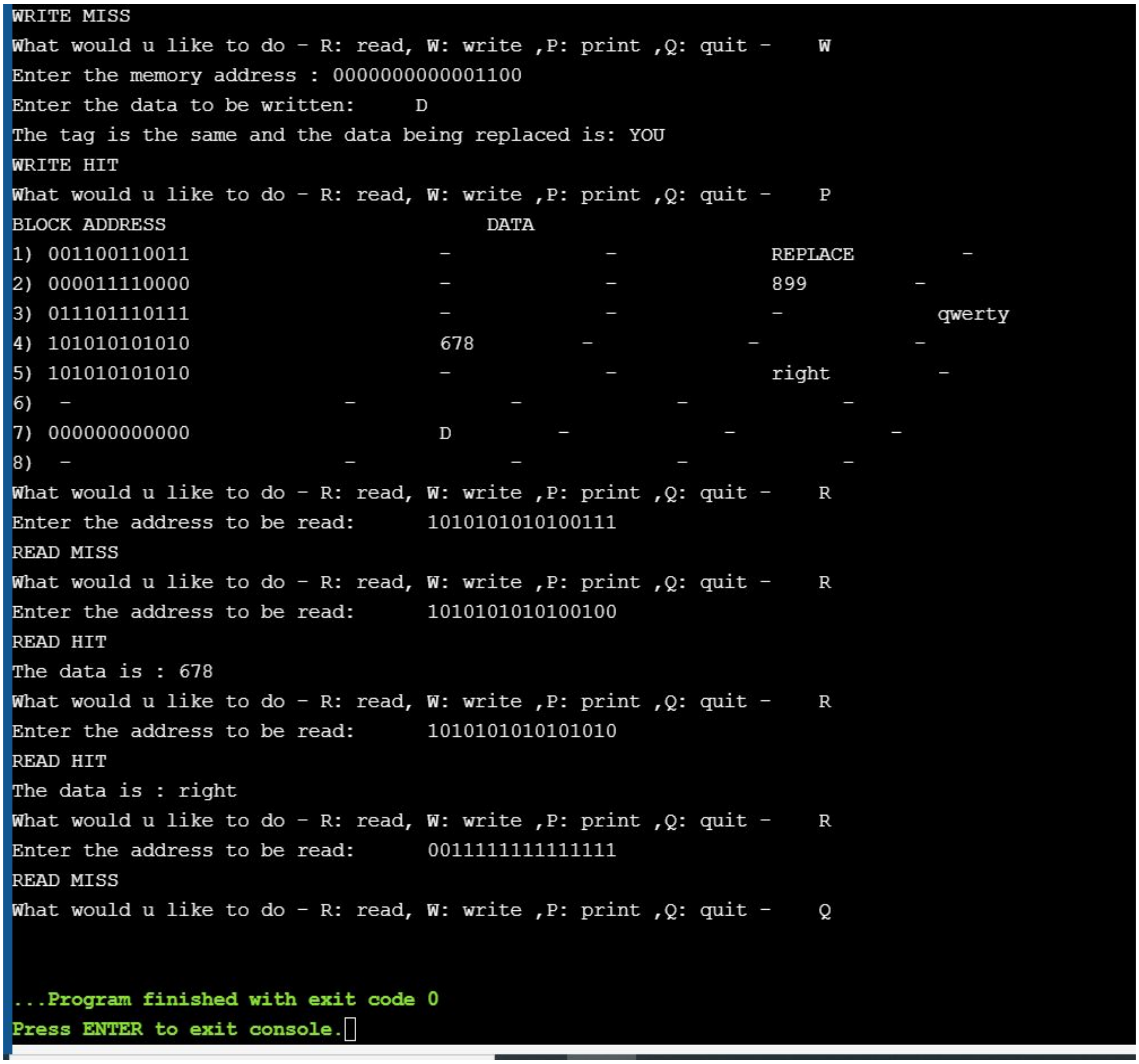


input

```
Enter the number of Cache Lines:     8
Enter the block size:     4
Enter the type of cache you would like to choose : 'FA' - Fully Associative Cache, 'DM' - Direct Mapped Cache, 'SA' - Set Associative Cache -   SA
Enter K for a K-way Set Associative cache:   2
This is a 2 way associative mapped cache
What would u like to do - R: read, W: write ,P: print ,Q: quit -   W
Enter the memory address : 0000000000001100
Enter the data to be written:     YOU
WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -    W
Enter the memory address : 1111000011110011
Enter the data to be written:     55
WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -    W
Enter the memory address : 1010101010101010
Enter the data to be written:     right
WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -    W
Enter the memory address : 0111011101110111
Enter the data to be written:     qwerty
WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -    R
Enter the address to be read:      1010101010101010
READ HIT
The data is : right
What would u like to do - R: read, W: write ,P: print ,Q: quit -    P
BLOCK ADDRESS                        DATA
1) 111100001111              -            -            -            55
2)  -               -            -            -            -
3) 011101110111              -            -            -            qwerty
4)  -               -            -            -            -
5) 101010101010              -            -            right        -
6)  -               -            -            -            -
7) 000000000000              YOU        -            -            -
8)  -               -            -            -            -
What would u like to do - R: read, W: write ,P: print ,Q: quit -    W
```



```
What would u like to do - R: read, W: write ,P: print ,Q: quit -    W
Enter the memory address : 1010101010100100
Enter the data to be written:     678
WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -    W
Enter the memory address : 1111000011110000
Enter the data to be written:     hit
WRITE HIT
What would u like to do - R: read, W: write ,P: print ,Q: quit -    W
Enter the memory address : 0000111100000010
Enter the data to be written:     899
WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -    P
BLOCK ADDRESS                           DATA
1) 111100001111                     hit         -             -             55
2) 000011110000                     -             -             899         -
3) 011101110111                     -             -             -             qwerty
4) 101010101010                     678         -             -             -
5) 101010101010                     -             -             right         -
6)  -                       -             -             -             -
7) 000000000000                     YOU         -             -             -
8)  -                       -             -             -             -
What would u like to do - R: read, W: write ,P: print ,Q: quit -    W
Enter the memory address : 0011001100110010
Enter the data to be written:     REPLACE
The removed tag and data is :  111100001111   -->   hit            -            -            55
WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -    W
Enter the memory address : 0000000000001100
Enter the data to be written:     D
The tag is the same and the data being replaced is: YOU
WRITE HIT
What would u like to do - R: read, W: write ,P: print ,Q: quit -    P
BLOCK ADDRESS                           DATA
1) 001100110011                     -             -             REPLACE         -
2) 000011110000                     -             -             899         -
```



```
WRITE MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -    W
Enter the memory address : 0000000000001100
Enter the data to be written:     D
The tag is the same and the data being replaced is: YOU
WRITE HIT
What would u like to do - R: read, W: write ,P: print ,Q: quit -    P
BLOCK ADDRESS                           DATA
1) 001100110011                      -             -            REPLACE          -
2) 000011110000                      -             -            899          -
3) 011101110111                      -             -             -            qwerty
4) 101010101010                     678          -             -             -
5) 101010101010                      -             -            right          -
6)  -                       -             -             -             -
7) 000000000000                      D         -             -             -
8)  -                       -             -             -             -
What would u like to do - R: read, W: write ,P: print ,Q: quit -    R
Enter the address to be read:      1010101010100111
READ MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -    R
Enter the address to be read:      1010101010100100
READ HIT
The data is : 678
What would u like to do - R: read, W: write ,P: print ,Q: quit -    R
Enter the address to be read:      1010101010101010
READ HIT
The data is : right
What would u like to do - R: read, W: write ,P: print ,Q: quit -    R
Enter the address to be read:      0011111111111111
READ MISS
What would u like to do - R: read, W: write ,P: print ,Q: quit -    Q


...Program finished with exit code 0
Press ENTER to exit console.
```