Computer Organisation Final Project

Cache Assignment

| Ananya Jain | 2019408 |

Basic Project Description:

This project aims at stimulating a Cache in a memory subsystem, which is much smaller than the main memory but acts as a means through which the access and storage of Data in the memory becomes efficient in relation to the time while also maintaining its correctness.

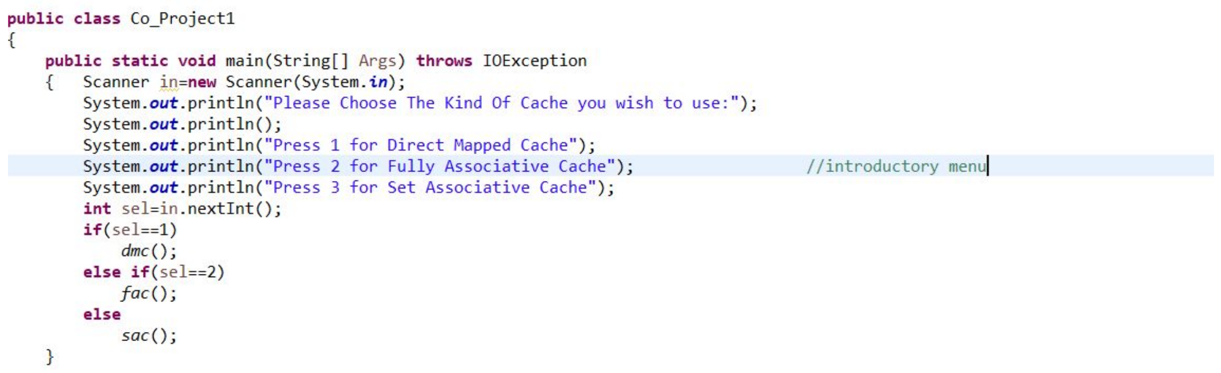
My Project simulates the L1 cache in a memory and is capable of performing the following functions:

- Read
- Write
- Replace
- Evict
- Insertion

The simulation is dynamic and is based on the input provided by the user.Also I have maintained a Main Memory too,which allows the seamless usage of the cache.

Input Description:

The word length in this project code can be at max 16-bit, but lower values are also acceptable.
The data stored can be at max 8 bytes since i have used long to store data values.
The user first Inputs their choice of Cache by choosing any one of the three caches available.Once chosen,the following inputs are asked for:



```java
public class Co_Project1
{
    public static void main(String[] Args) throws IOException
    {   Scanner in=new Scanner(System.in);
        System.out.println("Please Choose The Kind Of Cache you wish to use:");
        System.out.println();
        System.out.println("Press 1 for Direct Mapped Cache");
        System.out.println("Press 2 for Fully Associative Cache");                //introductory menu
        System.out.println("Press 3 for Set Associative Cache");
        int sel=in.nextInt();
        if(sel==1)
            dmc();
        else if(sel==2)
            fac();
        else
            sac();
    }
```

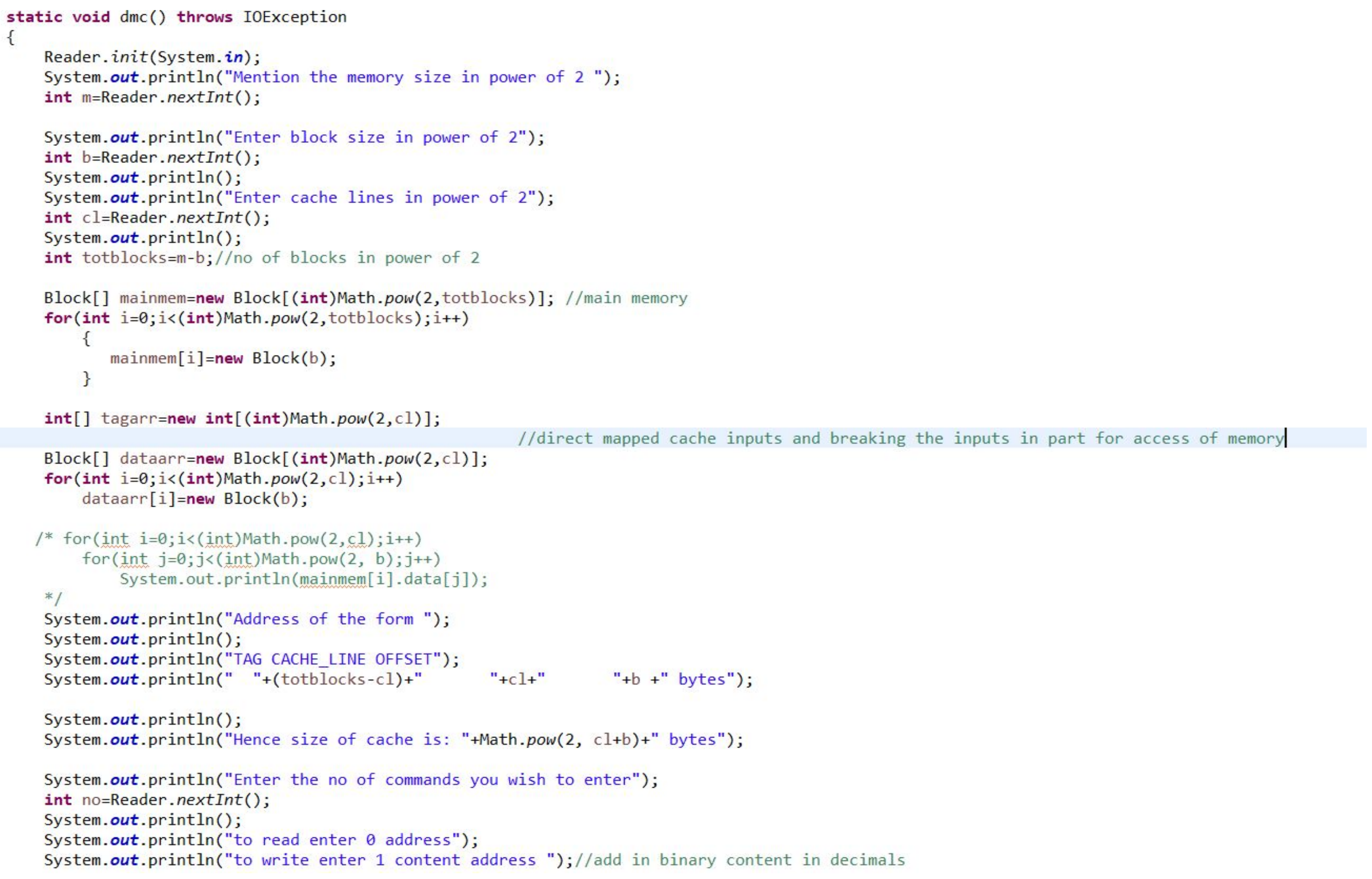
The constraints of the inputs are as follows-

All inputs are to be provided in Powers of 2 only

Direct Mapped Cache and Fully Associative Cache

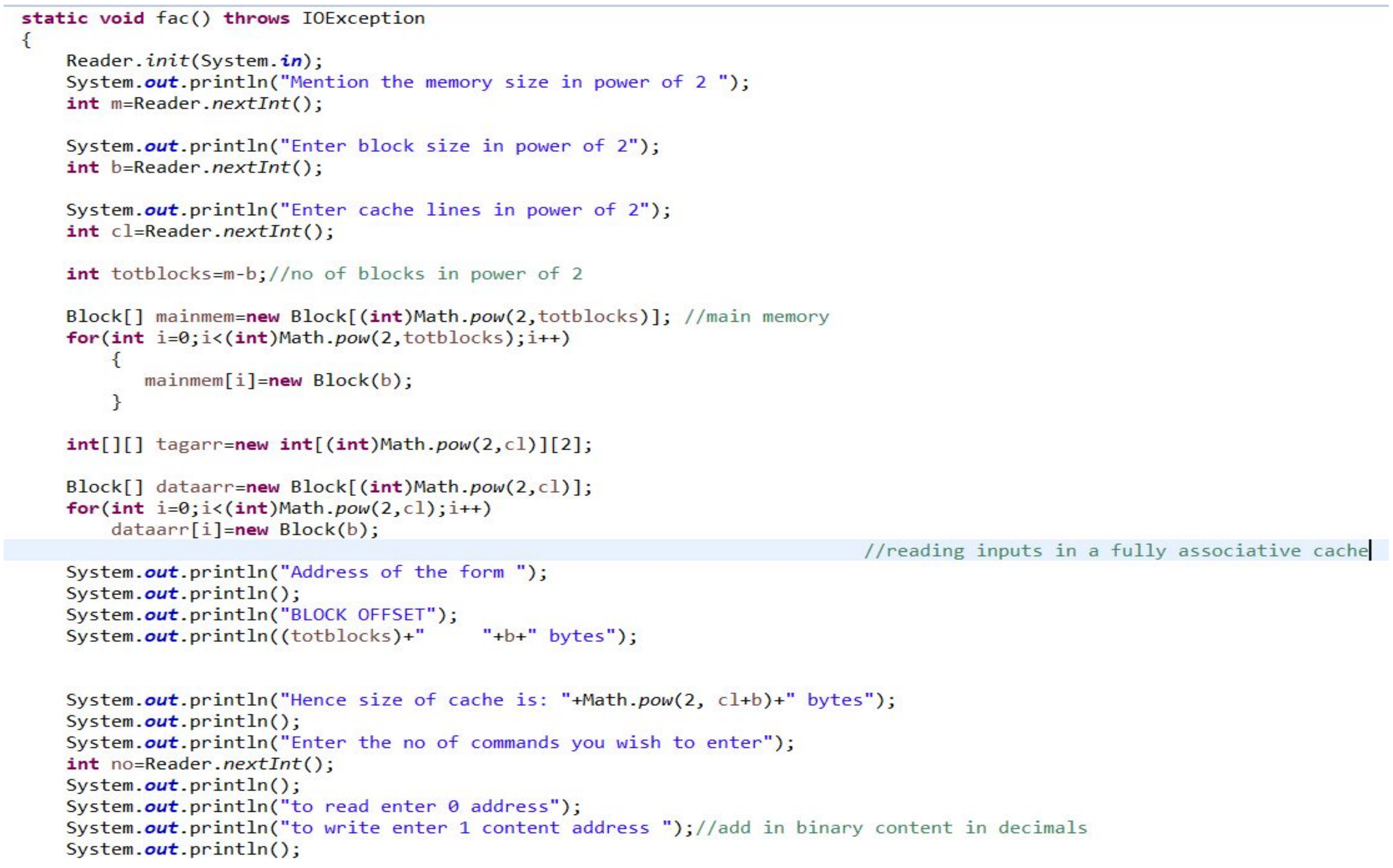
- ❏ Size of Memory: N<=16
- ❏ Bits for a Block: B<16
- ❏ Cache Line:CL<=(N-B)

INPUTS OF DIRECT MAPPED CACHE



```
static void dmc() throws IOException
{
    Reader.init(System.in);
    System.out.println("Mention the memory size in power of 2 ");
    int m=Reader.nextInt();

    System.out.println("Enter block size in power of 2");
    int b=Reader.nextInt();
    System.out.println();
    System.out.println("Enter cache lines in power of 2");
    int cl=Reader.nextInt();
    System.out.println();
    int totblocks=m-b;//no of blocks in power of 2

    Block[] mainmem=new Block[(int)Math.pow(2,totblocks)]; //main memory
    for(int i=0;i<(int)Math.pow(2,totblocks);i++)
        {
           mainmem[i]=new Block(b);
        }

    int[] tagarr=new int[(int)Math.pow(2,cl)];
                                                              //direct mapped cache inputs and breaking the inputs in part for access of memory
    Block[] dataarr=new Block[(int)Math.pow(2,cl)];
    for(int i=0;i<(int)Math.pow(2,cl);i++)
        dataarr[i]=new Block(b);

   /* for(int i=0;i<(int)Math.pow(2,cl);i++)
        for(int j=0;j<(int)Math.pow(2, b);j++)
            System.out.println(mainmem[i].data[j]);
    */
    System.out.println("Address of the form ");
    System.out.println();
    System.out.println("TAG CACHE_LINE OFFSET");
    System.out.println("  "+(totblocks-cl)+"       "+cl+"       "+b +" bytes");

    System.out.println();
    System.out.println("Hence size of cache is: "+Math.pow(2, cl+b)+" bytes");

    System.out.println("Enter the no of commands you wish to enter");
    int no=Reader.nextInt();
    System.out.println();
    System.out.println("to read enter 0 address");
    System.out.println("to write enter 1 content address ");//add in binary content in decimals
```

INPUTS OF A FULLY ASSOCIATIVE CACHE

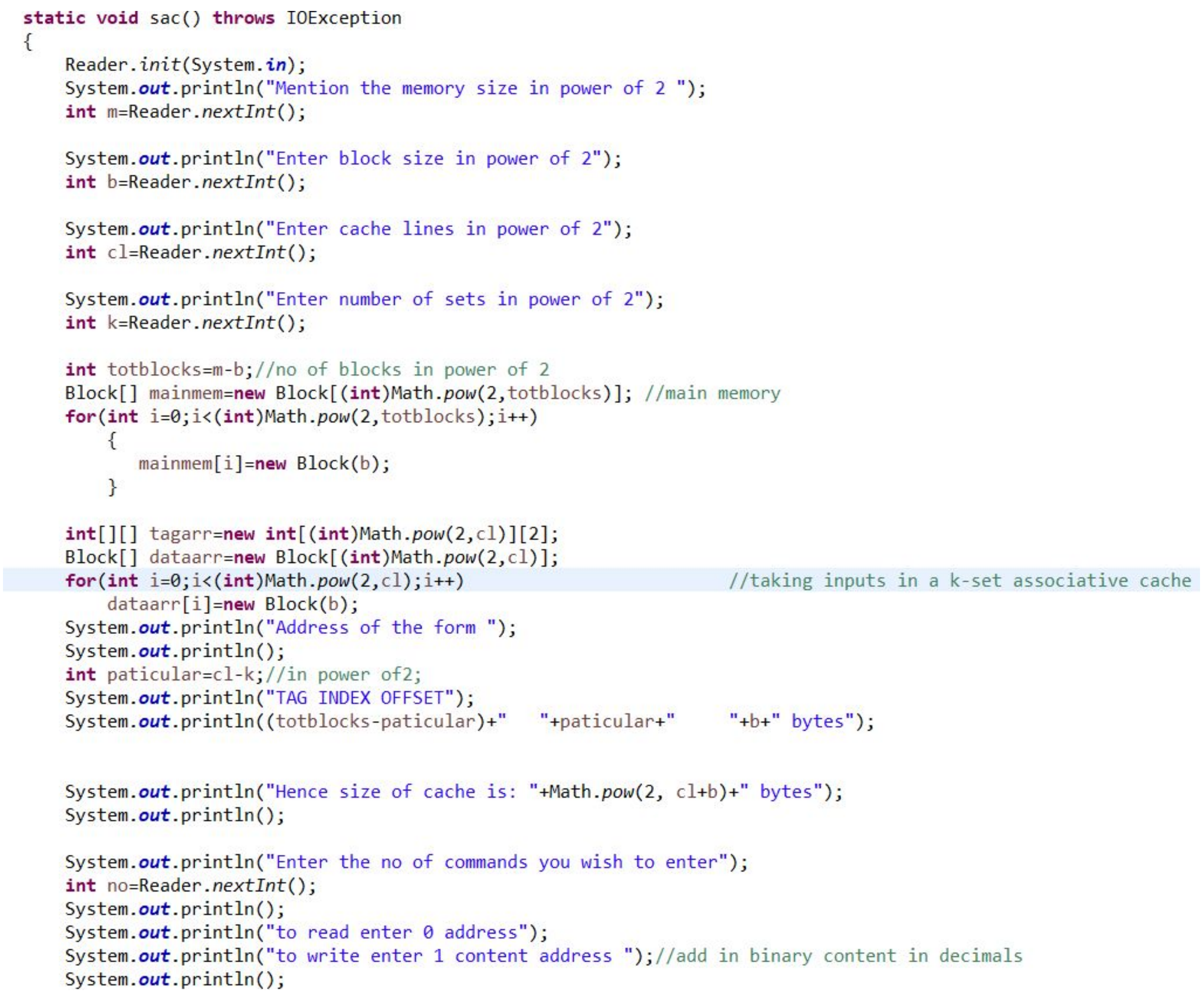


```java
static void fac() throws IOException
{
    Reader.init(System.in);
    System.out.println("Mention the memory size in power of 2 ");
    int m=Reader.nextInt();

    System.out.println("Enter block size in power of 2");
    int b=Reader.nextInt();

    System.out.println("Enter cache lines in power of 2");
    int cl=Reader.nextInt();

    int totblocks=m-b;//no of blocks in power of 2

    Block[] mainmem=new Block[(int)Math.pow(2,totblocks)]; //main memory
    for(int i=0;i<(int)Math.pow(2,totblocks);i++)
        {
           mainmem[i]=new Block(b);
        }

    int[][] tagarr=new int[(int)Math.pow(2,cl)][2];

    Block[] dataarr=new Block[(int)Math.pow(2,cl)];
    for(int i=0;i<(int)Math.pow(2,cl);i++)
        dataarr[i]=new Block(b);
                                                                    //reading inputs in a fully associative cache
    System.out.println("Address of the form ");
    System.out.println();
    System.out.println("BLOCK OFFSET");
    System.out.println((totblocks)+"     "+b+" bytes");


    System.out.println("Hence size of cache is: "+Math.pow(2, cl+b)+" bytes");
    System.out.println();
    System.out.println("Enter the no of commands you wish to enter");
    int no=Reader.nextInt();
    System.out.println();
    System.out.println("to read enter 0 address");
    System.out.println("to write enter 1 content address ");//add in binary content in decimals
    System.out.println();
```

Set Associative Cache

- Size of Memory: N<=16
- Bits for a Block:B<16
- Cache Line:Cl: <=N-B
- Sets:K<CL

INPUTS OF A SET ASSOCIATIVE CACHE



```
static void sac() throws IOException
{
    Reader.init(System.in);
    System.out.println("Mention the memory size in power of 2 ");
    int m=Reader.nextInt();

    System.out.println("Enter block size in power of 2");
    int b=Reader.nextInt();

    System.out.println("Enter cache lines in power of 2");
    int cl=Reader.nextInt();

    System.out.println("Enter number of sets in power of 2");
    int k=Reader.nextInt();

    int totblocks=m-b;//no of blocks in power of 2
    Block[] mainmem=new Block[(int)Math.pow(2,totblocks)]; //main memory
    for(int i=0;i<(int)Math.pow(2,totblocks);i++)
        {
           mainmem[i]=new Block(b);
        }

    int[][] tagarr=new int[(int)Math.pow(2,cl)][2];
    Block[] dataarr=new Block[(int)Math.pow(2,cl)];
    for(int i=0;i<(int)Math.pow(2,cl);i++)                        //taking inputs in a k-set associative cache
        dataarr[i]=new Block(b);
    System.out.println("Address of the form ");
    System.out.println();
    int paticular=cl-k;//in power of2;
    System.out.println("TAG INDEX OFFSET");
    System.out.println((totblocks-paticular)+"   "+paticular+"     "+b+" bytes");


    System.out.println("Hence size of cache is: "+Math.pow(2, cl+b)+" bytes");
    System.out.println();

    System.out.println("Enter the no of commands you wish to enter");
    int no=Reader.nextInt();
    System.out.println();
    System.out.println("to read enter 0 address");
    System.out.println("to write enter 1 content address ");//add in binary content in decimals
    System.out.println();
```

As given by the prompts in the code,before any further display the format of the instruction that needs to be input is displayed to the user,emphasising on the bits required for each part of the address in accordance to the inputs provided.

The size of the cache is also displayed.

User then chooses the number of commands they wish to perform.

Format for Load instruction:

- 1-data-address

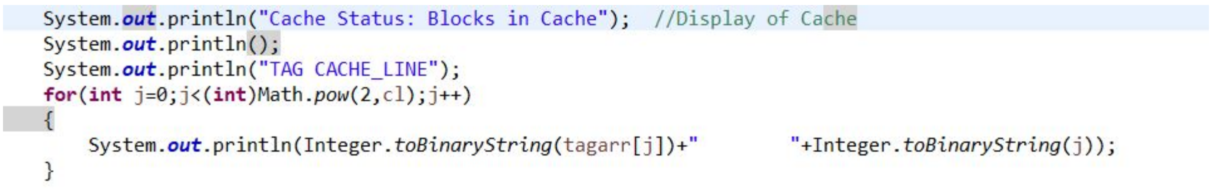
Format for Store instruction

- 0-address

After each instruction accomplishment the confirmation message is displayed,as well as there was a cache miss .

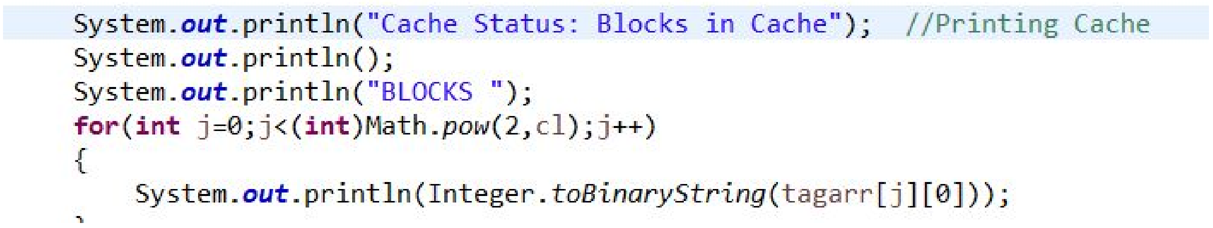
Also the entire cache is displayed after each instruction,showing the block addresses present in the caches,with their demarcation.

CACHE IN CASE OF DIRECT MAPPED CACHE



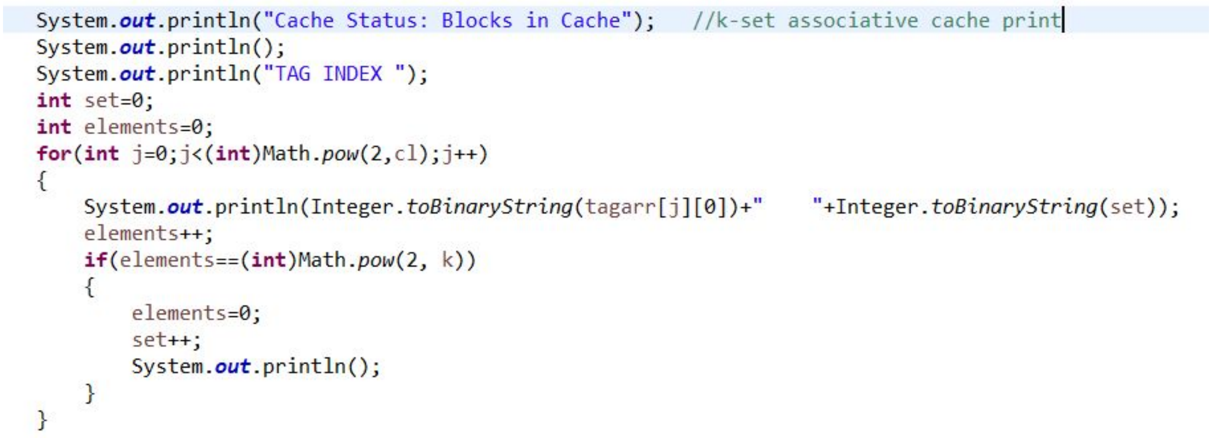
```java
System.out.println("Cache Status: Blocks in Cache");  //Display of Cache
System.out.println();
System.out.println("TAG CACHE_LINE");
for(int j=0;j<(int)Math.pow(2,cl);j++)
{
    System.out.println(Integer.toBinaryString(tagarr[j])+"        "+Integer.toBinaryString(j));
}
```

CACHE IN CASE OF FULLY ASSOCIATIVE CACHE



```java
System.out.println("Cache Status: Blocks in Cache");  //Printing Cache
System.out.println();
System.out.println("BLOCKS ");
for(int j=0;j<(int)Math.pow(2,cl);j++)
{
    System.out.println(Integer.toBinaryString(tagarr[j][0]));
```

CACHE IN CASE OF K-SET ASSOCIATIVE CACHE



```java
System.out.println("Cache Status: Blocks in Cache");   //k-set associative cache print
System.out.println();
System.out.println("TAG INDEX ");
int set=0;
int elements=0;
for(int j=0;j<(int)Math.pow(2,cl);j++)
{
    System.out.println(Integer.toBinaryString(tagarr[j][0])+"    "+Integer.toBinaryString(set));
    elements++;
    if(elements==(int)Math.pow(2, k))
    {
        elements=0;
        set++;
        System.out.println();
    }
}
```

Logic Description:

1) Fully Associative Cache

This Cache is based on the very logic explained in the book.For better understanding I would take an example and show how I have implemented this in my code.

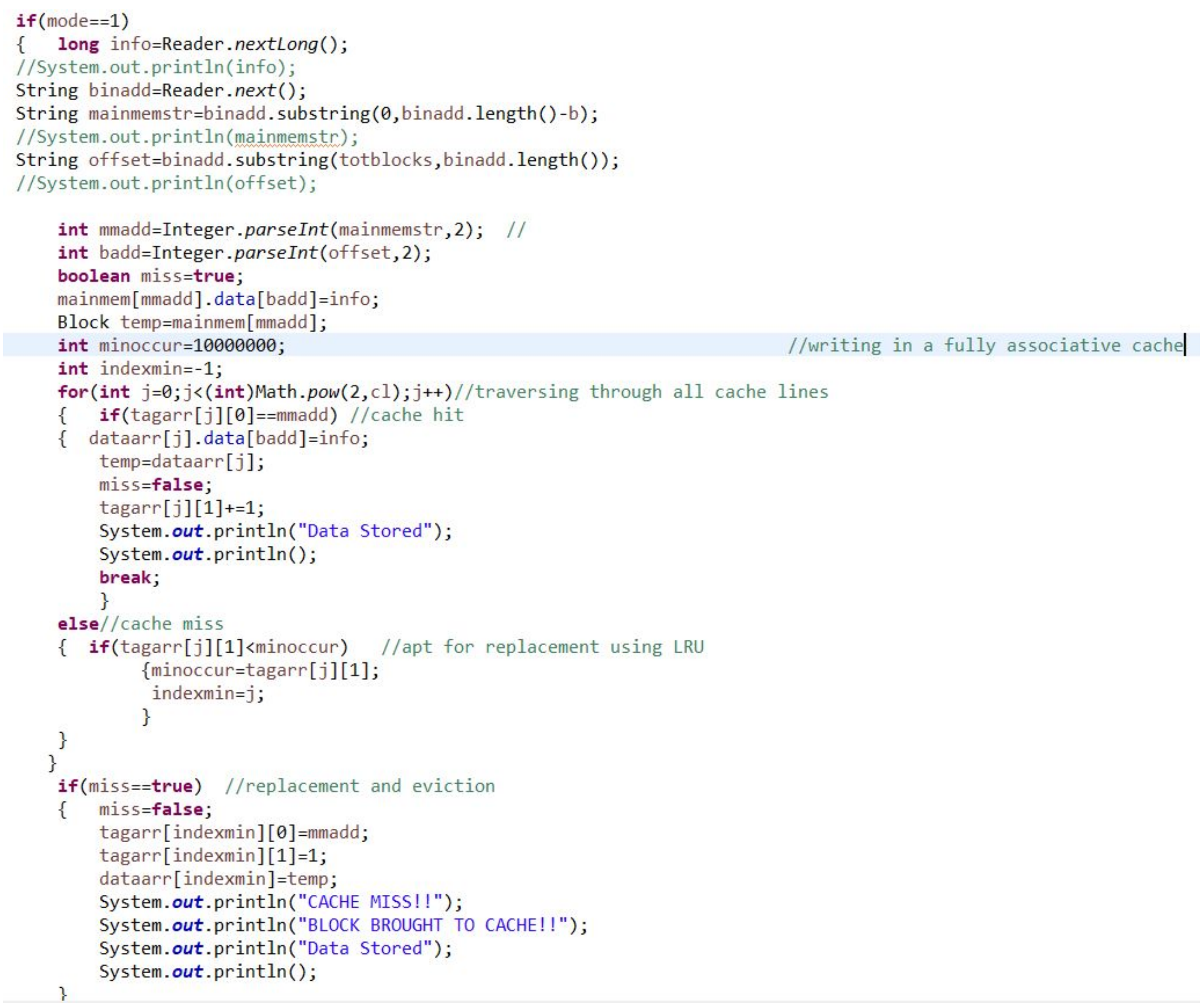
Consider,

A memory Size of 2^{16} bytes(m),Block Size of 2^{6} bytes(b) and 2^{7} cache lines(cl).

This results in the implementation of an array as the main memory,a tag array which stores the tags or block addresses and a data array which stores these blocks.Each block is also created which is in fact an array too.

Next computations are done to find the total blocks in the system,which is the size of main memory/size of a block.The resultant number when expressed in powers of 2 gives us the bits needed to represent each block uniquely.

The memory address of each block would contain 10 bits and in the address 6 bits would be allotted to the offset,that is the destination of the data within the block.



```
if(mode==1)
{   long info=Reader.nextLong();
//System.out.println(info);
String binadd=Reader.next();
String mainmemstr=binadd.substring(0,binadd.length()-b);
//System.out.println(mainmemstr);
String offset=binadd.substring(totblocks,binadd.length());
//System.out.println(offset);

    int mmadd=Integer.parseInt(mainmemstr,2);  //
    int badd=Integer.parseInt(offset,2);
    boolean miss=true;
    mainmem[mmadd].data[badd]=info;
    Block temp=mainmem[mmadd];
    int minoccur=10000000;                                    //writing in a fully associative cache
    int indexmin=-1;
    for(int j=0;j<(int)Math.pow(2,cl);j++)//traversing through all cache lines
    {   if(tagarr[j][0]==mmadd) //cache hit
    {  dataarr[j].data[badd]=info;
        temp=dataarr[j];
        miss=false;
        tagarr[j][1]+=1;
        System.out.println("Data Stored");
        System.out.println();
        break;
        }
    else//cache miss
    {  if(tagarr[j][1]<minoccur)   //apt for replacement using LRU
            {minoccur=tagarr[j][1];
             indexmin=j;
            }
    }
   }
    if(miss==true)  //replacement and eviction
    {   miss=false;
        tagarr[indexmin][0]=mmadd;
        tagarr[indexmin][1]=1;
        dataarr[indexmin]=temp;
        System.out.println("CACHE MISS!!");
        System.out.println("BLOCK BROUGHT TO CACHE!!");
        System.out.println("Data Stored");
        System.out.println();
    }
```

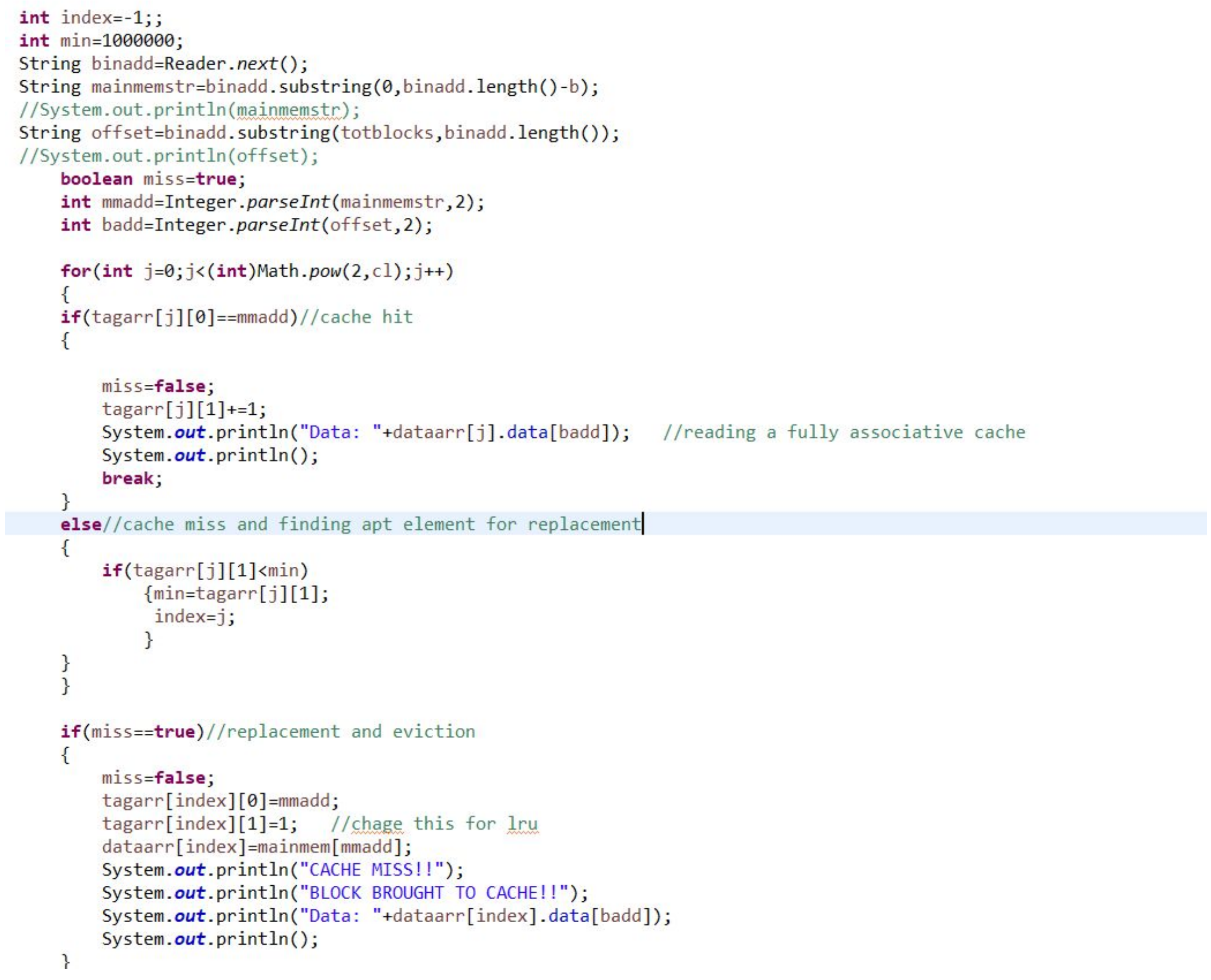
Not let me take an example and try and put forth how my code works:

Consider 1010110101100101

Breaking this memory address into 2 parts we get
Block address:1010110101
Offset:100101

The address of this block in main memory is calculated by converting 1010110101 to binary and the element in the Block that has to be accessed is found out by converting 100101 to binary.
Hence,
given a command say 1 20 1010110101100101,that is write data 20 in the memory location 1010110101100101,my code using the write back method first writes 20 in the block address 100101 which resides within a block at index location 1010110101 when converted to decimal, then in the tag array Block address:1010110101 is compared to every tag to see if the block is located there,hence traversal of the entire cache is performed to find the desired block.
If found at the corresponding address in the data array the block is accessed and further the offset is accessed and data is updated.
In case of a cache miss, that is data is not found in the cache, then the block is accessed from the memory and replaced with a particular block in the cache using the LRU technique(more details later),hence data is stored in the cache.

Similarly data can be read from the cache too.



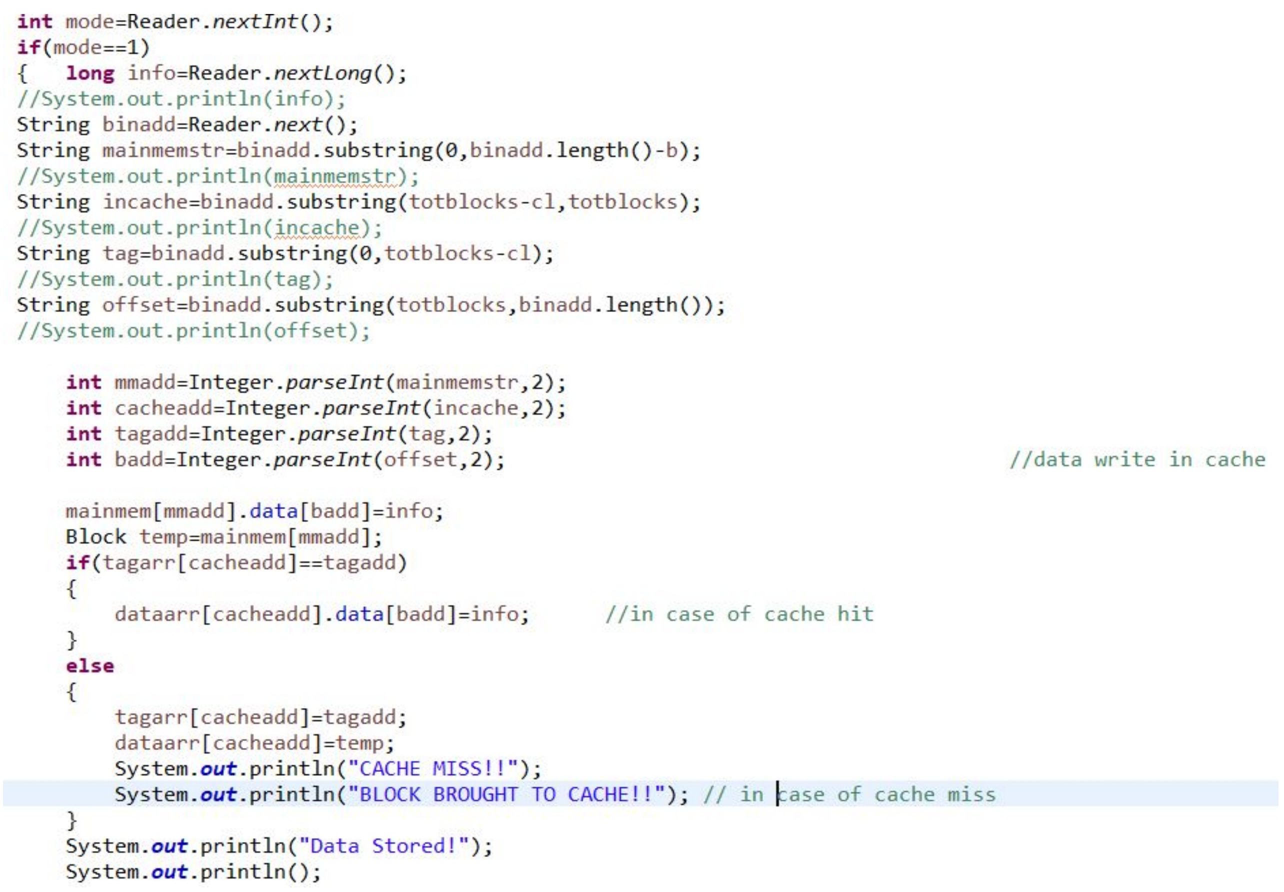
```
int index=-1;;
int min=1000000;
String binadd=Reader.next();
String mainmemstr=binadd.substring(0,binadd.length()-b);
//System.out.println(mainmemstr);
String offset=binadd.substring(totblocks,binadd.length());
//System.out.println(offset);
    boolean miss=true;
    int mmadd=Integer.parseInt(mainmemstr,2);
    int badd=Integer.parseInt(offset,2);

    for(int j=0;j<(int)Math.pow(2,cl);j++)
    {
    if(tagarr[j][0]==mmadd)//cache hit
    {

        miss=false;
        tagarr[j][1]+=1;
        System.out.println("Data: "+dataarr[j].data[badd]);   //reading a fully associative cache
        System.out.println();
        break;
    }
    else//cache miss and finding apt element for replacement
    {
        if(tagarr[j][1]<min)
            {min=tagarr[j][1];
             index=j;
            }
    }
    }

    if(miss==true)//replacement and eviction
    {
        miss=false;
        tagarr[index][0]=mmadd;
        tagarr[index][1]=1;   //chage this for lru
        dataarr[index]=mainmem[mmadd];
        System.out.println("CACHE MISS!!");
        System.out.println("BLOCK BROUGHT TO CACHE!!");
        System.out.println("Data: "+dataarr[index].data[badd]);
        System.out.println();
    }
```

LRU

Using this technique I assigned a count to each element in the tag array,whenever a block is accessed either through read or write,the count is incremented and whenever a candidate is to be found for replacement to inclusion of a block in the cache the block with the least count is chosen for replacement.

2) Direct Mapped Cache

In this cache furter optimisation to reduce access time for data is implemented, taking an example I would explain the code considering memory of 16 bits,
block size:64 bytes
and no of cache line 128
The address can hence be divided in 3 parts,tag cache line and offset since 64 is 2^6,the last six bits are reserved as offset.Total blocks are size of main memory/block size,hence here each block can be uniquely represented using 10 bits since cl as 2^7, 7 bits are used to represent a block within a cache and a particular block we wish to find can be found by comparing the tag of the block.



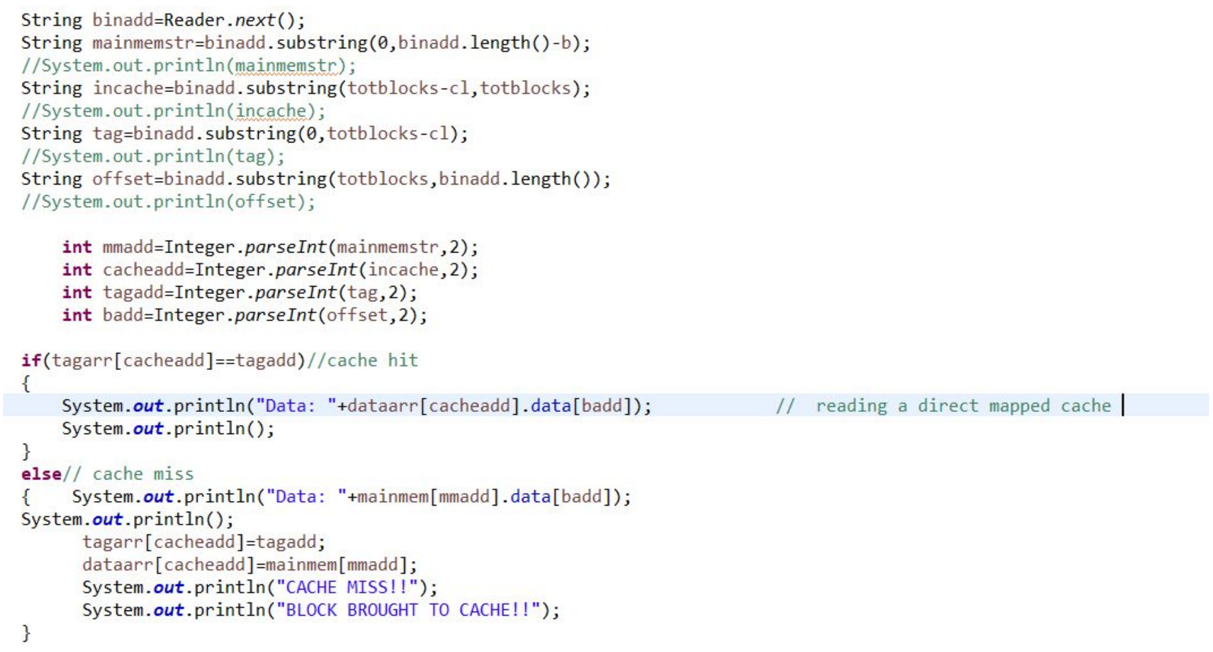
```
int mode=Reader.nextInt();
if(mode==1)
{    long info=Reader.nextLong();
//System.out.println(info);
String binadd=Reader.next();
String mainmemstr=binadd.substring(0,binadd.length()-b);
//System.out.println(mainmemstr);
String incache=binadd.substring(totblocks-cl,totblocks);
//System.out.println(incache);
String tag=binadd.substring(0,totblocks-cl);
//System.out.println(tag);
String offset=binadd.substring(totblocks,binadd.length());
//System.out.println(offset);

    int mmadd=Integer.parseInt(mainmemstr,2);
    int cacheadd=Integer.parseInt(incache,2);
    int tagadd=Integer.parseInt(tag,2);
    int badd=Integer.parseInt(offset,2);                                   //data write in cache

    mainmem[mmadd].data[badd]=info;
    Block temp=mainmem[mmadd];
    if(tagarr[cacheadd]==tagadd)
    {
        dataarr[cacheadd].data[badd]=info;      //in case of cache hit
    }
    else
    {
        tagarr[cacheadd]=tagadd;
        dataarr[cacheadd]=temp;
        System.out.println("CACHE MISS!!");
        System.out.println("BLOCK BROUGHT TO CACHE!!"); // in case of cache miss
    }
    System.out.println("Data Stored!");
    System.out.println();
```

Let's consider an example for better understanding:
Consider
1 20 1010110101100101 the instruction to write in memory,

Here 20 is stored in the block offset 100101 in the block 1010110101.
The way of accessal of this memory block is what differs in the Direct Mapped Cache.

In my code this address is further divided in subparts. 101 0110101 100101: that is tag cache line and offset.The cache line contains only a part of the memory address of the block, here 7 bits.Hence a series of multiple blocks which differ in only the first 3 bits of the memory address have a single place for residence in the cache line.So the block which is being addressed to in the command lies in the decimal representation of 1010110101 in the main memory.

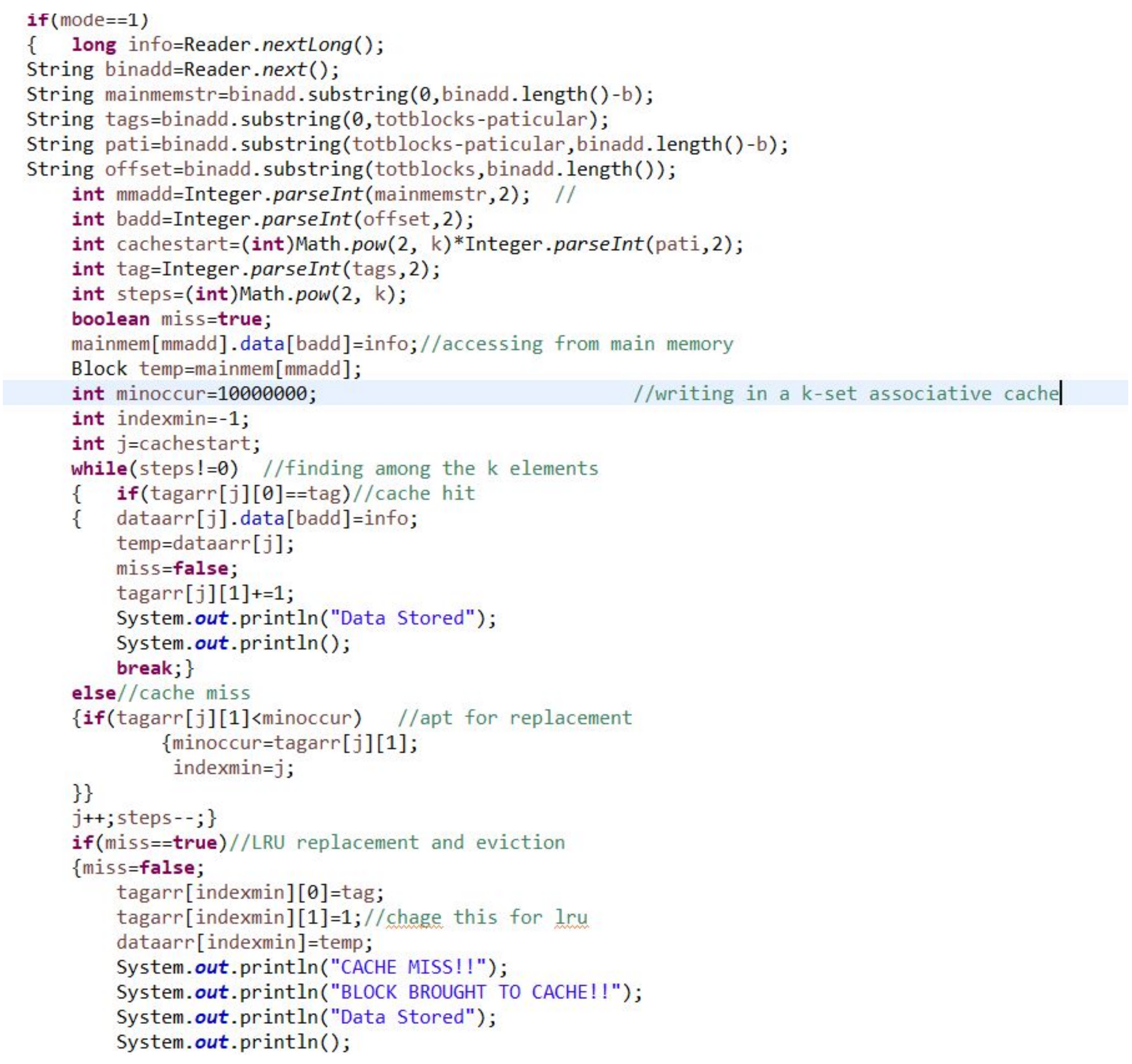
If the cache exists in the cache line, then it would reside in the decimal representation of the address 0110101 in the cache.On accessing the tag stored in the location if one finds the tag to match the one given in the address the code then accesses the data array and hence accesses the offset within the block and manipulates the data as per the instruction.
If there is a cache miss,then in such a case the block is brought from the main memory and the tag and data overwritten.
Similarly, in the case of a read instruction a cache is accessed in a similar fashion.



```
String binadd=Reader.next();
String mainmemstr=binadd.substring(0,binadd.length()-b);
//System.out.println(mainmemstr);
String incache=binadd.substring(totblocks-cl,totblocks);
//System.out.println(incache);
String tag=binadd.substring(0,totblocks-cl);
//System.out.println(tag);
String offset=binadd.substring(totblocks,binadd.length());
//System.out.println(offset);

    int mmadd=Integer.parseInt(mainmemstr,2);
    int cacheadd=Integer.parseInt(incache,2);
    int tagadd=Integer.parseInt(tag,2);
    int badd=Integer.parseInt(offset,2);

if(tagarr[cacheadd]==tagadd)//cache hit
{
    System.out.println("Data: "+dataarr[cacheadd].data[badd]);            //  reading a direct mapped cache
    System.out.println();
}
else// cache miss
{     System.out.println("Data: "+mainmem[mmadd].data[badd]);
System.out.println();
      tagarr[cacheadd]=tagadd;
      dataarr[cacheadd]=mainmem[mmadd];
      System.out.println("CACHE MISS!!");
      System.out.println("BLOCK BROUGHT TO CACHE!!");
}
```

3)Set Associative Cache

A set associative cache is a combination of both a direct mapped cache and a fully associative cache, hence the code is also a combination of the 2.
Consider an example:
Say the size of main memory is 16 bits,block size is 64 bytes and there are 128 cache lines. Let's assume it is an 8 Set associative cache, hence there would be eight blocks in a single set.

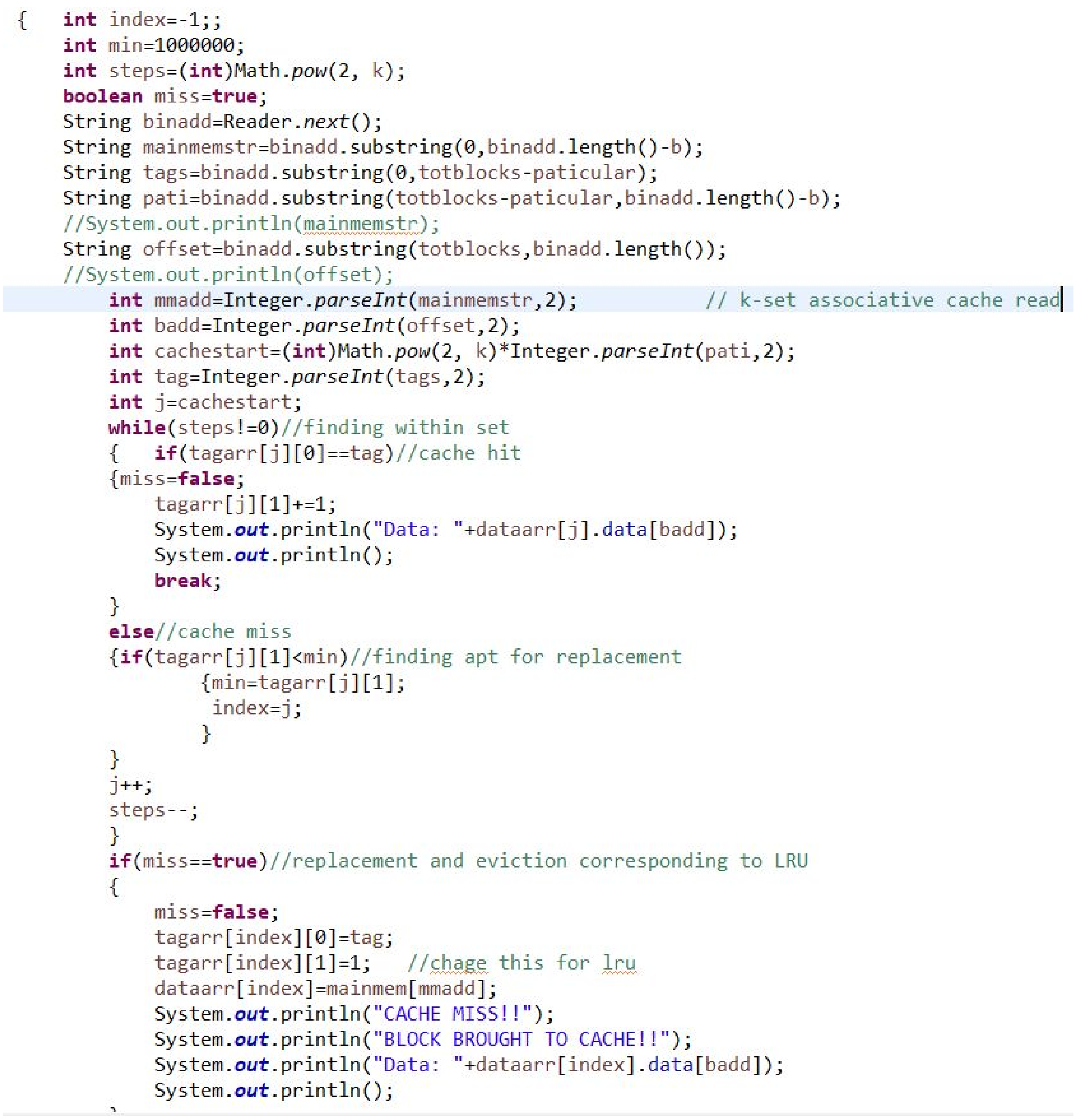


```
if(mode==1)
{   long info=Reader.nextLong();
String binadd=Reader.next();
String mainmemstr=binadd.substring(0,binadd.length()-b);
String tags=binadd.substring(0,totblocks-paticular);
String pati=binadd.substring(totblocks-paticular,binadd.length()-b);
String offset=binadd.substring(totblocks,binadd.length());
    int mmadd=Integer.parseInt(mainmemstr,2);  //
    int badd=Integer.parseInt(offset,2);
    int cachestart=(int)Math.pow(2, k)*Integer.parseInt(pati,2);
    int tag=Integer.parseInt(tags,2);
    int steps=(int)Math.pow(2, k);
    boolean miss=true;
    mainmem[mmadd].data[badd]=info;//accessing from main memory
    Block temp=mainmem[mmadd];
    int minoccur=10000000;                          //writing in a k-set associative cache
    int indexmin=-1;
    int j=cachestart;
    while(steps!=0)  //finding among the k elements
    {   if(tagarr[j][0]==tag)//cache hit
    {   dataarr[j].data[badd]=info;
        temp=dataarr[j];
        miss=false;
        tagarr[j][1]+=1;
        System.out.println("Data Stored");
        System.out.println();
        break;}
    else//cache miss
    {if(tagarr[j][1]<minoccur)   //apt for replacement
            {minoccur=tagarr[j][1];
             indexmin=j;
    }}
    j++;steps--;}
    if(miss==true)//LRU replacement and eviction
    {miss=false;
        tagarr[indexmin][0]=tag;
        tagarr[indexmin][1]=1;//chage this for lru
        dataarr[indexmin]=temp;
        System.out.println("CACHE MISS!!");
        System.out.println("BLOCK BROUGHT TO CACHE!!");
        System.out.println("Data Stored");
        System.out.println();
```

The working of the code is as follows:
Assume the command
1 20 1010110101010010 which demands the storage of 20 in the memory location 1010110101010010.
This address has been divided into various subparts:
101011 0101 010010, that is Tag ,Index and Offset.While offset specifies the unique location within a block where the data lies,the index refers to its place in the cache and the tag is needed to uniquely identify the specific block.Hence the code first accesses the start of the set though the decimal representation of the index,then it compares the subsequent k elements with the tag to find the unique block.Once found the very index is accessed in the data array and using the offset the data at the specific location is manipulated.

In case of a cache miss, the LRU method is used to find the block apt for replacement and the desired block is brought from the memory.

The position of the desired block in the memory can be found by the decimal representation of the tag+offset in the main memory.Similarly in case of a read instruction data is accessed from the memory.



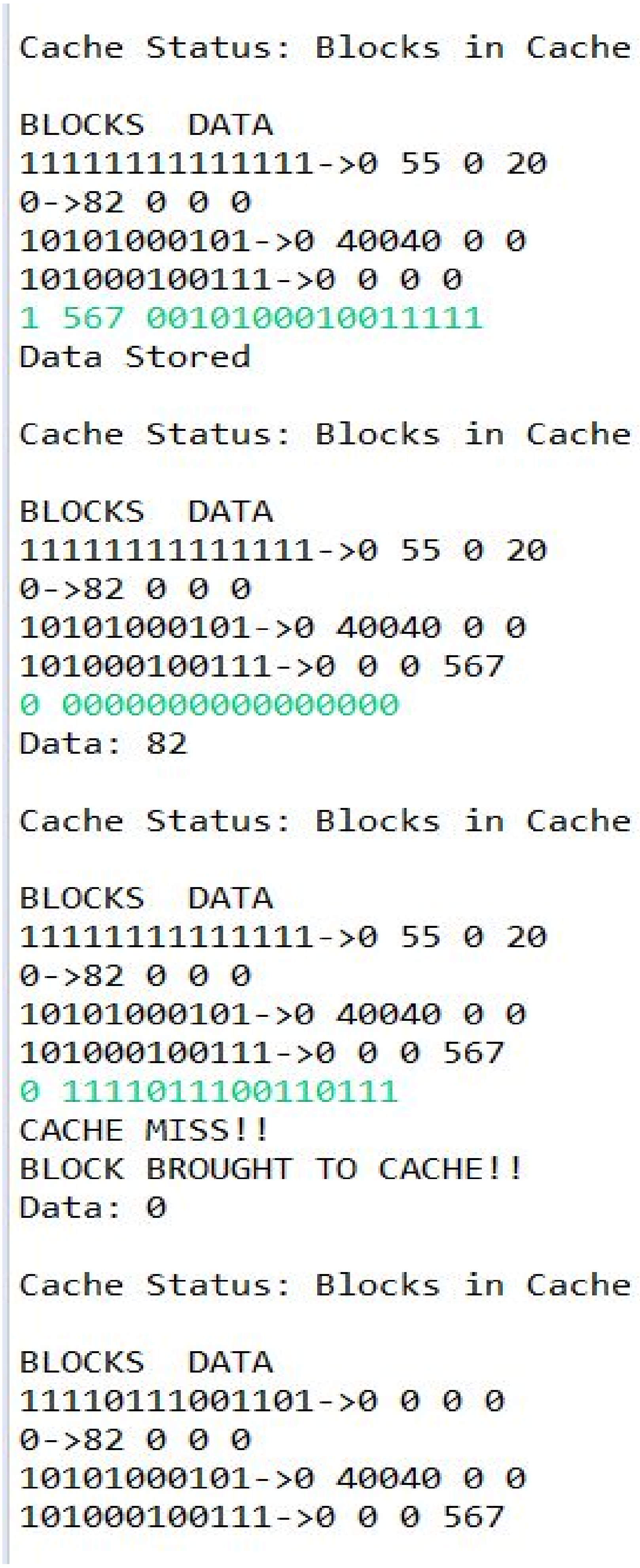
```
{   int index=-1;;
    int min=1000000;
    int steps=(int)Math.pow(2, k);
    boolean miss=true;
    String binadd=Reader.next();
    String mainmemstr=binadd.substring(0,binadd.length()-b);
    String tags=binadd.substring(0,totblocks-paticular);
    String pati=binadd.substring(totblocks-paticular,binadd.length()-b);
    //System.out.println(mainmemstr);
    String offset=binadd.substring(totblocks,binadd.length());
    //System.out.println(offset);
        int mmadd=Integer.parseInt(mainmemstr,2);           // k-set associative cache read
        int badd=Integer.parseInt(offset,2);
        int cachestart=(int)Math.pow(2, k)*Integer.parseInt(pati,2);
        int tag=Integer.parseInt(tags,2);
        int j=cachestart;
        while(steps!=0)//finding within set
        {   if(tagarr[j][0]==tag)//cache hit
        {miss=false;
            tagarr[j][1]+=1;
            System.out.println("Data: "+dataarr[j].data[badd]);
            System.out.println();
            break;
        }
        else//cache miss
        {if(tagarr[j][1]<min)//finding apt for replacement
                {min=tagarr[j][1];
                 index=j;
                }
        }
        j++;
        steps--;
        }
        if(miss==true)//replacement and eviction corresponding to LRU
        {
            miss=false;
            tagarr[index][0]=tag;
            tagarr[index][1]=1;   //chage this for lru
            dataarr[index]=mainmem[mmadd];
            System.out.println("CACHE MISS!!");
            System.out.println("BLOCK BROUGHT TO CACHE!!");
            System.out.println("Data: "+dataarr[index].data[badd]);
            System.out.println();
```

The above is exactly how my code functions.

Working Programs:

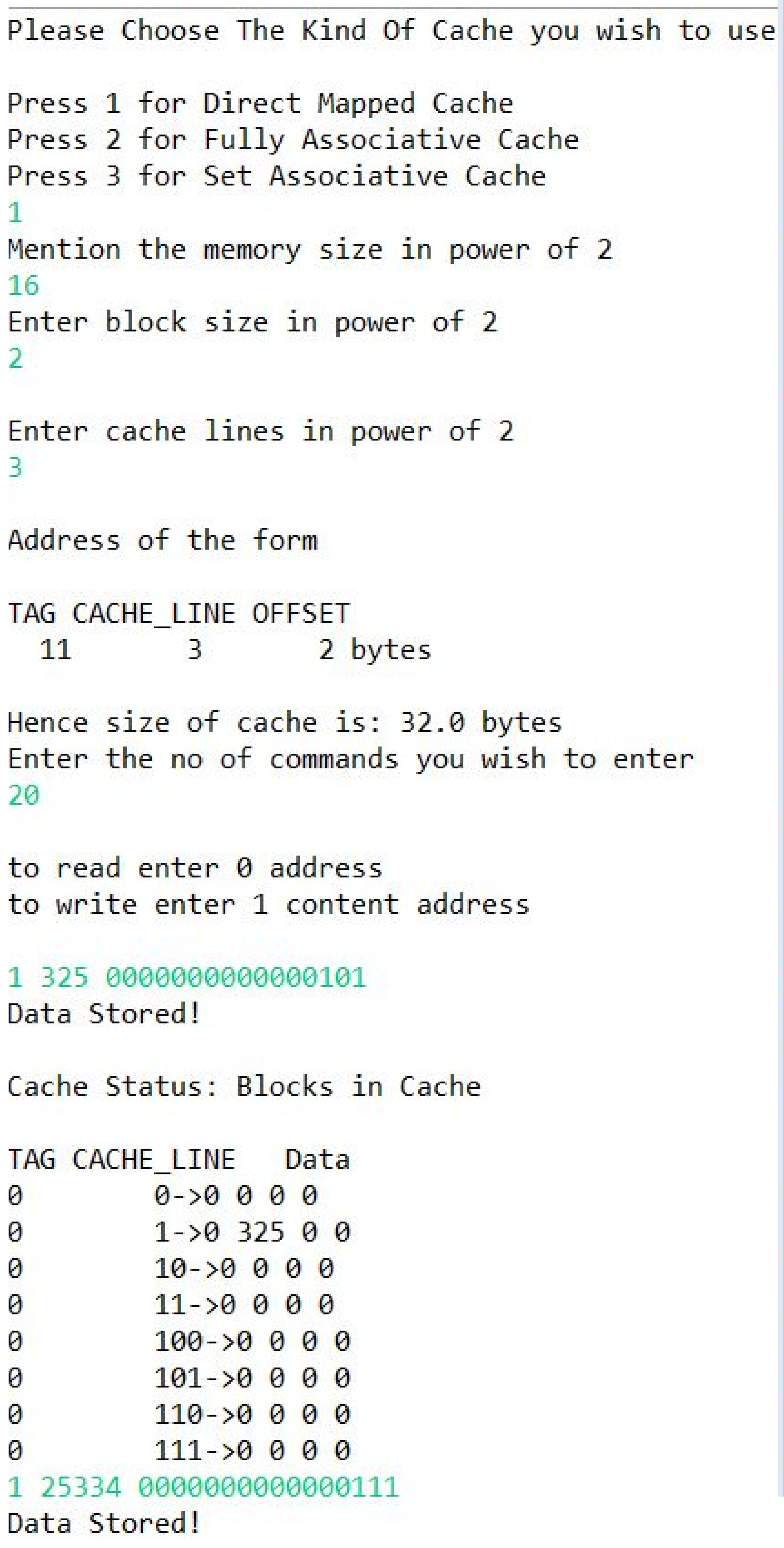
Fully Associative Cache

```
Please Choose The Kind Of Cache you wish to use:

Press 1 for Direct Mapped Cache
Press 2 for Fully Associative Cache
Press 3 for Set Associative Cache
2
Mention the memory size in power of 2
16
Enter block size in power of 2
2
Enter cache lines in power of 2
2
Address of the form

BLOCK OFFSET
14     2 bytes
Hence size of cache is: 16.0 bytes

Enter the no of commands you wish to enter
20

to read enter 0 address
to write enter 1 content address

1
20 1111111111111111
CACHE MISS!!
BLOCK BROUGHT TO CACHE!!
Data Stored

Cache Status: Blocks in Cache

BLOCKS  DATA
11111111111111->0 0 0 20
0->0 0 0 0
0->0 0 0 0
0->0 0 0 0
1 82 0000000000000000
Data Stored

Cache Status: Blocks in Cache

BLOCKS  DATA
11111111111111->0 0 0 20
0->82 0 0 0
0->0 0 0 0
0->0 0 0 0
1 55 1111111111111101
Data Stored

Cache Status: Blocks in Cache

BLOCKS  DATA
11111111111111->0 55 0 20
0->82 0 0 0
0->0 0 0 0
0->0 0 0 0
1 40040 0001010100010101
CACHE MISS!!
BLOCK BROUGHT TO CACHE!!
Data Stored

Cache Status: Blocks in Cache

BLOCKS  DATA
11111111111111->0 55 0 20
0->82 0 0 0
10101000101->0 40040 0 0
0->0 0 0 0
0 0001010100010100
Data: 0

Cache Status: Blocks in Cache

BLOCKS  DATA
11111111111111->0 55 0 20
0->82 0 0 0
10101000101->0 40040 0 0
0->0 0 0 0
0 0010100010011101
CACHE MISS!!
BLOCK BROUGHT TO CACHE!!
Data: 0
```

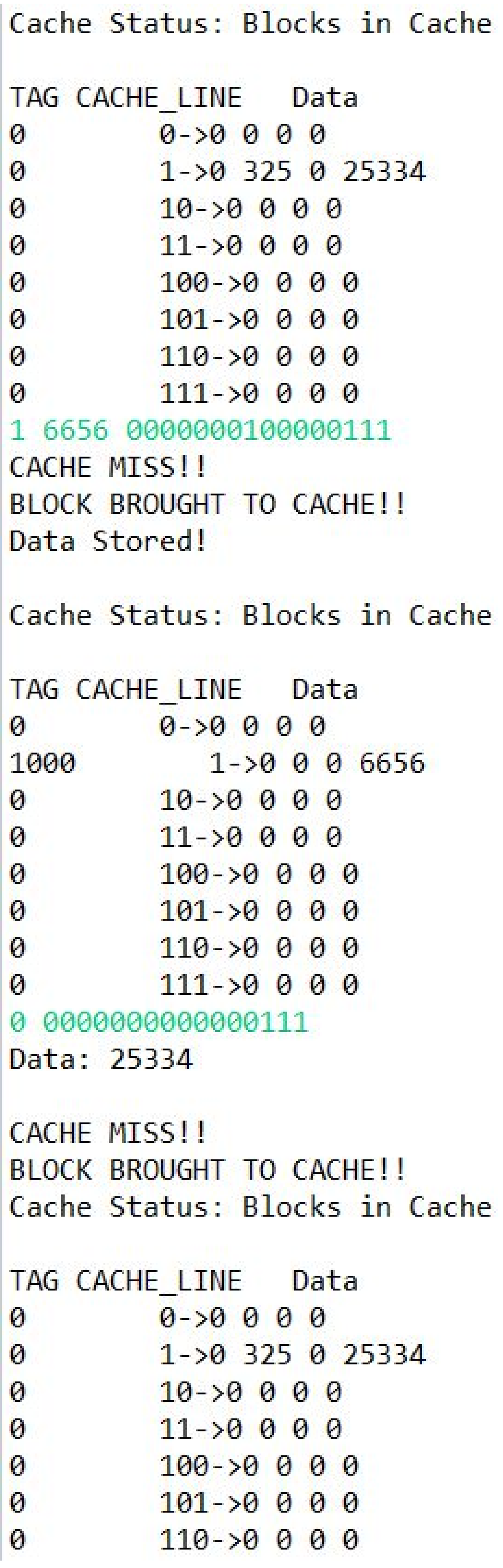


```
Cache Status: Blocks in Cache

BLOCKS  DATA
11111111111111->0 55 0 20
0->82 0 0 0
10101000101->0 40040 0 0
101000100111->0 0 0 0
1 567 0010100010011111
Data Stored

Cache Status: Blocks in Cache

BLOCKS  DATA
11111111111111->0 55 0 20
0->82 0 0 0
10101000101->0 40040 0 0
101000100111->0 0 0 567
0 0000000000000000
Data: 82

Cache Status: Blocks in Cache

BLOCKS  DATA
11111111111111->0 55 0 20
0->82 0 0 0
10101000101->0 40040 0 0
101000100111->0 0 0 567
0 1111011100110111
CACHE MISS!!
BLOCK BROUGHT TO CACHE!!
Data: 0

Cache Status: Blocks in Cache

BLOCKS  DATA
11110111001101->0 0 0 0
0->82 0 0 0
10101000101->0 40040 0 0
101000100111->0 0 0 567
```

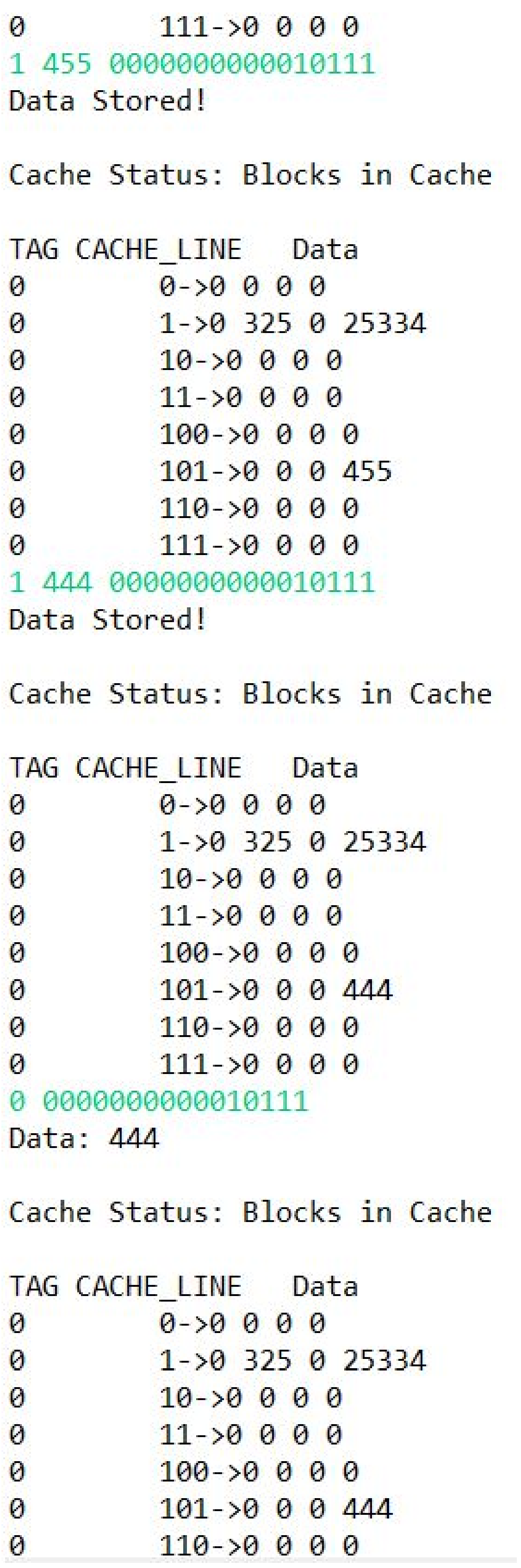
Direct Mapped Cache



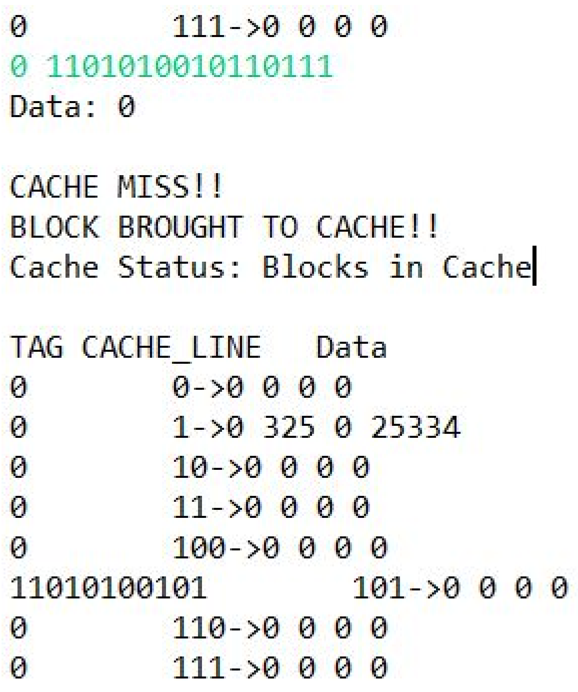
```
Please Choose The Kind Of Cache you wish to use

Press 1 for Direct Mapped Cache
Press 2 for Fully Associative Cache
Press 3 for Set Associative Cache
1
Mention the memory size in power of 2
16
Enter block size in power of 2
2

Enter cache lines in power of 2
3

Address of the form

TAG CACHE_LINE OFFSET
  11        3       2 bytes

Hence size of cache is: 32.0 bytes
Enter the no of commands you wish to enter
20

to read enter 0 address
to write enter 1 content address

1 325 0000000000000101
Data Stored!

Cache Status: Blocks in Cache

TAG CACHE_LINE   Data
0        0->0 0 0 0
0        1->0 325 0 0
0        10->0 0 0 0
0        11->0 0 0 0
0        100->0 0 0 0
0        101->0 0 0 0
0        110->0 0 0 0
0        111->0 0 0 0
1 25334 0000000000000111
Data Stored!
```



```
Cache Status: Blocks in Cache

TAG CACHE_LINE   Data
0        0->0 0 0 0
0        1->0 325 0 25334
0        10->0 0 0 0
0        11->0 0 0 0
0        100->0 0 0 0
0        101->0 0 0 0
0        110->0 0 0 0
0        111->0 0 0 0
1 6656 0000000100000111
CACHE MISS!!
BLOCK BROUGHT TO CACHE!!
Data Stored!

Cache Status: Blocks in Cache

TAG CACHE_LINE   Data
0        0->0 0 0 0
1000        1->0 0 0 6656
0        10->0 0 0 0
0        11->0 0 0 0
0        100->0 0 0 0
0        101->0 0 0 0
0        110->0 0 0 0
0        111->0 0 0 0
0 0000000000000111
Data: 25334

CACHE MISS!!
BLOCK BROUGHT TO CACHE!!
Cache Status: Blocks in Cache

TAG CACHE_LINE   Data
0        0->0 0 0 0
0        1->0 325 0 25334
0        10->0 0 0 0
0        11->0 0 0 0
0        100->0 0 0 0
0        101->0 0 0 0
0        110->0 0 0 0
```

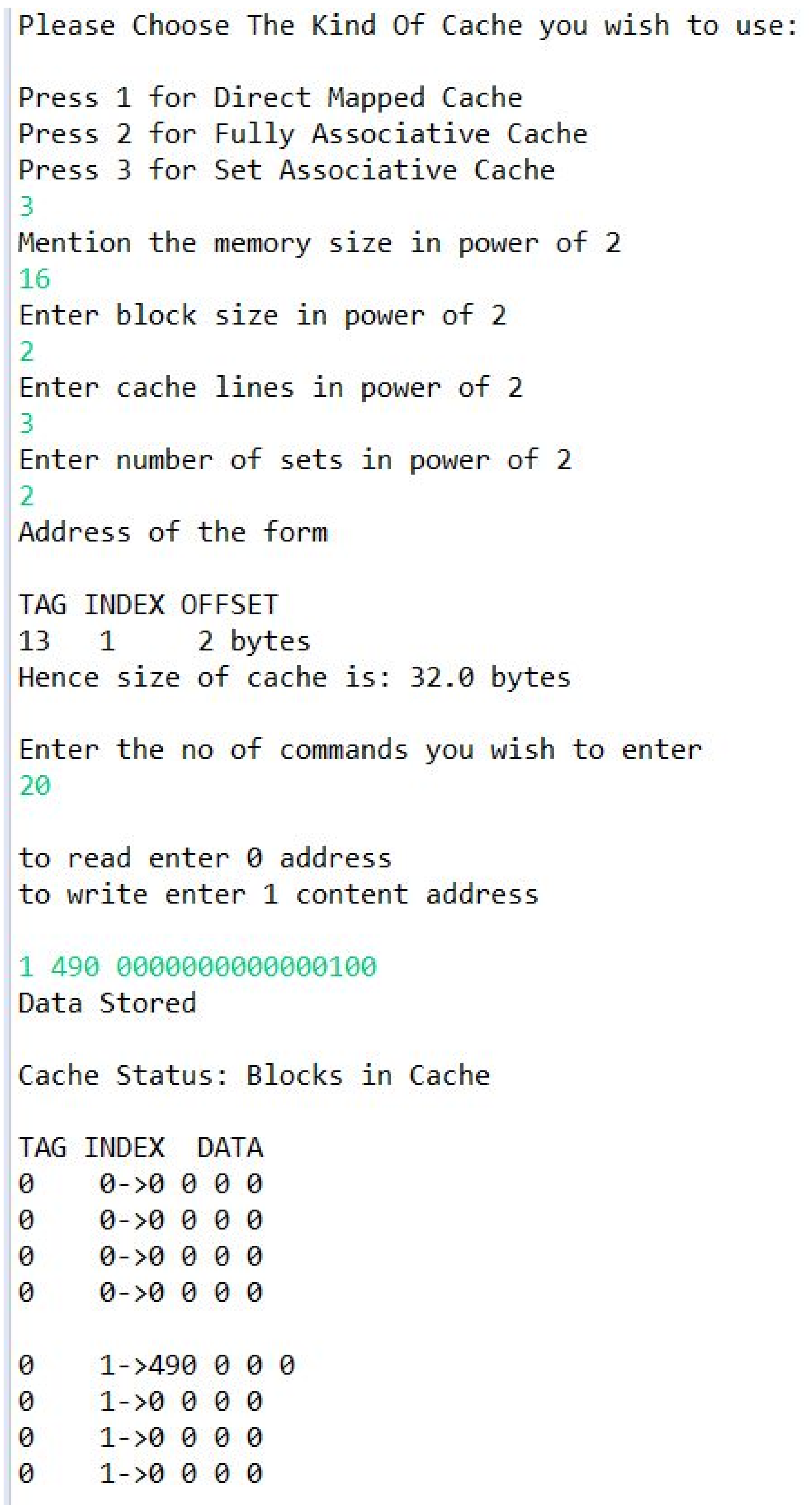


```
0        111->0 0 0 0
1 455 0000000000010111
Data Stored!

Cache Status: Blocks in Cache

TAG CACHE_LINE   Data
0        0->0 0 0 0
0        1->0 325 0 25334
0        10->0 0 0 0
0        11->0 0 0 0
0        100->0 0 0 0
0        101->0 0 0 455
0        110->0 0 0 0
0        111->0 0 0 0
1 444 0000000000010111
Data Stored!

Cache Status: Blocks in Cache

TAG CACHE_LINE   Data
0        0->0 0 0 0
0        1->0 325 0 25334
0        10->0 0 0 0
0        11->0 0 0 0
0        100->0 0 0 0
0        101->0 0 0 444
0        110->0 0 0 0
0        111->0 0 0 0
0 0000000000010111
Data: 444

Cache Status: Blocks in Cache

TAG CACHE_LINE   Data
0        0->0 0 0 0
0        1->0 325 0 25334
0        10->0 0 0 0
0        11->0 0 0 0
0        100->0 0 0 0
0        101->0 0 0 444
0        110->0 0 0 0
```

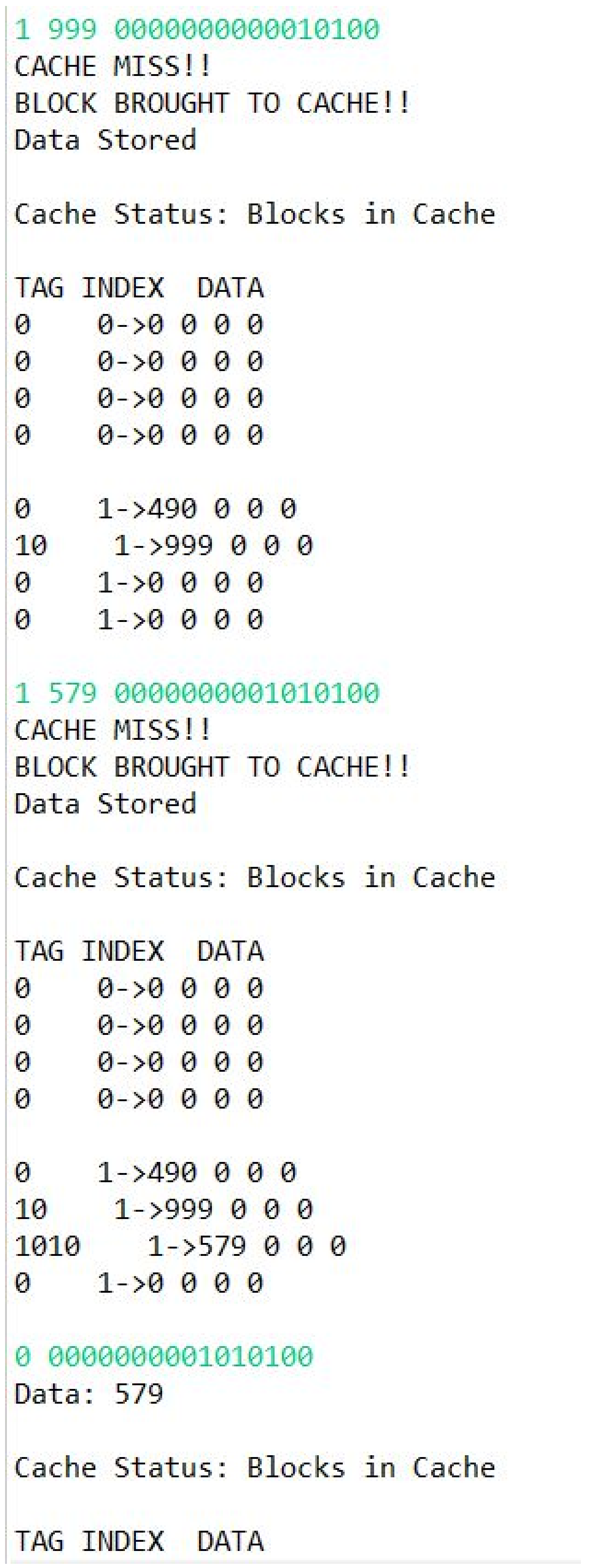


```
0        111->0 0 0 0
0 1101010010110111
Data: 0

CACHE MISS!!
BLOCK BROUGHT TO CACHE!!
Cache Status: Blocks in Cache

TAG CACHE_LINE   Data
0        0->0 0 0 0
0        1->0 325 0 25334
0        10->0 0 0 0
0        11->0 0 0 0
0        100->0 0 0 0
11010100101        101->0 0 0 0
0        110->0 0 0 0
0        111->0 0 0 0
```

Set Associative Cache



```
Please Choose The Kind Of Cache you wish to use:

Press 1 for Direct Mapped Cache
Press 2 for Fully Associative Cache
Press 3 for Set Associative Cache
3
Mention the memory size in power of 2
16
Enter block size in power of 2
2
Enter cache lines in power of 2
3
Enter number of sets in power of 2
2
Address of the form

TAG INDEX OFFSET
13   1     2 bytes
Hence size of cache is: 32.0 bytes

Enter the no of commands you wish to enter
20

to read enter 0 address
to write enter 1 content address

1 490 0000000000000100
Data Stored

Cache Status: Blocks in Cache

TAG INDEX  DATA
0    0->0 0 0 0
0    0->0 0 0 0
0    0->0 0 0 0
0    0->0 0 0 0

0    1->490 0 0 0
0    1->0 0 0 0
0    1->0 0 0 0
0    1->0 0 0 0
```



```
1 999 0000000000010100
CACHE MISS!!
BLOCK BROUGHT TO CACHE!!
Data Stored

Cache Status: Blocks in Cache

TAG INDEX  DATA
0    0->0 0 0 0
0    0->0 0 0 0
0    0->0 0 0 0
0    0->0 0 0 0

0    1->490 0 0 0
10    1->999 0 0 0
0    1->0 0 0 0
0    1->0 0 0 0

1 579 0000000001010100
CACHE MISS!!
BLOCK BROUGHT TO CACHE!!
Data Stored

Cache Status: Blocks in Cache

TAG INDEX  DATA
0    0->0 0 0 0
0    0->0 0 0 0
0    0->0 0 0 0
0    0->0 0 0 0

0    1->490 0 0 0
10    1->999 0 0 0
1010    1->579 0 0 0
0    1->0 0 0 0

0 0000000001010100
Data: 579

Cache Status: Blocks in Cache

TAG INDEX  DATA
```

```
0    0->0 0 0 0
0    0->0 0 0 0
0    0->0 0 0 0
0    0->0 0 0 0

0    1->490 0 0 0
10    1->999 0 0 0
1010    1->579 0 0 0
0    1->0 0 0 0

1 789 0000011001010100
CACHE MISS!!
BLOCK BROUGHT TO CACHE!!
Data Stored

Cache Status: Blocks in Cache

TAG INDEX  DATA
0    0->0 0 0 0
0    0->0 0 0 0
0    0->0 0 0 0
0    0->0 0 0 0

0    1->490 0 0 0
10    1->999 0 0 0
1010    1->579 0 0 0
11001010    1->789 0 0 0

1 404 1101011101001010
CACHE MISS!!
BLOCK BROUGHT TO CACHE!!
Data Stored

Cache Status: Blocks in Cache

TAG INDEX  DATA
1101011101001    0->0 0 404 0
0    0->0 0 0 0
0    0->0 0 0 0
0    0->0 0 0 0
```

```
0    1->490 0 0 0
10    1->999 0 0 0
1010    1->579 0 0 0
11001010    1->789 0 0 0

0 0000000000000011
Data: 0

Cache Status: Blocks in Cache

TAG INDEX  DATA
1101011101001    0->0 0 404 0
0    0->0 0 0 0
0    0->0 0 0 0
0    0->0 0 0 0

0    1->490 0 0 0
10    1->999 0 0 0
1010    1->579 0 0 0
11001010    1->789 0 0 0

0 1110101101011110
CACHE MISS!!
BLOCK BROUGHT TO CACHE!!
Data: 0

Cache Status: Blocks in Cache

TAG INDEX  DATA
1101011101001    0->0 0 404 0
0    0->0 0 0 0
0    0->0 0 0 0
0    0->0 0 0 0

1110101101011    1->0 0 0 0
10    1->999 0 0 0
1010    1->579 0 0 0
11001010    1->789 0 0 0

1 980 1110101101011110
Data Stored
```

```
Cache Status: Blocks in Cache

TAG INDEX  DATA
1101011101001    0->0 0 404 0
0    0->0 0 0 0
0    0->0 0 0 0
0    0->0 0 0 0

1110101101011    1->0 0 980 0
10    1->999 0 0 0
1010    1->579 0 0 0
11001010    1->789 0 0 0
```
