

COMPUTER ORGANIZATION

DOCUMENT

SAMAKSH GUPTA

2019200

GROUP 3

ASSUMPTIONS I HAVE MADE

The programme is built as a 32-bit machine system. Hence the Physical address generated is of 32 bits. This 32-bit address is changed to a logical address as per mapping which I will discuss individually. But all in all, as per the rules of each mapping the 32-bit address is divided to generate Tag, Cache Line (In direct), Block Offset. The input first asks you for the cache size. Then it asks for Number of Cache Lines. Then it asks for the Line size or block size. Note: Line size= Block size. Since we don't have to maintain a main memory therefore, we don't need data on number of block lines.

There is No Main Memory, as instructed "No need to maintain a main memory."

The replacement happens by RANDOM policy. Since, nothing is mentioned to us, I didn't use LRU or FIFO but I went with RANDOM.

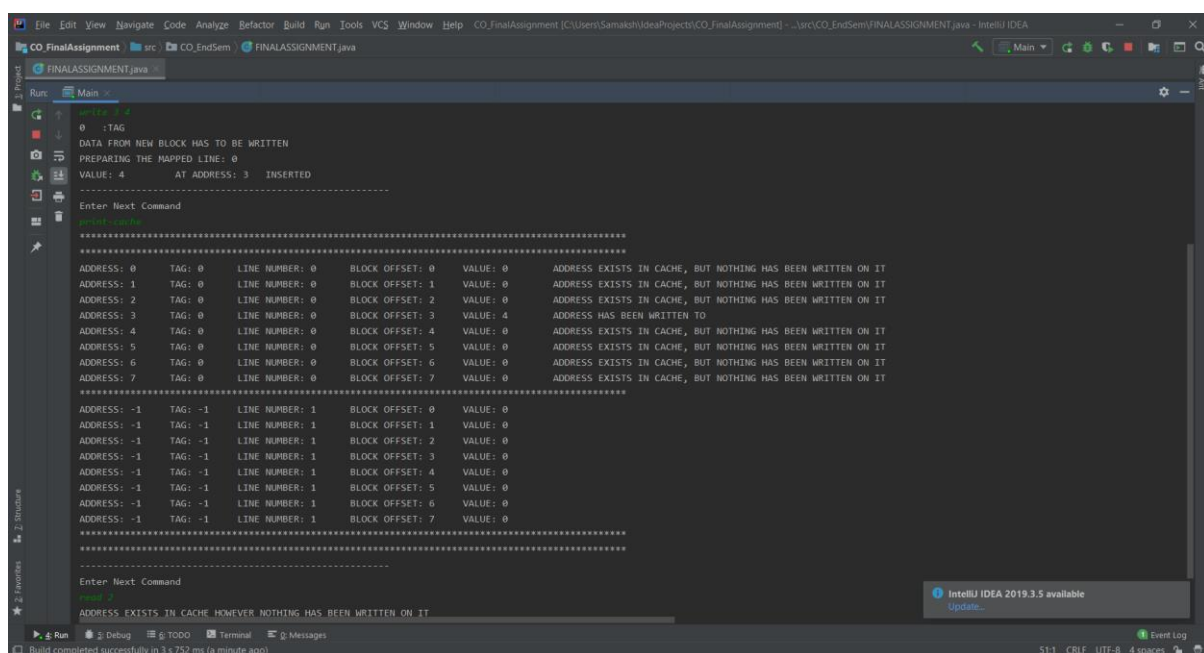
I have initialised the Arrays with address -1 and value 0 and tag -1. Thus, you cannot write the value 0 in any of the mapping. There is no reason to chose 0. I could've picked -1. But zero just describes it best. Cause literally nothing is there. You can however, write and read from address 0.

When you write on an address, you must know that you cannot write on a single address unless the tag is already present in the cache. If the tag isn't

matched in the cache then the whole block has to come in cache and then you can write on the address.

Say, you want to write 3 4. Means you want to write 4 on address 3. Previously none of the addresses 0 or 1 or 2 are present in cache. Thus, you just don't bring address 3 but also 0 1 and 2..till 7 (Block Size=8). Hence, if your next command is to read 0 or 1 or 2.. or 7. It will not be a cache Miss. But It will print that the address is present but nothing has been written on it.

But the addresses on which You have written, will show the proper address number with Value in the print Whole Cache command. Or a cache hit if you simply read from that address on which you wrote.



```
Run: Main
0 :TAG
DATA FROM NEW BLOCK HAS TO BE WRITTEN
PREPARING THE MAPPED LINE: 0
VALUE: 4 AT ADDRESS: 3 INSERTED
Enter Next Command
printWholeCache
=====
ADDRESS: 0 TAG: 0 LINE NUMBER: 0 BLOCK OFFSET: 0 VALUE: 0 ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
ADDRESS: 1 TAG: 0 LINE NUMBER: 0 BLOCK OFFSET: 1 VALUE: 0 ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
ADDRESS: 2 TAG: 0 LINE NUMBER: 0 BLOCK OFFSET: 2 VALUE: 0 ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
ADDRESS: 3 TAG: 0 LINE NUMBER: 0 BLOCK OFFSET: 3 VALUE: 4 ADDRESS HAS BEEN WRITTEN TO
ADDRESS: 4 TAG: 0 LINE NUMBER: 0 BLOCK OFFSET: 4 VALUE: 0 ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
ADDRESS: 5 TAG: 0 LINE NUMBER: 0 BLOCK OFFSET: 5 VALUE: 0 ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
ADDRESS: 6 TAG: 0 LINE NUMBER: 0 BLOCK OFFSET: 6 VALUE: 0 ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
ADDRESS: 7 TAG: 0 LINE NUMBER: 0 BLOCK OFFSET: 7 VALUE: 0 ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
=====
ADDRESS: -1 TAG: -1 LINE NUMBER: 1 BLOCK OFFSET: 0 VALUE: 0
ADDRESS: -1 TAG: -1 LINE NUMBER: 1 BLOCK OFFSET: 1 VALUE: 0
ADDRESS: -1 TAG: -1 LINE NUMBER: 1 BLOCK OFFSET: 2 VALUE: 0
ADDRESS: -1 TAG: -1 LINE NUMBER: 1 BLOCK OFFSET: 3 VALUE: 0
ADDRESS: -1 TAG: -1 LINE NUMBER: 1 BLOCK OFFSET: 4 VALUE: 0
ADDRESS: -1 TAG: -1 LINE NUMBER: 1 BLOCK OFFSET: 5 VALUE: 0
ADDRESS: -1 TAG: -1 LINE NUMBER: 1 BLOCK OFFSET: 6 VALUE: 0
ADDRESS: -1 TAG: -1 LINE NUMBER: 1 BLOCK OFFSET: 7 VALUE: 0
=====
Enter Next Command
ADDRESS EXISTS IN CACHE HOWEVER NOTHING HAS BEEN WRITTEN ON IT
```

This is what I mean. When you write in address 3 and then read from address 2. It will not be address Not found but Address Exists and nothing has been written on it. Also, The line 1 is perfectly empty thus it's addresses and tag is initialised to -1.

The user will get a lot of information when he puts his write command. He will know which line we are going in. He'll know if during replacement which line and what all pre written elements are being deleted.

Input Format

Enter Cache size:

Enter Number of cache Lines:

Enter Block Size:

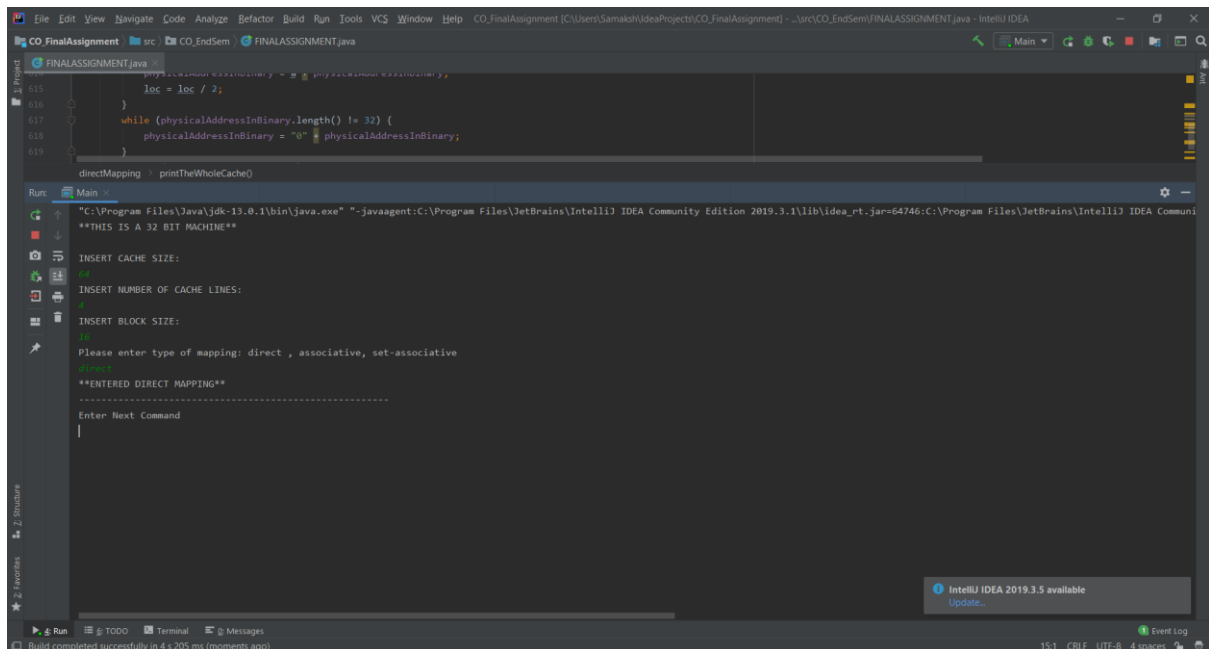
Then you need to enter the type of mapping you need:

“direct”

“associative”

“set-associative”

Pick anyone by entering any of them without the quotes.



If you select set-associative, you would be asked for one more input. That will be n . The ways to break in sets.

After you enter the standard inputs you need to give commands of read and write. The command pattern is your command followed by integers.

To Print the complete cache you type- "print-cache" (without the quotes).

If it is a read command then "read" followed by address (in decimal)

read, address

example: -

read 45

You want to know the contents at address 45.

If it is a write command then write followed by address followed by value

write, address, value

example: -

write 36 23

It means you requested the programme to write the value 23 at address 36.

EXPLANATION OF CODE

Language Used: JAVA

Classes Made: 5 [MAIN, directMapping, associativeMapping, set-associativeMapping, Node]

Node contains 3 attributes. [data, tag, address]

Global variable of all three classes Node[][]cacheArray and physicalAddressInBinary, cache size, line size, cache lines.

Main class has the psvm which contains all the input instructions. As per your command the switch case decides which function to call from which mapping. The mapping you select results in creation of an object of that mapping class and thus you can use it's functions to map by the rules of that format.

The three classes are directMapping, associativeMapping, setAssociativeMapping.

Node class is the main object which contains the information. This holds all the information about what you have to enter. Objects of this class are stored with attributes of address, value and tag. (It's not possible to assign tag to array.)

All three classes contain methods to convert the passed address into a binary string (physical address). Then one to again convert to decimal.

```
public String convertToBinary(int loc) {
    int a;
    while (loc > 0) {
        a = loc % 2;
        physicalAddressInBinary = a + physicalAddressInBinary;
        loc = loc / 2;
    }
    while (physicalAddressInBinary.length() != 32) {
        physicalAddressInBinary = "0" + physicalAddressInBinary;
    }
    return (physicalAddressInBinary);
}
public int convertToDecimal(String s) {
    return (Integer.parseInt(s, 2));
}
```

Then each class have their methods to extract tag, cache Line (in direct) and block offset.

Then There is a method to convert binary string to decimal in order to use these addresses. As per the rules of each mapping data is inserted to cache Array on write and data is fetched from the Array in case of read. If it is a cache miss, then miss shall be reported to the user.

Writing is different for each mapping thus discussed individually.

Note that in set associative cache array is a triple array[][][]. One for set, then inside each set cache lines and inside each line block size worth of elements.

DIRECT MAPPING

When you enter a write instruction, you give the address and the value.

This address is converted into 32-bit binary address. That is the global variable of my direct Mapping class. Then I break the address to generate TAG, CACHE LINE, BLOCK OFFSET. This division happens by three separate functions. One for each. I preserve the global variable and assign dummy variables its value in each function to not disturb this global variable. As it'll be used later.

Block offset = $\log(\text{base}2)\text{Block Size}$

This would give the number of bits required for Block offset. Then these many bits are extracted from the binary string from the last and this Block offset string is again converted into decimal integer. This will be the offset.

LINE INDEX= $\log(\text{base}2)\text{Cache Lines}$

This would give me bits required for cache lines. I will extract these many bits from my remaining String (block offset bits have been extracted). Then the string is converted to decimal.

TAG

The remaining bits are my TAG.

I have all three required parameters in decimal value. Now I need to put the data at the appropriate location.

CODE

So in this class I have a double array ([[]]). For cache lines and contents of cache Lines. My double Array is of type Node. Node has three attributes- Value or data, Tag, address.

To write I first need to find a Line which contains the same tag as the one we extracted from the address. Since we also extracted the Line Number in direct Mapping. Hence, we will go to that line and check the Tag it contains. If the tag matches, we write the current word in its block Offset position. If, however the tag doesn't match we delete the contents of the entire line in which this current data has to go. As this new word belongs to some other block. Then we put this number in that line and at its block offset position in the array.

For read instruction, just simply go to cache Line, Block offset and check if there is any content present at that location in the Array. If there is then it is a cache hit, else it is a cache Miss.

If there is a Cache Hit print

CACHE HIT

ADDRESS: 000000010....

Value: 45

For Cache Miss simply print

CACHE MISS

ADDRESS NOT FOUND: 0b0101....

If we start our command by read 10 then it will give a miss as we haven't written anything at the address.

Then we write 10 13. This means write 13 at address 10. 10 is 1010. Say the next instruction is read 10. Then We should get a cache Hit as we just wrote in cache. Then if we read from 74. 74 is 1001010 in binary. Note that 74 and 10 will give the same line number and block offset. However, will have the different tags. Thus read 74 shall give a cache miss and not a hit. Now if I write 74 43. Then the data of cache line containing address 10 has to be deleted as this new data of 43 from a separate block has to be written in the cache. Thus, now read 10 would give a cache miss and read 74 will give a cache hit and the value of 43 will be printed. Then if I overwrite 56 at address 74 and read 74. 56 will be printed as it is a cache Hit. Since one line can only contain certain specific blocks this has to be done. I have attached a screen shot of input and output of the programme. To end the programme just type "end".

```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help CO_FinalAssignment [C:\Users\Samash\IdeaProjects\CO_FinalAssignment] - \src\CO_EndSem\FINALASSIGNMENT.java - IntelliJ IDEA
Project: CO_FinalAssignment
src \ CO_EndSem \ FINALASSIGNMENT.java
Run Main
0 :TAG
DATA FROM NEW BLOCK HAS TO BE WRITTEN
PREPARING THE MAPPED LINE: 1
VALUE: 13 AT ADDRESS: 10 INSERTED
-----
Enter Next Command
read 10
**CACHE HIT**
ADDRESS: 10
VALUE: 13
-----
Enter Next Command
read 74
**CACHE MISS**
ADDRESS NOT FOUND: 74
-----
Enter Next Command
write 74 43
4 :TAG
DATA FROM NEW BLOCK HAS TO BE WRITTEN
PREPARING THE MAPPED LINE: 1
VALUE: 13 DELETED FROM ADDRESS: 10
VALUE: 43 AT ADDRESS: 74 INSERTED
-----
Enter Next Command
read 10
**CACHE MISS**
ADDRESS NOT FOUND: 10
-----
Enter Next Command
read 74
**CACHE HIT**
56
-----
Enter Next Command
end
All files are up-to-date (2 minutes ago)
IntelliJ IDEA 2019.3.5 available
Update...
521 CRLF UTF-8 4 spaces
```

FULLY ASSOCIATIVE MAPPING

Here when you give a write instruction you provide address and a value. Address is converted into a 32-bit binary string. Which is used as the global variable for this class. In fully associative mapping, there is no such thing called as a fixed line number. In this mapping any block can go in any line. Thus, we first need to check if the block pre exists in a line. If it does then it'll have an element whose tag will match the tag of the element we are trying to insert. If the tag matches, we insert this element in that very line at its block offset position. If, however the tag doesn't match then I look for an empty line in the cache. If I found an empty line then I insert the value in the offset position in that line. Here the block Number= Tag.

```
int lineNullCounter=0;
for (int i = 0; i < CL; i++) {
    for (int j = 0; j < B; j++){
        if (cacheArray[i][j] == null){
            lineNullCounter++;
        }
    }
}
//Empty Line will have BlockSize number of nulls.
if(lineNullCounter== B){
    cacheArray[i][blockOffset]=val;
    return;}
}
```

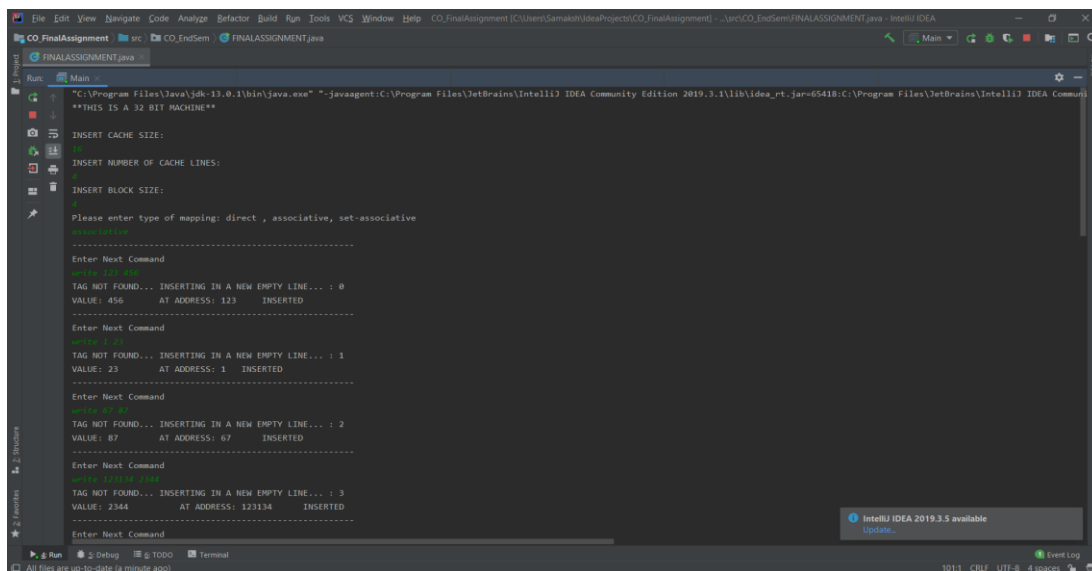
If I do not enter the if condition that means the cache is full in terms of all lines have elements of certain blocks and thus the loop will terminate and I will proceed to the next part in which I will randomly generate a line and remove all the contents of that line. Then I will insert this element in that line's offset position.

To read data. First, I will look for the tag in each line. If my tag matches that of the line then I would look in the block offset position if that line. Since there is not information about line number as line is taken randomly thus, we need to linearly look in each line for the tag. If there is data in the offset position if the matching tag line then I print CACHE HIT and the value. If it's a miss then the address doesn't exist in Cache and hence Cache Miss.

In the 32 bit address. $\log_2(\text{Block size})$ number of bits from the LSB side are for the block offset and rest of the bits make up the tag. Here Tag actually equals to the block number. Thus it is pretty easy to search and extract tag.

2 separate functions do this job. One extracts the tag other extracts blockoffset by string slicing.

THE FOLLOWING SCREEN SHOTS COVER ALL CASES



```
Run: Main --
"C:\Program Files\Java\jdk-13.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.3.1\lib\idea_rt.jar=65418:C:\Program Files\JetBrains\IntelliJ IDEA Commu
**THIS IS A 32 BIT MACHINE**

INSERT CACHE SIZE:
10

INSERT NUMBER OF CACHE LINES:
10

INSERT BLOCK SIZE:
1

Please enter type of mapping: direct , associative, set-associative
direct

Enter Next Command

TAG NOT FOUND... INSERTING IN A NEW EMPTY LINE... : 0
VALUE: 456 AT ADDRESS: 123 INSERTED

Enter Next Command

TAG NOT FOUND... INSERTING IN A NEW EMPTY LINE... : 1
VALUE: 22 AT ADDRESS: 1 INSERTED

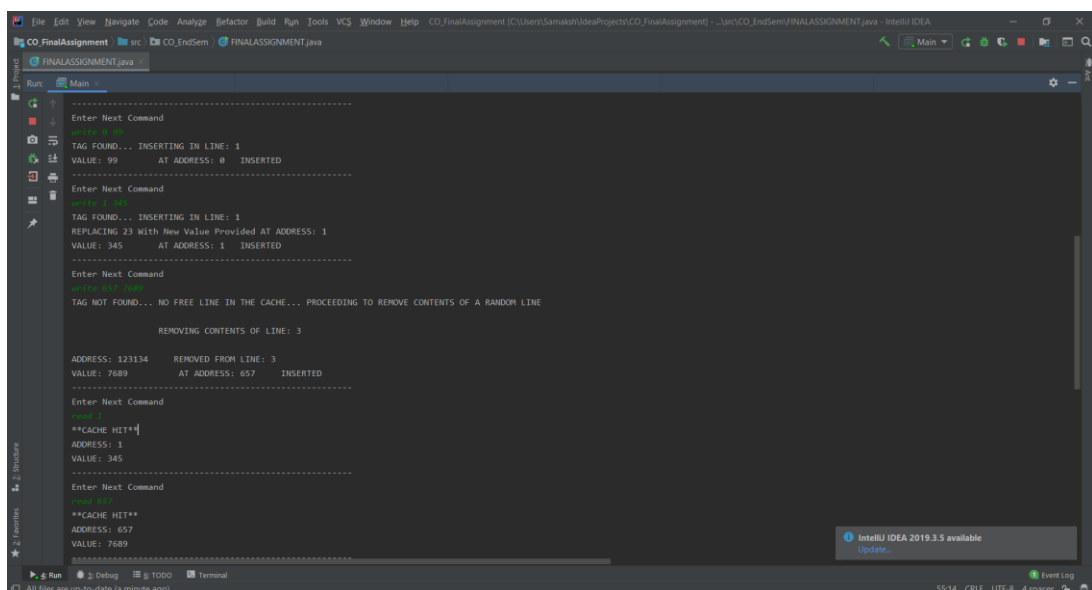
Enter Next Command

TAG NOT FOUND... INSERTING IN A NEW EMPTY LINE... : 2
VALUE: 87 AT ADDRESS: 67 INSERTED

Enter Next Command

TAG NOT FOUND... INSERTING IN A NEW EMPTY LINE... : 3
VALUE: 2344 AT ADDRESS: 123134 INSERTED

Enter Next Command
```



```
Run: Main --
Enter Next Command

TAG FOUND... INSERTING IN LINE: 1
VALUE: 99 AT ADDRESS: 0 INSERTED

Enter Next Command

TAG FOUND... INSERTING IN LINE: 1
REPLACING 23 With New Value Provided AT ADDRESS: 1
VALUE: 345 AT ADDRESS: 1 INSERTED

Enter Next Command

TAG NOT FOUND... NO FREE LINE IN THE CACHE... PROCEEDING TO REMOVE CONTENTS OF A RANDOM LINE

REMOVING CONTENTS OF LINE: 3

ADDRESS: 123134 REMOVED FROM LINE: 3
VALUE: 7689 AT ADDRESS: 657 INSERTED

Enter Next Command

TAG FOUND... INSERTING IN LINE: 1
ADDRESS: 1
VALUE: 345

Enter Next Command

TAG NOT FOUND... NO FREE LINE IN THE CACHE... PROCEEDING TO REMOVE CONTENTS OF A RANDOM LINE

REMOVING CONTENTS OF LINE: 3

ADDRESS: 123134 REMOVED FROM LINE: 3
VALUE: 7689 AT ADDRESS: 657 INSERTED

Enter Next Command
```

```

Enter Next Command
>add 456
ADDRESS EXISTS IN CACHE HOWEVER NOTHING HAS BEEN WRITTEN ON IT
ADDRESS IS IN LINE: 3
-----
Enter Next Command
>add 120
=====
ADDRESS : 120 TAG: 30 LINE NUMBER: 0 BLOCK OFFSET: 0 VALUE: 0 ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
ADDRESS : 121 TAG: 30 LINE NUMBER: 0 BLOCK OFFSET: 1 VALUE: 0 ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
ADDRESS : 122 TAG: 30 LINE NUMBER: 0 BLOCK OFFSET: 2 VALUE: 0 ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
ADDRESS : 123 TAG: 30 LINE NUMBER: 0 BLOCK OFFSET: 3 VALUE: 456 ADDRESS HAS BEEN WRITTEN TO
=====
ADDRESS : 0 TAG: 0 LINE NUMBER: 1 BLOCK OFFSET: 0 VALUE: 99 ADDRESS HAS BEEN WRITTEN TO
ADDRESS : 1 TAG: 0 LINE NUMBER: 1 BLOCK OFFSET: 1 VALUE: 345 ADDRESS HAS BEEN WRITTEN TO
ADDRESS : 2 TAG: 0 LINE NUMBER: 1 BLOCK OFFSET: 2 VALUE: 0 ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
ADDRESS : 3 TAG: 0 LINE NUMBER: 1 BLOCK OFFSET: 3 VALUE: 0 ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
=====
ADDRESS : 64 TAG: 16 LINE NUMBER: 2 BLOCK OFFSET: 0 VALUE: 0 ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
ADDRESS : 65 TAG: 16 LINE NUMBER: 2 BLOCK OFFSET: 1 VALUE: 0 ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
ADDRESS : 66 TAG: 16 LINE NUMBER: 2 BLOCK OFFSET: 2 VALUE: 0 ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
ADDRESS : 67 TAG: 16 LINE NUMBER: 2 BLOCK OFFSET: 3 VALUE: 87 ADDRESS HAS BEEN WRITTEN TO
=====
ADDRESS : 656 TAG: 164 LINE NUMBER: 3 BLOCK OFFSET: 0 VALUE: 0 ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
ADDRESS : 657 TAG: 164 LINE NUMBER: 3 BLOCK OFFSET: 1 VALUE: 7689 ADDRESS HAS BEEN WRITTEN TO
ADDRESS : 658 TAG: 164 LINE NUMBER: 3 BLOCK OFFSET: 2 VALUE: 0 ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
ADDRESS : 659 TAG: 164 LINE NUMBER: 3 BLOCK OFFSET: 3 VALUE: 0 ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
=====
Enter Next Command

```

SET ASSOCIATIVE

After converting the 32-bit address in binary. You extract offset location, set location and tag. Number of sets= CL/n .

Here $\log(\text{base } 2)(\text{Block size})$ bits are for block offset.

$\log(\text{base } 2)(CL/n)$ bits are for set

Remaining are the tag.

Array would be like this [SET][LINE NUMBER][BLOCK OFFSET]

So, you extract the tag bits and decide which set your element address should go to. Then you look for the tag in each line of that sequentially. As write happens randomly in any line in a predefined set. If you find the tag then you insert your current element in the offset position in that line where you found the tag. If you don't find the tag then you look for an empty line. You insert nodes with value 0 and address which they should have in that line and then insert this element. Now all the elements in this line have the same tag but one 1 of them has been written to. Rest of the addresses will exist but haven't been written upon.

If you don't find any empty line and the tag doesn't match in any line. Then you randomly select a line and delete the contents of that line, and insert the new addresses and values.

```

private int generateTag() {
String x= physicalAddressInBinary;
StringBuilder y= new StringBuilder();
int bitsToLeave= generateSetBits()+ blockOffsetBits();

int bitsRequired= 32- bitsToLeave;
int i=0;
while(i<bitsRequired){
    y.append(x.charAt(i));
    i++;}

return convertToDecimal(y.toString());
}

```

This code gets the tag. As you first remove the last bits that are supposed to go for set and offset. The remaining bits from the start are your tag.

```

private int generateBlockOffsetAddress() {
    int bitsrequired = blockOffsetBits();
    String x = physicalAddressInBinary;
    StringBuilder BlockAddress = new StringBuilder();
    int i = 32 - bitsrequired;
    while (bitsrequired > 0) {
        BlockAddress.append(x.charAt(i));
        bitsrequired--;
        i++;
    }

    return convertToDecimal(BlockAddress.toString());
}

```

This code slices the binary address and takes out the last 'bitsRequired' number of characters to generate the offset.

```

private int generateSetAddress(){
    int bitsRequired= generateSetBits();
    String x = physicalAddressInBinary;
    int removeBits= blockOffsetBits()+generateSetBits();
    StringBuilder y= new StringBuilder();
    int i= 32-removeBits;

    while(bitsRequired>0){
        y.append(x.charAt(i));
        i++;
        bitsRequired--;
    }

    return convertToDecimal(y.toString());
}

```

This will be the set number. Number of sets = $CL/n...$ thus there would be $\log(\text{base } 2)$ Of those sets bits required to represent. Hence we extract those bits after removing offset number of bits from the end.

Read instruction outputs the value and address and reports if it is a hit or a miss or the address is present but there isn't any value in it.

```

C:\Program Files\Java\jdk-13.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.3.1\lib\idea_rt.jar=65476:C:\Program Files\JetBrains\IntelliJ IDEA Commu
**THIS IS A 32 BIT MACHINE**

INSERT CACHE SIZE:
INSERT NUMBER OF CACHE LINES:
INSERT BLOCK SIZE:

Please enter type of mapping: direct , associative, set-associative
enter n:
**ENTERED 2-WAY SET ASSOCIATIVE MAPPING **

Enter Next Command
value 33
TAG NOT FOUND... INSERTING IN A NEW EMPTY LINE... : 0   OF SET: 0
VALUE: 33   AT ADDRESS: 0   INSERTED

Enter Next Command
value 324
ADDRESS EXISTS IN CACHE HOWEVER NOTHING HAS BEEN WRITTEN ON IT
ADDRESS IS IN LINE: 0   OF SET: 0

Enter Next Command
value 324
TAG NOT FOUND... INSERTING IN A NEW EMPTY LINE... : 0   OF SET: 1
VALUE: 324   AT ADDRESS: 678   INSERTED

Enter Next Command
value 324
TAG NOT FOUND... INSERTING IN A NEW EMPTY LINE... : 1   OF SET: 0
  
```

```

TAG NOT FOUND... NO FREE LINE IN THE SET... PROCEEDING TO DELETE A RANDOM LINE
DELETE FROM LINE: 0   OF SET: 0
ADDRESS: 0   DELETED FROM LINE: 0   OF SET: 0
ADDRESS: 1   DELETED FROM LINE: 0   OF SET: 0
VALUE: 213   AT ADDRESS: 768   INSERTED

Enter Next Command

=====
ADDRESS : 768   TAG: 96   SET: 0   LINE NUMBER: 0   BLOCK OFFSET: 0   VALUE: 213   ADDRESS HAS BEEN WRITTEN TO
ADDRESS : 769   TAG: 96   SET: 0   LINE NUMBER: 0   BLOCK OFFSET: 1   VALUE: 0   ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
ADDRESS : 770   TAG: 96   SET: 0   LINE NUMBER: 0   BLOCK OFFSET: 2   VALUE: 0   ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
ADDRESS : 771   TAG: 96   SET: 0   LINE NUMBER: 0   BLOCK OFFSET: 3   VALUE: 0   ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
=====
ADDRESS : 121312 TAG: 15164 SET: 0   LINE NUMBER: 1   BLOCK OFFSET: 0   VALUE: 0   ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
ADDRESS : 121313 TAG: 15164 SET: 0   LINE NUMBER: 1   BLOCK OFFSET: 1   VALUE: 0   ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
ADDRESS : 121314 TAG: 15164 SET: 0   LINE NUMBER: 1   BLOCK OFFSET: 2   VALUE: 45   ADDRESS HAS BEEN WRITTEN TO
ADDRESS : 121315 TAG: 15164 SET: 0   LINE NUMBER: 1   BLOCK OFFSET: 3   VALUE: 0   ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
=====
ADDRESS : 676   TAG: 84   SET: 1   LINE NUMBER: 0   BLOCK OFFSET: 0   VALUE: 0   ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
ADDRESS : 677   TAG: 84   SET: 1   LINE NUMBER: 0   BLOCK OFFSET: 1   VALUE: 0   ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
ADDRESS : 678   TAG: 84   SET: 1   LINE NUMBER: 0   BLOCK OFFSET: 2   VALUE: 324   ADDRESS HAS BEEN WRITTEN TO
ADDRESS : 679   TAG: 84   SET: 1   LINE NUMBER: 0   BLOCK OFFSET: 3   VALUE: 0   ADDRESS EXISTS IN CACHE, BUT NOTHING HAS BEEN WRITTEN ON IT
=====
ADDRESS : 0   TAG: -1   SET: 1   LINE NUMBER: 1   BLOCK OFFSET: 0   VALUE: 0
ADDRESS : 0   TAG: -1   SET: 1   LINE NUMBER: 1   BLOCK OFFSET: 1   VALUE: 0
ADDRESS : 0   TAG: -1   SET: 1   LINE NUMBER: 1   BLOCK OFFSET: 2   VALUE: 0
ADDRESS : 0   TAG: -1   SET: 1   LINE NUMBER: 1   BLOCK OFFSET: 3   VALUE: 0
=====
  
```

Print cache has been explained before. But see the line 1 of set 1. It doesn't have any description because it hasn't been written to ever. None of the

addresses got written on line 1 of set 1 ever. The address -2 there doesn't mean that address -1 exists in 4 places. I used -1 to initialise it. That just depicts nothing is in that line of that set. It is empty.

FAQ

Q- HOW TO BREAK OUT FROM THE PROGRAMME?

ANS- TYPE "end" (without quotes)

Q- What If I give a wrong input?

ANS- As long as it is not an input mismatch, your wrong instruction will be ignored. I mean if you type- "writy 12 34". This spelling error will not result in programme crash but will be ignored. However, If you input this command during the time when you had to enter cache size then that'll be an input mismatch of String and int, thus programme will throw an exception.

Q- What if I input the value as 0.

ANS- You cannot do that. You cannot pass 0 in the value input, however 0 can be passed as an address.

Q- Can I pass a negative value?

ANS- Yes you can pass a negative value, but not any negative address.

Q- What has the Tag been pre assigned as?

ANS- Tag has been pre assigned as -1 in Node class itself.

Q- What has the address been pre assigned as?

ANS- Address has been pre assigned as -1 in Node class itself.

Q- When do you initialise the array?

When you select the mode, I have made the next line to initialise this array under the mapping. Thus, the array is filled with Nodes, which have 0 as there value, -1 as there tag.

EXCEPTIONS

YOU CANNOT INPUT ANY ADDRESS THAT IS MORE THAN $(2^{32}-1)$. This number cannot be read by JVM as it is not precise in integer. Also, it will not make sense as my machine is 32-bit machine.

YOU CANNOT enter the value as 0.

YOU CAN enter the address as 0. It is allowed to write on address 0.

THE ADDRESS -1 depicts the line was never written to.

Sets in n-way arrangement has been broken by three star lines, A line has been broken by 1 star line.

Do NOT enter a value that is more than the precision for int. The value in Node class is of type integer not long.

DO NOT give float values as input value or as address.

I KNOW in reality when there is a cache miss, you search in bigger memory and then you extract data from there and write in the smaller memory. However, we were not supposed to maintain any main memory thus that is ruled out. Secondly, if I do that then all the read instructions eventually become write instructions with a value as I like or as user gave before this block was evicted. However, there is no way to know which value was evicted before as we are NOT SUPPOSED TO MAINTAIN A MAIN MEMORY. Thus reading should either given a cache miss or a hit or the most I could do is report that although the address exists in cache but nothing has been written on it by the user.