**Documentation**

Bhavesh Sood

Inputs Required by the user:

* CL - No. of Cache Lines (Should be in power of 2)
* B ­- Block size in bytes. (Should be in power of 2)

Then the Main menu appears, where the user is supposed to enter the Cache Type he wishes to use (from the options as integer values).

Now for each type of Cache there are some general commands to use the Cache. These are –

* **read address**: This tells the Cache to read this address . Here address is of type integer for easy input (it is converted to a 32 bit binary number later).Eg. read 1024 – This tells the Cache to read the value at address 1024 .
* **write address value**: This tells the Cache to write this particular value at this particular address. Here both address and value are of type integer.Eg. write 64 125 – This means that we are telling the cache to write value 125 at address location 64 in memory.
* **print** : This command is used to print the current status of the Cache.
* **exit**: Used to go back to the main menu.

(Even only writing the first letter for the command works)

For a N-way Set Asscoiative Cache – One needs to provide input for N also (Should be in power of 2).

These are all the inputs that one needs to provide.

Detailed Explanation of Code:

Lets go through it in the sequential manner of the Code.

![A screenshot of a cell phone

Description automatically generated]()

This is the starting screen of the program.

Till now our No. of bits for offset is already set , as the power of 2 which is equal to the block size.

Note if the cache lines or block size was not in powers of 2 it would ask you to provide them again.

We also know the Cache Size now.

In the end we reach the main menu.

Here we have 3 types of Caches to choose from.

Lets first view the internal structure of the Caches and how is the LRU replacement scheme working for them.

For all the three Caches , I have used a Hashmap called directmap,

where the key is – a string, and the value – is and array of integers.

So the concept used here is that the key is the tag and the array of integers is the block.

So each tag is a key and since all tags are unique we can use a hashmap.

Now say we want to write value 33 at byte no. 129 in our computer.

Then we will find the tag of 129 after converting it to 32 bits and then at the byte no. 2 of this block as 129= 128 +1 = 27+ 1 , we write the 33.

This 1 is the index of the array which will hold the value 33 that is associated with the tag of 129.

Showing this in the Associative Cache.

![A screenshot of a cell phone

Description automatically generated]()

Now since the tag was same , though offset different, we get a Cache hit , since it’s in the same block.

Now let’s look at the LRU replacement scheme.

To implement the LRU scheme I maintained a list<string> ( basically a queue ) of type string.Now whenever I get a query on any tag that is in the Cache , meaning a Cache hit occurred , I find this tag in the list and erase it , and then push it in the back of the queue.

This way all the recently used blocks stay towards the end of the list.

Now any time I have to evict a block, I evict it from the front of the queue.

This is because now I know that it is **least recently used** block.

This scheme is applicable only is Associative Cache and Set Asscociative (where for each set this queue is maintained).

For Direct Mapped Cache - I have made a Tag array. Which stores all the tags of every block in the Cache. Now I computed the index of an address by using the cache size, say cache has 128 cache lines , then every block in the Cache can be assigned an index based on the 7 bits (27=128) after the offset.

![A screenshot of a social media post

Description automatically generated]()

Now if you see the blue marked indexes. You will understand that even though our Cache had empty space , still the second write , write 667 66 , caused a

Cache miss , this is because the index of it was same as that of the address 165.

This is the flaw of Direct Mapped Cache. Here every tag is stored only in the index no. of the address. Since our cache had 4 cache line , thus 2 bit index.

LRU in N – Way Set Associative.

![A screenshot of text

Description automatically generated]()



In this image you can see that both 72 and 200 have the same index for a Cache of 4 Cache lines. Now the cache at first assigns both of them into two slots available to this set of index 1.

Now if you see 72 is again read from the cache , therefore making address 200 as the least recently used cache.

Therefore when address 456 is inserted , which also had index 1 , and now since there is no space left in the set , it evicts the least recently used block.

And one can see here it is tag of address 200.

Assumptions:

* If you read an address which isn’t in the cache it will show Cache miss and then after this read it would be inserted in the cache.However if you read it again then its value would be found -1.(Since main memory is not there we don’t know the actual value).
* The empty places on print command, just mean that no block in inserted here.
* If there is a Cache hit , and you read an address which wasn’t written to before the default value is 0.