Documentation

Direct Mapped Cache

In a direct mapped cache, there is only one particular cache line for every block. The line number of the block can be calculated by taking the modulus of the block number by the number of cache lines.

Assumptions:

- The total number of words that a system can incorporate is user dependent but should be in a power of 2.
- The block size should specify how many words a single block contains.
- The total number of operations either read or write must be entered for the program to work.
- The program will terminate once the number of operations is completed.
- The cache line size is equal to the block size.
- In case of a read miss the program will show “Cache miss!” and show the structure. However, in a read hit the program will display the value read and also print the cache structure.
- Only integer inputs are allowed.
- Word size is assumed to be 1 byte.
- All the inputs are in integer be it address or other inputs.
- Initially, the cache is empty and for first read operation it will show “Cache miss!”.

Working:

- The program asks for all the inputs: number of words, size of cache, cache lines, block size, number of queries or operations and then the type of operation.
- If the user wants to read an address 0 must be entered, if a value is to be written then 1 must be entered.
- In case of a read operation the program will ask for the address.
- In case of a write operation the program will ask for the address and the value to be written.
- For read operation, if the block (in which the required address is present) is present in the cache then “Cache hit!” is displayed along with the value to be read and the entire cache structure. But if the required block is not present the “Cache miss!” is displayed along with entire cache structure.
- For write operation, there is no particular cache miss or hit. It does not matter if the address to be written at is present in the one of the lines of cache or not. Whatever, be the address the block containing the address will be brought to the cache and written over. The entire cache structure is printed.

Implementation:

- As soon as the number of cache lines, block size and number of words are received by the user the cache and the memory get built.
- For cache, I have created an array whose size is equal to the number of cache lines. Initially all the elements of the cache are set to be -1.
- For main memory, I have created a 2D array having number of rows as the number of blocks in the system and the number of columns as the block size.
- The cache array (named cache) contains the block number that is inserted in the particular cache line. For example, in a cache of four lines having one block each, let us say L_0, L_1, L_2, and L_3 contain B_0, B_1, B_2, and B_3, then the cache array would look like

0	1	2	3

- The block array, on the other hand (named blocks) will contain in each row the words in that block. For example, there are four blocks each having four words then all the entries of the array will be 0 as nothing is written in the memory when the program starts, and hence will look like:

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

- So, if the address 3 is written with 3 then blocks array will be modified as:

0	0	0	3
0	0	0	0
0	0	0	0
0	0	0	0

- In a read operation, I pass the parameters the address, the block size, number of cache lines, the cache array, and the blocks array. The block number in which the required address is present is calculated by (address) / (block size). Now (block number) % (number of cache lines) will tell the line in cache in which the block has to be inserted (if not yet inserted) or the block may be present according to the direct mapping. If that block number is not equal to the content contained in the cache array at the index of the line number then it is a cache miss but the program brings the required block into its specified location in the cache so that it can be used later. However, if the block number is equal to the content at the specific location then the required value is printed along with a message “Cache hit!” and the entire cache structure.
- In a write operation, I pass the parameters same as the read operation but with the value that is to be written at the address. Its working is also similar to the read operation, the difference lies after the it is confirmed if the block is contained in the cache or not. If the block is contained in the cache then the value is written in the required place in the blocks array following the write-through policy. However, if the block is not present in the cache then it is brought from the main memory in the cache and then written at.

- For printing the structure of the cache, I have implemented a function called printCache demanding parameters the cache and the blocks array. It iterates through the cache array which contains the block number. If the block number comes out to be -1 then it means that particular line of the cache is empty and hence 0 is printed. But if the block number is not -1 then the contents of the block are printed.

Fully Associative Mapped Cache

In this type of mapping any block can go in any of the lines of the cache. There is no particular place for any block. Hence, the number of cache misses is reduced.

Assumptions:

- The total number of words that a system can incorporate is user dependent but should be in a power of 2.
- The block size should specify how many words a single block contains.
- The total number of operations either read or write must be entered for the program to work.
- The program will terminate once the number of operations is completed.
- The cache line size is equal to the block size.
- In case of a read miss the program will show “Cache miss!” and show the structure. However, in a read hit the program will display the value read and also print the cache structure.
- Only integer inputs are allowed.
- Word size is assumed to be 1 byte.
- All the inputs are in integer be it address or other inputs.
- Initially, the cache is empty and for first read operation it will show “Cache miss!”.
- Replacement is done according to the first in first out policy. A kind of LRU.

Working:

- The program asks for all the inputs: number of words, size of cache, cache lines, block size, number of queries or operations and then the type of operation.
- If the user wants to read an address 0 must be entered, if a value is to be written then 1 must be entered.
- In case of a read operation the program will ask for the address.
- In case of a write operation the program will ask for the address and the value to be written.
- For read operation, if the block (in which the required address is present) is present in the cache then “Cache hit!” is displayed along with the value to be read and the entire cache structure. But if the required block is not present the “Cache miss!” is displayed along with entire cache structure.

- For write operation, there is no particular cache miss or hit. It does not matter if the address to be written at is present in the one of the lines of cache or not. Whatever, be the address the block containing the address will be brought to the cache and written over. The entire cache structure is printed.

Implementation:

- A public queue is maintained to keep track of which block is being accessed and the replacement can be done according to the first in first out policy.
- As soon as the inputs are received a cache array and a 2D blocks array are created.
- The implementation of the cache array and the blocks array is the same as the direct mapped cache. The only difference lies in the placing of the block number in the cache array.
- In a read operation, the required parameters are passed and the cache array is iterated over to check if the required block number is present in the cache array. If it is present then it is a cache hit and the required value is printed along with the entire cache structure and that particular block number is added to the queue.
- However, if the block number is not found then it is a cache miss, the message is printed along with the entire cache structure. But we still need to bring that block in the cache and add it to the queue. Again, the cache is iterated over to check if any line is empty. If there is an empty line then that block is inserted in that line. But if there is no empty line then the last used block is replaced. And if the cache is empty then the zeroth line is occupied by the block.
- The write operation also works in the similar manner as the read operation works. The difference lies only in the message being printed. No message like “Cache hit!” or “Cache miss!”. The value will be written no matter if it is present in the cache or not. If it is present then the value is written normally but if it is not then it is brought from the main memory and then written over. The replacement technique is the same as the read operation.
- For printing the cache, the function used is the same as the direct mapped cache.

N-Way Set Associative Mapped Cache

This type of mapping is a combination of the associative mapping and direct mapping. In this mapping the cache is divided into sets and the blocks have their definite set in the cache (direct mapping feature). However, a set contains two or more cache lines and a block which has to be in some particular set can be in any of the lines in the set (fully associative feature).

Assumptions:

- The total number of words that a system can incorporate is user dependent but should be in a power of 2.

- The block size should specify how many words a single block contains.
- The total number of operations either read or write must be entered for the program to work.
- The program will terminate once the number of operations is completed.
- The cache line size is equal to the block size.
- In case of a read miss the program will show “Cache miss!” and show the structure. However, in a read hit the program will display the value read and also print the cache structure.
- Only integer inputs are allowed.
- Word size is assumed to be 1 byte.
- All the inputs are in integer be it address or other inputs.
- Initially, the cache is empty and for first read operation it will show “Cache miss!”.
- Replacement is done according to the first in first out policy. A kind of LRU.
- N is a power of 2.

Working:

- The program asks for all the inputs: number of words, size of cache, cache lines, block size, number of queries or operations, the number of set and then the type of operation.
- If the user wants to read an address 0 must be entered, if a value is to be written then 1 must be entered.
- In case of a read operation the program will ask for the address.
- In case of a write operation the program will ask for the address and the value to be written.
- For read operation, if the block (in which the required address is present) is present in the cache then “Cache hit!” is displayed along with the value to be read and the entire cache structure. But if the required block is not present the “Cache miss!” is displayed along with entire cache structure.
- For write operation, there is no particular cache miss or hit. It does not matter if the address to be written at is present in the one of the lines of cache or not. Whatever, be the address the block containing the address will be brought to the cache and written over. The entire cache structure is printed.

Implementation:

- As soon as all the inputs are received from the user the program executes and the number of sets is calculated along with the number of cache lines in the set.
- For the implementation of the cache a 2D array is maintained with number of rows equal to the number of sets and number of columns equal to the cache lines in each set.

- The implementation of the main memory is the same as the other two mappings discussed above.
- Initially, all the indices of the set array contain -1. For example, in a system of two sets having four cache lines the set array would like:

-1	-1	-1	-1
-1	-1	-1	-1

- For read operation, all the required parameters are passed into the function and the block number is calculated. Now the set number is calculated which may contain the desired block. For example, in the above example if a block is present in second set then it could be anywhere in the second row. For the representation it is assumed to be the third index: BN = block number.

-1	-1	-1	-1
-1	-1	BN	-1

  The required set is iterated upon to check if the block number is present. If it is present then there is a cache hit and the required value is printed with the message and the structure of cache. But if it is not present then it is a cache miss and the required block is brought to the cache in the required set. However, the program checks if there is an empty cache line in the set. If any cache line is empty then block is moved to that cache line else replacement is done according to the first in first out policy which is achieved through the public queue maintained and adding to it the block number being worked upon.
- The implementation of the write operation is the same as the read operation, the only difference being there is no message and value is simultaneously written to the main memory.
- The print function takes in parameters the cache, the main memory and the number of sets. For each set it iterates upon the cache lines and prints the contents.

How I dealt with tag and offset?

In a cache line only one block can come as the block size and the size of the cache line is same. So, a **tag** is unique identifier for a group of data. Because different regions of memory may be mapped into a block, the tag is used to differentiate between them. For this I directly mention in the cache array the number of the block so the I need not calculate tag specifically.

Offset tells which word in a particular block is to be worked upon. For this I iterate over the contents of the block and check if the word number equals the address. Thus, offset need not be calculated as it is incorporated in the iteration.