DOCUMENTATION

CACHE MAPPING AND 2-LEVEL CACHE SEARCHING

(Note: Bonus task included with the end-sem assignment)

Project By:
Shruti Jha (2019274)

The problem statement is: “Write a program that allows loading into a cache of size S/2 and searching the cache using the 3 different mapping techniques”.
Bonus task: “Build a 2 level cache. The size of the level 1 cache is S/2 and the size of level 2 cache is S.”
This document contains information about how the code developed in java implements this problem statement and the bonus task.

1) About Cache Mapping :

This program in java implements cache mapping at level-1 (L1) of cache by using the following three different cache mapping techniques:

1. Direct mapping
2. Fully Associative Mapping
3. K-way Set Associative Mapping

Below is the explanation of how these three kinds of mappings are done from the main memory to cache L1:

(NOTE: Assume that the size of each block in memory is B words. Hence the size of each cache line is also B words. Take the number of cache lines as CL and the total number of bits required to represent a memory address as N. Also, all logs mentioned are base 2).

- *Direct Mapping:*

  The N bits used to represent MAR can be divided as shown for direct mapping:

N-(log(CL)+log(B)) bits: tag	log(CL) bits: cache line number	log(B) bits: block offset

  In this mapping, each block of the main memory can be stored only in a particular cache line. This cache line number is represented by the log(CL) bits shown above.

  Hence in direct mapping, if the CPU requests for MAR, we search for the memory address only in that cache line in which it can be possibly stored. If it is found then it is a cache HIT, otherwise a cache MISS.

- *Fully Associative Mapping:*

  The N bits used to represent MAR can be divided as shown for fully associative mapping:

N-log(B) bits: block number	log(B) bits: block offset

  In this mapping, any block of the main memory can be stored in any cache line. Hence in fully associative mapping, if the CPU requests for MAR, we need to check all the cache lines to search for the particular memory address. If it is found, then it is a cache HIT, otherwise a cache MISS.

- *k-way Set Associative Mapping:*

  In k-way set associative mapping, the cache lines are divided into sets. The number of sets ST= number of cache lines/k.

  The N bits used to represent MAR can be divided as shown for k-way set associative mapping:

N-(log(ST)+log(B)) bits: tag	log(ST) bits: set number	log(B) bits: block offset

  In this mapping, each block of the main memory can be stored only in a particular set in the cache. This set number is given by the log(ST) bits shown above. Within this set, the block can be stored in any cache line.

  Hence in k-way set associative mapping, if the CPU requests for MAR, we search for the memory address only in the set in which it can be possibly stored. Within a set, we need to check all the cache lines. If it is found, it is a cache HIT, otherwise a cache MISS.

2) About 2-level cache searching:

This program in java implements the 2-level cache searching using the following method :

- If the CPU requests for a memory address MAR to complete a task, it is first searched in cache level-1 (called L1).
- If it is found in L1, then it is called a cache L1 HIT. If it is not found, then it is called a cache L1 MISS.
- In case of L1 MISS, the address is searched in cache level-2 (called L2).

- If the address is found in L2, it is called a cache L2 HIT, otherwise it is called a cache L2 MISS.
- In case of cache L2 miss, the address is finally searched in the main memory.

The level-2 of cache (L2) ,is usually of larger size than level-1 of cache (L1), storing more number of blocks of main memory than cache L1. Although, most of the time the memory address is found in L1 only, it is possible that the need to search for a block in L2 arises.

3)Machine requirements:

- The main memory is considered to be 16-bit. Hence, there can be 2^16 (i.e. 65536) words with addresses in the range 0 to 65535.
- Should have at least 2^16 (i.e. 65536) words of memory.

4)Workflow:

There are 2 levels of cache implemented in this program. For level-1 cache (L1), there are three different cache structures designed for each kind of mapping. Only in the case of a cache MISS in L1, the control goes to level-2 cache (L2). All the three mapping types in L1 share the same structure of L2.

- First an array of size 2^16 is made to represent the main memory. The integer data allocated to each memory address is the same as its index number in the array.
- Then, three HashMaps are declared to store the key-value pairs for each kind of mapping in cache level-1. In case of direct mapping, the key is the cache line number and the value is the tag string of the block it is storing as explained in

section 1. For fully associative mapping, the key is the block number loaded into each cache line and the value is the cache line number. In case of k-way set associative mapping, the key is the cache line number and the value is the tag string of the block loaded into the line as explained above.

- Three separate arrays are also declared to store the data from the main memory for each type of cache mapping.
- The HashMaps in each case are randomly allocated some block from the main memory depending on the kind of mapping. In each type of mapping, only those blocks are pre-loaded into each cache line where they are allowed to be stored. For example, in direct mapping, each block can only be stored in a particular cache line.
- A separate HashMap is declared for level-2 cache. The size of this level is twice that of level-1. In this map, the key is the cache line number while the value is a string representing the block number stored in that line. Again an array of twice the size as before is declared to store the data from the main memory.
- After the user enters the input values, three functions are called to search for the address MAR in the cache L1, according to the three mapping techniques mentioned above.
- *SEARCH AT LEVEL-1 (L1):*

  In each kind of mapping, a function is called which searches for MAR in L1.
  The block number of each MAR is identified by ignoring the offset bits from the end.

  In case of direct mapping, the cache line number is identified from the 16-bit representation of MAR (by taking the bits as explained in section 1). The tag stored with that line in the HashMap is checked, and if both of them match, then it is a cache HIT in L1, otherwise it is a cache MISS.

  In case of a HIT, the line where the block is stored is displayed along with the data stored in MAR. In case of a cache MISS, the tag stored at the calculated line

number is deleted and the tag bit of MAR is stored in its place. This is because in this case each block can be stored only in a particular line. The data of the block to which MAR belongs to is also copied into the cache array. After this, the function to search for MAR in L2 is called.

In case of fully associative mapping, since any block can be stored in any line, the whole HashMap is searched to check if the concerned block is stored in it or not. If the block is found, then it is a cache HIT, and the line number stored with it is displayed. In case it is not found, it is a cache MISS. In case of a miss, the block to which MAR belongs to randomly replaces any other block at some cache line since it can be stored in any line. The data of the block to which MAR belongs to is also copied into the cache array of this mapping. After this, the function to search for MAR in L2 is called.

In the case of k-way set associative mapping, the set number in which MAR can belong to is determined from its binary representation. The tag of each cache line which belongs to this set is searched in the HashMap, and if any tag matches the tag of MAR, then it is a cache HIT. The line number where it was found is displayed.

If no tag matches, then it is cache MISS. In case of a miss, the block to which MAR belongs to replaces the tag of the block stored in the first cache line of the set in which it can be stored. This is because in this mapping, each block can belong to only one set, but within a set it can belong to any cache line. The data of the block to which MAR belongs to is also copied into the cache array of this mapping. After this, the function to search for MAR in L2 is called.

- *SEARCH AT LEVEL-2 (L2):*

  When there is a cache miss at level-1 in any of the three cases, the function which implements cache searching in L2 is called. In this function, the HashMap for L2 storing the cache line number- block number pairs of each block stored in each line of cache L2 is searched. If the block to which MAR belongs to is stored

in this map, then its a cache HIT in L2, otherwise it is a cache MISS. In case of HIT, the line number where the block is loaded is displayed. The data stored at MAR is also displayed.

In case of a MISS, the block to which MAR belongs to randomly replaces some other block in the cache L2. The block numbers and the line where replacement is done is also displayed. The data of the block to which MAR belongs to is also copied into the cache L2 array.

- *NOTE:*

  1)In each case of a cache MISS at any of the 2 levels, the block to which MAR belonged to replaced some other block in the cache according to the mapping considered. The reason for the importance of replacement in cache is the concept of ***'temporal locality'***. It is a concept which states that if a resource is accessed at some point of time, then most likely it will be accessed again in a short period of time.Hence if there is a cache MISS, then placement of this block in the cache guarantees that if a memory address close to MAR is searched in the cache (most programs access memory blocks close to each other, for example array implementation), it will increase the chances of a cache HIT.

  2)Before declaring a cache HIT or MISS, the cache contents of both L1 and L2 are displayed so that the user can confirm the result.

  3)*Data loss is not possible on replacement of blocks in the cache since write operations are not handled in this assignment.*

5)Input Format:

Since the cache size is variable and can be decided by the user, the user needs to provide the following 5 inputs to the program in the order they are mentioned:

- The size of block : B.
- The number of cache lines: CL

- The value of k for k-way set associative mapping: k
- The number of queries Q, that is, the number of memory addresses to be searched in the cache.
- The memory address that needs to be searched in the cache: MAR, for each query.

6) Input Range:

- The value of B should only be a power of 2.
- The value of CL can only be a power of 2.
- The value of k should only be a power of 2. Also, the value of k must be less than or equal to the number of cache lines CL.
- There is no limit on the number of queries.
- Since the main memory is considered to be 16-bit in this program, it can only store 65536 addresses in the range of 0 to 65535. Hence, MAR entered must lie in this range.

7)Output Format:

For each query, the output is in the following format:

- The first line of output is the size of the main memory and the number of blocks in it.
- The second line of output is the block number to which the entered memory address MAR belongs to in the main memory.
- The next three paragraphs of output display the cache L1 contents (i.e. which memory block is stored in which cache line) and the result of search for MAR in

cache L1 according to direct mapping, fully associative mapping, and k-way set associative mapping, respectively.

- If there is a cache HIT at level-1 (L1), then the cache line number at which MAR was found is displayed. The data stored in the cache at MAR is also displayed.
- If there is a cache MISS at level-1 (L1), then the contents of cache L2 are displayed. After this the result of the search in L2 is displayed.
- If there is a cache HIT at level 2, then the cache line where it was found is displayed along with the data stored in that address.
- In case of cache MISS at level 2, the block which MAR belongs to replaces some other block in one of the cache lines.These block numbers and cache line number is also displayed.
- Finally,for the cache MISS at level 1, the block which MAR belongs to replaces some other block in one of the cache lines according to the mapping technique considered.

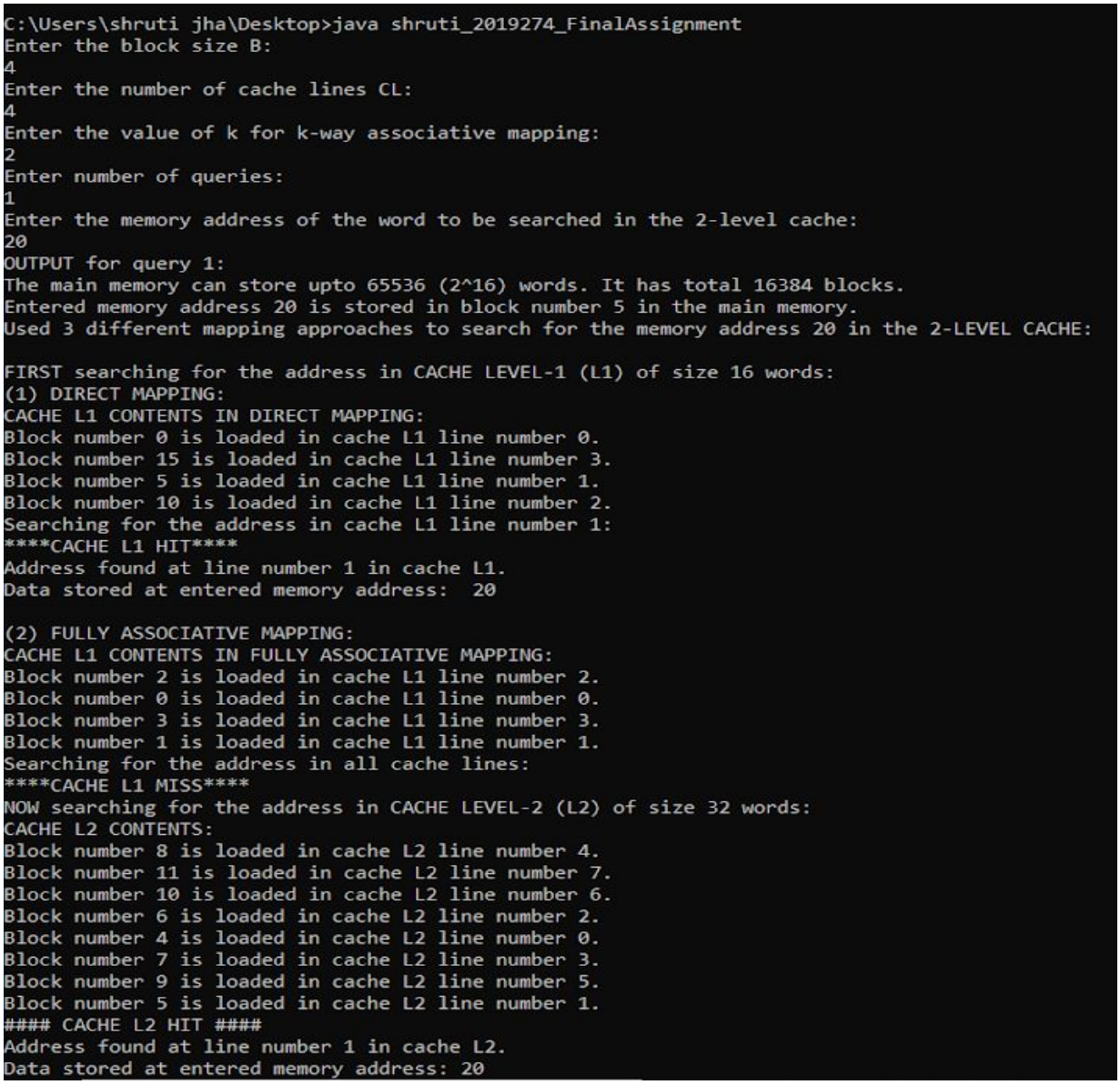
8) Errors Handled:

The program handles 4 kinds of errors:

- If the entered address is outside the range of 0 to 65535, then an error message is shown and execution is aborted.
- If the size of cache requested by the user is greater than the main memory size (i.e. 2^16=65535), then an error message is thrown and the execution is aborted. This is because cache size is always much lesser than the main memory size.
- If the value of k is greater than the number of cache lines, then an error message is shown and execution is aborted.
  This is because the value of k indicates how many cache lines will belong to each set. If k becomes greater than CL, then no sets are possible.

- If the values of either CL or B or k is not a power of 2, then an error is thrown and the execution is stopped (this condition is given in assignment).

Here is a screenshot of the output of of the documented java code implementing cache mapping and 2-level cache searching:



```
#### CACHE L2 HIT ####
Address found at line number 1 in cache L2.
Data stored at entered memory address: 20
Because of cache MISS in L1, block number 1 replaced with block number 5 in line number 1 in cache L1.

(3) 2-WAY SET ASSOCIATIVE MAPPING:
Total number of sets in cache L1= 2
CACHE L1 CONTENTS IN k-WAY SET ASSOCIATIVE MAPPING:
Block number 0 is loaded in cache L1 line number 0.
Block number 7 is loaded in cache L1 line number 3.
Block number 2 is loaded in cache L1 line number 1.
Block number 5 is loaded in cache L1 line number 2.
Searching for the address in cache L1 set number 1:
****CACHE L1 HIT****
Address found at line number 2, set number 1 in cache L1.
Data stored at entered memory address: 20
```