CO ENDSEM ASSIGNMENT

Cache Implementation

Meetakshi Setiya
Roll no. 2019253
Section-A, Group-2

DOCUMENTATION

1- About the project

The source code is an attempt at implementing and simulating a single level cache memory (without the Main Memory) and its working in a computer system. The machine is 32-bit i.e. the size of the instruction is 32-bits. Size of a word has been assumed to be 1 byte.
There are three different types of cache mapping as implemented in the source file-

1) *Direct Mapping*- mapping each block of Main Memory to a single cache line. The instruction (address) is divided into three parts - tag, index and offset.

2) *Fully Associative Mapping*- in this type of mapping, a block from the Main Memory can go anywhere i.e. to any block in the cache. The instruction (address) is divided into two parts - tag and offset.

3) *N-set Associative Mapping*- it is an enhanced form of direct mapping where instead of having only a single cache line that a block of Main Memory can map to, there is a set of n lines that can hold multiple blocks at a single index simultaneously. The instruction (address) is divided into three parts - tag, index and offset.

The Least Recently Used or LRU cache replacement policy is employed to free up space in the cache in case it is full (in case of Fully Associative Mapping) or in case a cache set representing a particular index is full (in case of N-set Associative Mapping).
An age-bit is maintained to keep track of the least recently modified/accessed block to preserve the idea of temporal locality. When a new block is added to the cache, it has the maximum age-bit. Writing/Modifying/Reading from the data in a block also increases its age-bit. At any point if eviction is needed, the block with the least age-bit represents the least recently modified/accessed block and this block is thus evicted to make room for the latest block.

2- Running the code

The code is written in Java so it requires the Java Development Toolkit to be installed in the system on which it is going to be run.

Step 1- Open terminal/command prompt
Step 2- Go to the directory that contains the source code
Step 3- Type javac MeetakshiSetiya_2019253_FinalAssignment.java to compile the program.
Step 4- Type java MeetakshiSetiya_2019253_FinalAssignment to run the program.

In case the file has been renamed, use the updated name instead of MeetakshiSetiya_2019253_FinalAssignment.

3- Input/Output format

The program asks for the following inputs-

1) Cache size and block size
2) Choice of cache mapping method (Direct, Fully Associative or N-set Associative) and the requisite arguments required, if any for a particular mapping method (N, in case of N-set Associative mapping). It then follows the same mapping method for the entire program execution.
3) Choice of operation to be performed on the cache (read data from an address, write data to an address or display the cache contents)
4) Requisite arguments pertaining to the chosen cache operation

Execution continues till an exception/error is reported or when the user explicitly inputs the instruction to quit execution.

For performing a **write** operation, the user is prompted to enter a 32-bit binary address instruction and a 1-byte (or 8-bit) integer data to be stored at that address. Note that the input integer data item can only be a non-negative 8-bit number since the word size is predefined as 1 byte. It can only take a value between 0 to 255.

For performing a **read** operation, the user is prompted to enter a 32-bit binary address instruction. If the address is found in the cache, the program outputs “Cache Hit” and the data stored at the requested address. Please note that all addresses that have been loaded to the cache and not yet written to have a default value of -1. If the requested read address is not loaded into the cache, the program outputs “Cache Miss”.

There is also an option to **display** the contents of the cache at any point of time. It will display the non-null contents of the cache i.e. only those blocks that have been referenced previously and are present in the cache. Different cache blocks/cache lines are separated by horizontal lines for clarity.

NOTE- When there is a write/read in some existing cache block, according to LRU, its age bit changes i.e. it becomes more recently accessed. However, the apparent order of the blocks while printing the cache will remain the same. LRU will be observed when data is being evicted.

4- Important Conditions, Error and Exception Handling

The user needs to follow some important conditions while inputting various data into the program. While in some cases, these input errors are ignored and a new correct input is asked for, in other cases, it leads to an exception that would stop the program abruptly.

1) The cache size and block size must be in powers of two and cache size must always be greater than the block size. Both of them must be non-zero.
2) The user must choose the correct number corresponding to the desired mapping method.
3) In case of N-set Associative Mapping, the size of set N can never exceed the total number of blocks in the cache. N must be a power of 2.
4) Input address must be a valid 32-bit binary number. The input may be less than 32 digits though and the requisite number of 0s would be concatenated to the left of the binary address.
5) The data to be stored at an address must be 8-bits unsigned integer only i.e. 0 to 255.

These conditions, if not followed, would cause the **IllegalArgumentException** prompting the user to restart the program and try again.

The user would have to choose between read, write, display cache and quit execution operations. In case there is an invalid input, the program still continues and would prompt the user to enter a choice till it is valid.

5- A Brief Overview of the Functions Implemented

1) *boolean check_power_of_2(int)*

Checks whether the input argument is a power of two or not. Returns True if the number is a power of 2, False otherwise.

2) *int find_min_index(int n, int lru[], int actual_start_index)*

Returns the index of the least recently used/accessed cache line for the implementation of LRU algorithm.

3) *void write_data(String[] tag, int[][] data, int n, int offset, String address, int inp, int[] lru)*

Writes the input data ‘inp’ at the address “address” in the cache. Stores the respective tag and stores/updates the age-bit (for LRU implementation).

4) *void read_data(String tag[], int[][] data, int n, int offset, String address, int[] lru)*

Reads the data present at the "address". If the block is not present in the memory, "Cache Miss" is displayed as the output. If the block is successfully located, the data present at that address is displayed. Note that all the offsets in a block loaded into the cache have a default value -1. This is also a "Cache Hit" and the default value is displayed. Also updates the age-bit in case of a cache-hit (for LRU implementation).

5) *void display_cache(String[] tag, int[][] data, int offset, int n)*

Prints the contents of the cache present at a point of time. Doesn't print the blocks with null values.

6) *public static void main(String args[])*

The main function.

6- Other Information

Mention that since there is no main memory, once a block is evicted from the cache, its existing information is lost.

Short comments have been provided in the source code to explain a bit of the functioning.

The code is written in Java so a JDK toolkit must be installed on the system to run the program.

N-set associative mapping with N=1 behaves as Direct Mapping and with N= the number of cache blocks i.e. Cache size/block size behaves as Full-Associative Mapping.

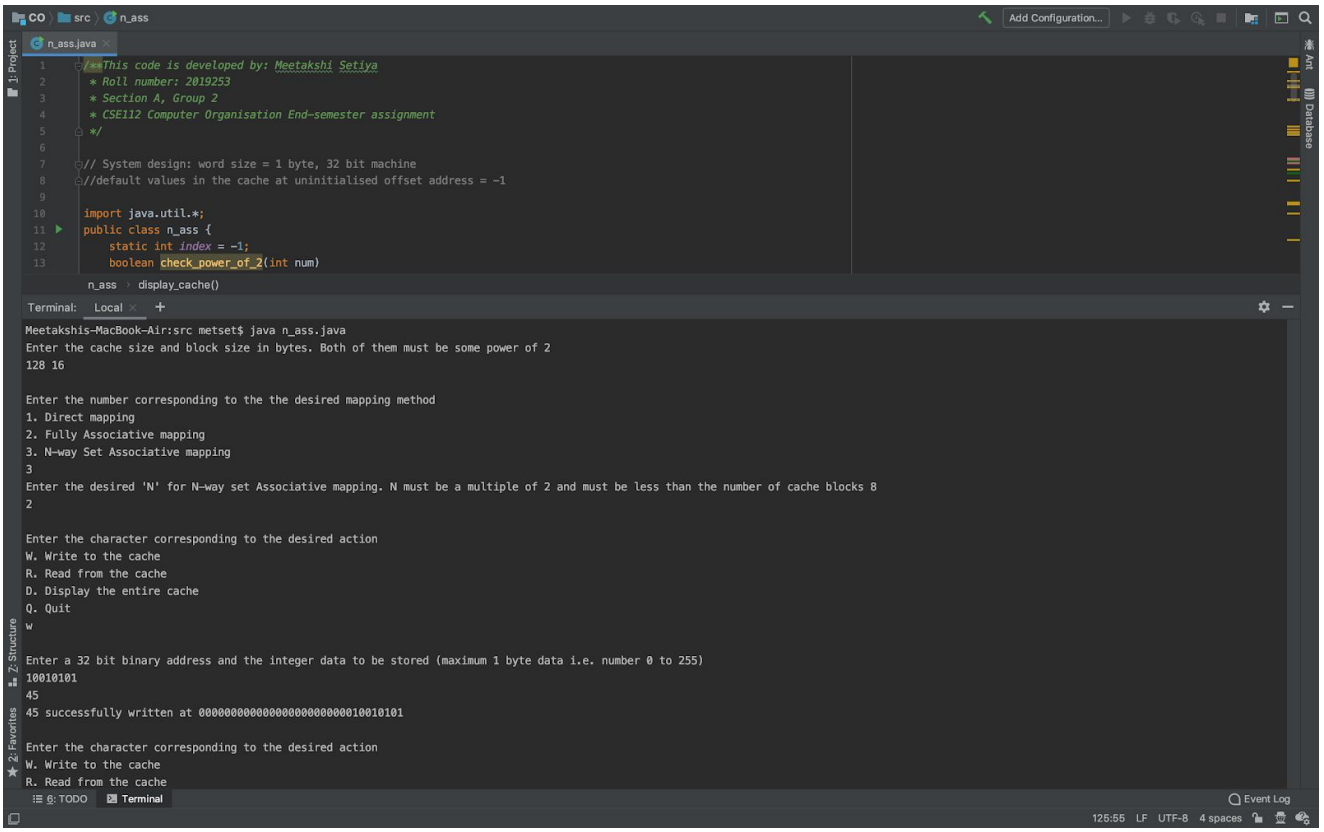
7. Files in the Zip File

1) Java Source Code: MeetakshiSetiya_2019253_FinalAssignment.java
2) Documentation file: MeetakshiSetiya_2019253_FinalAssignment.pdf

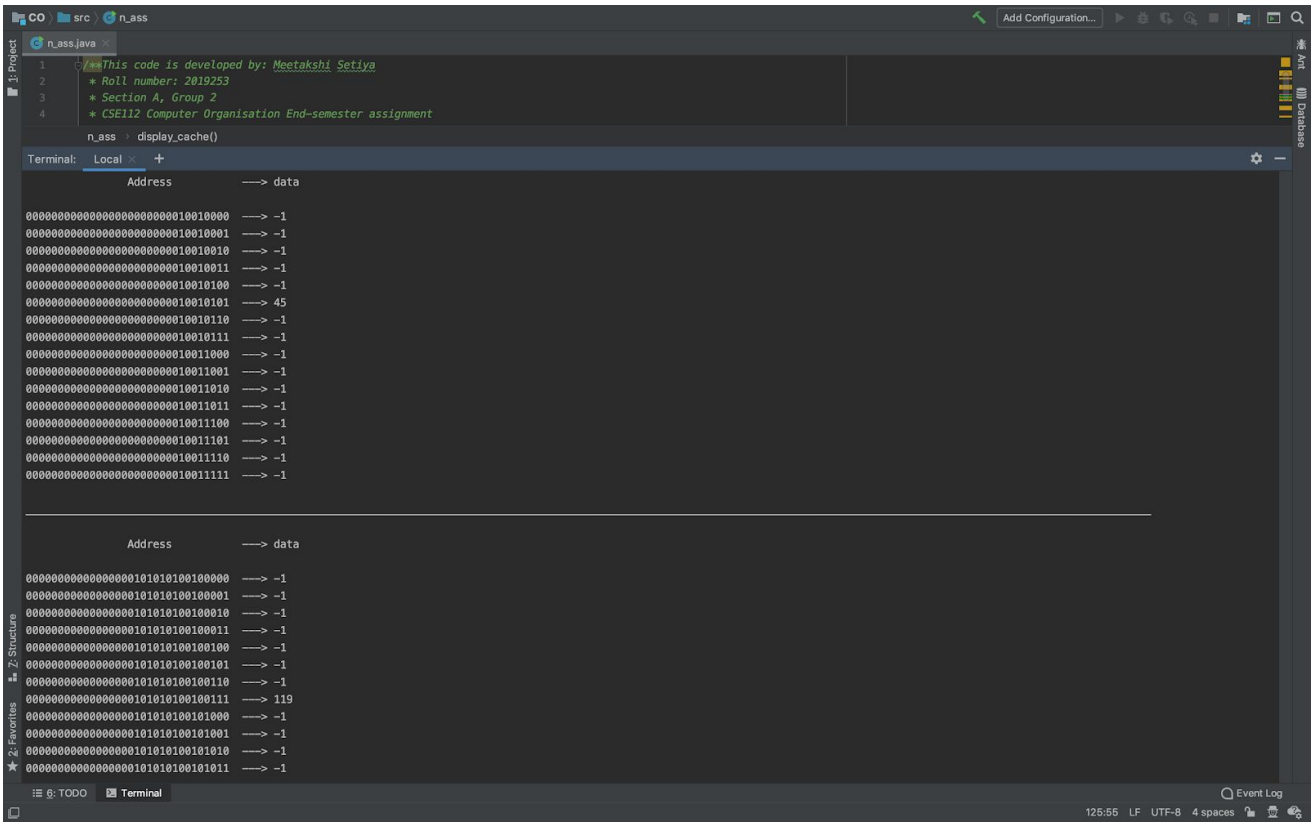
8- Screenshots of the Code Outputs

Note that these are only sample screenshots. **They in no way illustrate the complete functionality of the code**. These are provided just to give an idea of how the code is going to look like when it is run.

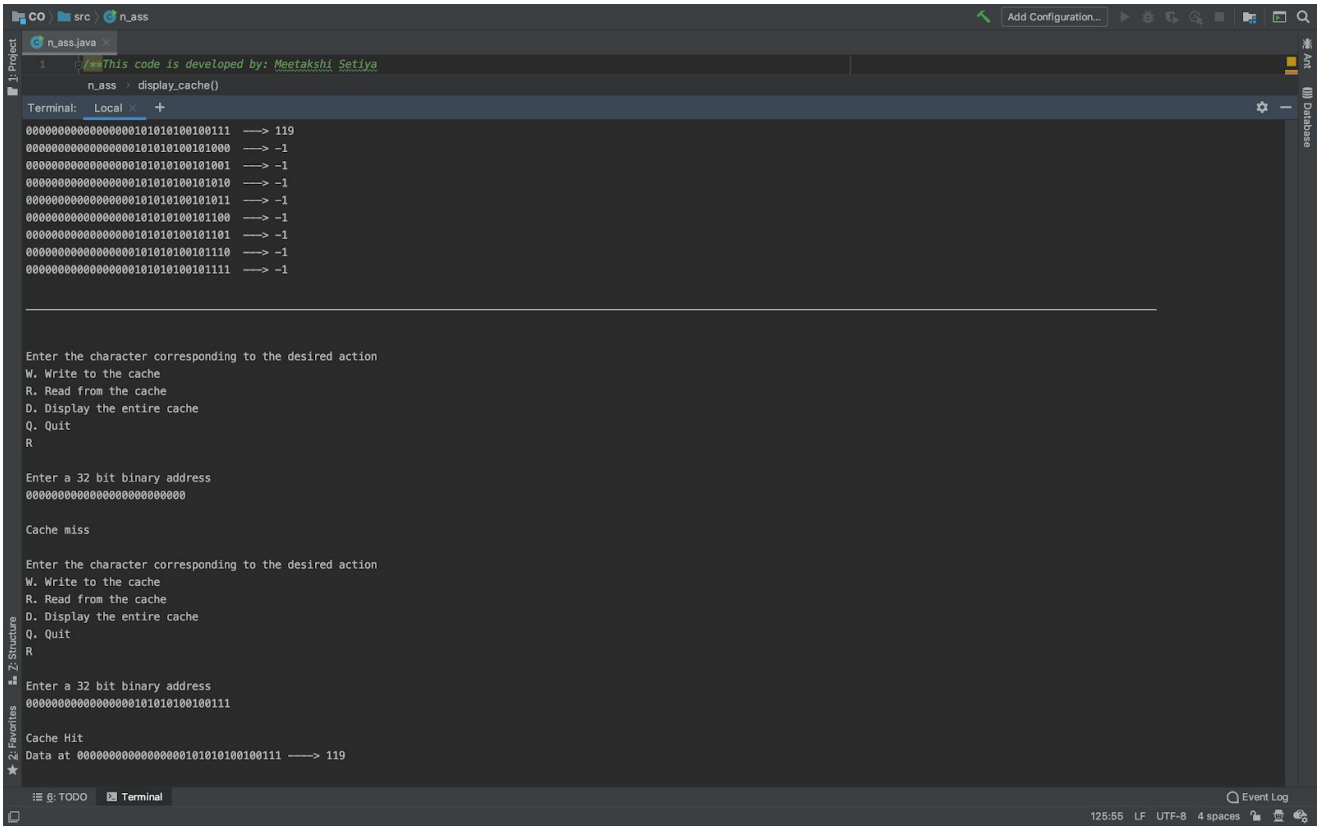
Writing to the cache via N-set Associative Mapping method. N=2 in this case.



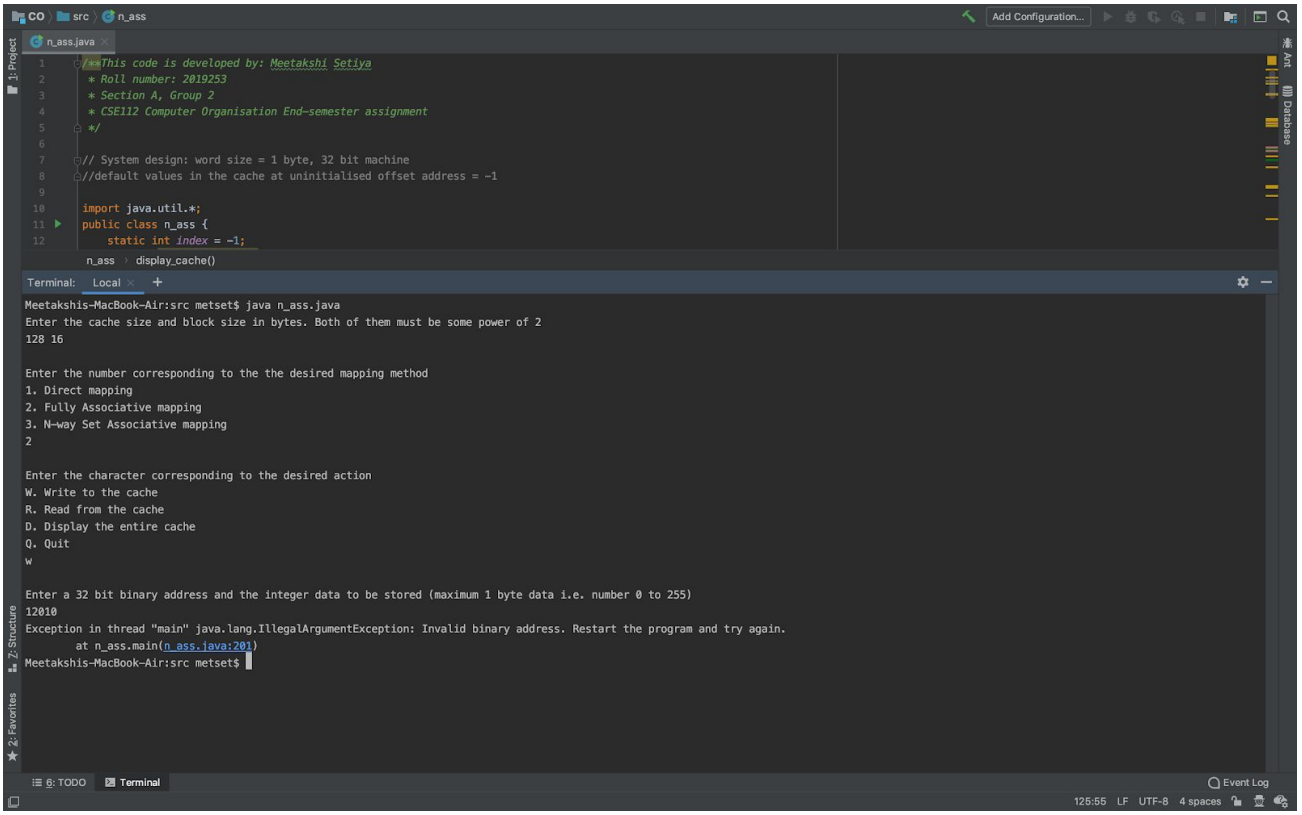
Displaying the contents of the cache. Note how the default value is -1 and the value 45 appears at address 00000000000000000000000010010101



Reading from the cache. Cache miss and hit shown clearly.



An example exception occurrence - IllegalArgumentException (invalid 32-bit binary address)


CO
src
n_ass
Add Configuration...
n_ass.java
/**This code is developed by: Meetakshi Setiya
* Roll number: 2019253
* Section A, Group 2
* CSE112 Computer Organisation End-semester assignment
*/
// System design: word size = 1 byte, 32 bit machine
//default values in the cache at uninitialised offset address = -1
import java.util.*;
public class n_ass {
static int index = -1;
n_ass
display_cache()
Terminal:
Local
Meetakshis-MacBook-Air:src metset$ java n_ass.java
Enter the cache size and block size in bytes. Both of them must be some power of 2
128 16
Enter the number corresponding to the the desired mapping method
1. Direct mapping
2. Fully Associative mapping
3. N-way Set Associative mapping
2
Enter the character corresponding to the desired action
W. Write to the cache
R. Read from the cache
D. Display the entire cache
Q. Quit
w
Enter a 32 bit binary address and the integer data to be stored (maximum 1 byte data i.e. number 0 to 255)
12010
Exception in thread "main" java.lang.IllegalArgumentException: Invalid binary address. Restart the program and try again.
at n_ass.main(n_ass.java:201)
Meetakshis-MacBook-Air:src metset$
1: Project
Z: Structure
2: Favorites
Ant
Database
6: TODO
Terminal
Event Log
125:55
LF
UTF-8
4 spaces