Presented to Mr. Ronald Pascual,

Department of Computer Technology – College of Computer Studies

De La Salle University – Manila

Term 3, A.Y. 2023-2024

Direct Cache Simulator

In Partial Fulfillment of the Course Requirements for

Introduction to Computer Organization and Architecture 2

(CSARCH2)

Submitted by:

Dimaculangan, Aldwin Renzel P.

Olores, Sean Andrei P.

Panugayan, Enzo Arkin

Santos, Alyssa Jane C.

July 31, 2024

I. Introduction

The study's output simulates Direct Cache Mapping. In short, cache memory is a high-speed storage area in a computer— acting as a buffer between the CPU and main memory. The buffer cache memory creates allows for faster data access, thereby increasing the overall performance and the efficiency of a system, which helps majorly for running applications, particularly similar to those that run in our modern age today, efficiently. This makes cache memory a vital component in modern computing systems.

Direct cache mapping, which the project output will be emulating, is a technique used to identify the location of main memory addresses. As each block of main memory links to only one cache line, when one is already occupied and a new memory block can be loaded, the existing block is replaced with the new one, resulting in the previous block being discarded. This practice makes sure that the cache is updated continuously with the most relevant, recent information.

A problem with utilizing this technique is that cache conflicts may occur if multiple blocks direct the program to the same cache line, which opens up the possibility of causing more frequent replacements and cache misses. The developers, in turn, for this paper, will be navigating the aforementioned technique to utilize it in a manner wherein this problem does not arise; contributing to their knowledge of correct development practices when dealing with direct cache mapping.

II. Methodology

A. Web Application Interface

The cache simulator was developed using HTML, CSS, and Javascript with a design that mimics a game interface. This web application requires the user to input data. After inputting the data, corresponding outputs are displayed. The simulator offers an engaging platform for exploring cache behavior, making it easier to grasp complex concepts related to computer systems and memory management.

a. Input

Input	Expected Input
Block Size	Number of words per block
MM Memory Size	Number of blocks/words
Cache Memory Size	Number of blocks/words
Sequence	Sequence of Blocks/Words to be accessed in main memory.
Cache Access Time	Time taken in accessing the cache
Memory Access Time	Time taken in accessing the memory
Cache Miss Memory	Mode on how the cache should fetch or write data.

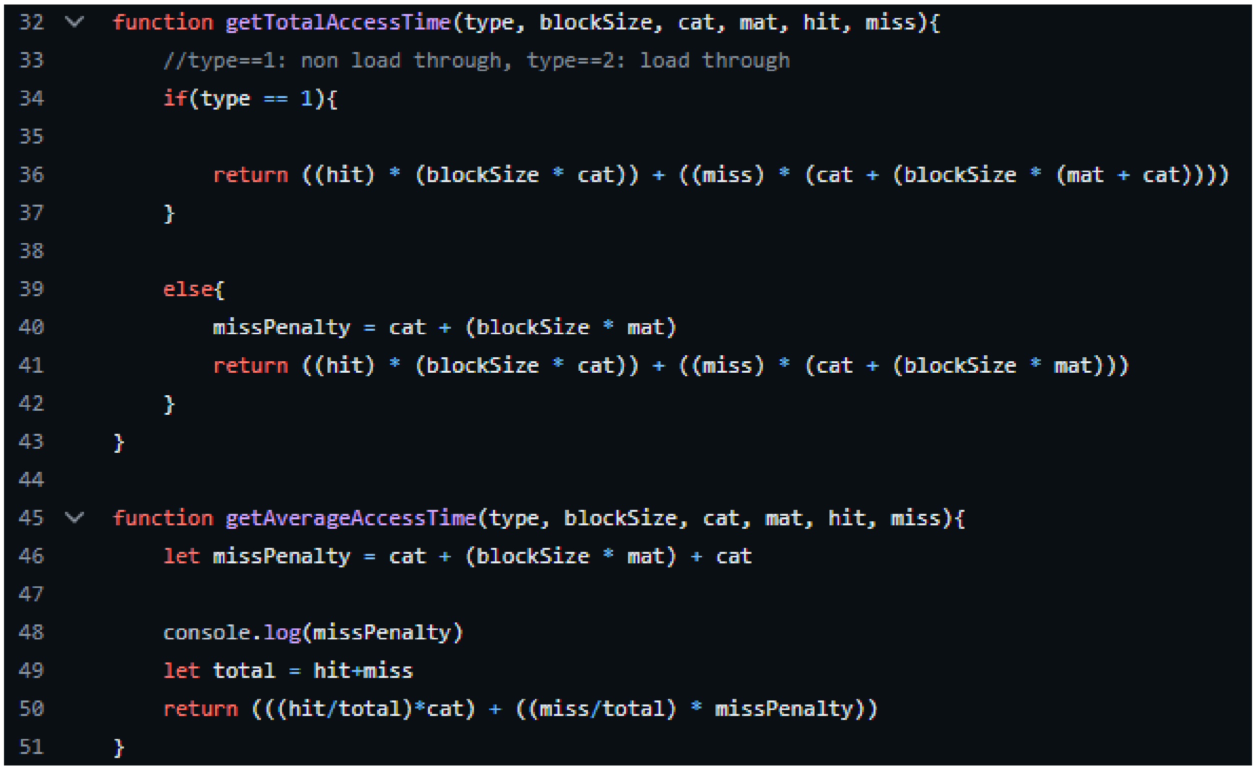
b. Output

Output	Expected Output
Cache Hits	Number of successful hits
Cache Miss	Number of misses
Miss Penalty	Time penalty received when there is a Cache Miss
Hit Rate	Hits / Total Number of Accesses
Miss Rate	Misses / Total Number of Accesses
Average Memory Access Time	[(Hit Rate) x (CAT)] + [(Miss Rate) x (Miss Penalty)]
Total Memory Access Time	Non Load Through Total access time: [(# Hit) x (Block Size x CAT)] + [(# Miss) x (Check Cache + (Block Size * (MAT + CAT)))] Load Through Total access time: [(# Hit) x (Block Size x CAT)] + [(# Miss) x (Check Cache + (Block Size * MAT))]

B. Web Application Programming

The properties of each block are located in block.js. This includes the physical attributes of the block, along with the code that allows the blocks to be moved around. This is further supported by the file util.js which can detect if there are collisions between blocks and which direction the collision occurs, together with checking if two or more blocks are vertical to each other, which allows for blocks to be stacked on top of each other instead of one block covering the other.

The outputs are computed in the file directMapping.js. This includes getting the total access time, average access time, along with computing for miss penalties and hit or miss rates. The program is also capable of computing both non load through and load through cache misses.



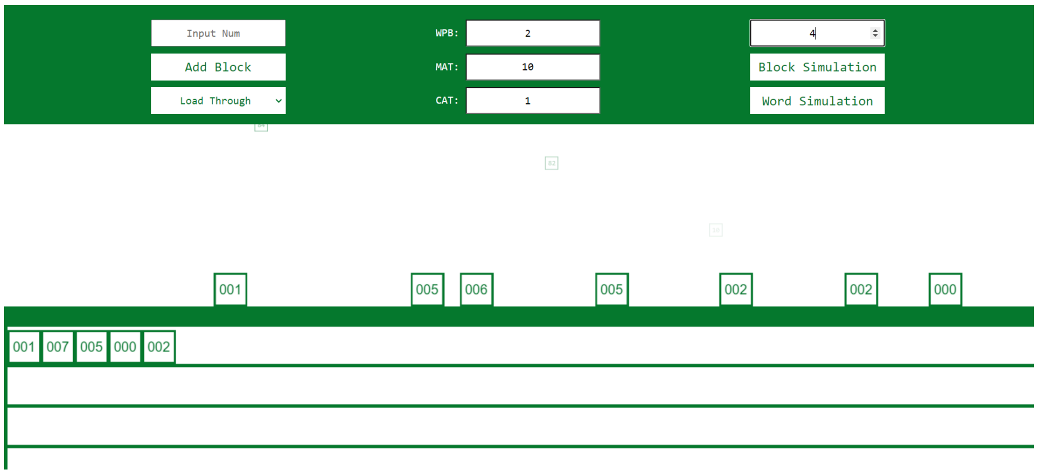
```
function getTotalAccessTime(type, blockSize, cat, mat, hit, miss){
    //type==1: non load through, type==2: load through
    if(type == 1){

        return ((hit) * (blockSize * cat)) + ((miss) * (cat + (blockSize * (mat + cat))))
    }

    else{
        missPenalty = cat + (blockSize * mat)
        return ((hit) * (blockSize * cat)) + ((miss) * (cat + (blockSize * mat)))
    }
}

function getAverageAccessTime(type, blockSize, cat, mat, hit, miss){
    let missPenalty = cat + (blockSize * mat) + cat

    console.log(missPenalty)
    let total = hit+miss
    return (((hit/total)*cat) + ((miss/total) * missPenalty))
}
```

Figure: Functions to get total and average access times

III. Results and Discussion

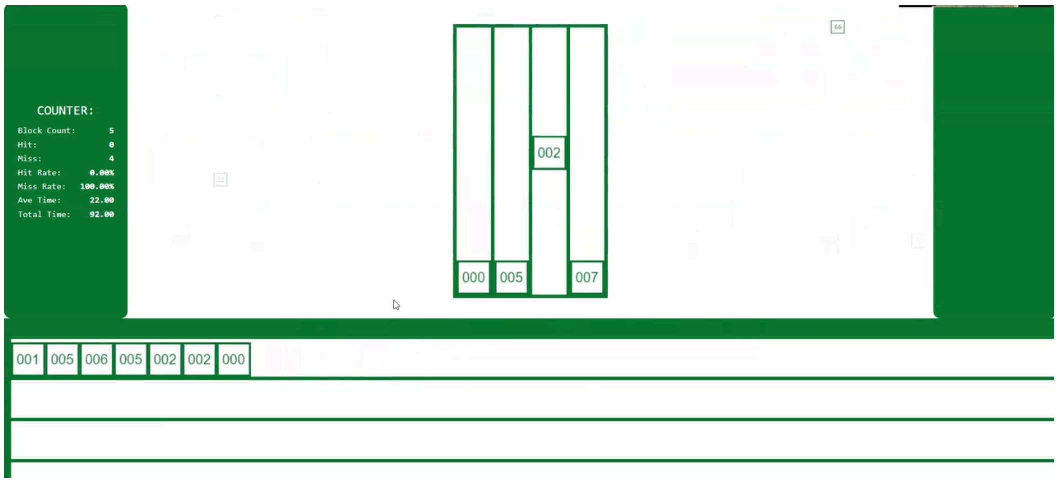
The resulting simulator is a web application hosted on <site> that showcases direct cache mapping in a gamified approach. When the user creates blocks, they are given the freedom to arrange the blocks in any way they prefer, this arrangement denotes the order that the memory gets accessed.

There are 6 inputs required from the user before starting the simulator. These inputs are as follows: Block Size, MM Memory Size, Cache Memory Size, Sequence, Cache Access Time, and Memory Access.



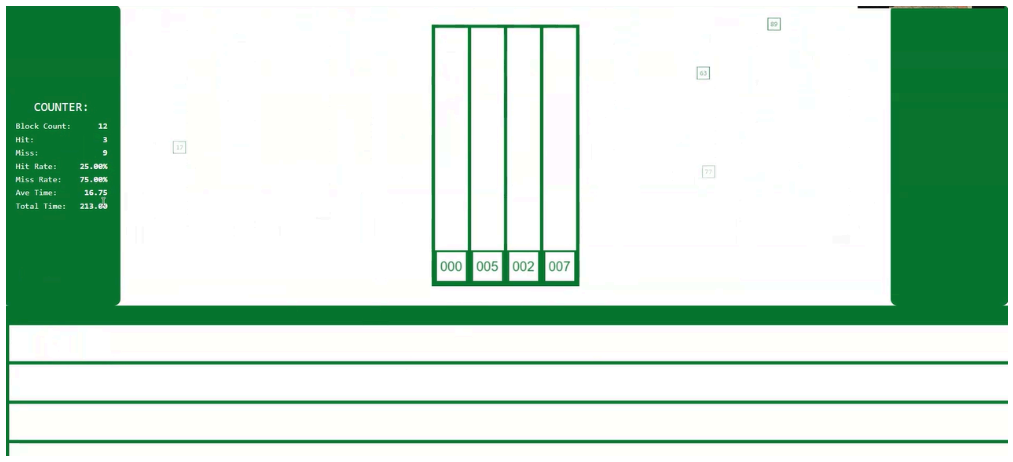
WPB: 2
MAT: 10
CAT: 1
4

The figure above shows a sample input where inputted blocks are 1,7,5,0,2,1,5,6,5,2,2,0 along with the Word per Block of 2, Memory Access Time of 10, Cache Access Time of 1, and Cache Size of 4.



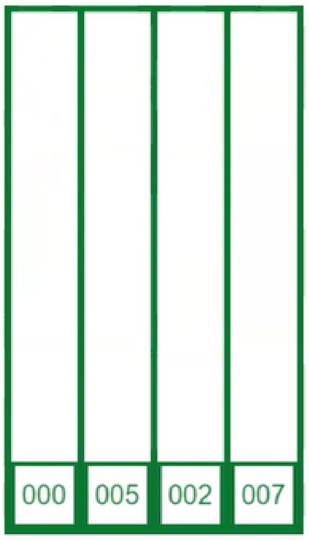
```
COUNTER:
Block Count:        5
Hit:                0
Miss:               4
Hit Rate:       0.00%
Miss Rate:    100.00%
Ave Time:       22.00
Total Time:     92.00
```

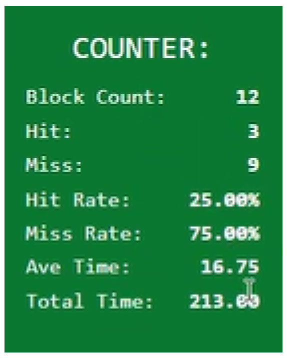
The next page shows the simulation results for this input. As this is a gamified approach, the user can visually see how these blocks are located and replaced along with the metrics on the side providing insight on Cache Hits, Cache Miss, Miss Penalty, Hit Rate, Miss Rate, Average Memory Access Time, and Total Memory Access Time.



```
COUNTER:
Block Count:    12
Hit:             3
Miss:            9
Hit Rate:   25.00%
Miss Rate:  75.00%
Ave Time:    16.75
Total Time: 213.00
```

At the end of the simulation, the final state of the cache can be visualized, along with the final numbers for the mentioned data about the output. A snapshot of the final state of the cache and the table displaying the outputs can be seen below.





```
COUNTER:
Block Count:    12
Hit:             3
Miss:            9
Hit Rate:   25.00%
Miss Rate:  75.00%
Ave Time:    16.75
Total Time: 213.00
```

IV. Conclusion

Overall, through thorough research in related literature as well as utilizing fundamentally sound developmental practices, the Direct Cache Mapping Simulator, serving as the paper’s output, effectively explores cache systems through an engaging and straightforward interface, making it an efficient tool especially for users just starting to grasp the essentials regarding direct-mapped caching.

The program mainly does this through offering a clear view as to how data is assigned to these cache lines, and in turn, how they are discarded. This easy-to-follow presentation of data was designed by the team with two concepts in mind: user-friendliness and engagement quality. As such, the output offers an interface that is simple enough to make navigating the program seamless while still being able to get information as to how direct cache mapping operates across to the user— which is really the main point of the program in the first place.

As a conclusion, backed by the variable of facts aforementioned, the developed web application creates a hands-on experience for the user that is both easy to understand and fun to use, effectively making the product a great tool for anyone that wants to learn how data storage in direct cache mapping works.