# Cache Simulator: Assignment Report

Ashwin Agrawal
Roll No: CS22BTECH11009

November 23, 2023

## 1 Introduction

This report presents the implementation and results of the Cache Simulator assignment for the CS2323 Lab-6. The objective of this assignment is to simulate the behavior of a cache memory system based on various replacement and write policies. I have correctly implemented all 4 parts given in the problem statement.

## 2 Coding Approach

The Cache Simulator was implemented following a systematic approach to handle various Replacement and Write Policies. The implementation can be divided into several key steps:

- First, the program reads the configurations from the configuration file.
- Then it reads the access sequence from the **cache.access** file and converts those access sequences from **hex** format to **binary** format and stores the in an vector of pair of strings called **input_vector** where the 1st element of the pair denotes the Access Mode and the 2nd element of the pair denotes the Address in Binary Format

```
1    vector<pair<string, string>> input_vector; // Here the first
     element of the pair is the mode and the second element is the
     address
2
```

Listing 1: Vector Containing Access Mode and Sequence

- According to the configuration, the bits required for the set, offset and tag are calculated. After which the corresponding function according to the Replacement Policy is called.

```
1    int number_of_sets = size_of_cache / (block_size *
     associativity);
2    int set_bits = log2(number_of_sets);
3    int offset_bits = log2(block_size);
4    int tag_bits = 32 - set_bits - offset_bits; // It was clearly
     mentioned in the assignment pdf that we can assume 32 bit
     address always
```

```
5
6       if (replacement_policy == "LRU")
7       {
8           deal_LRU(number_of_sets, tag_bits, set_bits, associativity,
        replacement_policy, writeback_policy, input_vector);
9       }
10
```

Listing 2: Calculating Bits And Calling the Respective Functions

- Inside each of these Functions, the Cache as a 2-D vector is initialized, which is populated with the cacheElemts and the **Valid Bit** is set to 0 .

```
1       vector<vector<cacheElements *>> LRU_cache(number_of_sets,
       vector<cacheElements *>(associativity));
2       for (int i = 0; i < number_of_sets; i++)
3       {
4           for (int j = 0; j < associativity; j++)
5           {
6               cacheElements *temp = new cacheElements;
7               temp->valid = 0;
8               LRU_cache[i][j]= temp;
9           }
10      }
11
```

Listing 3: Population the Cache For the First Time

- The following denotes the structure of a cacheElement

```
1 struct cacheElements
2 {
3       string tag;
4       int valid;
5       int dirty;
6 };
7 typedef struct cacheElements cacheElements;
8
```

Listing 4: Structure of cacheElement

- Then According to the Replacement Policy, a suitable datastructure for maintaining the Reaplacement Sequence is created

```
1       vector<vector<int>> access_time(number_of_sets, vector<int>(
       associativity, 0)); // vector to store access time.
2
3
```

Listing 5: DataStructure for maintaining LRU replacement Sequence

- Then I start Iterating over the **input_vector**, for every access, I calculate the set and the tag. And I check If its a hit or miss.

```
1       for (int i = 0; i < input_vector.size(); i++)
2       {
3           string mode = input_vector[i].first;
4           string address = input_vector[i].second;
5
```

```
        string tag = address.substr(0, tag_bits);
        string set = address.substr(tag_bits, set_bits);
        int set_number;
        if (set_bits != 0)
        {
            set_number = binToDec(set);
        }
        else
        {
            set_number = 0;
        }

        bool hit = false;
        int hit_index;
        for (int i = 0; i < associativity; i++)
        {
            if (random_cache[set_number][i]->valid == 1 &&
   random_cache[set_number][i]->tag == tag)
            {
                hit = true;
                hit_index = i;
                break;
            }
        }

```

Listing 6: Iterating Over the Access Sequence

- According to the Access Mode, write policy, Replacement Policy and the fact that it is a hit or miss, I make the necessary changes to the cache and print the result.

```
if (hit){
    //Printing Output and Changing the access time.
}
else{
if ((mode == "W" && writeback_policy == "WB")||(mode == "R")){
            bool is_line_full = true;
            int index_of_empty_location = -1;
            for (int i = 0; i < associativity; i++){
                if (random_cache[set_number][i]->valid == 0){
                    index_of_empty_location = i;
                    is_line_full = false;
                    break;
                }
            }
            if (is_line_full){
                //Replacing the least Accessed element
            }
            else{
                //Adding that data to the cache.
            }
        }
}

```

Listing 7: Changes Made to Cache

# 3 Usage

To use the Cache Simulator, provide the Cache Configuration in the **cache.config** file and the access sequence in the **cache.access** file. And the please run the following commands.

```
g++ source_code.cpp

./a.out
Address: 0x20203302, Set: 0x00, Miss, Tag: 0x202033
Address: 0x20203302, Set: 0x00, Hit, Tag: 0x202033
Address: 0x20202011, Set: 0x02, Miss, Tag: 0x202020
Address: 0x20202011, Set: 0x02, Hit, Tag: 0x202020
Address: 0x20203302, Set: 0x00, Hit, Tag: 0x202033
Address: 0x20202011, Set: 0x02, Hit, Tag: 0x202020
```

Listing 8: Running the Cache Simulator

# 4 Testing

Testing Procedure for Code Correctness

1. Conducted separate tests for each replacement policy (FIFO, LRU, Random).
2. Checked for expected cache hit/miss outcomes based on the chosen policy.
3. Conducted Tests for different combinations of write policies.
4. Tested on random test cases with diverse access sequences to assess the simulator's correctness.
5. Verified correct initialization of tag_bits, set_bits and set_number.

# 5 Conclusion

The assignment enhanced my understanding of Cache and the different concepts related to it.