CS2323 - Computer Architecture
Lab - 6 (Cache Simulator)

Soham Rajesh Pawar
CS22BTECH11055

November 12, 2023

1 Coding Approach

The program is implemented in C++ for Cache Simulation. Here is an overview of the coding approach:

1.1 Class block

The program defines a class called `block`, which stores the tag (when converted to an integer).

Listing 1: class block

```
class block{

private:
        int tag;

public:
        block(int val) : tag(val){}
        int get() {return tag;}
};
```

1.2 Parsing the Input

The program uses functions like `substr()`(inbuilt), `Hex_to_Binary()`, `Extend()` and `stoul()`(inbuilt) to parse the input, convert the parsed hexadecimal string to binary string, extend the binary string to 32 bits and convert the binary strings to integer.

1.2.1 Hex_to_Binary()

Listing 2: Hex_to_Binary() function

```
string HexToBinary(const string& hexString) {

        string binary = "";

        size_t start = (hexString.size() > 2 && hexString.
           substr(0, 2) == "0x") ? 2 : 0;

        for (size_t i = start ; i < hexString.size() ; ++i) {

                int decimal;

                if (isdigit(hexString[i])) decimal = hexString[
                   i] - '0';
```

```
                        else decimal = toupper(hexString[i]) - 'A' +
                            10;

                        binary += bitset<4>(decimal).to_string();
                }

                return binary;
        }
```

1.2.2 Extend()

Listing 3: Extend() function

```
        string Extend(const string& binary) {

            int zeros = 32 - binary.length();

            string extended_binary = string(zeros, '0') + binary;

            return extended_binary;
        }
```

1.3 Handling Direct Mapped Cache and Set Associative Cache.

Listing 4: Logic to handle non associative caches.

```
if (associativity > 0) {

        int index_bits = (int)(log(index) / log(2));
        int tag_bits = address_bits - (index_bits + block_bits);
        vector<block> cache[index];
        for (int i = 0 ; i < index ; i++) {

                vector<block> set;
                cache[i] = set;
        }

        while (getline(access_file, line)) {

                bool in_set = 0;
                int mode;
                int word_request_set;
                int word_request_tag;
                string word_request_address;
                mode = (line[0] == 'R' ? 1 : 0);
                word_request_address = Extend(HexToBinary(line.substr
                    (3)));
                word_request_tag = stoul(word_request_address.substr(0,
                     tag_bits), nullptr, 2);
                word_request_set = stoul(word_request_address.substr(
                    tag_bits, index_bits), nullptr, 2);

                block processor_call(word_request_tag);

                for (int i = 0 ; i < cache[word_request_set].size() ; i
                    ++) {

```

```
28                          if (cache[word_request_set].at(i).get() ==
                                word_request_tag) {
29
30                                  total_hits++;
31                                  in_set = 1;
32                                  if (replace == "LRU") {
33
34                                          cache[word_request_set].erase(
                                                cache[word_request_set].
                                                begin() + i);
35                                          cache[word_request_set].
                                                push_back(processor_call);
36                                  }
37                                  break;
38                          }
39                  }
40
41                  if (!in_set) {
42
43                          total_misses++;
44
45                          if (mode || write != "WT") {
46
47                                  if (cache[word_request_set].size() <
                                        associativity) cache[
                                        word_request_set].push_back(
                                        processor_call);
48                                  else {
49
50                                          if (replace == "LRU") cache[
                                                word_request_set].erase(
                                                cache[word_request_set].
                                                begin());
51                                          else if (replace == "FIFO")
                                                cache[word_request_set].
                                                erase(cache[word_request_set
                                                ].begin());
52                                          else {
53
54                                                  random_device seed;
55                                                  mt19937 gen(seed());
56
57                                                  uniform_int_distribution
                                                        <int> random(0,
                                                        cache[
                                                        word_request_set].
                                                        size() - 1);
58                                                  cache[word_request_set
                                                        ].erase(cache[
                                                        word_request_set].
                                                        begin() + random(gen
                                                        ));
59                                          }
60
61                                          cache[word_request_set].
                                                push_back(processor_call);
62                                  }
63                          }
```

```
                }

                cout << "Address : " << line.substr(3) << ", Set : " <<
                    hex << showbase << word_request_set << (in_set == 1
                    ? ", Hit" : ", Miss") << ", Tag : " << hex <<
                   showbase << word_request_tag << endl;
        }
}
```

1.4 Handling Fully Asssociative Cache.

Listing 5: Logic to handle fully associative cache.

```
else {

        int tag_bits = address_bits - block_bits;
        vector<block> cache;

        while (getline(access_file, line)) {

                bool in_set = 0;
                int mode;
                int word_request_tag;
                string word_request_address;
                mode = line[0] == 'R' ? 1 : 0;
                word_request_address = Extend(HexToBinary(line.substr
                   (3)));
                word_request_tag = stoul(word_request_address.substr(0,
                    tag_bits), nullptr, 2);

                block processor_call(word_request_tag);

                for (int i = 0 ; i < cache.size() ; i++) {

                        if (cache.at(i).get() == word_request_tag) {

                                total_hits++;
                                in_set = 1;
                                if (replace == "LRU") {

                                        cache.erase(cache.begin() + i);
                                        cache.push_back(processor_call)
                                           ;
                                }
                                break;
                        }
                }

                if (!in_set) {

                        total_misses++;

                        if (mode || write != "WT") {


                                if (cache.size() < index) cache.
                                   push_back(processor_call);
                                else {
```

```
41
42                                          if (replace == "LRU") cache.
                                                erase(cache.begin());
43                                          else if (replace == "FIFO")
                                                cache.erase(cache.begin());
44                                          else {
45
46                                                  random_device seed;
47                                                  mt19937 gen(seed());
48
49                                                  uniform_int_distribution
                                                        <int> random(0,
                                                        cache.size() - 1);
50                                                  cache.erase(cache.begin
                                                        () + random(gen));
51                                          }
52                                  }
53
54                                  cache.push_back(processor_call);
55                          }
56                  }
57
58                  cout << "Address : " << line.substr(3) << (in_set == 1
                        ? ", Hit" : ", Miss") << ", Tag : " << hex <<
                        showbase << word_request_tag << endl;
59          }
60  }
```

2 Explanation

Using the configurations provided in the config file number of sets, index_bits, block_bits and other useful quantities are calculated. These are then used to create a data structure which can model the behaviour of the required cache.

For a non associative cache an array of vector of the object block proves to be a useful data structure to simulate the cache. For a fully associative cache a vector of the object block is sufficient.

The crux of the problem is to realise that the only difference between the read and write modes appears when the writeback policy is **WriteThrough** and the status of the word is a miss.
In every other case if the status is a hit we renew block(only if the replacement policy is LRU) and update the hit counter
And if the status is a miss we check if the corresponding set(the whole cache in case of fully associative cache) has reached maximum occupancy(i.e the associativity). If not then we append the block to the set, else we remove the victim based on the policy of replacement and then append the block to the set. Finally we update the miss counter.
In case the writeback policy is **WriteThrough** then on a miss we only update the miss counter.

3 Conclusion

The program successfully simulates a cache with the given configuration settings and correctly prints the data regarding the requested word. There is error handling included in the code, however it is not as robust as an actual simulator that could be found online as that is not the point of the assignment.