Lab 6 Report

Yash Patel

CS22BTECH11047

1 Introduction

This report describes the code for a basic implementation of cache memory. It describes the cache behaviour and outputs the index/set for the given address, whether it is a hit or miss, and the TAG being stored in the cache. All the parts given in the question are implemented in the code.

2 Algorithm Description/Coding Approach

The code employs a straightforward algorithm that includes the following steps:

1. Extract all the information from the files and store it.
2. If any address (in hexadecimal) does not have 8 digits, add required leading zeroes, and convert the addresses to binary number system.
3. Calculate offset, set bits and tag bits from the given information.
4. For implementing cache, create a 2-D array, where the sets (lines) are the rows and associativity is the number of columns.
5. Now, iterate through each instruction:
    - Extract tag and set from the address.
    - Calculate set index to get the set to which the address maps.
    - Now, search the tag in the set, which we calculated above.
    - If we found the tag in the set, then it is a Cache HIT, otherwise, it is a Cache MISS.
    - If it is a hit and the replacement policy is LRU, then we update the last access time, else we are not required to do anything in any other case.
    - If it is a miss with write mode and write policy write through (and NO Allocate), directly main memory is updated and we don't need to do anything in the cache. Otherwise, we need to fetch the block from the main memory to cache.
    - If there is space available for the fetched block, then we directly place it in the cache, else replacement policy is to be used.
6. Approach for FIFO: Remove the block from front and add new block at the last (just like a queue).
7. Approach for LRU: Store last access time for each block. Search the block with the largest last access time and replace it with the new block.
8. Approach for RANDOM: Select a number randomly from [0, associativity-1], and replace it with the new block.

3 Testing and Validation

The code has been tested and validated using some sample inputs and expected outputs. Testing involved different combination of cache configurations. The address files were mainly made randomly but with some condition such that there were some chances of getting the same address again, due to which a cache hit can occur. Also, i calculated total hits and total misses in a program to get a good idea about the working.