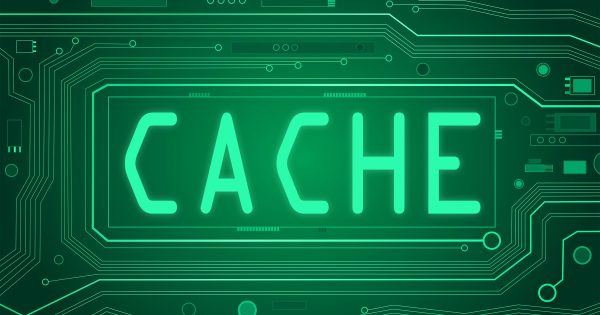
**ECE 585**

**Cache Coherence Project**



Team 12

Members:

Alex Beaulier

Shreya Bendre

Anvitha Bandla

Amrutha Anil

[Introduction 2](#_Toc66473229)

[Team Roles and Responsibilities 2](#_Toc66473230)

[System Specification 2](#_Toc66473231)

[Background 3](#_Toc66473232)

[Design Decisions 7](#_Toc66473233)

[Validation Summary 8](#_Toc66473234)

[Appendix A: Source Code for cache and associative modules 10](#_Toc66473235)

[Appendix B: Simulation results along with usage statistics 10](#_Toc66473236)

[References 13](#_Toc66473237)

# Introduction

The team was responsible for the design and simulation of a split L1 cache for a new 32-bit processor. It could be used with up to three other processors in a shared memory configuration and the system employs the MESI protocol.

# Team Roles and Responsibilities

See source code file header authors for specific module contributions by team members. Debugging and final report contributions also very similar. Acknowledging Shreya for describing MESI protocol in report and Anvitha for trace file draft.

Additional acknowledgement to Rahul Marathe.[2] Used and modified cache simulator for starting basis of second modeling attempt of project.

# System Specification

**SYSTEM REQUIREMENTS:**

System shall use up to three other processors

System shall employ MESI protocol for cache coherence

L1 Cache

* + - Instruction cache - Four way set associative
      * 16k sets and 64 byte lines
    - Data cache - 8-way set associative
      * 16k sets and 64 byte lines
      * Is write back using write allocate
      * Is write back except for first write to line which is write through

Both caches employ LRU replacement policy

Both Caches are backed by shared L2 cache

Cache must be described and simulated in Verilog, C or C++.

* + - Verilog doesn’t need to be synthesizable, clock accurate and doesn’t need to store/retrieve data.

System Modes

* + - Mode 1 the simulation shall display only the required summary of usage statistics and responses to 9s in the trace file and nothing else.
    - Mode 2 shall display everything from mode 1 but also display communication messages to the L2 described above and nothing else.
    - Additional modes may be implemented but not present in the demo

**SYSTEM OUTPUTS:**

System shall maintain and report following key statistics of cache usage for each cache and display upon completion of the trace:

* + - # of cache reads
    - # of cache writes
    - # of cache hits
    - # of cache misses
    - # cache hit ratio

In order to maintain inclusivity and implement MESI protocol, L1 caches may have to communicate with the shared L2 Cache, to simulate the system shall display the following messages:

* + - Return data to L2 <address>
      * In response to a 4 in trace file, cache shall signal that it’s returning the data for that line (if present and modified)
    - Write to L2 <address>
      * Operation is used to write back a modified line to the L2 upon eviction from L1 cache. Also used for initial write through when cache line is written for the first time so L2 knows its been modified and has the correct data
    - Read from L2 <address>
      * Operation is used to obtain data from L2 on an L1 cache miss
    - Read for ownership from L2 <address>
      * This operation used to obtain data from L2 on an L1 cache write miss

**DOCUMENTATION**

1. Documented, described and simulated cache in Verilog(not clock accurate)
2. Validation Documentation
3. Specification Checklist

# Background

The topics discussed for understanding the specification are; caches, inclusivity, LRU replacement policy, MESI protocol, and various mathematical equations within the cache.

Caches

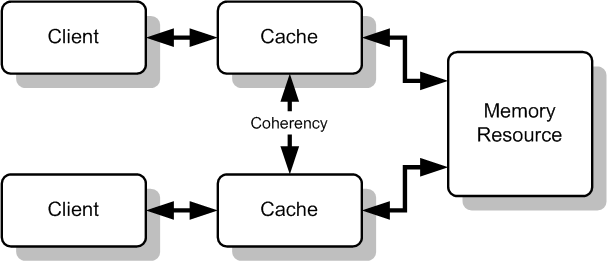
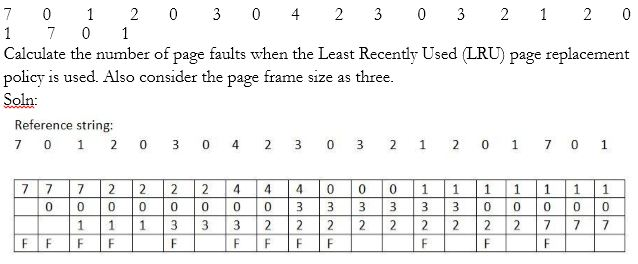
A general diagram for Cache Coherency is given below. Two processors separately receive data and communicate with each other to maintain commonly accessed memory to serve clients. 

Figure 1: [LINK](https://www.extremetech.com/wp-content/uploads/2016/10/Cache_Coherency_Generic.png)

LRU Replacement

Least Recently Used (LRU) replacement policy replaces the page that hasn’t been used in the longest period of time. It was made to better the efficiency of the optimal page replacement algorithm. The idea is that pages that have been heavily used will probably be used again in the next requests. Pages that haven’t been used will likely remain unused and are statistically outliers to be removed. To implement, it is necessary to maintain a linked list of all pages with the most recently used page at the front and the least recently used page in the rear. The list must be updated on every memory reference. The page must be found in the list, deleted and then moved to the front. This is very time consuming. This algorithm is also advantageous as it can never result in more than N-times more page faults than Optimal (OPT) algorithm. The LRU has a weakness in degenerated performance under common reference patterns. [1]



*Figure 2: LRU replacement example*

In figure 2, the first three rows are the page frames. A fault is noted by F. Note how when a new number is called, the least recently used is replaced such as the 2 for the 7. A hit first occurs with the 0 and 0 moves to the top of the order(not shown well) and takes three cycles to be eliminated. In summary this is the definition of an LRU replacement policy.

There are two methods for LRU replacement policies. The first method uses the equation N=log2(n!). For example if n=4 ways, $!=24, N=log2(24) => 5 bits to track the ordering in a state machine.

This is very complex with n! States. Another implementation is through the use of counters. A counter is associated with cache lines except when un-initialized, each counter is unique. In this case, a cache hit sets the counter of the corresponding line to 0. Increments all counters having a smaller value than the original value in the counter and counters with larger values remain unchanged. A cache miss the largest counter value is the victim with the counter reset to zero for a new line. All other counters are incremented. The counter method requires more bits but is simpler. For example, in an 8 way associative set, 2^3, and 3 bits x 8 ways = 3 bytes per set. In a 4 way associative set 2 bits per line 4 = 2^2 and 2 bits x 4 ways = 8 bits total or one byte. [4-Caches-with Notes Powerpoint, slide 29]

The LRU will be designed using the counter method.This method appears as follows for an 8 way set. Each way receives 3 separate bits and are associated with each tag.

MRU is 000 and LRU is 111

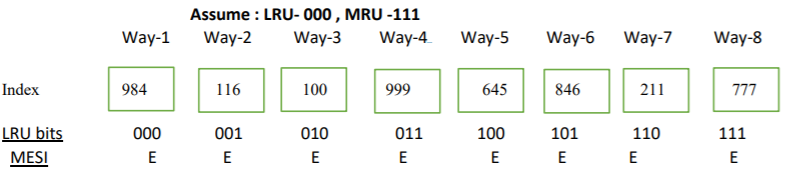


Figure 3: LRU Replacement usage.

MESI Protocol

The MESI protocol was specified for this project.

MESI Protocol is a cache coherence protocol. It is one of the most common protocols and supports write back, which is used in our project. Write back caches save a lot of bandwidth unlike the write through caches. Write back uses of a “dirty bit” which indicates that the data in cache is different from that of the main memory. The cache coherence protocol uses a coherence hardware called as the snooping hardware. MESI is an acronym, where each letter stands for the state that the cache line can be in.

Modified

* The cache copy is the only valid copy
* No other processors have the updated copy
* Main memory copy is out of date

Exclusive

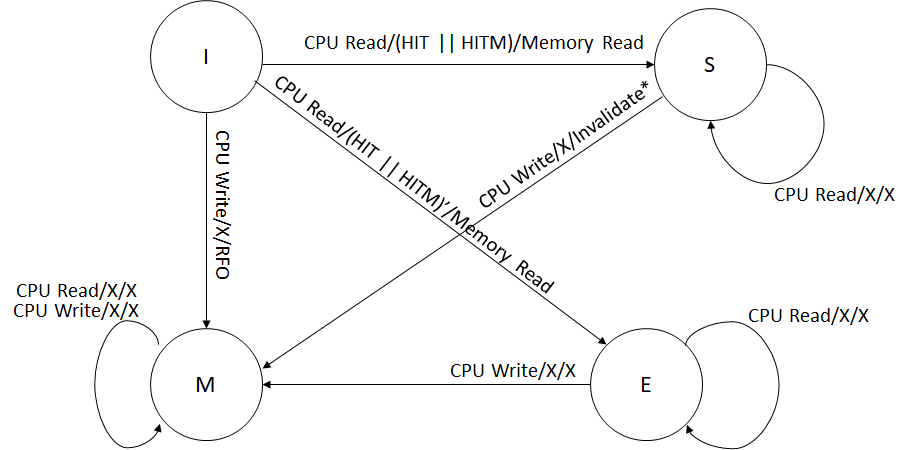
* No other processor has the cached copy
* The cache and the memory are identical

Shared

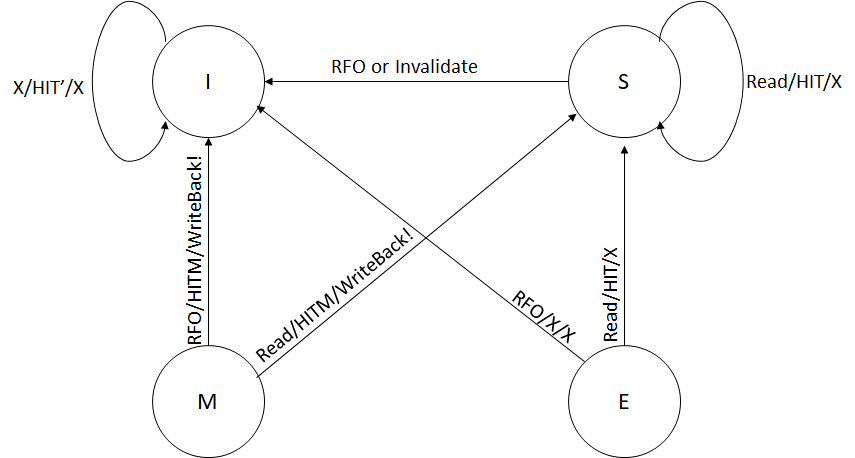
* At least one other processor has the cache copy
* The cache and the memory are identical

Invalid

* The cache entry is invalid



*Figure 4: State Transitions for Cache Controller of Processor Accessing Memory*



*Figure 5: State Transitions for Cache Controller of Snooping Processor*

Additional Mathematical Equations

Calculations for tag bits (for both Data and Instruction cache)

There are 32 address bits

There are 64 bytes per line.

64 = 2^6 ⇒ 6 Byte Select Bits

There are 16K sets

16K = 2^14 ⇒ 14 bits for Index Select

Therefore,

Tag bits = 32 - 6 - 14 = 12 bits

8 way Set - 2^3 => 8 && 3 bits \* 8 ways = 24 unique bits(Data Cache) \* 2^14 sets

4 way Set - 2^2 => 2 && 2 bits \* 4 ways = 8 unique bits(Instruction Cache) \* 2^14 set

# Design Decisions

Team had the most experience in Verilog for implementation therefore it was the chosen language for this project.

1. Mode and trace file are taken as arguments from compile command line
2. Each line of the trace file is read, then command and the address is determined. Address is broken down into byte select, index and tag bits.
3. The commands/operations are broken down into individual cases and executed.
4. The cache line status is determined by
   1. Checking if it is a hit
   2. If not hit, check if there are any invalid lines
   3. If any invalid lines exists, check for LRU and write accordingly
   4. If no invalid lines, perform evict operations
5. Most recently used is set as 000 and LRU is 111 as well as use the counter method.
6. A variable keeps track if the trace file is finished. If Finish flag is raised, the statistics are printed out.
7. The program displays cache contents only for valid sets.
8. During program initialization, all the lines are invalidated thus setting the lines to Invalid MESI state, tag bits are set to 0 and LRU bits are reset.

**Discussion and Results**

Debugging provided many insights by using the verilog $display() statements. Issues that occurred were incorrectly setting the quantity of bits such as 3 bits LRU implementation copied into the instruction cache which uses 2 bits. Another issue was printing invalid statements. To verify our results the team was debugging the MESI states for the scenario examples provided by Yuchen. We printed out over 5,000 lines as there were 16k sets resulting in an interrupt by the program. Other errors were subroutines within each MESI state. The MESI state diagram was debugged continuously until all state transitions were met.

# Validation Summary

The following requirements were originally listed and have been met.

**SYSTEM REQUIREMENTS:**

System shall use up to three other processors

System shall employ MESI protocol for cache coherence

L1 Cache

* + - Instruction cache - Four way set associative
      * 16k sets and 64 byte lines
    - Data cache - 8-way set associative
      * 16k sets and 64 byte lines
      * Is write back using write allocate
      * Is write back except for first write to line which is write through

Both caches employ LRU replacement policy

Both Caches are backed by shared L2 cache

Cache must be described and simulated in Verilog, C or C++.

* + - Verilog doesn’t need to be synthesizable, clock accurate and doesn’t need to store/retrieve data.

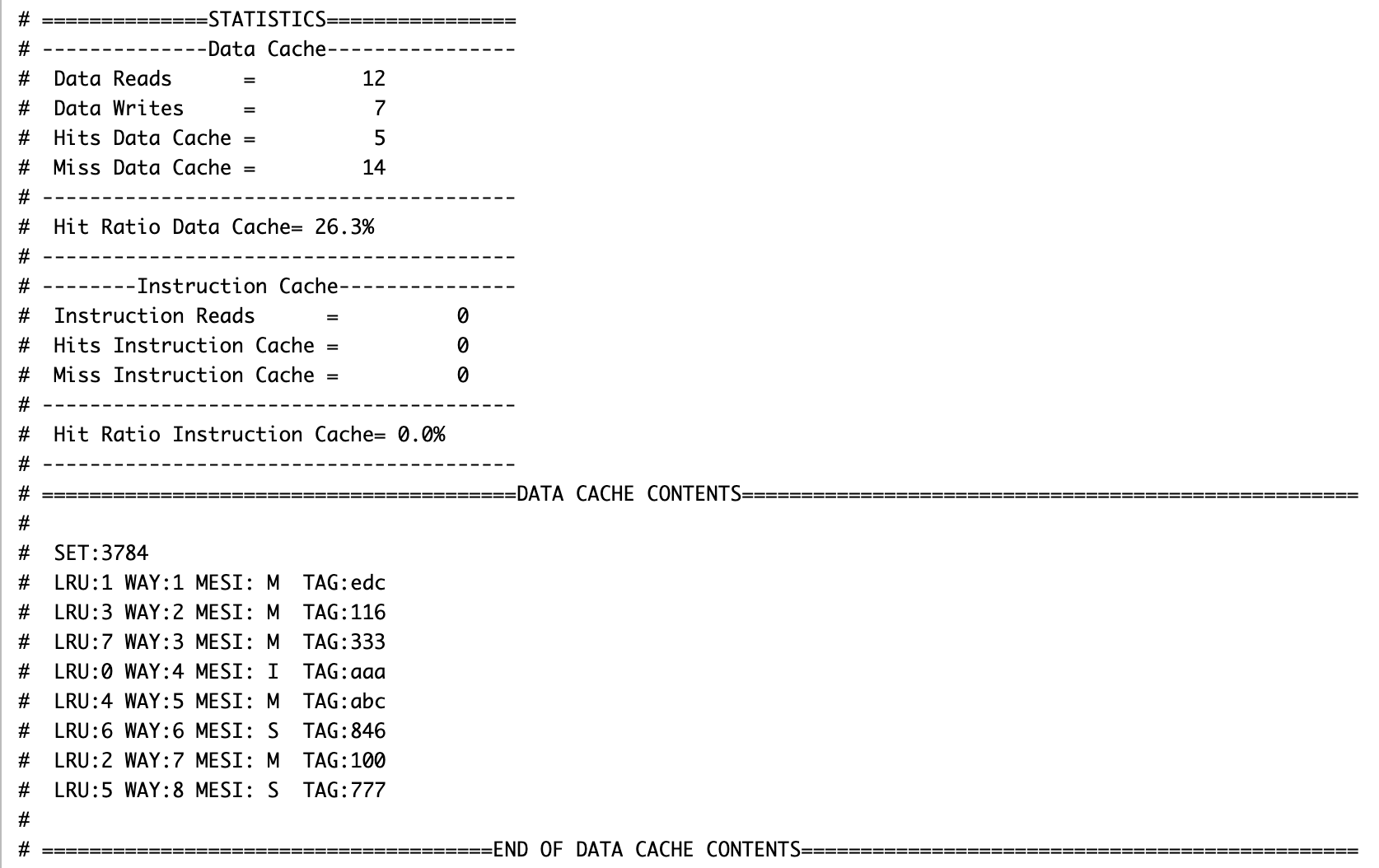
System Modes

* + - Mode 1 the simulation shall display only the required summary of usage statistics and responses to 9s in the trace file and nothing else.
    - Mode 2 shall display everything from mode 1 but also display communication messages to the L2 described above and nothing else.
    - Additional modes may be implemented but not present in the demo

**SYSTEM OUTPUTS:**

System shall maintain and report following key statistics of cache usage for each cache and display upon completion of the trace:

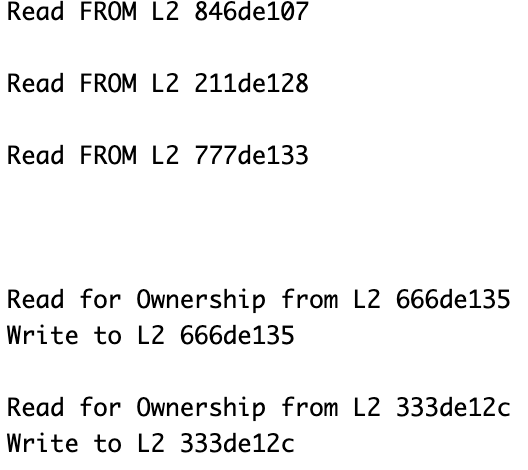
* + - # of cache reads
    - # of cache writes
    - # of cache hits
    - # of cache misses
    - #cache hit ratio



*Statistics generated in the project*

In order to maintain inclusivity and implement Mesi protocol, L1 caches may have to communicate with the shared L2 Cache, to simulate the system shall display the following messages:

* + - Return data to L2 <address>
      * In response to a 4 in trace file, cache shall signal that it’s returning the data for that line (if present and modified)
    - Write to L2 <address>
      * Operation is used to write back a modified line to the L2 upon eviction from L1 cache. Also used for initial write through when cache line is written for the first time so L2 knows its been modified and has the correct data
    - Read from L2 <address>
      * Operation is used to obtain data from L2 on an L1 cache miss
    - Read for ownership from L2 <address>
      * This operation used to obtain data from L2 on an L1 cache write miss



*Communication Messages generated in Mode 2*

**DOCUMENTATION**

Documented, described and simulated cache in Verilog(not clock accurate)

Validation Documentation

Specification Checklist

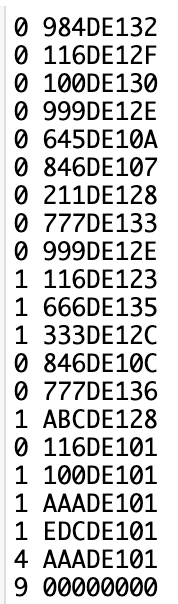
# 

# Appendix A: Source Code for cache and associative modules

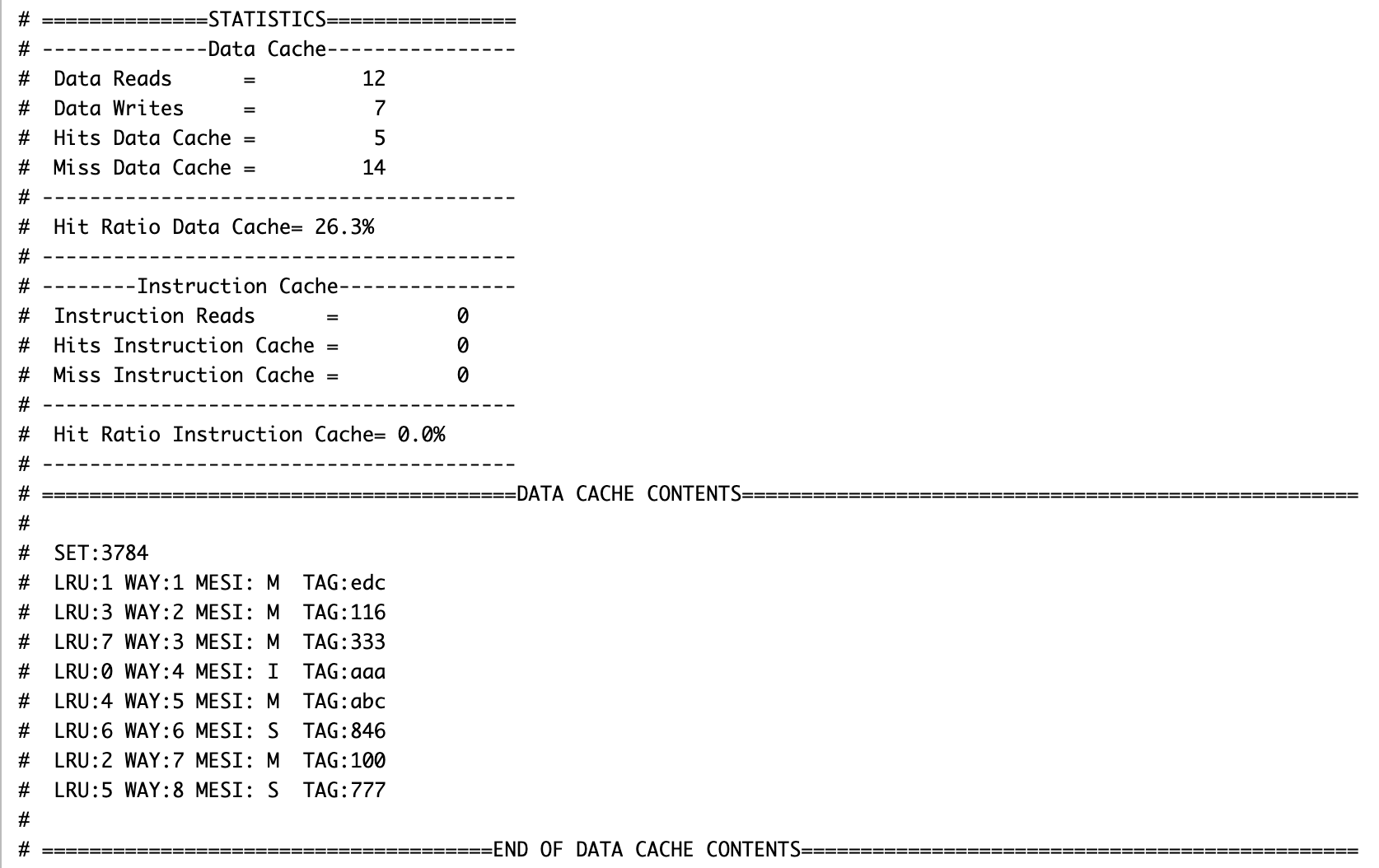
(see folder documents)

# Appendix B: Simulation results along with usage statistics

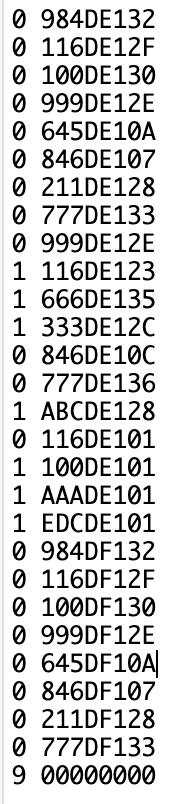
Trace 1: Operations on same set (trace given in the project explanation file)



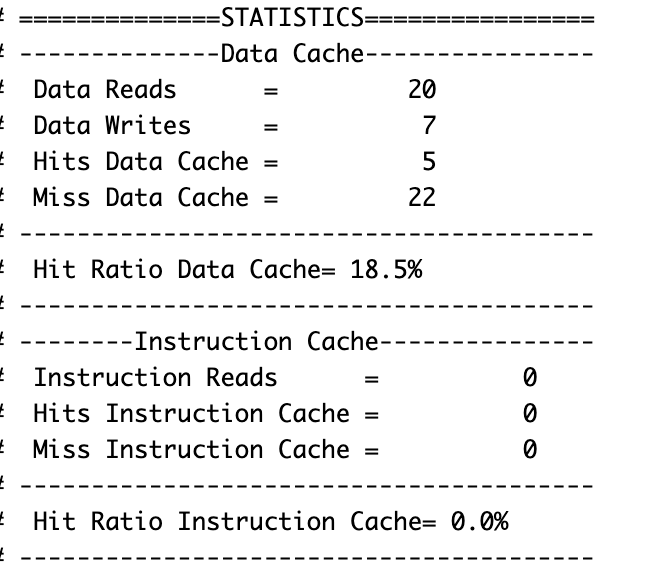
Results for trace 1:

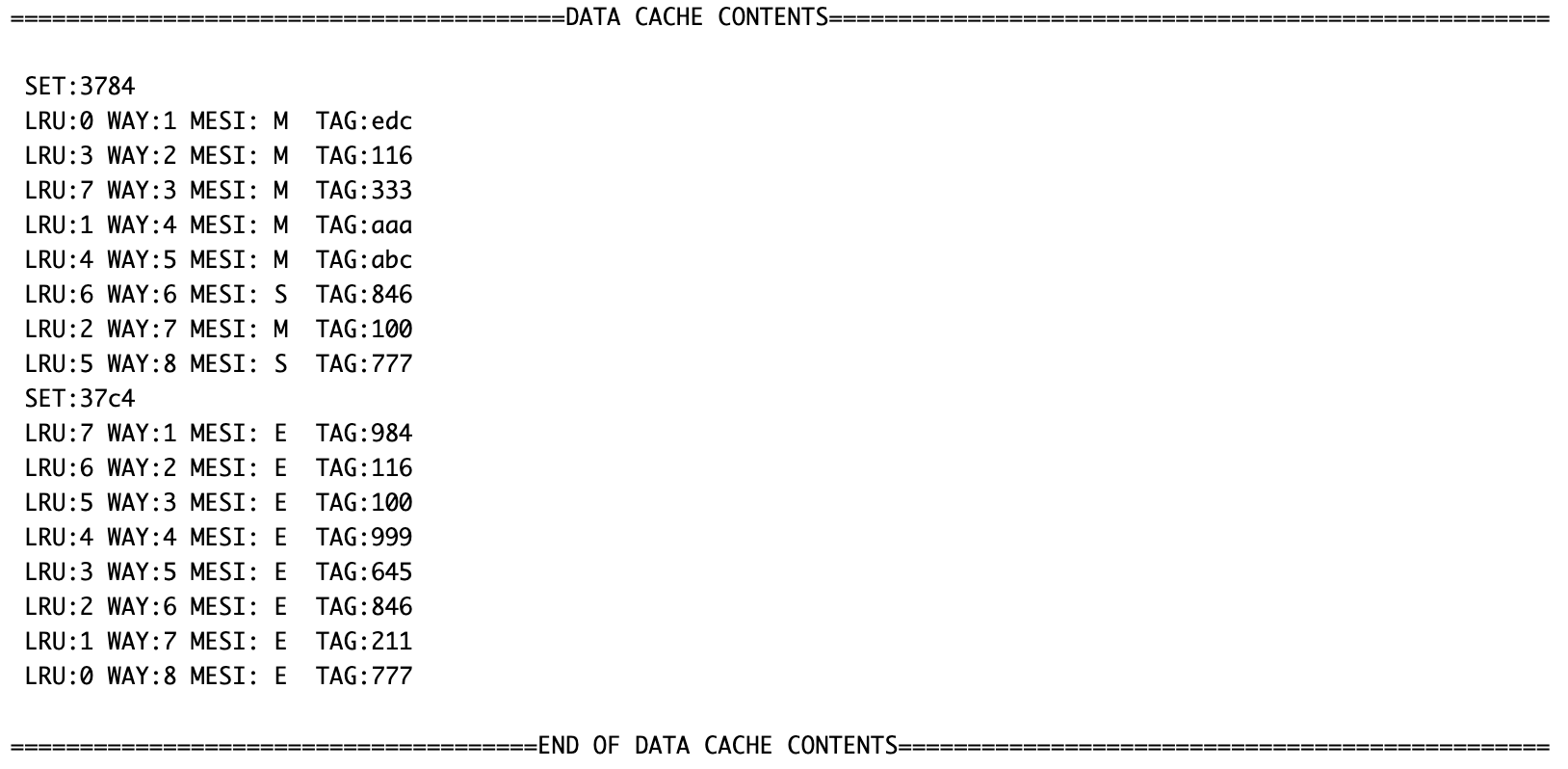


Trace 2: Operations on multiple sets

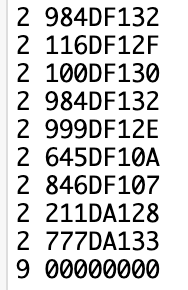


Results for trace 2:

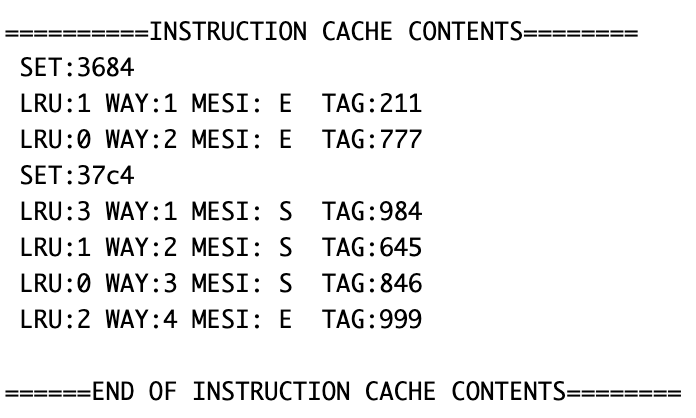
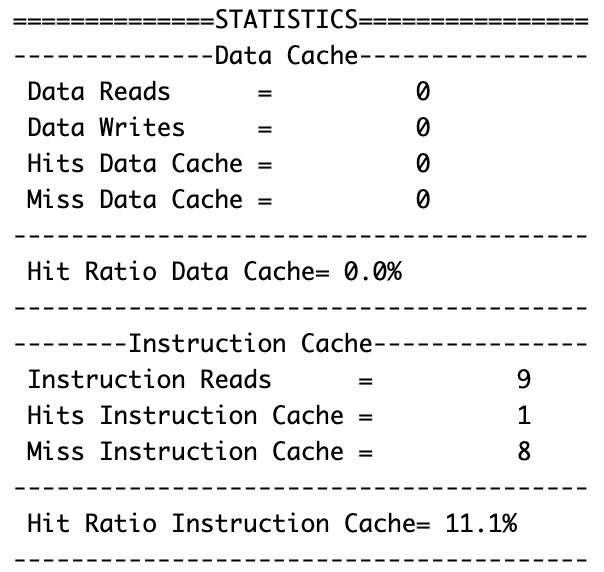




Trace 3: Instruction Operations on multiple sets



Result for Trace 3:



Link to Diagrams - State Diagram, Original Draft

# References

[1] Linked List Definition

<https://www.ques10.com/p/8619/explain-lru-page-replacement-policy-with-suitable-/>

[2] Cache Simulator – Used as a basis and modified after approval from Yuchen 3/7/2021

[Github.com](https://github.com/RahulMarathe94/L1-Cache-SImulator-using-MESI-Protocol)