****

**Cache Simulator**

By

Peter Kinsella

30/10/2019

This Report is submitted in partial fulfilment of the requirements of the Honours Degree in Electrical and Electronic Engineering (DT021) of the Technological University Dublin

Contents

[Introduction 2](#_Toc24354166)

[Objective 2](#_Toc24354167)

[Theory 2](#_Toc24354168)

[Associative Memory 2](#_Toc24354169)

[Direct Mapped 3](#_Toc24354170)

[2-way Set Associative 4](#_Toc24354171)

[Code 5](#_Toc24354172)

[Associative Memory 5](#_Toc24354173)

[Direct Mapped 6](#_Toc24354174)

[2-way Set Associative 7](#_Toc24354175)

[Results 8](#_Toc24354176)

[Associative Memory 8](#_Toc24354177)

[Direct Mapped 9](#_Toc24354178)

[2-way Set Associative 9](#_Toc24354179)

[Analysis 9](#_Toc24354180)

[Comments and Conclusion 10](#_Toc24354181)

[Bibliography 11](#_Toc24354182)

[Appendix A: Code 12](#_Toc24354183)

List of figures

[Figure 1 Associative memory cache design [1] 2](#_Toc24354184)

[Figure 2 Direct mapped cache design [2] 3](#_Toc24354185)

[Figure 3 2-way set associative cache design [3] 4](#_Toc24354186)

[Figure 4 Struct to represent SRAM 5](#_Toc24354187)

[Figure 5 operation of simulation 5](#_Toc24354188)

[Figure 6 Struct to represent SRAM 6](#_Toc24354189)

[Figure 7 Operation of the Direct Mapped simulator 6](#_Toc24354190)

[Figure 8 Structs to represent the two SRAM's for addresses 7](#_Toc24354191)

[Figure 9 Operation of the 2-way Set Associative simulator 7](#_Toc24354192)

[Figure 10 Expected hits and misses for each simulation 8](#_Toc24354193)

[Figure 11 Associative memory results 8](#_Toc24354194)

[Figure 12 Direct mapped cache design results 9](#_Toc24354195)

[Figure 13 2-way set associative cache design results 9](#_Toc24354196)

# Introduction

## Objective

The aim of this assignment is to simulate the function of three different cpu cache storage systems, associative memory, direct mapped and 2-way set associative.

## Theory

### Associative Memory

Associative memory is the simplest cache storage system of the three that will be discussed. The main concept behind this system is that a series of 32 bit registers are used to store addresses which have associated 32 bit registers used to store data. When an address comes in, before it goes to the DRAM to get the information it is compared to the stored addresses in the associative memory cache. A series of comparators are used to compare the address to the individually stored addresses in the cache. If a match is made, a hit, then the data is pulled from the associated data register. If no match occurs, a miss, then a series of things occur. Instead of getting the data from the associated registers the data is gotten from the DRAM. The address is also saved in a new index in a 32 bit address register and its associated data register is updated to contain the data gathered from the DRAM. This method allows information to be gathered quicker if it has already been gathered before. This is because the data registers are in a SRAM cache which is smaller but much quicker than the DRAM cache and the DRAM cache is only accessed when an address hasn’t been seen before. The main downfall to associative memory tough is the fact that for the system to be efficient the SRAM cache would have to be fairly large. Therefore, for a modern computer system this system just isn’t viable as it would take up too much space with the large cache and the large amount of comparators to go with each register.

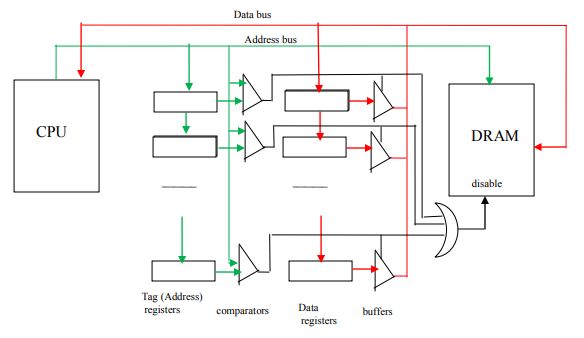


Figure 1 Associative memory cache design [1]

### Direct Mapped

Direct mapped cache is a system that fixes the sizing issue of the associative memory. In Direct mapped two separate SRAM caches are used to store addresses and data and uses a single comparator to compare the stored addresses to the ones coming in. This system works by the incoming address being split into three parts. An upper half (tag), a lower half (set) and a byte number. The system works by using the set of the address as an index for the first cache. Each 32 bit register is split up into 4 bytes within the cache and the byte number is used as an index for which byte is being pointed at in that 32 bit register. Within each byte the tag of an address is stored. This way when an address comes in and is split the set is used to locate which tag is stored in a specific byte. That tag is then compared to the tag of the incoming address. If they match then the same set and byte number are used to access a second cache where a corresponding piece of data is stored. If the two tags do not match then the DRAM is accessed and the data is collected from there. That data is then stored in the second cache using the set and byte number so it can be accessed at a later point. The tag of the address is also then stored in the first cache so that the system can get a hit in the future. This system is not without its flaws though. The main flaw with this system is that multiple addresses can have the same set and byte number while having a different tag. This causes issues as when that new address comes in and sees that the tags do not match, it will overwrite the old tag with the tag of the new address at the same point in the first cache. Thus meaning that when the first address comes back in, the system will get a miss as the tag is no longer being stored.

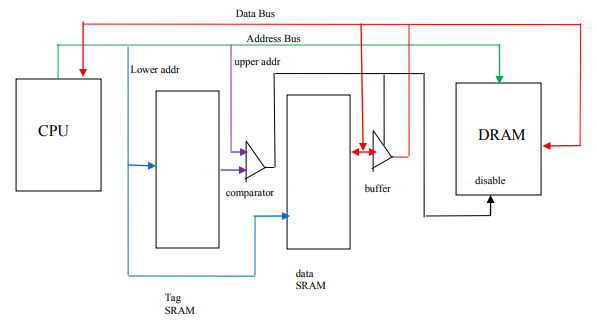


Figure 2 Direct mapped cache design [2]

### 2-way Set Associative

2-way set associative system uses the same method as direct mapped but solves the issue of over writing addresses to a certain extent. The difference between the two systems is that 2-way set associative uses 2 address caches and 2 data caches instead of one of each. This allows for double the amount of addresses as well as allowing for 2 addresses to be stored that have the same set and byte number but different tags. The way this system works is by adding in a least recently used (lru) flag. This flag indicates which cache was overwritten last so that the system knows to write to the other one rather than continually writing over the same caches register. Other than this difference the principle operations of the two system are identical. Though this method does only allow for two addresses with the same set and byte number. Thankfully the system can be scaled indefinitely to fit however many addresses need to be allowed for in a system. At that point the decision comes down to a size restriction and how big you want the storage to be for the amount of addresses you want to be able to store at a given time.

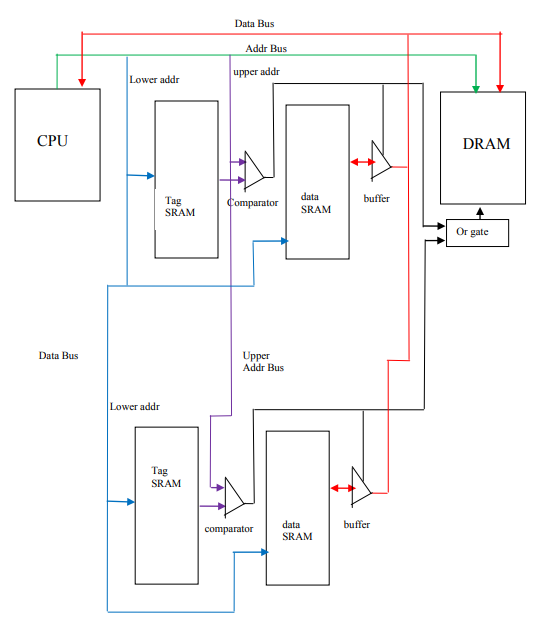


Figure 3 2-way set associative cache design [3]

# Code

## Associative Memory

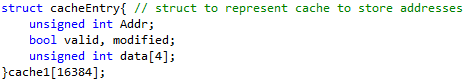


Figure 4 Struct to represent SRAM

The above piece of code is what was used in the simulation to represent the SRAM cache. The struct was used as it allows one to have multiple pieces of data to have the same variable name while differing by their location in the struct array. This represents all the different registers in the SRAM cache that will hold similar information such as addresses and data. In the associative memory design, it specifically holds the full address as it is what is stored and compared to the incoming address in the simulation.

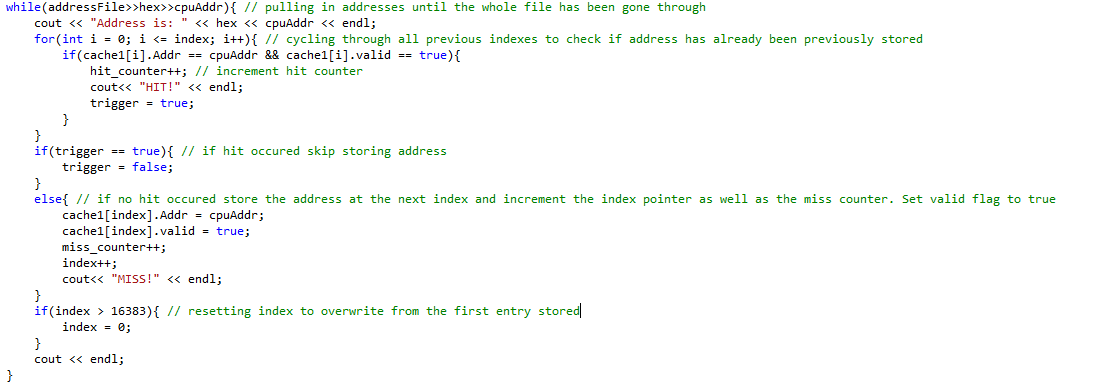


Figure 5 operation of simulation

The above piece of code is the operation of the simulator. The process that the code goes through is done in a single while loop. This while loop pulls in addresses from a text file, one by one, that are then run through the simulator. This while loop lasts until there are no more addresses to pull in and then the number of hits and misses are printed. For a hit to occur the addresses pulled in is checked against every stored address in the cache struct. If there is a match and its valid flag is set to true then a hit is printed and the hit counter is incremented. If no match occurs then the pulled in address is stored in the next indexed struct and the miss counter is incremented. The index is then incremented. When the index reaches a point of being greater than the size of the struct array, it is then reset so that if a new address comes in it will overwrite the oldest address first.

## Direct Mapped

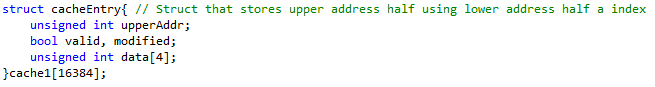


Figure 6 Struct to represent SRAM

In Direct mapped the struct is laid out very similarly to that of the associative memory. The only difference is that in direct mapped only the upper half of the address, the tag, is being stored. Therefore, a variable name change needed to be made.

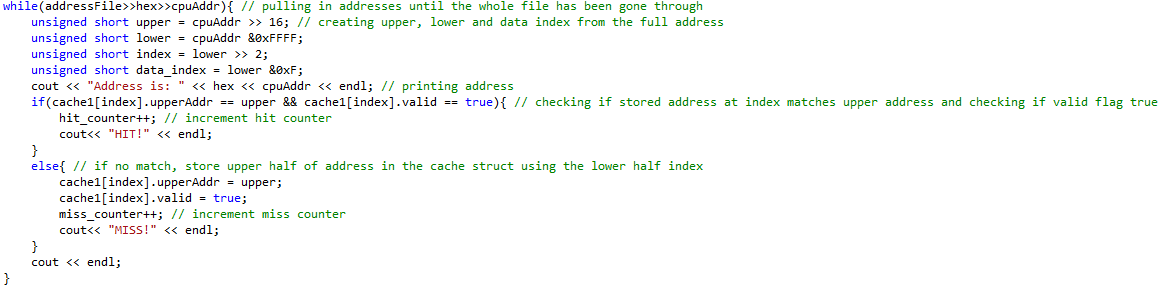


Figure 7 Operation of the Direct Mapped simulator

The operation is quite different though. In the operation for direct mapped, the while loop stays the same but the address is then broken up into different parts. The first 16 bits of the address are known as the tag and are what get stored in the cache struct. To get the top 16 bits the address is shifted to the right 16 bits so that only the top 16 bits are valuable data. To get the lower half of the address the full address is ANDed with 16 1 bits. This makes the top bits go to 0. This shifting and ANDing is then also done to split the lower half into a top 14 bits, set, and a lower 2 bits, byte number. The set is then used as an index to check the cache struct for a stored tag and to see if it matches the incoming tag. If they do match and the valid flag is set to true then a hit occurs and the counter is incremented. If not the tag is then stored at that set index for later checks.

## 2-way Set Associative

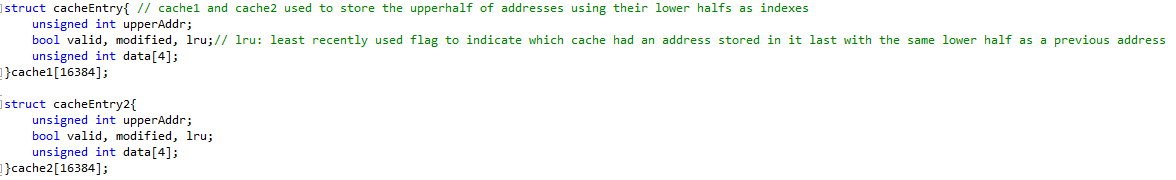


Figure 8 Structs to represent the two SRAM's for addresses

The two way set associative cache struct is the same as the direct mapped struct however it has one minor change. A least recently used flag, lru. There is also two structs as there are two address SRAMs in the real structure.

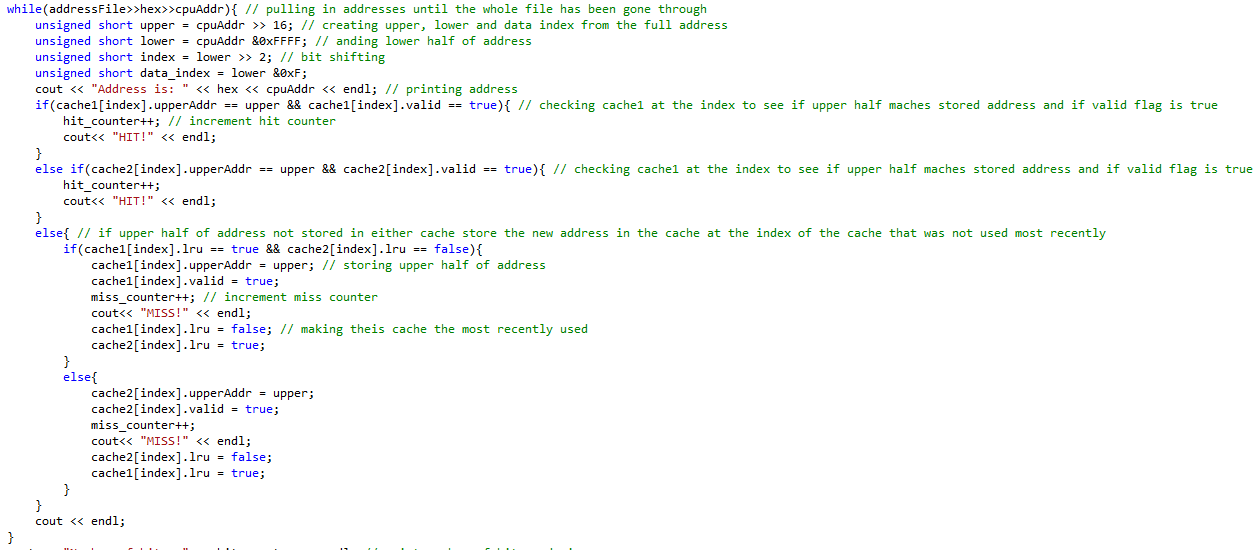


Figure 9 Operation of the 2-way Set Associative simulator

The operation of the 2-way set associative simulator is also very similar to the direct mapped. It uses the same system of checking the tag using the same set index, however is does this check for both structs to see if either have the address stored. Then when a miss occurs, when the address is being stored the lru flag is checked in each of the structs to see which struct was overwritten last. This is used so that the same struct isn’t continually overwritten. Then the lru flags are toggled.

# Results

The following image shows when a hit or a miss should occur when the addresses given are fed into each of the different simulations. For the created simulations to work correctly they should get the exact same result and should be able to handle an infinite amount more addresses coming in afterwards.

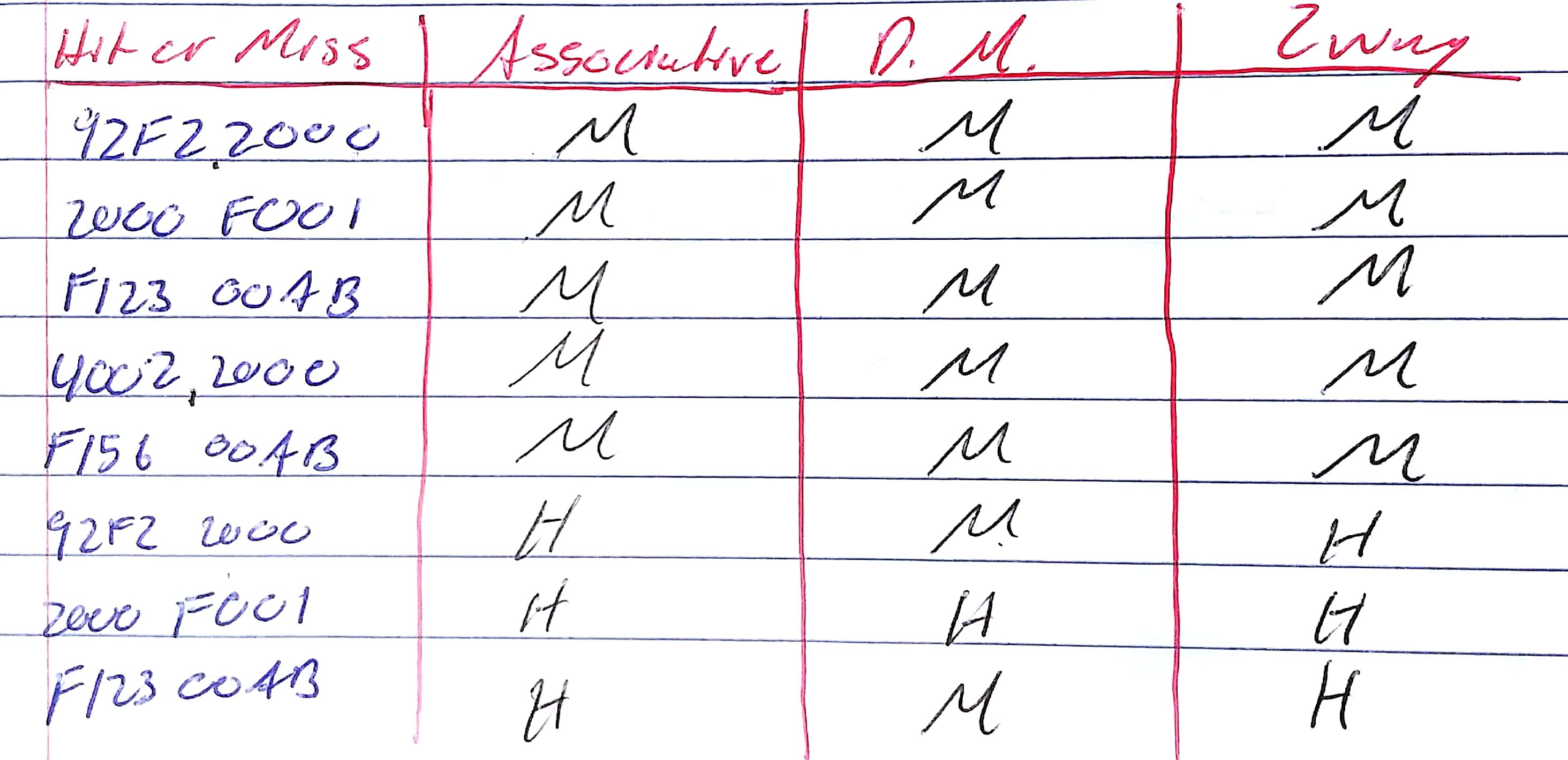


Figure 10 Expected hits and misses for each simulation

## Associative Memory

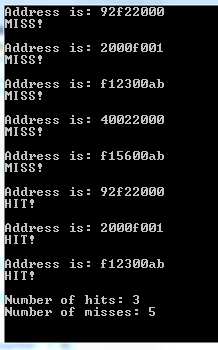


Figure 11 Associative memory results

## Direct Mapped

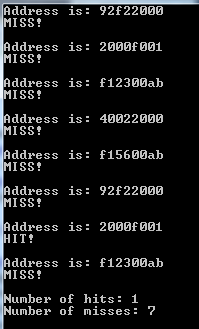


Figure 12 Direct mapped cache design results

## 2-way Set Associative



Figure 13 2-way set associative cache design results

## Analysis

From the above results it can be seen what addresses were sent in and when a miss or hit occurred for the particular address based on which simulator was being used. Based on the results it can be seen that associative and 2-way set associative have the best output and number of hits. The results also match those of the expected results, which shows that the simulators are performing as they should be.

# Comments and Conclusion

In conclusion, the different simulators while having different operations have different advantages and disadvantages. The associative being that it can store all addresses but that to do so the SRAM must be very large. The direct mapped fixes this size issue, however, it cants store the same amount of addresses and if two addresses have the same lower half then the first one gets overwritten. 2-way set associative fixes this issue, can store two addresses with the same lower half, and is scalable for more addresses. When it comes to deciding which method to use the benefits and downfalls have to be weighed to determine which is the optimal solution for the task at hand.

# Bibliography

* [1] – Lynch, Raymond “Associative Cache” [Online] Available: <http://raylynch.dx.am/FT2814/associative%20cache.pdf> [Accessed: 29th October 2019].
* [2] – Lynch, Raymond “Direct Mapped Cache” [Online] Available: <http://raylynch.dx.am/FT2814/direct%20mapped%20cache.pdf> [Accessed: 29th October 2019].
* [3] – Lynch, Raymond “2 Way Set Associative Cache” [Online] Available: <http://raylynch.dx.am/FT2814/2%20way%20set%20associative%20cache.pdf> [Accessed: 29th October 2019].

# Appendix A: Code

