CSCE 230301 - Computer Organization and Assembly Language Programming

Summer 2022

Cache Simulator Project

Names: Yehia Zakaria Elkasas, Ali Hussein Yassine, Rana El Gahawy

IDs: 900202395, 900204483,900202822

Table of Contents

Introduction

In this project, we studied the effect of cache parameters, cache line size and number of ways, on the hit and miss rates. We had 6 memory generators, apart from the function, where we entered the addresses manually to validate the functionality of the code. After checking the validation, we used each memory generation to perform 2 experiments; one where the number of ways is constant (4) and the cache line size is variable (16,32,64,128), and the other was fixing the cache line size (16) and the number of ways was changing (1,2,4,8). After each experiment, we derived a conclusion whether the independent variable was affecting the hit and miss rates or not.

Implementation

The cache is constructed as follows; a vector of vector of cache lines/blocks, and the cache line is a struct that has 2 member variables (boolean valid bit and unsigned integer tag).

Main Function

The cache is resized according to the number of sets, and the cache line is resized according to the number of ways. For 1 million times, a memory generator returns a 32 bit address that will be passed to the cache simulator function. This function returns whether this address resulted in a hit or a miss, and both are saved in counters.
There are options for validation, as for each iteration, the address and hit/miss. For further debugging, there is an option to print the whole cache after the iterations.
Finally, the hit and miss percentages are output to the user.

Cache Simulator Function

The tag and the index are extracted from the address. We go to the index set and check in each way whether the valid bit is 1 and the tag is the same. If so, we return a hit. Otherwise, if there is a block in the set with valid bit 0, we update it with the new tag, but if there is no capacity (the whole set is full), we randomly replace one of the blocks with the new tag and finally return a miss.

Data Analysis

MemGen1()

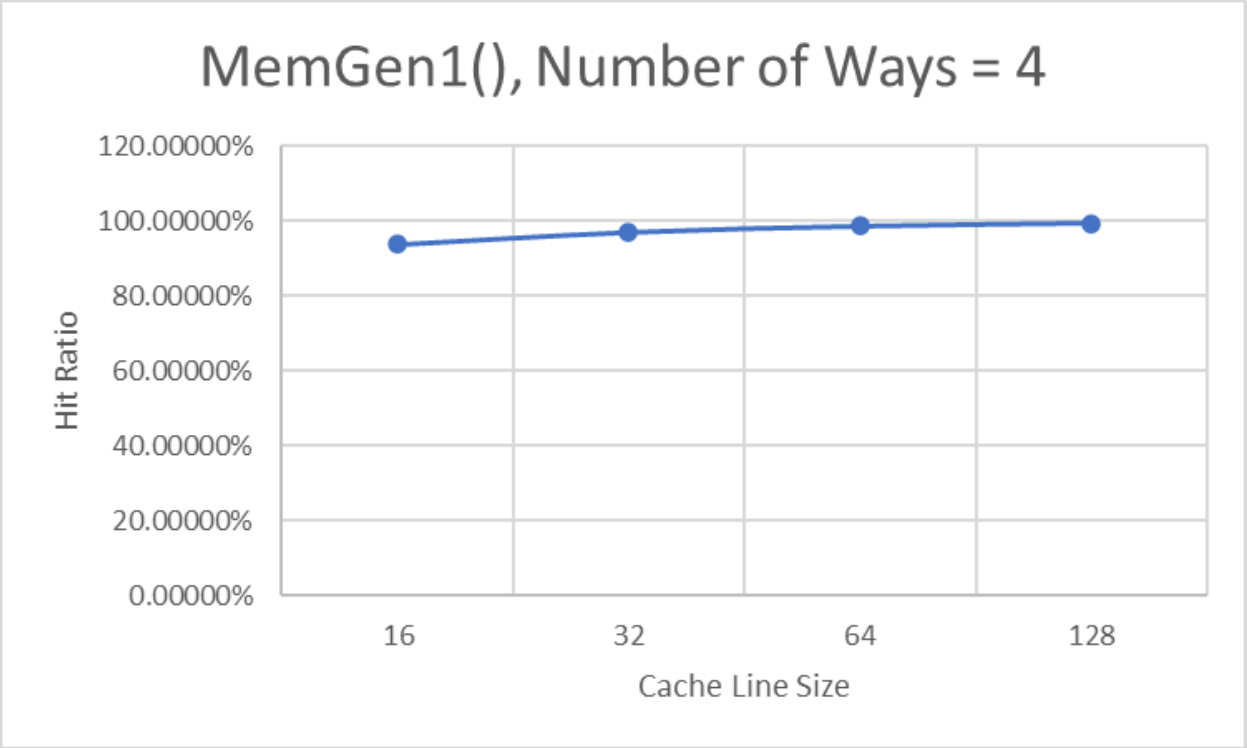


Figure 1.MemGen1() hit ratio experiment 1

As per the title of the graph, we fixed the number of ways to be 4, and the cache line size was the independent variable (16,32,64,128). MemGen1() is a function that returns an address sequential everytime calles (1,2,3,...etc). As we can see, the hit rate increases slowly, as we increase the cache line size. However, the rate of increase slows down as we increase the size.
Analysis:
To analyze this increase, as the line size increases, it can store more data. To elaborate, the space locality concept is used, as more data is stored from the main memory every time an address is accessed and returns a miss, and more words are stored in each block, according to the cache size, so the probability of accessing data is higher. Therefore, hit rate increases, as we increase the cache line size. As a result, we can conclude that cache line size slightly increases the hit ratio.

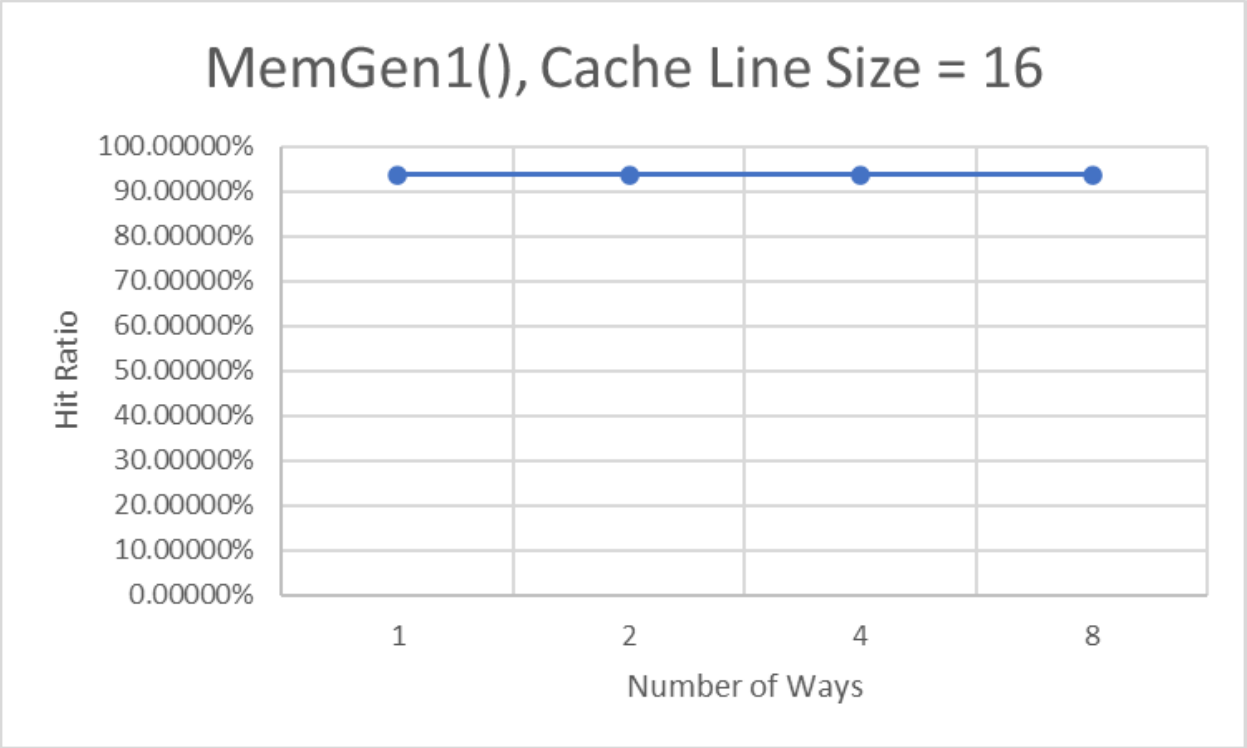


Figure 2.MemGen1() hit ratio experiment 2

In this experiment, we fixed the cache line size to be 16, and varied the number of ways (1,2,4,8) to check how changing the number of ways will affect the hit rate. MemGen1() is a function that returns an address sequential everytime calles (1,2,3,...etc).
Analysis:
As we increase the number of ways, the number of sets decrease, but that will not affect the number of misses, as different tags will result in misses, so in every number of ways, there will be the same misses due different tags. As long as the cache line size is the same, the same number of words/amount of data will be stored, and this is what affects the hit rate. Therefore, the same hit rate, which is 93.75%. To conclude, changing the number of ways does not affect the hit or miss rate.

MemGen2()

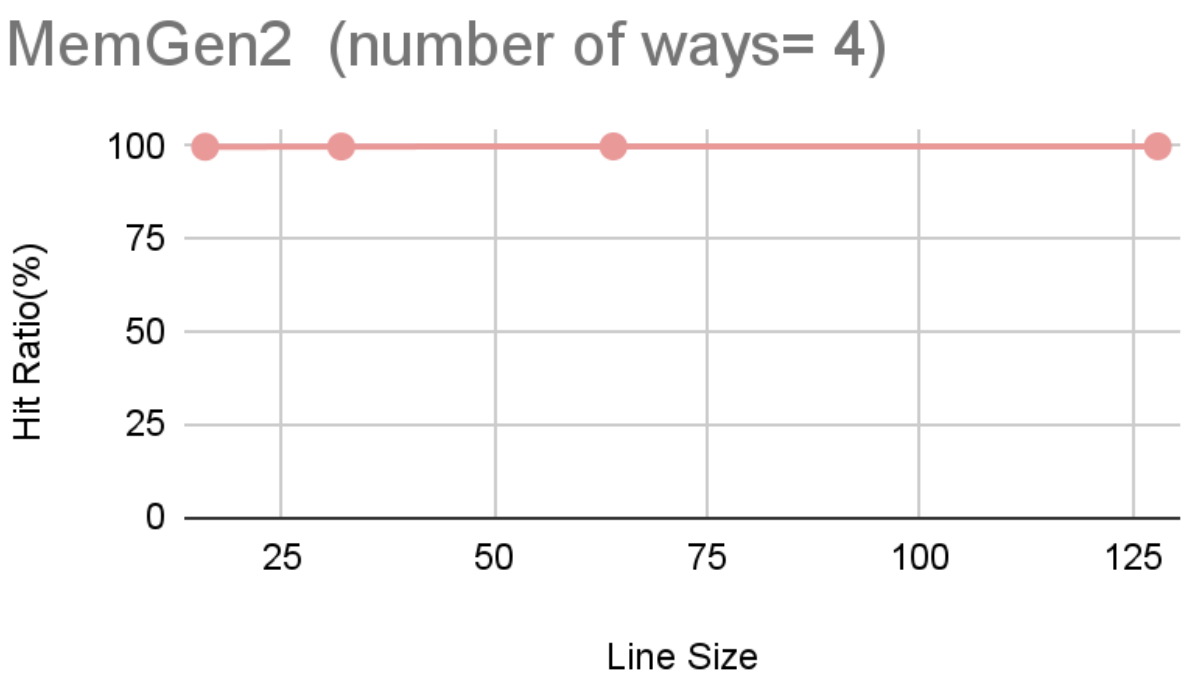


Figure 3.MemGen2() hit ratio experiment 1

Title: This graph represents the variation of the hit ratio in the set associative cache of fixed cache size (64KB) and fixed number of ways (4) as line size changes from 16 to 32 to 64 then 128.

Analysis:

The results shown make sense since by increasing the line size we are making better use of the spatial locality, thus each line now contains more data and this leads to the possibility of more hits.Thus, hit ratio increases.

In this experiment the hit ratio increases as the line size increases.Despite the fact that the percentage in the hit ratio seems to be increasing slowly, yet this is only because the number of iterations is huge (1 million iterations) so a 0.5% increase for example is a lot.

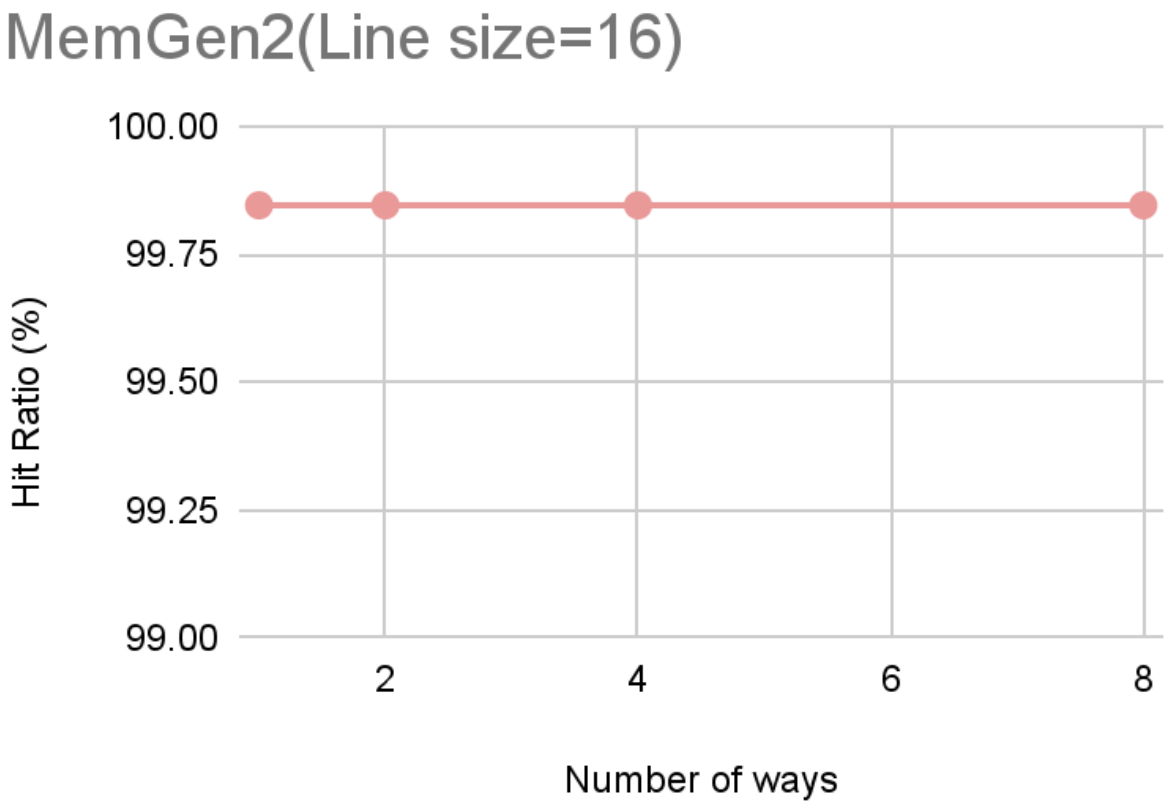
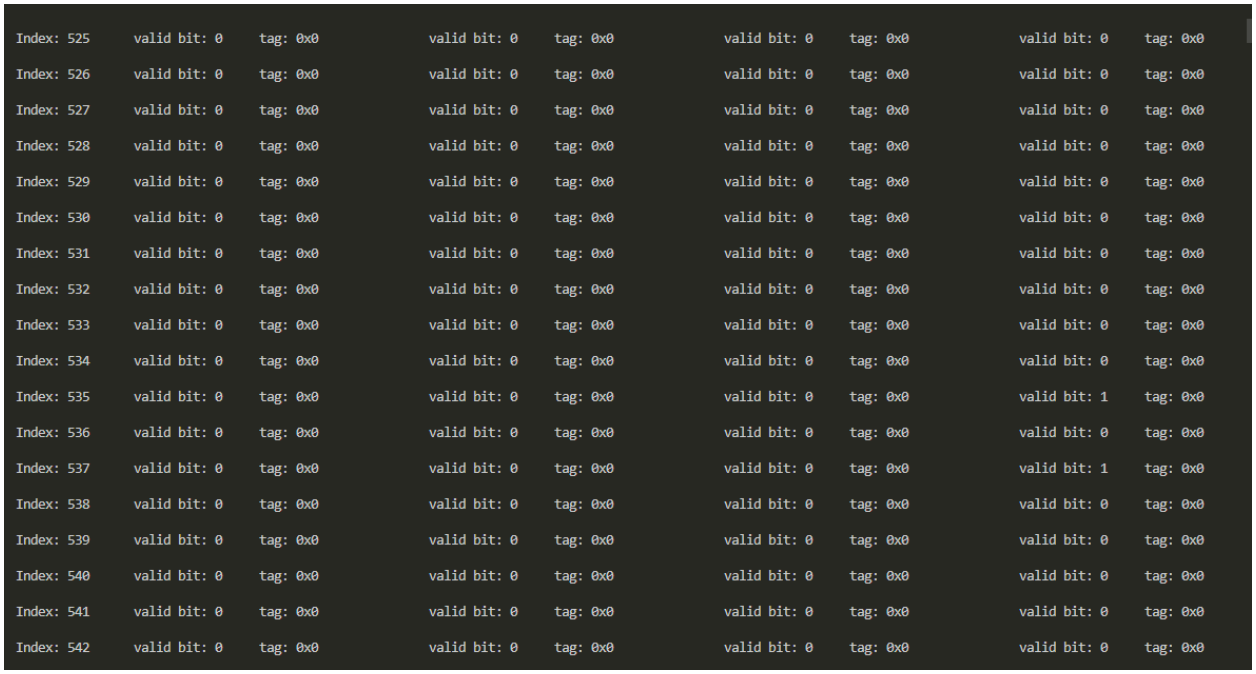


Figure 4.MemGen2() hit ratio experiment 2

Title: This graph represents the variation of the hit ratio in the set associative cache of fixed cache size (64KB) and fixed line size (16) as number of ways changes from 1 to 2 to 4 then 8.



Index: 525	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0
Index: 526	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0
Index: 527	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0
Index: 528	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0
Index: 529	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0
Index: 530	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0
Index: 531	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0
Index: 532	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0
Index: 533	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0
Index: 534	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0
Index: 535	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 1	tag: 0x0
Index: 536	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0
Index: 537	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 1	tag: 0x0
Index: 538	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0
Index: 539	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0
Index: 540	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0
Index: 541	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0
Index: 542	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0	valid bit: 0	tag: 0x0

Pic 1.Screenshot of the cache for MemGen2() in exp 2

Analysis:
This experiment shows constant values despite the change in the number of ways.
At first, this seemed odd because increasing the number of ways means that there are more lines per set so we can cache more memory into the same index which should result in higher hit ratio.
Why?

A logical explanation is that this memory generator does not make use of the number of ways no matter what this number is.
As shown in Figure 5 above (4 way cache) all the tags are zero and the same applies to the 1-way cache, 2-way cache and 8-way cache. This is clear proof that in each set only one block is used so tags will never mismatch and jump to the next block in the set.
In other words, this memory generator results in addresses that will not utilize having more than one line per set.

Memgen3()

In this section we collected the data from the function memgen3 and generated two different experiments where the variable was once the number of ways and in the other it was the cache line size.

First:

In this experiment we fixed the number of ways to 4, hence we have a 4-way associative set, then we varied the line/block size: 16, 32, 64, 128 to test the functionality of the simulator. We used two different scales a very small one to express the insignificant changes and another one to express the significant ones:

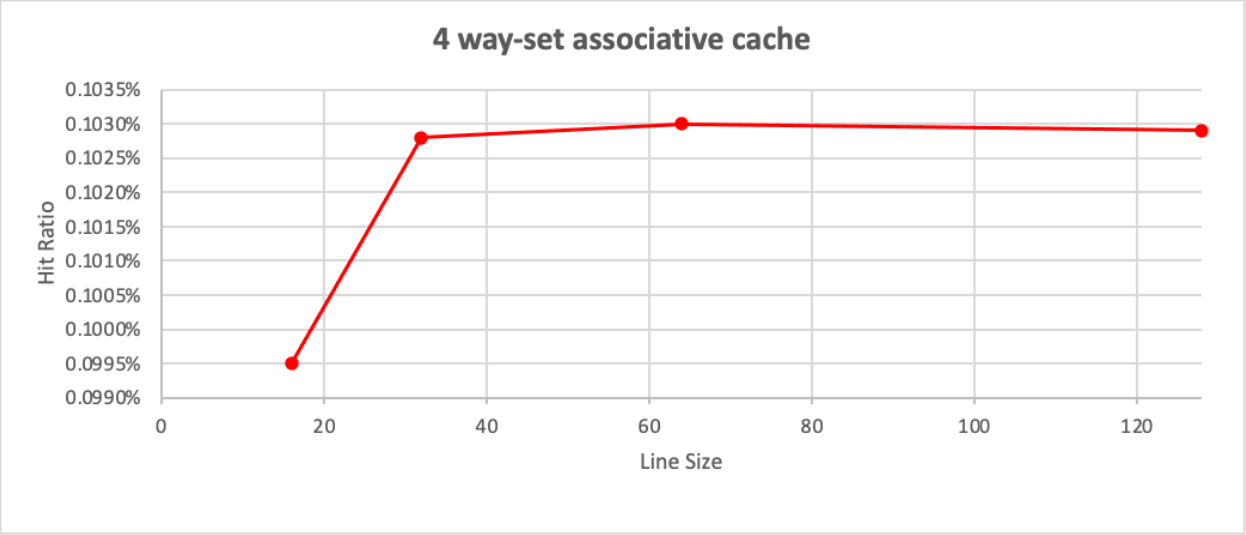


Figure 5.MemGen3() hit ratio experiment 1

In a small scale it is obvious from the graph that the hit ratio by increasing the number of bytes per line as it makes use of the spatial locality principle and stores more addresses per line so that possibility of having a nearby address gives a hit. On the other hand, if we would find that there is a slight decrease in the last point on the graph as when the cache line is 128 bytes the hit ratio decreases by 0.0001% which shouldn't be the case. When we tested it a couple of times it actually kept getting different values which were all very close to each another, yet there was still some differences.

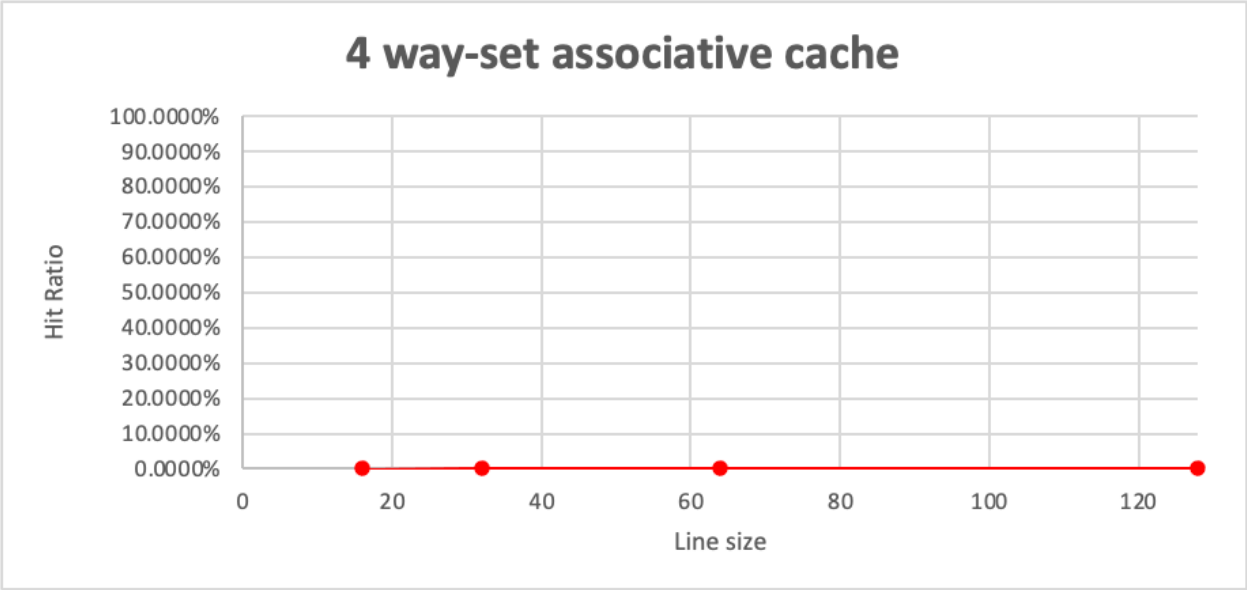


Figure 6.MemGen3() hit ratio experiment 1

One a larger scale the graph seems constant on zero. This can be justified if we put into consideration that the simulator replaces a random line in the set whenever it is full and as the function memgen3 gives random numbers there won't be a certain pattern that can detect which lines in which sets are replaced each run as it does not utilize the spatial and temporal locality principle. In addition the DRAM size is 64 MB and the cache is 64 KB, consequently the possibility of finding a random number from the DRAM that exists in the cache is 0.097% which is very small and nearly zero. The cache size and the DRAM size are both constant, hence the possibility will remain the same for the different line sizes so it is logical for the graph to be constant at zero.

Second:

This experiment fixes the line size to 16 bytes and changes the number of ways to 1, 2, 4, 8 to test the behavior of the simulator among the function memgen3 that generates random numbers.

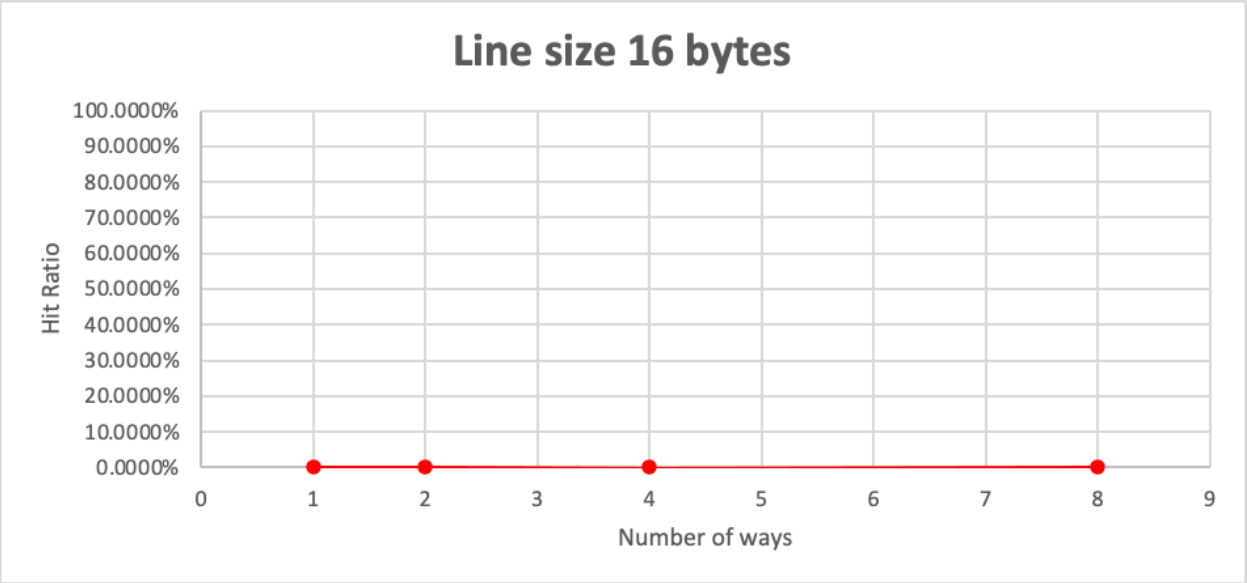


Figure 7.MemGen3() hit ratio experiment 2

The data collected from this function (memgen3) shows a slight insignificant increase in the cache hit ratio when increasing the number of ways from 1 to two, yet it is followed by a slight decrease of 0.0026% when we increase the number of sets to 4. This can be only justified by the fact that the function memgen 3 gives random numbers that do not follow a certain pattern and the lines are replaced randomly in the set each time there is a miss, and as the DRAM and cache sizes are constant across the different number of ways, so by increasing the number of ways we are decreasing the number of sets, consequently the increase in the hit ratio resulting from increasing the number of ways is associated with a decrease in the hit ratio resulting from decreasing the number of the sets so it is justifiable that the line in the graph is constant. The value of this line is zero because as explained in experiment one the possibility of finding a random number is 0.097% which is really small and nearly zero.

Memgen4()

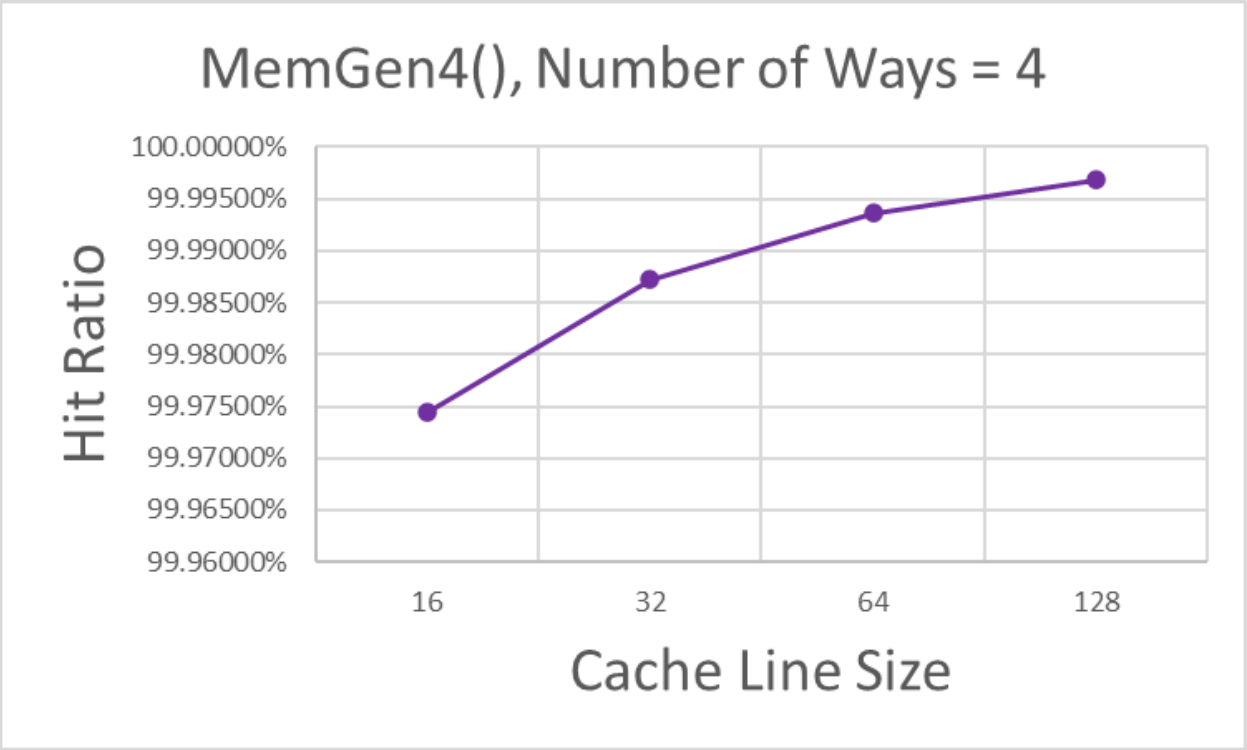


Figure 8.MemGen4() hit ratio experiment 1

In this experiment, we fixed the number of ways (4), and we had the cache line size as the independent variable (16,32,64,128). Memgen4() is a similar function to Memgen1(), as both functions return the addresses sequentially during the iterations, so that is why the pattern here looks so much similar to that of Memgen1().
Analysis:
The hit ratio is increasing while increasing the line size, but the rate of increase slows down as the cache line size because it makes use of the space locality concept and stores more data in each block to increase the hit ratio. Therefore, as we see a similar pattern on a different memory generation function, we can use that as more evidence of cache line size increasing the hit ratio.

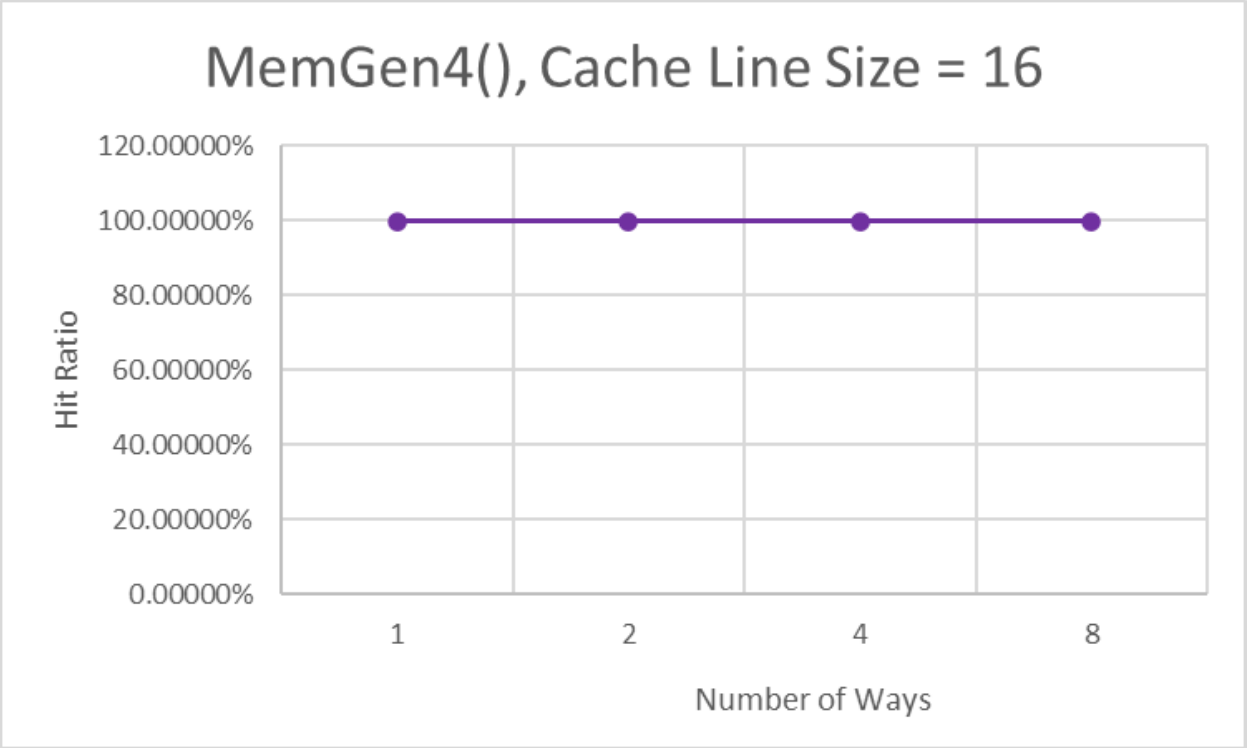


Figure 9.MemGen4() hit ratio experiment 2

In this experiment, we fixed the cache line size (16) and changed the number of ways (1,2,4,8).
Analysis:
We can see that the hit ratio is almost 100%, as Memgen4() has a maximum address of 4096 which has a tag of 0. During the first iteration of each set, there will be a miss in the first place, but since the memory generation is a sequential loop, it will be recorded as hits for the rest of iterations. The biggest address will be 4096, which has 256 as an index. Therefore, there will be 256 total misses, and the rest of the 1 million iterations will be hits. In this experiment, increasing the number of ways will not change anything because the addresses have the same tags, and the number of sets will decrease, therefore the same 256 indices changes will result in the same misses and output the same hit rate.

Memgen5()

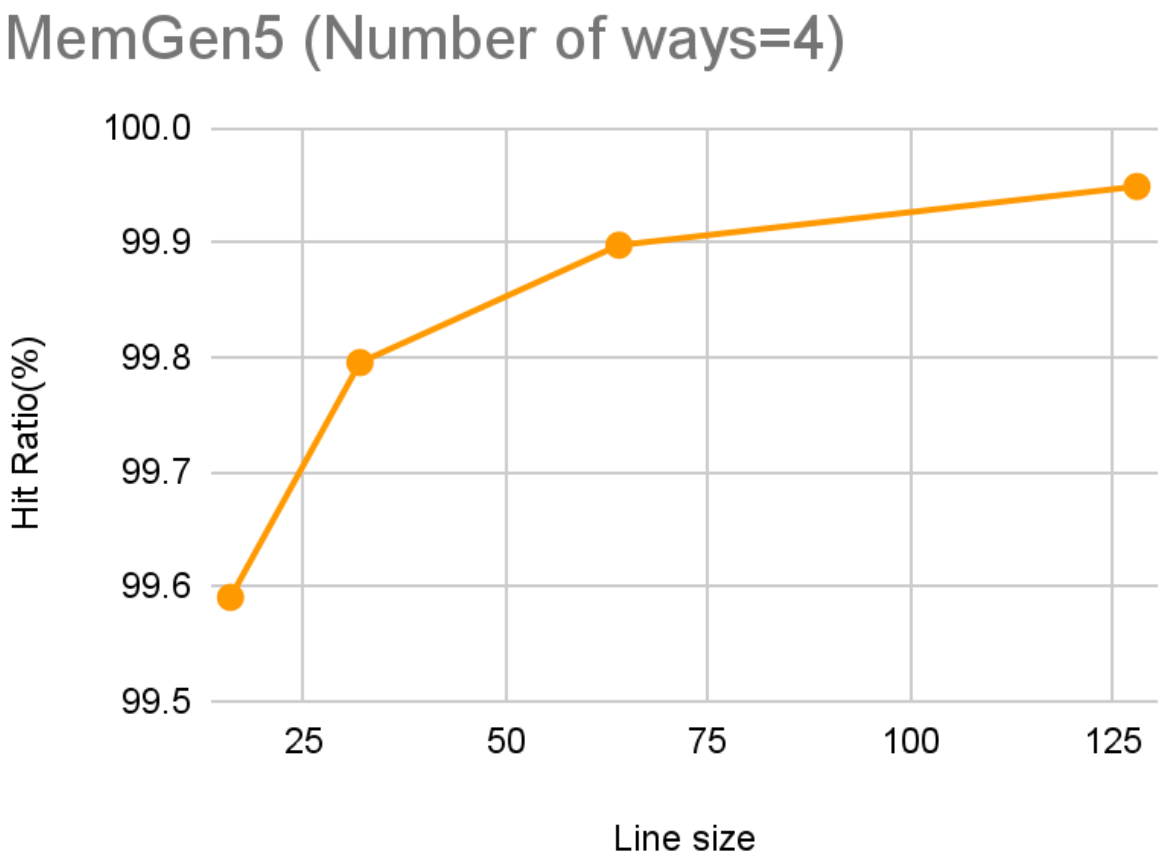


Figure 10.MemGen5() hit ratio experiment 1

Title: This graph represents the variation of the hit ratio in the set associative cache of fixed cache size (64KB) and fixed number of ways (4) as line size changes from 16 to 32 to 64 then 128.
Analysis:
There is a very slight increase that is almost negligible when looking at the graph.
The results shown indicate that the hit ratio increases as the line size increases and this is very logical. Looking at the memory generator we see that memgen5() generates sequential numbers from 0 to 1024*64.
When the line size increases the hit ratio will increase because the spatial locality is exploited. Thus, as the spatial locality is utilized better then more memory addresses can be fetched per block in every time. The latter indicates that more hits will be made.

Example: **If line size is 16**
-The first number generated is a compulsory miss.
-In this line what is stored now is this byte with the upcoming 15 bytes.
-Thus, the next 15 iterations are hits.
If line size is changed to 32
-The first number generated is a compulsory miss.
-This line npw stores the previous byte and the next 31 bytes, thus next 31 iterations are hits.
This simple example indicates that it is logical to see an increase in the hit ratio.

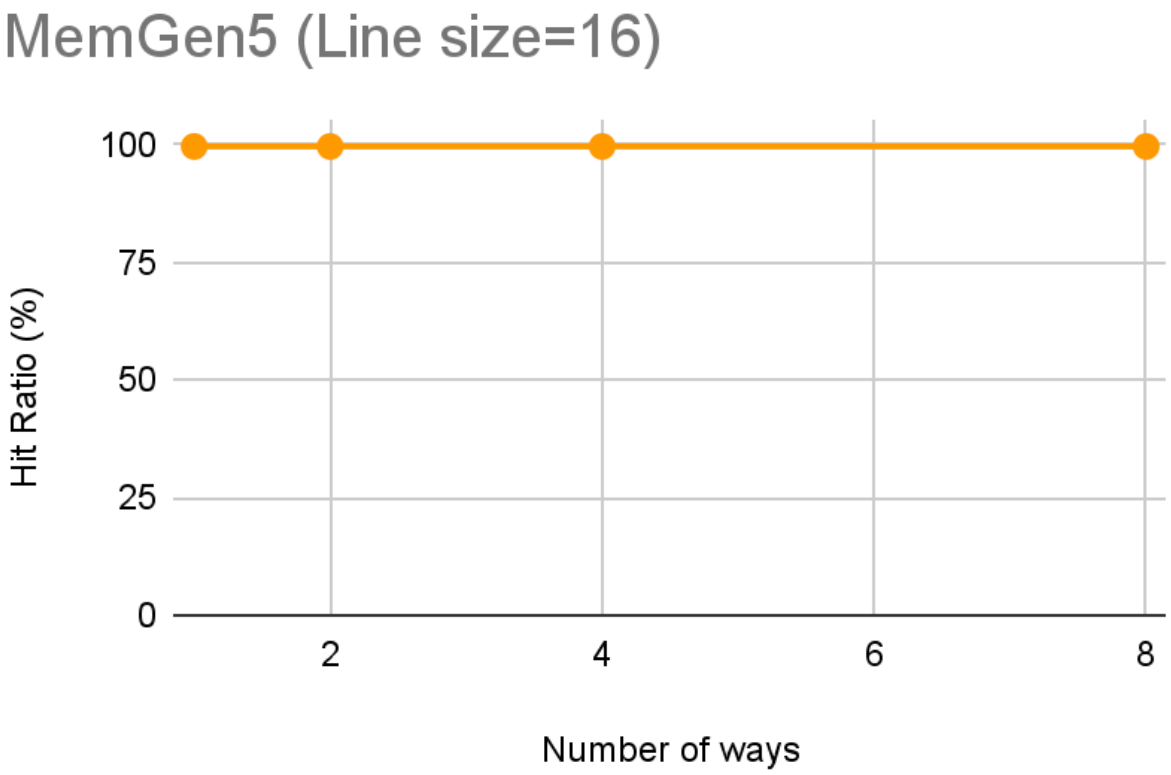
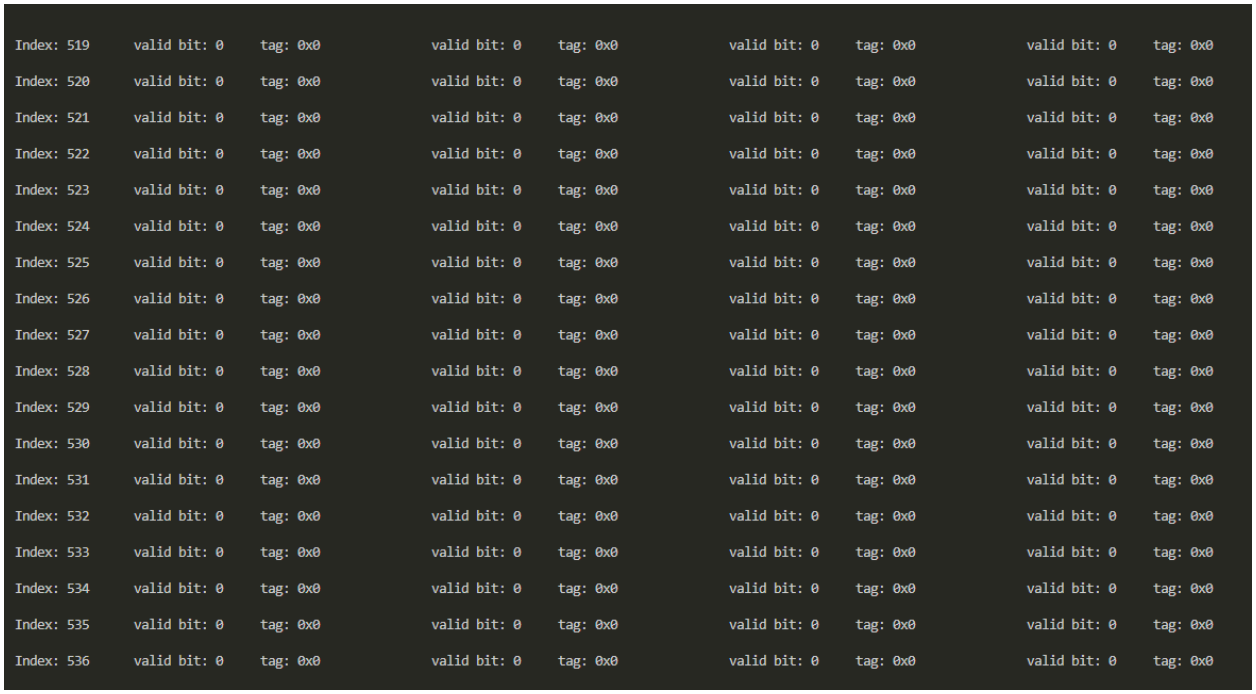


Figure 11.MemGen5() hit ratio experiment 2

Title: This graph represents the variation of the hit ratio in the set associative cache of fixed cache size (64KB) and fixed line size (16) as number of ways changes from 1 to 2 to 4 then 8.



Pic 2.Screenshot of the cache for MemGen5() in exp 2

Analysis:

This experiment shows constant values despite the change in the number of ways.

At first, this seemed odd because increasing the number of ways means that there are more lines per set so we can cache more memory into the same index which should result in higher hit ratio.

Why?

A logical explanation is that this memory generator does not make use of the number of ways no matter what this number is. As shown in Pic 4 above (4 way cache) all the tags are zero and the same applies to the 1-way cache, 2-way cache and 8-way cache.

This is clear proof that in each set only one block is used so tags will never mismatch and jump to the next block in the set.

In other words, this memory generator results in addresses that will not utilize having more than one line per set.

Memgen6()

In this section we collected the data from the function memgen6 and generated two different experiments where the variable was once the number of ways and in the other it was the cache line size.

First:

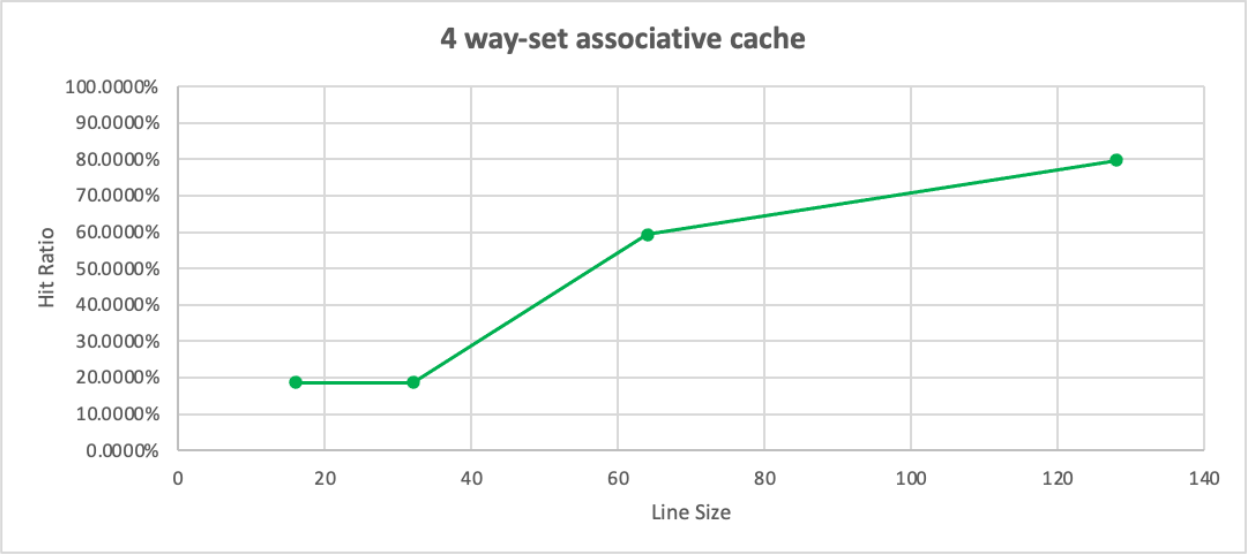
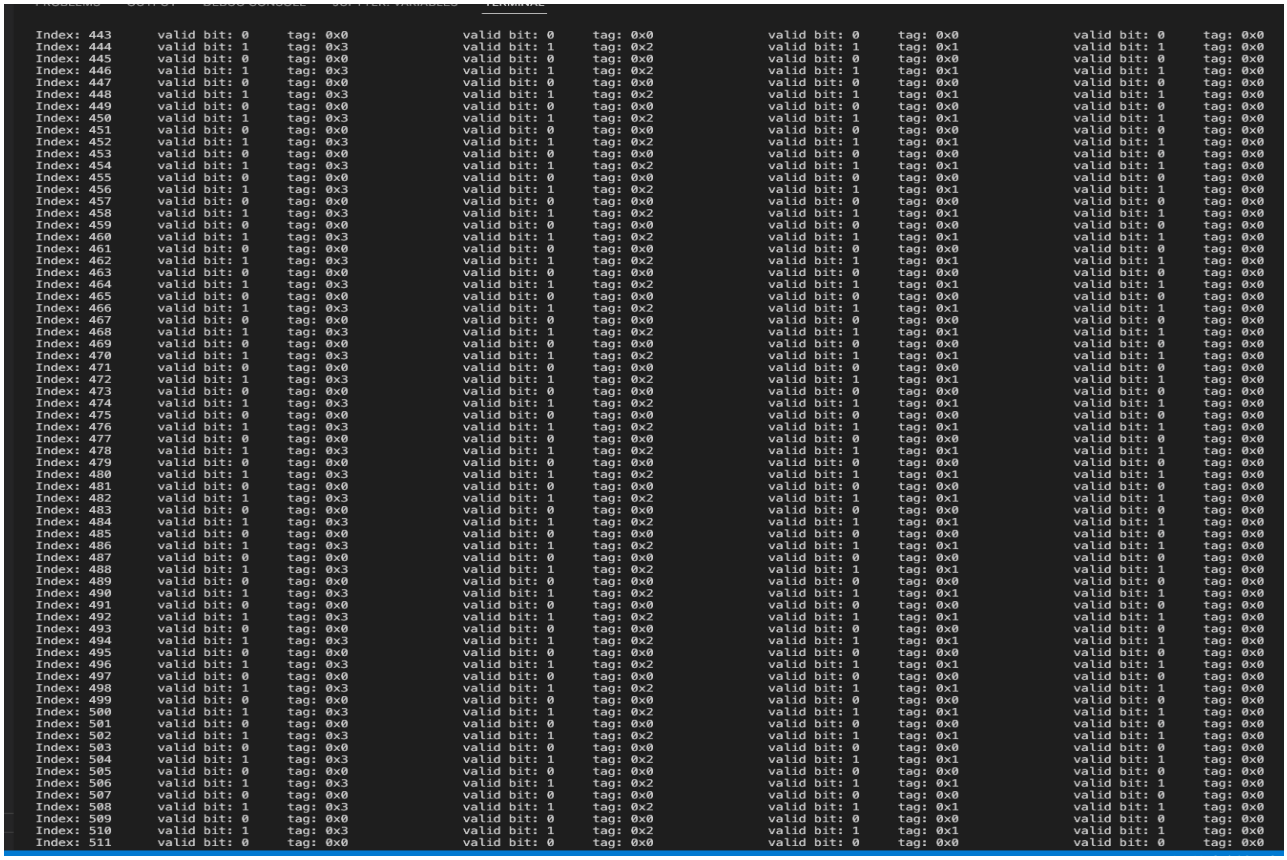


Figure 12.MemGen6() hit ratio experiment 1

The data collected from the function memgen 3 shows that the graph remains constant in the first two cases when the lize size was 16 then 32. After analyzing the data of this function and the pattern in which it generates the addresses we found that the tags of all these numbers are 0,1,2,3. Furthermore, knowing that the cache size is fixed so by increasing the cache line size by a factor of 2 we are actually decreasing the number of sets by the same factor. In the first case we had all the ways of the cache indexed by an even number full, while in the second case since we increased the number of offset bits and decreased the number of index bits so the whole cache becomes full this is illustrated in the screenshot below. As a result the benefit gained from increasing the line size is compensated by decreasing the number of sets, so the hit ratio remains constant. The hit ratio then rapidly increases (nearly three times) from 32 to 64 and from 64 to 128. This can be justified because as we said before all the numbers has the same 4 tags (0, 1, 2, 3) and by doubling the cache line size we are decreasing the number of sets so the number of misses line in each set decreases and the data keeps being generated in a loop which increases the hit ratio.



Pic 3.Screenshot of the cache for MemGen2() in exp 2

Second:

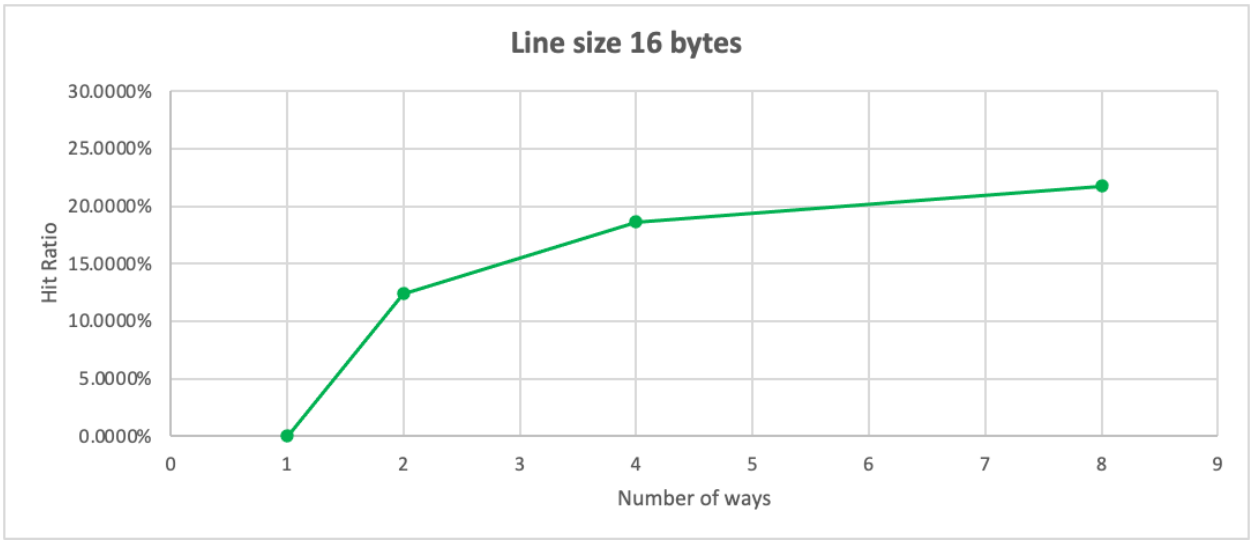


Figure 13.MemGen6() hit ratio experiment 2

As it is evident from the graph the hit ratio increases significantly when we increase the number of ways while it starts with zero. This can be justified as the one-way set associative cache actually represent a direct map cache which has a very high miss ratio in this example since the DRAM size is 64MB then we have 26 bit to represent the address 4 of them are for the offset, 12 of them for the index and the rest are for the tag. After analyzing the data from this generator and the size of cache is this experiment we found that for every index there are seven different tags and the numbers generated have fixed index so we keep overwriting the previous tags whenever getting a new one and since we have the data generated in a loop there will be 100% miss rate. This experiment is a great example of making the best use of associativity of the set as it puts to consideration the temporal locality principle that we are more likely to use the data nearly used having more than one block per set allows the simulator to write the data in a new block instead of overwriting the already existing tag, so when there is a loop the chances of having the previous tags in another block is high. As a result, by increasing the number of sets we are increasing the hit ratio.