The American University in Cairo

School of Sciences and Engineering

CSCE230301 - Comp Org.& Assembly Lang Prog

Summer 2021

Dr. Mohamed Shalaan

Project 2 / Cache Performance

Basant Elhussein Abdelaal / 900192802

Hashem Khaled Abdelfatah / 900192812

Table of Contents:

Program Implementation:

To create the cache simulator, the following steps where implemented:

1. Cache Initialization

The cache characteristics (cache size, line size and number of ways) are defined using macros. Then calculations needed (number of lines, index bits, offset bits) are calculated using those macros.
A 2d data structure (using a vector of vectors) is used to implement the cache, where the number of rows is the number of lines and the number of columns is the number of ways, and the data saved is a pair of the valid bit and tag of the saved address (initially all zeros).

2. Caching the memory addresses and returning whether it's a hit or a miss

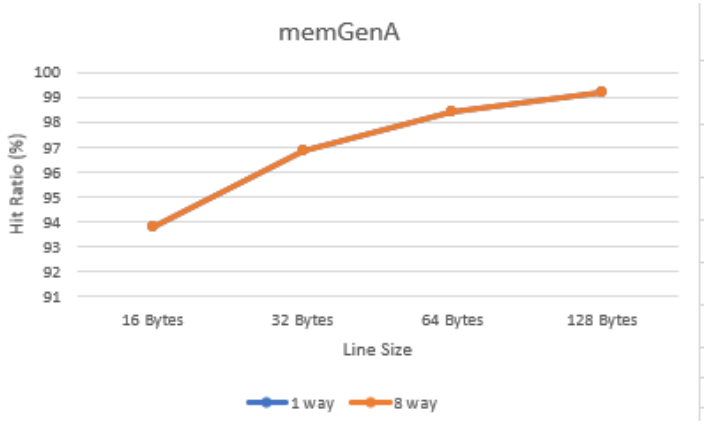
The behavior of the cache memory is simulated. First, the index bits and tag bits are calculated using bitwise operations (shifting right to start with the desired bits and then getting modulus with 1<<no. of bits).
The index determines the row in the cache. Then, we iterate over the columns (representing the ways) to check the valid bit and tag. If it matches the address, then return a HIT. Otherwise, if it finds an empty block (valid bit = 0), the tag bit is saved and the valid bit is set to 1 and returns a MISS. Otherwise, a randomly selected block is used to save the new address tag replacing a previously saved one and returns a MISS.

Data representation and analysis:

Two experiments were performed to evaluate the cache performance under different characteristics. The first experiment was to variate the line size between 16, 32, 64 and 132, and the number of ways between 1 and 8 ways. The second experiment was to fix the line size at 32 bytes and variate the number of ways between 1, 2, 4, 8, 16 and 32.
For each setting, 6 memory generators were used to generate 1000000 memory addresses. The program was run, and data (hit ratio) was collected in spreadsheets (uploaded and named as exp1 and exp2).

1. Experiment 1:

1)



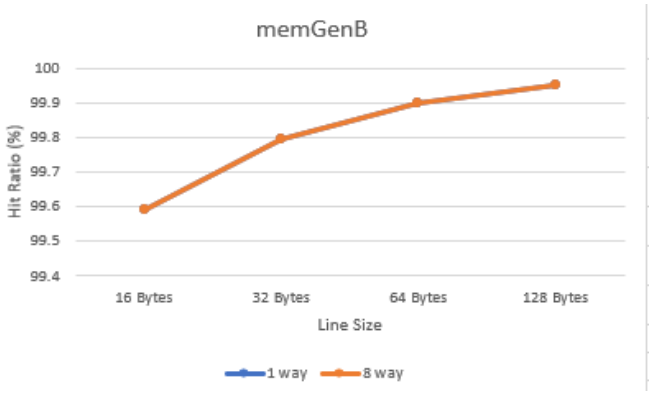
Graph (1) memGenA hit ratio experiment 1

The data collected using memGenA shows that the hit ratio increased as the line size increased. This makes sense because as we increase the line size, the spatial locality is utilized better as more memory addresses could be fetched in a single block at a time. Therefore, this will increase the hit ratio.
When increasing the number of ways, there was no effect. This is, however, should not be the case since increasing the number of ways increases the number of blocks per set and thus more memory addresses can be cached into the same index which should result in a higher hit ratio.
A possible explanation of the collected results is that this memory generator generates memory addresses that do not utilize having more than one block per set. We tried to validate this assumption and found that all this generator generates the same tag for consecutively generated addresses, as shown in Pic 1. Therefore, only the first time a different tag appears is considered a MISS and all what after is a HIT and having more than a block in each set will not have an effect.

```
Address no.: 888821, tag: 13
Address no.: 888822, tag: 13
Address no.: 888823, tag: 13
Address no.: 888824, tag: 13
Address no.: 888825, tag: 13
Address no.: 888826, tag: 13
Address no.: 888827, tag: 13
Address no.: 888828, tag: 13
Address no.: 888829, tag: 13
Address no.: 888830, tag: 13
Address no.: 888831, tag: 13
Address no.: 888832, tag: 13
Address no.: 888833, tag: 13
Address no.: 888834, tag: 13
Address no.: 888835, tag: 13
Address no.: 888836, tag: 13
Address no.: 888837, tag: 13
Address no.: 888838, tag: 13
Address no.: 888839, tag: 13
Address no.: 888840, tag: 13
Address no.: 888841, tag: 13
Address no.: 888842, tag: 13
Address no.: 888843, tag: 13
Address no.: 888844, tag: 13
Address no.: 888845, tag: 13
Address no.: 888846, tag: 13
Address no.: 888847, tag: 13
Address no.: 888848, tag: 13
```

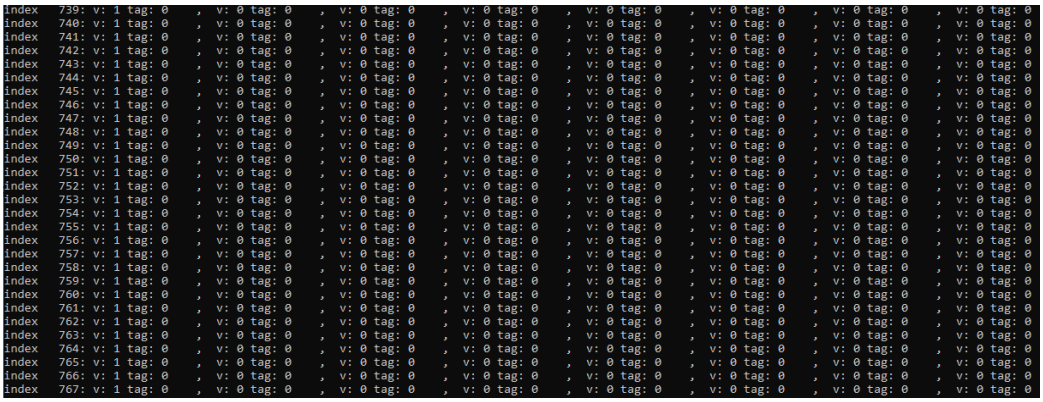
Pic 1

2)



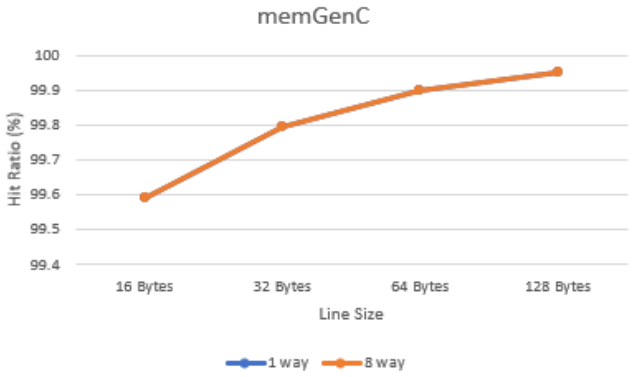
Graph (2) memGenB hit ratio experiment 1

The data collected using memGenB shows that the hit ratio increased as the line size increased. This makes sense because as we increase the line size, the spatial locality is utilized better as more memory addresses could be fetched in a single block at a time. Therefore, this will increase the hit ratio.
When increasing the number of ways, there was no effect. This is, however, should not be the case since increasing the number of ways increases the number of blocks per set and thus more memory addresses can be cached into the same index which should result in a higher hit ratio.
A possible explanation of the collected results is that this memory generator generates memory addresses that do not utilize having more than one block per set. We tried to validate this assumption and found that all the addresses generated by this memory generator have a tag of 0 as shown in Pic 2, a screenshot of the 8-way cache below. Therefore, only one block per set was utilized and the tags never mismatched to be saved in another block.



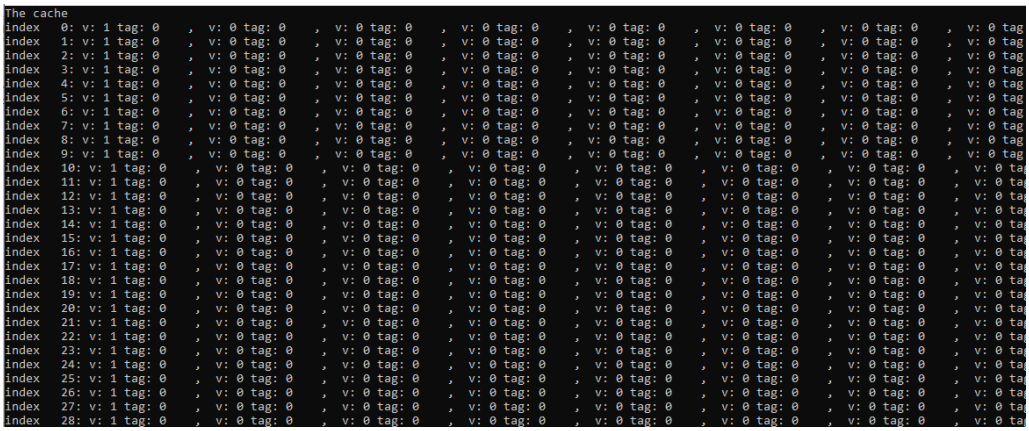
Pic 2

3)



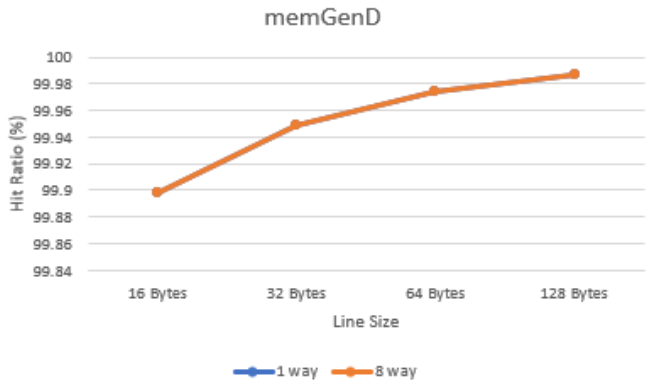
Graph (3) memGenC hit ratio experiment 1

The data collected using memGenC shows that the hit ratio increased as the line size increased. This makes sense because as we increase the line size, the spatial locality is utilized better as more memory addresses could be fetched in a single block at a time. Therefore, this will increase the hit ratio.
When increasing the number of ways, there was no effect. This is, however, should not be the case since increasing the number of ways increases the number of blocks per set and thus more memory addresses can be cached into the same index which should result in a higher hit ratio.
A possible explanation of the collected results is that this memory generator generates memory addresses that do not utilize having more than one block per set. We tried to validate this assumption and found that all the addresses generated by this memory generator have a tag of 0 as shown in Pic 3, a screenshot of the 8-way cache below. Therefore, only one block per set was utilized and the tags never mismatched to be saved in another block.



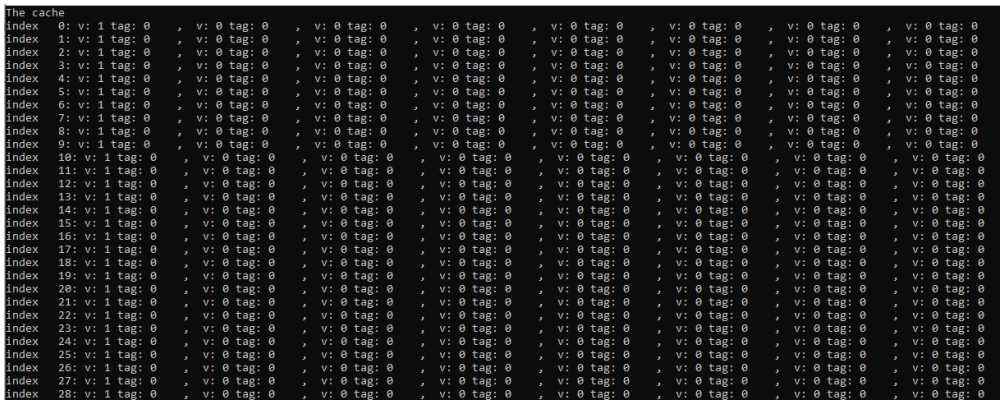
Pic 3

4)



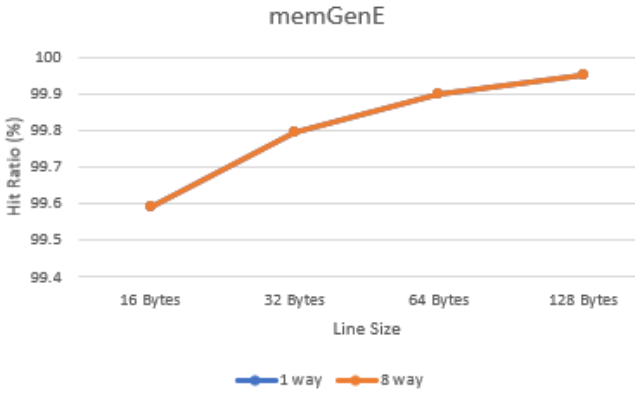
Graph (4) memGenD hit ratio experiment 1

The data collected using memGenD shows that the hit ratio increased as the line size increased. This makes sense because as we increase the line size, the spatial locality is utilized better as more memory addresses could be fetched in a single block at a time. Therefore, this will increase the hit ratio.
When increasing the number of ways, there was no effect. This is, however, should not be the case since increasing the number of ways increases the number of blocks per set and thus more memory addresses can be cached into the same index which should result in a higher hit ratio.
A possible explanation of the collected results is that this memory generator generates memory addresses that do not utilize having more than one block per set. We tried to validate this assumption and found that all the addresses generated by this memory generator have a tag of 0 as shown in Pic 4, a screenshot of the 8-way cache below. Therefore, only one block per set was utilized and the tags never mismatched to be saved in another block.



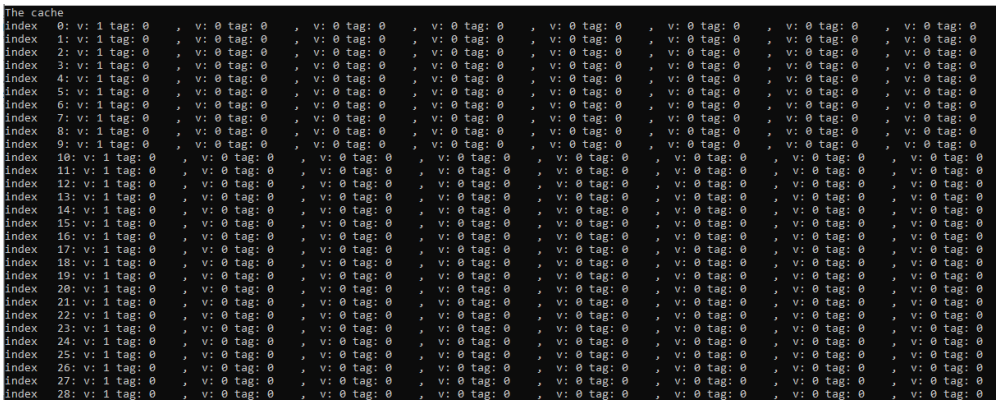
Pic 4

5)



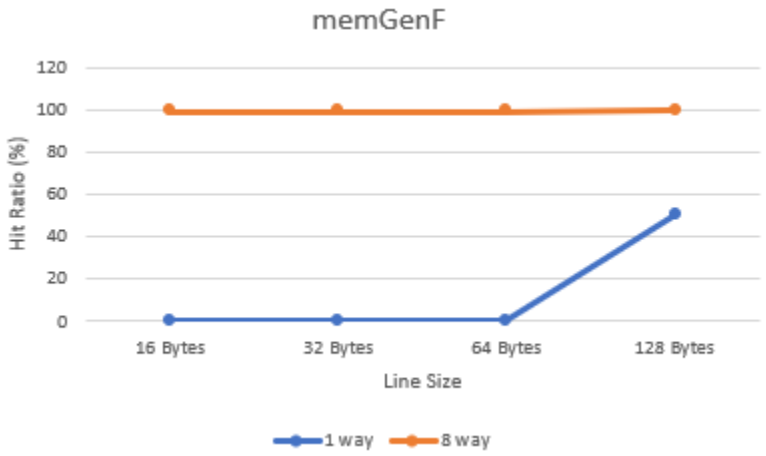
Graph (5) memGenE hit ratio experiment 1

The data collected using memGenE shows that the hit ratio increased as the line size increased. This makes sense because as we increase the line size, the spatial locality is utilized better as more memory addresses could be fetched in a single block at a time. Therefore, this will increase the hit ratio.
When increasing the number of ways, there was no effect. This is, however, should not be the case since increasing the number of ways increases the number of blocks per set and thus more memory addresses can be cached into the same index which should result in a higher hit ratio.
A possible explanation of the collected results is that this memory generator generates memory addresses that do not utilize having more than one block per set. We tried to validate this assumption and found that all the addresses generated by this memory generator have a tag of 0 as shown in Pic 5, a screenshot of the 8-way cache below. Therefore, only one block per set was utilized and the tags never mismatched to be saved in another block.



Pic 5

6)

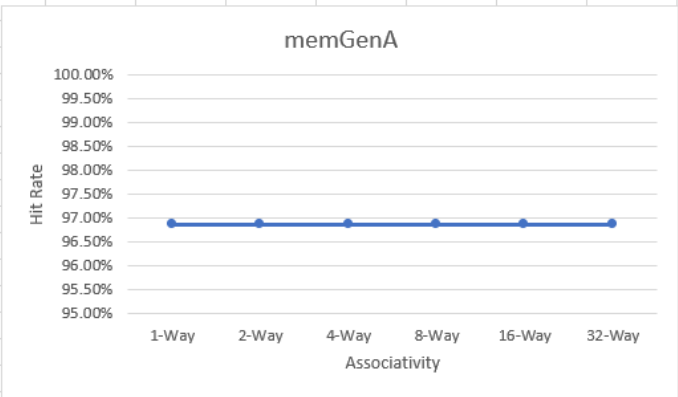


Graph (6) memGenF hit ratio experiment 1

The data collected using memGenF shows that:
At 1 way, the hit ratio increased from 0% to 50% when the line size increased to 128 bytes. This makes sense because as we increase the line size, the spatial locality is utilized better as more memory addresses could be fetched in a single block at a time. Therefore, this will increase the hit ratio. However, having the hit ratio fixed at 0% for line size 16, 32 and 64 seems a bit odd but can be explained when looking into the differently generated tags by memGenF at those sizes which led to having always MISS.
At 8 way, the hit ratio increased significantly (all above 99%) which makes sense because increasing the number of ways leads to having more blocks per index that addresses can be fetched to which increases the probability of having a HIT. Also, the hit ratio slightly increased at 128 bytes which aligns with the effect of increasing the line size and spatial locality.

2. Experiment 2:

1)



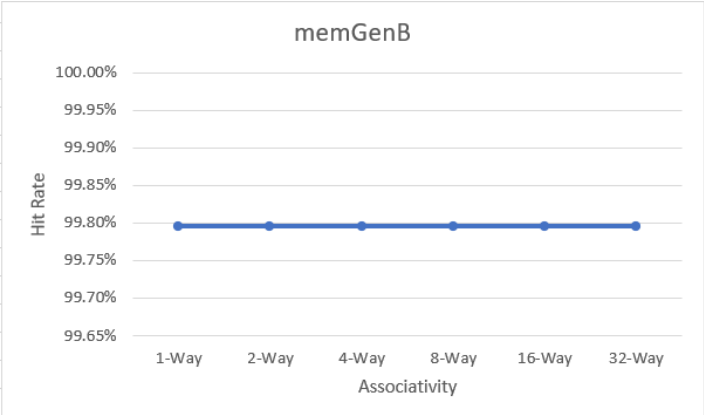
Graph (7) memGenA hit ratio experiment 2

The data collected using memGenA shows that the hit ratio does not change by changing the number of ways. This seems to be an odd behavior since increasing the number of ways increases the number of blocks per set and thus more memory addresses can be cached into the same index which should result in a higher hit ratio. A possible explanation of the collected results is that this memory generator generates memory addresses that do not utilize having more than one block per set. We tried to validate this assumption and found that all this generator generates the same tag for consecutively generated addresses, as shown in Pic 6. Therefore, only the first time a different tag appears is considered a MISS and all what after is a HIT and having more than a block in each set will not have an effect.

```
Address no.: 888821, tag: 13
Address no.: 888822, tag: 13
Address no.: 888823, tag: 13
Address no.: 888824, tag: 13
Address no.: 888825, tag: 13
Address no.: 888826, tag: 13
Address no.: 888827, tag: 13
Address no.: 888828, tag: 13
Address no.: 888829, tag: 13
Address no.: 888830, tag: 13
Address no.: 888831, tag: 13
Address no.: 888832, tag: 13
Address no.: 888833, tag: 13
Address no.: 888834, tag: 13
Address no.: 888835, tag: 13
Address no.: 888836, tag: 13
Address no.: 888837, tag: 13
Address no.: 888838, tag: 13
Address no.: 888839, tag: 13
Address no.: 888840, tag: 13
Address no.: 888841, tag: 13
Address no.: 888842, tag: 13
Address no.: 888843, tag: 13
Address no.: 888844, tag: 13
Address no.: 888845, tag: 13
Address no.: 888846, tag: 13
Address no.: 888847, tag: 13
Address no.: 888848, tag: 13
```

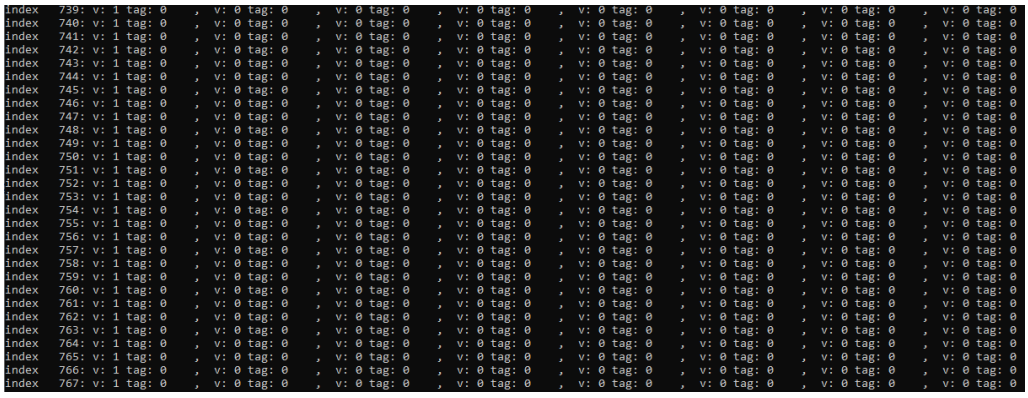
Pic 6

2)



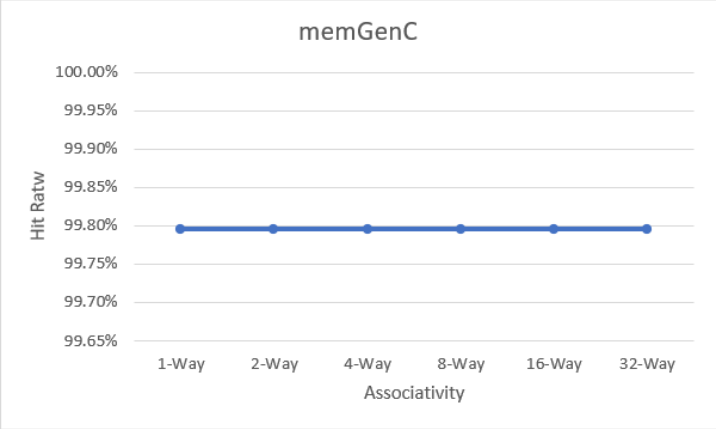
Graph (8) memGenB hit ratio experiment 2

The data collected using memGenB shows that the hit ratio does not change by changing the number of ways. This seems to be an odd behavior since increasing the number of ways increases the number of blocks per set and thus more memory addresses can be cached into the same index which should result in a higher hit ratio. A possible explanation of the collected results is that this memory generator generates memory addresses that do not utilize having more than one block per set. We tried to validate this assumption and found that all the addresses generated by this memory generator have a tag of 0 as shown in Pic 7, a screenshot of the 8-way cache below. Therefore, only one block per set was utilized and the tags never mismatched to be saved in another block.



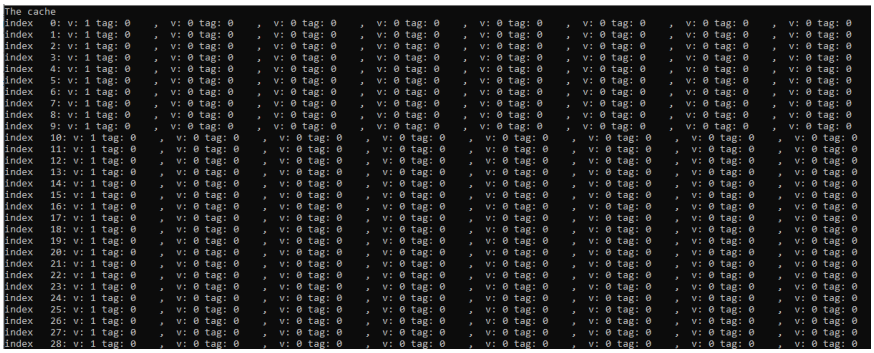
Pic 7

3)



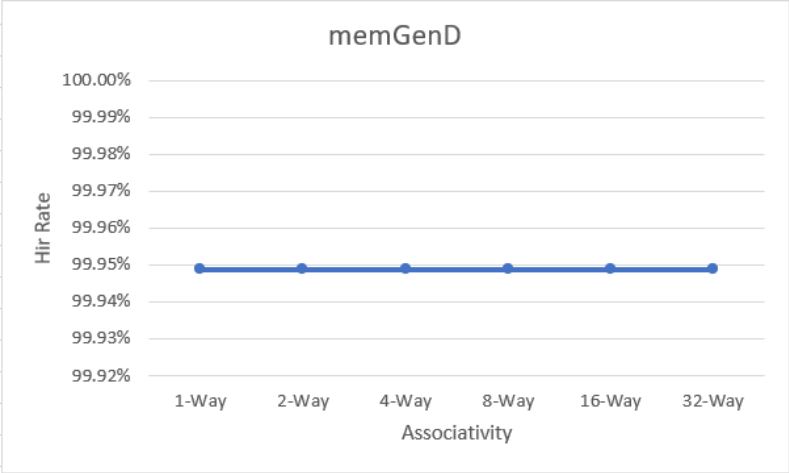
Graph (9) memGenC hit ratio experiment 2

The data collected using memGenC shows that the hit ratio does not change by changing the number of ways. This seems to be an odd behavior since increasing the number of ways increases the number of blocks per set and thus more memory addresses can be cached into the same index which should result in a higher hit ratio. A possible explanation of the collected results is that this memory generator generates memory addresses that do not utilize having more than one block per set. We tried to validate this assumption and found that all the addresses generated by this memory generator have a tag of 0 as shown in Pic 8, a screenshot of the 8-way cache below. Therefore, only one block per set was utilized and the tags never mismatched to be saved in another block.



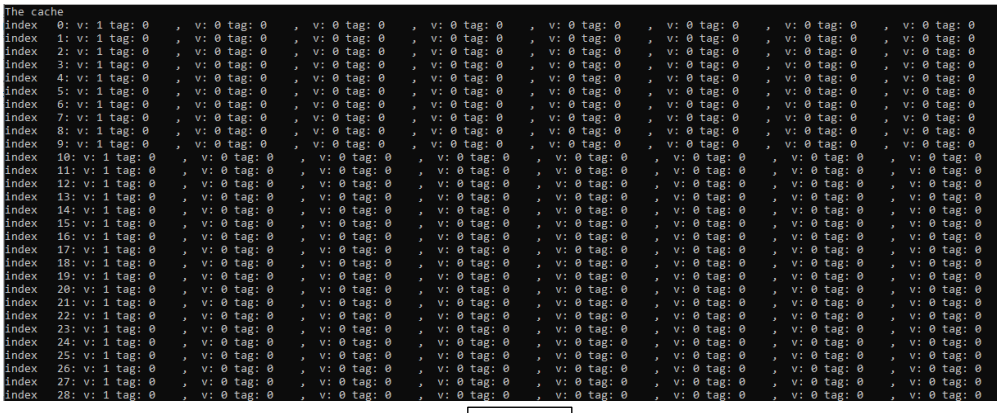
Pic 8

4)



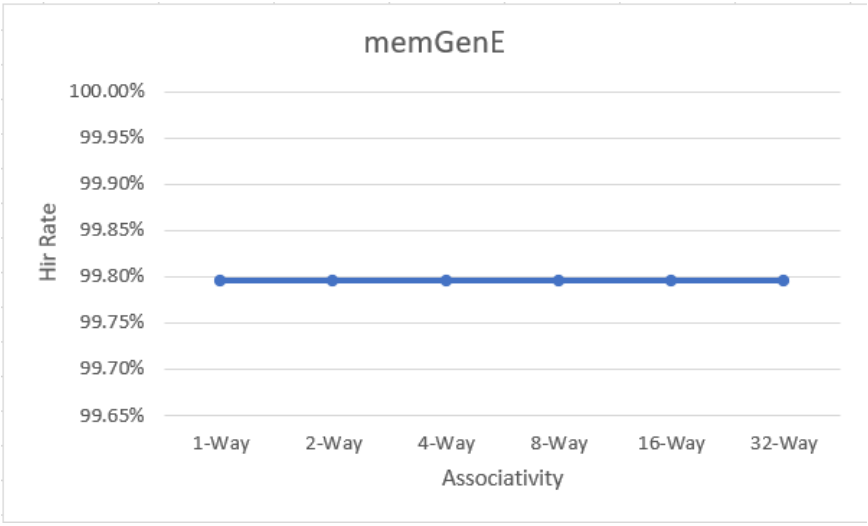
Graph (10) memGenD hit ratio experiment

The data collected using memGenD shows that the hit ratio does not change by changing the number of ways. This seems to be an odd behavior since increasing the number of ways increases the number of blocks per set and thus more memory addresses can be cached into the same index which should result in a higher hit ratio. A possible explanation of the collected results is that this memory generator generates memory addresses that do not utilize having more than one block per set. We tried to validate this assumption and found that all the addresses generated by this memory generator have a tag of 0 as shown in Pic 9, a screenshot of the 8-way cache below. Therefore, only one block per set was utilized and the tags never mismatched to be saved in another block.



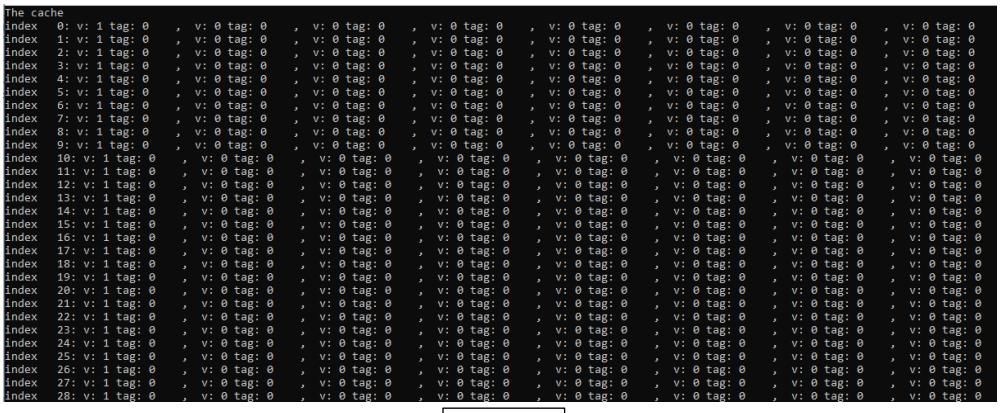
Pic 9

5)



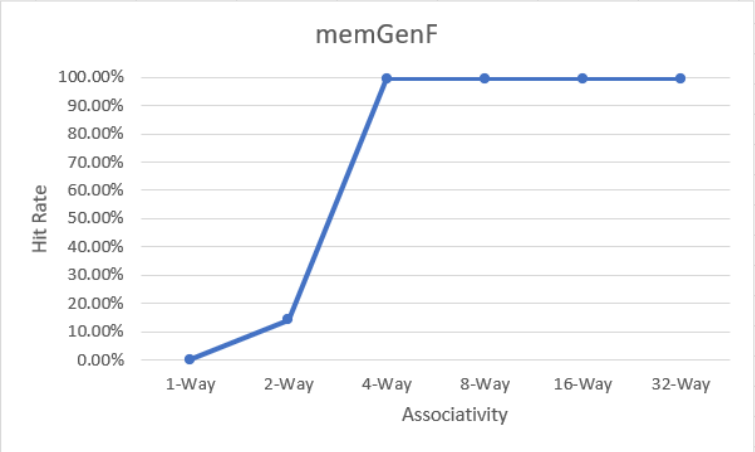
Graph (11) memGenE hit ratio experiment 2

The data collected using memGenE shows that the hit ratio does not change by changing the number of ways. This seems to be an odd behavior since increasing the number of ways increases the number of blocks per set and thus more memory addresses can be cached into the same index which should result in a higher hit ratio. A possible explanation of the collected results is that this memory generator generates memory addresses that do not utilize having more than one block per set. We tried to validate this assumption and found that all the addresses generated by this memory generator have a tag of 0 as shown in Pic 10, a screenshot of the 8-way cache below. Therefore, only one block per set was utilized and the tags never mismatched to be saved in another block.



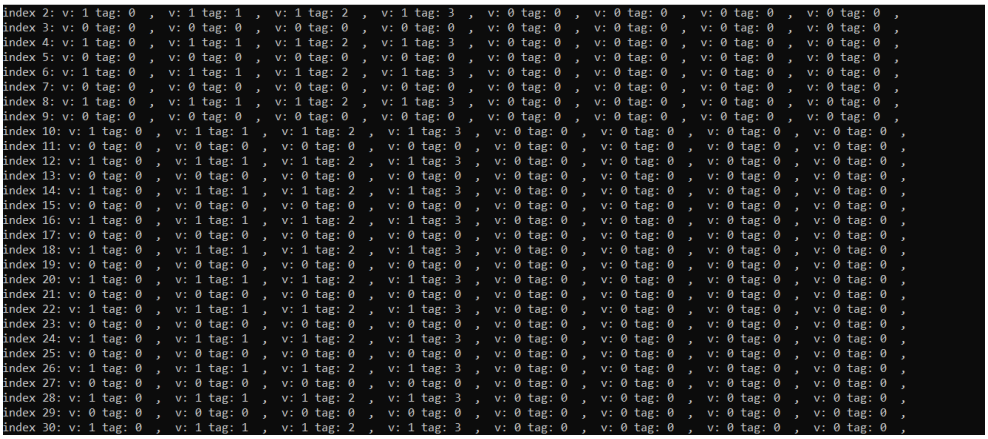
Pic 10

6)



Graph (12) memGenF hit ratio experiment 2

The data collected from memGenF shows that the hit ratio increases by increasing the associativity. In other words, increasing the number of ways leads to an increase in the hit ration, which makes sense because increasing the number of ways increases the number of blocks per set and thus more memory addresses can be cached into the same index which results in a higher hit ratio. From the data, the hit ratio increases slightly when we increase the number of ways from 1-way to 2-way, however the hit ratio increases rapidly when we increase the number of ways form 2-way to 4-way and after that it stays constant and a possible explanation for that is at this moment the memory generator generates memory addresses that do not utilize having more than 4 blocks per set. We tried to validate this assumption and found that all the addresses generated by this memory generator have a tag ranging from 0 up to 3 which means only 4 possible values for the tag as shown in Pic 11, a screenshot of the 8-way cache below. Therefore, only four blocks per set was utilized and the tags never exceeds 3 to be saved in the fifth block of the set.



Pic 11