Ali Ghazal 900171722

Omer Hassan 900171920

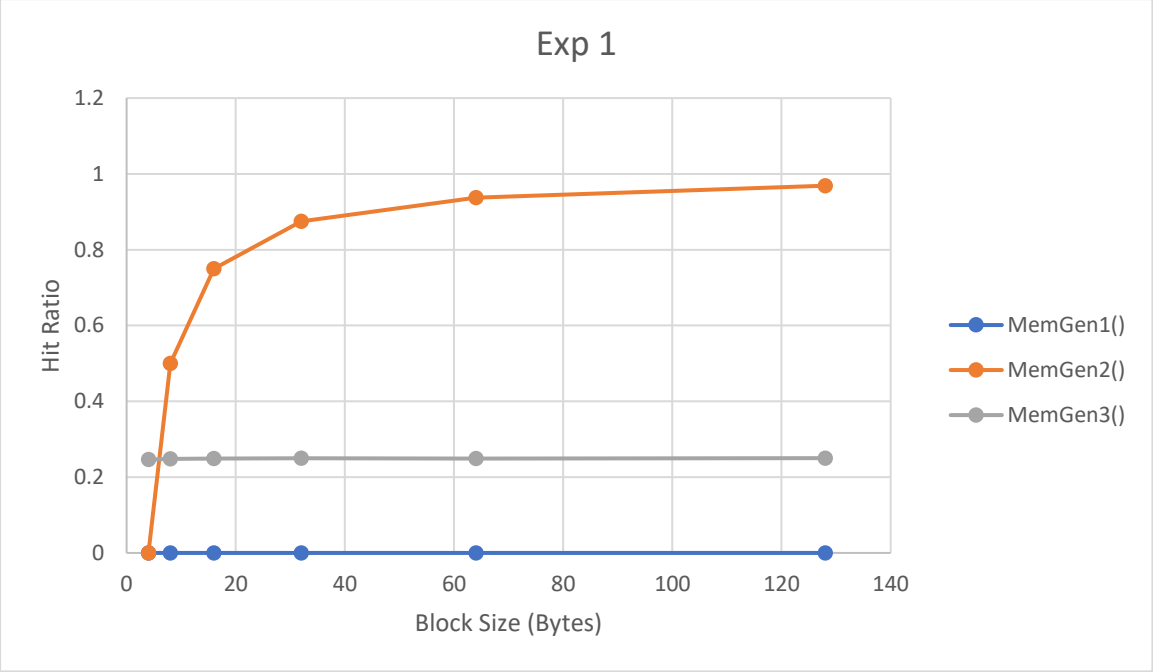
Final Report and Data Analysis

Assembly Project #2

Experiment 1 Results.

Cache Size is fixed to 64KB.

Hits for 1000000 Addresses for different memGens	MemGen1()	MemGen2()	MemGen3()
Block size 4	0	0	246185
Block size 8	0	499999	247949
Block size 16	0	749999	249141
Block size 32	0	874999	249374
Block size 64	0	937499	249096
Block size 128	0	968749	250075



Analysis:

1) **Accessing defined positions in memory (memGen1)**

Accessing defined positions in memory which are all multiple map to the same slot in the cache results in a constant performance regardless of the size of the block size. There are only 16 addresses that will be mapped to the same slot in the cache resulting in 100% miss rate or 0 hits.

Only the first one would be a cold start miss, while all the others would be conflict misses.

2) Accessing increasingly contagious memory addresses (memGen2)

The performance different is very clear in this case (which is representative of the normal process of accessing the memory due to the concept of spatial locality).

Apparently, the performance increased logarithmically (1/2, 3/4, 7/8,…) with increasing the block size by multiples of two since the number of loaded addresses after every miss increases. In other words, since address are multiple of 4 every blocksize /4 addresses map to the same position. So it will be zero hits at first then 1/2 of the times and then 3/4 and so on giving the graph shown.

Initially, most misses are cold start misses. However, depending on the size of the cache, conflict misses starts to appear.

3) Accessing totally random addresses in the memory (memGen3)

The performance of all caches converged into a very close number -- namely, a hit ratio of 0.25. This is probably because this function generates numbers that are between 0 and 256KB-1 and so the effective addresses will be 64/block size and the random effective addresses by the function will be =256/block size. Therefore, we will have a hit every 3 misses or in other words the hits will be 1/4 =64/256 and will be constant to 0.25 all over the cycles.

Experiment 2

Block Size is fixed to 64 Bytes

Level 1 Cache has a size of 256 KB and 11 Cycles speed.

AMAT=Speed+ Miss rate * memory speed

	MemGen1()	**MemGen2()**	**MemGen3()**
Miss rate	0.000016	0.06250	0.004096
AMAT	11.0016	17.2501	11.4096

Level 1 Cache with size 32KB (4Cycles), Level 2 with 256KB (11 Cycles).

AMAT_Level 2 = L2_Speed + miss_L2/miss_L1 * Memory_speed

AMAT_total = L1_Speed + miss_L1 /1000000 * AMAT_Level 2

	MemGen1()	**MemGen2()**	**MemGen3()**
Miss Rate of Level 1	1	0.062501	0.874877
Miss Rate of Level 2	0.000016	1	0.0047
AMAT	15.0016	10.9376	14.0382

Level 1 Cache with size 32KB (4Cycles), Level 2 with 256KB (11 Cycles), Level 3 with 8 MB (30 Cycles).

AMAT_Level 3 = L3_Speed + miss_L3/ miss_L2 * Memory_speed

AMAT_Level 2 = L2_Speed + miss_L2/ miss_L1 * AMAT_Level 3

AMAT_total = L1_Speed + miss_L1 /1000000 * AMAT_L2

	MemGen1()	**MemGen2()**	**MemGen3()**
Miss Rate of Level 1	1	0.06250	0.874877
Miss Rate of Level 2	0.000016	1	0.0047
Miss Rate of Level 3	1	1	1
AMAT	15.0021	12.8126	14.1561

Analysis:

1) **Accessing defined positions in memory (memGen1)**

In the one-level cache of size 256 kb, the 16 memory positions generated periodically from memGen1 are mapped into unique indices. Consequently, after the first 16 miss (cold start misses), they are loaded into the memory and the rest of the memory searches would be hits. And this is why we are getting an AMAT of 11.0016 (very close to actual accessing time of 11.00).

On the other hand, in the two-level and three-level caches, level-one's size in decreased to 32 kb, consequently, all those positions are mapped into one index which keeps on overwriting its content with each miss (conflict misses). However, those misses are mapped into unique indices in the second level of the cache, so they are all result in hits in level two, without a need to search the third level or the memory.

2) **Accessing increasingly contagious memory addresses (memGen2)**

In all three cache designs, we have a block size of 64 byte. So, after each miss in level one, we load a block of 64 byte which result in a 64 hit.

So, one-level cache has the worst performance since each miss costs 100 cycle to access the memory. However, using multi-level cache increase the core performance of level-one access, which decrease the entire AMAT.

However, the performance difference between the two and three level caches results from the cost of searching through the level-three before retrieving the needed value form memory.

Initially, this memory access mode results in cold start misses. However, depending on the size of the cache, conflict misses starts to appear.

3) Accessing totally random addresses in the memory (memGen3)

The nature of the memGen3 is that it generates an address (% 256 * 1024), so, in the one-level cache with exactly 256 kb, this was exactly all that is needed to fill the cache and then finding all those possible addresses in the cache. Consequently, the one-level cache had the best performance.

However, in the multi-level designs (two and three) the second level in both designs has a size of 256kb, so it takes this amount of misses and access between the first two levels all possible memory addresses could be loaded into the cache.

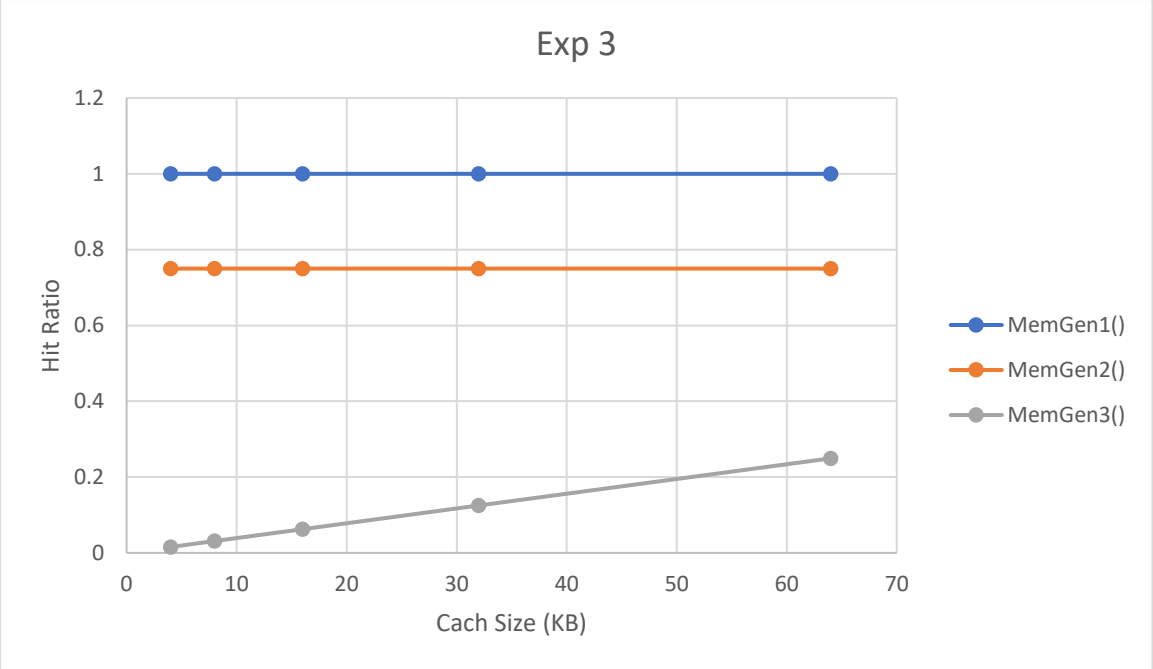
Also, the performance decreases again in the three-level cache because we are adding the cost of searching in level-three before visiting the memory to retrieve the needed address.

Note that the reason why there is these number of hits is explained in detail in exp 1 as it depends on the nature of every memGen function.

Experiment 3

Block Size is fixed to 16 bytes with MRU Replacement. We report on the number of hits for each million address (hit ratio).

Cache Size (KB)	4	8	16	32	64
MemGen1()	999984	999984	999984	999984	999984
MemGen2()	749999	749999	749999	749999	749999
MemGen3()	15563	31128	62478	125234	249337



Analysis and Conclusion: -

Note that no capacity misses in memgen1 and 2 since we don't search for the same address twice. So, all the misses are compulsory. Capacity misses appear only in memgen3 and it doesn't affect hit ration by a big number

MemGen1():

Since MemGen1() generates only 16 distinct addresses, and the cache is full associative and so the addresses can be placed in any line in the cache. Hence, there will be 16 misses (cold start misses) at first and after that all of them will be hits if the number of lines of the cache exceeds 16 lines so that there will be no overwrite which is the case in all cache sizes given. The final hit ratio for any size of cache give will be 1 M-16=9999984 hits and it will be constant as shown in the graph. Note that there will be no need to use the replacement policy**.**

MemGen2():

MemGen2() produces consecutive outputs that are multiples of 4 and it repeats every 64 MB which is more than what is used in the 6 experiments. Since the block size is 16 Bytes, every 4 addresses will be mapped to the same index in the cache. Therefore, we will have one miss in every 4 hits because we will move every 4 addresses to the same place. So, there will be 1/4 of the whole time as misses (capacity misses) and the hit ration will be 3/4 or .75 as shown. Note that it's independent of the cache size and the replacement policy because the addresses are distinct it's only the Block size that affects it (i.e. if the block size is 32 the hit ration will be 7/8 or 0.875. So, the hit rate will be constant and equal to 3/4 for all cache size.

MemGen3():

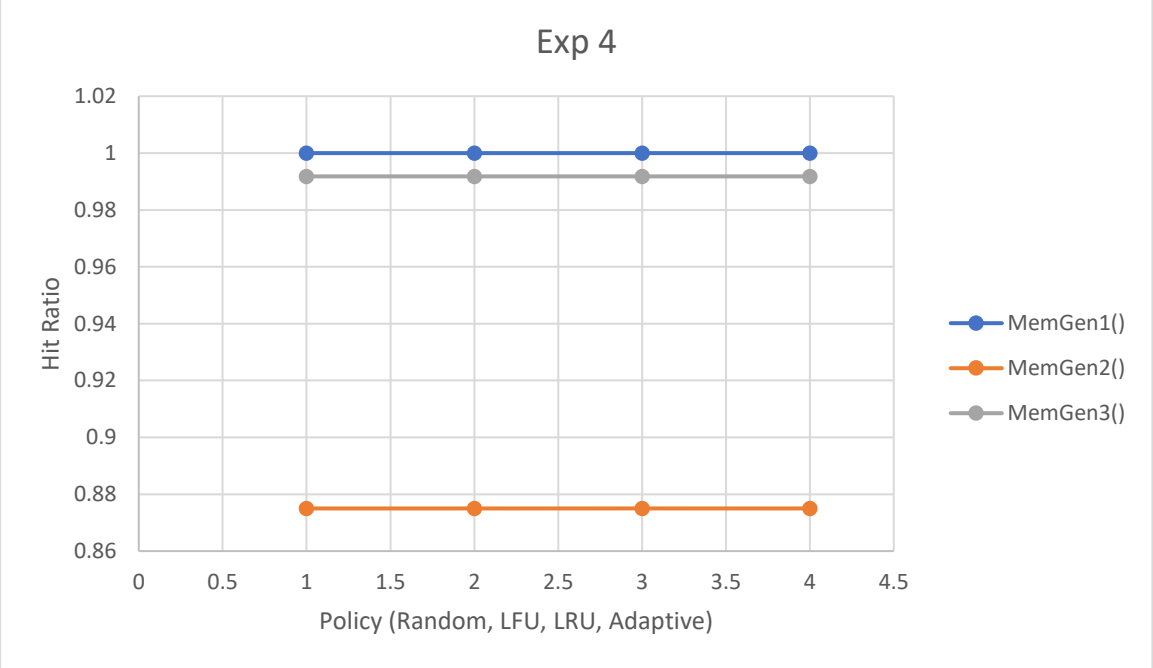
The function generates random values that ranges from zero to 256 KB-1 so all the number are less than 256 KB. The effective address or the number of indices will be (256KB/block size) addresses that will be searched and stored in the cache. So the hit ratio we can get given 16 bytes block size will be around (size of cache /256) which will give about 4/256 or 0.015625 in the cache size of 4 KB and 8/256 or 0.03125 and will increase by a factor of 2 till we reach 1/4 or .25 for 64KB cache, which

matches the given line with slope 2 in the graph. (In conclusion, we will have a hit every (cache size /256) time excluding the first time misses where there is nothing in the cache giving the increasing line shown in the figure.) Note that the replacement policy isn't a key factor here since the hit ratio is a probability relation in the cache size and we need to replace constant number of times for each cache size. Even though, the replacement policy and the capacity misses affect the hit ration by small differences.

Experiment 4

Block Size is fixed to 32 bytes with cache size 256 KB. We report on the hit ratio.

	MemGen1()	MemGen2()	MemGen3()
Random Replacement	**999984**	**874999**	**991808**
LFU Replacement	**999984**	**875000**	**991808**
LRU Replacement	**999984**	**874999**	**991808**
Adaptive LRU/LFU Replacement	**999985**	**875001**	**991808**



Analysis and Conclusion: -

Note that no capacity misses in memgen1 and 2 and 3 since we don't search for the same address twice. So, all the misses are compulsory.

MemGen1():

Since MemGen1() generates only 16 distinct addresses, and the cache is full associative and so the addresses can be placed in any line in the cache. Hence, there will be 16 misses at first (cold start misses) and after that all of them will be hits if the number of lines of the cache exceeds 16 lines so that there will be no overwrite which is the case in all cache sizes given. The final hit ratio for any size of cache give will be 1 M-16=9999984 hits. Note that there will be no need to use the

replacement policy as discussed in the previous experiment. So, the results is independent of the replacement policy, producing a constant graph as shown in the graph.

MemGen2():

MemGen2() produces consecutive outputs that are multiples of 4 and it repeats every 64 MB which is more than what is used in the 6 experiments. Since the block size is 32 Bytes, every 8 addresses will be mapped to the same index in the cache. Therefore, we will have one miss in every 8 hits because we will move every 8 addresses to the same place (capacity misses). So, there will be 1/8 of the whole time as misses and the hit ratio will be (1-1/8) of 7/8 (0.875) as shown. Note that it's independent of the cache size and the replacement policy because the addresses are distinct it's only the Block size that affects it . So, the hit rate will be constant and equal to 0.875 for all replacement policies.

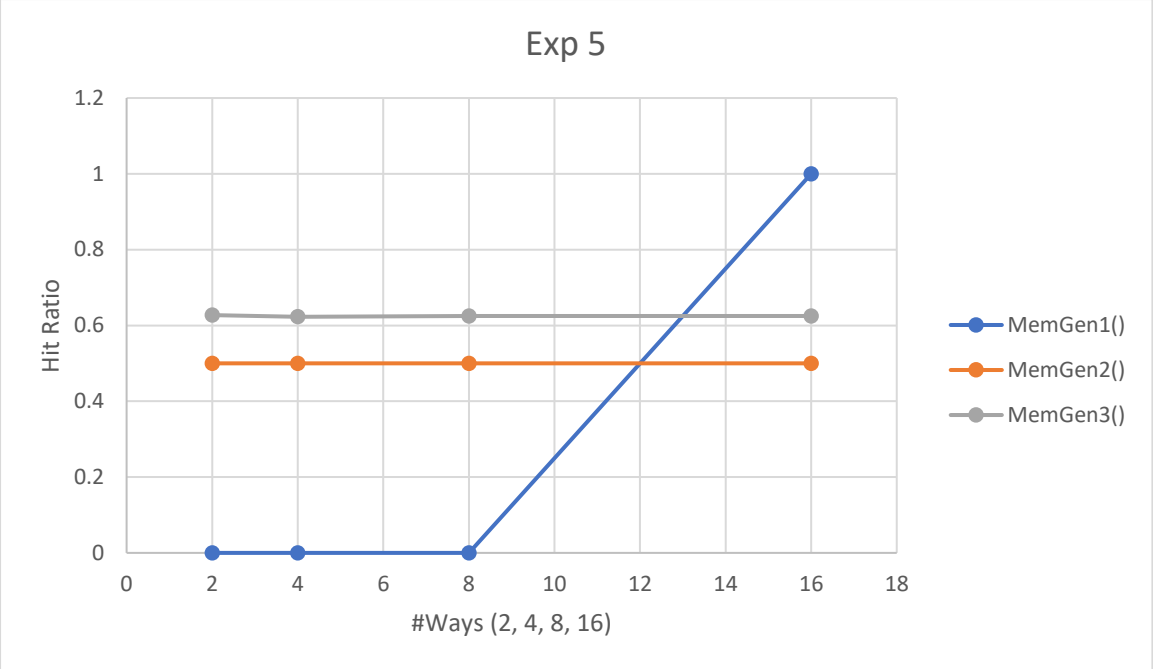
MemGen3():

The function generates random values that ranges from zero to 256 KB-1 so all the number are less than 256 KB. The effective address or the number of indices will be (256KB/block size) addresses that will be searched and stored in the cache. So the hit ratio we can get given 16 bytes block size will be around (size of cache) /256. Since cache size is 256 KB and Block size is 32B and they are constant all the way. We will have 256/32=8 K indices (effective addresses that are generated by MemGen3()), and at the same time, 8 K lines exists in the cache. So we will only have 8K misses at first (cold start misses). Then, the address will be found in the cache in the rest of the cycles since no overwrite will happen. Therefore, the hit ration is constant to 1M-8K=1M-8192=991808 hits and it will be constant all the way and no replacement policy will be ever needed to be used as shown in the graph.

Experiment 5

Cache Size is fixed to 16 KB and block size is 8 bytes. We report on the hits (hit ratio)

#Ways	2-way	4-way	8-way	16-way
MemGen1()	0	0	0	999984
MemGen2()	499999	499999	499999	499999
MemGen3()	62723	62283	62485	62493



Analysis and Conclusion: -

MemGen1():

Since MemGen1() generates only 16 distinct addresses, they will be mapped to only one set in the set associative. Hence, if the number of ways is less than 16 there will be overwrite every time (capacity) and so giving the hit rate of zero. If it's 16- way, we will have only 16 misses (cold start) and the rest will be hits giving 1M-16 =9999984 hits which explains this sudden jump in the graph.

MemGen2():

MemGen2() produces consecutive outputs that are multiples of 4 and it repeats every 64 MB which is more than what is used in the 6 experiments. Since the block size is 8 Bytes, every 2 addresses will be mapped to the same index in the cache. Therefore, we will have one miss in every 1 hit because we will move every 2 addresses to the same place. So, there will be 1/2 of the whole time as misses and the hit ratio will be 1/2. Note that it's independent of the cache size and dependent on the block size as explained before. So the graph will be constant to approximately 0.5 as shown and it's independent on the number of ways in the cache.

MemGen3():

The function generates random values that ranges from zero to 256 KB-1 so all the number are less than 256 KB. The effective address or the number of indices will be (256KB/block size) addresses that will be searched and stored in the cache. So the hit ratio we can get given 16 bytes block size will be around (size of cache) /256. Since cache size is 256 KB and Block size is 8 B and they are constant all the way. We will have 256/8=32 K indices (effective addresses that are generated by MemGen3()), and at the same time, 2 K lines exists in the cache. Therefore, the hit ratio will be cachelines /32 or 2/32 or 1/16 (approximately 0.0625). The hit ration is dependent on the cache size and independent of the block size and the number of ways as explained before. So the hit rate will be constant to 0.0625 approximately as in the graph.

Experiment 6

Block Size is fixed to 64 Bytes.

Level 1 Cache has a size of 256 KB , 11 Cycles speed, and 8 Ways.

AMAT=speed+miss rate*memory speed

	MemGen1()	MemGen2()	MemGen3()
Miss rate	**0.000016**	**0.06251**	**0.004096**
AMAT	**11.0016**	**17.2501**	**11.4096**

Level 1 is 8-way Cache with size 32KB (4Cycles), Level 2 is 8-way with 256KB (11 Cycles).

AMAT_Level 2 = L2_Speed + miss_L2/miss_L1 * Memory_speed

AMAT_total = L1_Speed + miss_L1 /1000000 * AMAT_Level 2

	MemGen1()	MemGen2()	MemGen3()
Miss Rate of Level 1	**0.000016**	**0.062501**	**0.004096**
Miss Rate of Level 2	**1**	**1**	**1**
AMAT	**15.0016**	**10.9376**	**14.0364**

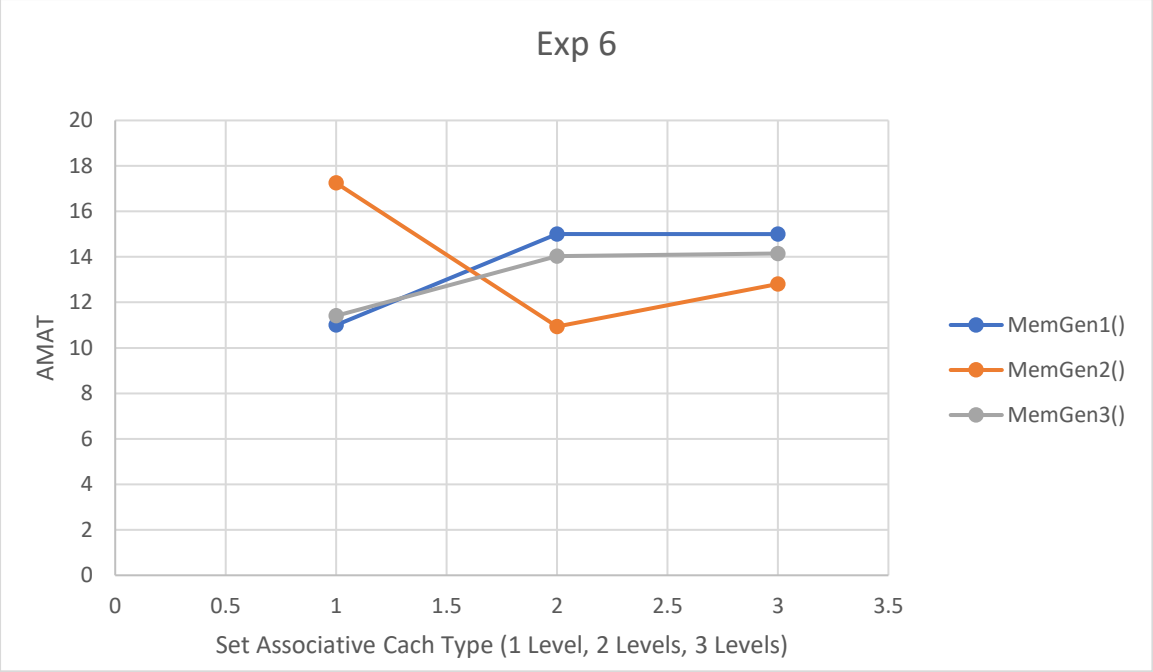
Level 1 is 8-Way Cache with size 32KB (4Cycles), Level 2 8-way with 256KB (11 Cycles), Level 3 is 16-way with 8 MB (30 Cycles).

AMAT_Level 3 = L3_Speed + miss_L3/ miss_L2 * Memory_speed

AMAT_Level 2 = L2_Speed + miss_L2/ miss_L1 * AMAT_Level 3

AMAT_total = L1_Speed + miss_L1 /1000000 * AMAT_L2

	MemGen1()	MemGen2()	MemGen3()
Miss Rate of Level 1	**0.0000016**	**0.062501**	**0.004096**
Miss Rate of Level 2	**1**	**1**	**1**
Miss Rate of Level 3	**1**	**1**	**1**
AMAT	**15.0021**	**12.8126**	**14.1493**



1) Accessing defined positions in memory (memGen1)

In an 8-way, one-level cache of size 256 kb , the 16 memory positions generated periodically from memGen1 are mapped into exactly two sets each with 8 ways. Consequently, after the first 16 miss, (cold start) they are loaded into the memory and the rest of the memory searches would be hits. And this is why we are getting an AMAT of 11.0016 (very close to actual accessing time of 11.00).

On the other hand, in the two-level and three-level caches, level-one's size in decreased to 32 kb, consequently, all those positions are mapped into one set which keeps on overwriting its content with each miss. However, those misses are mapped into two sets in the second level of the cache, so they are all result in hits in level two, without a need to search the third level or the memory.

2) Accessing increasingly contagious memory addresses (memGen2)

In all three cache designs, we have a block size of 64 byte. So, after each miss in level one, we load a block of 64 byte which result in a 64 hit.

So, one-level cache has the worst performance since each miss costs 100 cycle to access the memory. However, using multi-level cache increase the core performance of level-one access, which decrease the entire AMAT.

However, the performance decreases after increasing the number of levels because we are adding the cost of searching through them before retrieving those memory addresses from the main memory.

3) Accessing totally random addresses in the memory (memGen3)

The nature of the memGen3 is that it generates an address (% 256 * 1024), so, in the one-level cache with exactly 256 kb, this was exactly all that is needed to fill the cache and then finding all those possible addresses in the cache. Consequently, the one-level cache had the best performance.

However, in the multi-level designs (two and three) the second level in both designs has a size of 256kb, so it takes this amount of misses and access between the first two levels all possible memory addresses could be loaded into the cache.

Also, the performance decreases again in the three-level cache because we are adding the cost of searching in level-three before visiting the memory to retrieve the needed address.

Note that the reasons and the parameter affecting for every number of hits produced by the functions is explained in details in experiment 5 as it depends on the nature of the memGen functions.

Work and collaboration

Ali Ghazal worked on ex 1, 2, 4,

Omer Mohamed worked on ex 3, 5

And we collaborated on working on ex 6 and writing the report