Assembly Project II – Cache Simulation

submitted by:

Alaa A. Anani 900163247

Kirollos N. Sorour 900153483

This is a cache simulator that simulates three types of caches with different mapping methods. (Direct mapped, n-way set associative and Fully Associative). In this project, after implementing the code for the simulation, experiments are done to find out what the properties of a perfect cache are.

MemGen[i] functions are used to generate 32-bit addresses that are used to read and write to the cache and each function has its own concept to represent:

MemGen1: This generating function generates addresses sequentially by incrementing the address value by 1 and taking the mod with respect to the size of a 128 KB main memory. It is easy to exploit the concept of Spatial Locality using this function.

MemGen2: This generating function generates random addresses coming from a 32-KB main memory. It does not represent spatial nor temporal locality exactly. It could represent a program full of jump instructions since it does not have a defined pattern.

MemGen3: This generating function returns (addr += (64 * 1024)) % (512 * 1024), which means that the function has this pattern of generated addresses: 64K, 128K, 192K, 256K,320K, 384K, 448K, 0, and then the sequence starts again. In all cases, the address is more than or equal to the cache sizes we have, so they get mapped to location zero direct in mapping (Which shows that the function will show us the behaviour of direct mapped caches with temporal locality). More will be explained later in the experiment results.

Direct Mapped Cache:

Experiment 1:

The relation between cache performance in terms of hit ratio with the block size of the cache was studied. The cache size is 32 KB and the block size was varied from 2-128 using steps that are powers of 2. The hit ratio was recorded using three types of memory address generators.

Results Experiment 1:

Block size	(Hit Ratio) MemGen1	Hit Ratio MemGen2	Hit Ratio MemGen3
4	0.750000	0.991808	0
8	0.875000	0.995904	0
16	0.937500	0.997952	0
32	0.968750	0.998976	0
64	0.984375	0.999488	0
128	0.992187	0.999744	0

Table 1

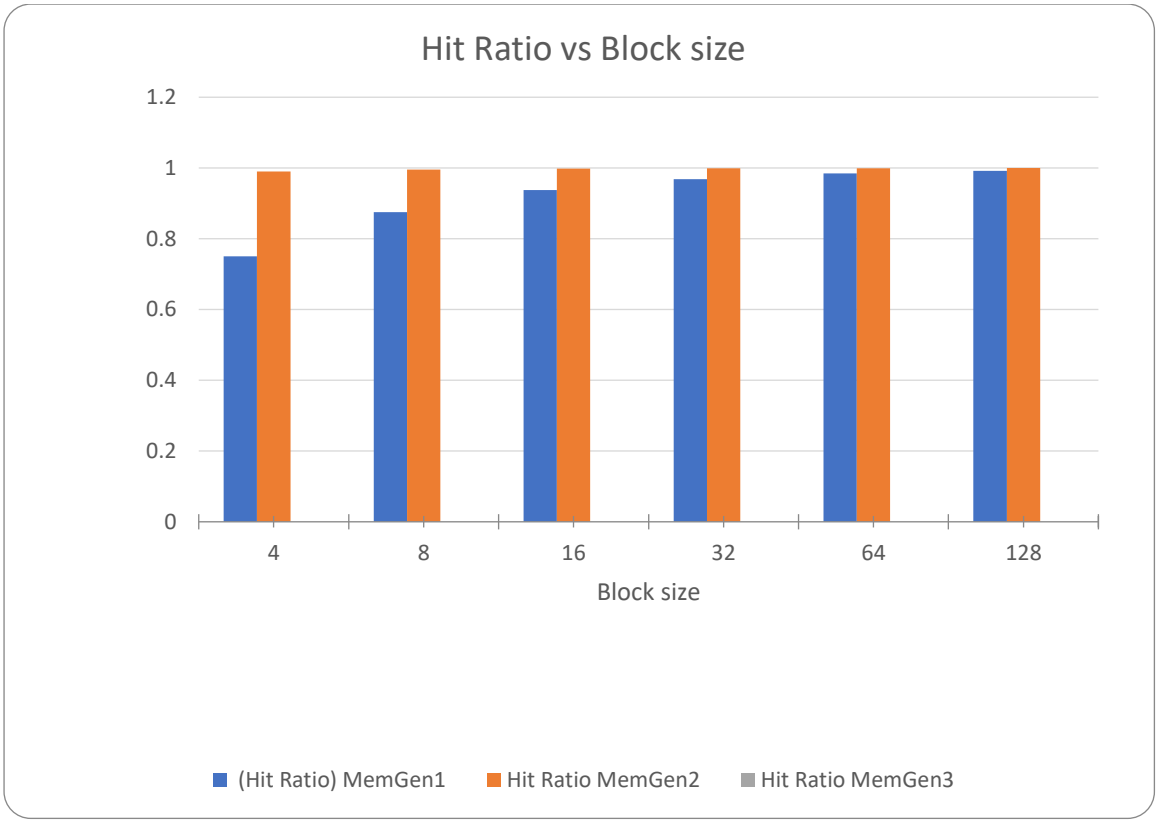


Figure 1: Hit ratio Vs. Block size in a direct mapped cache

- **MemGen1 (Spatial Locality)**
  Increasing block size increases the hit ratio in a direct mapped cache and takes advantage of spatial locality well. Smaller blocks do not take advantage of spatial locality to the maximum as much as large blocks. However, too large blocks means that there are fewer blocks in the cache, thus more conflict misses since more blocks get mapped to the same block.
- **MemGen2 (Random Locality)**
  Increasing block size also increased the hit ratio.
- **MemGen3 (Temporal Locality)**
  Increasing block size did not have an effect on the hit ratio that remained 0 throughout all of the value. The reason why it is 0 is because direct mapped caches have lower associativity since each block in the main memory gets mapped to one and only block in the cache. When faced with a cache generator that generates addresses that will get mapped to the exact same block each time, direct mapping fails and gives 0 hits since the value is replaced each time.

Conclusions from Experiment 1:

Directly mapped caches takes advantage really well of spatial locality specifically if the cache block size is increased, however too large block sizes might lead to increasing the number of conflict misses as well. When it comes to temporal locality, direct mapped caches do not take advantage of it and due to their lower associativity, the hit ratio decreases dramatically.

Experiment 2:

This experiment studies how the write policy affects the AMAT. Write policies are: Write-Through and Write-Back. This is done on a directly mapped cache of size 8KB and a block of size 32 bytes.

AMAT = HitTime + MissRate*MissPenaly.

Results Experiment 2:

Write Policy	AMAT MemGen1	AMAT MemGen2	AMAT MemGen3
Write Through	4.54167	9.33469	11
Write Back	1.62244	9.25333	11

Table 2: Write policy Vs. AMAT

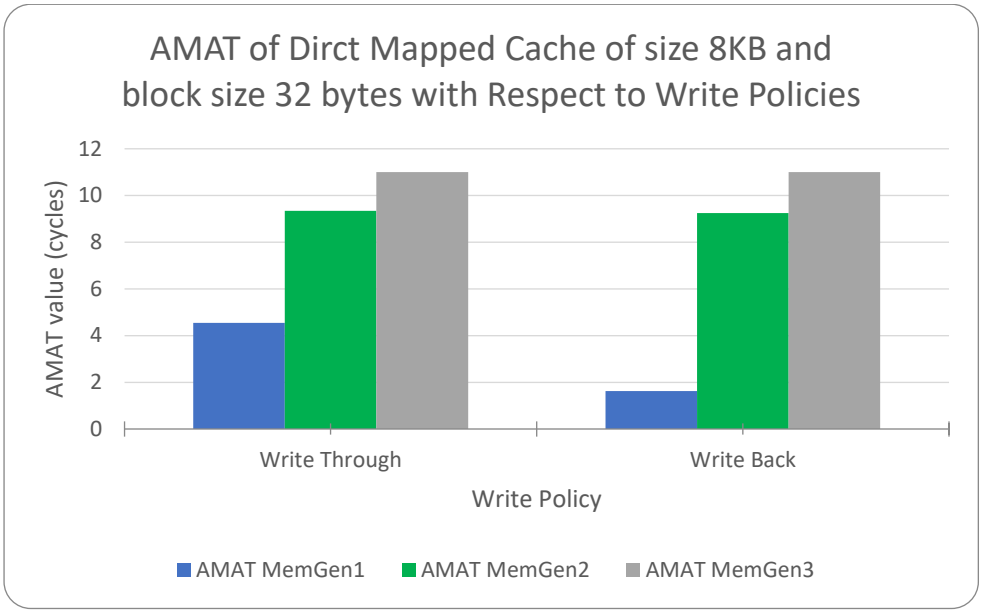


Figure: Write policy Vs. AMAT

Analysis of Results:

The AMAT is higher when the policy is write-through since in write-through whenever it is written to the cache, it is written to the main memory, which takes 20 cycles and results in higher CPI.

However, in write-back a dirty bit is there to indicate whether a clock is replaced or not, if yes, then before it is replaced it is written back to main memory so that the values are consistent between the cache and the main mem.

The AMAT is the lowest in MemGen1 , since the miss rate is also lower since direct mapped caches take advantage of spatial locality.

The AMAT is the highest in MemGen3 since the miss rate is too high since directly mapped caches do not deal well with temporal locality due to their low associativity.

Experiment 3:

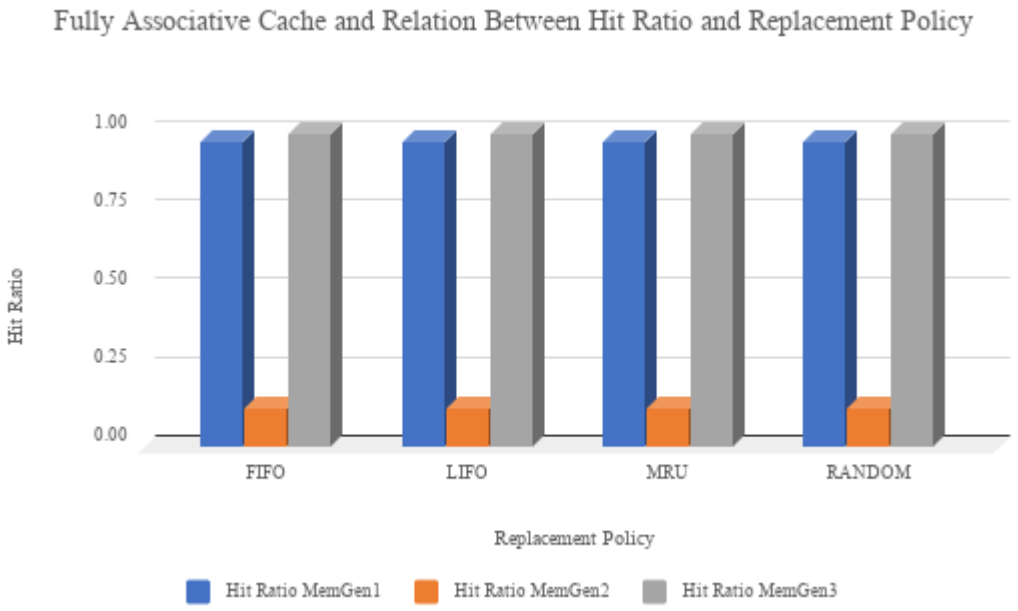
This experiment is on a Fully-Associative cache. It is to study how the hit ratio changes with the replacement policy (block size: 32 bytes, cache size: 4KB).

The policies are: FIFO (First In First Out), LIFO (Last In First Out), MRU (Most Recently Used), and Random.

Results of Exp. 3:

Replacement Policy	Hit Ratio MemGen1	Hit Ratio MemGen2	Hit Ratio MemGen3
FIFO	9.69E-01	0.125356	0.999992
LIFO	9.70E-01	0.125323	0.999992
MRU	9.70E-01	0.125326	0.999992
RANDOM	0.969618	0.125368	0.999992

Table 3.



We can see how close the values are even with different replacement policies. This happens due to the fact that the block size is large and the cache size is also big which would result on a reasonable number of blocks in the cache. since this is a fully associative cache, we can see that MemGen1 generates consequent addresses which makes the block size is the only factor to determine the hit ratio because for each miss you'll have 31 hits no matter the replacement policy is, this ratio will remain the same. for MemGen3, it is obvious that it only accesses 8 addresses; therefore, only 8 blocks would be filled and they will get all the result accesses as hits. However, for MemGen2, the one that accesses memory randomly, we can see that the hit rate is much smaller than the other 2 MemGens since it has a high probability of repeating

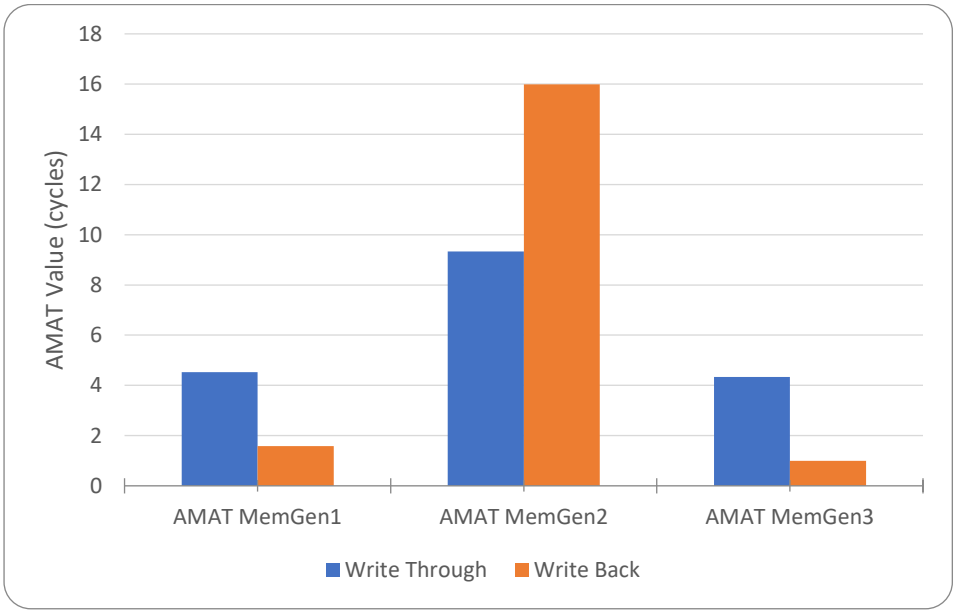
itself which would lead to a higher miss rate in the cache. we can see also that nearly all the replacement policies are the same for MemGen2.

Experiment 4:

AMAT Vs. Write policies in fully associative.

Results of Exp. 4:

Write Policy	AMAT MemGen1	AMAT MemGen2	AMAT MemGen3
Write Through	4.5301	9.33084	4.33339
Write Back	1.58772	15.9926	1.00008



MemGen1:

Write through policy following program has higher AMAT than Write-back, which makes sense since write through results in higher CPI. Adding to that the miss ratio is lower when using MemGen1 for fully associative since it takes advantage of spatial locality.

MemGen2:

The write back AMAT is higher than the write-through AMAT, the assignment is random so the results cannot be explained.

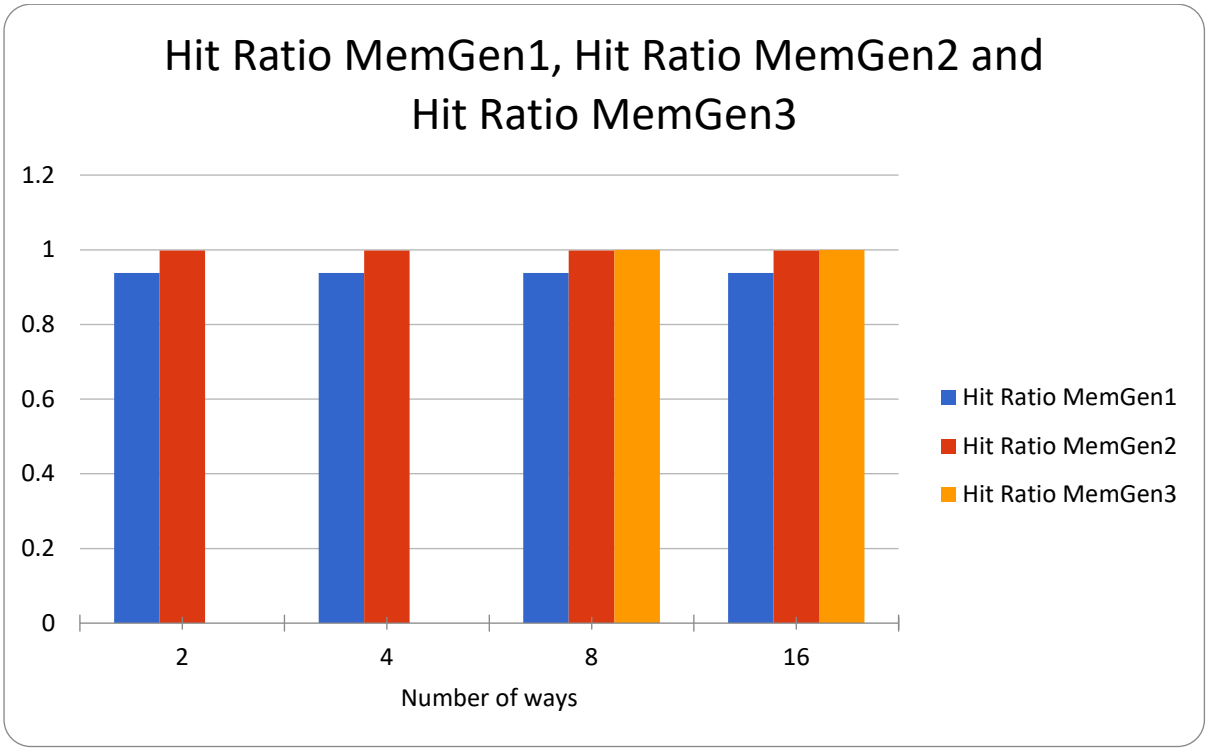
MemGen3:

Write through has higher AMAT than Write-back since in write back, using MemGen3 and taking advantage of temporal locality, writing back does not occur. Unlike in write through, the writing occurs 30% of the time of the program.

Experiment 5:

To Study how the hit ratio changes with the number of ways: 2, 4, 8 and 16 (block size: 16 bytes, cache size: 64KB, FIFO replacement).

Number of ways	Hit Ratio MemGen1	Hit Ratio MemGen2	Hit Ratio MemGen3
2	0.9375	0.997952	0
4	0.9375	0.997952	0
8	0.9375	0.997952	0.99992
16	0.9375	0.997952	0.99992



Analysis of results:

MemGen1:

High hit ratio that remained the same no matter what n is since when addresses are sequentially created and mapped to the sets, one miss is generated when an address is mapped to a block and the rest 15 bytes are hits, and so on moving block by block.

MemGen2:

we can see that the hit ratio is the same since in random accessing the memory, it seems that MemGen2 does not get affected by the number of blocks per set. unlike the theoretical expectation of its behavior since the number of ways is inversely proportional to the number of conflict misses. however, one can explain its behavior due to the fact that the cache size is very big that the number of sets would be great regardless of the number of ways.

MemGen3:

When n was 2 and 4, the hit ratio was 0 since MemGen3 used to create addresses mapped to the exact set each time (Set 0) and it overwrites the blocks each time when capacity misses occur. Now that the number of sets decreased to 512 and 256 and blocks per set increased, MemGen3 sequence 64K, 128K, 192K, 256K,320K, 384K, 448K, 0 repeats itself after filling 8 blocks within a set, thus there is high hit ratio since the first time it fills the first 8 blocks in set 0, the rest of the blocks are hits, and so on.