Lab Assignment – 3 Report

Neelima Rajarikam (nzr0022), Ravinder Putta (rzp0039), Saleel Shetye (sas0087)

Abstract: *Main memory has always been the bottleneck between processor and main memory. To bridge this gap, caches have been introduced. The speed at which the programs are executed on a computer does not only depend on the processor speed and main memory but also on how the data is brought into the cache. The most effective way of assessing the cache performance would be by simulation. Hence in this Lab3 programming assignment, a cache simulator is developed using MARS which is based on MIPS assembly language. This simulation measures the impact of locality, cache size, block size and block replacement policy on the miss rate and access time. After simulation, the results indicate that as cache size increased, miss rate decreased. Also, higher associativity caches offered slightly better performance.*

Introduction: Cache is a high-speed buffer memory between the processor and main memory. It has been introduced to bridge the latency time between the processor and main memory. When processor generates a memory reference it is first searched in the cache. If the processor finds this data in the cache it is a *cache hit* otherwise a *cache miss*. In the event of a miss, the requested block will be brought into the cache to satisfy that request. The bigger the block, better is the locality. During this process, if the cache is full or two different blocks in the memory maps to the same position in the cache (direct-mapped), then the block in the cache must be replaced.

When more than one address in the main memory maps to the same block in the cache, those blocks from memory will be brought into the same position even if there are some vacant entries in the cache. This type of cache is called *Direct-Mapped cache*. In this type, even though other blocks in the cache are empty, main memory still uses the same block every time if their address maps to the same index. As this cache doesn't use empty blocks, it performs badly in the case when the same blocks are replaced frequently. But the performance is slightly better in the presence of locality.

In the direct-mapped cache, since some blocks are unused, Associative-cache (n-way) has been introduced. In n-way associative cache, each set will have 'n' entries and when the same reference is generated from the main memory it can make use of any of the 'n' entries in the set, thereby solving the problem of unused blocks in direct-mapped cache. Here each entry in the set is a block. Since any entry can be used in this cache, it results in better performance. When all the entries in associative cache gets filled up, and a block replacement has to be done, it makes use of block replacing algorithms.

There are several block replacement algorithms like FIFO (First In, First Out), LRU (Least Recently Used), Random Replacement Algorithm etc. No algorithm is suitable for all the conditions. In this assignment, direct-mapped cache with no replacement algorithm and fully associative cache with LRU and FIFO are simulated by changing other cache parameters like block size & cache size to observe the cache behavior. In this paper, some more terms will be introduced below and the methods used to implement this cache simulator using a tool called MARS which is based on MIPS assembly language and observed results like reduction in miss rate with the increase in cache size are discussed in the conclusion.

Body:

To measure the cache performance precisely, it is better to have wide range of references. In this simulation, for each block size and cache size, one million addresses with some spatial and temporal locality are generated by using different cache types (Direct Mapped and Fully Associative) and block replacement algorithms. To generate references with some locality, Markov chain is used. Then the miss rate and effective memory access time are calculated and compared to *computed access time* to evaluate the cache performance.

Markov Chain:

In caches, locality of reference is a phenomenon where the same addresses or values are frequently accessed in the cache. There are two basic types of locality as follows:

<u>Temporal locality:</u> In temporal locality, items accessed recently are likely to be accessed again in the near future.

<u>Spatial locality:</u> In spatial locality, items whose addresses are near to one another tend to be referenced close together in time.

In this Lab-3 assignment, Markov chain is used to generate one million memory references with some spatial and temporal locality. It has two states which works as follows:

New state: In the new state (initial state), the generator will generate a random address A in the range of 0 to $2^{30}-1$, followed by ***m*** aligned addresses starting from A. Here, the length ***m*** is a random number uniformly distributed with mean 256. This address A will then be added to a circular buffer of size 12.

Old state: In the old state, a random address is picked from the circular buffer to produce m aligned addresses starting from A. Here again, the length ***m*** is a random number uniformly distributed with mean 256.

After generating certain number of addresses which will be based on m, the next state is determined using the probabilities shown in figure 1. It implies that, for every two new state executions it will result in 8 old state executions.

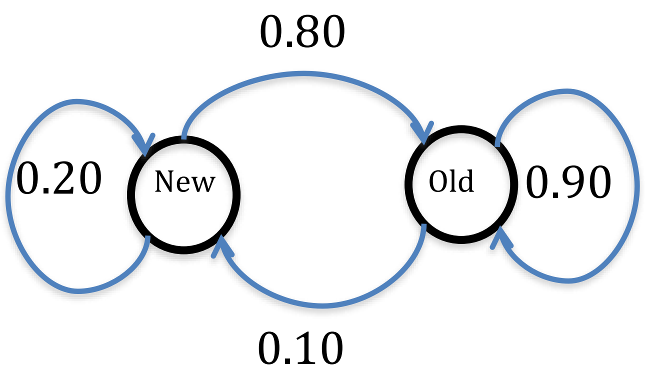


Figure 1. Markov Chain

Cache Replacement Policies:

FIFO: The way this algorithm works is that when the cache is full, it replaces the block that was placed first (not at the 1st position but the one placed in cache initially) in the cache with the desired block, and the next replacement will be the second placed block and so on.

Example:

Memory References: 2, 5.

Existing References in the cache:

Blocks	25	65	80	96
Order	1	2	3	4

After Applying FIFO:

Blocks	2	5	80	96
Order	1	2	3	4

In the above example, the cache contains 4 blocks. “Blocks” represents the block numbers and “Order” indicates the sequence in which the blocks are referenced. After placing the blocks 25, 65, 80 and 96 in the cache the entire memory in the cache is occupied by these blocks. To accommodate the new block 2 which is referenced, one of the blocks in the cache needs to be replaced as it is full. As the block 25 was placed first and is present in the cache from a long time. So it will be replaced by block 2 and 65 will be replaced by block 5.

LRU: Least Recently used algorithm keeps track of all the blocks referenced in the cache and how many times they have been referenced in the past. When a new block is referenced and if the cache is full, the new block will replace a block which has been used least number of times.

Example:

Memory References: 2, 5.

Existing References in the cache:

Reference	25	65	80	96
Order of reference	3	4	1	2

After Applying LRU:

Reference	25	65	2	5
Order of reference	3	4	5	6

In the same example as above the cache containing 4 blocks, “Order of reference” represents the order in which the block is used after fetching it into cache. Once the cache is full, to accommodate the new block 2 which is referenced, block 80 will be replaced as it is not used recently. Next block 5 replaces 96.

Method of Implementation:

In this lab-3 assignment, a cache simulator is developed using a tool called MARS (MIPS Assembler and Runtime Simulator). This is done by generating memory references with some spatial and temporal locality. Locality is ensured in the references using the Markov generator.

After generating enough references all these references are stored in a single array. Now the simulation is run by changing other cache parameters like block size, cache size, cache type and block replacement algorithms to calculate miss rate and effective memory access time.

For each reference, in order to determine if it's a hit or a miss, tag of that reference is compared with the tags of the existing addresses in the cache.

A memory address is a collection of tag, index and offset bits.

Memory address:

Tag	Index	Offset

When offset and index bits are removed from the address, the resulting bits form the tag of that address.

Number of bits in the offset = power of 2 of block size.

Number of bits in index = power of 2 of number of blocks.

Number of blocks in the cache is calculated as follows:

Number of blocks in the cache = Cache size / Block size

Simulating Direct Mapped Cache:

To simulate the direct-mapped cache, an array with the size equal to the number of blocks in the cache called tag array is used. Another array validity array of same size containing the valid bit of each address is used.

For every memory reference generated randomly, the simulator will perform all the above calculations to derive and compare the tag with the existing addresses' tags. If that reference has a validity bit of 1 and the value of the tag matches with the existing address present in the tag array it is considered as a hit and the hit counter will be incremented. Otherwise it is treated as a miss and miss counter will be incremented.

Whenever a miss is encountered, a block needs to be fetched from main memory to cache. Time taken to copy a block from main memory to the cache is called *Miss Penalty*. In this simulation, miss penalty is considered as equal to number of words in a block.

After calculating the number of hits and misses for all the randomly generated references, hit rate and miss rate is calculated as follows:

Hit rate = Number of hits / Total number of references

Miss rate = 1- Hit rate

Calculation of Effective Memory Access Time:

Effective memory access time (ema) is declared as a variable and is used throughout the life cycle of the program.

Whenever a hit is encountered the value of the ema is incremented by a value of *Hit Time* (In this simulation, Hit Time is equal to 1 cc).

Whenever a miss is encountered, first the tag is compared to check whether it is a hit and after confirmation of miss the required block needs to fetched into the cache. So, the value of the ema is incremented by a value of *Miss Time* (Hit Time + Miss Penalty).

The calculated value of Effective memory access time (ema) is compared with the "computed access time"

*Computed access time = Hit time + (Miss rate * Miss Penalty)*

Simulating Fully Associative cache:

In a Fully associative cache, the number of index bits is equal to zero. So to calculate the tag, only offset bits will be removed from the address bits and the resulting value is considered as tag.

Memory address:

Tag	Offset

For fully associative cache along with the calculation of the above parameters of Hit rate, Miss Rate and Miss Penalty, the counter of the memory reference is maintained in a separate array called a *clock array*. If the array is full it represents that the cache is full. When the array is full and if a miss occurs an element in the array needs to be replaced. To handle this scenario different replacement algorithms of LRU (Least Recently Used) and FIFO (First In First Out) are used in this simulation.

FIFO Implementation:

- If a hit is encountered, hit counter is incremented and Hit time is added to effective memory access time.
- If a miss occurs and the cache is not full, the required block will be placed in an empty cell of the tag array and validity bit and clock array will be updated with the respective values of 1 and the counter for that address at the same index. Miss counter is

incremented and Miss Time is added to effective memory access time.

- When a miss occurs and if the cache is full, then the new block will replace the least counter valued element in the clock array and the respective values of tag and validity bit are updated at the same index in the tag and validity arrays. Miss counter is incremented and Miss Time is added to effective memory access time.

LRU Implementation:

- If a hit is encountered, hit counter is incremented and the value in the clock array will be updated with the new counter value of that address. Hit time is added to effective memory access time.
- If a miss occurs and the cache is not full, the required block will be placed in an empty cell of the tag array and validity bit and clock array will be updated with the respective values of 1 and the counter for that address at the same index. Miss counter and effective memory access are updated. Miss counter is incremented and miss time is added to effective memory access time.
- When a miss occurs and if the cache is full, then the new block will replace the least counter valued element in the clock array and the respective values of tag and validity bit are updated at the same index in the tag and validity arrays. Miss counter is incremented and Miss Time is added to effective memory access time.

Simulating Four Way Associative cache:

It is known that increased associativity decreases miss rate. But, with too much associativity performance is effected because of the number of comparators. In this lab-3 assignment, a four-way associative cache is also implemented and then miss rate and effective memory access time are calculated.

The implementation of four-way associative cache is very similar to the fully associative cache discussed above. Since the number of blocks inside each set is 4, number of arrays of tag, clock and validity used are equal to 4.

Once a reference is generated, a hit or a miss is calculated by searching for the tag in all the four tag arrays. The values in the arrays are updated accordingly depending on a miss or a hit. If the cache is full, block replacement algorithms are implemented in the similar manner as implemented in fully associative cache.

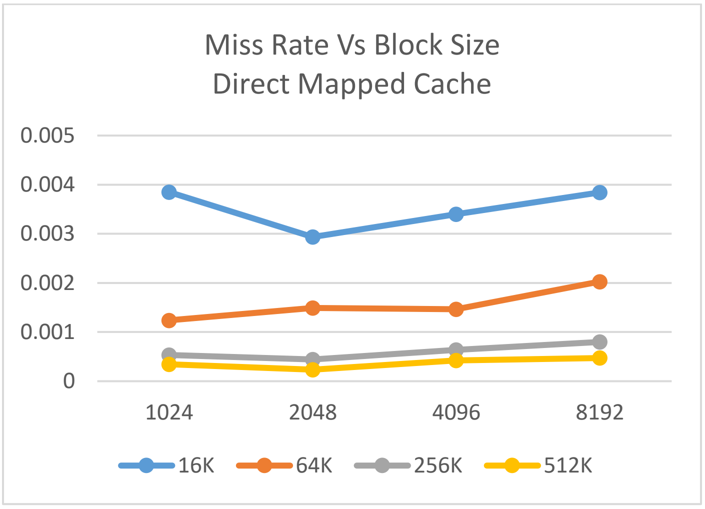
Graphs:

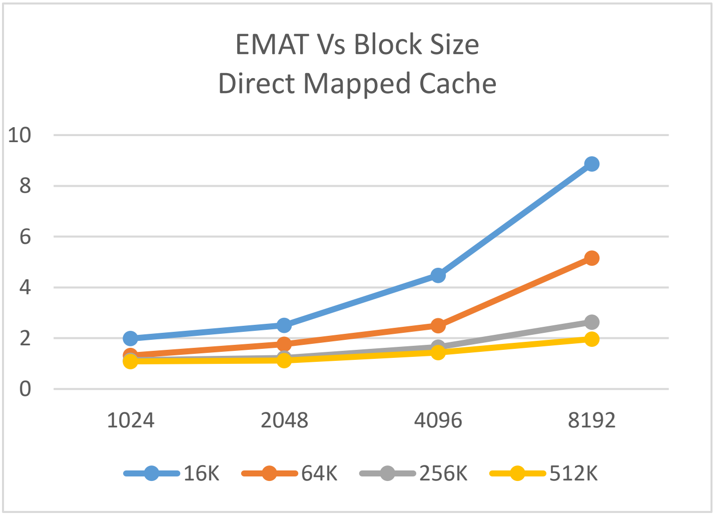
Two graphs of Miss Rate vs. Block Size and Effective Memory Access Time vs. Block Size are plotted for every combination of cache size and block size of different cache types of FIFO and LRU using the values calculated above.

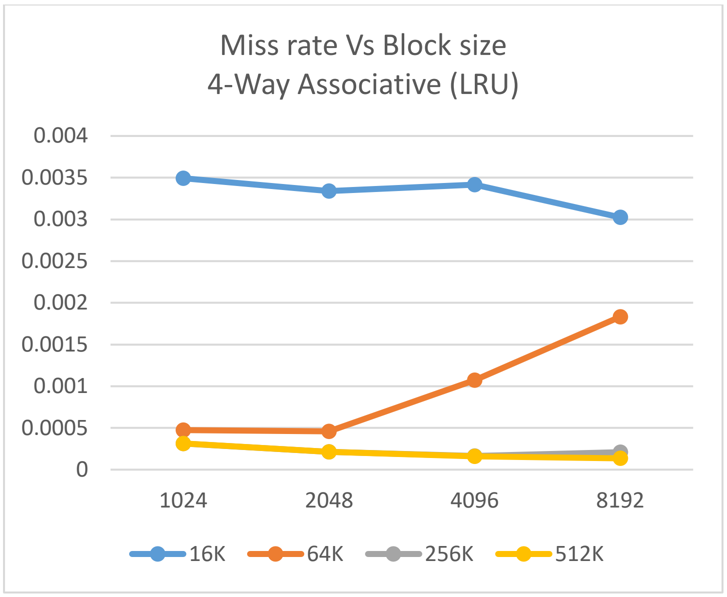
For Miss Rate vs. Block Size graph block size is represented on X-axis and miss rate on Y-axis for every cache size.

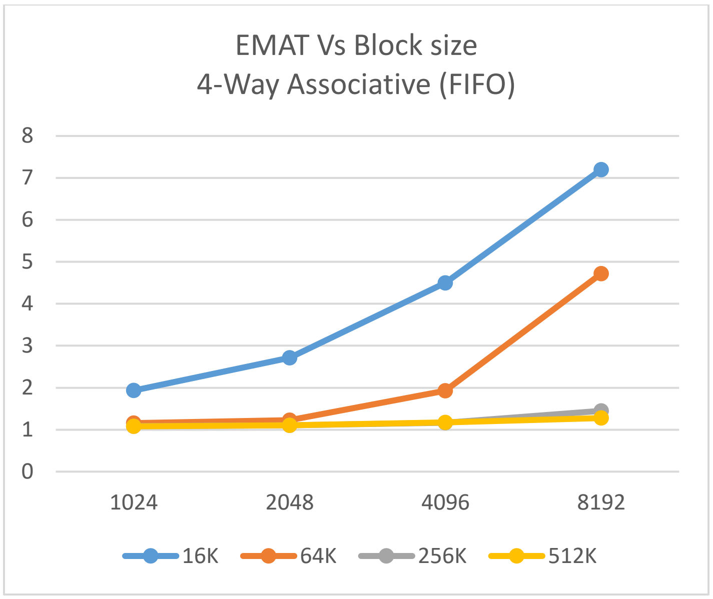
For Effective Memory Access Time vs. Block Size block size is represented on X-axis and Effective memory access time is represented on Y-axis for every cache size.

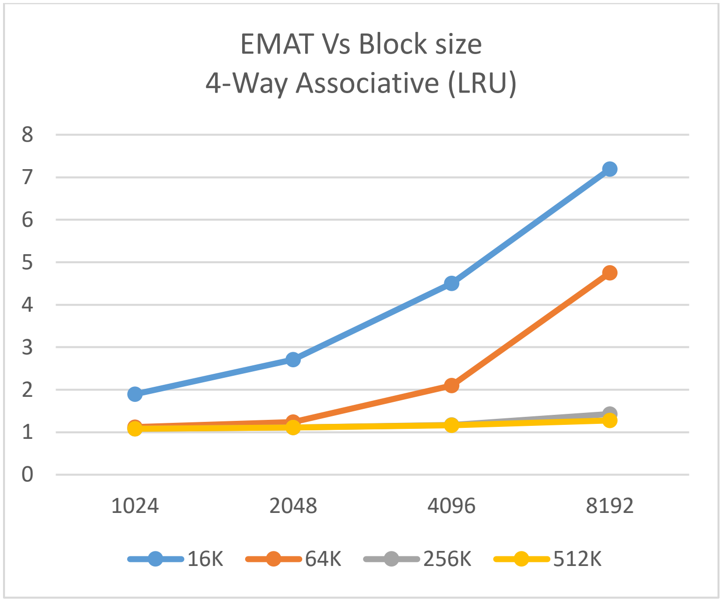
In all the graphs plotted below, it can be observed that miss rate decreased with the increase in cache size. Also, for the same cache size, with the increase in block size memory access time (ema) increased.

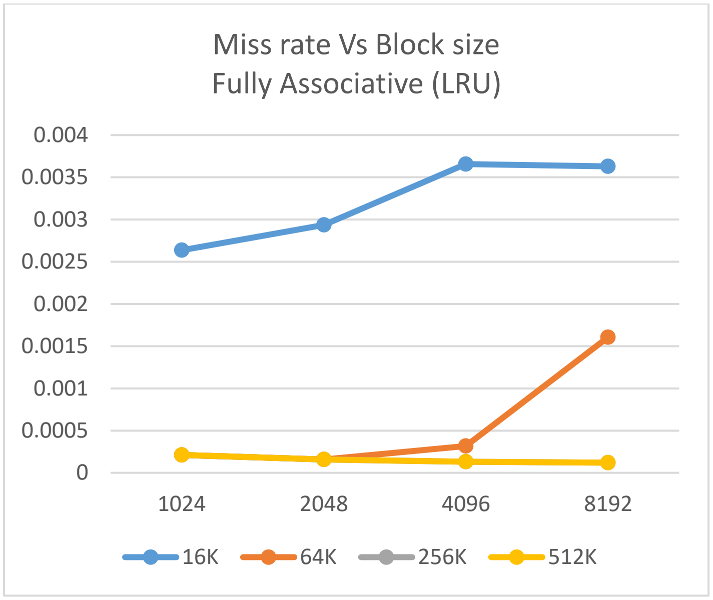


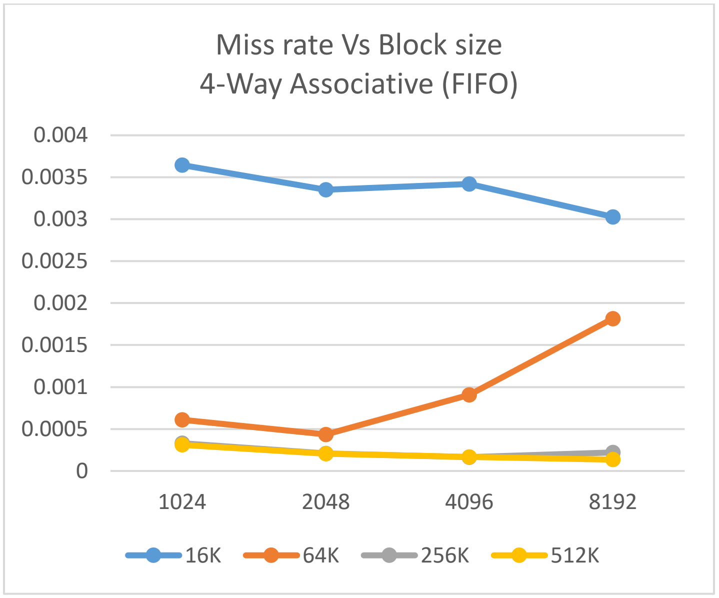


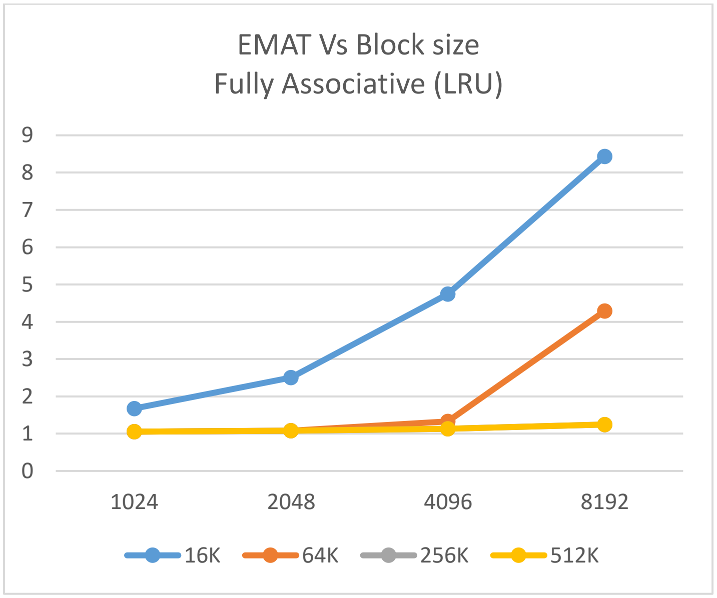

Miss rate Vs Block size
4-Way Associative (LRU)
0.004
0.0035
0.003
0.0025
0.002
0.0015
0.001
0.0005
0
1024
2048
4096
8192
16K
64K
256K
512K

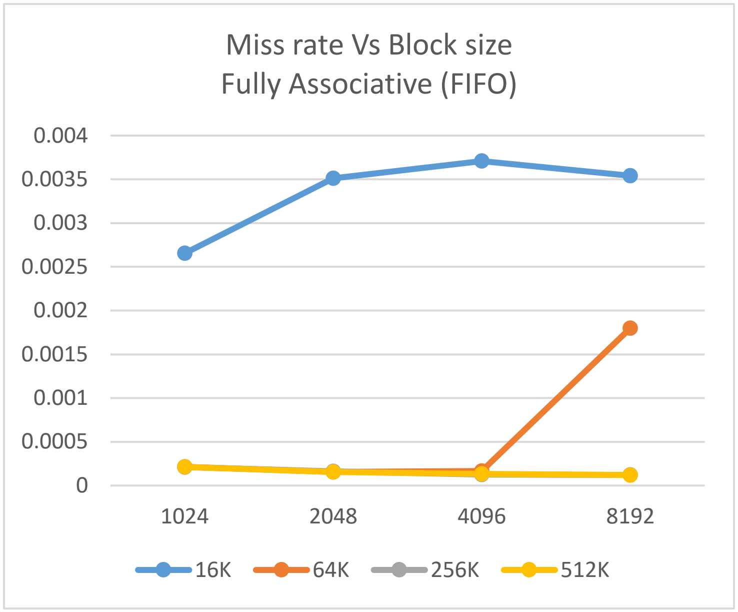

EMAT Vs Block size
4-Way Associative (FIFO)
8
7
6
5
4
3
2
1
0
1024
2048
4096
8192
16K
64K
256K
512K

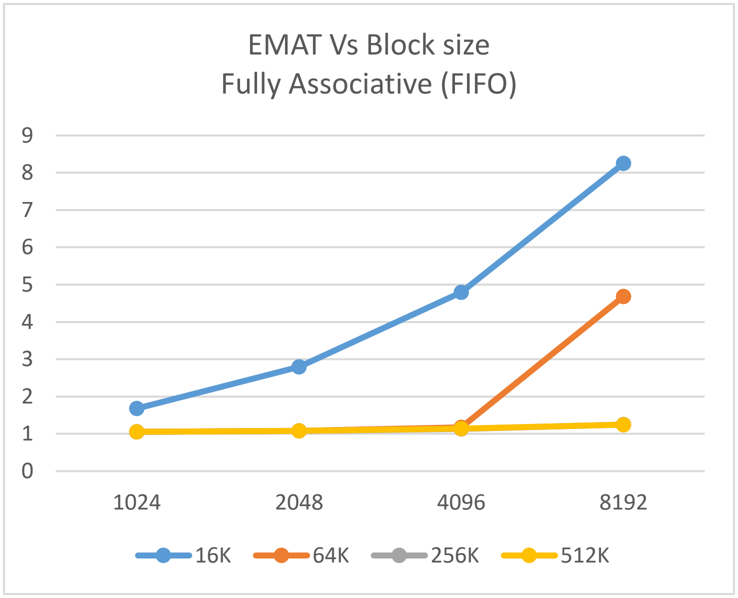

EMAT Vs Block size
4-Way Associative (LRU)
8
7
6
5
4
3
2
1
0
1024
2048
4096
8192
16K
64K
256K
512K


Miss rate Vs Block size
Fully Associative (LRU)
0.004
0.0035
0.003
0.0025
0.002
0.0015
0.001
0.0005
0
1024
2048
4096
8192
16K
64K
256K
512K


Miss rate Vs Block size
4-Way Associative (FIFO)
0.004
0.0035
0.003
0.0025
0.002
0.0015
0.001
0.0005
0
1024
2048
4096
8192
16K
64K
256K
512K


EMAT Vs Block size
Fully Associative (LRU)
9
8
7
6
5
4
3
2
1
0
1024
2048
4096
8192
16K
64K
256K
512K





Conclusion: The purpose of this simulation was to discuss the effect of cache parameters (cache size, block size) on miss rate and effective memory access time for different cache types and block replacement algorithms. Apart from the above parameters locality has been introduced in this lab-3 assignment to assess its impact on miss rate and memory access time (emat). By changing the above parameters, miss rate and emat are assessed through a simulator which is developed using MARS, a MIPS assembly language based tool. Caches are fast and when a good block replacement algorithm is used, it greatly reduces the number of misses and then increases the system's efficiency effectively. An efficient cache has less miss rate and memory access time. From the graphs plotted above, the following facts are observed for all the cache-types (Direct- Mapped, Four-way associative and Fully Associative cache) and block replacement algorithms:

- For the same block size, as the cache size increased, miss-rate decreased.
- As block size increases, the time required to fetch the block into cache (miss penalty) increases. Therefore, for the same cache size, as the block size increased, memory access time increased along with the increase in block size.

Miss rate analysis for all cache types:

As cache size increased, miss rate decreased as compulsory misses decreased. But, it is observed that there are slight variations for different cache types as mentioned below:

- For direct mapped cache, as block size increased with in the same cache, miss rate increased because of capacity and conflict misses.

- For four-way associative cache, as block size increased with in the same cache, miss rate increased slightly but not as much as in the case of direct - mapped cache. It is because conflict misses are reduced but not eliminated completely.

- For fully associative cache, as block size increased with in the same cache, the miss rate is not increased because conflict misses are eliminated completely.

While observing the results of this simulation, the case of 4-way associative cache with 1KB block size is ignored as it results in only 2 blocks which cannot form at least one set.

Therefore, it is observed from above graphs and the simulated data, the overall miss rate reduced because of the high locality and the better performance is observed in the case of higher associative cache. LRU and FIFO offered same performance for n-way associative caches. Also, for Directed-Mapped cache and n-way associative cache (using different replacement algorithms), similar performance is observed because of high locality.

Note: The detailed values of the simulation are attached in the excel "Cache Analysis Values.xls"