REPORT:

Measure the latency:

To measure the block size and associativity we use arrays.

For better precision of time measurement we use assembly instructions like (rdtsc)

And instructions like lfence such that we don't want the insruction to get executed via cache

Actual Specifications of L1 cache present:

```
saisiddarth@SaiSiddarth:~$ getconf -a | grep CACHE
LEVEL1_ICACHE_SIZE                 32768
LEVEL1_ICACHE_ASSOC                8
LEVEL1_ICACHE_LINESIZE             64
LEVEL1_DCACHE_SIZE                 49152
LEVEL1_DCACHE_ASSOC                12
LEVEL1_DCACHE_LINESIZE             64
LEVEL2_CACHE_SIZE                  1310720
LEVEL2_CACHE_ASSOC                 20
LEVEL2_CACHE_LINESIZE              64
LEVEL3_CACHE_SIZE                  8388608
LEVEL3_CACHE_ASSOC                 8
LEVEL3_CACHE_LINESIZE              64
LEVEL4_CACHE_SIZE                  0
LEVEL4_CACHE_ASSOC                 0
LEVEL4_CACHE_LINESIZE              0
```

The code which we will use in Ubuntu to find Specifications of cache is:

getconf -a | grep CACHE

Reverse Engineering of L1 Cache Memory

Identify the Cache Size:

Consider the cache size as empty initially and initiate a LOAD request to &array[0], which brings a block of data into the cache.As per your assumption the block size can be either size of 32B or 64B.Lets us find out the block size using Reverse engineering

Explanation or method used:

As we initiated a LOAD request to &array[0].That implies the data in the vicinity of array[0] will also be bought into the cache.So the access latency for the neighboring addresses of array[0] should be less as it is a cache hit as they are already present in the cache so we need to compare access latency for array[8] & array[16] to determine whether the block size is 32B or 64B.

By observing the graphs we can say that the access latency between array[7] & array[8] not so high when compared with array[15] & array[16]. That mean there is a miss while LOADing the array[16] as it is not present in the L1 cache so the processor should go to main memory to get the data of array[16].

Justification:

In the similar way we can see the difference between the access latency of array[31] & array[32].From this we can say that from index 0 to 15 gets fit in a block of cache.That total of 16 integers can fit in a block of cache implies

Total no of integers=16

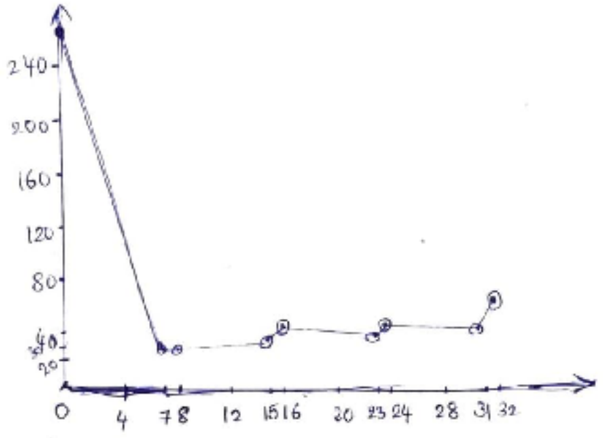
Size of each integer=32 bits=4 bytes

Size of cache block=16*4 bytes

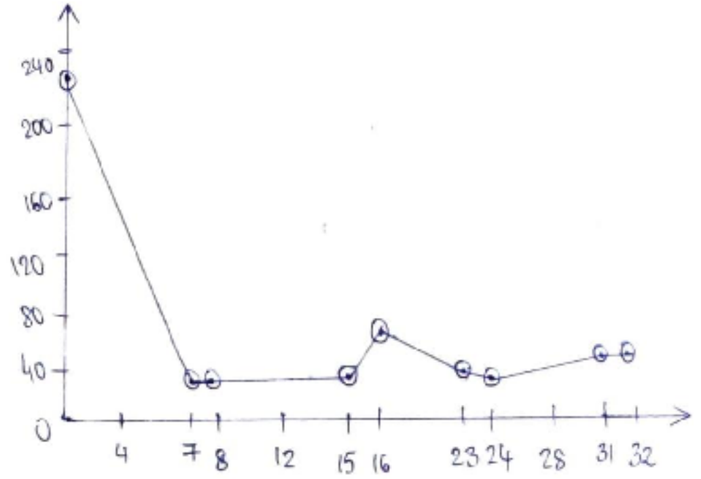
=64B

Therefore by using the cache access latency we can conclude that block size of my processor is 64B.

Plots:




```
Cache Access latency for a[0] 252
Cache Access latency for a[7] 29
Cache Access latency for a[8] 28
Cache Access latency for a[15] 30
Cache Access latency for a[16] 35
Cache Access latency for a[23] 33
Cache Access latency for a[24] 35
Cache Access latency for a[31] 34
Cache Access latency for a[32] 51
```




```
Cache Access latency for a[0] 224
Cache Access latency for a[7] 28
Cache Access latency for a[8] 28
Cache Access latency for a[15] 28
Cache Access latency for a[16] 53
Cache Access latency for a[23] 37
Cache Access latency for a[24] 33
Cache Access latency for a[31] 37
Cache Access latency for a[32] 37
```

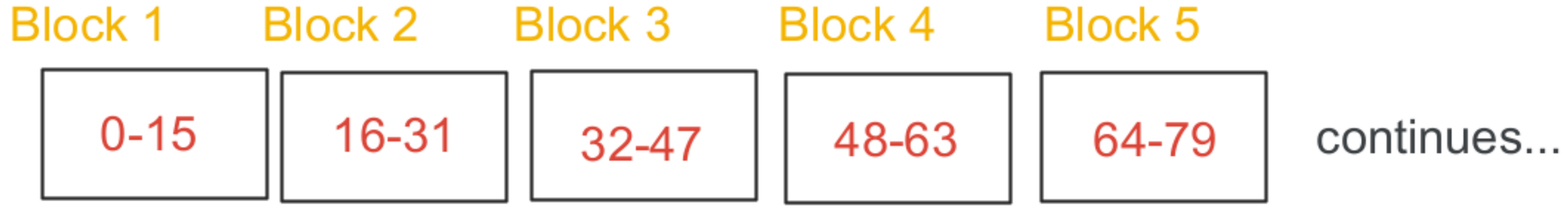
By above graphs and calculating the average for two graphs we can conclude that the Cache Block size is 64Bytes or 64B.

Identify the Associativity of the cache:

From the above process we found that block size as 64B.Now we need to find associativity of cache.Similar to the above method we need to LOAD different positions of array such that we should get a significant amount of difference between cache latency access.Associativity of cache can be 4 or 8.All we need is to generate LOAD requests with addresses such that all these addresses bring cache blocks that belong to the same set

Explanation or method used:

As we know that block size is 64B.Total of 16 integers get assigned into a block. The sequence of indices in which they get assigned in the cache is as follows



From the graphs clearly we can observe the cache latency access is higher for array[128] that means the cache block which consists of array[128] is evicted by a newly brought-in cache block. In such a case, the number of blocks brought into the cache set is one more than the associativity of the cache,.As more latency access time is performed to evicted the least recently used block in the cache set(using Random, LRU, Pseudo-LRU methods) and bring in new block into the cache set.

The same scenario happens with array[256] so we can infer the cache associativity from this observation.

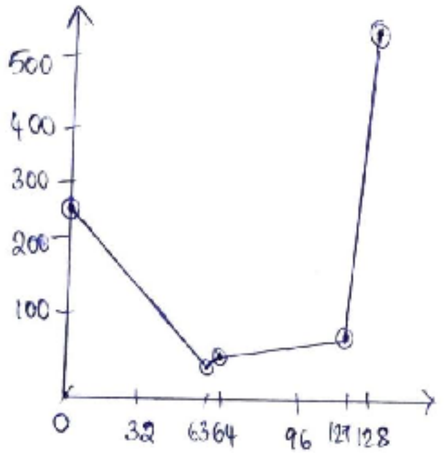
Justification:

While accessing all the blocks.If the access latencies are almost the same, indicating that the cache blocks are present in the cache set, we try to bring in a new cache block to the same cache set and repeat the process. At any time, if we observe high access latency when we access a cache block, indicating that the cache block is evicted by a newly brought-in cache block.

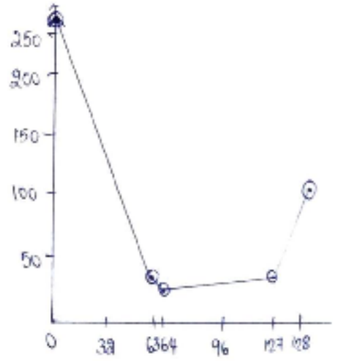
In this case array[128] is getting higher latency.Now lets find associativity:

Total number of integers in the set=128
Size of each integer datatype=32 bits=4 bytes
Total size of each set=128*4B=512B
As we know each cache block size=64B
Associativity of the cache is=512B/64B=8.

Plots:




```
Cache Access latency for a[0] 256
Cache Access latency for a[63] 34
Cache Access latency for a[64] 36
Cache Access latency for a[127] 51
Cache Access latency for a[128] 540
```




```
Cache Access latency for a[0] 275
Cache Access latency for a[63] 37
Cache Access latency for a[64] 32
Cache Access latency for a[127] 35
Cache Access latency for a[128] 103
```

Therefore from above two graphs and avg values we can conclude that the associativity of my cache is 8