Assignment I

Problem Bank 37

Assignment Description:

The assignment aims to provide deeper understanding of cache by analysing its behaviour using cache implementation of CPU- OS Simulator. The assignment has three parts.

- Part I deals with Cache Memory Management with Direct Mapping
- Part II deals with Cache Memory Management with Associative Mapping
- Part III deals with Cache Memory Management with Set Associative Mapping

Submission: You will have to submit this documentation file and the name of the file should be GROUP-NUMBER.pdf. For Example, if your group number is 1, then the file name should be GROUP-1.pdf.

Submit the assignment by **22nd December 2021, through canvas only**. File submitted by any means outside CANVAS will not be accepted and marked.

In case of any issues, please drop an email to the course TAs, Ms. Michelle Gonsalves (michelle.gonsalves@wilp.bits-pilani.ac.in).

Caution!!!

Assignments are designed for individual groups which may look similar and you may not notice minor changes in the assignments. Hence, refrain from copying or sharing documents with others. Any evidence of such practice will attract severe penalty.

Evaluation:

- The assignment carries 13 marks
- Grading will depend on
    - Contribution of each student in the implementation of the assignment
    - **Plagiarism or copying will result in -13 marks**

**************************FILL IN THE DETAILS GIVEN BELOW****************

Assignment Set Number: 37

Group Name: 117

Contribution Table:

Contribution (This table should contain the list of all the students in the group. Clearly mention each student's contribution towards the assignment. Mention "No Contribution" in cases applicable.)

Sl. No.	Name (as appears in Canvas)	ID NO	Contribution
1.	Sai Prabhanjan Reddy Kalluru	2021FC04136	100%
2.	Nayak Vinayak Vinod	2021FC04135	100%
3.	Niyati Gangwal	2021FC04140	100%

Resource for Part I, II and III:

- Use following link to login to "eLearn" portal.
  - https://elearn.bits-pilani.ac.in
- Click on "My Virtual Lab – CSIS"
- Using your canvas credentials login in to Virtual lab
- In "BITS Pilani" Virtual lab click on "Resources". Click on "Computer Organization and software systems" course.
  - Use resources within "LabCapsule3: Cache Memory"

Code to be used:

The following code written in STL Language, implements Sorting of elements in an array using Selection Sort technique.

```
program SelectionSort
VAR a array(10) INTEGER
VAR len byte
VAR i byte
VAR j byte
VAR p byte
VAR q byte
VAR x byte
VAR n byte
a(1)=15
a(2)=20
a(3)=8
a(4)=80
a(5)=30
```

```
a(6)=35
len = 6

for n = 1 to len
        writeln("num",a(n))
next

for i = 1 to len
        for j = i+1 to len

                p = a(i)
                q = a(j)
                if p > q then
                        x = p
                        a(i) = q
                        a(j) = x
                end if
        next
next
writeln("Sorted Array in ascending order")
for n = 1 to len
        writeln("num",a(n))
next
end
```

General procedure to convert the given STL program into ALP:

- Open CPU OS Simulator. Go to **advanced tab** and press **compiler** button
- Copy the above program in **Program Source** window
- Open **Compile** tab and press **compile** button
- In **Assembly Code,** enter **start address** and press **Load in Memory** button
- Now the assembly language program is available in CPU simulator.
- Set speed of execution to **FAST.**
- Open I/O console
- To run the program press **RUN** button.

General Procedure to use Cache set up in CPU-OS simulator

- After compiling and loading the assembly language code in CPU simulator, press “Cache-Pipeline” tab and select cache type as “both”. Press “SHOW CACHE” button.
- In the newly opened cache window, choose appropriate cache Type, cache size, set blocks, replacement algorithm and write policy.

Part I: Direct Mapped Cache

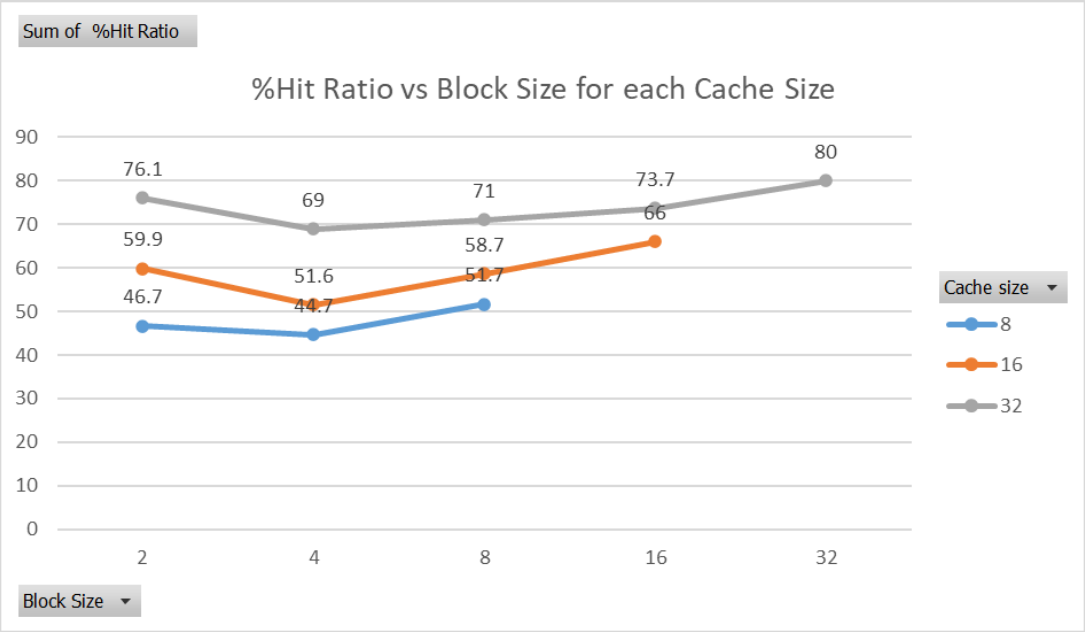
a) Execute the above program by setting block size to 2, 4, 8, 16 and 32 for cache size = 8, 16 and 32. Record the observation in the following table.

Block Size	Cache size	# Hits	# Misses	% Miss Ratio	%Hit Ratio
2	8	347	396	53.3	46.7
4		332	411	55.3	44.7
8		384	359	48.3	51.7
2	16	445	298	40.1	59.9
4		383	360	48.4	51.6
8		436	307	41.3	58.7
16		490	253	34	66
2	32	565	178	23.9	76.1
4		512	231	31	69
8		527	216	29	71
16		547	196	26.3	73.7
32		594	149	20	80

Note:

1. The colour formatting for miss % ratio is lesser the miss is green and vice versa for red
2. The colour formatting for hit % ratio is more the hits is green and vice versa for red

b) Plot a single graph of Cache hit ratio Vs Block size with respect to cache size = 8, 16 and 32. Comment on the graph that is obtained.



Observations/Comments:

1. The performance or hit ratio is dependent on the cache size.
2. Greater the cache size, the hit ratio levels are high, and the influence of block size is minimal
3. But the combination of block size 2 with any cache is having hit ratio close to 75th percentile in the result of experimentation w.r.t that cache
4. The efficient combination is always where cache size = block size and this is practical since the smaller block size will lead to have higher replacement rate

c) Fill the below table and write a small note on your observation from **data cache**.
   - Block Size =4
   - Cache Size = 16
   - Cache Type = Direct Mapped

Addresses	Data	Miss (%)
0008	0C	48.4
0009	02	48.4
0010	00	48.4
0011	1F	48.4
0028	00	48.4
0029	23	48.4
0030	02	48.4
0031	00	48.4
0032	50	48.4
0033	00	48.4
0034	00	48.4
0035	00	48.4
0068	00	48.4
0069	00	48.4
0070	07	48.4
0071	00	48.4

Observations:

1. The number of address lines correspond to the cache size = 16
2. The missing block size is equal to four consecutive addresses which is block size
3. With iterations, the miss addresses are different between each iteration
4. Observed the byte addressable memory limit as 0-255 in line with Data from 00-FF this is because of block size is 4, word bits are 2

Part II: Associative Mapped Cache

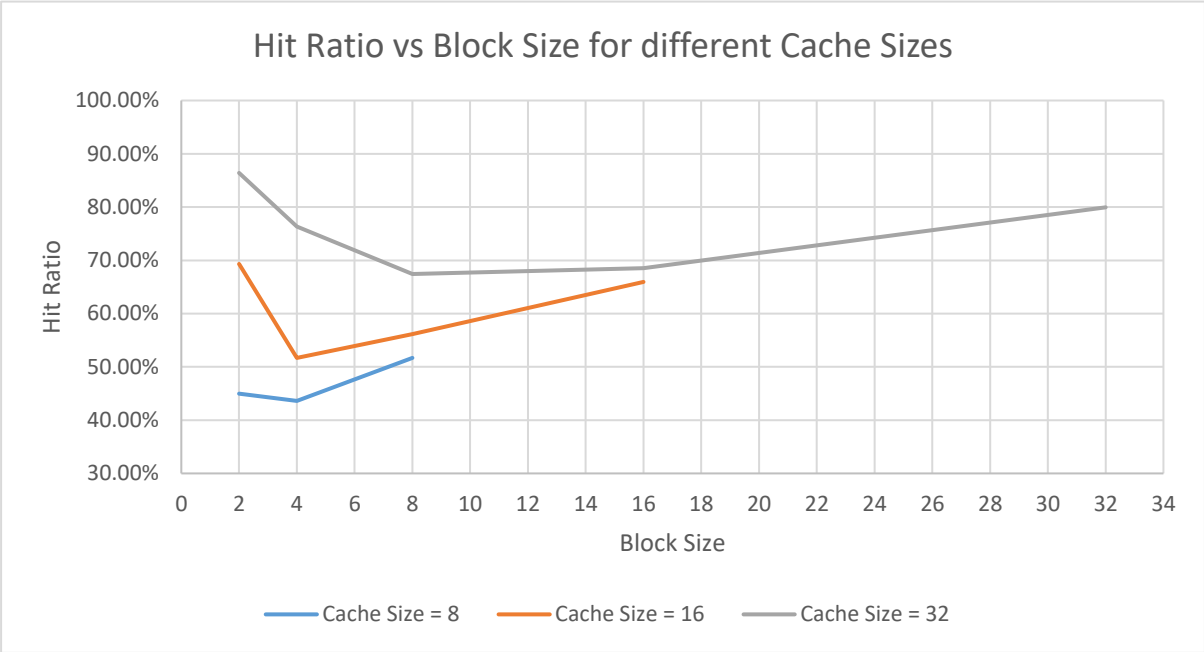
A) Execute the above program by setting block size to 2, 4, 8, 16 and 32 for cache size = 8, 16 and 32. Record the observation in the following table.

LRU Replacement Algorithm					
Block Size	**Cache Size**	**# Hits**	**# Misses**	**% Miss Ratio**	**% Hit Ratio**
2	8	334	409	55.05%	44.95%
4		324	419	56.39%	43.61%
8		384	359	48.32%	51.68%
2	16	515	228	30.69%	69.31%
4		384	359	48.32%	51.68%
8		417	326	43.88%	56.12%
16		490	253	34.05%	65.95%
2	32	642	101	13.59%	86.41%
4		567	176	23.69%	76.31%
8		501	242	32.57%	67.43%
16		509	234	31.49%	68.51%
32		594	149	20.05%	79.95%

Note:

- The colour coding for miss and hit ratio columns is based on the hit ratio column
- Highest hit ratio is darkest shade of blue.
- Lowest hit ratio is darkest shade of red.
- All other values are in between these two bounds with white being the moderate.

B) Plot a single graph of Cache hit ratio Vs Block size with respect to cache size = 8, 16 and 32. Comment on the graph that is obtained.



For the given program i.e., Selection Sort; executing the program on CPU-OS Simulator with an LRU replacement algorithm and only using the d-cache (caching only data) gave the results as plotted above. The following observations can be made from the graph.

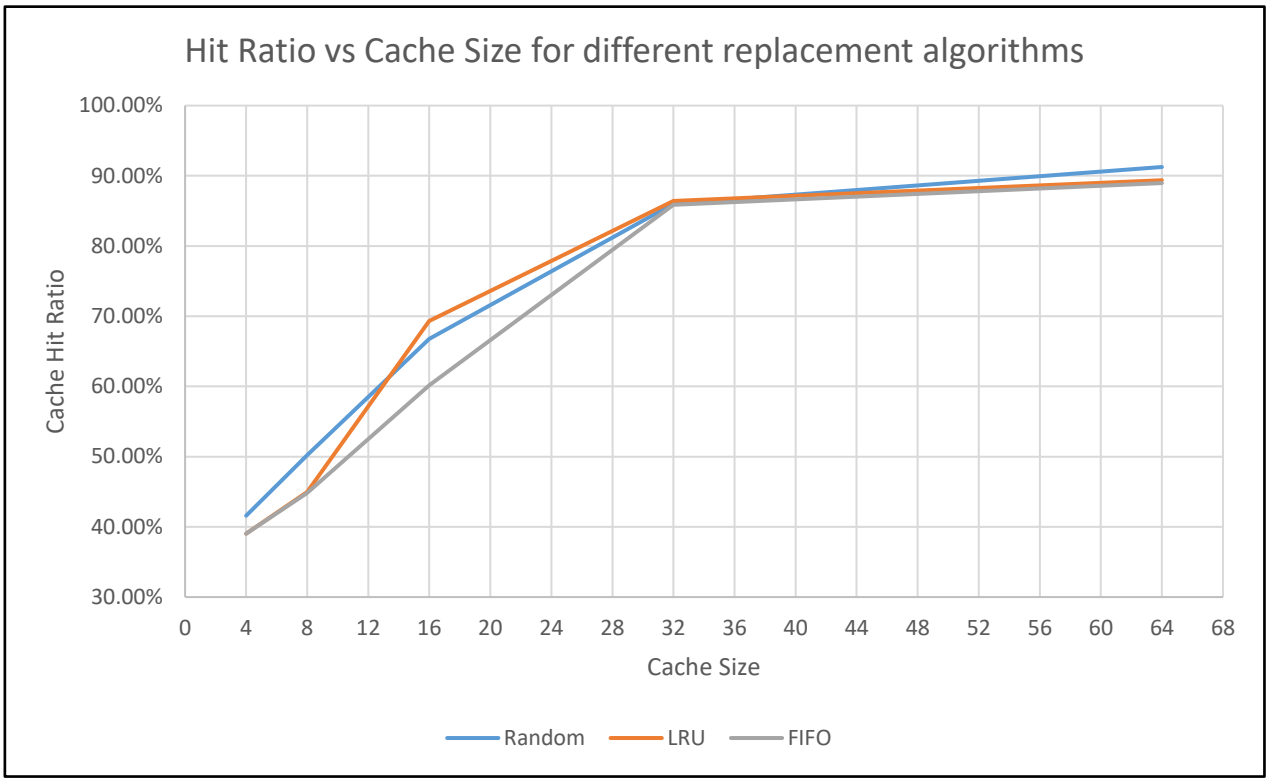
- ***Irrespective of the block size, increasing cache size gives higher hit ratio.***
- There is a slight variation in hit ratio for a given cache size when we change block sizes, but it seems that block size = cache size or block size = 2 works better than any other intermittent block size.
- ***In general, increasing block size for a given cache size improves the hit ratio. This could be observed post block size = 4 in cache sizes = [8,16] line plots and post block size = 8 in cache size = 32 so, line plot respectively.***
- Theoretically, increasing block size means spatial locality is better utilized and we generally see an improvement in the hit ratio for algorithms which use spatial locality the most. The initial decrease in hit ratio for cache sizes of 16 and 32 is an interesting observation and we're not yet into sorting algorithms, so I can't comment on the same yet; but that is something we might be able to answer after sorting algorithms ae covered in DSAD. For now, I can only think that when block size = 2, we are loading two elements from main memory and the first out of those two is always the lowest in that group given the sequential nature of the array [15, 20 | 8, 80 | 30, 35] and selection sort continually tries to find minimal element iteratively from what we could gather from internet; so maybe it has something to do with that.

C) Fill up the following table for three different replacement algorithms and state which replacement algorithm is better and why?

Replacement Algorithm: Random				
Block Size	**Cache Size**	**# Misses**	**# Hits**	**Hit Ratio**
2	4	434	309	41.59%
2	8	370	373	50.20%
2	16	247	496	66.76%
2	32	104	639	86.00%
2	64	65	678	91.25%
Replacement Algorithm: FIFO				
Block Size	**Cache Size**	**# Misses**	**# Hits**	**Hit Ratio**
2	4	453	290	39.03%
2	8	410	333	44.82%
2	16	296	447	60.16%
2	32	105	638	85.87%
2	64	82	661	88.96%
Replacement Algorithm: LRU				
Block Size	**Cache Size**	**# Misses**	**# Hits**	**Hit Ratio**
2	4	453	290	39.03%
2	8	409	334	44.95%
2	16	228	515	69.31%
2	32	101	642	86.41%
2	64	79	664	89.37%

Note:
While random seems to be performing at par with the other two replacement algorithms, LRU in my opinion is the algorithm to go with. I have provided my reasoning for the same after plotting the graph below.

D) Plot the graph of Cache Hit Ratio Vs Cache size with respect to different replacement algorithms. Comment on the graph that is obtained.



It could be seen that for the given program, all three algorithms are performing on a comparatively close scale. But still, we can observe that

- The choice of replacement algorithm has a reasonably significant (if not a very huge) impact on the cache hit ratio when the cache size is very small.
  - In that too, Random seems to perform better on average than LRU & FIFO for small cache sizes (observation is from multiple iterations).
  - For small size cache, random replacement algorithm for programs like selection sort seem to be performing well; however, it's not deterministic in nature so it's possible that we might get lower hit ratio in some cases.
  - ***The performance improvement of random over LRU strictly speaking is not too high. I would therefore prefer LRU knowing it's deterministic nature even for smaller cache sizes.***
- As the cache size increases, the choice of replacement algorithm seems to no longer make a big impact on the cache hit ratio. In such cases where cache size is high, we can consciously try to avoid random and go with either of the other two to avoid any stochasticity and go for either LRU or FIFO since the performance is at par if not significantly lower than Random replacement algorithm.
- ***Logically it makes sense to use LRU as it will replace those blocks which have been in the cache longest without having been accessed; irrespective of whatever the cache size be.***

Part III: Set Associative Mapped Cache

Execute the above program by setting the following Parameters:

- Number of sets (Set Blocks): 2 ways
- Cache Type: Set Associative
- Replacement: LRU/FIFO/Random

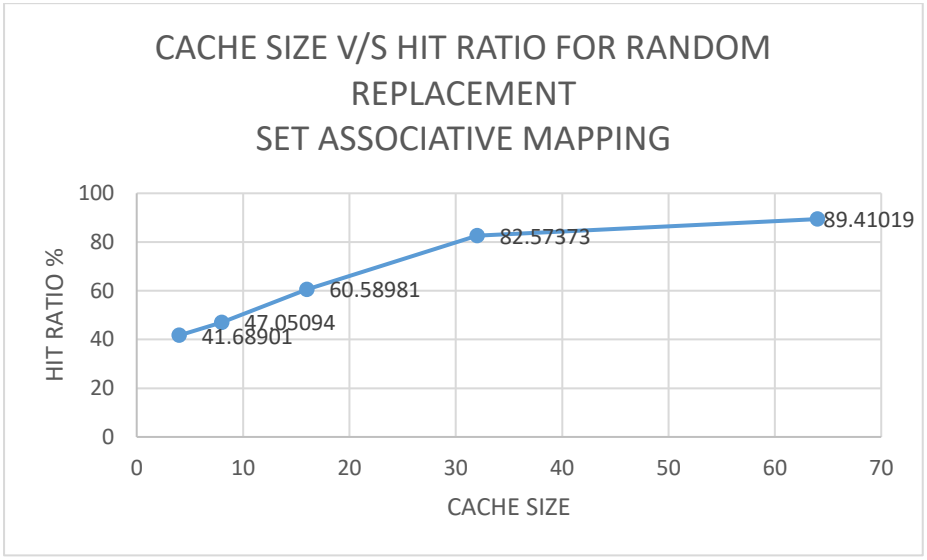
a) Fill up the following table for three different replacement algorithms and state which replacement algorithm is better and why?

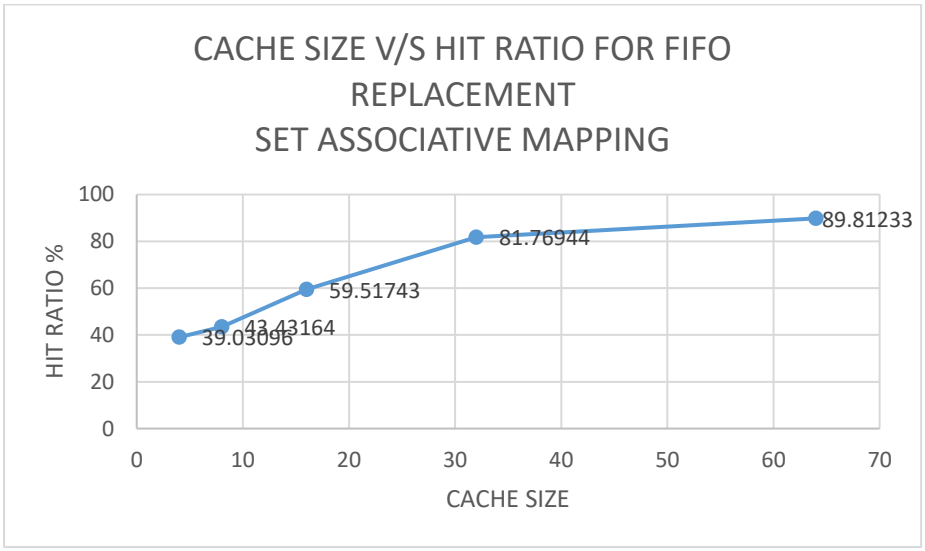
Replacement Algorithm: Random				
Block Size	Cache size	Miss	Hit	Hit ratio%
2	4	435	311	41.69
2	8	395	351	47.05
2	16	294	452	60.59
2	32	130	616	82.57
2	64	79	667	89.41
Replacement Algorithm: FIFO				
Block Size	Cache size	Miss	Hit	Hit ratio%
2	4	453	290	39.03
2	8	422	324	43.43
2	16	302	444	59.52
2	32	136	610	81.77
2	64	76	670	89.81
Replacement Algorithm: LRU				
Block Size	Cache size	Miss	Hit	Hit ratio%
2	4	456	290	38.87
2	8	409	337	45.17
2	16	279	467	62.60
2	32	131	615	82.44
2	64	76	670	89.81

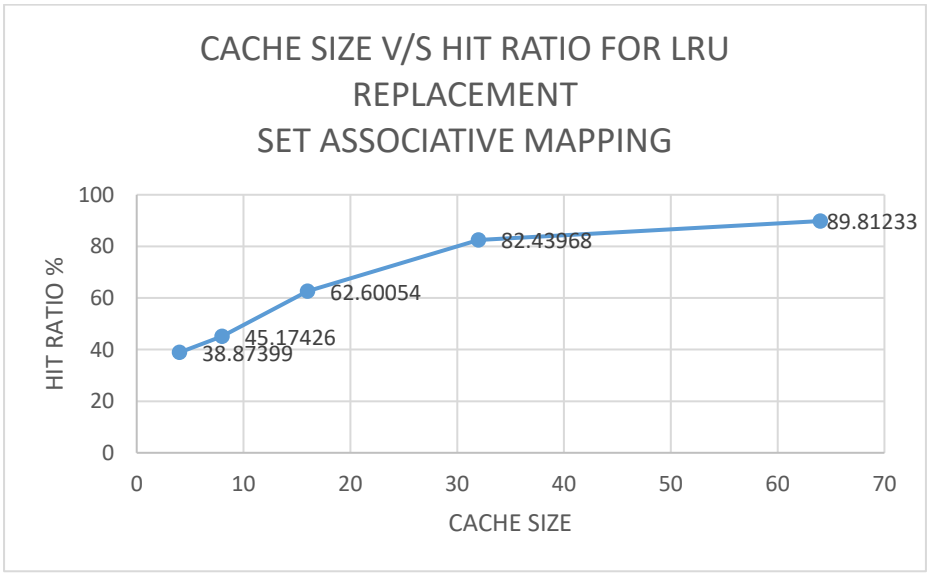
Observations:

1. From the results obtained, can say that Random algorithm works better for smaller cache size i.e., 4 & 8. Whereas there is almost similar performance of all three algorithms for cache size higher than 8.
2. But in Overall, for selection sort use case (current problem), temporal locality is important so in theory, LRU works on the notion of keeping temporal locality so reliance on it is efficient over FIFO for the problem in hand.

Plot the graph of Cache Hit Ratio Vs Cache size with respect to different replacement algorithms. Comment on the graph that is obtained.







Observations:

1. Significantly better performance of Random Algorithm for smaller cache size. And comparably similar performance of all 3 as the size increases.
2. Couple of observations are already listed above for the tabular answer (1)

c) Fill in the following table and analyse the behaviour of Set Associate Cache. Which one is better and why?

Replacement Algorithm: LRU				
Block Size, Cache size	Set Blocks	Miss	Hit	Hit ratio
2, 64	2 – Way	76	670	89.81233
2, 64	4 – Way	65	681	91.28686
2, 64	8 – Way	62	684	91.68901

Observations:

1. 8-way block size certainly have a better performance as it has higher associativity and are more flexible to allocate space for data in cache.
2. For a set associate cache unlike direct mapped, the programme must search all the address lines within a set so this combined approach will give us best of both from direct map and fully associate cache techniques.
3. Also, can be observed the more lines per set, the performance is better.