Assignment I

Problem Bank 21

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 **20th June 2022** **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.

  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. Remember that any kind of group changes after the announcement of the assignment is not allowed.

  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: Problem Bank 21

Group Name: Group-21

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	Arun M M	2021SC04065	**100%** Recorded observations for all the memory mapping techniques, verified other member's values, consolidated them, and generate the graph for all test cases, and provided comments on all the graphs, and tables.
2	DESHPANDE AADITYA PRASAD	2021SC04064	**100%** Recorded observations for all the memory mapping techniques and verified them with team also helped to generate the graph. Discussed and finalised the observation with the team.
3	RISAB BISWAS	2021SC04063	**100%** Recorded observation for all the memory mapping techniques and verified for each of them with the team to make the final draft. Helped team to generate the graph. Discussed and

			finalised the observations with the team

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 into 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 searching of an element (key) in an array using linear search technique.

```
program LinearSearch
        var a array(50) byte

        writeln("Array Elements: ")

        for n = 0 to 10
                a(n) = n
                writeln(a(n))
        next

        key = 10
        writeln("Key to be searched: ",key)

        found = 0

        for n = 0 to 20
                temp = a(n)
                if temp = key then
                        found = 1
                        writeln("Key Found",temp)
                        break
                end if
        next
        if found <> 1 then
                writeln("Key Not Found")
        end if
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

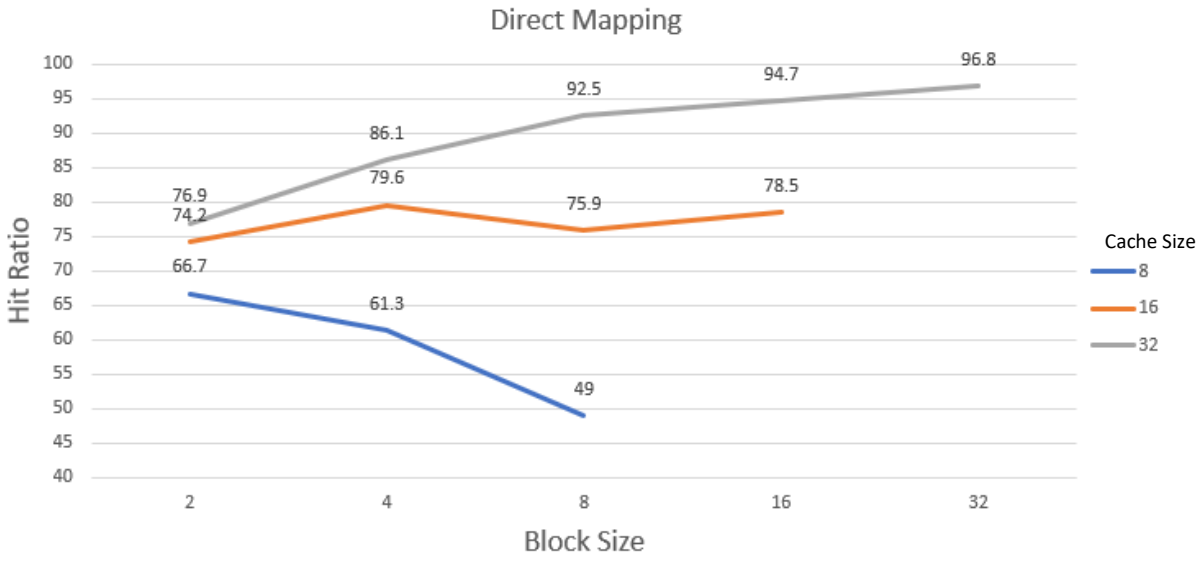
- After compiling and loading the assembly language code in CPU simulator, press “Cache-Pipeline” tab and select cache type as “both”. Press “SHOW CACHE” button.
- In the newly opened cache window, choose appropriate cache Type, cache size, set blocks, replacement algorithm and write back policy.

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	124	62	33.3	66.7
4		114	72	38.7	61.3
8		91	95	51.0	49
2	16	138	48	25.8	74.2
4		148	38	20.4	79.6
8		141	45	24.1	75.9
16		146	40	21.5	78.5
2	32	143	43	23.1	76.9
4		160	26	13.9	86.1
8		172	14	7.53	92.5
16		176	10	5.38	94.7
32		180	6	3.23	96.8

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.



From the graph, we can make the following observations

1. With the effect of increasing the cache size we could observed that the overall hit ratio for all Block sizes also gets increased.
2. When the cache size = 32 and Block size = 32 the Hit Ratio% increased significantly.
3. For each of the Cache Sizes, the Hit Ratio for block size = 2 is very close to that of block size = cache sizes and even exceeds it as the Cache Size is increased.
4. With that by increasing the cache size, the hit ratio improved, thus the performance of the program improved. Higher the cache size, the functioning of CPU is better.

c) Fill the below table and write a small note on your observation from **data cache**.

- Block Size = 2
- Cache Size = 8
- Cache Type = Direct Mapped

Addresses	Data	Miss (%)
0124	**00**	**33.3%**
0125	**00**	**33.3%**
0118	**20**	**33.3%**
0119	**46**	**33.3%**
0120	**6F**	**33.3%**
0121	**75**	**33.3%**
0122	**6E**	**33.3%**
0123	**64**	**33.3%**

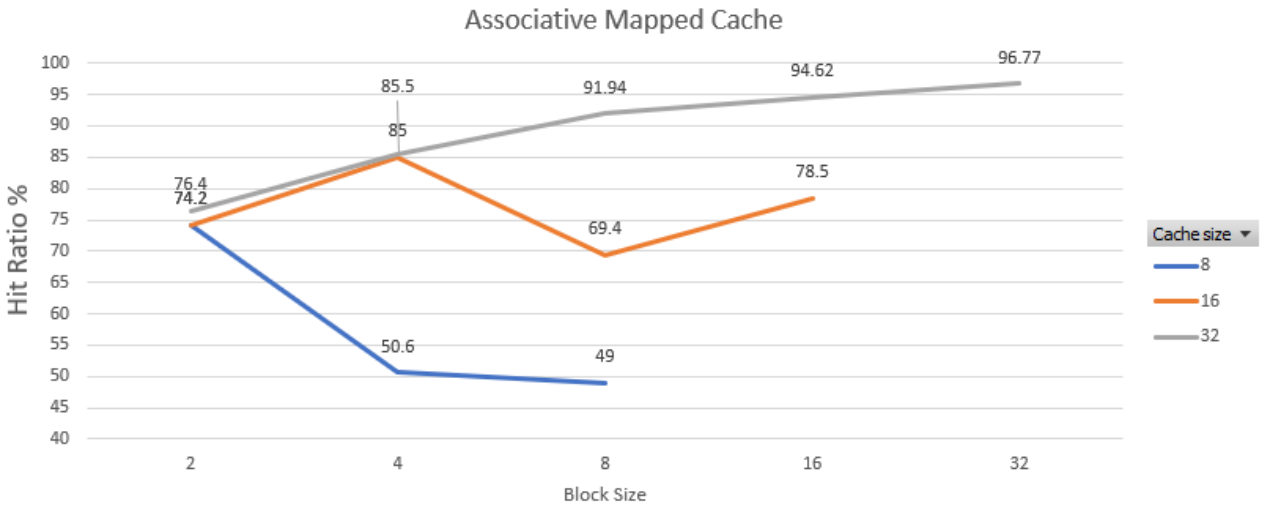
With the above table we observed that how interactive data being assigned to individual address, also we observed that totally 8 address has been used when block size = 2 and cache size =8 for Direct Mapped.

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	138	48	25.8	74.2
4		94	92	49.4	50.6
8		91	95	51.0	49
2	16	138	48	25.8	74.2
4		158	28	15	85
8		129	57	30.6	69.4
16		146	40	21.5	78.5
2	32	142	44	23.6	76.4
4		159	27	14.5	85.5
8		171	15	8.06	91.94
16		176	10	5.38	94.62
32		180	6	3.23	96.77

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.



From the graph, we can make the following observations:

1. With an increase in the Cache Size, the Hit Ratio for all corresponding Block sizes increase too. However, the performance dip at the intermediate block sizes is more significant than for Direct Mapping.
2. For cache Sizes 8 and 16, the Hit Ratio for block size = 2 is relatively aligned together, whereas cache size =32, the Hit Ratio of block size =2-bit higher hit radio than others.
3. We can observe that for all the cache sizes, the final block size related value is identical to that for Direct Mapping.
4. We can also observe that for block size = 2, the Hit Ratios are relatively equal for Direct Mapping except block size =2 and Cache size =8.
5. With that by increasing the cache size, the hit ratio improved, thus the performance of the program improved. Higher the cache size, the functioning of CPU is better

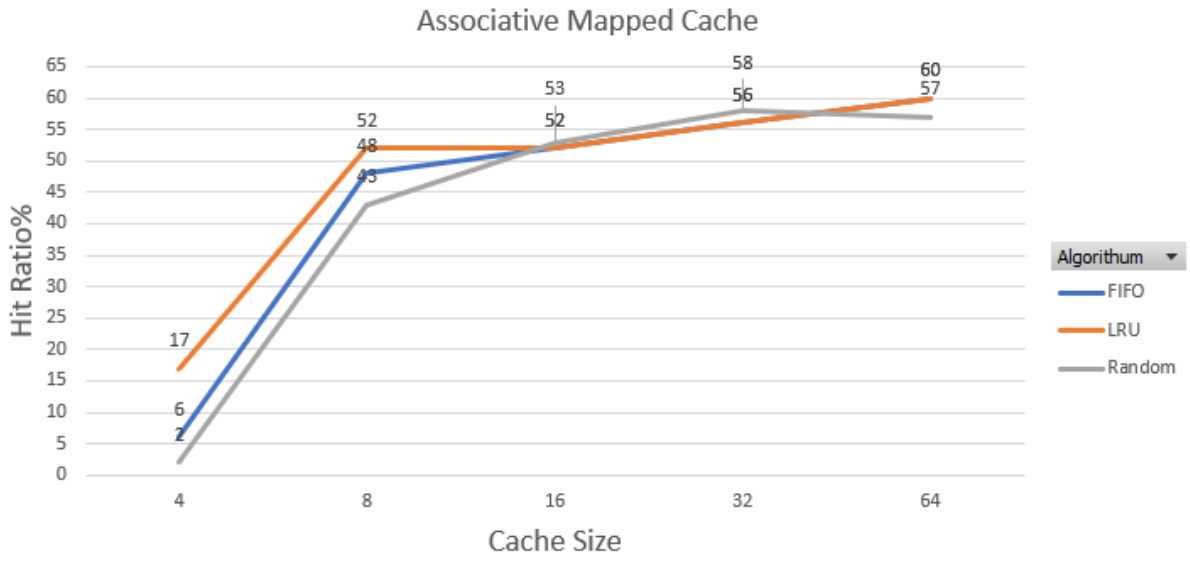
c) Fill up the following table for three different replacement algorithms and state which replacement algorithm is better and why?

Replacement Algorithm: Random				
Block Size	Cache size	Miss	Hit	Hit ratio
2	4	98	88	2
2	8	57	129	43
2	16	47	139	53
2	32	42	144	58
2	64	43	143	57
Replacement Algorithm: FIFO				
Block Size	Cache size	Miss	Hit	Hit ratio
2	4	94	92	6
2	8	52	134	48
2	16	48	138	52
2	32	48	138	52
2	64	40	146	60
Replacement Algorithm: LRU				
Block Size	Cache size	Miss	Hit	Hit ratio
2	4	83	103	17
2	8	48	138	52
2	16	48	138	52
2	32	44	142	56
2	64	40	146	60

1. From the values, we can see that the LRU algorithm has the highest hit ratio comparted to other two algorithm (cache sizes = 2, Block size = 2).
2. We can see that FIFO has the highest hit ratio at cache size = 64, where it ties with LRU for 100%.
3. We can see that LRU, and FIFO algorithm has the same hit ratio for cache sizes of 16, 32, and 64.

4. Based on the variations of Random Algorithm as observed by the group members, we can infer that the consistent **best** algorithm in this case will be **LRU** with the FIFO algorithm being a close second place because of the slight variations on every iteration with same values.

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



From the graph we can observe that:

1. For higher values of cache size, the Radom replacement algorithm performs better consistently as observed by all team members.
2. With increasing cache size, we observe that the FIFO and LRU algorithms improve at a faster rate than the Random algorithm.
3. At the cache size of 64, we can see that LRU & FIFO algorithms are almost identical in terms of hit ratio where Radom algorithm hit radio bit lower than other two algorithms.
4. Also, we observed that the Miss %, Hit %, and Hit Ratio are identical for LRU & FIFO where the block size = 2, Cache size = 16 and 32.

Part III: Set Associative Mapped Cache

Execute the above program by setting the following Parameters:

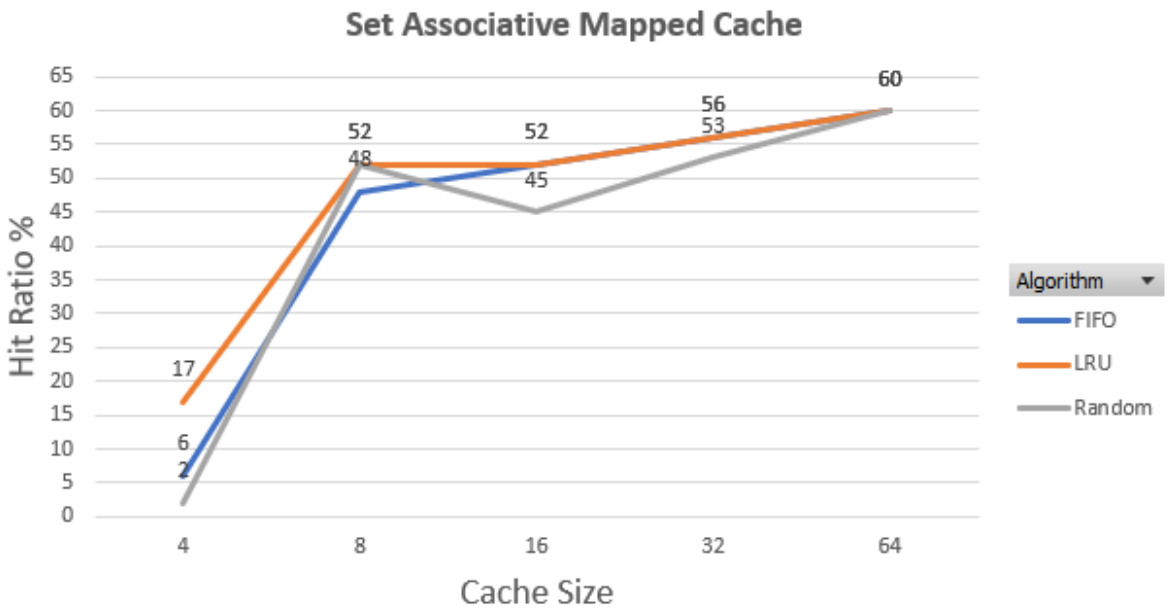
- Number of sets (Set Blocks): 2 way
- Cache Type: Set Associative
- Replacement: LRU/FIFO/Random

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	98	88	2
2	8	48	138	52
2	16	55	131	45
2	32	47	139	53
2	64	40	146	60
	Replacement Algorithm: FIFO			
Block Size	Cache size	Miss	Hit	Hit ratio
2	4	94	92	6
2	8	52	134	48
2	16	48	138	52
2	32	44	142	56
2	64	40	146	60
	Replacement Algorithm: LRU			
Block Size	Cache size	Miss	Hit	Hit ratio
2	4	83	103	17
2	8	48	138	52
2	16	48	138	52
2	32	44	142	56
2	64	40	146	60

1. From the given table values comparison of hit ratio, we can observe that the **LRU** algorithm is the best algorithm.
2. FIFO replacement algorithm being a close second since Hit ratio for Black =2, Cache size =4 and 8 slightly lesser than the LRU algorithm.

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



From the graph, we can observe that:

1. For higher values of cache size, we can see that Random algorithm has a better hit ratio. This has also been consistent in all tests of all team members.

2. As the cache size gets close to 64, we can see that the hit ratios of all three algorithms become identical without any variation.

3. As with Associative Mapping, we can see that LRU and FIFO have a steeper slope with FIFO again having the steepest slope of the three.

4. Another point of observation is that across all the tests, the sum of number of hits and number of misses always adds up to 186.

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	40	146	60
2, 64	4 – Way	40	146	60
2, 64	8 – Way	40	146	60

1. From the values, we can see that all 2, 4, 8-way are equally performing for block size =2 and Cache size =64.
2. With that we can assume that in this case, all are equally good