

Assignment Set Number: 46

Group Name: 126

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	Anuroop Bisaria	2021FC04237	100%
2	Harshita Bhati	2021FC04241	100%
3	Vipin Indoria	2021FC04245	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 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 binary search technique.

```
program BinarySearch
VAR a array(10) INTEGER
for n = 0 to 9
    a(n) = n
    writeln (a(n))
next
VAR key INTEGER
VAR first INTEGER
VAR last INTEGER
VAR middle INTEGER
```

```

VAR temp INTEGER
key = 3
writeln("Key to be searched",key)
first = 0
last = 9
middle = (first+last)/2
while first <= last
    temp = a(middle)
    if temp = key then
        writeln("Key Found",middle)
        break
    end if
    if key > temp then
        first = middle + 1
    else
        last = middle - 1
    end if
    middle = (first+last)/2
wend
if first > last then
    writeln("Key Not Found")
end if
end

```

General procedure to convert the given STL program in to 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.

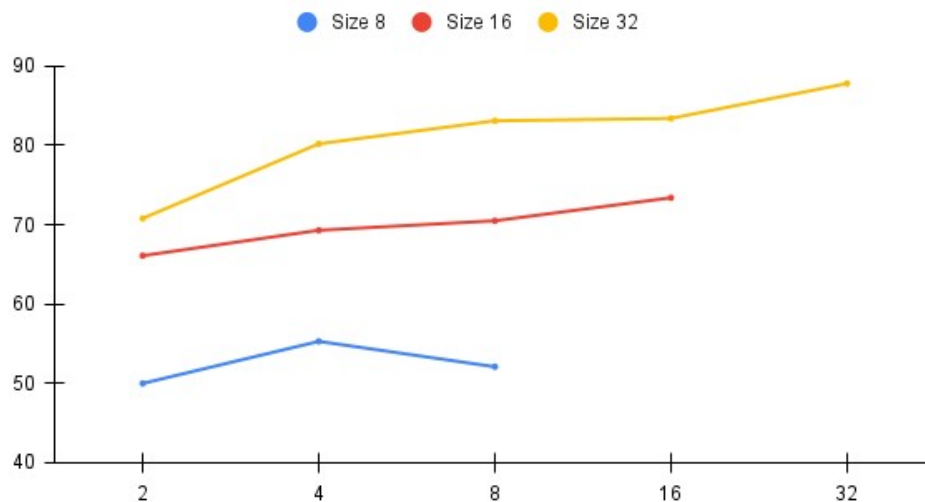
Part I: Direct Mapped Cache

- 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	171	171	50	50
4		189	153	44.7	55.3
8		178	164	47.9	52.1
2	16	226	116	33.9	66.1
4		237	105	30.7	69.3
8		241	101	29.5	70.5
16		251	91	26.6	73.4
2	32	242	100	29.2	70.8
4		274	68	19.8	80.2
8		284	58	16.9	83.1
16		285	57	16.6	83.4
32		300	42	12.2	87.8

- 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.

Cache Hit% vs Block Size for Different Cache Sizes



Cache hit rate improves as the block size gets bigger, resulting in fewer compulsory misses by improving spatial locality. Cache hit rate also improves with increase in cache size, due to smaller caches resulting in thrashing.

The scale of the values in the graph is too small to reflect this, but very large block sizes can actually hurt the cache hit percentage, due to increased potential of conflict misses with the larger working set.

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

- Block Size = 8
- Cache Size = 16
- Cache Type = Direct Mapped

Addresses	Data	Miss (%)
0048	03	29.5
0049	02	29.5
0050	00	29.5
0051	03	29.5
0052	02	29.5
0053	00	29.5
0054	03	29.5
0055	02	29.5
0056	00	29.5
0057	03	29.5
0058	02	29.5
0059	00	29.5
0060	03	29.5
0061	03	29.5
0062	4B	29.5
0063	65	29.5

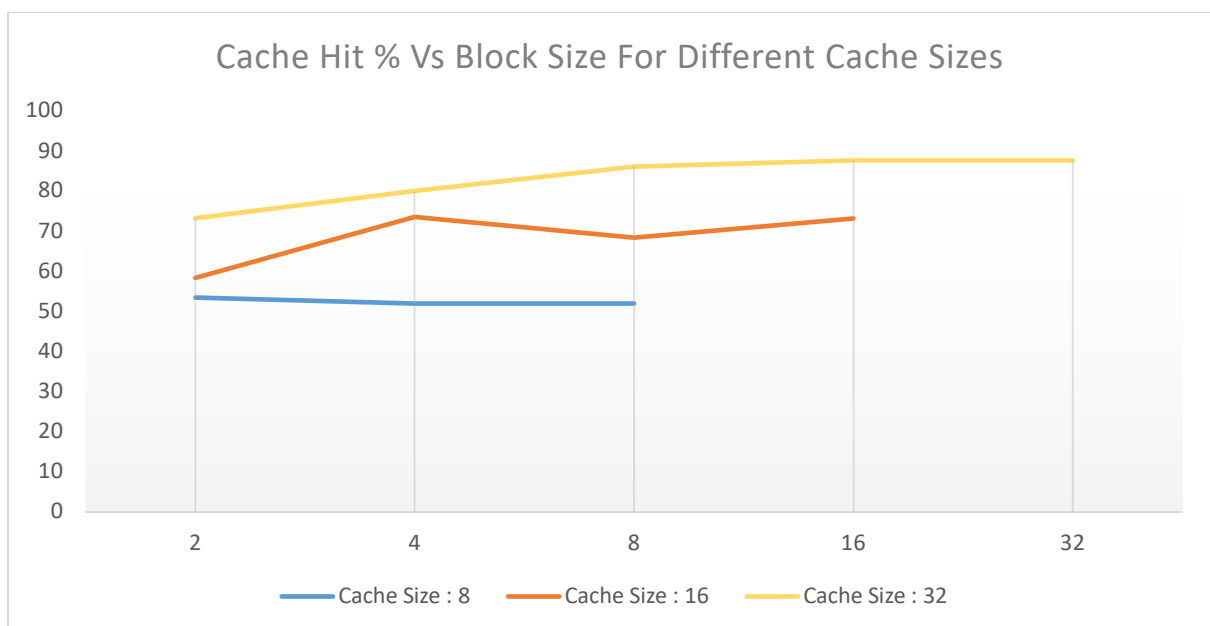
At the end of execution, Block 0 is directly mapped to memory addresses 48-55, and Block 1 is directly mapped to memory addresses 56-63.

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	183	159	46	54
4		178	164	48	52
8		178	164	48	52
2	16	200	142	42	58
4		252	90	26	74
8		234	108	32	68
16		251	91	27	73
2	32	251	91	27	73
4		274	68	20	80
8		295	47	14	86
16		300	42	12	88
32		300	42	12	88

- 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.

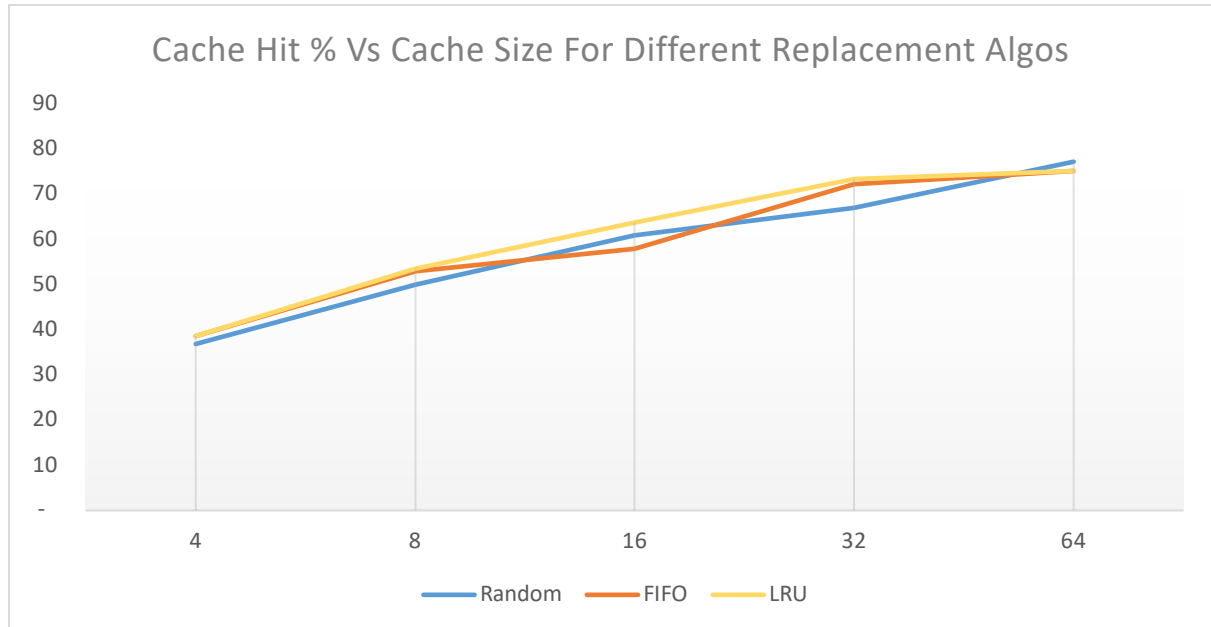


- For LRU Replacement Algorithm, value of Cache hit ratio increases with increase in Cache Size from 8 to 32.
- It was observed from the plot that with lower Cache Size (size=8) even if Block Size is increased there is no significant change in value of Cache hit ratio.
- Whereas for Cache Size of 16 it is linearly increasing and decreasing after each increment of Block size from 2 to 32.
- When Cache size was taken as 32 with each increment of Block size from 2 to 32 Hit Ratio keeps on increasing till it reaches Block size of 16.

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	216	126	37
2	8	171	171	50
2	16	134	208	61
2	32	113	229	67
2	64	78	264	77
Replacement Algorithm: FIFO				
Block Size	Cache size	Miss	Hit	Hit ratio
2	4	210	132	39
2	8	162	182	53
2	16	144	198	58
2	32	95	247	72
2	64	85	257	75
Replacement Algorithm: LRU				
Block Size	Cache size	Miss	Hit	Hit ratio
2	4	210	132	39
2	8	159	183	54
2	16	142	250	64
2	32	91	251	73
2	64	85	257	75

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



- LRU Replacement Algorithm has the highest value of Cache Hit ratio giving very good performance in overall use.
- Of the three, LRU is the best while FIFO is the worst with less hit ratio when block size increases from 8 to 16 and random comes in somewhere between.

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?

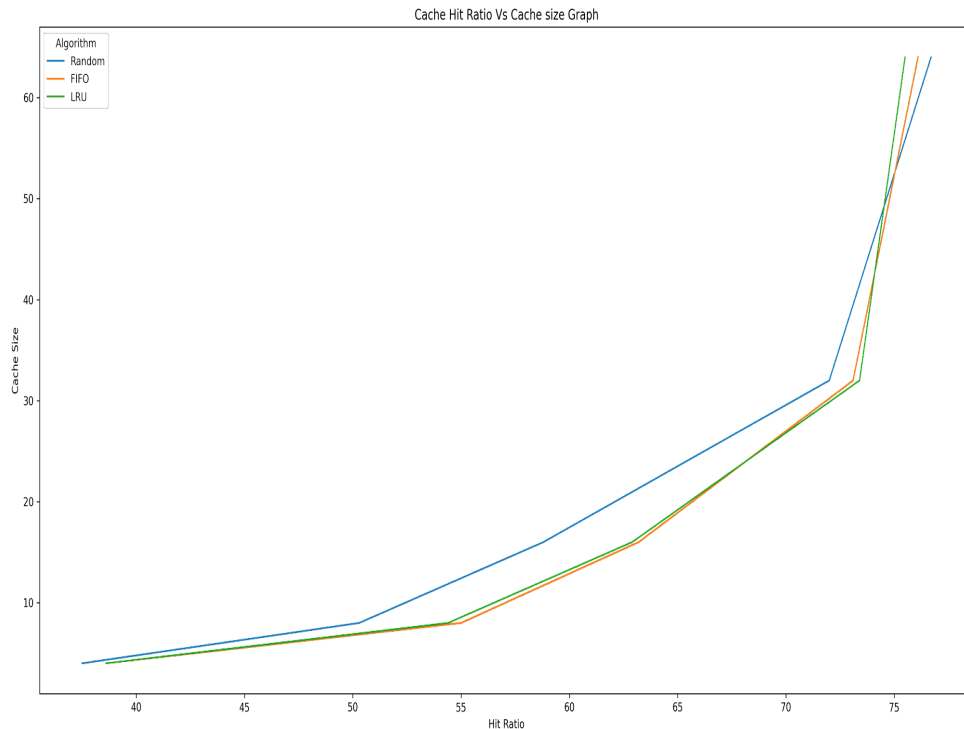
Replacement Algorithm: Random				
Block Size	Cache size	Miss	Hit	Hit ratio
2	4	214	128	37.5
2	8	170	172	50.3
2	16	141	201	58.8
2	32	96	246	72
2	64	80	262	76.7
Replacement Algorithm: FIFO				
Block Size	Cache size	Miss	Hit	Hit ratio
2	4	210	132	38.6
2	8	154	188	55
2	16	126	216	63.2
2	32	92	250	73.1
2	64	82	260	76.1
Replacement Algorithm: LRU				
Block Size	Cache size	Miss	Hit	Hit ratio
2	4	210	132	38.6
2	8	156	186	54.4
2	16	127	215	62.9
2	32	91	251	73.4
2	64	84	258	75.5

Summary:

It is observed that for binary search code in this specific scenario set blocks 2-way and block size 2, FIFO in set associative gives better performance than other replacement algorithms i.e. Random and LRU due to higher hit ratio than others.

Performance of LRU and FIFO is almost identical for smaller cache sizes.

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



Summary:

It is observed that for binary search code in this specific scenario set blocks 2-way and block size 2, FIFO in set associative gives better performance compared to LRU and Random. It is observed that for small cache size FIFO is giving better performance but for larger cache Random gives better performance.

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

Replacement Algorithm: LRU				
Block Size, Cache size	Set Blocks	Miss	Hit	Hit ratio
2, 64	2 – Way	84	258	75.5
2, 64	4 – Way	82	260	76.1
2, 64	8 – Way	82	260	76.1

Summary:

It is observed that for this specific scenario the hit ratio is directly proportional to the set blocks and saturated at higher values. This suggests that increasing the cache size too much does not add any significant improvement in hit ratio. Thus having 4-way set blocks is most optimal for this given specific scenario.