Understanding of cache

Objective:

To provide deeper understanding of cache by analysing it's 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

Code to be used:

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

```
program BubbleSort

var a array(5) byte
```

```
a(0) = 4
a(1) = 1
a(2) = 3
a(3) = 5
a(4) = 2

var len byte
var temp byte
var l1 byte
var l2 byte
var x1 byte
var x2 byte
var j byte
var j1 byte
var k byte
var i byte

len = 5
l1 = len - 1

for k =0 to len
        write(a(k),"  ")
next
writeln("")
writeln("Bubble Sort Starts")

for i = 0 to l1
        l2 = len - i - 1
        for j=0 to l2
                j1 = j + 1
                x1 = a(j)
                x2 = a(j1)
                if x1 > x2 then
                temp = a(j1)
                a(j1) = a(j)
                a(j) = temp
                end if
        next
        for k =0 to len
                write(a(k),"  ")
        next
        writeln("")

next

writeln("Bubble Sort Ends")

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

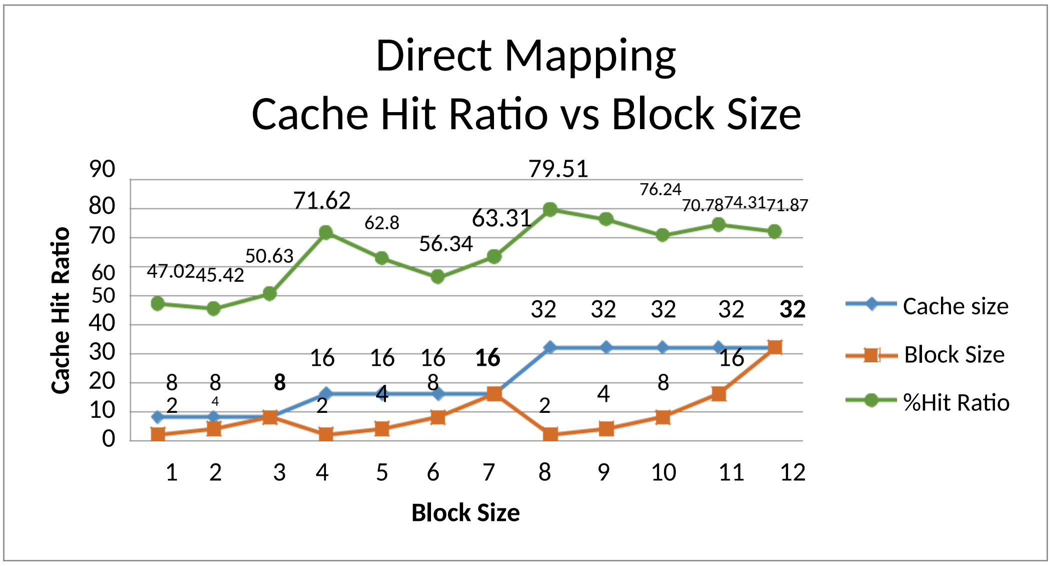
- After compiling and loading the assembly language code in CPU simulator, press **"**Cache-Pipeline**"** tab and select cache type as **"**data cache**"**. Press **"**SHOW CACHE..**"** button.
- In the newly opened cache window, choose appropriate cache Type, cache size, set blocks, replacement algorithm 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	% MissRatio	%Hit Ratio
2	8	560	631	52.98	47.02
4		541	650	54.58	45.42
8		603	588	49.37	50.63
2	16	853	338	28.38	71.62
4		748	443	37.20	62.80
8		671	520	43.66	56.34
16		754	437	36.69	63.31
2	32	947	244	20.49	79.51
4		908	283	23.76	76.24
8		843	348	29.22	70.78
16		885	306	25.69	74.31
32		856	335	28.13	71.87

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.



In a direct mapped cache, each address maps to a unique block and set. If a set is full when new data must be loaded, the block in that set is replaced with the new data.

From the graph, we can make the following observations:

Smaller blocks do not take maximum advantage of spatial locality whereas larger block size take advantage of spatial locality.

Increasing the block size means reduction in the number of cache lines and more the competition between program data for these lines.
Reducing the number of lines impacts the number of separate address blocks that can be accommodated in the cache.

The larger the block size, the more time it takes to fetch this block size from memory. Increases miss penalty, and consumes more memory bandwidth.
If block size is too big relative to cache size, there are fewer blocks available, miss rate will go up as too few cache blocks.

If the cache size and block size are same then there will only be one entry in the cache. If item accessed, likely to be accessed again soon but unlikely to be accessed again immediately. The next access will likely to be a miss again. Continually loading data into the cache but discard data (force out) before use it again. There is a decrease in hit ratio.

Cache memory is an extremely fast memory type and used to reduce the average time to access data from the Main memory. As cache size increases, hit ratio is getting increased.

c) Now, select cache type as **"**instruction cache**"**. Fill in the following table and analyse the behaviour of Direct Mapped Cache. Which one is better with respect to Miss Ratio?

Block Size, Cache size	Miss	Hit	Miss Ratio
2, 8	712	659	51.93
2, 16	454	917	33.11
2, 32	374	997	27.28

Larger size of cache could decrease the 'miss rate' when adjacent memory locations are accessed, i.e. good for the spatial locality. Larger cache size will help eradicate conflict (collision) in multiple memory locations mapped to the same cache location.

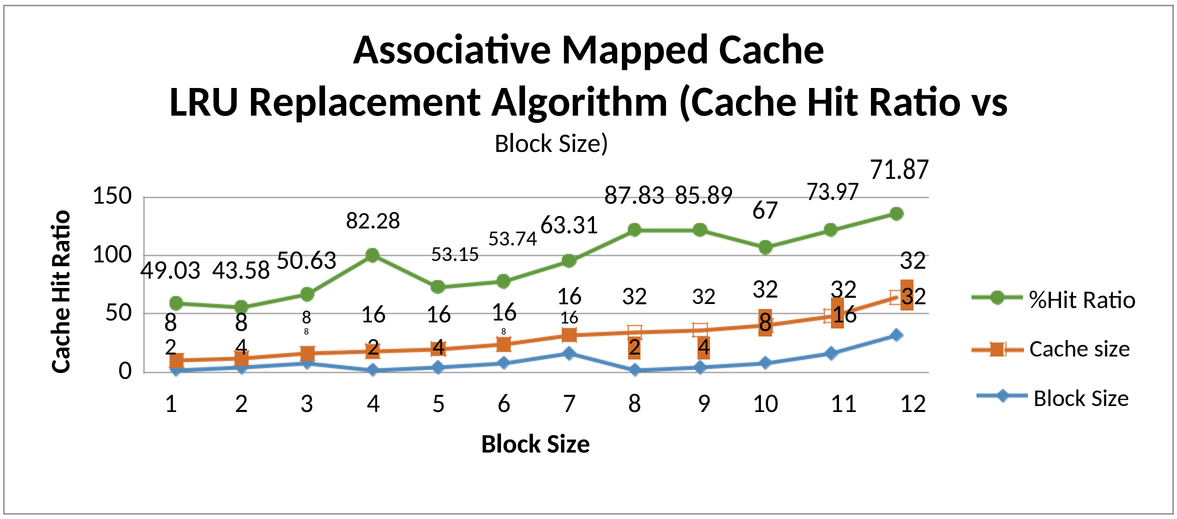
Larger cache size adds the opportunity of fragmentation and false sharing in a multiprocessor system. Block size 2 and Cache size 32 is better with respect to Miss Ratio.

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	% MissRatio	%Hit Ratio
2	8	584	607	50.97	49.03
4		519	672	56.42	43.58
8		603	588	49.37	50.63
2	16	980	211	17.72	82.28
4		633	558	46.85	53.15
8		640	551	46.26	53.74
16		754	437	36.69	63.31
2	32	1046	145	12.17	87.83
4		1023	168	14.11	85.89
8		798	393	33.00	67.00
16		881	310	26.03	73.97
32		856	335	28.13	71.87

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.



In case of associative map cache, blocks can go anywhere in the cache. Here we need to compare with all tags in entire cache in parallel to see if data is present in the cache or not. There will be no conflict misses in this case as data can go anywhere in the cache.

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. We can also observe that for block size = 2, the Hit Ratios are higher than that for Direct Mapping.
3. We can observe that for all the cache sizes, the final block size related value is identical to that for Direct Mapping.
4. If the cache size and block size are same then there will only be one entry in the cache. If item accessed, likely to be accessed again soon but unlikely to be accessed again immediately. The next access will likely to be a miss again. Continually loading data into the cache but discard data (force out) before use it again. There is a decrease in hit ratio. This could be clearly seen in case of cache size = 32 and block size = 32.

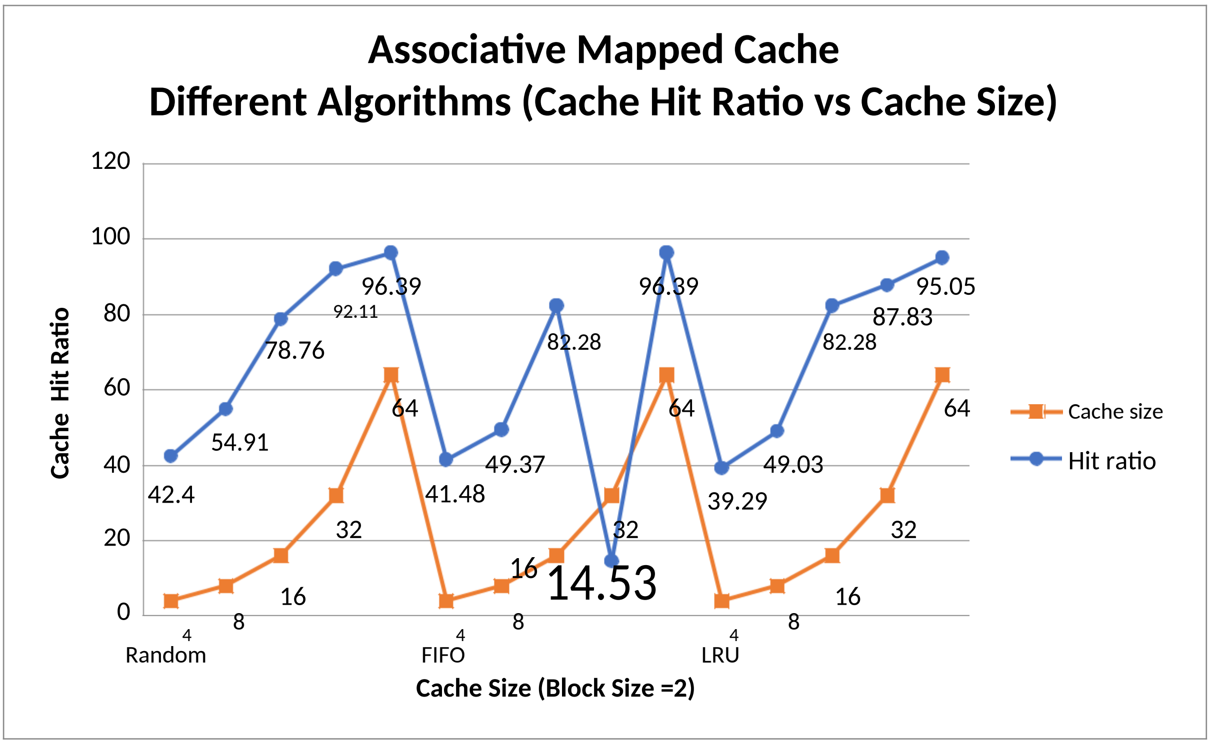
c) Fill up the following table for three different replacement algorithms and state which replacement algorithm is better and why?

Replacement Algorithm : Random				
Block Size	Cache size	Miss	Hit	Hit ratio
2	4	686	505	42.40
2	8	537	654	54.91
2	16	253	938	78.76
2	32	94	1097	92.11
2	64	43	1148	96.39
Replacement Algorithm : FIFO				
Block Size	Cache size	Miss	Hit	Hit ratio
2	4	697	494	41.48
2	8	603	588	49.37
2	16	211	980	82.28
2	32	173	1018	14.53
2	64	43	1148	96.39
Replacement Algorithm : LRU				
Block Size	Cache size	Miss	Hit	Hit ratio
2	4	723	468	39.29
2	8	607	584	49.03
2	16	211	980	82.28
2	32	145	1046	87.83
2	64	59	1132	95.05

LRU replacement algorithm is better because it discards the least recently used item from the cache in order to make space for the new data item. In order to achieve this, history of all data items that is which data item is used when, is kept.

It provides better performance but cost of implementation is much more. Key advantage of this policy is its simple implementation, time and space overhead is constant.

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 above table we can infer that FIFO and LRU are having almost same hit ratio which are better than the hit ratio provided by random replacement algorithm at higher block sizes.

The cache hit ratio is increasing with increase in Cache size value with respect to all the three replacement algorithms.

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 ?

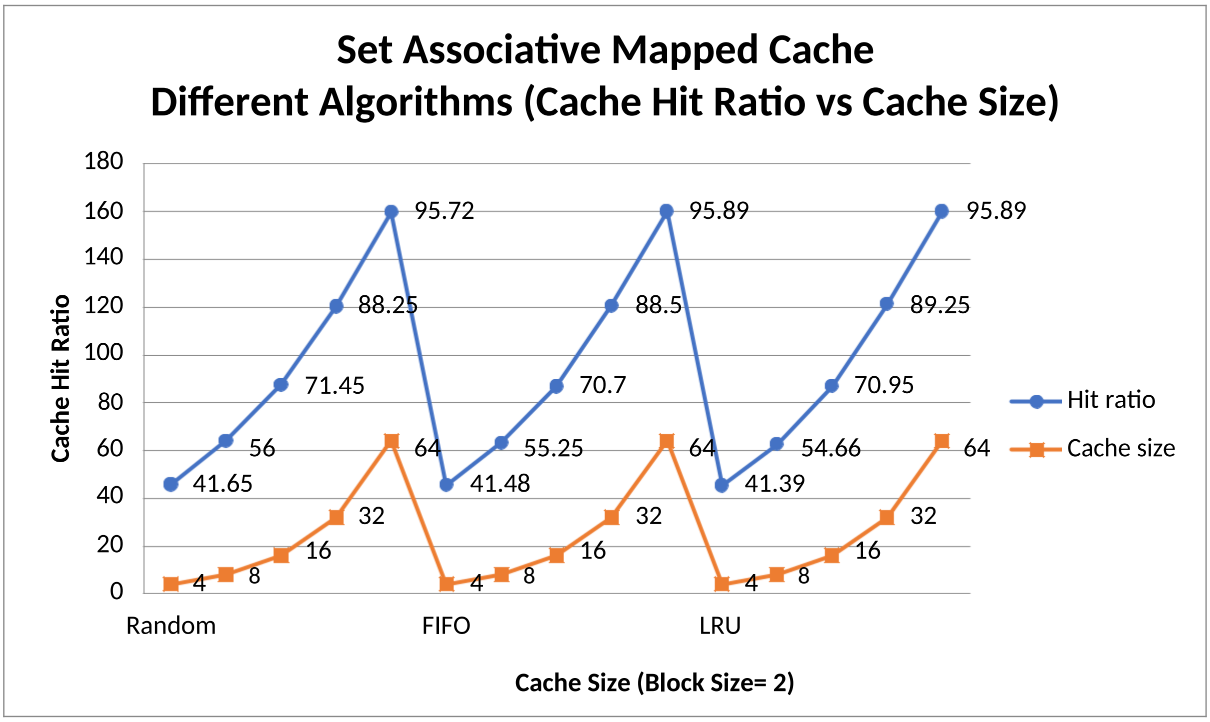
Replacement Algorithm : Random				
Block Size	Cache size	Miss	Hit	Hit ratio
2	4	695	496	41.65
2	8	524	667	56.00
2	16	340	851	71.45
2	32	140	1051	88.25
2	64	51	1140	95.72
Replacement Algorithm : FIFO				
Block Size	Cache size	Miss	Hit	Hit ratio
2	4	697	494	41.48
2	8	533	658	55.25
2	16	349	842	70.70
2	32	137	1054	88.50
2	64	49	1142	95.89
Replacement Algorithm : LRU				
Block Size	Cache size	Miss	Hit	Hit ratio
2	4	698	493	41.39
2	8	540	651	54.66
2	16	346	845	70.95
2	32	128	1063	89.25
2	64	49	1142	95.89

LRU replacement algorithm is better because it discards the least recently used item from the cache in order to make space for the new data item. In order to achieve this, history of all data items that is which data item is used when, is kept.

It provides better performance but cost of implementation is much more. Key advantage of this policy is its simple implementation, time and space overhead is constant

LRU is the best 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(s), we can observe that:

1. From the plot, we can interpret that all the three replacement algorithms are providing similar hit ratio values at a particular cache size for set associative mapping.

2. For lower values of cache size, we can see that Random algorithm has a better hit ratio compared to other algorithms.

3. Slope is steepest in all three algorithms.

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	49	1142	95.89
2, 64	4 – Way	50	1141	95.80
2, 64	8 – Way	50	1141	95.80

Set Block 2-Way is better. For a two-way set associative cache, only one extra bit is necessary to tell which one line has been accessed most recently. This is accomplished by setting the bit if the one of the line is accessed and clearing the bit when the other is accessed. And the other will be the least recent used line. For an n-way set associative cache, here are some methods to implement the LRU replacement algorithm.

· The counter method - needs n*log(n) bits and logic to count and compare. (log is base 2)

- The encoded ordering method - needs log(n!) bits and logic (FSM) to update the ordering.

Both methods are very costly. That is why LRU is seldom used for set associative caches with more that 4 ways. (E.g. a 4-way set-associative cache using the second method will need 5 bits, while an 8-way set-associative cache will need 16 bits using the second method.). That is why a heuristic LRU is usually used.·