**Assignment I**

**Problem Bank 04**

**Assignment Description:**

The assignment aims to provide deeper understanding of cache by analysing it’s behaviour using cache implementation 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 **26th December 2020, 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](mailto: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:**

**Group Name:**

**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** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**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 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 13

        a(n) = n

writeln(a(n))

next

key = 9

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

found = 0

for n = 0 to 13

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 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**

1. 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 | 136 | 73 | 34.9282 | 65.0717 |
| 4 | 127 | 82 | 39.2344 | 60.7655 |
| 8 | 96 | 113 | 54.0669 | 45.9330 |
| 2 | 16 | 150 | 59 | 28.2296 | 71.7703 |
| 4 |  | 161 | 48 | 22.9665 | 77.0334 |
| 8 |  | 154 | 55 | 26.3157 | 73.6842 |
| 16 |  | 159 | 50 | 23.9234 | 76.0765 |
| 2 | 32 | 161 | 48 | 22.9665 | 77.0334 |
| 4 |  | 178 | 31 | 14.8325 | 85.1674 |
| 8 |  | 190 | 19 | 9.0909 | 90.9090 |
| 16 |  | 196 | 13 | 6.22 | 93.7799 |
| 32 |  | 200 | 9 | 4.3062 | 95.6937 |

1. 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.

**Comments:**

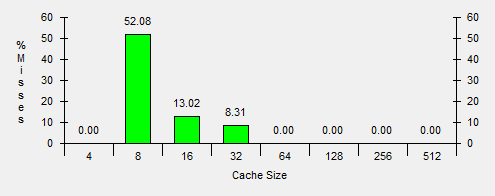
* As the Cache size increases, the hit ratio also increases due to the increase in number of cache lines.

1. 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 | 188 | 173 | 52.0775 |
| 2, 16 | 47 | 314 | 13.0193 |
| 2, 32 | 30 | 331 | 8.3102 |

**Comments:**

* As the program is accessing 2 blocks that maps to the same line repeatedly, cache misses are very high (thrashing) when cache size is lower.
* When the block size is constant, as the Cache size increases, the miss ratio decreases. Hence, the combination of Block Size = 2 and Cache Size = 32 is the best with respect to Miss Ratio.



**Part II: Associative Mapped Cache**

1. 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 | 151 | 58 | 27.7512 | 72.2488 |
| 4 | 123 | 86 | 41.1483 | 58.8517 |
| 8 | 96 | 113 | 54.0670 | 45.9330 |
| 2 | 16 | 155 | 54 | 25.8373 | 74.1627 |
| 4 |  | 172 | 37 | 17.7033 | 82.2967 |
| 8 |  | 153 | 56 | 26.7943 | 73.2057 |
| 16 |  | 159 | 50 | 23.9234 | 76.0766 |
| 2 | 32 | 157 | 52 | 24.8804 | 75.1196 |
| 4 |  | 176 | 33 | 15.7895 | 84.2105 |
| 8 |  | 189 | 20 | 9.5694 | 90.4306 |
| 16 |  | 196 | 13 | 6.2201 | 93.7799 |
| 32 |  | 200 | 9 | 4.3062 | 95.6938 |

1. 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.

**Comments:**

* When Cache size = 8, the hit ratio decreases as the block size increases.
* When Cache size = 16, hit ratio increases goes up and down as block size increases.
* When Cache size is higher i.e., 32, the hit ratio increases with the increase in block size due to improved locality of reference.

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 | 102 | 107 | 51.1962 |
| 2 | 8 | 70 | 133 | 65.5172 |
| 2 | 16 | 57 | 152 | 72.7273 |
| 2 | 32 | 47 | 162 | 77.5120 |
| 2 | 64 | 45 | 164 | 78.4689 |
| Replacement Algorithm: FIFO | | | | |
| Block Size | Cache size | Miss | Hit | Hit ratio |
| 2 | 4 | 105 | 104 | 49.7608 |
| 2 | 8 | 58 | 151 | 72.2488 |
| 2 | 16 | 54 | 155 | 74.1627 |
| 2 | 32 | 52 | 157 | 75.1196 |
| 2 | 64 | 45 | 164 | 78.4688 |
| Replacement Algorithm: LRU | | | | |
| Block Size | Cache size | Miss | Hit | Hit ratio |
| 2 | 4 | 115 | 94 | 44.9761 |
| 2 | 8 | 52 | 157 | 75.1196 |
| 2 | 16 | 52 | 157 | 75.1196 |
| 2 | 32 | 50 | 159 | 76.0765 |
| 2 | 64 | 45 | 164 | 78.4688 |

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

**Comments:**

* FIFO and LRU replacement algorithms performed better than Random replacement algorithm.
* When Cache size = 8 or 16 and Block size = 2; the LRU(Least Recently Used) algorithm exhibited the highest hit ratio compared to FIFO and Random replacement algorithm. Hence, we can say that, at lower sizes of cache, LRU performs better compared to other two replacement algorithms.
* All the algorithms exhibit the same Hit ratio when Cache size = 64. Hence, from this we can say that at higher sizes of cache, the choice of replacement algorithm doesn’t vary the Hit Ratio.

**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 | 143 | 66 | 31.5789 |
| 2 | 8 | 92 | 117 | 55.9808 |
| 2 | 16 | 73 | 136 | 65.0717 |
| 2 | 32 | 58 | 151 | 72.2488 |
| 2 | 64 | 45 | 164 | 78.4688 |
| Replacement Algorithm : FIFO | | | | |
| Block Size | Cache size | Miss | Hit | Hit ratio |
| 2 | 4 | 105 | 104 | 49.7607 |
| 2 | 8 | 57 | 152 | 72.7272 |
| 2 | 16 | 54 | 155 | 74.1626 |
| 2 | 32 | 48 | 161 | 77.0334 |
| 2 | 64 | 45 | 164 | 78.4688 |
| Replacement Algorithm : LRU | | | | |
| Block Size | Cache size | Miss | Hit | Hit ratio |
| 2 | 4 | 115 | 94 | 44.9760 |
| 2 | 8 | 52 | 157 | 75.1196 |
| 2 | 16 | 52 | 157 | 75.1196 |
| 2 | 32 | 47 | 162 | 77.5119 |
| 2 | 64 | 45 | 164 | 78.4688 |

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

**Comments:**

* From the above graph, we can notice that the Hit Ratio of LRU is better than FIFO in most of the cases.
* Considering the fact that the LRU has to maintain the access history for each block, which slowers down the cache, we can say that FIFO is the best replacement policy that can be used in real-time.

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 | 45 | 164 | 78.4688 |
| 2, 64 | 4 – Way | 45 | 164 | 78.4688 |
| 2, 64 | 8 – Way | 45 | 164 | 78.4688 |

* It can be noticed that the Hit ratio is same in all the above cases when Block size and cache size are constant. But, comparatively the 2-way set associative cache can hold 2 blocks in each set, which reduces the required tag bits and thereby reducing the number of comparators required. In other words, the hardware complexity of 2-way set associative cache is lower, making it the better choice.