## Assignment Number: 1

## Problem Statement: 6

Write a program for addition of two numbers in memory and save the result in given memory location, considering the following conditions

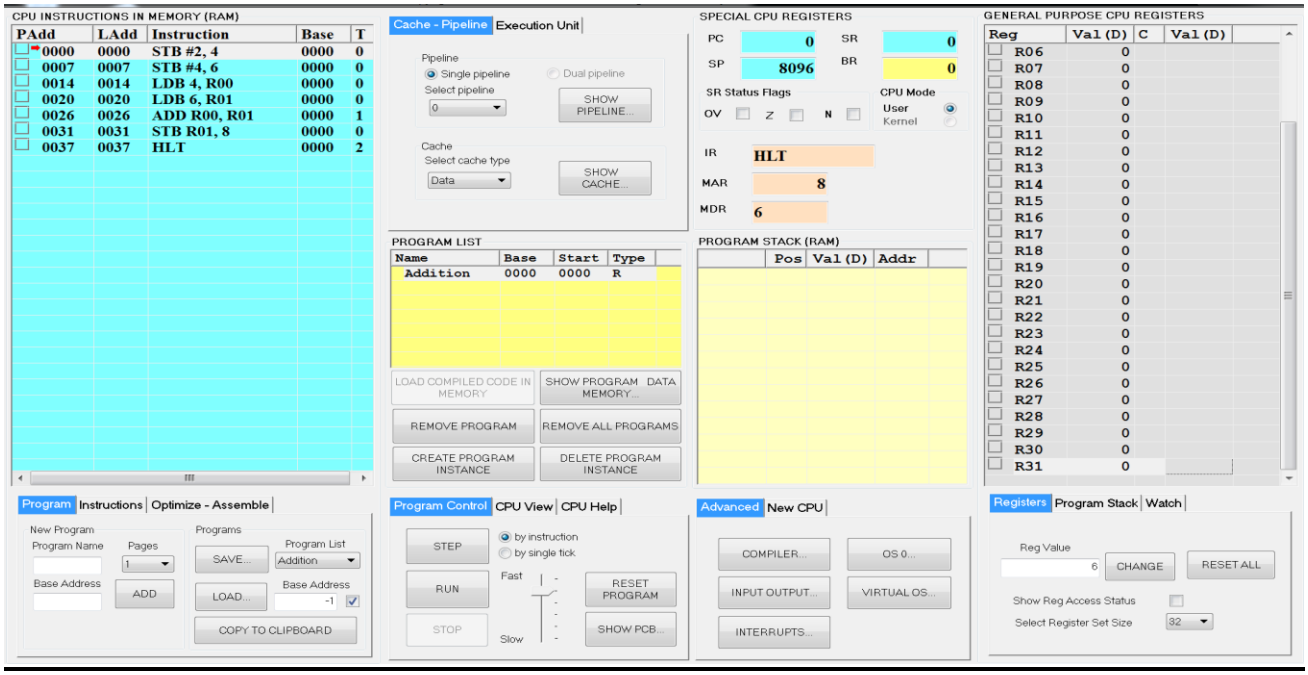
a) The content of memory location 4 is 2.
b) The content of memory location 6 is 4.
c) Add the content of above two memory location and save the result in memory location 8.

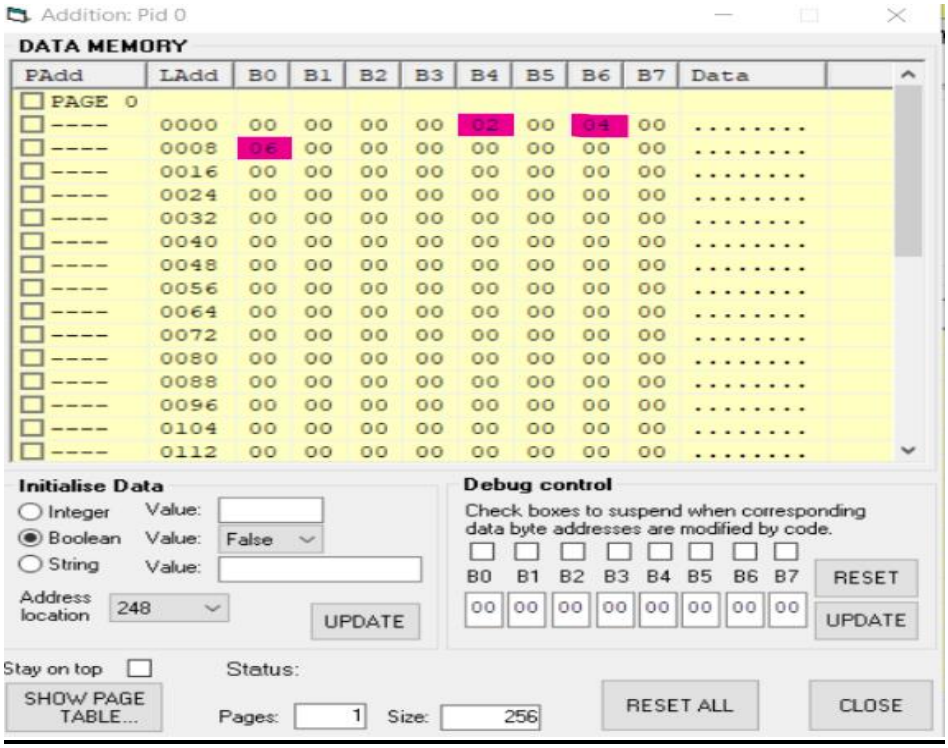
## Contribution Table:

| Sl. No. | Name | ID NO | Contribution |
|---|---|---|---|
| 1 | **Praven H** | **2018AC04536** | • Program creation.<br>• Understanding and capture results for Direct mapping.<br>• Pipeline understanding-knowledge sharing to the team.<br>• Inference writing for cache concepts. |
| 2 | **Jaikumar T V** | **2018AC04556** | • Program modification.<br>• Understanding and capture results for Associative cache type(Set and Full associative)<br>• Inference writing for Pipeline concepts<br>• Documentation of results and consolidation. |
| 3 | **Prasanna R** | **2018AC04552** | • Program review.<br>• Analysis of Direct mapping cache type.<br>• Capture results without Pipeline.<br>• Inference writing for cache concepts |
| 4 | **Avinash Srivastava** | **2018AC04564** | • Initial analysis of program.<br>• Validation of Associative mapping results.<br>• Capture results with pipeline enabled.<br>• Inference writing for Pipeline concepts |
| 5 | **Ravi Teja Maripina** | **2018AC04558** | • Instruction set analysis to use in program.<br>• Re-run for all the input data provided and analysed the results.<br>• Verification of results.<br>• Inference writing based on final output. |

## Solution:

AdditionOFTwoNumbers.sas



- **Enter cache parameters in Table below:**

**Data cache:**

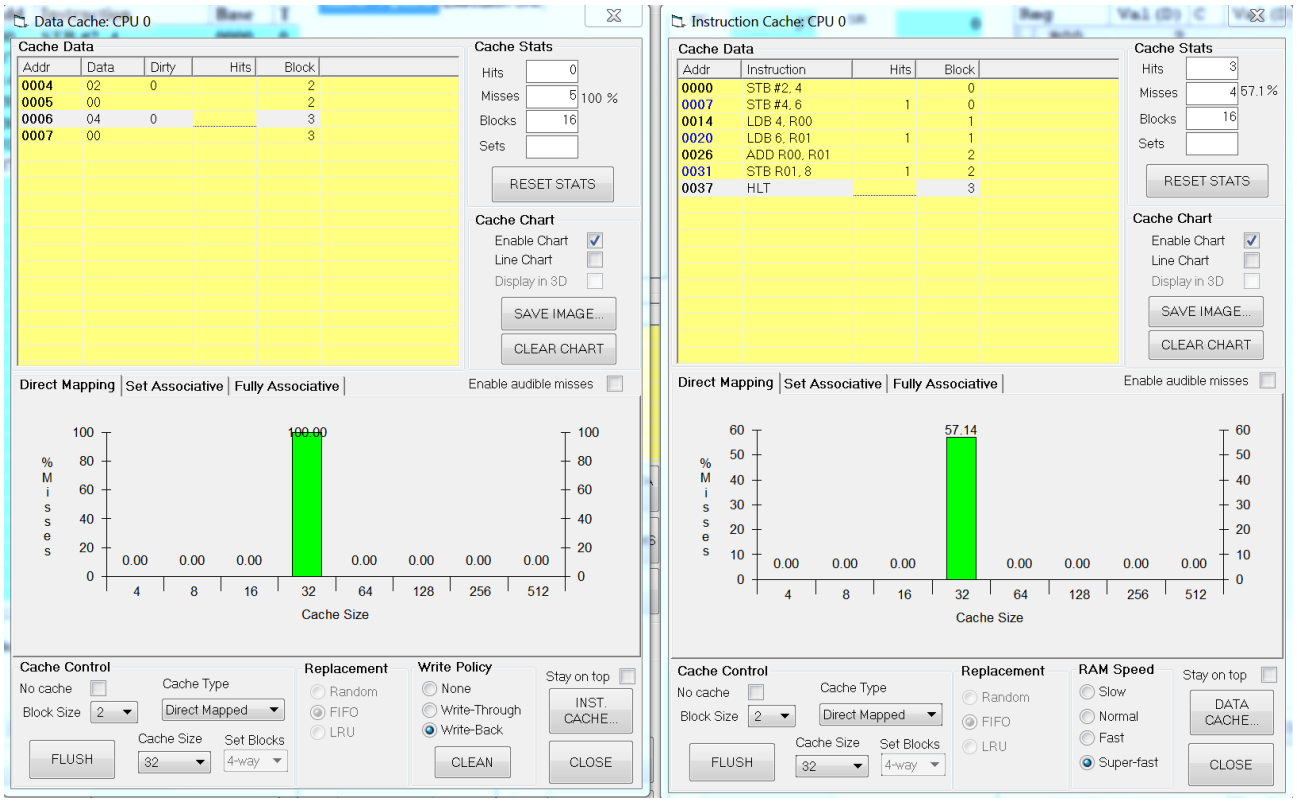
| Block Size | No of blocks | Miss | Hit | Miss ratio |
|---|---|---|---|---|
| 2 | 16 | 5 | 0 | 100% |
| 4 | 16 | 4 | 0 | 80% |
| 8 | 16 | 4 | 1 | 80% |
| 16 | 16 | 3 | 2 | 60% |
| 32 | 16 | 3 | 2 | 60% |

**Instruction cache:**

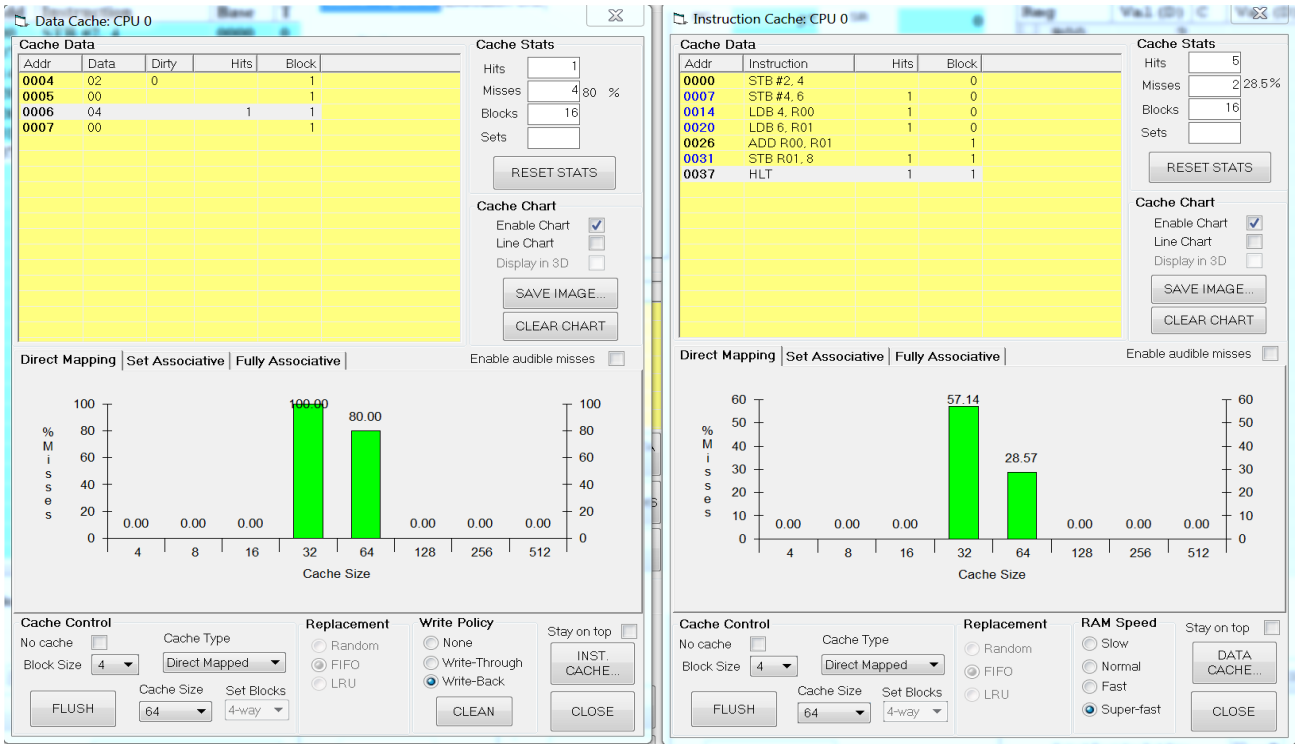
| Block Size | No of blocks | Miss | Hit | Miss ratio |
|---|---|---|---|---|
| 2 | 16 | 4 | 3 | 57.14% |
| 4 | 16 | 2 | 5 | 28.57% |
| 8 | 16 | 1 | 6 | 14.29% |
| 16 | 16 | 1 | 6 | 14.29% |
| 32 | 16 | 1 | 6 | 14.29% |

**SUPPORTING SCREEN SHOTS FOR THE ABOVE TABLE**

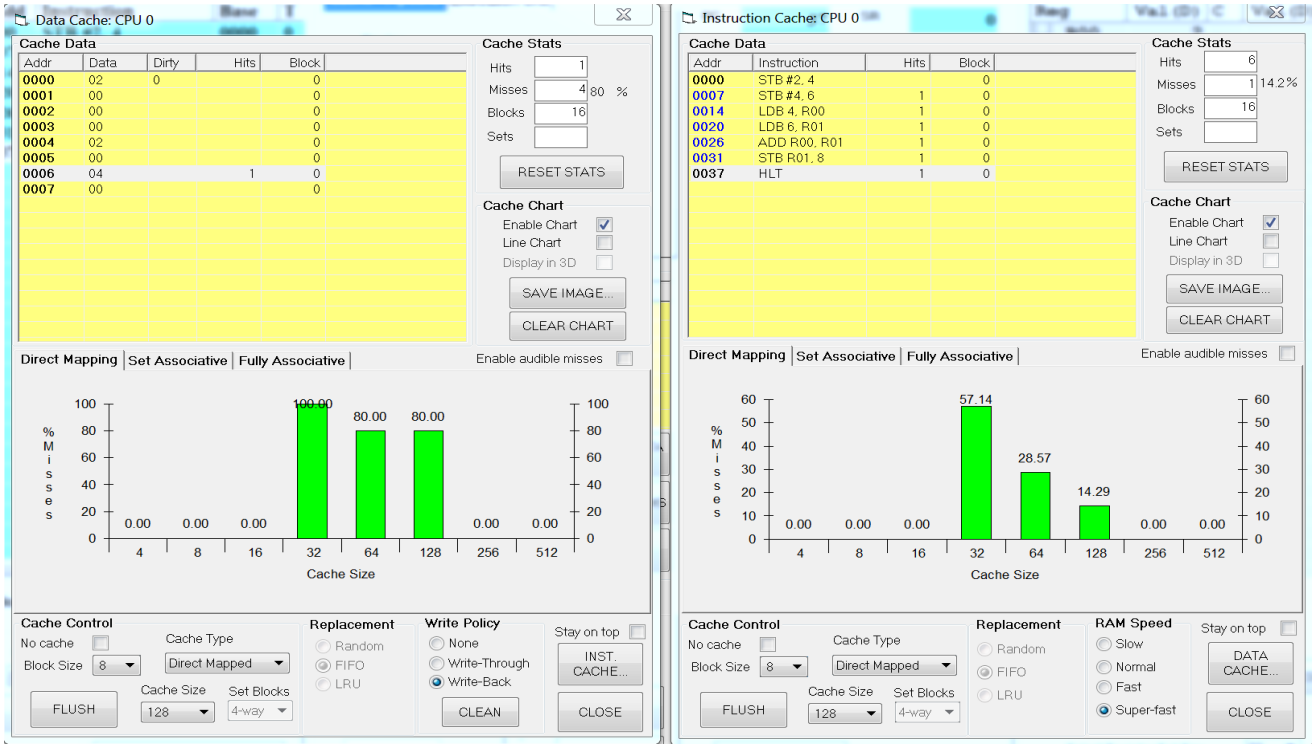
**Block size: 2, Data cache miss ratio:100%, Instruction cache miss ratio:57.14%**



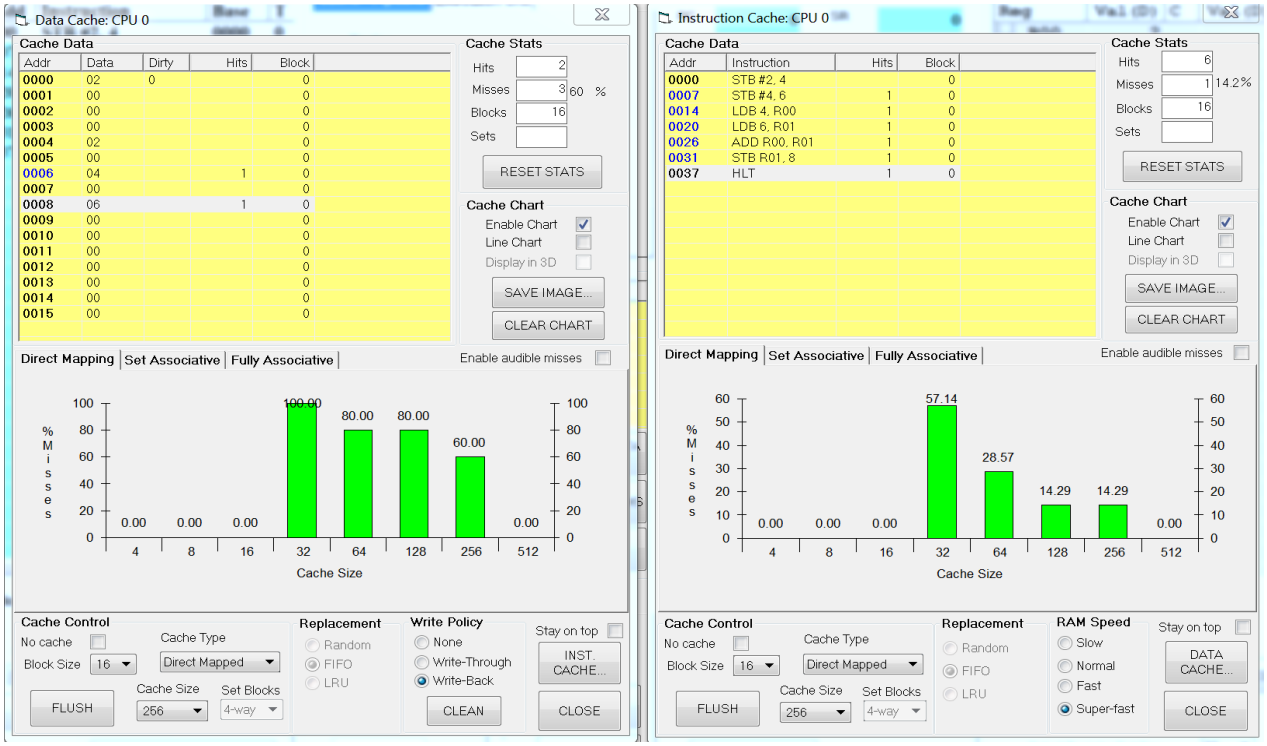
## Block size: 4, Data cache miss ratio:80 %, Instruction cache miss ratio:28.57 %



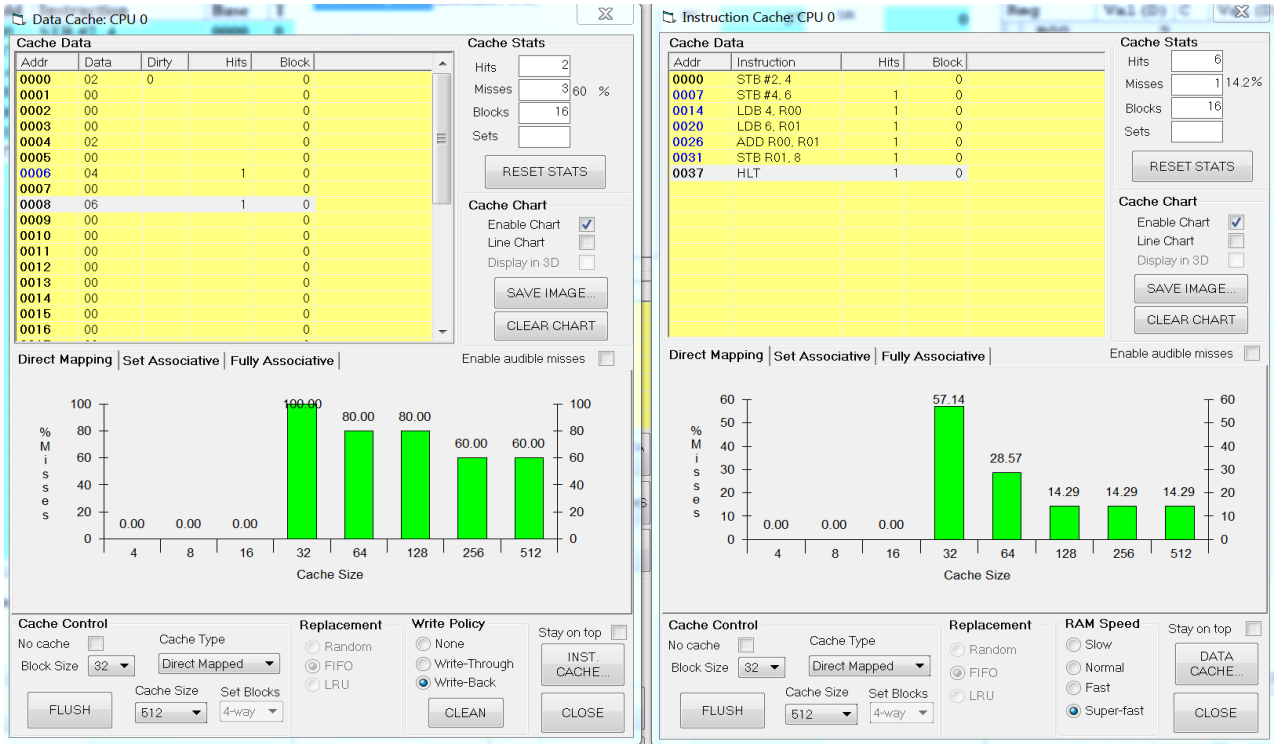

## Block size: 8, Data cache miss ratio:80 %, Instruction cache miss ratio:14.29 %



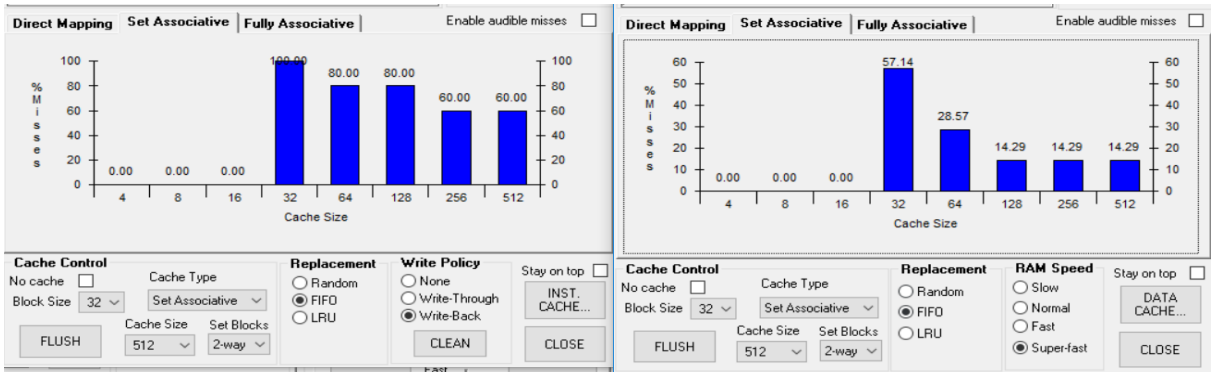
## Block size: 16, Data cache miss ratio:60%, Instruction cache miss ratio:14.29%



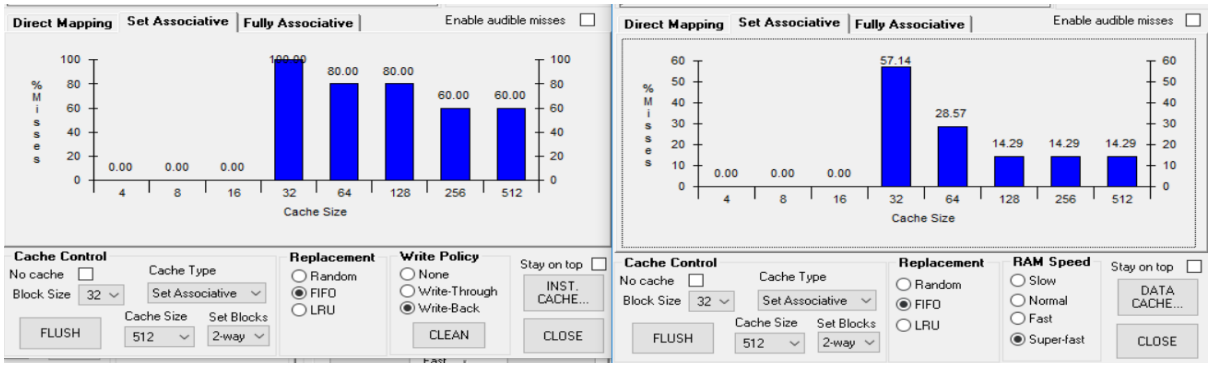

## Block size: 32, Data cache miss ratio:60%, Instruction cache miss ratio:14.29%



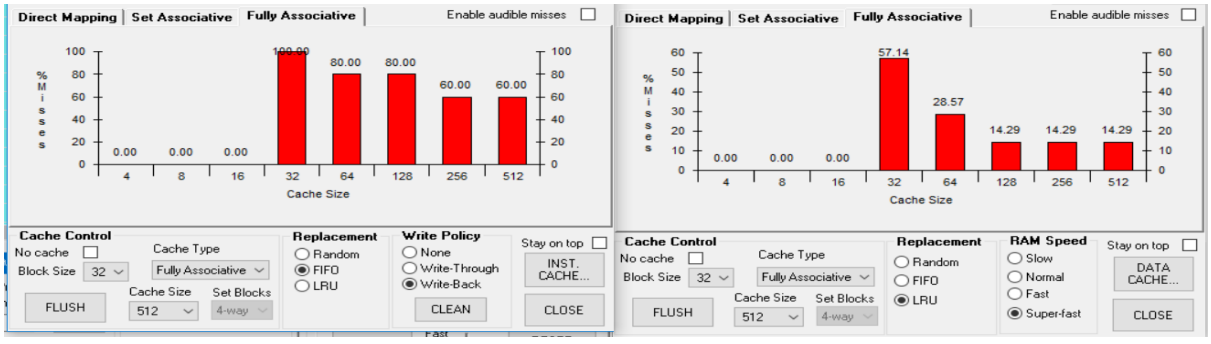
## Set Associative 2 way:




## Set Associative 4 way :




## Fully Associative :

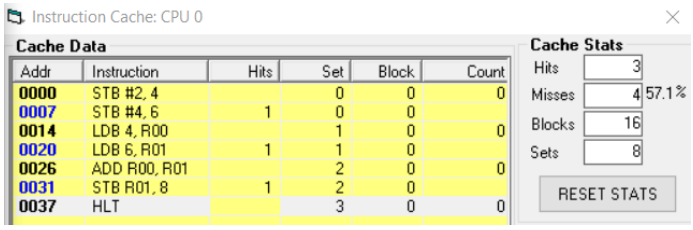
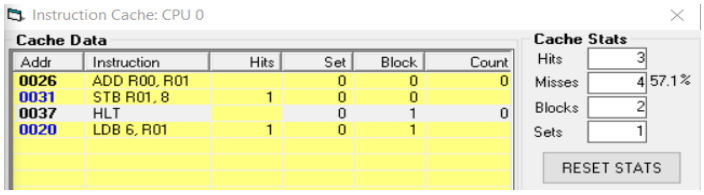
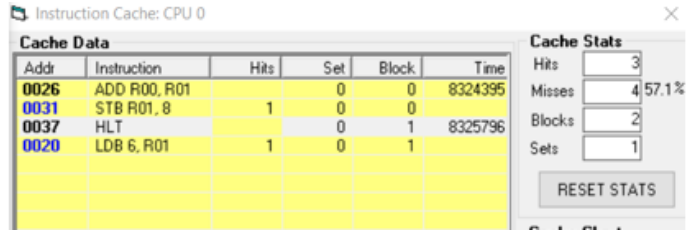
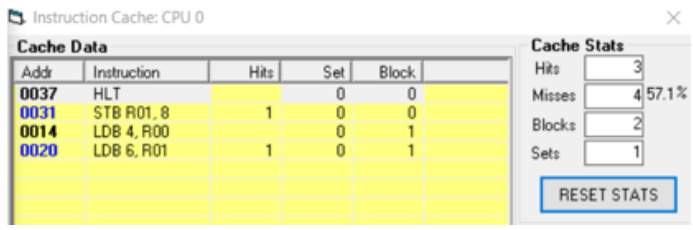



- From all these screen shots it is seen that for 16 blocks(i.e Number of blocks is 16) the miss ratio is same for all cache types i.e Direct Mapping, Set associative and Fully associative.

- **Cache plot should highlight results as shown in Table**

In order to notice the replacement policy for the given program, reduced the block number to 2 and the block size is 2 and the cache size is 4.

By this modification, it is seen that Instruction cache will hold maximum of four instructions.

| Graph | Comments |
|---|---|
|  | Reference screen shot when number of blocks is 16. All the 7 instructions of the program is loaded. |
| Below Screenshots show how replacement policy works when number of blocks is less. | |
| **LRU**<br>  | The last four instructions are cached which are the least recently used ones. |
| **FIFO**<br>  | The last four instructions are cached which is the “last in” instructions. i.e First in instructions are replaced. |
| **Random**<br>  | This cache can retain any instructions as it is random. From the screen shot it is seen that instructions are cached in random manner. |

## Inference

Observations from the parameters table and different cache types.

- The cache performance improves with larger block size. i.e Miss rate decreases with increase in block size.

- Replacement policy is applicable only for Set associative and full associative cache types. Not applicable for direct mapping.

- For a given table parameters the miss ratio is same irrespective of any cache types i.e Direct Mapping, Set associative and Fully associative.
  This is because the number of blocks is always 16 and the program deals with few instructions and memory locations which will not have any impact on miss ratio.

- Based on understanding it is seen that for large instruction set and data storage in diverse memory locations will give difference in the hit ratio using different replacement policies (LRU, FIFO and Random).

- In order to notice the replacement policy working for the given program, reduced the block number to 2 and the block size is 2 and the cache size is 4. By this modification, it is seen that Instruction cache will hold maximum of four instructions out of total seven instructions at any given point of time.

  LRU - Holds the last four instructions.

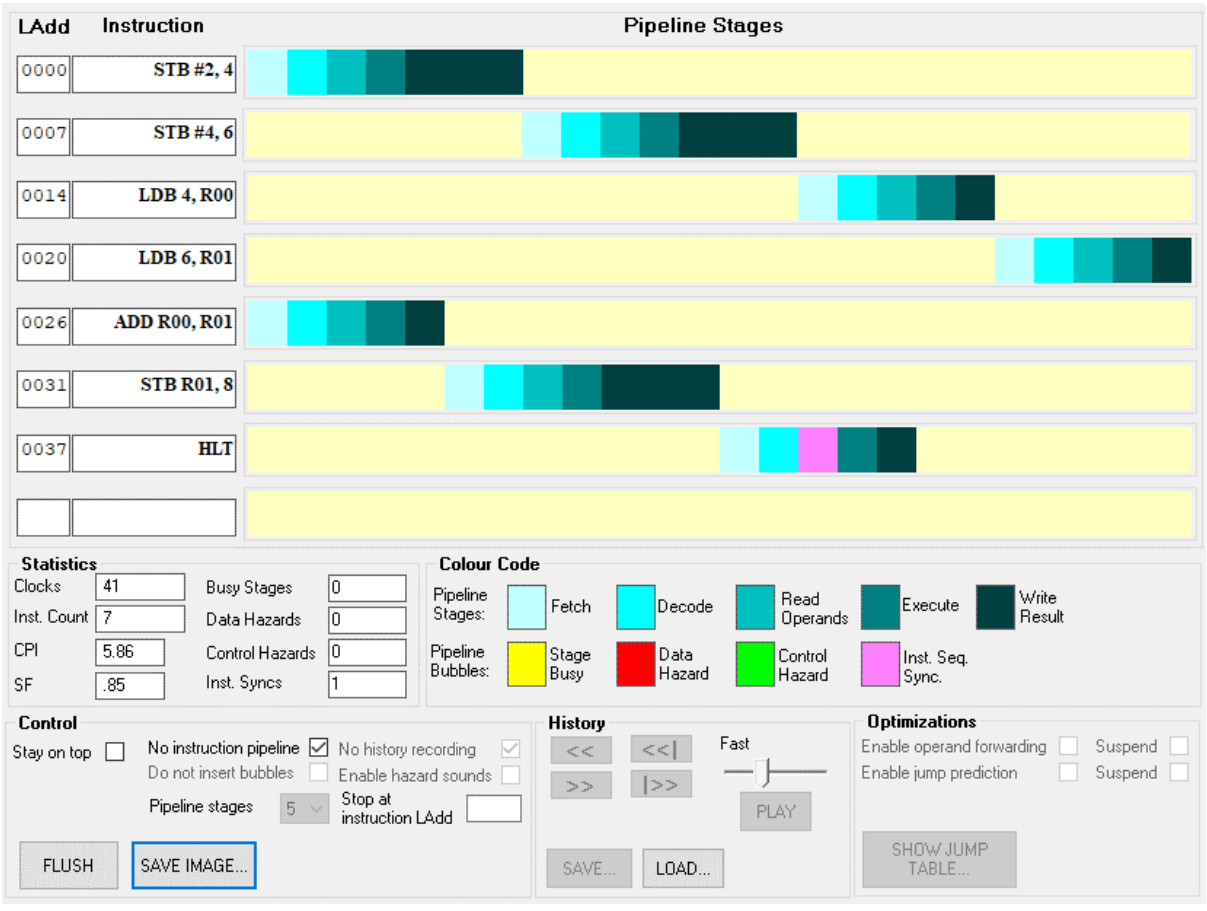
  FIFO - Holds the last four instructions.

  Random - Holds any four instructions.

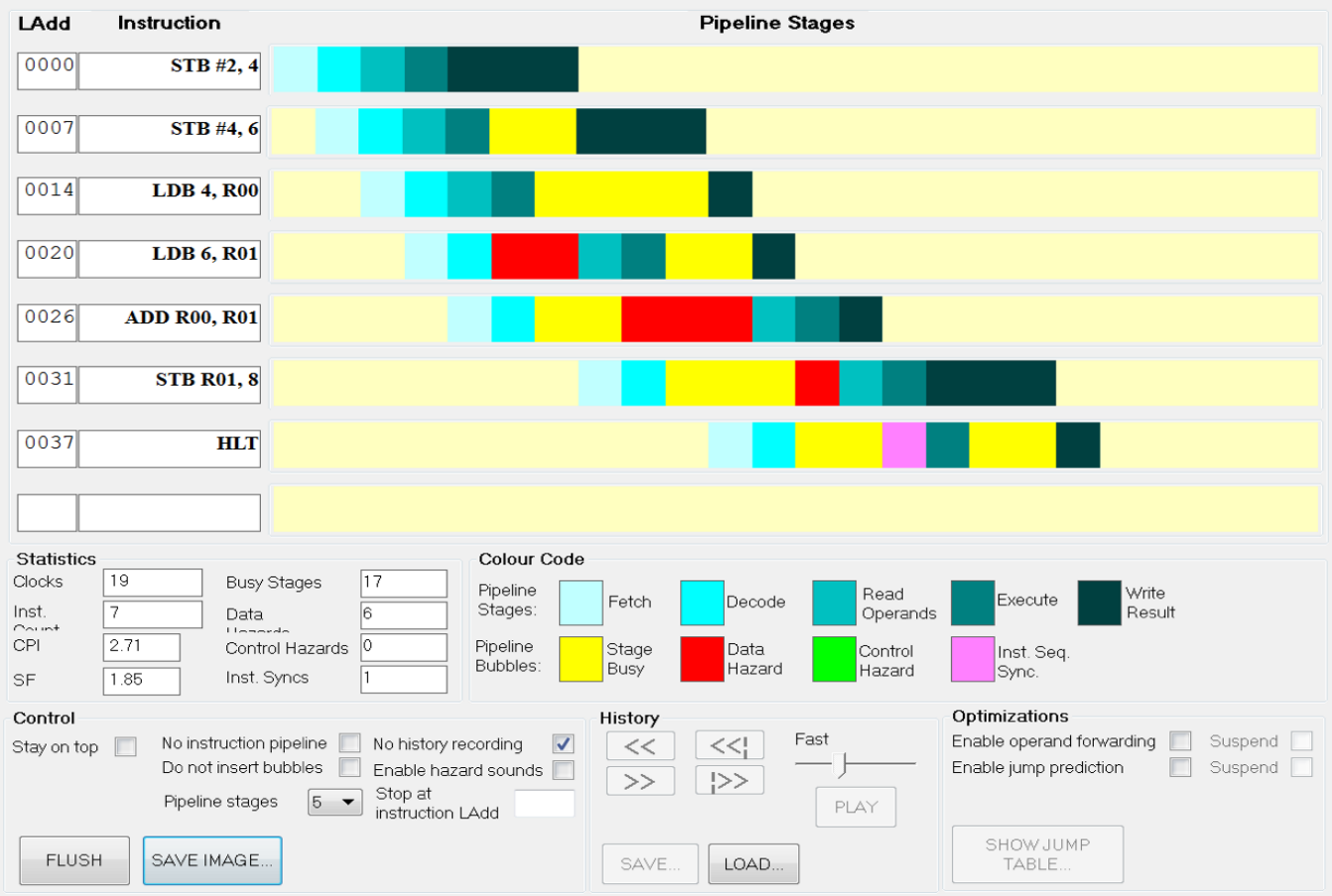
- **Pipelining concepts to be illustrated as below:**

| With/Without pipeline | Register values | Data memory Location | Data memory Location | Data memory Location | Instruction count | CPI | SF | clocks |
|---|---|---|---|---|---|---|---|---|
| Without pipeline | R00, R01 | 2 | 4 | 6 | 7 | 5.86 | 0.85 | 41 |
| With pipeline | R00, R01 | 2 | 4 | 6 | 7 | 2.71 | 1.85 | 19 |

**Without pipeline:**



## With pipeline:




### Pipeline Inference

- **Clocks –** The number of clocks reduced with instruction pipeline enabled. Value reduced from 41 to 19
- **CPI** – Clocks per instruction reduced to a greater extent with instruction pipeline enabled. Value reduced from 5.86 to 2.71
- There are few pipeline bubbles like stage busy, data hazard also present.

  **For Instance**: Data hazard in instruction **ADD R01,R02.**
  Data hazard happened because the ADD instruction on R01,R02 depends on the result of the previous instructions LDB 4,R00 and LDB 6,R01 which are still in the pipeline.