

## **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:**

<b>Sl. No.</b>	<b>Name</b>	<b>ID NO</b>	<b>Contribution</b>
1	<b>Praven H</b>	<b>2018AC04536</b>	<ul style="list-style-type: none"><li>• Program creation.</li><li>• Understanding and capture results for Direct mapping.</li><li>• Pipeline understanding-knowledge sharing to the team.</li><li>• Inference writing for cache concepts.</li></ul>
2	<b>Jaikumar T V</b>	<b>2018AC04556</b>	<ul style="list-style-type: none"><li>• Program modification.</li><li>• Understanding and capture results for Associative cache type(Set and Full associative)</li><li>• Inference writing for Pipeline concepts</li><li>• Documentation of results and consolidation.</li></ul>
3	<b>Prasanna R</b>	<b>2018AC04552</b>	<ul style="list-style-type: none"><li>• Program review.</li><li>• Analysis of Direct mapping cache type.</li><li>• Capture results without Pipeline.</li><li>• Inference writing for cache concepts</li></ul>
4	<b>Avinash Srivastava</b>	<b>2018AC04564</b>	<ul style="list-style-type: none"><li>• Initial analysis of program.</li><li>• Validation of Associative mapping results.</li><li>• Capture results with pipeline enabled.</li><li>• Inference writing for Pipeline concepts</li></ul>
5	<b>Ravi Teja Maripina</b>	<b>2018AC04558</b>	<ul style="list-style-type: none"><li>• Instruction set analysis to use in program.</li><li>• Re-run for all the input data provided and analysed the results.</li><li>• Verification of results.</li><li>• Inference writing based on final output.</li></ul>

## Solution:



AdditionOfTwoNumbers.sas

The screenshot displays a CPU emulator interface with several panels:

- CPU INSTRUCTIONS IN MEMORY (RAM):** A table listing instructions with columns PAdd, LAdd, Instruction, Base, and T.
- Cache - Pipeline:** Controls for Pipeline (Single/Dual), Select pipeline, and Cache (Data).
- Execution Unit:** Controls for PC, SP, SR, BR, SR Status Flags, CPU Mode (User/Kernel), IR, MAR, and MDR.
- SPECIAL CPU REGISTERS:** Displays PC (0), SP (8096), SR (0), BR (0), IR (HLT), MAR (8), and MDR (6).
- PROGRAM LIST:** A table with columns Name, Base, Start, and Type, showing a program named 'Addition'.
- PROGRAM STACK (RAM):** A table with columns Pos, Val (D), and Addr.
- GENERAL PURPOSE CPU REGISTERS:** A table with columns Reg, Val (D), C, and Val (D), showing registers R06 through R31.
- Program Control:** Buttons for STEP, RUN, STOP, and a slider for Fast/Slow execution.
- Advanced:** Buttons for COMPILER, OS, INPUT OUTPUT, VIRTUAL OS, and INTERRUPTS.
- Registers:** A panel for Reg Value, CHANGE, RESET ALL, Show Reg Access Status, and Select Register Set Size.

The screenshot displays a DATA MEMORY window with a table of memory addresses and values:

PAdd	LAdd	B0	B1	B2	B3	B4	B5	B6	B7	Data
PAGE 0										
0000	0000	00	00	00	00	02	00	04	00	.....
0008	0008	00	00	00	00	00	00	00	00	.....
0016	0016	00	00	00	00	00	00	00	00	.....
0024	0024	00	00	00	00	00	00	00	00	.....
0032	0032	00	00	00	00	00	00	00	00	.....
0040	0040	00	00	00	00	00	00	00	00	.....
0048	0048	00	00	00	00	00	00	00	00	.....
0056	0056	00	00	00	00	00	00	00	00	.....
0064	0064	00	00	00	00	00	00	00	00	.....
0072	0072	00	00	00	00	00	00	00	00	.....
0080	0080	00	00	00	00	00	00	00	00	.....
0088	0088	00	00	00	00	00	00	00	00	.....
0096	0096	00	00	00	00	00	00	00	00	.....
0104	0104	00	00	00	00	00	00	00	00	.....
0112	0112	00	00	00	00	00	00	00	00	.....

Below the table, there are sections for Initialisation Data and Debug control.

**Initialisation Data:**

- Integer Value: [ ]
- Boolean Value: False
- String Value: [ ]
- Address location: 248
- UPDATE button

**Debug control:**

- Check boxes to suspend when corresponding data byte addresses are modified by code.
- Buttons for B0, B1, B2, B3, B4, B5, B6, B7.
- RESET button
- UPDATE button

At the bottom, there are buttons for SHOW PAGE TABLE..., Pages: 1, Size: 256, RESET ALL, and CLOSE.

- Enter cache parameters in Table below:

**Data cache:**

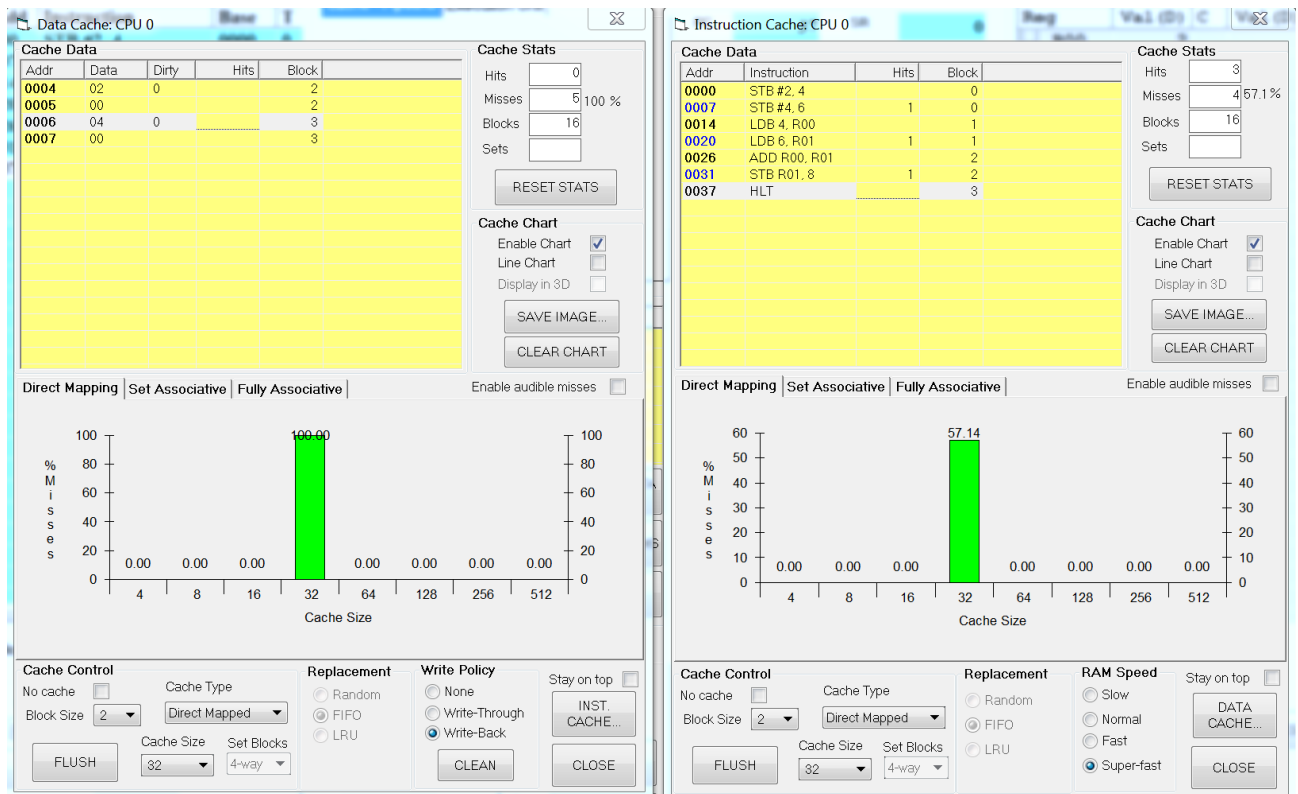
Block Size	No of blocks	Miss	Hit	Miss ratio
2	16	5	0	100%
4	16	4	0	80%
8	16	4	1	80%
16	16	3	2	60%
32	16	3	2	60%

**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%**



### Data Cache: CPU 0

Addr	Data	Dirty	Hits	Block
0004	02	0		1
0005	00			1
0006	04		1	1
0007	00			1

**Cache Stats**

Hits:   
 Misses:  80 %  
 Blocks:   
 Sets:

**Cache Chart**

Enable Chart ☒  
 Line Chart ☐  
 Display in 3D ☐

Direct Mapping | Set Associative | Fully Associative

Enable audible misses ☐

**Cache Control**

No cache ☐  
 Block Size:

Cache Type:

Cache Size:   
 Set Blocks:

**Replacement**

☐ Random  
☐ FIFO  
☐ LRU

**Write Policy**

☐ None  
☐ Write-Through  
☒ Write-Back

Stay on top ☐

### Instruction Cache: CPU 0

Addr	Instruction	Hits	Block
0000	STB #2, 4		0
0007	STB #4, 6		0
0014	LDB 4, R00	1	0
0020	LDB 6, R01	1	0
0026	ADD R00, R01		1
0031	STB R01, 8	1	1
0037	HLT	1	1

**Cache Stats**

Hits:   
 Misses:  28.5 %  
 Blocks:   
 Sets:

**Cache Chart**

Enable Chart ☒  
 Line Chart ☐  
 Display in 3D ☐

Direct Mapping | Set Associative | Fully Associative

Enable audible misses ☐

**Cache Control**

No cache ☐  
 Block Size:

Cache Type:

Cache Size:   
 Set Blocks:

**Replacement**

☐ Random  
☐ FIFO  
☐ LRU

**RAM Speed**

☐ Slow  
☐ Normal  
☐ Fast  
☒ Super-fast

Stay on top ☐

### Data Cache: CPU 0

Addr	Data	Dirty	Hits	Block
0000	02	0		0
0001	00			0
0002	00			0
0003	00			0
0004	02			0
0005	00			0
0006	04		1	0
0007	00			0

**Cache Stats**

Hits:   
 Misses:  80 %  
 Blocks:   
 Sets:

**Cache Chart**

Enable Chart ☒  
 Line Chart ☐  
 Display in 3D ☐

Direct Mapping | Set Associative | Fully Associative | Enable audible misses ☐

**Cache Control**

No cache ☐ Cache Type:   
 Block Size:   
 Cache Size:  Set Blocks:

**Replacement**

☐ Random  
☐ FIFO  
☐ LRU

**Write Policy**

☐ None  
☐ Write-Through  
☒ Write-Back

Stay on top ☐

### Instruction Cache: CPU 0

Addr	Instruction	Hits	Block
0000	STB #2. 4		0
0007	STB #4. 6	1	0
0014	LDB 4. R00	1	0
0020	LDB 6. R01	1	0
0026	ADD R00. R01	1	0
0031	STB R01. 8	1	0
0037	HLT	1	0

**Cache Stats**

Hits:   
 Misses:  14.2 %  
 Blocks:   
 Sets:

**Cache Chart**

Enable Chart ☒  
 Line Chart ☐  
 Display in 3D ☐

Direct Mapping | Set Associative | Fully Associative | Enable audible misses ☐

**Cache Control**

No cache ☐ Cache Type:   
 Block Size:   
 Cache Size:  Set Blocks:

**Replacement**

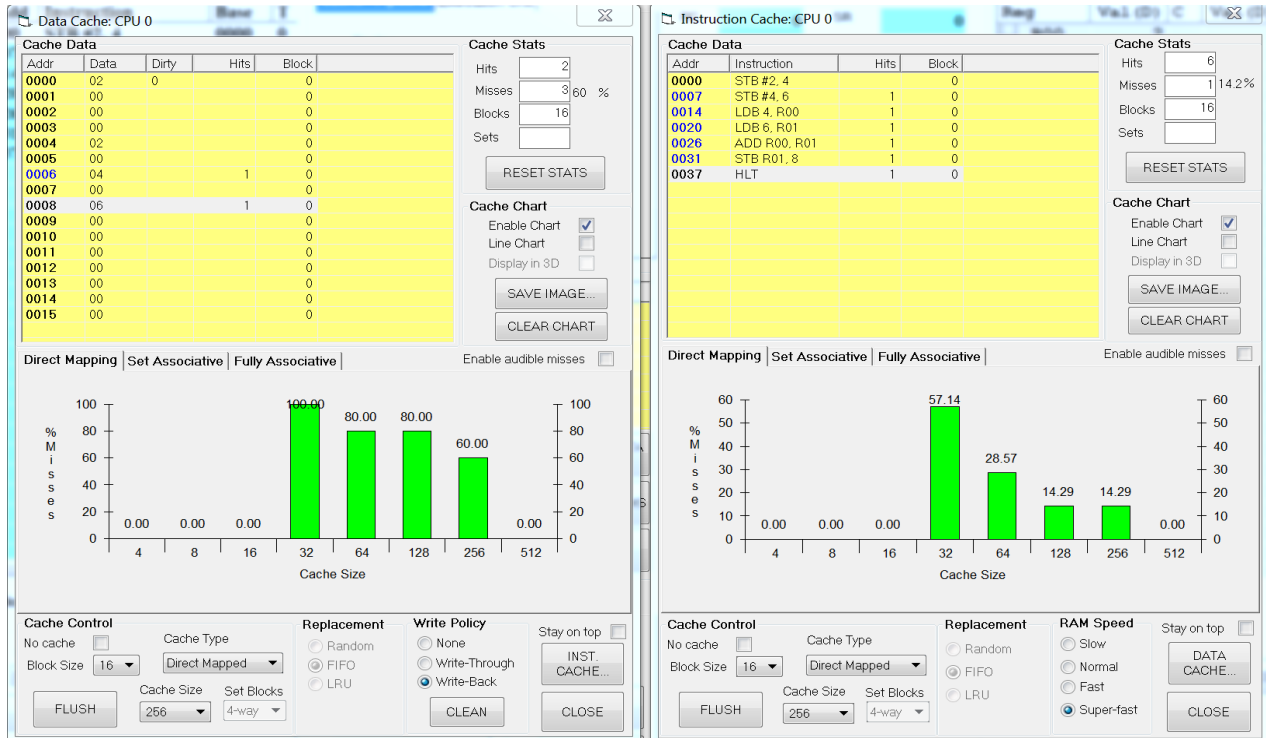
☐ Random  
☒ FIFO  
☐ LRU

**RAM Speed**

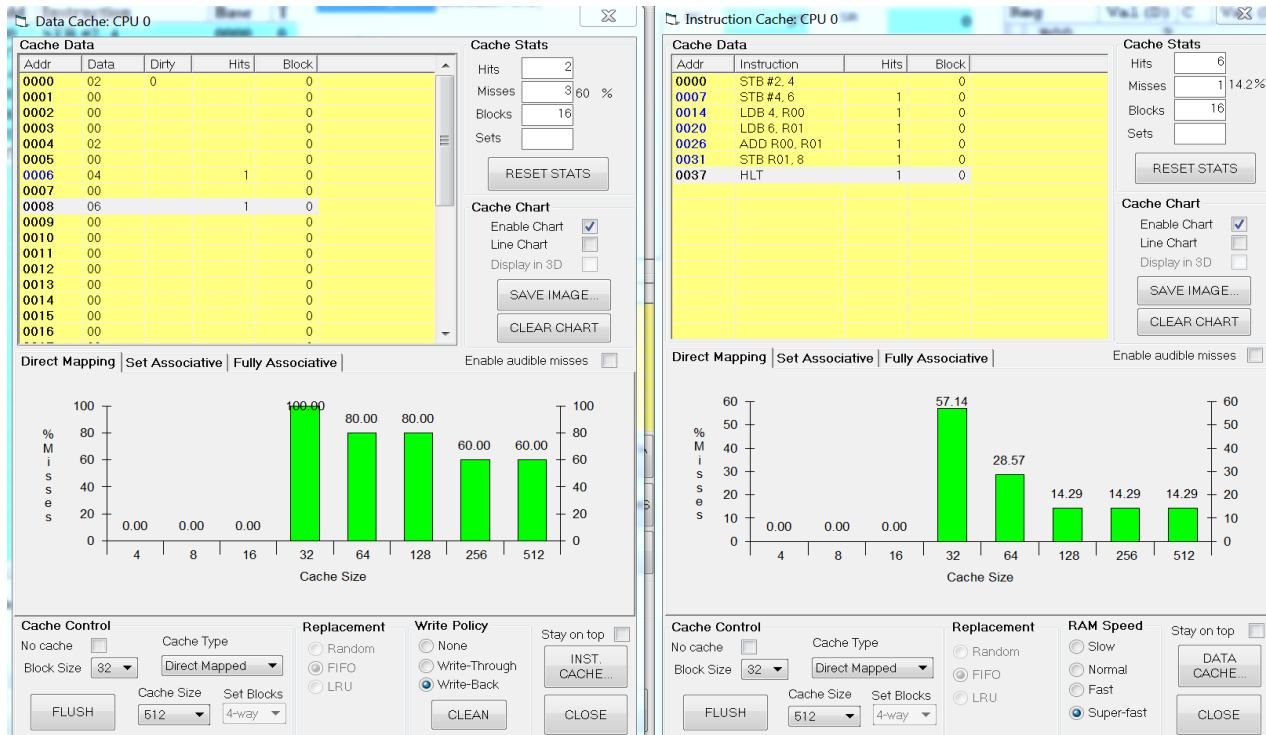
☐ Slow  
☐ Normal  
☐ Fast  
☒ Super-fast

Stay on top ☐

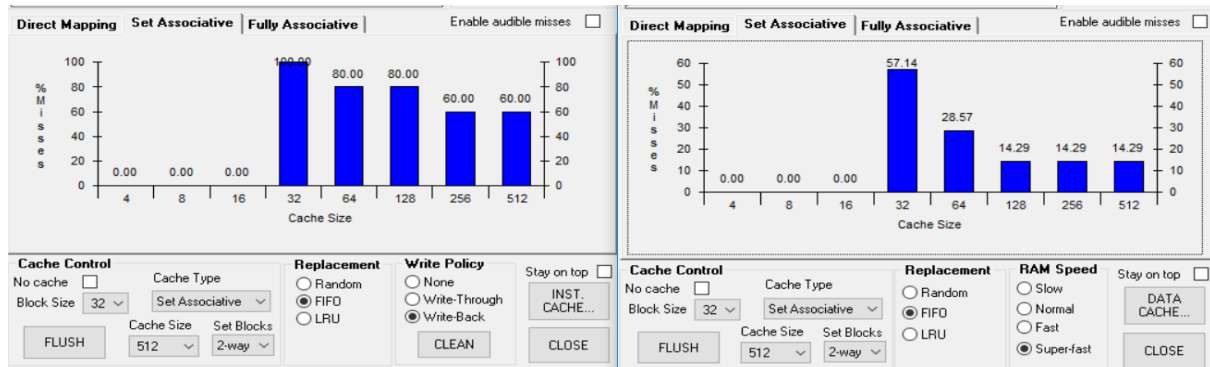
**Block size: 16, Data cache miss ratio:60%, Instruction cache miss ratio:14.29%**



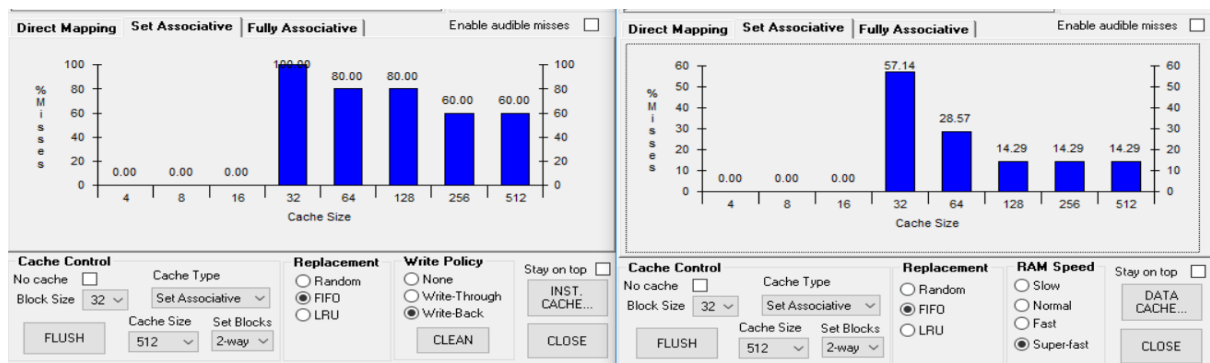
**Block size: 32, Data cache miss ratio:60%, Instruction cache miss ratio:14.29%**



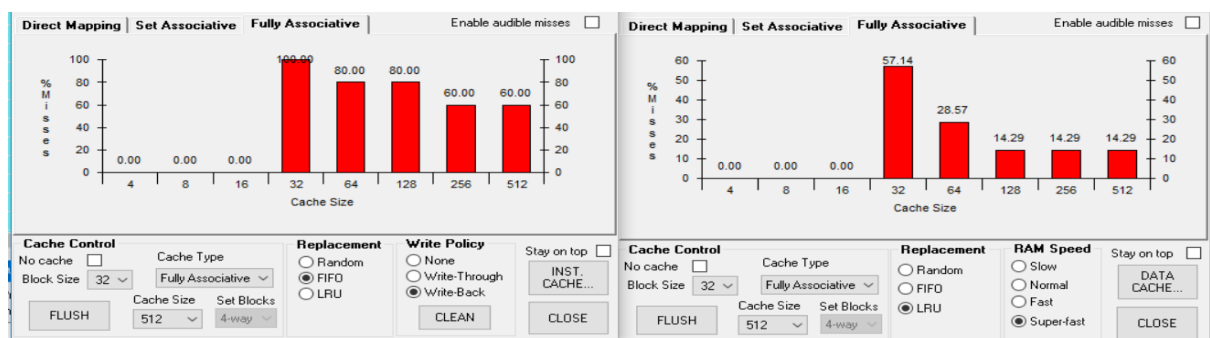
## Set Associative 2 way:



## 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

Instruction Cache: CPU 0

Addr	Instruction	Hits	Set	Block	Count
0000	STB #2, 4		0	0	0
0007	STB #4, 6	1	0	0	0
0014	LDB 4, R00		1	0	0
0020	LDB 6, R01	1	1	0	0
0026	ADD R00, R01		2	0	0
0031	STB R01, 8	1	2	0	0
0037	HLT		3	0	0

Cache Stats

Hits: 3  
Misses: 4 57.1%  
Blocks: 16  
Sets: 8

RESET STATS

## 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

Instruction Cache: CPU 0

Addr	Instruction	Hits	Set	Block	Count
0026	ADD R00, R01		0	0	0
0031	STB R01, 8	1	0	0	0
0037	HLT		0	1	0
0020	LDB 6, R01	1	0	1	0

Cache Stats

Hits: 3  
Misses: 4 57.1%  
Blocks: 2  
Sets: 1

RESET STATS

The last four instructions are cached which are the least recently used ones.

### FIFO

Instruction Cache: CPU 0

Addr	Instruction	Hits	Set	Block	Time
0026	ADD R00, R01		0	0	8324395
0031	STB R01, 8	1	0	0	
0037	HLT		0	1	8325796
0020	LDB 6, R01	1	0	1	

Cache Stats

Hits: 3  
Misses: 4 57.1%  
Blocks: 2  
Sets: 1

RESET STATS

The last four instructions are cached which is the "last in" instructions. i.e First in instructions are replaced.

### Random

Instruction Cache: CPU 0

Addr	Instruction	Hits	Set	Block
0037	HLT		0	0
0031	STB R01, 8	1	0	0
0014	LDB 4, R00		0	1
0020	LDB 6, R01	1	0	1

Cache Stats

Hits: 3  
Misses: 4 57.1%  
Blocks: 2  
Sets: 1

RESET STATS

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.

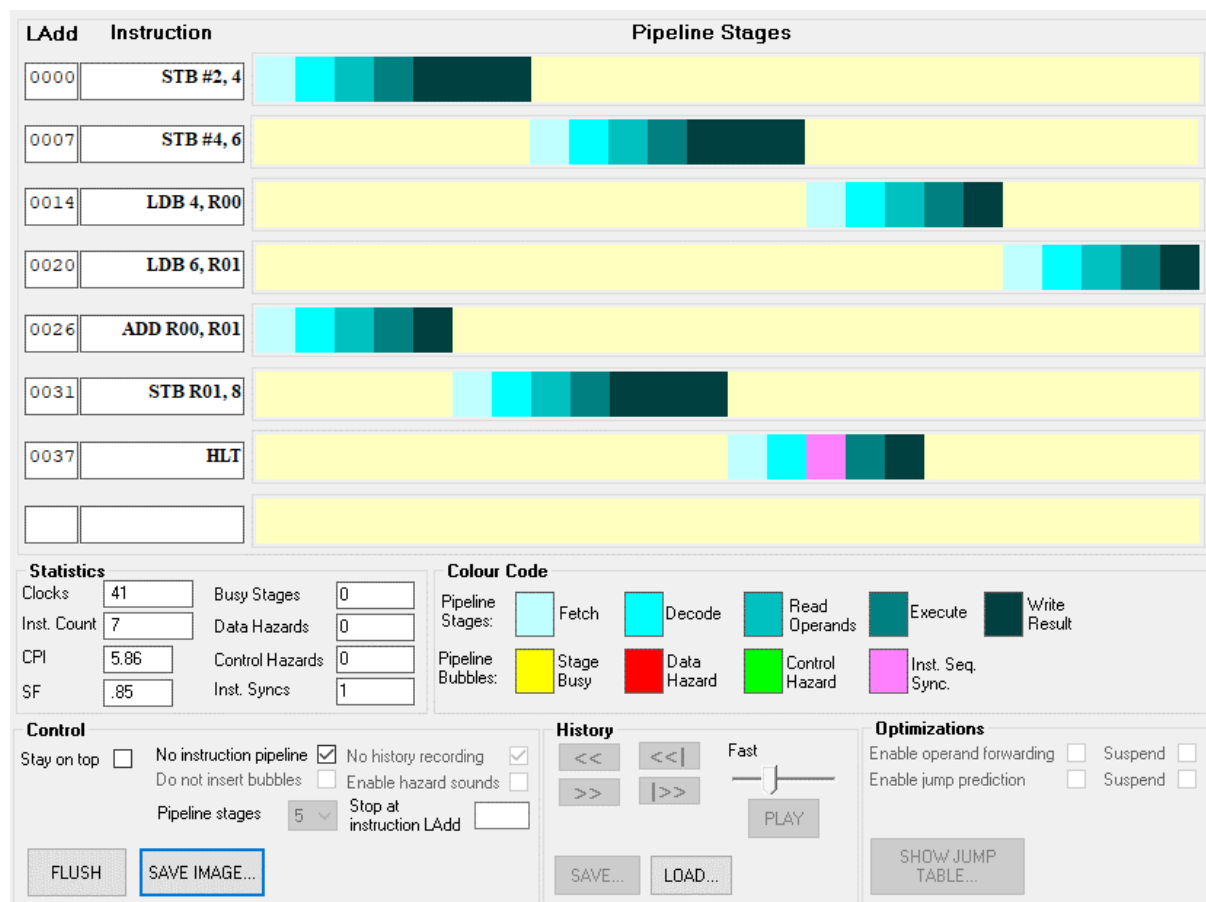
- The cache performance improves with larger block size. i.e Miss rate decreases with increase in block size.
- Replacement policy is applicable only for Set associative and full associative cache types. Not applicable for direct mapping.
- For a given table parameters the miss ratio is same irrespective 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.