To validate the program, we created test cases that cover different cache sizes and different number of ways. Addresses were created to test each scenario. The results were calculated. The cache was simulated comprehensively in test case 1 at each step. The calculated results were compared to the output results of the program to validate its performance.

Test Case 1:

Cache Size: 64 bytes

Cache Line size: 8 bytes

No. ways: 1

Number of lines = cache size / cache line size = 8 lines

Index bits = log2(number of lines) = 3 bits

Offset bits = log2(cache line size) = 3 bits

The cache in the beginning:

Index	Valid	Tag
0	0	-
1	0	-
2	0	-
3	0	-
4	0	-
5	0	-
6	0	-
7	0	-

Table (1)

Addresses:

1)

Address	Index	Tag	Hit/Miss	Replace
0	0	0	M	N

The cache after accessing the address:

Index	Valid	Tag
0	1	0
1	0	-
2	0	-
3	0	-

4	0	-
5	0	-
6	0	-
7	0	-

Table (2)

2)

Address	Index	Tag	Hit/Miss	Replace
128	0	2	M	Y

The cache after accessing the address:

Index	Valid	Tag
0	1	2
1	0	-
2	0	-
3	0	-
4	0	-
5	0	-
6	0	-
7	0	-

Table (3)

3)

Address	Index	Tag	Hit/Miss	Replace
184	7	2	M	N

The cache after accessing the address:

Index	Valid	Tag
0	1	2
1	0	-
2	0	-
3	0	-
4	0	-
5	0	-

6	0	-
7	1	2

Table (4)

4)

Address	Index	Tag	Hit/Miss	Replace
104	5	1	M	N

The cache after accessing the address:

Index	Valid	Tag
0	1	2
1	0	-
2	0	-
3	0	-
4	0	-
5	1	1
6	0	-
7	1	2

Table (5)

5)

Address	Index	Tag	Hit/Miss	Replace
296	5	4	M	Y

The cache after accessing the address:

Index	Valid	Tag
0	1	2
1	0	-
2	0	-
3	0	-
4	0	-
5	1	4
6	0	-
7	1	2

Table (6)

6)

Address	Index	Tag	Hit/Miss	Replace
187	7	2	H	N

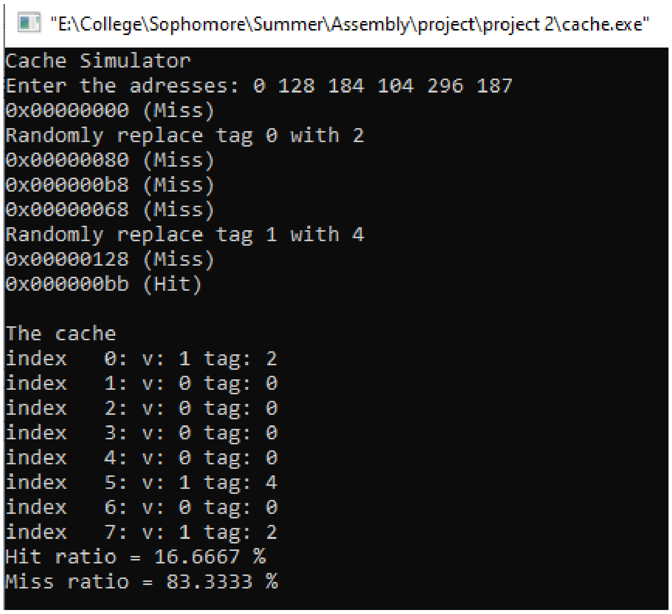
The cache after accessing the address:

Index	Valid	Tag
0	1	2
1	0	-
2	0	-
3	0	-
4	0	-
5	1	4
6	0	-
7	1	2

Table (7)

Hit Ratio = 1/6 = 16.67%

Results of the program:



"E:\College\Sophomore\Summer\Assembly\project\project 2\cache.exe"

```
Cache Simulator
Enter the adresses: 0 128 184 104 296 187
0x00000000 (Miss)
Randomly replace tag 0 with 2
0x00000080 (Miss)
0x000000b8 (Miss)
0x00000068 (Miss)
Randomly replace tag 1 with 4
0x00000128 (Miss)
0x000000bb (Hit)

The cache
index   0: v: 1 tag: 2
index   1: v: 0 tag: 0
index   2: v: 0 tag: 0
index   3: v: 0 tag: 0
index   4: v: 0 tag: 0
index   5: v: 1 tag: 4
index   6: v: 0 tag: 0
index   7: v: 1 tag: 2
Hit ratio = 16.6667 %
Miss ratio = 83.3333 %
```

Pic (1)

The results in Pic (1) show that all the addresses had the correct HIT/MISS result, and the end cache matched the expected one. This validates the program for this test case.

Test Case 2:

Cache Size: 64 bytes

Cache Line size: 8 bytes

No. ways: 2

Number of lines = cache size / cache line size = 8 lines

Index bits = log2(number of lines) = 3 bits

Offset bits = log2(cache line size) = 3 bits

The cache in the beginning:

Index (set)	Block 0		Block 1	
	Valid	Tag	Valid	Tag
0	0	-	0	-
1	0	-	0	-
2	0	-	0	-
3	0	-	0	-
4	0	-	0	-
5	0	-	0	-
6	0	-	0	-
7	0	-	0	-

Table (8)

Using the same logic as in above but replacing the saved address only when all the blocks in a set (specific index) are filled with different addresses.

Address	Index	Tag	Hit/Miss	Replace
0	0	0	M	N
128	0	2	M	N
184	7	2	M	N
104	5	1	M	N
296	5	4	M	N
187	7	2	H	N
109	5	1	H	N
67	0	1	M	Y (Random)
259	0	4	M	Y (Random)
1	0	0	M/H (According to random replacement)	Y/N

Table (9)

Hit Ratio = 3/10 = 0.3 = 30% → if tag 0 was not replaced

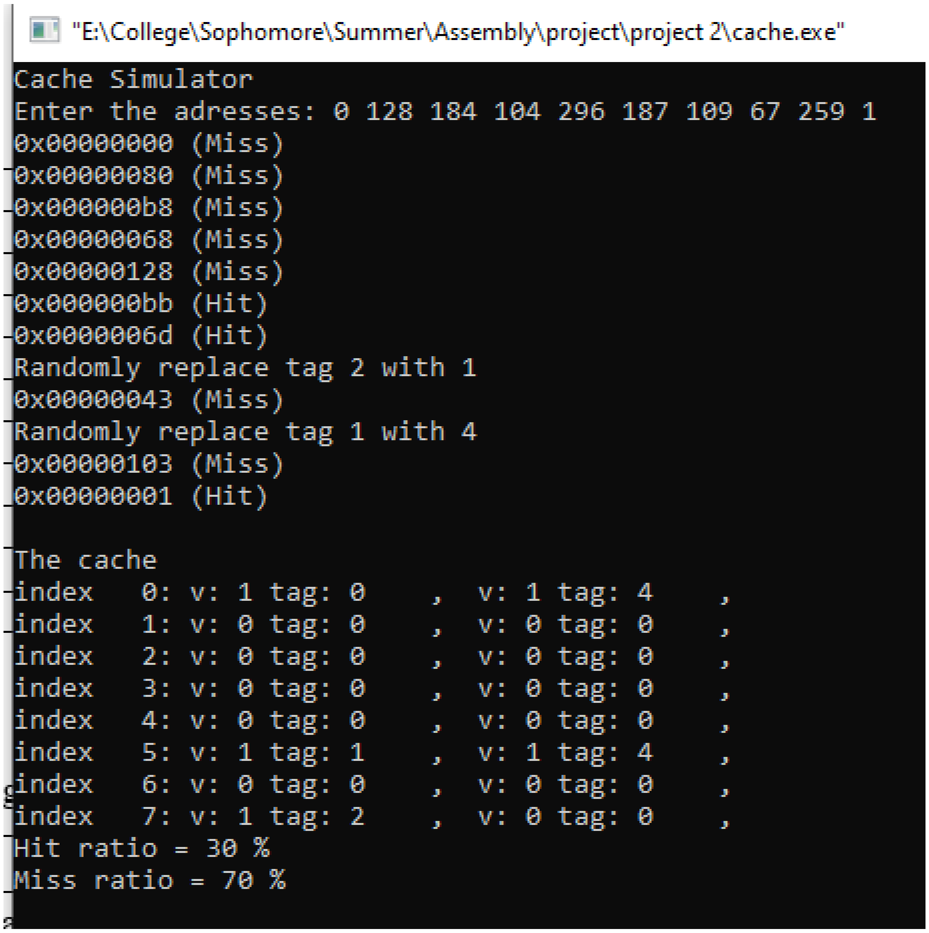
Hit Ratio = 2/10 = 1/5 = 0.2 = 20% → if tag 0 was replaced

Due to Random Replacement happened in index 0, we are sure that 4 is present in one way and either 0, 1, or 2 in the other:

Index (set)	Block 1		Block 2	
	Valid	Tag	Valid	Tag
0	1	4	1	0/1/2
1	0	-	0	-
2	0	-	0	-
3	0	-	0	-
4	0	-	0	-
5	1	1	1	4
6	0	-	0	-
7	1	2	0	-

Table (10)

The result of the program:


"E:\College\Sophomore\Summer\Assembly\project\project 2\cache.exe"

```
Cache Simulator
Enter the adresses: 0 128 184 104 296 187 109 67 259 1
0x00000000 (Miss)
0x00000080 (Miss)
0x000000b8 (Miss)
0x00000068 (Miss)
0x00000128 (Miss)
0x000000bb (Hit)
0x0000006d (Hit)
Randomly replace tag 2 with 1
0x00000043 (Miss)
Randomly replace tag 1 with 4
0x00000103 (Miss)
0x00000001 (Hit)

The cache
index   0: v: 1 tag: 0    ,  v: 1 tag: 4    ,
index   1: v: 0 tag: 0    ,  v: 0 tag: 0    ,
index   2: v: 0 tag: 0    ,  v: 0 tag: 0    ,
index   3: v: 0 tag: 0    ,  v: 0 tag: 0    ,
index   4: v: 0 tag: 0    ,  v: 0 tag: 0    ,
index   5: v: 1 tag: 1    ,  v: 1 tag: 4    ,
index   6: v: 0 tag: 0    ,  v: 0 tag: 0    ,
index   7: v: 1 tag: 2    ,  v: 0 tag: 0    ,
Hit ratio = 30 %
Miss ratio = 70 %
```

Pic (2)

The results in Pic (2) show that all the addresses had the correct HIT/MISS result, and the end cache matched the expected one. This validates the program for this test case.

Test Case 3:

1-way set associative cache.

Cache line size: 8 bytes (2 words)

Cache size: 128 bytes

Number of ways: 1

Memory address is 16 bits

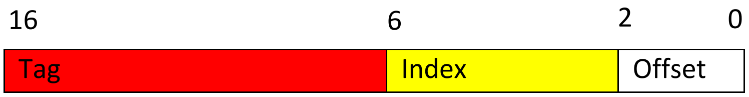
Number of lines = cache size / cache line size = 128 / 8 = 16 line

Offset bits = $\log_2$(cache line size) = $\log_2(8)$ = 3 bits

Index bits = $\log_2$(number of lines) = $\log_2(16)$ = 4 bits

Tag bits = physical Address – (Index Bits + offset Bits) = 16 – (3+4) = 16 – 7 = 9 bits

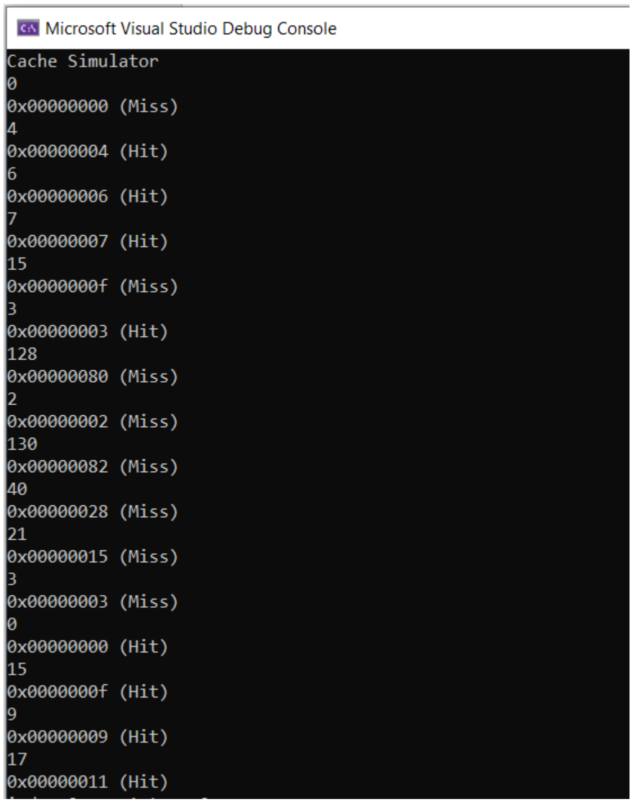
Byte Address:



Byte Address	Byte Address in Binary	Address Index	Address Tag	Hit/ Miss	Replace
0	0000000000000000	0000	000000000	M	N
4	0000000000000100	0000	000000000	H	N
6	0000000000000110	0000	000000000	H	N
7	0000000000000111	0000	000000000	H	N
15	0000000000001111	0001	000000000	M	N
3	0000000000000011	0000	000000000	H	N
128	0000000010000000	0000	000000001	M	Y
2	0000000000000010	0000	000000000	M	Y
130	0000000010000010	0000	000000001	M	Y
40	0000000000101000	0101	000000000	M	N
21	0000000000010101	0010	000000000	M	N
3	0000000000000011	0000	000000000	M	Y
0	0000000000000000	0000	000000000	H	N
15	0000000000001111	0001	000000000	H	N
9	0000000000001001	0001	000000000	H	N
17	0000000000010001	0010	000000000	H	N

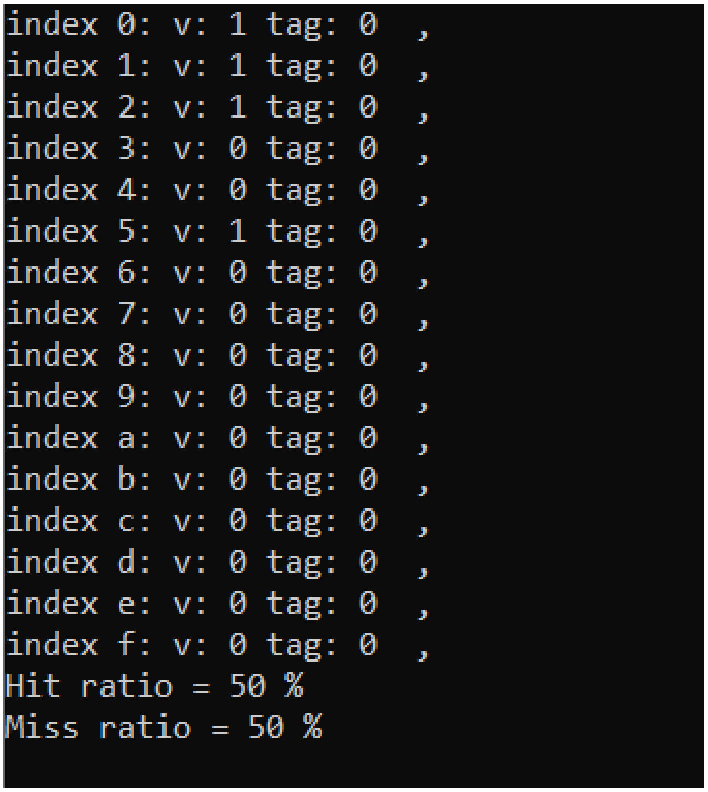
Table (11)

Hit ratio = $\frac{8}{16} * 100\ \% = 50\%$



```
Cache Simulator
0
0x00000000 (Miss)
4
0x00000004 (Hit)
6
0x00000006 (Hit)
7
0x00000007 (Hit)
15
0x0000000f (Miss)
3
0x00000003 (Hit)
128
0x00000080 (Miss)
2
0x00000002 (Miss)
130
0x00000082 (Miss)
40
0x00000028 (Miss)
21
0x00000015 (Miss)
3
0x00000003 (Miss)
0
0x00000000 (Hit)
15
0x0000000f (Hit)
9
0x00000009 (Hit)
17
0x00000011 (Hit)
```

Pic (3)



```
index 0: v: 1 tag: 0  ,
index 1: v: 1 tag: 0  ,
index 2: v: 1 tag: 0  ,
index 3: v: 0 tag: 0  ,
index 4: v: 0 tag: 0  ,
index 5: v: 1 tag: 0  ,
index 6: v: 0 tag: 0  ,
index 7: v: 0 tag: 0  ,
index 8: v: 0 tag: 0  ,
index 9: v: 0 tag: 0  ,
index a: v: 0 tag: 0  ,
index b: v: 0 tag: 0  ,
index c: v: 0 tag: 0  ,
index d: v: 0 tag: 0  ,
index e: v: 0 tag: 0  ,
index f: v: 0 tag: 0  ,
Hit ratio = 50 %
Miss ratio = 50 %
```

Pic (4)

The results in Pic (3) and Pic (4) show that all the addresses had the correct HIT/MISS result, and the end cache matched the expected one. This validates the program for this test case.

Test Case 4:

2-way set associative cache.

Cache line size: 8 bytes (2 words)

Cache size: 128 bytes

Memory address is 16 bits

Number of ways: 2

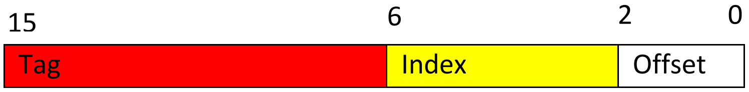
Number of lines = cache size / cache line size = 128 / 8 = 16 line

Offset bits = log2(cache line size) = log2(8) = 3 bits

Index bits = log2(number of lines) = log2(16) = 4 bits

Tag bits = physical Address – (Index Bits + offset Bits) = 16 – (3+4) = 16 – 7 = 9 bits

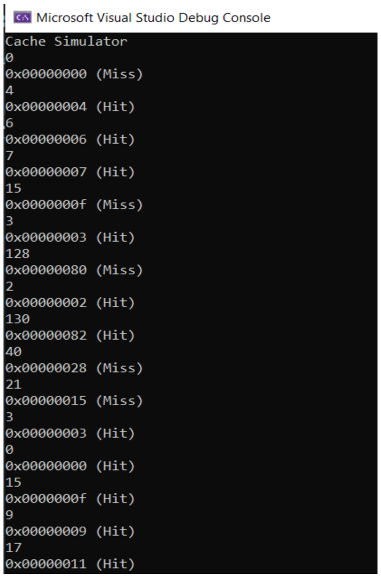
Byte Address:



Byte Address	Byte Address in Binary	Address Index	Address Tag	Position in set	Hit/ Miss	Replace
0	0000000000000000	0000	000000000	0	M	N
4	0000000000000100	0000	000000000	0	H	N
6	0000000000000110	0000	000000000	0	H	N
7	0000000000000111	0000	000000000	0	H	N
15	0000000000001111	0001	000000000	0	M	N
3	0000000000000011	0000	000000000	0	H	N
128	0000000010000000	0000	000000001	1	M	N
2	0000000000000010	0000	000000000	0	H	N
130	0000000010000010	0000	000000001	1	H	N
40	0000000000101000	0101	000000000	0	M	N
21	0000000000010101	0010	000000000	0	M	N
3	0000000000000011	0000	000000000	0	H	N
0	0000000000000000	0000	000000000	0	H	N
15	0000000000001111	0001	000000000	0	H	N
9	0000000000001001	0001	000000000	0	H	N
17	0000000000010001	0010	000000000	0	H	N

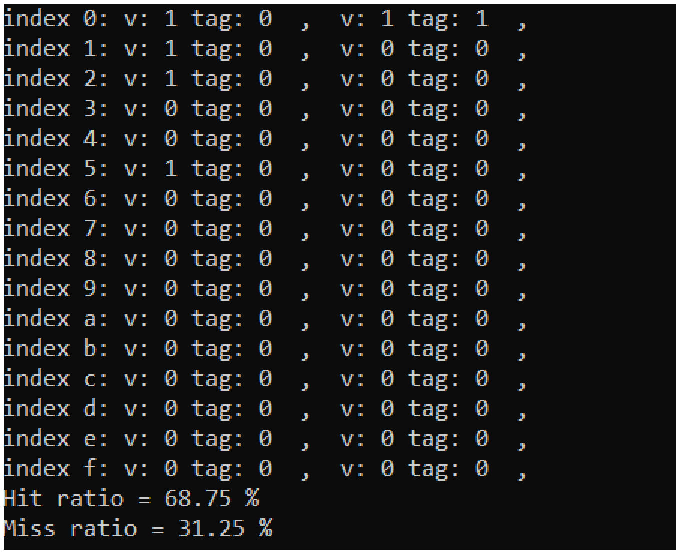
Table (12)

Hit ratio = $\frac{11}{16} * 100\ \% = 68.75\%$



```
Cache Simulator
0
0x00000000 (Miss)
4
0x00000004 (Hit)
6
0x00000006 (Hit)
7
0x00000007 (Hit)
15
0x0000000f (Miss)
3
0x00000003 (Hit)
128
0x00000080 (Miss)
2
0x00000002 (Hit)
130
0x00000082 (Hit)
40
0x00000028 (Miss)
21
0x00000015 (Miss)
3
0x00000003 (Hit)
0
0x00000000 (Hit)
15
0x0000000f (Hit)
9
0x00000009 (Hit)
17
0x00000011 (Hit)
```

Pic (5)



```
index 0: v: 1 tag: 0  ,  v: 1 tag: 1  ,
index 1: v: 1 tag: 0  ,  v: 0 tag: 0  ,
index 2: v: 1 tag: 0  ,  v: 0 tag: 0  ,
index 3: v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 4: v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 5: v: 1 tag: 0  ,  v: 0 tag: 0  ,
index 6: v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 7: v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 8: v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 9: v: 0 tag: 0  ,  v: 0 tag: 0  ,
index a: v: 0 tag: 0  ,  v: 0 tag: 0  ,
index b: v: 0 tag: 0  ,  v: 0 tag: 0  ,
index c: v: 0 tag: 0  ,  v: 0 tag: 0  ,
index d: v: 0 tag: 0  ,  v: 0 tag: 0  ,
index e: v: 0 tag: 0  ,  v: 0 tag: 0  ,
index f: v: 0 tag: 0  ,  v: 0 tag: 0  ,
Hit ratio = 68.75 %
Miss ratio = 31.25 %
```

Pic (6)

The results in Pic (5) and Pic (6) show that all the addresses had the correct HIT/MISS result, and the end cache matched the expected one. This validates the program for this test case.

Test Case 5:

4-way set associative cache.

Cache line size: 1 byte

Cache size: 32 bytes

Memory address is 16 bits

Number of ways: 4

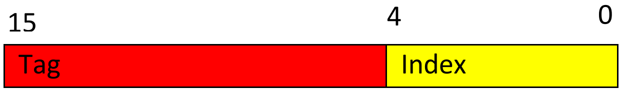
Number of lines = cache size / cache line size = 32/ 1 = 32 line

Offset bits = log2(cache line size) = log2(1) = 0 bits

Index bits = log2(number of lines) = log2(32) = 5 bits

Tag bits = physical Address – (Index Bits + offset Bits) = 16 – (0+5) = 16 – 5 = 11 bits

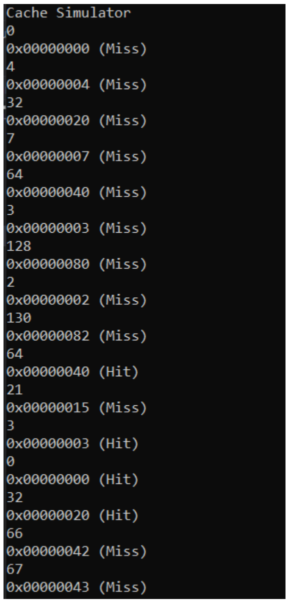
Byte Address:



Byte Address	Byte Address in Binary	Address Index	Address Tag	Position in set	Hit/ Miss	Replace
0	0000000000000000	00000	000000000	0	M	N
4	0000000000000100	00100	000000000	0	M	N
32	0000000000100000	00000	000000001	1	M	N
7	0000000000000111	00111	000000000	0	M	N
64	0000000001000000	00000	000000010	2	M	N
3	0000000000000011	00011	000000000	0	M	N
128	0000000010000000	00000	000000100	3	M	N
2	0000000000000010	00010	000000000	0	M	N
130	0000000010000010	00010	000000100	1	M	N
64	0000000001000000	00000	000000010	2	H	N
21	0000000000010101	10101	000000000	0	M	N
3	0000000000000011	00011	000000000	0	H	N
0	0000000000000000	00000	000000000	0	H	N
32	0000000000100000	00000	000000001	1	H	N
66	0000000001000010	00010	000000010	2	M	N
67	0000000001000011	00011	000000010	1	M	N

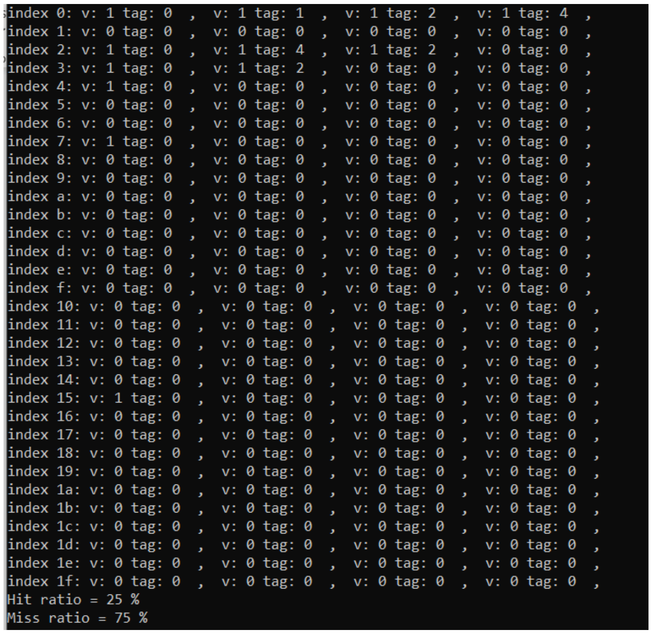
Table (13)

Hit ratio = $\frac{4}{16} * 100\ \% = 25\%$



```
Cache Simulator
0
0x00000000 (Miss)
4
0x00000004 (Miss)
32
0x00000020 (Miss)
7
0x00000007 (Miss)
64
0x00000040 (Miss)
3
0x00000003 (Miss)
128
0x00000080 (Miss)
2
0x00000002 (Miss)
130
0x00000082 (Miss)
64
0x00000040 (Hit)
21
0x00000015 (Miss)
3
0x00000003 (Hit)
0
0x00000000 (Hit)
32
0x00000020 (Hit)
66
0x00000042 (Miss)
67
0x00000043 (Miss)
```

Pic (7)



```
index 0: v: 1 tag: 0  ,  v: 1 tag: 1  ,  v: 1 tag: 2  ,  v: 1 tag: 4  ,
index 1: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 2: v: 1 tag: 0  ,  v: 1 tag: 4  ,  v: 1 tag: 2  ,  v: 0 tag: 0  ,
index 3: v: 1 tag: 0  ,  v: 1 tag: 2  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 4: v: 1 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 5: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 6: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 7: v: 1 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 8: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 9: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index a: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index b: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index c: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index d: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index e: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index f: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 10: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 11: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 12: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 13: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 14: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 15: v: 1 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 16: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 17: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 18: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 19: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 1a: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 1b: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 1c: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 1d: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 1e: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
index 1f: v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,  v: 0 tag: 0  ,
Hit ratio = 25 %
Miss ratio = 75 %
```

Pic (8)

The results in Pic (7) and Pic (8) show that all the addresses had the correct HIT/MISS result, and the end cache matched the expected one. This validates the program for this test case.