Mini Project - 3

Subject: EE 275
(Advance Computer Architecture)

Name: Sagar M. Shah

SJSU ID: 011426818

Department: Electrical engineering

Professor: Dr. Chang Choo

University: San Jose State University

Mini-project III is to be carried out individually. In this mini-project, you are to design a single-level cache simulator (assume von Neuman architecture) as discussed in class. The "instruction cycle-accurate" simulator may be written in HDL (i.e., Verilog or VHDL) or SDL (i.e., C/C++ or Python).

The simulator should accept the following cache design parameters and simulate the working of the benchmark programs in the mini-project 2:

1. Main memory size (in bytes)
2. Cache size (in bytes)
3. Line size (in bytes)
4. Cache placement algorithm (cache organization, i.e., direct mapped, set-associative, etc.)
5. Set size (i.e., set associativity)
6. Cache replacement algorithm (i.e., LRU, FIFO, random)
7. Write policy (i.e., write-back, write-through, or write-once)

It is suggested the size of the main memory, and thus the corresponding cache memory, be set very small, just enough to hold the benchmark program.

Your cache simulator should show the cache memory (tag and data among others) contents at each instruction execution cycle.

NOTE: To show the authenticity of your work, attach the last two digits of your student ID to every signal and variable name used in your Verilog/ VHDL/C/C++ code. For example, if your SID is 123456789, signal names should be x89, a89, etc. Each of the screen captures of your code and simulation results should bear these signal names.

• Program to be run over Cache Simulator:

```
.include "nios_macros.s"
.global _start
_start:
        movia r2, AVECTOR           /* Register r2 is a pointer to vector A */
        movia r3, BVECTOR           /* Register r3 is a pointer to vector B */
        movia r4, N
        ldw r4, 0(r4)               /* Register r4 is used as the counter for loop iterations */
        add r5, r0, r0              /* Register r5 is used to accumulate the product */
LOOP:   ldw r6, 0(r2)               /* Load the next element of vector A */
        ldw r7, 0(r3)               /* Load the next element of vector B */
        mul r8, r6, r7              /* Compute the product of next pair of elements */
        add r5, r5, r8              /* Add to the sum */
        addi r2, r2, 1              /* Increment the pointer to vector A */
        addi r3, r3, 1              /* Increment the pointer to vector B */
        subi r4, r4, 1              /* Decrement the counter */
        bgt r4, r0, LOOP            /* Loop again if not ?nished */
        stw r5, DOT_PRODUCT(r0)     /* Store the result in memory */
STOP: br STOP

N:
.word 9                     /* Specify the number of elements */
AVECTOR:
.word 0, 1, 1, 4, 2, 6, 8, 1, 8     /* Specify the elements of vector A SJSU ID*/
BVECTOR:
.word 2, 3, 3, 6, 4, 8, 0, 3, 0 /* Specify the elements of vector B Plus 2 to each Digit of SJSU ID*/
```

• Expected answer:

(0x2) +(1x3) +(1x3) +(4x6) +(2x4) +(6x8) +(8x0) +(1x3) +(8x0) = 89

- **Some knowledge about Cache Memory**

<u>Cache Performance</u>

- Average memory access time is a useful measure to evaluate the performance of a memory hierarchy configuration.

$$\text{Avg mem access time} = \text{hit time} + \text{miss rate} \times \text{miss penalty}$$

- It tells us how much penalty the memory system imposes on each access (on average).
- It can easily be converted into clock cycles for a particular CPU.
- But leaving the penalty in nanoseconds allows two systems with different clock cycles times to be compared to a single memory system.

<u>Cache Performance</u>

- There may be different penalties for Instruction and Data accesses.
- In this case, you may have to compute them separately.
- This requires knowledge of the fraction of references that are instructions and the fraction that are data.
- The text gives 75% instruction references to 25% data references.
- We can also compute the write penalty separately from the read penalty.

- This may be necessary for two reasons:

- Miss rates are different for each situation.
- Miss penalties are different for each situation.
- Treating them as a single quantity yields a useful CPU time formula:

$$\text{CPU time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \frac{\text{Memory access}}{\text{Instruction}} \times \text{Miss rate} \times \text{Miss penalty}\right) \times \text{Clock Cycle Time}$$

Effects of Cache Performance on CPU Performance

- Low CPI machines suffer more relative to some fixed CPI memory penalty.
- A machine with a CPI of 5 suffers little from a 1 CPI penalty.
- However, a processor with a CPI of 0.5 has its execution time tripled !

Cache miss penalties are measured in cycles, not nanoseconds.

- This means that a faster machine will stall more cycles on the same memory system.
- Amdahl's Law raises its ugly head again: Fast machines with low CPI are affected significantly from memory access penalties.

Improving Cache Performance

- The increasing speed gap between CPU and main memory has made the performance of the memory system increasingly important.
- 15 distinct organizations characterize the effort of system architects in reducing average memory access time.
- These organizations can be distinguished by:
  1. Reducing the miss rate.
  2. Reducing the miss penalty.
  3. Reducing the time to hit in a cache.

Reducing Cache Misses

- Components of miss rate: All of these factors may be reduced using various methods we'll talk about.
- Compulsory
  1. Cold start misses or first reference misses : The first access to a block can NOT be in the cache, so there must be a compulsory miss.
  2. These are suffered regardless of cache size.
- Capacity
  1. If the cache is too small to hold all of the blocks needed during execution of a program, misses occur on blocks that were discarded earlier.
  2. In other words, this is the difference between the compulsory miss rate and the miss rate of a finite size fully associative cache.

- <u>Conflict</u>

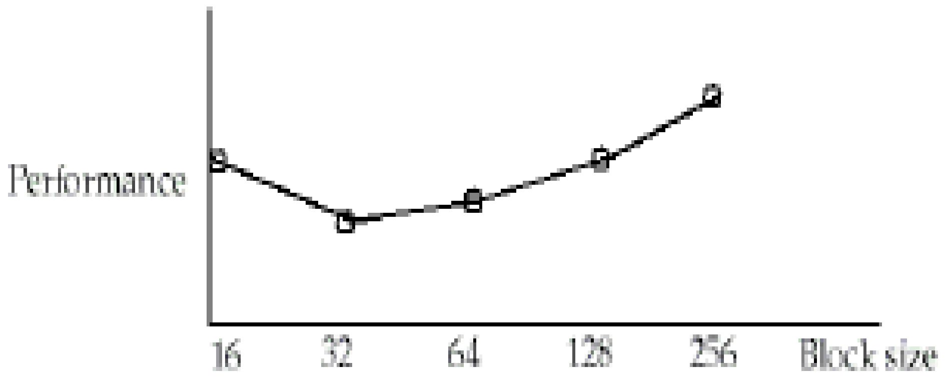
1. If the cache has sufficient space for the data, but the block can NOT be kept because the set is full, a conflict miss will occur.
2. This is the difference between the miss rate of a non fully associative cache and a fully associative cache.
3. These misses are also called collision or interference misses.

<u>**Reducing Cache Miss Rate**</u>

1. To reduce cache miss rate, we have to eliminate some of the misses due to the three C's.
2. We cannot reduce capacity misses much except by making the cache larger.
3. We can, however, reduce the conflict misses and compulsory misses in several ways:
4. Larger cache blocks
    - Larger blocks decrease the compulsory miss rate by taking advantage of spatial locality.
    - However, they may increase the miss penalty by requiring more data to be fetched per miss.
    - In addition, they will almost certainly increase conflict misses since fewer blocks can be stored in the cache.
    - And maybe even capacity misses in small caches.

Reducing Cache Miss Rate

- Larger cache blocks



- The performance curve is Ushaped because:
  1. Small blocks have a higher miss rate and
  2. Large blocks have a higher miss penalty (even if miss rate is not too high).
- High latency , high bandwidth memory systems encourage large block sizes since the cache gets more bytes per miss for a small increase in miss penalty.
- 32byte blocks are typical for 1KB, 4KB and 16KB caches while 64byte blocks are typical for larger caches.

Reducing Cache Miss Rate

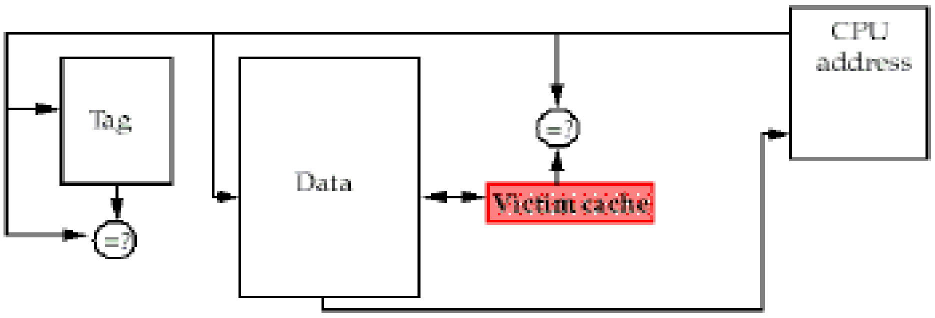
- Higher associativity
  1. Conflict misses can be a problem for caches with low associativity (especially directmapped).
  2. 2:1 cache rule of thumb : a directmapped cache of size N has the same miss rate as a 2way setassociative cache of size N/2.
  3. However, there is a limit higher associativity means more hardware and usually longer cycle times (increased hit time).

4. In addition, it may cause more capacity misses.
5. Nobody uses more than 8way set associative caches today, and most systems use 4way or less.
6. The problem is that the higher hit rate is offset by the slower clock cycle time.

Reducing Cache Miss Rate

- Victim caches

1. A victim cache is a small (usually, but not necessarily) fullyassociative cache that holds a few of the most recently replaced blocks or victims from the main cache.



2. Can improve miss rates without affecting the processor clock rate.
3. This cache is checked on a miss before going to main memory.
   - If found, the victim block and the cache block are swapped.

Reducing Cache Miss Rate

- Victim caches
  1. It can reduce capacity misses but is best at reducing conflict misses.
  2. It's particularly effective for small, direct mapped data caches.
  3. A 4 entry victim cache handled from 20% to 95% of the conflict misses from a 4KB directmapped data cache.
- Pseudo associative caches
  1. These caches use a technique similar to double hashing.
  2. On a miss, the cache searches a different set for the desired block.
     - The second (pseudo) set to probe is usually found by inverting one or more bits in the original set index .

  3.Note that two separate searches are conducted on miss.
     - The first search proceeds as it would for direct mapped cache.
     - Since there is no associative hardware, hit time is fast if it is found the first time.

Reducing Cache Miss Rate

- Pseudo associative caches
    1. While this second probe takes some time (usually an extra cycle or two), it is a lot faster than going to main memory.
- The secondary block can be swapped with the primary block on a "slow hit".
    2. This method reduces the effect of conflict misses.
    3. Also improves miss rates without affecting the processor clock rate.

- Hardware prefetch
    1. Prefetching is the act of getting data from memory before it is actually needed by the CPU.
    2. Typically, the cache requests the next consecutive block to be fetched with a requested block.
- It is hoped that this avoids a subsequent miss.

Reducing Cache Miss Rate

- Hardware prefetch
  1. This reduces compulsory misses by retrieving the data before it is requested.
  2. Of course, this may increase other misses by removing useful blocks from the cache.
     - Thus, many caches hold prefetched blocks in a special buffer until they are actually needed.
     - This buffer is faster than main memory but only has a limited capacity.
  3. Prefetching also uses main memory bandwidth.
     - It works well if the data is actually used.
     - However, it can adversely affect performance if the data is rarely used and the accesses interfere with `demand misses'

- C++ Implementation Of Cache Memory Simulator

```
#include<iostream>
#include<fstream>
#include<cstdlib>

#define N 9

using namespace std;

int lru[500][20];

//Function to change the tag order in the lru algo
int bringtotop(int set, int assoc, int x)
{
   int i, pos;
   for(i = 0;i < assoc;i++)
      if(lru[set][i] == x)
         pos = i;
   for(i = pos;i < assoc-1;i++)
      lru[set][i] = lru[set][i + 1];
   lru[set][assoc - 1] = x;
}

void plot(int Total_18, int Hit_18, int Miss_18);
long int changebase(char hex[], int Base_18);
int convert(char);

int main()
{
 int Main_Memory_18, Cache_Size_18, Cache_Placement_18, Associativity_18,
Block_Size_18, i, j, Number_Of_Blocks_18, Base_18, Write_Policy_18, r, alg, x, pos;
 long int Address_18;
 float Hit_Rate_18, Miss_Rate_18;
 char hex[20], Tracefile_18[20];
 int Number_Of_Set_18;
 int Check_18=0, Hit_18=0, Miss_18=0;
 int Sum_18=0;
 int A_Vector_18[9] = {0, 1, 1, 4, 2, 6, 8, 1, 8},
            B_Vector_18[9] = {2, 3, 3, 6, 4, 8, 0, 3, 0};

 cout<<"\nVALUE OF A VECTOR : {0, 1, 1, 4, 2, 6, 8, 1, 8}\n";

 cout<<"\nVALUE OF B VECTOR : {2, 3, 3, 6, 4, 8, 0, 3, 0}\n";

 cout<<"\nENTER THE MAIN MEMORY SIZE : \n ";
 cin>>Main_Memory_18;

 cout<<"\nENTER THE CACHE MEMORY SIZE : \n ";
 cin>>Cache_Size_18;

 cout<<"\nENTER THE BLOCK SIZE : \n ";
 cin>>Block_Size_18;

 cout<<"\nENTER THE CACHE PLACEMENT ALGORITHM : 1. FOR DIRECT
MAPPED   2. FOR SET-ASSOCIATIVE \n ";
 cin>>Cache_Placement_18;

 cout<<"\nENTER THE SET ASSOCIATIVITY : \n ";
 cin>>Associativity_18;

 cout<<"\nENTER TRACE FILE NAME : \n ";
 cin>>Tracefile_18;

 cout<<"\nENTER THE BASE OF NUMBER IN TRACE FILE : \n ";
 cin>>Base_18;

 cout<<"\nENTER THE CACHE REPLACEMENT ALGORITHM : \n1. FIFO  2.LRU
3. RANDOM \n ";
 cin>>alg;

 cout<<"\nENTER THE WRITE POLICY : \n1. WRITE THROUGH  2.WRITE
BACK  3. WRITE ONCE \n ";
 cin>>Write_Policy_18;

 Number_Of_Blocks_18 = Cache_Size_18 / Block_Size_18;
 Number_Of_Set_18 = Cache_Size_18 / (Associativity_18 * Block_Size_18);

 for(int i = 0; i <= N; i++)
           Sum_18 += A_Vector_18[i] * B_Vector_18[i];

 int cache[Number_Of_Set_18][Associativity_18];

 for(i = 0;i < Number_Of_Set_18;i++)
  for(j =0;j < Associativity_18;j++)
   cache[i][j] = -10; // Eliminating all garbage values in in the cache...

 int fifo[Number_Of_Set_18];
 for(i = 0;i < Number_Of_Set_18;i++)
  fifo[i] = 0;

   for(i = 0;i < Number_Of_Set_18;i++)
       for(j =0;j < Associativity_18;j++)
           lru[i][j] = j;

 ifstream infile;
 infile.open(Tracefile_18,ios::in);
 if(!infile)
 {
    cout<<"Error! File not found...";
    exit(0);
}

int set, tag, found;
while(!infile.eof()) //Reading each address from trace file
{

       if(Base_18 != 10)
       {
           infile >> hex;
           Address_18 = changebase(hex, Base_18);
       }
       else
           infile >> Address_18;

 set = (Address_18 / Block_Size_18) % Number_Of_Set_18;
 tag = Address_18 / (Block_Size_18 * Number_Of_Set_18);


 Check_18++;
 found = 0;
 for(i = 0;i < Associativity_18;i++)
  if(cache[set][i] == tag)
   {
       found = 1;
       pos = i;
   }

 if(found)
 {
     Hit_18++;
     if(alg == 2)
     {
            bringtotop(set, Associativity_18, pos);
     }
 }

 else
 {
          if(alg == 1)
          {
           i = fifo[set];

                         cache[set][i] = tag;
                         fifo[set]++;

                         if(fifo[set] == Associativity_18)
                  fifo[set] = 0;
          }
          else if(alg == 2)
          {
             i = lru[set][0];
             cache[set][i] = tag;
             bringtotop(set, Associativity_18, i);
          }
          else
          {
             r = rand() % Associativity_18;
             cache[set][r] = tag;

          }

  }

 }

 infile.close();
 system("clear");
 cout<<"\nNUMBER OF CHECKS : "<<Check_18;
 cout<<"\nNUMBER OF HITS : "<<Hit_18;
 cout<<"\nNUMBER OF MISSES : "<<Check_18 - Hit_18;
 Hit_Rate_18 = float(Hit_18) / float(Check_18);
 Miss_Rate_18 = float(Check_18 - Hit_18) / float(Hit_18);
 cout<<"\nHit Rate : "<<Hit_Rate_18;
 cout<<"\nMiss Rate : "<<Miss_Rate_18;
 cout<<"\nFINAL DOT PRODUCT OF TWO VECTOR IS : "<<Sum_18;
 plot(Check_18,Hit_18, Check_18-Hit_18);

 return 0;

}


int convert(char c)
{
   if(c == '1')
      return 1;

   else if(c == '2')
      return 2;

   else if(c == '3')
      return 3;

   else if(c == '4')
      return 4;

   else if(c =='5')
```

```
        return 5;

    else if(c == '6')
        return 6;

    else if(c == '7')
        return 7;

    else if(c == '8')
        return 8;

    else if(c == '9')
        return 9;

    else if(c == '0')
        return 0;

    else if( (c == 'a') || (c == 'A') )
        return 10;

    else if( (c == 'b') || (c == 'B') )
        return 11;

    else if( (c == 'c') || (c == 'C') )
        return 12;

    else if( (c == 'd') || (c == 'D') )
        return 13;

    else if( (c == 'e') || (c == 'E') )
        return 14;

    else if( (c == 'f') || (c == 'F') )
        return 15;

    else
        return 0;

}

//Function to change the Base of a number system to Decimal
long int changebase(char hex[], int Base_18)
{
    int pow = 1, len, i, j;
    char temp;
    long int dec;

    for(len = 0;hex[len] != '\0';len++);

    for(i = 0,j = (len - 1);i < j;i++,j--)
```

```
   {
      temp = hex[i];
      hex[i] = hex[j];
      hex[j] = temp;
   }

   pow = 1;
   dec = 0;
   for(i = 0;i < len;i++)
   {
      if(convert(hex[i] == -1))
      {
         dec = 0;
         break;
      }
      dec = dec + (pow* convert(hex[i]));
      pow*= Base_18;

   }
   return dec;

}


//Function to plot a graph...
void plot(int Total_18, int Hit_18, int Miss_18)
{

   cout<<"\n\n\n\n\t*********************HIT AND MISS RESULT THROUGH
GRAPH***********************\n\n";

   int Hit_Limit_18, Miss_Limit_18, i;
   Hit_Limit_18 = (float (Hit_18) / Total_18) * 30;
   Miss_Limit_18 = (float(Miss_18) / Total_18) * 30;

   cout<<"\n\t^";
   cout<<"\n\t|\n";
   for(i = 30;i >= 0;i--)
   {
      cout<<"\t";
      cout<<"|";
      cout<<"\t\t";

      //Total hit bar
      cout<<"|";
      if(i == 30)
         cout<<"----";
      else
         cout<<"    ";
      cout<<"|";
```

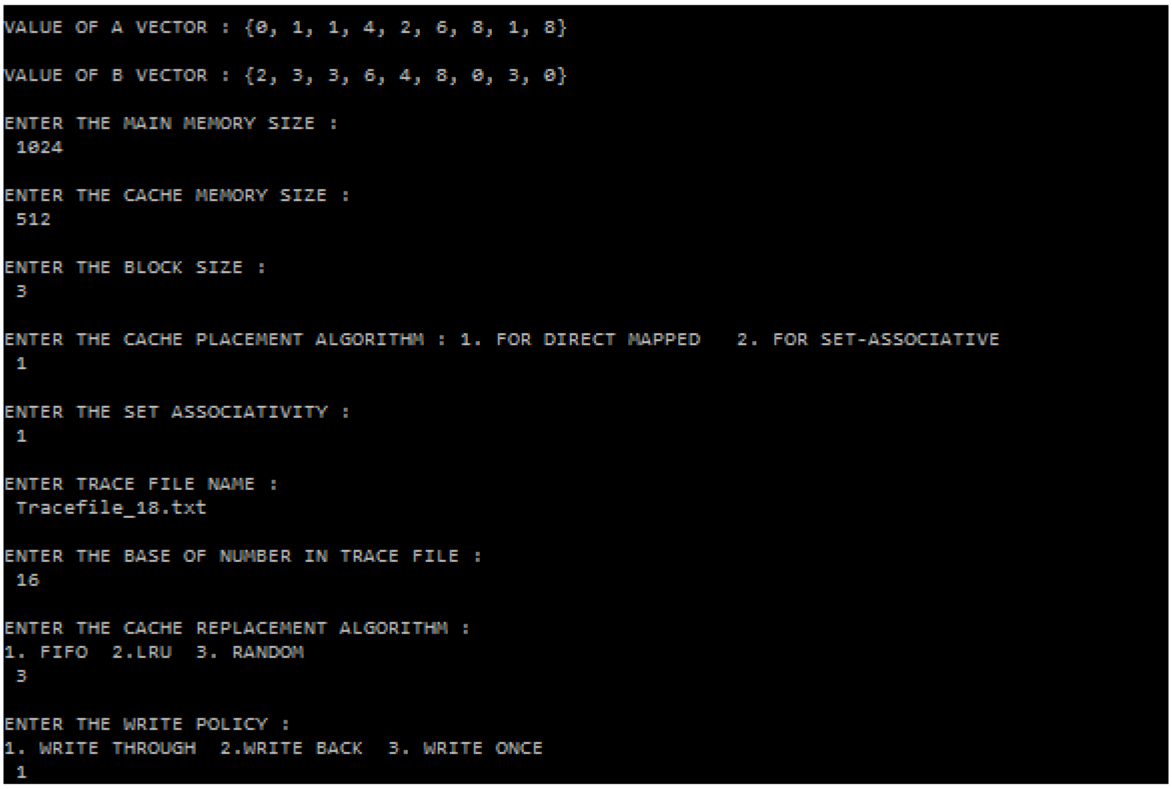
```
        cout<<"\t\t";

        //Hit Bar...
        if(i <= Hit_Limit_18)
           cout<<"|";
        else
           cout<<" ";

        if(i == Hit_Limit_18)
           cout<<"----";
        else
           cout<<"    ";

        if(i <= Hit_Limit_18)
           cout<<"|";
        else
           cout<<" ";
         cout<<"\t\t";

              //Miss Bar...
        if(i <= Miss_Limit_18)
           cout<<"|";
        else
           cout<<" ";

        if(i == Miss_Limit_18)
           cout<<"----";
        else
           cout<<"    ";

        if(i <= Miss_Limit_18)
           cout<<"|";
        else
           cout<<" ";

        cout<<"\n";

    }
    cout<<"\t------------------------------------------------------------------------------->";
    cout<<"\n\t\t\tTotal\t\t Hits\t\tMisses\n";
    cout<<"\t--------------------------SAGAR_SHAH_011426818-------------------------------
>\n\n\n";

}
```

• Output Procedure

1. After compilation when you run the program, black window will open.
2. It will show the value of vector A and vector B.
3. It will ask for Main Memory and Cache Memory Size as well as other Cache Memory related parameters which user must give.
4. It will ask for Trace file name which is "Tracefile_18.txt" in this program and user can change the name by changing it in main function.
5. User must give base of number which is stored in Trace file.
6. Output will show Number of Hit, Miss, Hit ration, Miss ration as well as dot product answer.
7. After output the graph will be provided to show difference between Total, Hit and Miss.

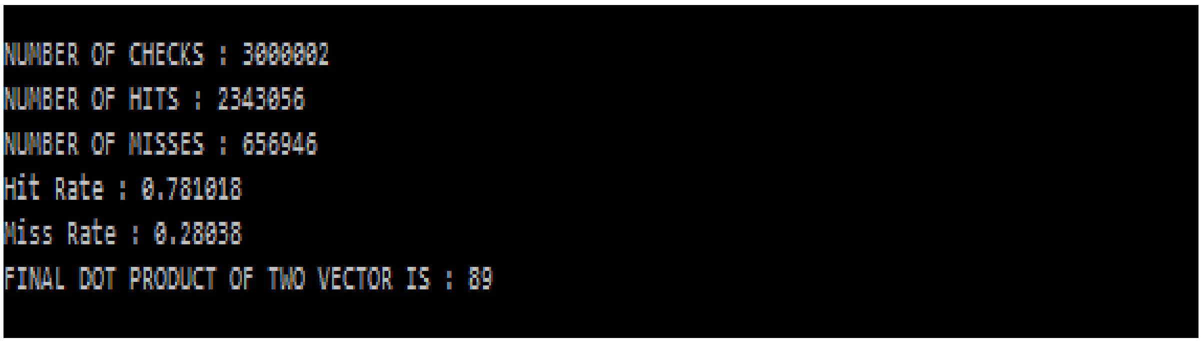
- Output Result's

- Input Data:



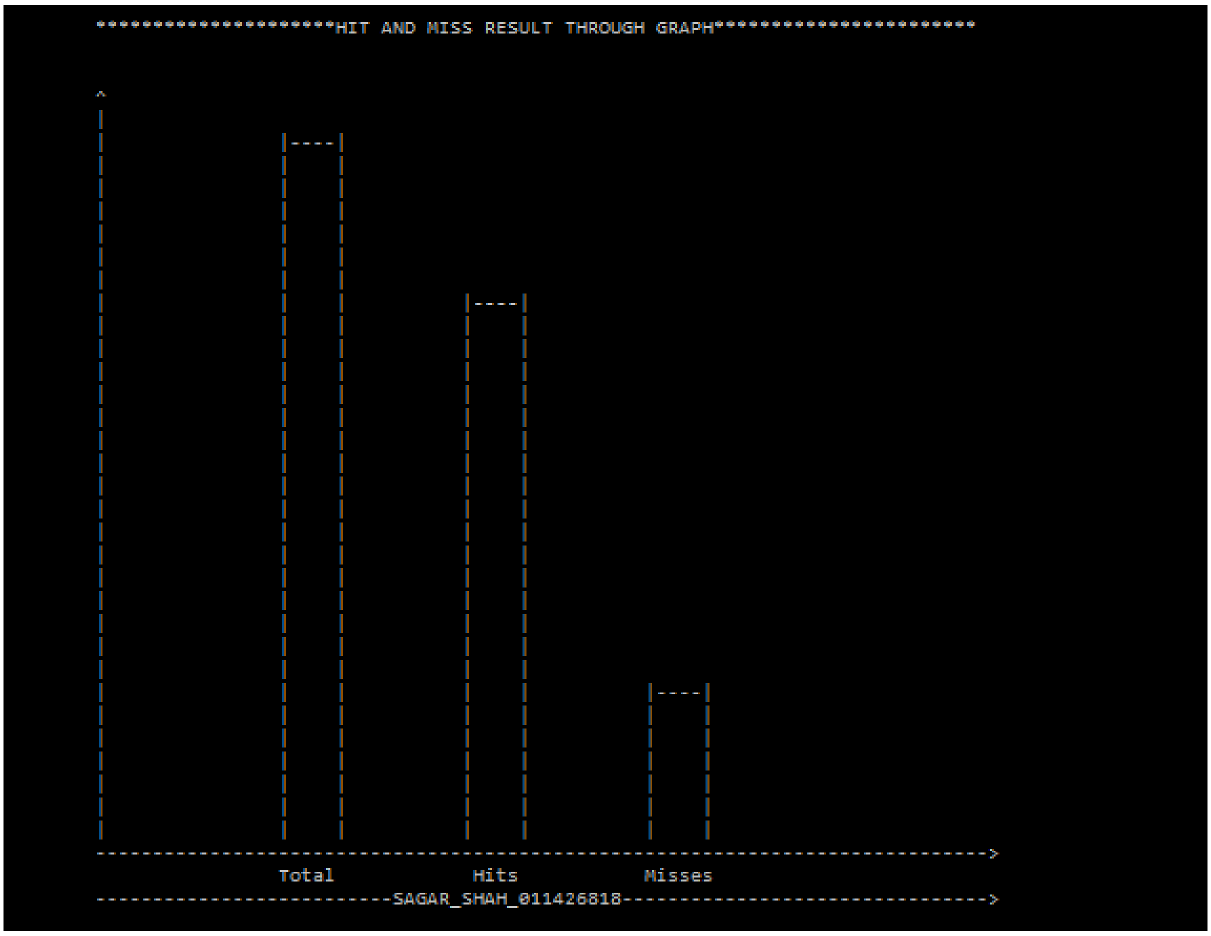
```
VALUE OF A VECTOR : {0, 1, 1, 4, 2, 6, 8, 1, 8}

VALUE OF B VECTOR : {2, 3, 3, 6, 4, 8, 0, 3, 0}

ENTER THE MAIN MEMORY SIZE :
 1024

ENTER THE CACHE MEMORY SIZE :
 512

ENTER THE BLOCK SIZE :
 3

ENTER THE CACHE PLACEMENT ALGORITHM : 1. FOR DIRECT MAPPED   2. FOR SET-ASSOCIATIVE
 1

ENTER THE SET ASSOCIATIVITY :
 1

ENTER TRACE FILE NAME :
 Tracefile_18.txt

ENTER THE BASE OF NUMBER IN TRACE FILE :
 16

ENTER THE CACHE REPLACEMENT ALGORITHM :
1. FIFO  2.LRU  3. RANDOM
 3

ENTER THE WRITE POLICY :
1. WRITE THROUGH  2.WRITE BACK  3. WRITE ONCE
 1
```

- Output Data:



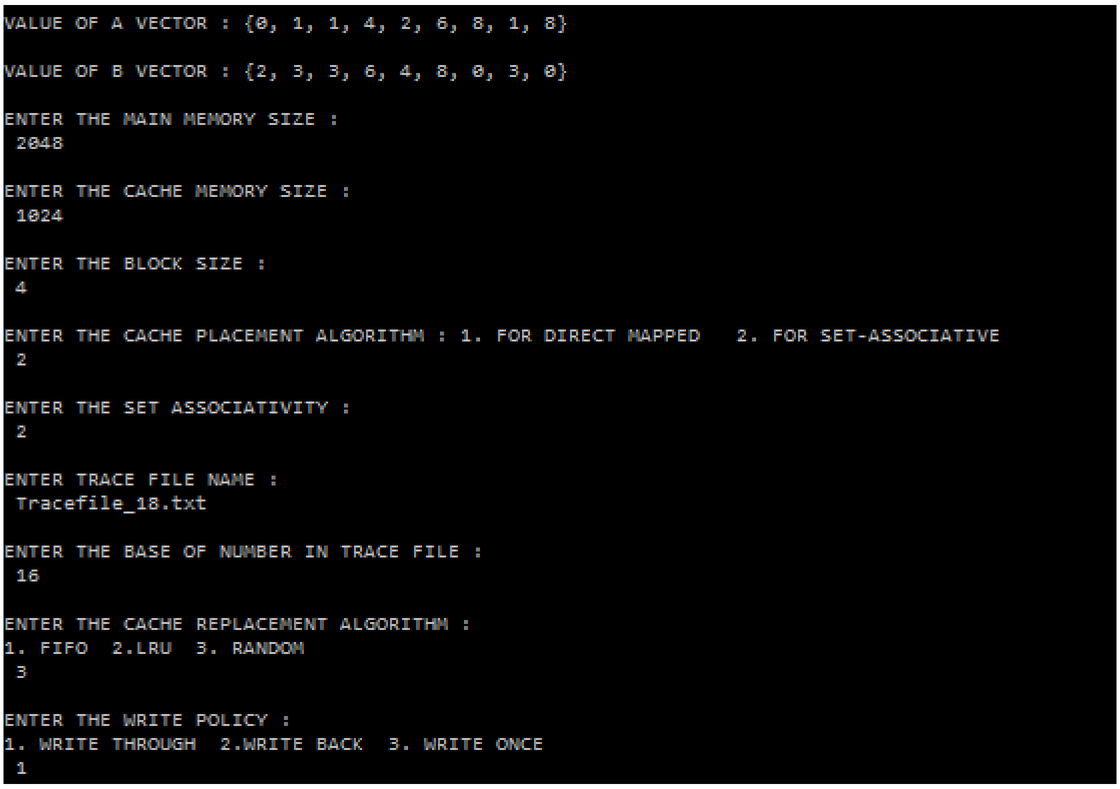
```
NUMBER OF CHECKS : 3000002
NUMBER OF HITS : 2343056
NUMBER OF MISSES : 656946
Hit Rate : 0.781018
Miss Rate : 0.28038
FINAL DOT PRODUCT OF TWO VECTOR IS : 89
```

- Graph:



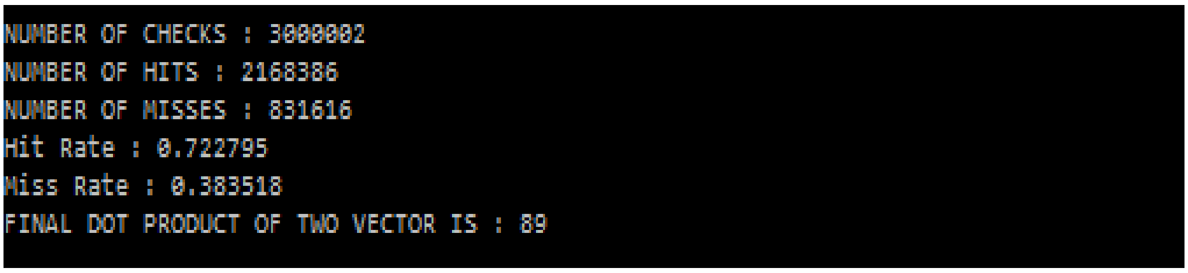
```
*********************HIT AND MISS RESULT THROUGH GRAPH***********************

                Total            Hits            Misses
-------------------------SAGAR_SHAH_011426818--------------------------------->
```

- Input Data:



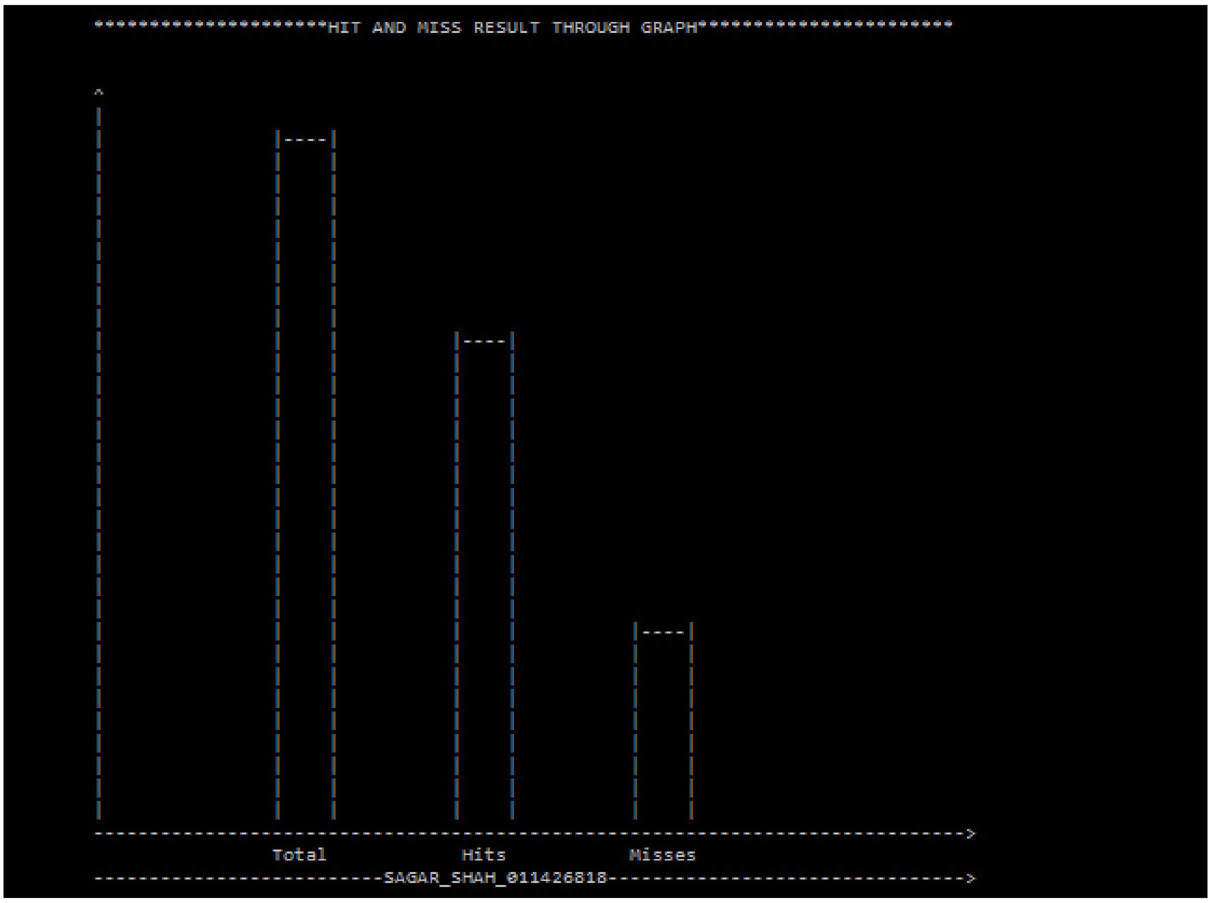
```
VALUE OF A VECTOR : {0, 1, 1, 4, 2, 6, 8, 1, 8}

VALUE OF B VECTOR : {2, 3, 3, 6, 4, 8, 0, 3, 0}

ENTER THE MAIN MEMORY SIZE :
 2048

ENTER THE CACHE MEMORY SIZE :
 1024

ENTER THE BLOCK SIZE :
 4

ENTER THE CACHE PLACEMENT ALGORITHM : 1. FOR DIRECT MAPPED   2. FOR SET-ASSOCIATIVE
 2

ENTER THE SET ASSOCIATIVITY :
 2

ENTER TRACE FILE NAME :
 Tracefile_18.txt

ENTER THE BASE OF NUMBER IN TRACE FILE :
 16

ENTER THE CACHE REPLACEMENT ALGORITHM :
1. FIFO  2.LRU  3. RANDOM
 3

ENTER THE WRITE POLICY :
1. WRITE THROUGH  2.WRITE BACK  3. WRITE ONCE
 1
```

- Output Data:



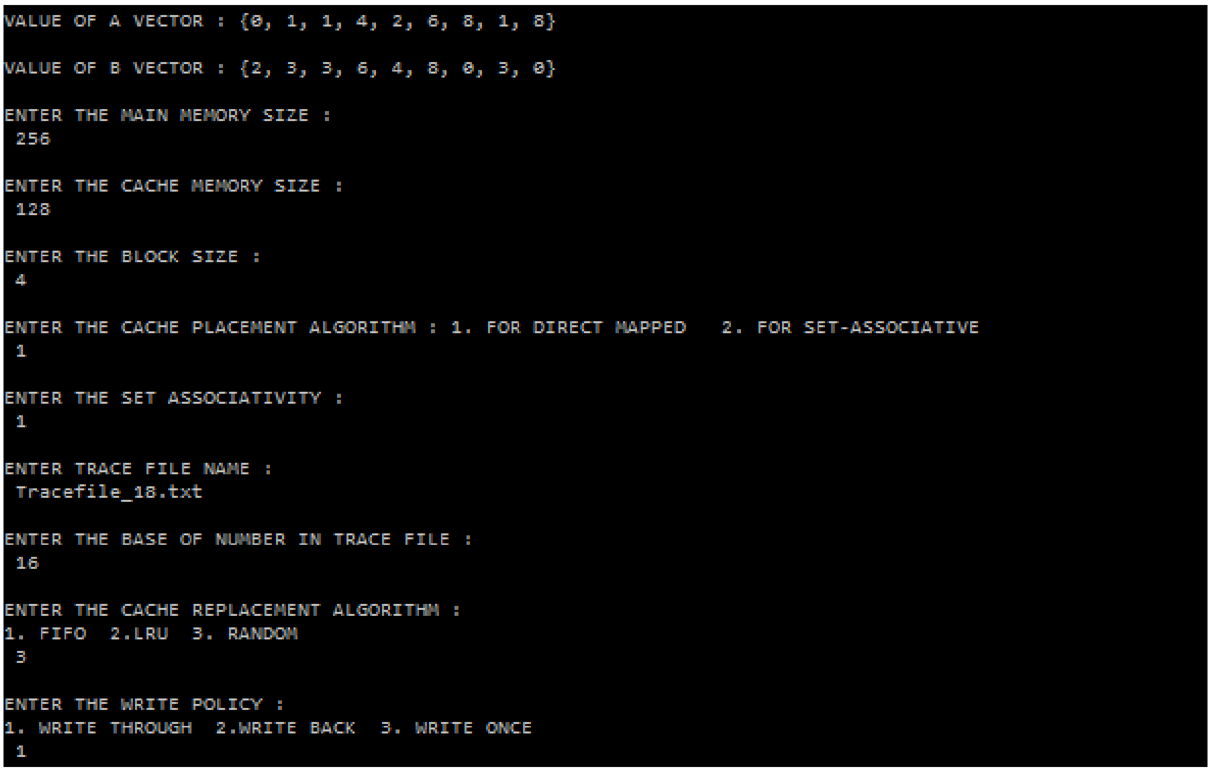
```
NUMBER OF CHECKS : 3000002
NUMBER OF HITS : 2168386
NUMBER OF MISSES : 831616
Hit Rate : 0.722795
Miss Rate : 0.383518
FINAL DOT PRODUCT OF TWO VECTOR IS : 89
```

- Graph:



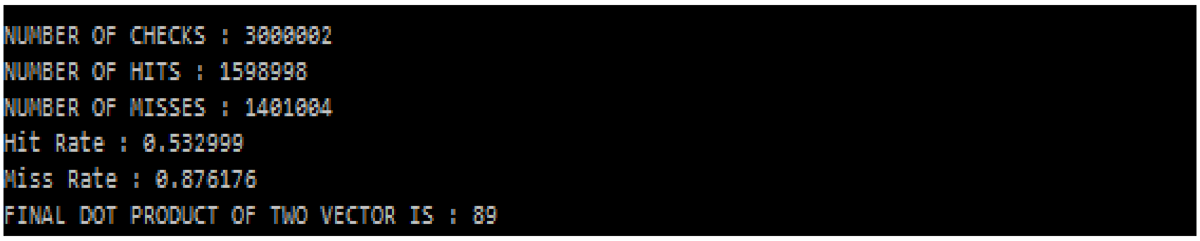
```
*********************HIT AND MISS RESULT THROUGH GRAPH***********************


^
|
|               |----|
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |          |----|
|               |    |          |    |
|               |    |          |    |
|               |    |          |    |
|               |    |          |    |
|               |    |          |    |
|               |    |          |    |
|               |    |          |    |
|               |    |          |    |
|               |    |          |    |
|               |    |          |    |
|               |    |          |    |
|               |    |          |    |
|               |    |          |    |          |----|
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
------------------------------------------------------------------------------->
                Total            Hits           Misses
--------------------------SAGAR_SHAH_011426818-------------------------------->
```

- Input Data:



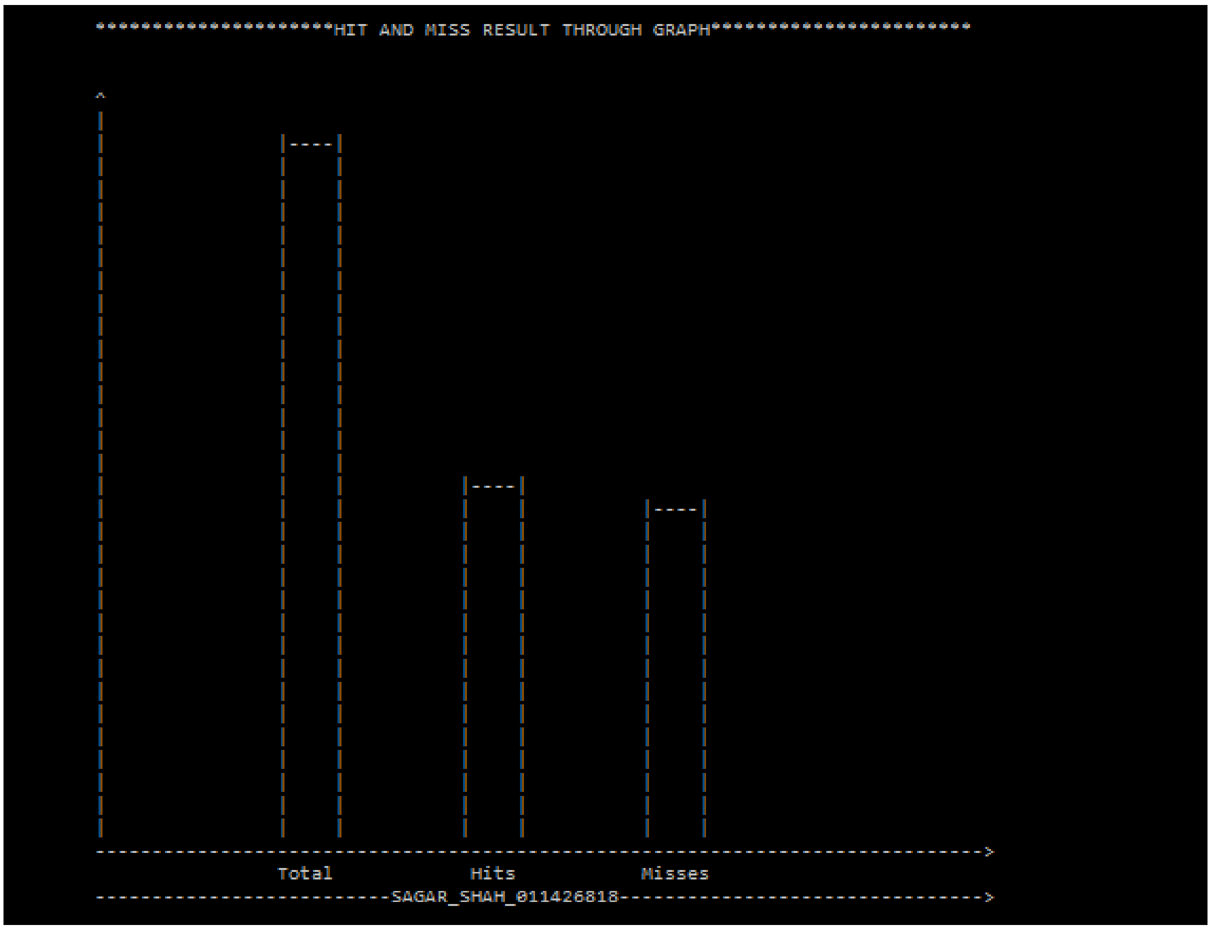
```
VALUE OF A VECTOR : {0, 1, 1, 4, 2, 6, 8, 1, 8}

VALUE OF B VECTOR : {2, 3, 3, 6, 4, 8, 0, 3, 0}

ENTER THE MAIN MEMORY SIZE :
 256

ENTER THE CACHE MEMORY SIZE :
 128

ENTER THE BLOCK SIZE :
 4

ENTER THE CACHE PLACEMENT ALGORITHM : 1. FOR DIRECT MAPPED   2. FOR SET-ASSOCIATIVE
 1

ENTER THE SET ASSOCIATIVITY :
 1

ENTER TRACE FILE NAME :
 Tracefile_18.txt

ENTER THE BASE OF NUMBER IN TRACE FILE :
 16

ENTER THE CACHE REPLACEMENT ALGORITHM :
1. FIFO  2.LRU  3. RANDOM
 3

ENTER THE WRITE POLICY :
1. WRITE THROUGH  2.WRITE BACK  3. WRITE ONCE
 1
```

- Output Data:



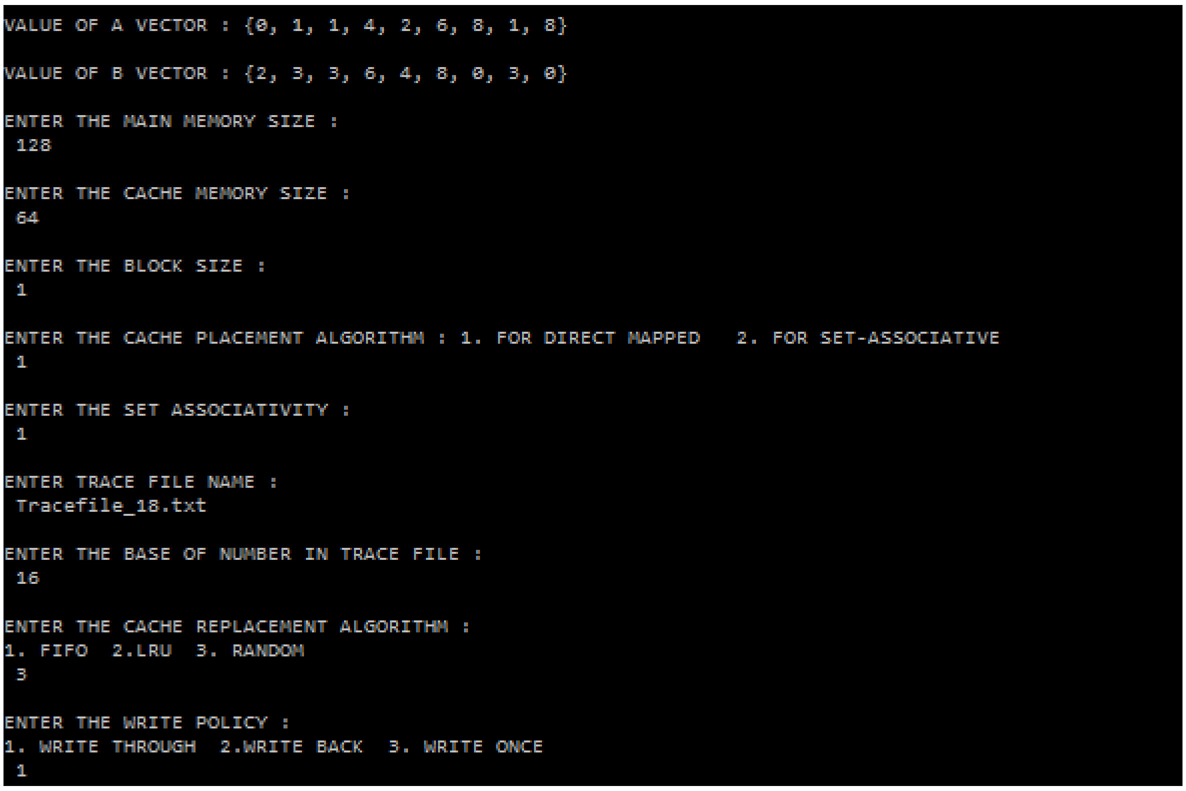
```
NUMBER OF CHECKS : 3000002
NUMBER OF HITS : 1598998
NUMBER OF MISSES : 1401004
Hit Rate : 0.532999
Miss Rate : 0.876176
FINAL DOT PRODUCT OF TWO VECTOR IS : 89
```

- Graph:



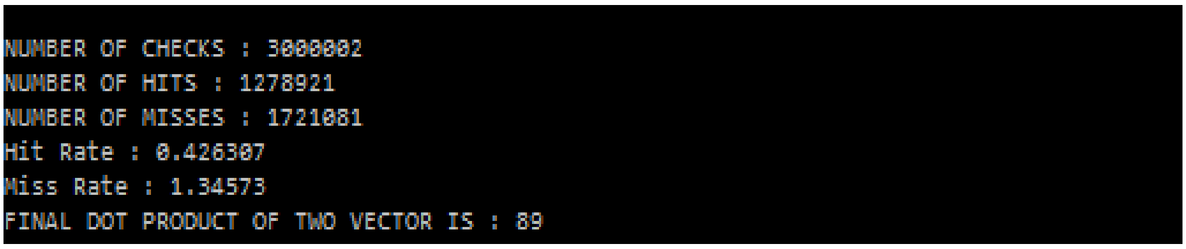
```
*********************HIT AND MISS RESULT THROUGH GRAPH***********************


^
|
|               |----|
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |          |----|
|               |    |          |    |          |----|
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
-------------------------------------------------------------------------------->
                Total            Hits           Misses
--------------------------SAGAR_SHAH_011426818--------------------------------->
```

- Input Data:



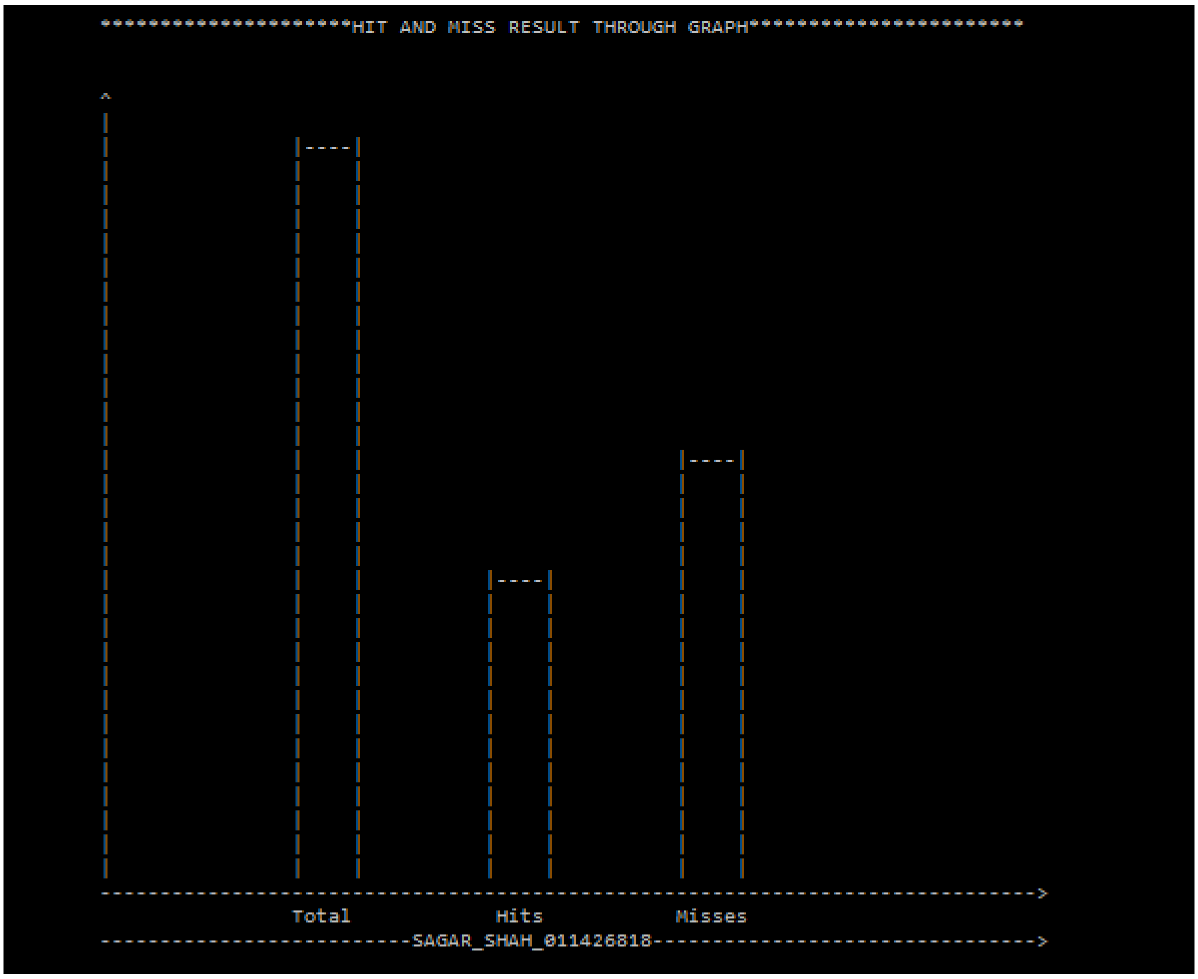
```
VALUE OF A VECTOR : {0, 1, 1, 4, 2, 6, 8, 1, 8}

VALUE OF B VECTOR : {2, 3, 3, 6, 4, 8, 0, 3, 0}

ENTER THE MAIN MEMORY SIZE :
 128

ENTER THE CACHE MEMORY SIZE :
 64

ENTER THE BLOCK SIZE :
 1

ENTER THE CACHE PLACEMENT ALGORITHM : 1. FOR DIRECT MAPPED   2. FOR SET-ASSOCIATIVE
 1

ENTER THE SET ASSOCIATIVITY :
 1

ENTER TRACE FILE NAME :
 Tracefile_18.txt

ENTER THE BASE OF NUMBER IN TRACE FILE :
 16

ENTER THE CACHE REPLACEMENT ALGORITHM :
1. FIFO  2.LRU  3. RANDOM
 3

ENTER THE WRITE POLICY :
1. WRITE THROUGH  2.WRITE BACK  3. WRITE ONCE
 1
```

- Output Data:



```
NUMBER OF CHECKS : 3000002
NUMBER OF HITS : 1278921
NUMBER OF MISSES : 1721081
Hit Rate : 0.426307
Miss Rate : 1.34573
FINAL DOT PRODUCT OF TWO VECTOR IS : 89
```

- Graph:



```
********************HIT AND MISS RESULT THROUGH GRAPH***********************


^
|
|               |----|
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |
|               |    |                          |----|
|               |    |                          |    |
|               |    |                          |    |
|               |    |                          |    |
|               |    |                          |    |
|               |    |          |----|          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
|               |    |          |    |          |    |
------------------------------------------------------------------------------->
                Total            Hits          Misses
--------------------------SAGAR_SHAH_011426818--------------------------------->
```

- **Conclusion:**
- I implemented a Cache Memory Simulator in C++ to simulate dot product of 2 vectors namely
  Avector:011426818
  Bvector:233648030
  And got the output “89”
- Calculation of Number of Hits, Misses, Hit and Miss ratio is displayed at the end of program.
- With the help of Graph the difference between Total, Hits and Misses is shown.
- If the size of Cache and Main Memory is large the Hit rate is greater and if small Miss rate is greater.