NAME: Meith Navlakha
SAP: 60004190068
POA Practical

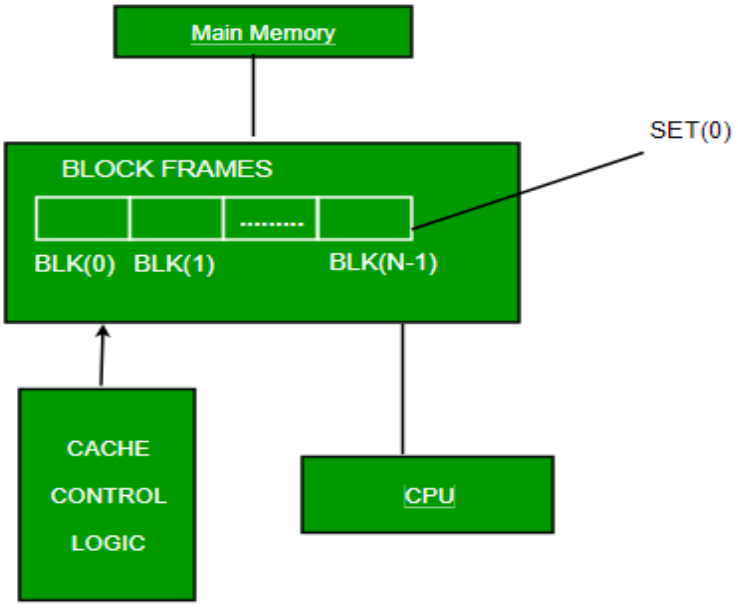
EXPERIMENT 6

AIM: To implement Fully Associative Mapping and Set Associative Mapping for 8bits and 16bits tag using UPC miniMIPS Simulator and C program.

THEORY:

Fully Associative Mapping:

Fully associative cache contains a single set with B ways, where B is the number of blocks. A memory address can map to a block in any of these ways. A fully associative cache also known as B-way set associative cache with one set. Here the associative memory is used to store the content and the addresses of the memory word. Any block can go into any line of the cache. This means that the word-id bits are used to identify which word in the block is needed, but the tag becomes all of the remaining bits. This enables the placement of any word at any place in the cache memory. It is the fastest and the most flexible mapping form.

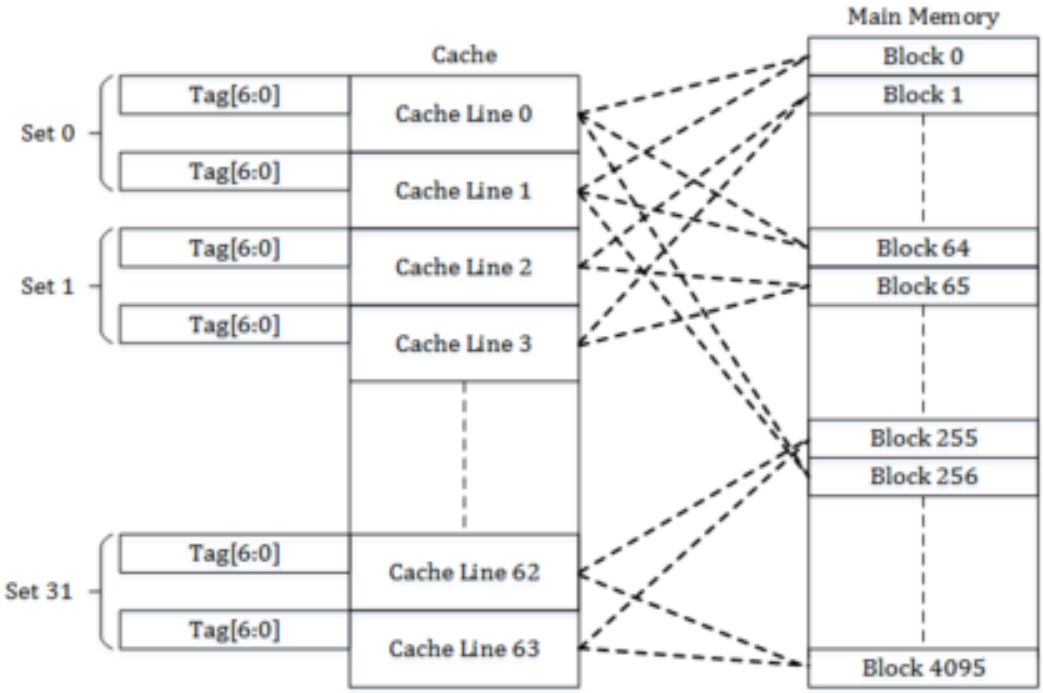


The program maintains an array with 8/16 tags and the most-recently-used ordering of the lines in the array mru[]. When each address is read from the trace file, it is compared to all of the tags in the cache in the first for loop. If the tag is found, a hit is recorded, and the mru[] array is updated using the mruUpdate() function, and the loop is exited via the break statement. A miss is detected when no matches are found after searching all 8/16 tags. In this case the loop index, i, will be set to 8/16. On a miss the least recently-used tag, which should be the last element in mru[], is chosen for replacement, the tag is updated, and the mru[] array is updated.

NAME: Meith Navlakha
SAP: 60004190068
POA Practical

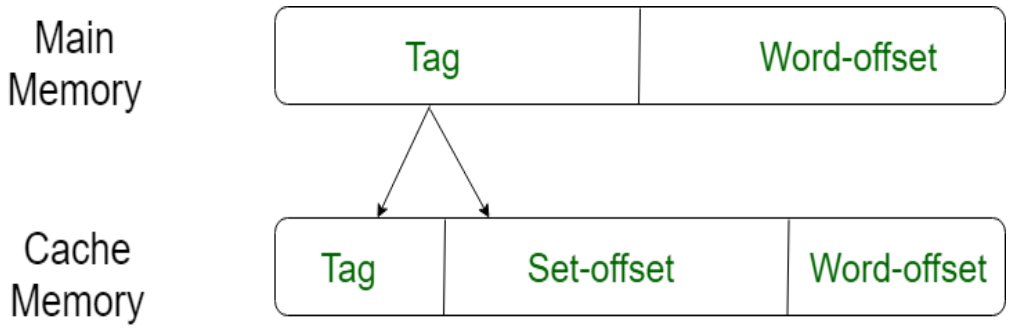
Set Associative Mapping:

In k-way set associative mapping, cache lines are grouped into sets where each set contains k number of lines. A particular block of main memory can map to only one particular set of the cache however, within that set, the memory block can map to any freely available cache line. Set-associative mapping allows that each word that is present in the cache can have two or more words in the main memory for the same index address. Set associative cache mapping combines the best of direct and associative cache mapping techniques.



The set of the cache to which a particular block of the main memory can map is given by:

Cache et number = (Main Memory Block Address) Modulo (Number of sets in Cache)



In the program, the 8 tags are stored as a 2 by 4 entry array, where the first array index selects between the lines in the 2-way set and the second index selects one of 4 lines. The second array, mru[], tracks the most recently used of the two lines in each set. When each address is read from the trace file, it is compared to all of the tags in the cache in the first for loop. If the tag is found, a hit is recorded, and the mru[] array is updated using the mruUpdate() function, and the loop is exited via the break statement else it's a miss and it is loaded into the memory.

CODE:

1) Fully-Associative Cache (Tag 8bit):

```
include <stdio.h>
int tag[8];
int mru[8] = {7,6,5,4,3,2,1,0};

void mruUpdate(int index)
{
   int i;
   // find index in mru
   for (i = 0; i < 8; i++)
      if (mru[i] == index)
         break;
   // move earlier refs one later
   while (i > 0) {
      mru[i] = mru[i-1];
      i--;
   }
   mru[0] = index;
}

int main( )
{
   int addr;
   int i, j, t;
   int hits, accesses;
   FILE *fp;

   fp = fopen("trace.txt", "r");
   hits = 0;
   accesses = 0;
```

```
    while (fscanf(fp, "%x", &addr) > 0) {
        accesses += 1;
        printf("%3d: 0x%08x ", accesses, addr);
        for (i = 0; i < 8; i++) {
            if (tag[i] == addr) {
                hits += 1;
                printf("Hit%d ", i);
                mruUpdate(i);
                break;
            }
        }
        if (i == 8) {
            /* allocate entry */
            printf("Miss ");
            i = mru[7];
            tag[i] = addr;
            mruUpdate(i);
        }
        for (i = 0; i < 8; i++)
            printf("0x%08x ", tag[i]);
        for (i = 0; i < 8; i++)
            printf("%d ", mru[i]);
        printf("\n");
    }
    printf("Hits = %d, Accesses = %d, Hit ratio = %f\n", hits, accesses,
((float)hits)/accesses);
    close(fp);
}
```

2) Fully-Associative Cache (Tag 16bit):

```
#include <stdio.h>
int tag[16];
int mru[16] = {15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0};

void mruUpdate(int index){
    int i;
    for (i = 0; i < 16; i++)
        if (mru[i] == index)
            break;
```

```
  while (i > 0) {
      mru[i] = mru[i-1];
      i--;
    }
    mru[0] = index;
}

int main()
{
    int addr;
    int i, j, t;
    int hits, accesses;
    FILE *fp;

    fp = fopen("trace.txt", "r");
    hits = 0;
    accesses = 0;
    while (fscanf(fp, "%x", &addr) > 0) {
       /* simulate fully associative cache with 8 words */
       accesses += 1;
       printf("%4d: 0x%08x ", accesses, addr);
       for (i = 0; i < 16; i++) {
          if (tag[i] == addr) {
             hits += 1;
             printf("Hit%d ", i);
             mruUpdate(i);
             break;
          }
       }
       if (i == 16) {
          /* allocate entry */
          printf("Miss ");
          i = mru[15];
          tag[i] = addr;
          mruUpdate(i);
       }
       for (i = 0; i < 16; i++)
          printf("0x%08x ", tag[i]);
       for (i = 0; i < 16; i++)
          printf("%d ", mru[i]);
```

```
        printf("\n\n");
    }
    printf("Hits = %d, Accesses = %d, Hit ratio = %f\n", hits, accesses,
((float)hits)/accesses);
    close(fp);
}
```

3) 2-way set-associative cache (Tag 8bit)

```
#include <stdio.h>
int tag[2][4];
int mru[4] = {1,1,1,1};

void mruUpdate(int index){
int i;
for (i = 0; i < 4; i++)
    if (mru[i] == index)
    break;
    while (i > 0) {
        mru[i] = mru[i-1];
        i--;
    }
    mru[0] = index;
}

int main(){
    int addr;
    int i, j;
    int hits, accesses;
    FILE *fp;

    fp = fopen("trace.txt", "r");
    hits = 0;
    accesses = 0;
    while (fscanf(fp, "%x", &addr) > 0) {
        printf("%3d: 0x%08x ", accesses, addr);
        for( i=0; i<2; i++){
            accesses += 1;
            for(j=0; j<4; j++){
                if (tag[i][j] ==addr){
```

```
                hits += 1;
                printf("Hit%d ", i);
                mruUpdate(i);
                break;

            }

        } if(tag[i][j]==addr){
            printf("Hit%d ", i);

        }else {
            printf("Miss");
            tag[i][j] = addr;
            //hits +=1;
            }

    } for(i = 0; i < 2; i++){
        for(j=0; j<4; j++){
            printf("0x%08x ", tag[i][j]);

        }

    }
    for (i = 0; i < 2; i++)
    printf("%d ", mru[i]);
    printf("\n");
  }
  printf("Hits = %d, Accesses = %d, Hit ratio = %f\n", hits, accesses,
((float)hits)/accesses);
  close(fp);
}
```

4) 2-way set-associative cache (Tag 16bit)

```
#include <stdio.h>
int tag[2][8]; // ERROR DECIDE SIZE
int mru[8] = {1,1,1,1,1,1,1,1};
void mruUpdate(int index){
int i;
```

```
for (i = 0; i < 8; i++)
   if (mru[i] == index)
   break;
   while (i > 0) {
      mru[i] = mru[i-1];
      i--;

   }

   mru[0] = index;

}

int main(){
   int addr;
   int i, j;
   int hits, accesses;
   FILE *fp;

   fp = fopen("trace.txt", "r");
   hits = 0;
   accesses = 0;
   while (fscanf(fp, "%x", &addr) > 0) {
      printf("%4d: 0x%08x ", accesses, addr);
      for( i=0; i<2; i++){
         accesses += 1;
         for(j=0; j<8; j++){
            if (tag[i][j] ==addr){
               hits += 1;
               printf("Hit%d ", i);
               mruUpdate(i);
               break;
            }
         } if(tag[i][j]==addr){
            printf("Hit%d ", i);

         }else {
            printf("Miss");
            tag[i][j] = addr;
            //hits +=1;
```

```
            }
        } for(i = 0; i < 2; i++){
            for(j=0; j<8; j++){
                printf("0x%08x ", tag[i][j]);

            }

        }

        for (i = 0; i < 4; i++)
        printf("%d ", mru[i]);

        printf("\n");

    }
    printf("Hits = %d, Accesses = %d, Hit ratio = %f\n", hits, accesses,
((float)hits)/accesses);
    close(fp);
}
```

OUTPUT:

1) Fully-Associative Cache (Tag 8bit):

input

```
 88: 0x80000018 Hit7 0x00000054 0x00000058 0x0000005c 0x8000000c 0x00000060 0x80000010 0x80000014 0x80000018 7 6 5 4 3
2 1 0
 89: 0x8000000c Hit3 0x00000054 0x00000058 0x0000005c 0x8000000c 0x00000060 0x80000010 0x80000014 0x80000018 3 7 6 5 4
2 1 0
 90: 0x00000064 Miss 0x00000064 0x00000058 0x0000005c 0x8000000c 0x00000060 0x80000010 0x80000014 0x80000018 0 3 7 6 5
4 2 1
 91: 0x80000010 Hit5 0x00000064 0x00000058 0x0000005c 0x8000000c 0x00000060 0x80000010 0x80000014 0x80000018 5 0 3 7 6
4 2 1
 92: 0x80000014 Hit6 0x00000064 0x00000058 0x0000005c 0x8000000c 0x00000060 0x80000010 0x80000014 0x80000018 6 5 0 3 7
4 2 1
 93: 0x80000018 Hit7 0x00000064 0x00000058 0x0000005c 0x8000000c 0x00000060 0x80000010 0x80000014 0x80000018 7 6 5 0 3
4 2 1
 94: 0x8000000c Hit3 0x00000064 0x00000058 0x0000005c 0x8000000c 0x00000060 0x80000010 0x80000014 0x80000018 3 7 6 5 0
4 2 1
 95: 0x00000068 Miss 0x00000064 0x00000068 0x0000005c 0x8000000c 0x00000060 0x80000010 0x80000014 0x80000018 1 3 7 6 5
0 4 2
 96: 0x80000010 Hit5 0x00000064 0x00000068 0x0000005c 0x8000000c 0x00000060 0x80000010 0x80000014 0x80000018 5 1 3 7 6
0 4 2
 97: 0x80000014 Hit6 0x00000064 0x00000068 0x0000005c 0x8000000c 0x00000060 0x80000010 0x80000014 0x80000018 6 5 1 3 7
0 4 2
 98: 0x80000018 Hit7 0x00000064 0x00000068 0x0000005c 0x8000000c 0x00000060 0x80000010 0x80000014 0x80000018 7 6 5 1 3
0 4 2
 99: 0x8000000c Hit3 0x00000064 0x00000068 0x0000005c 0x8000000c 0x00000060 0x80000010 0x80000014 0x80000018 3 7 6 5 1
0 4 2
100: 0x0000006c Miss 0x00000064 0x00000068 0x0000006c 0x8000000c 0x00000060 0x80000010 0x80000014 0x80000018 2 3 7 6 5
1 0 4
101: 0x80000010 Hit5 0x00000064 0x00000068 0x0000006c 0x8000000c 0x00000060 0x80000010 0x80000014 0x80000018 5 2 3 7 6
1 0 4
102: 0x80000014 Hit6 0x00000064 0x00000068 0x0000006c 0x8000000c 0x00000060 0x80000010 0x80000014 0x80000018 6 5 2 3 7
1 0 4
103: 0x80000018 Hit7 0x00000064 0x00000068 0x0000006c 0x8000000c 0x00000060 0x80000010 0x80000014 0x80000018 7 6 5 2 3
1 0 4
Hits = 76, Accesses = 103, Hit ratio = 0.737864
```

NAME: Meith Navlakha
SAP: 60004190068
POA Practical

2) Fully-Associative Cache (Tag 16bit):

```
  94: 0x8000000c Hit3 0x00000044 0x00000048 0x0000004c 0x8000000c 0x00000050 0x80000010 0x80000014 0x80000018 0x0000005
4 0x00000058 0x0000005c 0x00000060 0x00000064 0x00000038 0x0000003c 0x00000040 3 7 6 5 12 11 10 9 8 4 2 1 0 15 14 13

  95: 0x00000068 Miss 0x00000044 0x00000048 0x0000004c 0x8000000c 0x00000050 0x80000010 0x80000014 0x80000018 0x0000005
4 0x00000058 0x0000005c 0x00000060 0x00000064 0x00000068 0x0000003c 0x00000040 13 3 7 6 5 12 11 10 9 8 4 2 1 0 15 14

  96: 0x80000010 Hit5 0x00000044 0x00000048 0x0000004c 0x8000000c 0x00000050 0x80000010 0x80000014 0x80000018 0x0000005
4 0x00000058 0x0000005c 0x00000060 0x00000064 0x00000068 0x0000003c 0x00000040 5 13 3 7 6 12 11 10 9 8 4 2 1 0 15 14

  97: 0x80000014 Hit6 0x00000044 0x00000048 0x0000004c 0x8000000c 0x00000050 0x80000010 0x80000014 0x80000018 0x0000005
4 0x00000058 0x0000005c 0x00000060 0x00000064 0x00000068 0x0000003c 0x00000040 6 5 13 3 7 12 11 10 9 8 4 2 1 0 15 14

  98: 0x80000018 Hit7 0x00000044 0x00000048 0x0000004c 0x8000000c 0x00000050 0x80000010 0x80000014 0x80000018 0x0000005
4 0x00000058 0x0000005c 0x00000060 0x00000064 0x00000068 0x0000003c 0x00000040 7 6 5 13 3 12 11 10 9 8 4 2 1 0 15 14

  99: 0x8000000c Hit3 0x00000044 0x00000048 0x0000004c 0x8000000c 0x00000050 0x80000010 0x80000014 0x80000018 0x0000005
4 0x00000058 0x0000005c 0x00000060 0x00000064 0x00000068 0x0000003c 0x00000040 3 7 6 5 13 12 11 10 9 8 4 2 1 0 15 14

 100: 0x0000006c Miss 0x00000044 0x00000048 0x0000004c 0x8000000c 0x00000050 0x80000010 0x80000014 0x80000018 0x0000005
4 0x00000058 0x0000005c 0x00000060 0x00000064 0x00000068 0x0000006c 0x00000040 14 3 7 6 5 13 12 11 10 9 8 4 2 1 0 15

 101: 0x80000010 Hit5 0x00000044 0x00000048 0x0000004c 0x8000000c 0x00000050 0x80000010 0x80000014 0x80000018 0x0000005
4 0x00000058 0x0000005c 0x00000060 0x00000064 0x00000068 0x0000006c 0x00000040 5 14 3 7 6 13 12 11 10 9 8 4 2 1 0 15

 102: 0x80000014 Hit6 0x00000044 0x00000048 0x0000004c 0x8000000c 0x00000050 0x80000010 0x80000014 0x80000018 0x0000005
4 0x00000058 0x0000005c 0x00000060 0x00000064 0x00000068 0x0000006c 0x00000040 6 5 14 3 7 13 12 11 10 9 8 4 2 1 0 15

 103: 0x80000018 Hit7 0x00000044 0x00000048 0x0000004c 0x8000000c 0x00000050 0x80000010 0x80000014 0x80000018 0x0000005
4 0x00000058 0x0000005c 0x00000060 0x00000064 0x00000068 0x0000006c 0x00000040 7 6 5 14 3 13 12 11 10 9 8 4 2 1 0 15

Hits = 76, Accesses = 103, Hit ratio = 0.737864
```

3) 2-way set-associative cache (Tag 8bit)

```
172: 0x80000014 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x80000014 0x00000000 0x00000000 0x00000000 1
 1
174: 0x80000018 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x80000018 0x00000000 0x00000000 0x00000000 1
 1
176: 0x8000000c MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x8000000c 0x00000000 0x00000000 0x00000000 1
 1
178: 0x00000064 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x00000064 0x00000000 0x00000000 0x00000000 1
 1
180: 0x80000010 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x80000010 0x00000000 0x00000000 0x00000000 1
 1
182: 0x80000014 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x80000014 0x00000000 0x00000000 0x00000000 1
 1
184: 0x80000018 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x80000018 0x00000000 0x00000000 0x00000000 1
 1
186: 0x8000000c MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x8000000c 0x00000000 0x00000000 0x00000000 1
 1
188: 0x00000068 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x00000068 0x00000000 0x00000000 0x00000000 1
 1
190: 0x80000010 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x80000010 0x00000000 0x00000000 0x00000000 1
 1
192: 0x80000014 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x80000014 0x00000000 0x00000000 0x00000000 1
 1
194: 0x80000018 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x80000018 0x00000000 0x00000000 0x00000000 1
 1
196: 0x8000000c MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x8000000c 0x00000000 0x00000000 0x00000000 1
 1
198: 0x0000006c MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x0000006c 0x00000000 0x00000000 0x00000000 1
 1
200: 0x80000010 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x80000010 0x00000000 0x00000000 0x00000000 1
 1
202: 0x80000014 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x80000014 0x00000000 0x00000000 0x00000000 1
 1
204: 0x80000018 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x80000018 0x00000000 0x00000000 0x00000000 1
 1
Hits = 103, Accesses = 206, Hit ratio = 0.500000
```

NAME: Meith Navlakha
SAP: 60004190068
POA Practical

4) 2-way set-associative cache (Tag 16bit)

input

```
0x8000000c 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 1 1 1 1
 178: 0x00000064 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x00000064 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 1 1 1 1
 180: 0x80000010 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x80000010 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 1 1 1 1
 182: 0x80000014 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x80000014 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 1 1 1 1
 184: 0x80000018 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x80000018 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 1 1 1 1
 186: 0x8000000c MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x8000000c 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 1 1 1 1
 188: 0x00000068 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x00000068 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 1 1 1 1
 190: 0x80000010 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x80000010 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 1 1 1 1
 192: 0x80000014 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x80000014 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 1 1 1 1
 194: 0x80000018 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x80000018 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 1 1 1 1
 196: 0x8000000c MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x8000000c 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 1 1 1 1
 198: 0x0000006c MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0000006c 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 1 1 1 1
 200: 0x80000010 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x80000010 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 1 1 1 1
 202: 0x80000014 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x80000014 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 1 1 1 1
 204: 0x80000018 MissHit1 Hit1 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x80000018 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 1 1 1 1
Hits = 103, Accesses = 206, Hit ratio = 0.500000
```

CONCLUSION: In this experiment, I implemented Fully Associative Mapping and Set Associative Mapping for 8bits and 16bits tag using UPC miniMIPS Simulator and C program. First I generated a trace.txt file using UNC miniMIPS Simulator. Then coded Associative Mapping and Set Associative Mapping for 8bits and 16bits tag in C language. For Fully Associative Mapping with tag size 8bits, the Hit Ratio was 0.737 and with tag size 16bits, the hit ratio was 0.74. Similarly, for 2 way Set Associative, the hit ratio was 0.5. Finally, associative mapping is fast and easy to implement, however it is expensive to implement as it requires storing of addresses along with the data. The placement policy is a trade-off between direct-mapped and fully associative cache however Set Associative offers the flexibility of using replacement algorithms if a cache miss occurs but the placement policy will not effectively use all the available cache lines in the cache and suffers from conflict miss.