NAME: ALISTAIR SALDANHA
SAPID: 60009200024
BATCH: K1

Experiment 910

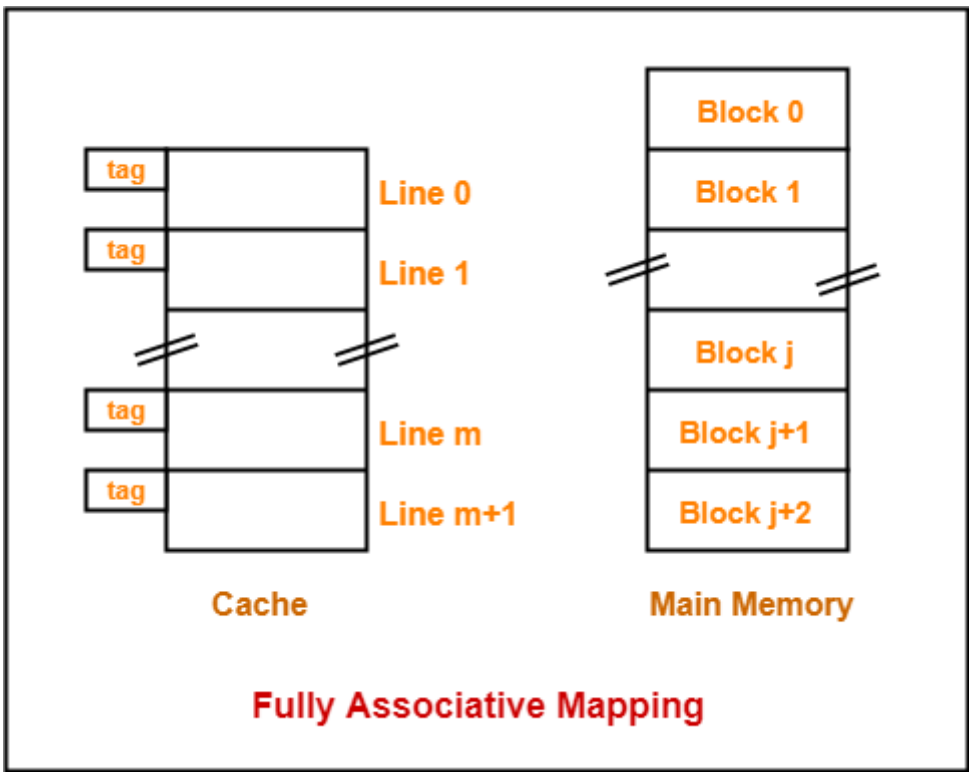
(Fully associative and set associative cache memory mapping)

Aim: Implement Fully associative and set associative cache memory mapping.

Theory:

Fully Associative Mapping-

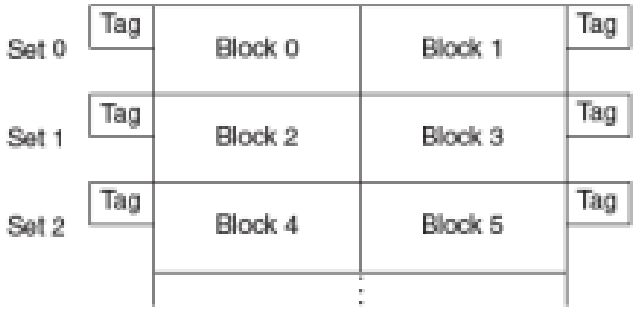
- A block of main memory can map to any line of the cache that is freely available at that moment.
- This makes fully associative mapping more flexible than direct mapping.
- All the lines of cache are freely available.
- Thus, any block of main memory can map to any line of the cache.
- Had all the cache lines been occupied, then one of the existing blocks will have to be replaced.



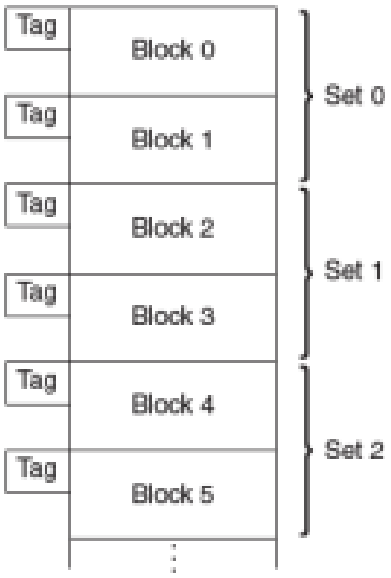
In fully associative mapping, the physical address is divided as-

Division of Physical Address in Fully Associative Mapping

Set associative cache combines the ideas of direct mapped cache and fully associative cache. An N-way set associative cache mapping is like direct mapped cache in that a memory reference maps to a particular location in cache. Unlike direct mapped cache, a memory reference maps to a set of several cache blocks, similar to the way in which fully associative cache works. Instead of mapping anywhere in the entire cache, a memory reference can map only to the subset of cache slots.
The number of cache blocks per set in set associative cache varies according to overall system design. For example, a 2-way set associative cache can be conceptualized as shown in the schematic below. Each set contains two different memory blocks.



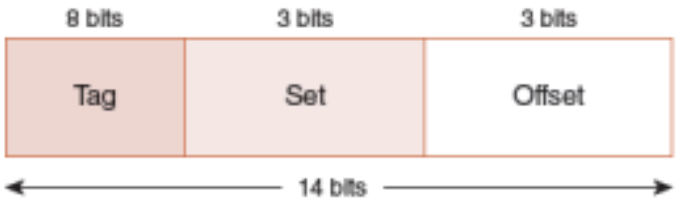
A Logical view of 2-way set associative cache



B Linear view of 2-way set associative cache

In set associative cache mapping, a memory reference is divided into three fields: tag, set, and offset. As with direct-mapped cache, the offset field chooses the byte within the cache block, and the tag field uniquely identifies the memory address. The set field determines the set to which the memory block maps.

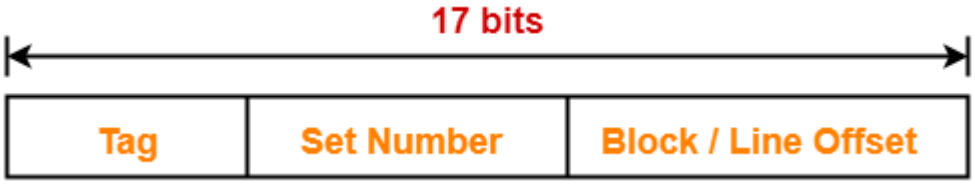
Suppose we are using 2-way set associative mapping with a byte-addressable main memory of 214 bytes and a cache with 16 blocks, where each block contains 8 bytes. – Cache has a total of 16 blocks, and each set has 2 blocks, then there are 8 sets in cache. – Thus, the set field is 3 bits, the offset field is 3 bits, and the tag field is 8 bits.



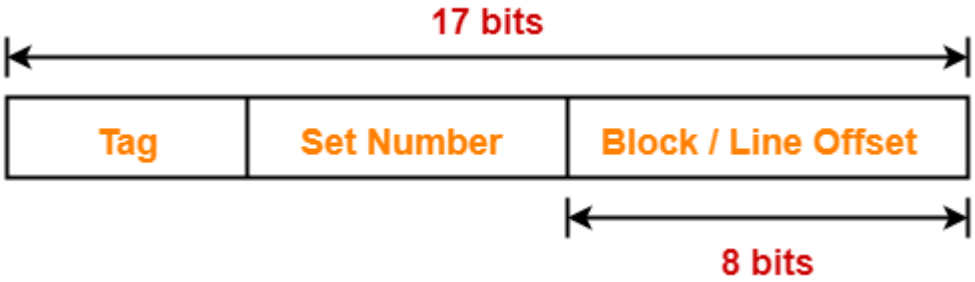
Problems on Set Associative Mapping

1. Consider a 2-way set associative mapped cache of size 16 KB with block size 256 bytes. The size of main memory is 128 KB. Find-
   Number of bits in tag
   Tag directory size
   Solution-
   Given-
   Set size = 2
   Cache memory size = 16 KB
   Block size = Frame size = Line size = 256 bytes
   Main memory size = 128 KB
   We consider that the memory is byte addressable.
   Number of Bits in Physical Address-

   We have,
   Size of main memory
   = 128 KB
   = 217 bytes
   Thus, Number of bits in physical address = 17 bits

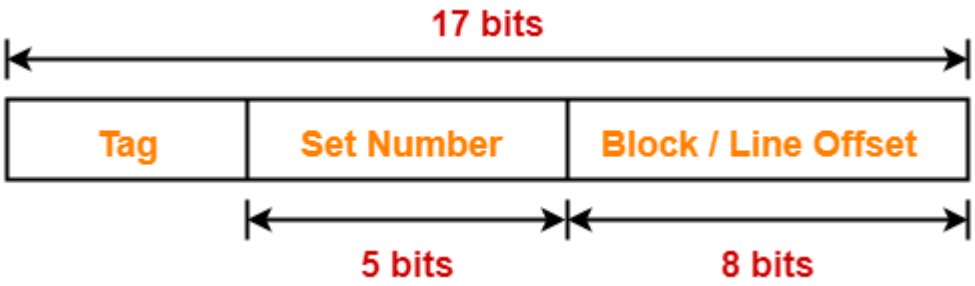


   Number of Bits in Block Offset-
   We have,
   Block size
   = 256 bytes
   = 28 bytes
   Thus, Number of bits in block offset = 8 bits

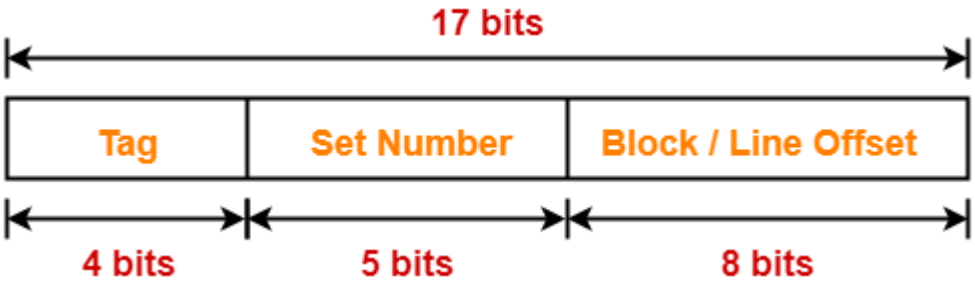


   Number of Lines in Cache-
   Total number of lines in cache
   = Cache size / Line size
   = 16 KB / 256 bytes

= 214 bytes / 28 bytes
= 64 lines
Thus, Number of lines in cache = 64 lines
Number of Sets in Cache-
Total number of sets in cache
= Total number of lines in cache / Set size
= 64 / 2
= 32 sets
= 25 sets
Thus, Number of bits in set number = 5 bits



Number of Bits in Tag-
Number of bits in tag
= Number of bits in physical address – (Number of bits in set number + Number of bits in block offset)
= 17 bits – (5 bits + 8 bits)
= 17 bits – 13 bits
= 4 bits
Thus, Number of bits in tag = 4 bits



Tag Directory Size-
Tag directory size
= Number of tags x Tag size
= Number of lines in cache x Number of bits in tag
= 64 x 4 bits
= 256 bits
= 32 bytes
Thus, size of tag directory = 32 bytes

Simulation of a fully associative cache with 8 lines each of 1 word.

Simulation program maintains an array with 8 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 tags. In this case the loop index, ***i***, will be set to 8. 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.

As before, the cache simulator outputs a line for every meomory access. When the end of the trace file is reached a summary is printed out.

```
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) {
      /* simulate fully associative cache with 8 words */
      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);
   }
```

Use trace.txt as below

FILE - trace.txt : **0x80000000 0x80000004 0x80000008 0x8000000C 0x00000020 0x80000010 0x80000014 0x80000018 0x8000000C 0x00000024 0x80000010 0x80000014 0x80000018 0x8000000C 0x00000028 0x80000010 0x80000014 0x80000018 0x8000000C 0x0000002C 0x80000010 0x80000014 0x80000018 0x8000000C 0x00000030 0x80000010 0x80000014 0x80000018 0x8000000C 0x00000034 0x80000010 0x80000014 0x80000018 0x8000000C 0x00000038 0x80000010 0x80000014 0x80000018 40 0x8000000C 0x0000003C 0x80000010 0x80000014 0x80000018 0x8000000C 0x00000040 0x80000010 0x80000014 0x80000018 0x8000000C 0x00000044 0x80000010 0x80000014 0x80000018 0x8000000C 0x00000048 0x80000010 0x80000014 0x80000018 0x8000000C 0x0000004C 0x80000010 0x80000014 0x80000018 0x8000000C 0x00000050 0x80000010 0x80000014 0x80000018 0x8000000C 0x00000054 0x80000010 0x80000014 0x80000018 0x8000000C 0x00000058 0x80000010 0x80000014 0x80000018 0x8000000C 0x0000005C 0x80000010 0x80000014 0x80000018 0x8000000C 0x00000060 0x80000010 0x80000014 0x80000018 0x8000000C 0x00000064 0x80000010 0x80000014 0x80000018 0x8000000C 0x00000068 0x80000010 0x80000014 0x80000018 0x8000000C 0x0000006C 0x80000010 0x80000014 0x80000018**

Lab Assignments to complete in this session

Compile and execute the fully-associative cache simulator provided above. Report the final number of hits and accesses output by the code. Based on the pattern of cache hits, estimate the hit rate of the given trace file generated by miniMIPs code fragment in the steady state (once the compulsary misses are accounted for).

Conclusion:

From this experiment, we learn about the importance of cache memory as it improves the efficiency of data retrieval. It stores program instructions and data that are used repeatedly in the operation of programs or information that the CPU is likely to need next. Fast access to these instructions increases the overall speed of the program. A major advantage of fully associative mapped cache is its simplicity and ease of implementation. However, it has its own disadvantage, which is that it is expensive as it requires storing addresses along with the data. Set Associative mapping, on the other hand, has the highest hit-ratio compared to two previous two cache memories discussed above. Thus, its performance is considerably better. However, Set-Associative cache memory is very expensive. As the size of set increases, the cost increases.

Code:

```c
#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, i, j, t, 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("%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);
    fclose(fp);
}
```

Output:

```
  1: 0x80000000 Miss 0x80000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0 7 6 5 4 3 2 1
  2: 0x80000004 Miss 0x80000000 0x80000004 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 1 0 7 6 5 4 3 2
  3: 0x80000008 Miss 0x80000000 0x80000004 0x80000008 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 2 1 0 7 6 5 4 3
  4: 0x8000000c Miss 0x80000000 0x80000004 0x80000008 0x8000000c 0x00000000 0x00000000 0x00000000 0x00000000 3 2 1 0 7 6 5 4
  5: 0x00000020 Miss 0x80000000 0x80000004 0x80000008 0x8000000c 0x00000020 0x00000000 0x00000000 0x00000000 4 3 2 1 0 7 6 5
  6: 0x80000010 Miss 0x80000000 0x80000004 0x80000008 0x8000000c 0x00000020 0x80000010 0x00000000 0x00000000 5 4 3 2 1 0 7 6
  7: 0x80000014 Miss 0x80000000 0x80000004 0x80000008 0x8000000c 0x00000020 0x80000010 0x80000014 0x00000000 6 5 4 3 2 1 0 7
  8: 0x80000018 Miss 0x80000000 0x80000004 0x80000008 0x8000000c 0x00000020 0x80000010 0x80000014 0x80000018 7 6 5 4 3 2 1 0
  9: 0x8000000c Hit3 0x80000000 0x80000004 0x80000008 0x8000000c 0x00000020 0x80000010 0x80000014 0x80000018 3 7 6 5 4 2 1 0
 10: 0x00000024 Miss 0x00000024 0x80000004 0x80000008 0x8000000c 0x00000020 0x80000010 0x80000014 0x80000018 0 3 7 6 5 4 2 1
 11: 0x80000010 Hit5 0x00000024 0x80000004 0x80000008 0x8000000c 0x00000020 0x80000010 0x80000014 0x80000018 5 0 3 7 6 4 2 1
 12: 0x80000014 Hit6 0x00000024 0x80000004 0x80000008 0x8000000c 0x00000020 0x80000010 0x80000014 0x80000018 6 5 0 3 7 4 2 1
 13: 0x80000018 Hit7 0x00000024 0x80000004 0x80000008 0x8000000c 0x00000020 0x80000010 0x80000014 0x80000018 7 6 5 0 3 4 2 1
 14: 0x8000000c Hit3 0x00000024 0x80000004 0x80000008 0x8000000c 0x00000020 0x80000010 0x80000014 0x80000018 3 7 6 5 0 4 2 1
 15: 0x00000028 Miss 0x00000024 0x00000028 0x80000008 0x8000000c 0x00000020 0x80000010 0x80000014 0x80000018 1 3 7 6 5 0 4 2
 16: 0x80000010 Hit5 0x00000024 0x00000028 0x80000008 0x8000000c 0x00000020 0x80000010 0x80000014 0x80000018 5 1 3 7 6 0 4 2
 17: 0x80000014 Hit6 0x00000024 0x00000028 0x80000008 0x8000000c 0x00000020 0x80000010 0x80000014 0x80000018 6 5 1 3 7 0 4 2
 18: 0x80000018 Hit7 0x00000024 0x00000028 0x80000008 0x8000000c 0x00000020 0x80000010 0x80000014 0x80000018 7 6 5 1 3 0 4 2
 19: 0x8000000c Hit3 0x00000024 0x00000028 0x80000008 0x8000000c 0x00000020 0x80000010 0x80000014 0x80000018 3 7 6 5 1 0 4 2
 20: 0x0000002c Miss 0x00000024 0x00000028 0x0000002c 0x8000000c 0x00000020 0x80000010 0x80000014 0x80000018 2 3 7 6 5 1 0 4
 21: 0x80000010 Hit5 0x00000024 0x00000028 0x0000002c 0x8000000c 0x00000020 0x80000010 0x80000014 0x80000018 5 2 3 7 6 1 0 4
 22: 0x80000014 Hit6 0x00000024 0x00000028 0x0000002c 0x8000000c 0x00000020 0x80000010 0x80000014 0x80000018 6 5 2 3 7 1 0 4
 23: 0x80000018 Hit7 0x00000024 0x00000028 0x0000002c 0x8000000c 0x00000020 0x80000010 0x80000014 0x80000018 7 6 5 2 3 1 0 4
 24: 0x8000000c Hit3 0x00000024 0x00000028 0x0000002c 0x8000000c 0x00000020 0x80000010 0x80000014 0x80000018 3 7 6 5 2 1 0 4
 25: 0x00000030 Miss 0x00000024 0x00000028 0x0000002c 0x8000000c 0x00000030 0x80000010 0x80000014 0x80000018 4 3 7 6 5 2 1 0
 26: 0x80000010 Hit5 0x00000024 0x00000028 0x0000002c 0x8000000c 0x00000030 0x80000010 0x80000014 0x80000018 5 4 3 7 6 2 1 0
 27: 0x80000014 Hit6 0x00000024 0x00000028 0x0000002c 0x8000000c 0x00000030 0x80000010 0x80000014 0x80000018 6 5 4 3 7 2 1 0
 28: 0x80000018 Hit7 0x00000024 0x00000028 0x0000002c 0x8000000c 0x00000030 0x80000010 0x80000014 0x80000018 7 6 5 4 3 2 1 0
 29: 0x8000000c Hit3 0x00000024 0x00000028 0x0000002c 0x8000000c 0x00000030 0x80000010 0x80000014 0x80000018 3 7 6 5 4 2 1 0
 30: 0x00000034 Miss 0x00000034 0x00000028 0x0000002c 0x8000000c 0x00000030 0x80000010 0x80000014 0x80000018 0 3 7 6 5 4 2 1
 31: 0x80000010 Hit5 0x00000034 0x00000028 0x0000002c 0x8000000c 0x00000030 0x80000010 0x80000014 0x80000018 5 0 3 7 6 4 2 1
 32: 0x80000014 Hit6 0x00000034 0x00000028 0x0000002c 0x8000000c 0x00000030 0x80000010 0x80000014 0x80000018 6 5 0 3 7 4 2 1
 33: 0x80000018 Hit7 0x00000034 0x00000028 0x0000002c 0x8000000c 0x00000030 0x80000010 0x80000014 0x80000018 7 6 5 0 3 4 2 1
 34: 0x8000000c Hit3 0x00000034 0x00000028 0x0000002c 0x8000000c 0x00000030 0x80000010 0x80000014 0x80000018 3 7 6 5 0 4 2 1
 35: 0x00000038 Miss 0x00000034 0x00000038 0x0000002c 0x8000000c 0x00000030 0x80000010 0x80000014 0x80000018 1 3 7 6 5 0 4 2
 36: 0x80000010 Hit5 0x00000034 0x00000038 0x0000002c 0x8000000c 0x00000030 0x80000010 0x80000014 0x80000018 5 1 3 7 6 0 4 2
 37: 0x80000014 Hit6 0x00000034 0x00000038 0x0000002c 0x8000000c 0x00000030 0x80000010 0x80000014 0x80000018 6 5 1 3 7 0 4 2
 38: 0x80000018 Hit7 0x00000034 0x00000038 0x0000002c 0x8000000c 0x00000030 0x80000010 0x80000014 0x80000018 7 6 5 1 3 0 4 2
 39: 0x00000040 Miss 0x00000034 0x00000038 0x00000040 0x8000000c 0x00000030 0x80000010 0x80000014 0x80000018 2 7 6 5 1 3 0 4
 40: 0x8000000c Hit3 0x00000034 0x00000038 0x00000040 0x8000000c 0x00000030 0x80000010 0x80000014 0x80000018 3 2 7 6 5 1 0 4
 41: 0x0000003c Miss 0x00000034 0x00000038 0x00000040 0x8000000c 0x0000003c 0x80000010 0x80000014 0x80000018 4 3 2 7 6 5 1 0
 42: 0x80000010 Hit5 0x00000034 0x00000038 0x00000040 0x8000000c 0x0000003c 0x80000010 0x80000014 0x80000018 5 4 3 2 7 6 1 0
 43: 0x80000014 Hit6 0x00000034 0x00000038 0x00000040 0x8000000c 0x0000003c 0x80000010 0x80000014 0x80000018 6 5 4 3 2 7 1 0
 44: 0x80000018 Hit7 0x00000034 0x00000038 0x00000040 0x8000000c 0x0000003c 0x80000010 0x80000014 0x80000018 7 6 5 4 3 2 1 0
 45: 0x8000000c Hit3 0x00000034 0x00000038 0x00000040 0x8000000c 0x0000003c 0x80000010 0x80000014 0x80000018 3 7 6 5 4 2 1 0
 46: 0x00000040 Hit2 0x00000034 0x00000038 0x00000040 0x8000000c 0x0000003c 0x80000010 0x80000014 0x80000018 2 3 7 6 5 4 1 0
 93: 0x80000014 Hit6 0x00000064 0x00000058 0x00000060 0x8000000c 0x0000005c 0x80000010 0x80000014 0x80000018 6 5 0 3 7 2 4 1
 94: 0x80000018 Hit7 0x00000064 0x00000058 0x00000060 0x8000000c 0x0000005c 0x80000010 0x80000014 0x80000018 7 6 5 0 3 2 4 1
 95: 0x8000000c Hit3 0x00000064 0x00000058 0x00000060 0x8000000c 0x0000005c 0x80000010 0x80000014 0x80000018 3 7 6 5 0 2 4 1
 96: 0x00000068 Miss 0x00000064 0x00000068 0x00000060 0x8000000c 0x0000005c 0x80000010 0x80000014 0x80000018 1 3 7 6 5 0 2 4
 97: 0x80000010 Hit5 0x00000064 0x00000068 0x00000060 0x8000000c 0x0000005c 0x80000010 0x80000014 0x80000018 5 1 3 7 6 0 2 4
 98: 0x80000014 Hit6 0x00000064 0x00000068 0x00000060 0x8000000c 0x0000005c 0x80000010 0x80000014 0x80000018 6 5 1 3 7 0 2 4
 99: 0x80000018 Hit7 0x00000064 0x00000068 0x00000060 0x8000000c 0x0000005c 0x80000010 0x80000014 0x80000018 7 6 5 1 3 0 2 4
100: 0x8000000c Hit3 0x00000064 0x00000068 0x00000060 0x8000000c 0x0000005c 0x80000010 0x80000014 0x80000018 3 7 6 5 1 0 2 4
101: 0x0000006c Miss 0x00000064 0x00000068 0x00000060 0x8000000c 0x0000006c 0x80000010 0x80000014 0x80000018 4 3 7 6 5 1 0 2
102: 0x80000010 Hit5 0x00000064 0x00000068 0x00000060 0x8000000c 0x0000006c 0x80000010 0x80000014 0x80000018 5 4 3 7 6 1 0 2
103: 0x80000014 Hit6 0x00000064 0x00000068 0x00000060 0x8000000c 0x0000006c 0x80000010 0x80000014 0x80000018 6 5 4 3 7 1 0 2
104: 0x80000018 Hit7 0x00000064 0x00000068 0x00000060 0x8000000c 0x0000006c 0x80000010 0x80000014 0x80000018 7 6 5 4 3 1 0 2
Hits = 77, Accesses = 104, Hit ratio = 0.740385
```