**Experiment 6**

**Aim**

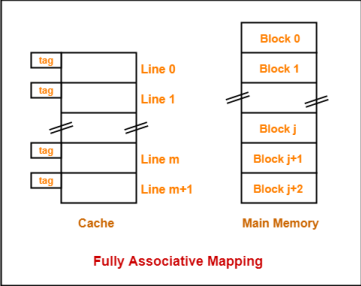
To implement fully associative mapping and set associative mapping.

**Theory**

A CPU cache is a hardware used by the central processing unit (CPU) of a computer to reduce the average cost (time) of accessing data from the main memory. A cache is a smaller, faster memory, located closer to a processor core, which stores copies of the data from frequently used main memory locations. Cache memory is costlier than main memory or disk memory but economical than CPU registers. It is an extremely fast memory type that acts as a buffer between RAM and the CPU, and holds frequently requested data and instructions so that they are immediately available to the CPU when needed.

**Fully associative mapping**

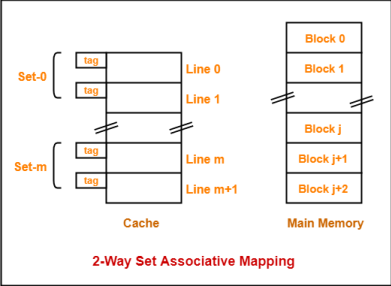
Fully Associative Mapping is a cache mapping technique that allows to map a block of main memory to any freely available cache line. In this type of mapping, the associative memory is used to store content and addresses of the memory word. Any block can go into any line of the cache. This means that 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 considered to be the fastest and the most flexible mapping form.



**Set associative mapping**

Set-associative cache is a trade-off between direct-mapped cache and fully associative cache. This form of mapping is an enhanced form of direct mapping where the drawbacks of direct

mapping is removed. Set associative addresses the problem of possible thrashing in the direct mapping method. It does this by saying that instead of having exactly one line that a block can map to in the cache, we will group a few lines together creating a ***set***. Then a block in memory can map to any one of the lines of a specific sea. 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.



**Prelab code for generating trace file**

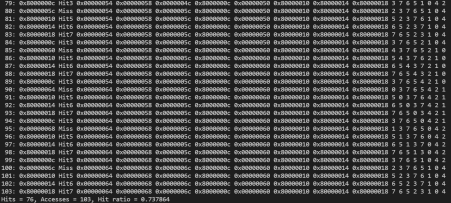
| main: addu $t0,$0,$0   addiu $t1,$0,80   addu $t2,$0,$0  loop: lw $t3,array($t0)   addu $t2,$t2,$t3   addiu $t0,$t0,4   bne $t0,$t1,loop  \*done: beq $0,$0,done  array: .word 1,2,3,4,5,6,7,8,9,10   .word 11,12,13,14,15,16,17,18,19,20 |
| --- |

**Fully Associative Mapping – 8 bit**

**Code:**

| #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);  pclose(fp);  } |
| --- |

**Output:**

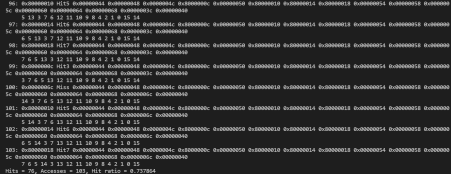
****

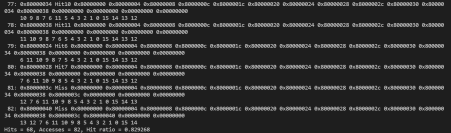
**Fully Associative Mapping – 16 bit**

**Code:**

| #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;  // find index in mru  for (i = 0; i < 16; 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 16 words \*/  accesses += 1;  printf("%3d: 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]);  printf("\n ");  for (i = 0; i < 16; i++)  printf("%d ", mru[i]);  printf("\n");  }  printf("Hits = %d, Accesses = %d, Hit ratio = %f\n", hits, accesses, ((float)hits)/accesses);  close(fp);  } |
| --- |

**Output: Fully Associative Mapping – 16 bit**

****

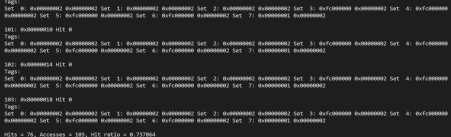
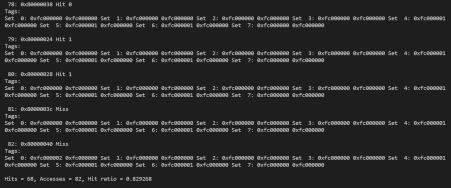


**Set Associative Mapping – 16 bit**

**Code:**

| #include <stdio.h>  #include <stdlib.h>  #include <math.h>  void main() {  int\*\* tags = malloc(2 \* sizeof(int\*));  tags[0] = malloc(16 \* sizeof(int));  tags[1] = malloc(16 \* sizeof(int));  int mru[8] = {1,1,1,1,1,1,1,1};  int sets = 8;  int addr;     int hits, accesses;  FILE \*fp;  fp = fopen("trace.txt", "r");  hits = 0;  accesses = 0;  while (fscanf(fp, "%x", &addr) > 0) {  accesses++;  int setNum = (addr >> 2) & (int)(pow(2, log2(sets)) - 1);  int addrTag = addr >> (2 + (int)log2(sets));  printf("\n%3d: 0x%08x ", accesses, addr);  int i;  for(i = 0; i < 2; ++i) {  if(tags[i][setNum] == addrTag) {  printf("Hit %d", i);  hits++;  mru[setNum] = i;  break;  }  }  if(i == 2) {  printf("Miss");  tags[(mru[setNum] + 1) % 2][setNum] = addrTag;  mru[setNum] = (mru[setNum] + 1) % 2;  }  printf("\nTags:\n");  for(int i = 0; i < sets; ++i) {  printf("Set %2d: 0x%08x 0x%08x ", i, tags[0][i], tags[0][1]);  }  printf("\n");  }  printf("\nHits = %d, Accesses = %d, Hit ratio = %f\n", hits, accesses, ((float)hits)/accesses);  fclose(fp);  } |
| --- |

**Output: Set Associative Mapping – 16 bit**

****

**Conclusion**

A cache is a smaller, quicker memory that stores copies of software instructions and data that are utilized regularly in the operation of programs and is positioned closer to the processor core. The program's overall speed is improved by having quick access to these commands. In this experiment, we have implemented fully associative and set associative memory cache and applied it to a generated trace.