CS2610: Computer Organization and Architecture

Lab 10: Report

Devansh Singh Rathore(111701011)

Objective

In this lab we used the C programming language to write simulators for several cache organizations and evaluate their performance. We were provided with a series of memory addresses which were to be accessed in trace.txt file.

Problem

Executing/Implementing the following:

(1) Direct Mapped Cache
(2) Fully Associative Cache
(3) 2 Way Set Associative Cache
(4) 4 Way Set Associative Cache

Implementation

(1)Direct Mapped Cache:

CODE:

```
#include<stdio.h>

int tag[8]; //Tag array: for storing tags in the cache

int main()
{
      int addr;           //address
      int i, t;           //index, tag
      int hits, accesses; //#hits, #accesses
      FILE *fp;
```

```
	fp = fopen("trace.txt", "r");	//Accessing trace.txt using
pointer

	hits = 0;		//initialization
	accesses = 0;

	while(fscanf(fp, "%x", &addr) > 0)
	{
		accesses++;			//incrementing the #accesses
		printf("%3d: 0x%08x ", accesses, addr); //printing the
#accesses and the new address

		//The bits in positions 0 and 1 are offset bits
		i = (addr >> 2) & 7; //The bits in position 2, 3 and 4 are the
index
		t = addr | 0x1f; //The bits 5 - 31 are the tag bits


		if(tag[i] == t) //Checking for a hit
		{
			hits ++; // incrementing the #hits
			printf("Hit%d ",i);
		}
		else //Otherwise its a miss
		{
			printf("Miss ");
			tag[i] = t;
		}

		for(i = 0;i < 8;i++) //printing the tag array data
		{
			printf("0x%08x\n", tag[i]);
		}
	}
	printf("Hits = %d, Accesses = %d, Hit rate = %f\n", hits, accesses,
(float)hits / accesses); // Print the number of hits, number of accesses
and the hit rate
	fclose(fp); // close the file used
	return 0;
}
```

OUTPUT OF CODE:
Number of Hits (#hits) = 68
Number of Accesses (#accesses) = 103
Hit rate = 0.660194

(2)Fully Associative Cache:

CODE:

```
#include<stdio.h>

int tag[8]; //Tag array: for storing tags in the cache
int mru[8] = {7, 6, 5, 4, 3, 2, 1, 0}; // The array for most recently used
cache spaces
//index 0 -> 7: lesser the index, more it is recent.

void mruUpdate(int way)
{
	int i;
	for(i = 0;i < 8;i++)
	{
		if(mru[i] == way) //For finding how recent way element was.
			break;
	}

	while(i > 0) //for updating the mru list.
	{
		mru[i] = mru[i-1];
		i--;
	}

	mru[0] = way; //updating the most recently used element.
}

int main()
{
	int addr;		//address
	int i, t;		//index, tag
```

```
      int hits, accesses; //#hits, #accesses
      FILE *fp;

      fp = fopen("trace.txt", "r");   //Accessing trace.txt using pointer

      hits = 0;           //initialization
      accesses = 0;

      while(fscanf(fp, "%x", &addr) > 0)
      {
            t = addr | 0x3;             //extracting the tag from the
address
            accesses ++;                //incrementing the #accesses
            printf("%3d: 0x%08x ", accesses, t);

            for(i = 0;i < 8;i++) // Searching the tag in the cache
            {
                  if(tag[i] == t) //Checking for a hit
                  {
                        hits ++; // incrementing the #hits
                        printf("Hit%d ", i);
                        mruUpdate(i); //for updating the mru list
                        break;
                  }
            }


            if(i == 8) //condition for miss
            {
                  printf("Miss ");
                  i = mru[7];          // Get the least recently used way
                  tag[i] = t;          // Place the new tag in the way
                  mruUpdate(i);        // Place this way at the start of
the mru array
            }

            for(i = 0; i < 8;i++) //Printing the tag values
            {
                  printf("0x%08x\n", tag[i]);
            }
            for(i = 0; i < 8;i++) // Printing the mru array
            {
```

```
                printf("%d ", mru[i]);
            }
        }

        printf("Hits = %d, Accesses = %d, Hit rate = %f\n", hits, accesses,
(float)hits / accesses); // Print the number of hits, number of accesses
and the hit rate
        fclose(fp); // close the file used
        return 0;
}
```

OUTPUT OF CODE:
Final mru list: 7 6 5 2 3 1 0 4
Number of Hits (#hits) = 76
Number of Accesses (#accesses) = 103
Hit rate = 0.737864

(3)2 Way Set Associative Cache:

CODE:

```
#include<stdio.h>

int tag[4][2];
int mru[4] = {1, 1, 1, 1};

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

        fp = fopen("trace.txt", "r");
        hits = 0;
        accesses = 0;
        while(fscanf(fp, "%x", &addr) > 0)
        {
                //Cache design::
                //Two offset bits - 0, 1
                //Two index bits - 2, 3
                //28 tag bits - [4,31]
```

```
            accesses++; // incrementing the number of accesses.
            int ind = (addr>>2)&3; //index

            int k;
            int tg = addr | 0xf; //tag

            //searching for that particular tag value
            for(k = 0;k < 2;k++) //iterates through the set corresponding
to 'ind' index.
            {
                  if(tag[ind][k] == tg) //For checking if Tag is in the kth
way of the ind set or not.
                  {
                        printf("Its a Hit");
                        mru[ind] = k;
                        hits++;
                        break;
                  }
            }

            if(k == 2) //Checking if Tag was not found, hence its a miss
            {
                  printf("Its a Miss");
                  //least recently used way corresponding to the ind set
                  int x = (mru[ind] + 1)%2;
                  //Updating the tag value
                  tag[ind][x] = tg;
                  //Changing the mru for this set to the way used now
                  mru[ind] = x;
            }

            for(int i = 0;i < 4;i++) //Printing the array
            {
                  printf(" 0x%08x 0x%08x\n",tag[i][0], tag[i][1]);
            }

            for(int i = 0;i < 4;i++) //Printing the mru array
                  printf("%d ", mru[i]);
            printf("\n");
      }

      printf("Hits = %d, Accesses = %d, Hit ratio = %f\n", hits, accesses,
```

```
(float)hits/accesses);
      fclose(fp);
      return 0;
}
```

OUTPUT OF CODE:
Number of Hits (#hits) = 76
Number of Accesses (#accesses) = 103
Hit rate = 0.737864

(4)4 Way Set Associative Cache:

Approach:
We will now take a 2D array for mru, for storing the ways for each set associative in mru ordering. We will update the mru according to the input set and way arguments. While rest of the problem remains same instead that the main loop inside the while loop will run upto index 4, and will check for miss condition if that index reaches value 4 instead of 2.

Observations

Key observations in performing the above executions :

1. There are different ways of cache access, and hit/miss rate varies according to the methods.
2. The hit rate of direct mapped cache here was lesser than the hit rate of fully/2 way set associative cache.
3. Hence the overall miss of direct mapped cache was more than the other methods.

Conclusions

We learnt a method for using the C programming language to write simulators for several cache organizations and evaluating their performance.