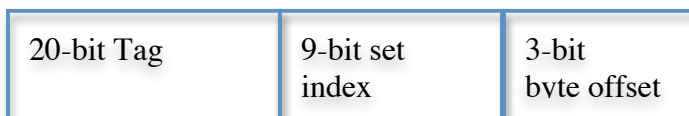


## Solutions To CSCE 4610/5610: Exam 2

1. (15%) Consider a processor with a 16 Kbyte 4-way set associative cache, with 8-byte lines. Show how you will translate 32-bit address to find the data in cache. Show the cache set to which the following (hex) address falls: ABCED8F8. How many bits are needed for tags in this cache?

**Key:**

A 16K Byte cache with 8-byte line will contain 2048 lines and with 4-way set associativity, we will have 512 sets (each set with 4 lines). Thus we need 9-bits to locate a set, and 3-bits to locate a byte within a line (since each line has 8bytes). So we will divide a 32 bit address as follows.



Address ABCED8F8 = 1010 1011 1100 1110 1101 1000 1111 1000  
 Yields 1 000 1111 1 = 287 as the set number (and byte zero in the line)  
 This leave the tag as ABCED = 1010 1011 1100 1110 1101

There are a total of 2048 lines and each line will have 20 bit tags. So the total number of bits for tags = 20\*2048 = 40,960 bits

2. (30%). Consider a system with L-1 and L-2 caches, main memory and disk (or secondary memory) with the following parameters. It takes 10 ns to access L-1cache, 100ns to access L-2 Cache, 500 ns to access RAM and 10ms to access disk. A program makes 250,000 memory references of which 215,000 are hits in L-1 cache, 235,000 are hit in L-2 cache, 245,000 are hit in RAM.

- a). What are the L-1, L2 cache and RAM miss ratios?
- b). Estimate the average access time assuming all memory accesses are reads.
- c). If we are using write-back policy for L-1, 75% all memory accesses are reads and 50% of all cache lines are dirty, what is the effective memory access time?

**Key:**

- a). L-1 Miss rate = 35000/25000 = 0.14 (or 14%)  
 global L-2 miss rate = 15000/250000 = 0.06 (6%)  
 local L-2 miss rate = 15000/35000 = 0.4286 (42.86%)  
 global RAM miss rate = 5000/250000 = 0.02 (2%)  
 local RAM miss rate = 5000/15000 = 0.33 (33%)

b). Access time = (215000/250000)\*10 + (20000/250000)\*100 + (10000/250000)\*500 + (5000/250000)\*10,000,000  
 = 0.86\*10 + 0.08\*100 + 0.04\*500 + 0.02\*10,000,000 = 200,036.6ns

c). We have a write back L-1, on a miss in L-1 we may have to write the dirty line back to memory 50% of the time. The information that 75% of all accesses are reads is not relevant,

since either on a read or a write, a miss will cause a dirty line to be written back. So, on a miss in L-1, we have to allow for 1.5 accesses to L2.

$$\begin{aligned} \text{Access time} &= (215000/250000)*10 + 1.5* (20000/250000)*100 + (10000/250000)*500 \\ &\quad + (5000/250000)*10,000,000 \\ &= 0.86*10 + 1.5* 0.08*100 + 0.04*500 + 0.02*10,000,000 = 200,040.6\text{ns} \end{aligned}$$

3. (25%) Assuming a memory system of previous problem, let us investigate the performance gains using Multithreading. Let us assume that the context switch time is 10ns and the CPU cycle time is 1ns.

a). If we context switch only on a miss in RAM, compute the parameters “run-length” and the time for the long latency operation. Using these, compute the maximum speed-up that can be achieved with multithreading and the number of threads needed to reach this speed up.

b). Repeat above calculations assuming a context switch on every L-2 cache miss.

**Key.**

a). We have context switch on a RAM, we have 5000 misses out of 250000 accesses or R = run length = 50 accesses = 50\*10 = 500ns in terms of L-1 access time = 500 instructions.

L = The miss penalty on RAM miss = 10ms = 10,000,000 ns.

$$\text{Maximum Speed up} = (R+L)/(R+C) = (500 + 10,000,000)/(500+10) = 19,608.8$$

$$\text{Numbers of threads needed} = 19,608.8$$

b). If we context switch on L-2 miss, we have 15,000 misses out of 250,000 references, or R= run length = 16.67 accesses = 166.7 ns in terms of L-1 accesses = 166.7 instructions

L= Miss penalty on a L-2 miss = 500ns (if you ignore RAM misses)

$$\text{Speed up} = (166.7+500)/(166.7+10) = 3.77$$

But, Miss penalty as far as L-2 is concerned should include the misses in RAM. Note of the 15000 accesses which are missed in L-2, 10,000 are hits in RAM and the remaining 5000 are hits in the disk.

The effective miss penalty for L-2

$$= (10,000/15000)*500 + (5000/15000)*10,000,000 = 3,333,667\text{ns}$$

$$\text{Speed up} = (166.7+3,333,667)/(166.7+10) = 18,867.19$$

4. (30%) Consider the following program segment in C.

```
for (k= 0; k < 50; k++)
{
    for (i =0; i < 100; i ++)
```

```

    {
        sum = 0;
        for (j =0; j < 100; j ++)
            if ( i !=j) sum = sum + a[i][j]*x[j];
        new_x[i] = (b[i] - sum)/a[i][ i];
    }
    for (i =0; i < 100; i ++)
        x[i] = new_x[i];
}

```

Let us only worry about memory accesses generated by the arrays a, b, x, new\_x (ignoring accesses for scalar variables like i, j, k, sum) and each element is one word (32 bits).

- What is the expected cache miss rate if each cache block is 16 bytes (and we will assume the cache is large enough so that any data brought into cache is not displaced). Initially the data is not in cache (cold start)
- What is the expected cache miss if the cache block size is increased to 32-bytes each?

**Key:**

- Each line of the cache is 16 byte and will hold 4 array elements. So each time a line is brought into cache, 4 consecutive array elements are brought into cache. Matrices are stored row-major and thus the 4 elements belong to the same row.

Consider the loop

```

    for (j =0; j < 100; j ++)
        if ( i !=j) sum = sum + a[i][j]*x[j];

```

Here we are access the elements of the ith row. So we will have  $100/4 = 25$  misses for accessing  $a[i][*]$ .

So for the outer i-loop which is repeated 100 times, we will have 25 misses for each  $a[i][*]$ .

Total a misses =  $100*25 = 2500$

We will have to bring x, new\_x, and b arrays only once, and we will have  $100/4 = 25$  misses each for b, new\_x and x arrays.

```

    for (i =0; i < 100; i ++)
        x[i] = new_x[i];

```

will not generate any additional misses since new\_x is already in cache.

So, the total number of misses =  $2500+25+25+25 = 2575$  misses.

Number of accesses:

Note that in

```

    for (j =0; j < 100; j ++)
        if ( i !=j) sum = sum + a[i][j]*x[j];

```

we have 100 accesses a and 100 access to x.

Then if we consider the outer loop

```
for (i =0; i < 100; i ++)  
  {  
    sum = 0;  
    for (j =0; j < 100; j ++)  
      if ( i !=j) sum = sum + a[i][j]*x[j];  
    new_x[i] = (b[i] - sum)/a[i][ i];
```

For each i iteration we have 100+1 (for a[i][i]) accesses to a; 100 access to x, 1 access to new\_x and one access to b. The total number of access are

```
100*(101) access to a;  
100*100 access to x  
100 accesses to new_x  
100 access to b
```

The total number of access = 20,300

The

```
for (i =0; i < 100; i ++)  
  x[i] = new_x[i];  
}
```

will add 100 accesses each to x and new\_x

The accesses for each k loop add to 20,500

Since we have 50 values for k, the total number of access = 50\*20,500 =102,500

So miss rate = 2575/102500 =0.25%

b). If the line size is doubled to 32 bytes, each line will have 8 elements. The total number of misses will be halved for accessing each array

We will have  $100/8 = 13$  misses (note we will have 4 wasted elements in the last access) for accessing each row of a, and 13 misses accessing each of the b, new\_x and x arrays.

Total misses =  $100*13 + 13 +13+13 = 1339$  misses

The miss rate =  $1339/102500 =0.13\%$