CSci 2021, Fall 2018 Written Exercise Set 4

Due: at beginning of lecture November 28

Submit this assignment on paper at **the beginning** of your lecture section. We strongly recommend that you type and print out your solutions. Please label your assignment with your name, UMN email address, and the time of your recitation section (10:10, 11:15, 12:20, or 1:25).

Problem 1: (This problem is closely related to O'Hallaron 6.26)

The following table gives the parameters for a number of different caches, where m is the number of physical address bits, C is the cache size (number of data bytes), B is the block size in bytes, E the number of cache sets (S), tag bits (t), set index bits (s), and block offset bits (b). Complete the table by entering the correct values in the missing fields. *Hint*: consider special cases such as direct-map and fully associative caches.

Cache	m	C	B	E	S	t	s	b
1.	32		16	1		22	6	4
2.	32	4096	512	8				
3.	32	2048		2	128			3
4.	64	8192		1	32	51		

Problem 2: (This problem is closely related to O'Hallaron 6.29)

The following table contains the contents of the cache. All addresses, tags, and values inside of the table are represented in hexadecimal. The cache used for this question has a few properties that are important to note:

- Addresses are 12 bits wide.
- Memory is byte addressable.
- Memory accesses are to 1-byte words (not 4-byte words).
- The cache is two-way set associative (E=2), with 4-byte block size (B=4) and four sets (S=4).

Set Index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	00	0	-	-	-	-
	FD	1	82	20	13	FF
1	00	1	EE	6A	42	41
	3F	0	-	-	-	-
2	01	1	76	2A	17	81
	63	0	-	-	-	-
3	2F	1	5C	24	65	12
	64	1	2B	94	AA	A2

A. The diagram below shows the layout of an address. Label each box in the diagram with the fields that appear in the address. Use the following symbols to label the diagram:

O - The Cache Offset
I - The Cache set index
T - The Cache Tag

11	10	9	8	7	6	5	4	3	2	1	0

B. For the given memory addresses, state if the access was a hit or a miss. If The read was a hit, state the value that it would return.

- Read 0xFD2
- Read 0x3F7
- Read 0x2fC
- Read 0x63A

Problem 3: (This problem is closely related to O'Hallaron 6.37)

This is a problem that tests your ability to analyze the cache behavior of C code. Assume we execute the three summation functions shown below under the following conditions:

- sizeof(double) = 8.
- The machine has a 16 KB cache with a 32-byte block size.
- The cache is two-way set associative and uses LRU eviction policy.
- Within the two loops, the code uses memory accesses only for the array data. The loop indices (i.e. variables i and j) and the values sum are held in registers.
- Array a is stored starting at memory address 0x8000000.

Please find the approximate cache miss rates for the case $N = 128$ in functions `sumA`, `sumB` and `sumC`. The three summation functions are shown below.

```
typedef double array_t[N][N]

int sumA(array_t a) {
    double i, j;
    double sum = 0;
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++) {
            sum += a[i][j];
        }
    return sum;
}

int sumB(array_t a) {
    double i, j;
    double sum = 0;
    for (j = 0; j < N; j+=2)
        for (i = 0; i < N; i+=2){
        sum += (a[i][j] + a[i][j+1]) +
               (a[i+1][j] + a[i+1][j+1];
        }
    return sum;
}
```

```
int sumC(array_t a) {
    double i, j;
    double sum = 0;
    for (i = 0; i < N; i+=2)
        for (j = 0; j < N; j++){
            sum += (a[i][j] + a[i+1][j]);
        }
    return sum;
}
```

Problem 4: (This problem is closely related to O'Hallaron 6.38)

A bitmap image is composed of pixels. Each pixel in the image is represented as four values: three for the primary colors(red, green and blue - RGB) and one for the transparency information defined as an alpha channel.

In this problem, you will compare the performance of direct mapped and 4-way associative caches for a square bitmap image initialization. Both caches have a size of 128 bytes. The direct mapped cache has 8-byte blocks while the 4-way associative cache has 4-byte blocks.

You are given the following definitions

```
typedef struct {
    unsigned char r;
    unsigned char g;
    unsigned char b;
    unsigned char a;
} pixel_t;

pixel_t  pixel[16][16];
register int i, j;
```

Also assume that

- sizeof(unsigned char) = 1
- pixel begins at memory address 0
- Both caches are initially empty
- The array is stored in row-major order
- Variables i,j are stored in registers and any access to these variables does not cause a cache miss

```
for (i = 0; i < 16; i ++){
    for (j = 0; j < 16; j ++){
        pixel[i][j].r = 0;
        pixel[i][j].g = 0;
        pixel[i][j].b = 0;
        pixel[i][j].a = 0;
    }
}
```

A. What fraction of the writes in the above code will result in a miss in the direct mapped cache?

B. Using code in part A, what fraction of the writes will result in a miss in the 4-way associative cache?

```
for (i = 0; i < 16; i ++){
    for (j = 0; j < 16; j ++){
        pixel[j % 4][i].r = 0;
        pixel[j % 4][i].g = 0;
        pixel[j % 4][i].b = 0;
        pixel[j % 4][i].a = 0;
    }
}
```

C. What fraction of the writes in the above code will result in a miss in the direct mapped cache?

D. Using code in part C, what fraction of the writes will result in a miss in the 4-way associative cache?

Hint: explain how you arrive at your answers for partial credit. You may find it helpful to consider the total number of writes and total number or misses. It may also be helpful to write out the memory access patterns by computing pixel addresses and calculating t, s, and b.