6.30 ◆

Suppose we have a system with the following properties:

- The memory is byte addressable.
- Memory accesses are to 1-byte words (not to 4-byte words).
- Addresses are 13 bits wide.
- The cache is 4-way set associative ($E = 4$), with a 4-byte block size ($B = 4$) and eight sets ($S = 8$).

Consider the following cache state. All addresses, tags, and values are given in hexadecimal format. The Index column contains the set index for each set of four lines. The Tag columns contain the tag value for each line. The V columns contain the valid bit for each line. The Bytes 0–3 columns contain the data for each line, numbered left to right starting with byte 0 on the left.

4-way set associative cache

Index	Tag	V	Bytes 0–3	Tag	V	Bytes 0–3	Tag	V	Bytes 0–3	Tag	V	Bytes 0–3
0	F0	1	ED 32 0A A2	8A	1	BF 80 1D FC	14	1	EF 09 86 2A	BC	0	25 44 6F 1A
1	BC	0	03 3E CD 38	A0	0	16 7B ED 5A	BC	1	8E 4C DF 18	E4	1	FB B7 12 02
2	BC	1	54 9E 1E FA	B6	1	DC 81 B2 14	00	0	B6 1F 7B 44	74	0	10 F5 B8 2E
3	BE	0	2F 7E 3D A8	C0	1	27 95 A4 74	C4	0	07 11 6B D8	BC	0	C7 B7 AF C2
4	7E	1	32 21 1C 2C	8A	1	22 C2 DC 34	BC	1	BA DD 37 D8	DC	0	E7 A2 39 BA
5	98	0	A9 76 2B EE	54	0	BC 91 D5 92	98	1	80 BA 9B F6	BC	1	48 16 81 0A
6	38	0	5D 4D F7 DA	BC	1	69 C2 8C 74	8A	1	A8 CE 7F DA	38	1	FA 93 EB 48
7	8A	1	04 2A 32 6A	9E	0	B1 86 56 0E	CC	1	96 30 47 F2	BC	1	F8 1D 42 30

A. What is the size (C) of this cache in bytes?

B. The box that follows shows the format of an address (1 bit per box). Indicate (by labeling the diagram) the fields that would be used to determine the following:

CO. The cache block offset
CI. The cache set index
CT. The cache tag

12	11	10	9	8	7	6	5	4	3	2	1	0

6.31 ◆◆

Suppose that a program using the cache in Problem 6.30 references the 1-byte word at address `0x071A`. Indicate the cache entry accessed and the cache byte value returned *in hex*. Indicate whether a cache miss occurs. If there is a cache miss, enter "—" for "Cache byte returned." *Hint:* Pay attention to those valid bits!

A. Address format (1 bit per box):

12	11	10	9	8	7	6	5	4	3	2	1	0

B. Memory reference:

Parameter	Value
Block offset (CO)	0x______
Index (CI)	0x______
Cache tag (CT)	0x______
Cache hit? (Y/N)	______
Cache byte returned	0x______

6.32 ◆◆

Repeat Problem 6.31 for memory address `0x16E8`.

A. Address format (1 bit per box):

12	11	10	9	8	7	6	5	4	3	2	1	0

B. Memory reference:

Parameter	Value
Cache offset (CO)	0x______
Cache index (CI)	0x______
Cache tag (CT)	0x______
Cache hit? (Y/N)	______
Cache byte returned	0x______

6.33 ◆◆

For the cache in Problem 6.30, list the eight memory addresses (in hex) that will hit in set 2.

6.34 ◆◆

Consider the following matrix transpose routine:

```c
typedef int array[4][4];

void transpose2(array dst, array src)
{
    int i, j;

```

```
        for (i = 0; i < 4; i++) {
            for (j = 0; j < 4; j++) {
                dst[j][i] = src[i][j];
            }
        }
    }
```

Assume this code runs on a machine with the following properties:

- `sizeof(int)` = 4.
- The `src` array starts at address 0 and the `dst` array starts at address 64 (decimal).
- There is a single L1 data cache that is direct-mapped, write-through, write-allocate, with a block size of 16 bytes.
- The cache has a total size of 32 data bytes, and the cache is initially empty.
- Accesses to the `src` and `dst` arrays are the only sources of read and write misses, respectively.

A. For each `row` and `col`, indicate whether the access to `src[row][col]` and `dst[row][col]` is a hit (h) or a miss (m). For example, reading `src[0][0]` is a miss and writing `dst[0][0]` is also a miss.

dst array

	Col. 0	Col. 1	Col. 2	Col. 3
Row 0	m	____	____	____
Row 1	____	____	____	____
Row 2	____	____	____	____
Row 3	____	____	____	____

src array

	Col. 0	Col. 1	Col. 2	Col. 3
Row 0	m	____	____	____
Row 1	____	____	____	____
Row 2	____	____	____	____
Row 3	____	____	____	____

6.35 ◆◆

Repeat Problem 6.34 for a cache with a total size of 128 data bytes.

dst array

	Col. 0	Col. 1	Col. 2	Col. 3
Row 0	____	____	____	____
Row 1	____	____	____	____
Row 2	____	____	____	____
Row 3	____	____	____	____

src array

	Col. 0	Col. 1	Col. 2	Col. 3
Row 0	____	____	____	____
Row 1	____	____	____	____
Row 2	____	____	____	____
Row 3	____	____	____	____

6.36 ◆◆

This problem tests your ability to predict the cache behavior of C code. You are given the following code to analyze:

```
    int x[2][128];
    int i;
    int sum = 0;

    for (i = 0; i < 128; i++) {
        sum += x[0][i] * x[1][i];
    }
```

Assume we execute this under the following conditions:

- sizeof(int) = 4.
- Array x begins at memory address 0x0 and is stored in row-major order.
- In each case below, the cache is initially empty.
- The only memory accesses are to the entries of the array x. All other variables are stored in registers.

Given these assumptions, estimate the miss rates for the following cases:

A. Case 1: Assume the cache is 512 bytes, direct-mapped, with 16-byte cache blocks. What is the miss rate?

B. Case 2: What is the miss rate if we double the cache size to 1,024 bytes?

C. Case 3: Now assume the cache is 512 bytes, two-way set associative using an LRU replacement policy, with 16-byte cache blocks. What is the cache miss rate?

D. For case 3, will a larger cache size help to reduce the miss rate? Why or why not?

E. For case 3, will a larger block size help to reduce the miss rate? Why or why not?

6.37 ◆◆

This is another problem that tests your ability to analyze the cache behavior of C code. Assume we execute the three summation functions in Figure 6.47 under the following conditions:

- sizeof(int) = 4.
- The machine has a 4 KB direct-mapped cache with a 16-byte block size.
- Within the two loops, the code uses memory accesses only for the array data. The loop indices and the value sum are held in registers.
- Array a is stored starting at memory address 0x08000000.

Fill in the table for the approximate cache miss rate for the two cases $N = 64$ and $N = 60$.

Function	$N = 64$	$N = 60$
sumA		
sumB		
sumC		

```
typedef int array_t[N][N];

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

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

Figure 6.47 Functions referenced in Problem 6.37.

6.38 ◆

3M decides to make Post-its by printing yellow squares on white pieces of paper. As part of the printing process, they need to set the CMYK (cyan, magenta, yellow, black) value for every point in the square. 3M hires you to determine the efficiency of the following algorithms on a machine with a 1,024-byte direct-mapped data cache with 16-byte blocks. You are given the following definitions:

```
struct point_color {
    int c;
    int m;
    int y;
    int k;
};

struct point_color square[16][16];
int i, j;
```

Assume the following:

- `sizeof(int)` = 4.
- `square` begins at memory address 0.
- The cache is initially empty.
- The only memory accesses are to the entries of the array `square`. Variables `i` and `j` are stored in registers.

Determine the cache performance of the following code:

```
    for (i = 15; i >= 0; i--){
        for (j = 15; j >= 0; j--) {
            square[i][j].c = 0;
            square[i][j].m = 0;
            square[i][j].y = 1;
            square[i][j].k = 0;
        }
    }
```

A. What is the total number of writes?

B. What is the total number of writes that hit in the cache?

C. What is the hit rate?

6.39 ◆

Given the assumptions in Problem 6.38, determine the cache performance of the following code:

```
    for (i = 15; i >= 0; i--){
        for (j = 15; j >= 0; j--) {
            square[j][i].c = 0;
            square[j][i].m = 0;
            square[j][i].y = 1;
            square[j][i].k = 0;
        }
    }
```

A. What is the total number of writes?

B. What is the total number of writes that hit in the cache?

C. What is the hit rate?

6.40 ◆

Given the assumptions in Problem 6.38, determine the cache performance of the following code:

```
for (i = 15; i >= 0; i--) {
    for (j = 15; j >= 0; j--) {
        square[i][j].y = 1;
    }
}
for (i = 15; i >= 0; i--) {
    for (j = 15; j >= 0; j--) {
        square[i][j].c = 0;
        square[i][j].m = 0;
        square[i][j].k = 0;
    }
}
```

A. What is the total number of writes?

B. What is the total number of writes that hit in the cache?

C. What is the hit rate?

6.41 ◆◆

You are writing a new 3D game that you hope will earn you fame and fortune. You are currently working on a function to blank the screen buffer before drawing the next frame. The screen you are working with is a 640 × 480 array of pixels. The machine you are working on has a 32 KB direct-mapped cache with 8-byte lines. The C structures you are using are as follows:

```
struct pixel {
    char r;
    char g;
    char b;
    char a;
};

struct pixel buffer[480][640];
int i, j;
char *cptr;
int *iptr;
```

Assume the following:

- `sizeof(char)` = 1 and `sizeof(int)` = 4.

- `buffer` begins at memory address 0.
- The cache is initially empty.
- The only memory accesses are to the entries of the array `buffer`. Variables `i`, `j`, `cptr`, and `iptr` are stored in registers.

What percentage of writes in the following code will hit in the cache?

```
    for (j = 639; j >= 0; j--) {
        for (i = 479; i >= 0; i--){
            buffer[i][j].r = 0;
            buffer[i][j].g = 0;
            buffer[i][j].b = 0;
            buffer[i][j].a = 0;
        }
    }
```

6.42 ◆◆

Given the assumptions in Problem 6.41, what percentage of writes in the following code will hit in the cache?

```
    char *cptr = (char *) buffer;
    for (; cptr < (((char *) buffer) + 640 * 480 * 4); cptr++)
        *cptr = 0;
```

6.43 ◆◆

Given the assumptions in Problem 6.41, what percentage of writes in the following code will hit in the cache?

```
    int *iptr = (int *)buffer;
    for (; iptr < ((int *)buffer + 640*480); iptr++)
        *iptr = 0;
```

6.44 ◆◆◆

Download the `mountain` program from the CS:APP Web site and run it on your favorite PC/Linux system. Use the results to estimate the sizes of the caches on your system.

6.45 ◆◆◆◆

In this assignment, you will apply the concepts you learned in Chapters 5 and 6 to the problem of optimizing code for a memory-intensive application. Consider a procedure to copy and transpose the elements of an $N \times N$ matrix of type `int`. That is, for source matrix S and destination matrix D, we want to copy each element $s_{i,j}$ to $d_{j,i}$. This code can be written with a simple loop,

```
void transpose(int *dst, int *src, int dim)
{
    int i, j;

```