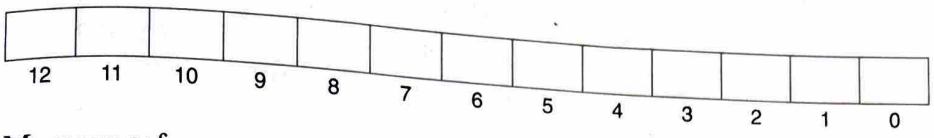
A. Address format (1 bit per box):



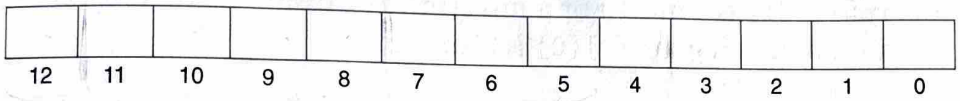
B. Memory reference:

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):



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:

```
1   typedef int array[4][4];
2
3   void transpose2(array dst, array src)
4   {
5       int i, j;
6
```

```
7           for (i = 0; i < 4; i++) {
8               for (j = 0; j < 4; j++) {
9                   dst[j][i] = src[i][j];
10              }
11          }
12      }
```

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:

```
1       int x[2][128];
2       int i;
```

```
3       int sum = 0;
4
5       for (i = 0; i < 128; i++) {
6           sum += x[0][i] * x[1][i];
7       }
```

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 2,048-byte direct-mapped data cache with 32-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 = 0; i < 16; i++){
    for (j = 0; j < 16; 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 miss in the cache?

C. What is the miss rate?

6.39 ◆

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

```
for (i = 0; i < 16; i++){
    for (j = 0; j < 16; j++) {
        square[j][i].c = 0;
        square[j][i].m = 0;
        square[j][i].y = 1;
        square[j][i].k = 0;
    }
}
```