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;

```

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

where the arguments to the procedure are pointers to the destination (dst) and source (src) matrices, as well as the matrix size N (dim). Your job is to devise a transpose routine that runs as fast as possible.

6.46 ◆◆◆◆

This assignment is an intriguing variation of Problem 6.45. Consider the problem of converting a directed graph g into its undirected counterpart g'. The graph g' has an edge from vertex u to vertex v if and only if there is an edge from u to v or from v to u in the original graph g. The graph g is represented by its *adjacency matrix* G as follows. If N is the number of vertices in g, then G is an $N \times N$ matrix and its entries are all either 0 or 1. Suppose the vertices of g are named $v_0, v_1, v_2, \ldots, v_{N-1}$. Then $G[i][j]$ is 1 if there is an edge from v_i to v_j and is 0 otherwise. Observe that the elements on the diagonal of an adjacency matrix are always 1 and that the adjacency matrix of an undirected graph is symmetric. This code can be written with a simple loop:

```
void col_convert(int *G, int dim) {
    int i, j;

    for (i = 0; i < dim; i++)
        for (j = 0; j < dim; j++)
            G[j*dim + i] = G[j*dim + i] || G[i*dim + j];
}
```

Your job is to devise a conversion routine that runs as fast as possible. As before, you will need to apply concepts you learned in Chapters 5 and 6 to come up with a good solution.

Solutions to Practice Problems

Solution to Problem 6.1 (page 620)

The idea here is to minimize the number of address bits by minimizing the aspect ratio $\max(r, c)/\min(r, c)$. In other words, the squarer the array, the fewer the address bits.

Organization	r	c	b_r	b_c	$\max(b_r, b_c)$
16×1	4	4	2	2	2
16×4	4	4	2	2	2
128×8	16	8	4	3	4
512×4	32	16	5	4	5
$1{,}024 \times 4$	32	32	5	5	5