Fall ’22 CIS 314 Assignment 7 – 100/100 points – Due Sunday, 11/27, 11:59 PM

Please submit individual source files for coding exercises (see naming conventions below) and a single solution document for non-coding exercises (.txt or .pdf only). Your code and answers need to be documented to the point that the graders can understand your thought process. Full credit will not be awarded if sufficient work is not shown.

1. [40] Assume that we’re going to simulate a cache with 256B blocks and 16 sets for a 32-bit architecture. Write a C program with the following functions, using only bitwise ops for the arithmetic (no division or modulo operators):

- (10) *unsigned int getOffset(unsigned int address)* – returns the byte offset of the address within its cache block.
- (10) *unsigned int getSet(unsigned int address)* – returns the cache set for the address.
- (10) *unsigned int getTag(unsigned int address)* – returns the cache tag for the address.
- (10) An int main() function to test your functions.

Here are some test runs:

```
0x12345678: offset: 78 - tag: 12345 - set: 6
0x87654321: offset: 21 - tag: 87654 - set: 3
```

Name your source file 7-1.c.

2. [60] Consider the following C code:

```
struct ColorPoint {
    long a;
    long r;
    long g;
    long b;
};
```

Also consider a 64B cache, organized as a single cache block. Assume the following:

- sizeof(long) == 8.
- *points* begins at memory address 0.
- The cache is initially empty.
- The only memory accesses are to the entries of the array points. Variables *i*, *j*, and *sum* are stored in registers (see code below).

Hint: Memory is row major and cache blocks are contiguous regions of memory (see chapter 6.2-6.3).

a. (20) Consider the following C code, in the context of the assumptions above:

```
long f(struct ColorPoint **points, int n) {
    long sum = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            sum += points[j][i].a;
            sum += points[j][i].r;
            sum += points[j][i].g;
            sum += points[j][i].b;
        }
    }
    return sum;
}
```

What is the miss rate of the cache when running this code assuming that *n* is 32? Answer in terms of *misses/(hits + misses)* in your solutions document.

Hint: identify the pattern of misses and hits for the first few structs and extrapolate.

b. (20) Consider the following C code, in the context of the assumptions above:

```
long g(struct ColorPoint **points, int n) {
    long sum = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            sum += points[i][j].a;
            sum += points[i][j].r;
            sum += points[i][j].g;
            sum += points[i][j].b;
        }
    }
    return sum;
}
```

What is the miss rate of the cache when running this code assuming that *n* is 32? Answer in terms of *misses/(hits + misses)* in your solutions document.

Hint: identify the pattern of misses and hits for the first few structs and extrapolate.

c. (20) Consider the following C code:

```
struct ColorPoint** create2DArray(int n) {
    // Array to hold a pointer to the beginning of each row
    struct ColorPoint **points =
        (struct ColorPoint **)malloc(n * sizeof(struct ColorPoint *));
    for (int i = 0; i < n; ++i) {
        // Array to hold each row
        points[i] =
            (struct ColorPoint *)malloc(n * sizeof(struct ColorPoint));
        for (int j = 0; j < n; ++j) {
            // Init the ColorPoint struct
            points[i][j].a = rand();
            points[i][j].r = rand();
            points[i][j].g = rand();
            points[i][j].b = rand();
```

```
        }
    }
    return points;
}

void free2DArray(struct ColorPoint** points, int n) {
    for (int i = 0; i < n; ++i) {
        free(points[i]);
    }
    free(points);
}
```

The above helper functions allocate (and free, respectively) a 2D (n x n) array of ColorPoint structs.

Using these methods, write an int main() function that allocates a 2048 x 2048 array of ColorPoint structs and then measures the time taken to call the *f* and *g* functions above with this array as an argument. Compile with -Og. Which is faster? Explain why this might be the case in your code comments. Don't forget to free the memory used for the array!

Hint: time how long it takes to call each function some constant number of times (e.g., 1000 – see examples from class); this will ensure that execution times aren't rounded down to 0.

Name your source file 7-2c.c.

Zip the source files and solution document (if applicable), name the .zip file <Your Full Name>Assignment7.zip (e.g., EricWillsAssignment7.zip), and upload the .zip file to Canvas (see Assignments section for submission link).