Fall ’17 CIS 314 Assignment 8 – 100/100 points – Due Friday, 12/1, 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. [20] Assume that you’re given the task of optimizing graphics software that operates in the CMYK (Cyan, Magenta, Yellow, Black) color space. Specifically, you need to determine the efficiency of the following algorithm on a machine with a 2048-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];
```

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 = 1;
        square[i][j].m = 0;
        square[i][j].y = 0;
        square[i][j].k = 0;
    }
}
```

A. What is the total number of memory writes? It may help to think in terms of movl instructions.

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

C. What is the miss rate?

Include your answers in your solutions document.

2. [80] Write a C program to simulate a simple 64B direct-mapped cache with 4B blocks (i.e., 16 cache sets). This program builds upon many of the topics covered in this course and will require a comprehensive knowledge of caching theory (see chapter 6.4). Your program should:

- Define a struct to represent a cache block. Your struct should define a bit that represents whether or not the block is valid (use an unsigned char), a 26-bit tag (use an unsigned int), and a 4-byte value (use an unsigned char[4]).
- Allocate enough memory to hold 16 cache blocks (1 per set). Initialize your cache such that all blocks are invalid.
- Repeatedly prompt the user to either:
    - Write a value – prompt the user for a 32-bit hex address and a 32-bit hex value. Write the 4-byte value at the appropriate location in the cache for the given address, evicting an existing block if necessary. You may assume that the user enters an address that is a multiple of 4 so that the beginning of the value aligns with the beginning of a cache block. This is to simulate moving an entire block from memory into the cache. If a valid block will be evicted, print the set index, tag, and value of that block. Either way, print the set index, tag, and value of the new block being written.
    - Read a byte – prompt the user for a 32-bit hex address. Print the set index and tag corresponding to the address. Also print "hit" and the hex value of the byte at that address if the cache contains a valid value corresponding to the address. Otherwise, print "miss" and indicate the reason for the miss (i.e., invalid block or tag mismatch).
    - Print values – print the set index, tag, and value for all valid blocks in the cache.
    - Quit – quit the simulation.
- Use only bitwise operators for address-translation arithmetic (i.e., computing the set index, block offset, and/or tag for a given address); do not use division or modulo arithmetic.
- Hint: use scanf(" %x") and printf("%x") to input and output hex values directly.
- Hint: use B&O'H 2.57 (from Assignment 1) as a guide for printing the cache values as individual bytes.

Name your source file 8-1.c.

Here is output from a sample run of the application (your output does not need to match exactly):

```
Enter 'r' for read, 'w' for write, 'p' to print, 'q' to quit: w
```

```
Enter 32-bit unsigned hex address: 0x0
Enter 32-bit unsigned hex value: 0xaabb
wrote set: 0 - tag: 0 - valid: 1 - value: bb aa 00 00
Enter 'r' for read, 'w' for write, 'p' to print, 'q' to quit: w
Enter 32-bit unsigned hex address: 0x8
Enter 32-bit unsigned hex value: 0xbbcc
wrote set: 2 - tag: 0 - valid: 1 - value: cc bb 00 00
Enter 'r' for read, 'w' for write, 'p' to print, 'q' to quit: p
set: 0 - tag: 0 - valid: 1 - value: bb aa 00 00
set: 2 - tag: 0 - valid: 1 - value: cc bb 00 00
Enter 'r' for read, 'w' for write, 'p' to print, 'q' to quit: w
Enter 32-bit unsigned hex address: 0x40
Enter 32-bit unsigned hex value: 0xccdd
evicting block - set: 0 - tag: 0 - valid: 1 - value: bb aa 00 00
wrote set: 0 - tag: 1 - valid: 1 - value: dd cc 00 00
Enter 'r' for read, 'w' for write, 'p' to print, 'q' to quit: p
set: 0 - tag: 1 - valid: 1 - value: dd cc 00 00
set: 2 - tag: 0 - valid: 1 - value: cc bb 00 00
Enter 'r' for read, 'w' for write, 'p' to print, 'q' to quit: r
Enter 32-bit unsigned hex address: 0x0
looking for set: 0 - tag: 0
found set: 0 - tag: 1 - offset: 0 - valid: 1 - value: dd
tags don't match - miss!
Enter 'r' for read, 'w' for write, 'p' to print, 'q' to quit: r
Enter 32-bit unsigned hex address: 0x4
looking for set: 1 - tag: 0
no valid set found - miss!
Enter 'r' for read, 'w' for write, 'p' to print, 'q' to quit: r
```

```
Enter 32-bit unsigned hex address: 0x8
looking for set: 2 - tag: 0
found set: 2 - tag: 0 - offset: 0 - valid: 1 - value: cc
hit!
Enter 'r' for read, 'w' for write, 'p' to print, 'q' to quit: r
Enter 32-bit unsigned hex address: 0x40
looking for set: 0 - tag: 1
found set: 0 - tag: 1 - offset: 0 - valid: 1 - value: dd
hit!
Enter 'r' for read, 'w' for write, 'p' to print, 'q' to quit: r
Enter 32-bit unsigned hex address: 0x41
looking for set: 0 - tag: 1
found set: 0 - tag: 1 - offset: 1 - valid: 1 - value: cc
hit!
Enter 'r' for read, 'w' for write, 'p' to print, 'q' to quit: q
```

Zip the source files and solution document (if applicable), name the .zip file <Your Full Name>Assignment8.zip (e.g., EricWillsAssignment8.zip), and upload the .zip file to Canvas (see Assignments section for submission link).