Winter ’23 CIS 314 Assignment 8 – 100/100 points – Due Friday, 3/17, 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), when appropriate. 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.

Consider the following C code, which is part of a program to simulate a 16B direct-mapped cache with 2B blocks (i.e., 8 cache sets):

```
#include <stdio.h>
#include <stdlib.h>

#define BLOCK_SIZE 2

struct Set {
    unsigned char data[BLOCK_SIZE];
    unsigned int tag; // Assume tag is at most 32 bits
    unsigned char valid; // valid bit (0 or 1)
};

struct Cache {
    struct Set *sets;
    int numSets;
};

unsigned int getOffset(unsigned int address) {
    // TODO – return unsigned-int value of offset bits from address
}

unsigned int getSet(unsigned int address) {
    // TODO – return unsigned-int value of set bits from address
}

unsigned int getTag(unsigned int address) {
    // TODO – return unsigned-int value of tag bits from address
}

struct Cache* mallocCache(int numSets) {
    // TODO - malloc a pointer to a struct Cache, malloc a pointer to an array
    // of struct Set instances (array length is numSets).  Also initialize
    // valid to 0 for each struct Set.  Return the struct Cache pointer.
}

void freeCache(struct Cache *cache) {
    free(cache->sets);
    free(cache);
}

void printSet(struct Set *set, int setIndex) {
    printf("set: %x - tag: %x - valid: %u - data:", setIndex, set->tag, set->valid);
    unsigned char *data = set->data;
    for (int i = 0; i < BLOCK_SIZE; ++i) {
        printf(" %.2x", data[i]);
    }
    printf("\n");
}

void printCache(struct Cache *cache) {
    // TODO - print all valid sets in the cache.
}

void readValue(struct Cache *cache, unsigned int address) {
    // TODO - check the cache for a cached byte at the specified address.
    // If found, indicate a hit and print the byte.  If not found, indicate
```

```c
    // a miss due to either an invalid line (cold miss) or a tag mismatch
    // (conflict miss).
}

void writeValue(struct Cache *cache, unsigned int address, unsigned char *newData) {
    unsigned int s = getSet(address);
    unsigned int t = getTag(address);

    struct Set *set = &cache->sets[s];
    if (set->valid && set->tag != t) {
        unsigned char *data = set->data;
        printf("evicting line - ");
        printSet(set, s);
    }

    unsigned char *data = set->data;
    for (int i = 0; i < BLOCK_SIZE; ++i) {
        data[i] = newData[i];
    }

    set->tag = t;
    set->valid = 1;

    printf("wrote set: ");
    printSet(set, s);
}

unsigned int readUnsignedIntFromHex() {
    char buffer[10];
    char *p = NULL;
    unsigned int n;
    while (1) {
        fgets(buffer, sizeof(buffer), stdin);
        n = strtoul(buffer, &p, 16);
        if (buffer != p) {
            break;
        }
        printf("Invalid input - try again: ");
    }
    return n;
}

int main() {
    struct Cache *cache = mallocCache(8);

    char buffer[10];
    char c;
    do {
        printf("Enter 'r' for read, 'w' for write, 'p' to print, 'q' to quit: ");
        fgets(buffer, sizeof(buffer), stdin);

        c = buffer[0];
        if (c == 'r') {
            printf("Enter 32-bit unsigned hex address: ");
            unsigned int a = readUnsignedIntFromHex();
            readValue(cache, a);
        } else if (c == 'w') {
            printf("Enter 32-bit unsigned hex address: ");
            unsigned int a = readUnsignedIntFromHex();

            printf("Enter 32-bit unsigned hex value: ");
            unsigned int v = readUnsignedIntFromHex();
            unsigned char *data = (unsigned char *)&v;
            writeValue(cache, a, data);
        } else if (c == 'p') {
            printCache(cache);
        }
    } while (c != 'q');

    freeCache(cache);
}
```

Copy and paste the above code into a C file and implement the following methods:

1. [10] getOffset

2. [10] getSet

3. [10] getTag

4. [20] mallocCache

5. [20] printCache

6. [30] readValue

See inline comments above for descriptions.

Hint: study chapter 6.4 and the writeValue procedure above carefully and ask questions about the parts that you don't understand!

Name your source file 8-1.c.

Here is output from a sample run of the program (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: set: 0 - tag: 0 - valid: 1 - data: bb aa
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: 0xccdd
wrote set: set: 4 - tag: 0 - valid: 1 - data: dd cc
Enter 'r' for read, 'w' for write, 'p' to print, 'q' to quit: p
set: 0 - tag: 0 - valid: 1 - data: bb aa
set: 4 - tag: 0 - valid: 1 - data: dd cc
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: set: 0 - tag: 0 - valid: 1 - data: bb aa
hit: bb
Enter 'r' for read, 'w' for write, 'p' to print, 'q' to quit: w
Enter 32-bit unsigned hex address: 0x10
Enter 32-bit unsigned hex value: 0xeeff
evicting line - set: 0 - tag: 0 - valid: 1 - data: bb aa
wrote set: set: 0 - tag: 1 - valid: 1 - data: ff ee
Enter 'r' for read, 'w' for write, 'p' to print, 'q' to quit: p
set: 0 - tag: 1 - valid: 1 - data: ff ee
set: 4 - tag: 0 - valid: 1 - data: dd cc
Enter 'r' for read, 'w' for write, 'p' to print, 'q' to quit: r
Enter 32-bit unsigned hex address: 0x4
looking for set: 2 - tag: 0
no valid line found - miss!
```

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).