CSE2421 AU12 HOMEWORK #1

**Problem 2.58**
Write a procedure “is_little_endian” that will return 1 when compiled and run on a little-endian machine, and will return 0 when compiled and run on a big_endian machine. This program should run on any machine, regardless of its word size.

**Problem 2.61**
Write C expressions that evaluate to 1 when the following conditions are true, and to 0 when they are false. Assume x is of type int.

- A. Any bit of x equals 1.
- B. Any bit of x equals 0.
- C. Any bit in the least significant byte of x equals 1.
- D. Any bit in the most significant byte of x equals 0.

Your code should follow the bit-level integer coding rules (see below), with the additional restriction that you may not use equality (==) or inequality (!=) tests.

**BIT-LEVEL INTEGER CODING RULES**
In several of the following problems, we will artificially restrict what programming constructs you can use to help you gain a better understanding of the bit-level, logic, and arithmetic operations of C. In answering these problems, your code must follow these rules:

**Assumptions**
- Integers are represented in two’s-complement form.
- Right shifts of signed data are performed arithmetically.
- Data type int is w bits long. For some of the problems, you will be given a specific value for w, but otherwise your code should work as long as w is a multiple of 8. You can use the expression sizeof(int)<<3 to compute w.

**Forbidden**
- Conditionals (if or ?:), loops, switch statements, function calls, and macro invocations.
- Division, modulus and multiplication.
- Relative comparison operators(<, >, <=, and >=).
- Casting, either explicit or implicit.

**Allowed operations**
- All bit-level and logic operations.
- Left and right shifts, but only with shift amounts between 0 and w-1.
- Addition and subtraction.
- Equality (==) and inequality (!=) tests (some of the problems do not allow these).
- Integer constants INT_MIN and INT_MAX.

**Problem 2.64**
Write code to implement the following function:
/* Return 1 when any odd bit of x equals 1; 0 otherwise. Assume w=32. */
int any_odd_one(unsigned x);

Your function should follow the bit-level integer coding rules (above), except that you may assume that data type int has w = 32 bits.
Problem 2.70
Write code for the function with the following prototype:

```c
int fits_bits(int x, int n);
```

Your function should follow the bit-level integer coding rules (above)

Problem 2.72
You are given the task of writing a function that will copy an integer “val” into a buffer “buf”, but it should do so only if enough space is available in the buffer. Here is the code you write:

```c
void copy_int(int val, void *buf, int maxbytes) {
    if (maxbytes - sizeof(val) >= 0)
        memcpy(buf, (void *) &val, sizeof(val));
}
```

This code makes use of the library function “memcpy”. Although its use is a bit artificial here, where we simply want to copy an int, it illustrates an approach commonly used to copy larger data structures.

You carefully test the code and discover that it always copies the value to the buffer, even when “maxbytes” is too small.

A. Explain why the conditional test in the code always succeeds. HINT: The sizeof operator returns a value of type size_t.
B. Show how you can rewrite the conditional test to make it work properly.

Problem 2.76
Suppose we are given the task of generating code to multiply integer variable x by various different constant factors K. To be efficient, we want to use only the operations +, -, and <<. For the following values of K, write C expressions to perform the multiplication using at most three operations per expression.

A. K = 17;
B. K = -7;
C. K = 60;
D. K = -112;

Problem 2.80
Write C expressions to generate the bit patterns that follow, where a\(^k\) represents k repetitions of symbol a. Assume a w-bit data type. Your code may contain references to parameters j and k, representing the values of j and k, but not a parameter representing w.

A. 1\(^w-k\) 0\(^k\)
B. 0\(^w-k-j\) 1\(^k\) 0\(^j\)