**CMPSC 210**
**Principles of Computer Organization**
**Fall 2016**
**John Wenskovitch**
`http://cs.allegheny.edu/~jwenskovitch/teaching/CMPSC210`

**Lab 7 – Bit Masking**
**Due (via Bitbucket) Wednesday, 2 November 2016**
**30 points**

# Lab Goals

- Practice bit masking in both C and MIPS
- Solve a programming challenge using floating-point numbers

# Assignment Details

Before the exam, we discussed floating-point numbers and the IEEE 754 standard. In short, all floating-point numbers conform to a standard representation. For single-precision numbers such as `float`s, that pattern is 1 bit for the sign, 8 bits for the exponent, and 23 bits for the significand. In this lab, you will be writing code in both C and MIPS to read a floating-point number from the user and break it apart into each of its three components.

Note that you have two weeks to finish this lab, but start early!

## Part 1: Bit Masking in C (15 points)

Bit masking is a useful way to obtain partial information from a stored value. For example, you can use a bit mask to recover an individual character from a string, or to get the last three digits of an integer. Another clear use is to break apart a floating-point number into the sign, exponent, and significand components.

For this program, you should request a floating-point number from the user, and print each of the following items:

- The hexadecimal representation of the number
- The binary representation of the number, with spaces to separate the sign/exponent/significand components
- The sign of the number
- The exponent of the number (non-biased)
- The significand of the number (don't forget the implicit leading 1)

Here is some sample output that you can try to imitate (red text is user input):

```
jwenskovitch$ ./decode
Enter a float value: -11.1
In hex: 0xc131999a
In binary: 1 10000010 01100011001100110011010
Sign: -
Exponent: 2^3
Significand: 1.387500

jwenskovitch$ ./decode
Enter a float value: .005
In hex: 0x3ba3d70a
In binary: 0 01110111 01000111101011100001010
Sign: +
Exponent: 2^-8
Significand: 1.280000
```

Here are some hints for approaching this programming challenge. Depending on how you decide to implement, you may not need all of them.

- To obtain the binary pattern of the input `float` value, try using the "`&`" and "`*`" operators and casting.
- Use `unsigned` rather than `int`.
- You may need to write code to convert integer values into binary. Think about writing it using the iterative algorithm that we used to convert decimal numbers into binary numbers a few weeks ago.
- Likewise, you may need to write code to convert integer values into hexadecimal. A similar approach will work here.
- To obtain the fractional part, you can just divide the integer value you get from applying your mask by the value $2^{23}$.
- The logical bit shift operators in C are "`<<`" to shift left and "`>>>`" to shift right. (No, that's not a typo.) For example, the instruction `6 >> 1` will shift the bit pattern for 6 (00...0110) to the left one position, yielding 12 (00...1100).
- The bit mask to find the sign of the number is 0x80000000. See if you can figure out the masks for the exponent and significand from there.

## Part 2: Bit Masking in MIPS (15 points)

Following the instructions above, write the same program with the same input/output using MIPS.

## Submission Details

For this assignment, your submission to your BitBucket repository should include the following:

1. A commented version of your bit masking C code.
2. A commented version of your bit masking MIPS code.

Before you turn in this assignment, you also must ensure that the course instructor has read access to your BitBucket repository that is named according to the convention `cs210f2016-<your user name>`.