Name: ______________________ CSCI 237

Practice Problems :: Bits and Ints

Please show your work to receive partial credit. Clearly indicate your final answer.

1. **Two's Complement:** For this question, please consider the following 12-bit **two's complement** representation of an integer **x**:

   x = 1001 1100 1011

   What is the decimal value of **x**?

   Since this is a 12 bit two's complement number, the leftmost bit will indicate the sign. The '1' implies this is a negative number. We can calculate the value of this number in one of two different ways.

   a.) Add all the bits together with the first bit's value being negatively weighted:

   $1\times(-2)^{11} + 1\times2^{8} + 1\times2^{7} + 1\times2^{6} + 1\times2^{3} + 1\times2^{1} + 1\times2^{0} =$

   $-2048 + 256 + 128 + 64 + 8 + 2 + 1 = -1589$

   b.) Figure out what positive number this represents by flipping all the bits and adding 1 before converting that unsigned number to its decimal representation and then insert a negative sign.

   ```
    x = 1001 1100 1011
   -x = 0110 0011 0101
   ```

   $1\times2^{10} + 1\times2^{9} + 1\times2^{5} + 1\times2^{4} + 1\times2^{2} + 1\times2^{0} = 1024 + 512+32+16+4+1=1589$

   What is the decimal value of **x** if we instead interpret it as a 12-bit unsigned number?

   We just use the equation for converting an unsigned binary number to decimal.

   $1\times2^{11} + 1\times2^{8} + 1\times2^{7} + 1\times2^{6} + 1\times2^{3} + 1\times2^{1} + 1\times2^{0} =$

   $2048 + 256 + 128 + 64 + 8 + 2 + 1 = 2507$

2. **Truncating:** Consider the primitive C types `short` and `char` on our lab's x86_64 systems. The type `short` is used to represent 16-bit two's complement numbers, and the type `char` is used to represent 8-bit two's complement numbers. What output is printed after executing the following C code snippet?

   ```
   short s = 0x58b;
   char c = (char) s;
   printf("%d", c);
   ```

   Hint: The `printf` format string character `%d` is used to print the decimal value of an integer type, including the primitive type `char`.

   Each hexadecimal digit is equivalent to 4 binary digits. Thus the binary representation is:

   0101 1000 1011

   When we cast a short to a char, we drop the leading byte of the short due to truncation. That means we are left with:

   1000 1011

   chars are considered signed, so we would interpret this byte as a twos complement representation. We would calculate its value in the following manner:

   $1\times(-2)^7 + 1\times2^3 + 1\times2^1 + 1\times2^0 = -128+8+2+1 = -117$

3. **Bit operations:** What does this C expression evaluate to? Please express your answer in hexadecimal.

   ```
   0xabbadeed & (0xff << 16);
   ```

   If we keep in mind that every hexadecimal digit corresponds to 4 binary digits, we see that 0xff takes up 8 binary digits. We are then shifting that value 16 binary digit positions left which is equivalent to 16/4 = 4 hexadecimal digits left. Thus, we get

   (0xff << 16) = 0xff0000

   When we do a bitwise and of this value with 0xabbadeed, only the bits in 0xabbadeed that align with non-zeros will be in our result:

   ```
        0xabbadeed
      & 0x00ff0000
    -------------
        0x00ba0000
   ```