CSCI210 Computer Architecture and Organization (aka: CS4)
Lab Assignment: Number Systems, memory addressing and some basic C

- Type your answers out in a document and submit as a PDF
- Don’t use a calculator; you’ll have to do this on a quiz without a calculator.
- DO use a calculator to check your work. You will get full credit for this lab simply by completing it. I WILL NOT go through and grade every single problem. You have the ability to check your work. DO IT.
- Be sure that you learn and understand the number system conversion algorithms

1. Write each of the following decimal numbers in binary:

    a. 3
    b. 9
    c. 15
    d. 18
    e. 7
    f. 10
    g. 5
    h. 16
    i. 22
    j. 29
    k. 37
    l. 64

2. Write each of the following binary numbers in decimal:

    a. 00100101
    b. 01001111
    c. 00010100
    d. 00010010
    e. 00011011
    f. 00001100
    g. 00110001
    h. 00101111
    i. 01001000
    j. 01100001
    k. 10100110
    l. 10110010

3. Write each of the following binary numbers in hexadeci-mal:

    a. 01100101
    b. 01001111
    c. 00010110
    d. 00010010
    e. 10001011
    f. 00111100
    g. 00110010
    h. 00100101
    i. 01001010
    j. 01101011
    k. 10100110
    l. 10111011

4. Write each of the following hexadecimal numbers in binary:

a.	0015h	g.	0050h
b.	00FFh	h.	0047h
c.	0110h	i.	0141h
d.	0018h	j.	0023h
e.	000Dh	k.	0064h
f.	003Ch	l.	ABCDh

5. Write each of the following hexadecimal numbers in decimal:

a.	00C5h	g.	0A30h
b.	001Fh	h.	06EAh
c.	0111h	i.	1CA3h
d.	0213h	j.	0B33h
e.	003Ch	k.	03B1h
f.	040Ch	l.	2DBAh

6. Add the following binary values together by hand. Convert each of the operands to decimal and check your answer. Assume 8 bit ***unsigned*** values. Note if there is any overflow.

```
 10101111
+11000011
```

```
 00111001
+11110011
```

```
 11111111
+11101110
```

7. Write each of the following ***8-bit signed*** binary integers in decimal:

a.	11111111	g.	00001111
b.	11110000	h.	10101111
c.	10000000	i.	11111100
d.	10000001	j.	01010101

8. Write each of the following signed decimal integers in ***8-bit*** binary notation: If any number cannot be represented as a signed 8-bit binary number, indicate this in your answer.

a.	-2	e.	+15
b.	-7	f.	-1
c.	-128	g.	-56
d.	-16	h.	+127

9. Perform binary subtraction (using 2s complement addition: negate the second operand and add) on the following values by hand. Show the answers in hex and decimal. Assume ***8 bit unsigned*** values.

   a. 0x2D - 0x1F

   b. 0xB0 - 0x7A

   c. 0xFF - 0x33

10. Add the following hexadecimal numbers together by hand. Convert each of the operands to decimal and check your answer. Assume 8 bit unsigned values. Note if there is any overflow. If there is any overflow write the incorrect answer.

    Divide the sum of the two digits by the base. The quotient becomes the carry value and the remainder is the sum digit.

```
       1
0x6A (A + B = 15h : 10 (A) + 11(B) = 21d)
0x4B
0xB5 (21 /16 = 1 remainder 5)
```

```
0x34  + 0x41 =
0x18  + 0x42 =
0xAC  + 0xCC =
0x2B  + 0xA7 =
```

11. Solve the following assuming ***byte addressable memory*** (each address points to a single byte)

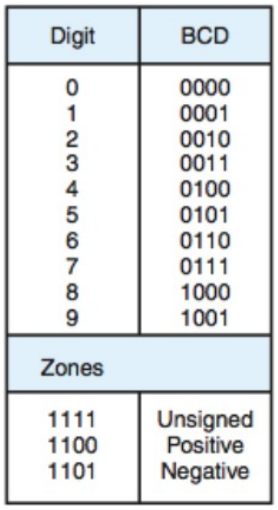
    a. You have a BYTE stored at address **0x1234**. What is the next available contiguous memory address?
    b. You have a 16 bit WORD stored at address **0xCAFE**. What is the next available contiguous memory address?
    c. You have a 32 bit ARM WORD stored at address **0xDEAD**. What is the next available contiguous memory address?
    d. Assume the address of var1 is **0x3E0** and address of var2 is **0x400**. How many bytes are used in var1?
    e. Address of var1 is **0x100** and its size is 64 bits. Var2 immediately follows var1, what is the offset of Var2 (in hex)?

12. Design your own numbering system. Decide upon a base and give it a name. (Don't use common number bases like 3, 4, 8, 12, 16) Show 3 examples of each of the following.

    a) Conversion to decimal
    b) Conversion from decimal
    c) Conversion to binary
    d) Conversion from binary
    e) Addition of two numbers

13. Binary coded decimal (BCD) is a numeric coding system used primarily in IBM mainframe and midrange systems. As its name implies, BCD encodes each digit of a decimal number to a 4 bit binary form. When stored in an 8 bit byte, the upper nibble is called the zone and the lower part is called the digit. (This convention comes to us from the days of punched cards where each column of the card could have a "zone punch" in one of the top 2 rows and a "digit punch" in one of the 10 bottom rows.) The high-order nibble in a BCD byte is used to hold the sign, which can have one of three values:

    a. An unsigned number is indicated with 1111
    b. A positive number is indicated with 1100
    c. A negative number is indicated with 1101.



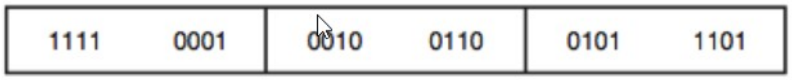
Digit	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
Zones	
1111	Unsigned
1100	Positive
1101	Negative

The digits of BCD numbers occupy only one nibble, so we can save on space and make computations simpler when adjacent digits are placed into adjacent nibbles, leaving one nibble for the sign. This process is known as packing and numbers thus stored are called packed decimal numbers.

Example: Represent -1265 in 3 bytes using packed BCD. The zoned decimal coding for 1265 is:

1111 0001 1111 0010 1111 0110 1111 0101

After packing, this string becomes: 0001 0010 0110 0101
Adding the sign after the low-order digit and padding the high-order digit with ones in 3 bytes we have:



1111 0001	0010 0110	0101 1101

Application:

You have stumbled on an unknown civilization while sailing around the world. The people, who call themselves Zebronians, do math using 40 separate characters (probably because there are 40 stripes on a zebra). They would very much like to use computers, but would need a computer to do Zebronian math, which would mean a computer that could represent all 40 characters. You are a computer designer and decide to help them. You decide the best thing is to use BCZ, Binary Coded Zebronian (which is like BCD except it codes Zebronian, not Decimal). How many bits will you need to represent each character if you want to use the ***minimum number of bits***? Do not worry about padding to 8 bits, just calculate the minimum number of actual bits.

C Programming Exercises: Be sure to review the provided examples (on Brightspace)

1) Write a C program to accomplish the following

- Count and display the frequency of individual numbers contained within an array of ints.
- Count and display the number of unique values in the array

if the array is {1, 2, 3, 3, 1, 3, 2, 1, 1, 2, 3, 3}

The output will be

```
1s = 4
2s = 3
3s = 5

There are 3 unique numbers
```

Requirements For Full Credit:
=============================

- This must be accomplished in ***O(N)time*** and you are allowed up to ***O(N)memory***
- You are required to use pointer notation just to practice with actual memory address math.
- While if statements are not specifically forbidden, the counting can be accomplished WITHOUT decisions. Do this. Huge if blocks will be egregiously penalized
- Define a macro for SIZE. I will be changing this value when I grade
- The array should be of SIZE elements. REMEMBER, I will be changing this when I grade it. It must respond accordingly

Fill the array with random numbers in the range 1 to ***(SIZE / 4)***. This ensures decent probability for duplicates. See the provided random number example

I may grade this with a SIZE of 10 or a SIZE of 10,000 . . . or anywhere in between

2) Explore the concept of byte addressable memory

This exercise will help you understand how memory is allocated and accessed in C using **arrays and pointers** of different primitive types (char, int, float, double). You will also explore **pointer arithmetic** and how different data types affect memory layout.

- In a new C file declare small arrays of different primitive types: char, int, float and double
- Declare pointers to the first elements
- Print the **beginning address of each array**
- Using the pointers, compute the difference between each contiguous array and display the values
- Use a loop to print the memory address and value of each element in each of the arrays. Also show the **difference** between each address
- **Reflection:** In comments address the following
    - What patterns did you notice in how different data types are stored in memory? How does pointer arithmetic reflect these patterns?
    - Why does incrementing a pointer move by different byte sizes depending on the data type? How does this impact working with arrays?

Submission:
===========
Attach all files to the Brightspace assignment.