Solution for Homework 2

Problem 1

a. ***What is the minimum number of bits that are required to uniquely represent the characters of English alphabet? (Consider upper case characters alone)***

The number of unique bit patterns using *i* bits is 2^i. We need at least 26 unique bit patterns. The cleanest approach is to compute $\log_2 26$ and take the ceiling (round up). This yields **5** as the answer. Trial and error is also an acceptable solution.

b. How many more characters can be uniquely represented without requiring additional bits?

With 5 bits, we can represent up to 32 (2^5) unique bit patterns; we can represent
32 - 26 = **6** more characters without requiring additional bits.

Problem 2

Using 7 bits to represent each number, write the representations of 23 and -23 in signed magnitude and 2's complement integers

	Signed Magnitude	**1's Complement**	**2's Complement**
23	0010111	0010111	0010111
-23	1010111	1101000	1101001

Problem 3

a. What is the largest positive number one can represent in a 12-bit 2's complement code? Write your result in binary and decimal.

In Binary: 011111111111
In Decimal: 2047 (Obtained by converting 011111111111 to decimal)
(1024+ 512+ 256+ 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1)

b. What is the greatest magnitude negative number one can represent in a 12-bit 2's complement code? Write your result in binary and decimal.

In Binary: 100000000000
In Decimal: -2048 (Obtained by converting 100000000000 to decimal) →
- (Not (100000000000) + 1) = -(011111111111 + 1) = -2048

c. ***What is the largest positive number one can represent in n-bit 2's complement code?***

$2^{n-1} - 1$

d. ***What is the greatest magnitude negative number one can represent in n-bit 2's complement code?***

-2^{n-1}

Problem 4

What are the 8-bit patterns used to represent each of the characters in the string "CS/ECE 252"? (Only represent the characters between the quotation marks.)
Note: There is space between ECE and 252.

From the ASCII Table,

Character	**Hexadecimal Equivalent**	**Binary Equivalent**
C	0x43	01000011
S	0x53	01010011
/	0x 2F	00101111
E	0x45	01000101
C	0x43	01000011
E	0x45	01000101
sp	0x20	00100000
2	0x32	00110010
5	0x35	00110101
2	0x32	00110010

Hence the string ***"CS/ECE 252"*** is represented as
0100001101010011001011110100010101000011
0100010100100000001100100011010100110010

Problem 5

Convert the following 2's complement binary numbers to decimal.

a. ***0110*** **→**

(0*8 + 1*4 + 1*2 + 0*1) = 6

b. ***1101*** **→**

- [NOT (1101) + 1] = - (0010 +1) = -3

c. ***01101111*** **→**

[0*128 + 1*64 + 1*32 + 0*16 + 1*8 + 1*4 + 1*2 + 1*1] = 111

d. ***1101101100011100*** **→**

- [**NOT** (1101101100011100) + 1] = - (0010010011100011 + 1) = -9444

Problem 6

Express the negative value -22 as a 2's complement integer, using eight bits. Repeat it for 16 bits and 32 bits. What does this illustrate with respect to the properties of sign extension as they pertain to 2's complement representation?

- **8 bit**

The 8-bit binary representation of 22 is 00010110.

So, -22 in 2's complement form is (NOT (00010110) + 1) = (11101001 + 1) = 11101010

- **16 bit**

The 16-bit binary representation of 22 is 00000000 00010110.

So, -22 in 2's complement form is (NOT (00000000 00010110) + 1) = (11111111 11101001 + 1) = 11111111 11101010

- **32 bit**

The 8-bit binary representation of 22 is 00000000 00000000 00000000 00010110.

So, -22 in 2's complement form is (NOT (00000000 00000000 00000000 00010110) + 1) = (11111111 11111111 11111111 11101001 + 1) = 11111111 11111111 11111111 11101010

The pattern that we see is that when we increase the number of bits in the 2's complement representation, we *sign extend*, meaning that we take the leftmost bit of the original representation and duplicate it into the added bits of the new representation.

Problem 7

Write the decimal equivalents for these IEEE floating point numbers.

a. ***0 01111111 00000000000000000000000***

$(-1)*0 * 1.0*2^{127-127} = (1.0)_2 = (1)_{10}$

b. ***1 01111110 10000000000000000000000***

$(-1)*1 * 1.1*2^{126-127} = (-0.11)_2 = (-0.75)_{10}$

Problem 8

Describe what conditions indicate overflow has occurred when two 2's complement numbers are added.

- When adding two numbers, overflow occurs when the two operands have the same leftmost bit and the leftmost bit of the answer is different.
- When the operands have differing leftmost bits, overflow cannot occur when adding them together because we are adding a positive number with a negative number which means we are actually subtracting. It implies that the result cannot be bigger than the operands. So there is no possibility of overflow in this case. The leftmost bit is frequently referred to as the Most Significant Bit (MSB for short).

Problem 9

De Morgan's Laws:

i. ***NOT (P AND Q) = (NOT P) OR (NOT Q)***

ii. ***NOT (P OR Q) = (NOT P) AND (NOT Q)***

Verify these for P = 1011 and Q = 1101

NOT (P) = NOT (1011) = 0100
NOT (Q) = NOT (1101) = 0010

i. (NOT P) OR (NOT Q) = 0110
NOT (1011 AND 1101) = NOT (1001) = 0110

⇨ NOT (P AND Q) = (NOT P) OR (NOT Q)

ii. NOT (P OR Q) = NOT (1011 OR 1101) = NOT (11110 = 0000

NOT (P) AND NOT (Q) = 0100 AND 0010 = 0000

NOT (P OR Q) = (NOT P) AND (NOT Q)

Problem 10

The following binary numbers are 4-bit 2's complement binary numbers. Which of the following operations generate overflow? Justify your answers by translating the operands and results into decimal.

a. 0011 + 1100

Adding in decimal equivalent:

3 + -[NOT(1100)+1] = 3 –(0011+1)$_2$= $(3\text{-}4)_{10}$ = -1_{10}

Adding in 2's complement:

0011 + 1100 = $(1111)_2$ = -[NOT(1111)+1] = -1_{10} **(No Overflow)**

b. 0111 + 1111

Adding in decimal equivalent:

7 + -[NOT(1111)+1] = 7 -1 = 6_{10}

Adding in 2's complement:

0111 + 1111 = $(0110)_2$ = 6_{10} **(No Overflow)**

c. 1110 + 1000

Adding in decimal equivalent:

-[NOT(1110) + 1] – [NOT(1000) + 1] = $-(10)_2$ – $(1000)_2$ = -2 + - 8 = -10

Adding in 2's complement:

1110 + 1000 = 0110 = 6_{10} **(Overflow!)**

d. 0110 + 0010

Adding in decimal equivalent:

6 + 2 = 8

Adding in 2's complement:

1 000 = - [NOT(1000) + 1] = -(0111+1) = -(1000) = -8 **(Overflow)**