CSE 370 Fall 1999
Solution for Assignment 1
10/04/99

1. Does not require a solution

2. Ditto

3.

a)

$15C.38_{16} = (1 \times 16^2 + 5 \times 16^1 + 12 \times 16^0 + 3 \times 16^{-1} + 8 \times 16^{-2})_{10}$
$= (1 \times 256 + 5 \times 16 + 12 \times 1 + 3 \times 0.0625 + 8 \times 0.00390625)_{10}$
$= (256 + 80 + 12 + 0.1875 + 0.03125)_{10}$
$= \mathbf{348.21875_{10}}$

b)

First, let's do the integer part:
$37_{10} = 2 \times 18_{10} + 1$
$18_{10} = 2 \times 9_{10} + 0$
$9_{10} = 2 \times 4_{10} + 1$
$4_{10} = 100_2$ (common knowledge, although you could get this by continuing)

Thus, $37_{10} = 100101_2$

Now, let's do the fractional part:

$0.24_{10} \times 2 = 0.48_{10} \Rightarrow 0$
$0.48_{10} \times 2 = 0.96_{10} \Rightarrow 0$
$0.96_{10} \times 2 = 1.92_{10} \Rightarrow 1$
$0.92_{10} \times 2 = 1.84_{10} \Rightarrow 1$
$0.84_{10} \times 2 = 1.68_{10} \Rightarrow 1$
$0.68_{10} \times 2 = 1.36_{10} \Rightarrow 1$
$0.36_{10} \times 2 = 0.72_{10} \Rightarrow 0$
$0.72_{10} \times 2 = 1.44_{10} \Rightarrow 1$
$0.44_{10} \times 2 = 0.88_{10} \Rightarrow 0$
$0.88_{10} \times 2 = 1.76_{10} \Rightarrow 1$

Now, read the bits from top to bottom, so:
$0.24_{10} = 0.0011110101_2$

We can now add the integer and the fractional results, so:
$\mathbf{37.24_{10} = 100101.0011110101..._2}$

4.

A few notes before we start. We only have four bits, so let's compute the range of numbers we can represent in each one of the cases:

- **Unsigned**: 0000_2 to 1111_2, so range is 0 to 15
- **Sign-Mag**: Max magnitude is $111_2 = 7$, so range is -7 to +7
- **1's complement**: Max is $0111_2 = 7$ since 1000 represents -7. So range is -7 to +7.

- **2’s complement**: Max is 0111_2 = 7 since 1000 represents -8. So range is -8 to +7.
- **FP**: Can’t represent -tive numbers since we don’t have a sign bit, so min is 0. Max is 1111, which is 0.11_2 x 2^{11_2} = 0.11_2 x 2^3 = 110_2 = 6. So the range is 0 to 6.
- **BCD**: We only have four bits, and that’s only one decimal digit, so the range is 0 to 9.

a)

```
  1000
x  -1
-----
overflow
```

- **Unsigned**: 1000 is 8, 8 x -1 = -8, that’s overflow, so we **have** an X.
- **Sign-Mag**: 1000 is -0, -0 x -1 = 0, that’s 0000 in sign-mag, so we **don’t have** an X.
- **1’s comp**: 1000 is -7, -7 x -1 = 7, that’s 0111 in 1’s comp, so we **don’t have** an X.
- **2’s comp**: 1000 is -8, -8 x -1 = 8, that’s overflow in 2’s comp, so we **have** an X.
- **FP**: 1000 is 0.10_2 x 2^{00_2} = 0.10_2, 0.10_2 x -1 = -0.10_2, that’s overflow, so we **have** an X.
- **BCD**: 1000 is 8, 8 x -1 = -8, that’s overflow, so we **have** an X

b)

```
  1000
+   1
-----
 0001
```

- **Unsigned**: 1000 is 8, 8 + 1 = 9, that’s 1001, so we **don’t have** an X.
- **Sign-Mag**: 1000 is -0, -0 + 1 = 1, that’s 0001 in sign-mag, so we **have** an X.
- **1’s comp**: 1000 is -7, -7 + 1 = -6, that’s 1001 in 1’s comp, so we **don’t have** an X.
- **2’s comp**: 1000 is -8, -8 + 1 = -7, that’s 1001 in 2’s comp, so we **don’t have** an X.
- **FP**: 1000 is 0.10_2 x 2^{00_2} = 0.10_2, 0.10_2 + 1 = 1.10_2 = 0.11_2 x 2^{01_2}, that’s 1101 in FP, so we **don’t have** an X.
- **BCD**: 1000 is 8, 8 + 1 = 9, that’s 1001 in BCD, so we **don’t have** an X.

c)

```
 1000
x   2
-----
 1001
```

- **Unsigned**: 1000 is 8, 8 x 2 = 16, that’s overflow, so we **don’t have** an X.
- **Sign-Mag**: 1000 is -0, -0 x 2 = 0, that’s 0000 or 1000 in sign-mag, so we **don’t have** an X.
- **1’s comp**: 1000 is -7, -7 x 2 = -14, that’s overflow, so we **don’t have** an X
- **2’s comp**: 1000 is -8, -8 x 2 = -16, that’s overflow, so we **don’t have** an X
- **FP**: 1000 is 0.10_2 x 2^{00_2} = 0.10_2, 0.10_2 x 2 = 1.0_2 = 0.10_2 x 2^{01_2}, which is 1001, so we **have** an X.
- **BCD**: 1000 is 8, 8 x 2 = 16, that’s overflow, so we **don’t have** an X.

d)

```
 1000
+   2
-----
```

overflow

- **Unsigned**: 1000 is 8, 8 + 2 = 10, that’s 1010, so we **don’t have** an X.
- **Sign-Mag**: 1000 is -0, -0 + 2 = 2, that’s 0010 in sign-mag, so we **don’t have** an X.
- **1’s comp**: 1000 is -7, -7 + 2 = -5, that’s 1010 in 1’s comp, so we **don’t have** an X.
- **2’s comp**: 1000 is -8, -8 + 2 = -6, that’s 1010 in 2’s comp, so we **don’t have** an X.
- **FP**: 1000 is 0.10_2 x $2^{00_2} = 0.10_2$, $0.10_2 + 2 = 10.10_2 = 0.101_2$ x 2^{10_2}, that’s approximately 1010 in FP, so we **don’t have** an X. Note that we don’t have overflow here. We are just loosing precision that's all. That's different from actually going outside of the range that the number system can handle.
- **BCD**: 1000 is 8, 8 + 2 = 10, that's overflow in BCD, so we **have** an X.

e)

```
 1000
+   1
-----
 1001
```

- **Unsigned**: 1000 is 8, 8 + 1 = 9, that's 1001, so we **have** an X.
- **Sign-Mag**: 1000 is -0, -0 + 1 = 1, that's 0001 in sign-mag, so we **don’t have** an X.
- **1’s comp**: 1000 is -7, -7 + 1 = -6, that's 1001 in 1's comp, so we **have** an X.
- **2’s comp**: 1000 is -8, -8 + 1 = -7, that's 1001 in 2's comp, so we **have** an X.
- **FP**: 1000 is 0.10_2 x $2^{00_2} = 0.10_2$, $0.10_2 + 1 = 1.10_2 = 0.11_2$ x 2^{01_2}, that's 1101 in FP, so we **don’t have** an X.
- **BCD**: 1000 is 8, 8 + 1 = 9, that's 1001 in BCD, so we **have** an X.

So, the table looks like this:

	Unsigned	Sign-Mag	1’s Comp	2’s Comp	FP	BCD
a	**X**			**X**	**X**	**X**
b		**X**				
c					**X**	
d						**X**
e	**X**		**X**	**X**		**X**

5
a)

Unsigned

```
  101010
+ 010110
--------
 1000000
```

This is **overflow**, since we can't express this in 6 bits. Note that unlike in the 1's and 2's complement cases, where a carry is not necessarily overflow, in unsigned, a carry ALWAYS indicates overflow.

Sign-Magnitude

101010 is a negative number whose magnitude is 01010
010110 is a positive number whose magnitude is 10110

Since 10110 is bigger than 01010, the sign of the result will be positive, and the magnitude of the result will be 10110 - 01010.

```
  10110
- 01010
-------
  01100
```

So, C is a positive number whose magnitude is 01100, thus **C = 001100**

1’s complement

```
  101010
+ 010110
--------
 1000000
+      1
--------
 1000001
```

We can now ignore the carry, so **C = 000001**. There is no overflow because we added a negative number with a positive one, and got a positive result, which is very normal.

2’s complement

```
  101010
+ 010110
--------
 1000000
```

We can now ignore the carry, so **C = 000000**. Again, there is no overflow for the same reason as in the 1’s complement case.

b)

We convert from 6-bit 2’s complement to 7-bit 2’s complement by finding what the value of the 6-bit bit stream and then convert that value into 7-bit 2’s complement notations.

Let’s start with A.

A = 101010. This is a negative number because the msb (most significant bit) is 1. The magnitude of A is therefore $010101_2 + 1 = 010110_2$ (got this by inverting the bits of A and then adding 1). Thus, the value of A is -010110_2.

To represent A in 7-bit 2’s complement, we need to represent 010110_2 in 7 bits, and then negate it (which in 2’s complement is inverting the bits, and adding 1). So:

Represent in 7 bits: 0010110
Invert: 1101001
Add 1: 1101010

Thus $\mathbf{A_{7bit}}$ **= 1101010**

Let’s do B.
B = 010110. This is a positive number because the msb is 0. So the value of B is 010110_2.

To represent B in 7-bit 2's complement, we just need to represent 010110_2 in 7 bits, which is 0010110.

Thus $\mathbf{B_{7bit} = 0010110}$

Note that if you follow the following rule, you get the same results:
"To increase the number of bits in 2's complement, just pad the bit stream to the left by inserting copies of the msb from the original number until you get the desired number of bits."

This is called sign extension, and that's the way that computers convert 2's complement numbers from smaller to larger number of bits. For example, if you have a 32-bit integer and you add it to a 64-bit integer, the 32-bit integer is first sign-extended using the above mechanism to 64-bits before the addition is done.

Now, we compute B-A

```
  10010110
 - 1101010
 ---------
  00101100
```

Thus, **D = 0101100.** Note that we had to pad B to the left with an additional 1 so that the subtraction works. The problem if we don't do this is that we get into a situation where we need to borrow a one to do the subtraction, but we don't have where to borrow it from. So, what we do is just add an extra 1, which we borrow. This is essentially the opposite of "ignore the carry": we are adding the extra 1 to make the bit-wise subtraction work.

6.

We first need to get the two numbers to the same exponent.
A = 0.1101_2 x 2 ^ 0100_2

B = 0.1111_2 x 2 ^ 0111_2
  = 111.1_2 x 2 ^ 0100_2

Since we now have numbers with the same exponent, we can add:

```
     0.1101₂
+  111.1₂
------------
+ 1000.0101₂
```

Thus, the result is 1000.0101_2 x 2 ^ 0100_2 = 0.10000101_2 x 2^{1000_2}
In the given floating point format, the result would be **C = 10001000**. Notice that we lost some precision.