Recitation #2

1. A)
$$\begin{array}{r} 0011 \\ +\,0010 \\ \hline 0101 \end{array}$$

B) $0010 - 0011$

$((\sim 0011) + 1) + 0010 \Rightarrow (1100 + 1) + 0010$

$$\begin{array}{r} 1101 \\ +\,0010 \\ \hline 1111 \end{array}$$

C)
$$\begin{array}{rcc} 0011 & & \\ \times\,0010 & & \\ \hline 0000 & 0 & 0 \\ 0011\leftarrow & 1 & 1 \\ 0000\leftarrow\leftarrow & 0 & 0 \\ +\,0000\leftarrow\leftarrow\leftarrow & 0 & 0 \\ \hline 000|0110 & & \end{array}$$

D)
$$\begin{array}{rcc} 0011 & & \\ \div\,0010 & & \\ \hline 0000 & 0 & 0 \\ \rightarrow 0011 & 1 & 1 \\ \longrightarrow 00|00 & 0 & 0 \\ +\longrightarrow 0|000 & 0 & 0 \\ \hline 0001\,R\,1 & & \end{array}$$

2. A)
```
int compl(int n) {
    int b = 0xF;
    return ~(b << n);
}
```

B)
```
int comph(int n){
    int b = 0xF;
    return (b << (4-n));
}
```

Recitation #3

Bias $= 2^{7-1}-1 \Rightarrow 2^6-1 = 64-1 = 63$

12.86)

Desc	Hex	M	E (with Bias)	V	Binary
-0	8000	0	-62	-0	1 0000000 00000000
smallest >2	4001	$\frac{257}{256}$	1	2.0039	0 1000000 00000001
512	4800	1	9	512	0 1001000 00000000
largest denorm	00FF	$\frac{127}{128}$	-63	1.9921	0 0000000 11111111
$-\infty$	FF00	0	64	∞	1 1111111 00000000
0x3BB0	3BB0	$\frac{27}{16}$	-4	$\frac{27}{256}$	0 0111011 1011 0000

Sign | Exp | Frac

12.87)

Bias $= 2^{5-1}-1 = 15$

1 01111 001	
$-$ 15-15 1.001 $\Rightarrow -\frac{9}{8}$	
0	
0 10110 011	
$+$ 22-15 1.011 $\Rightarrow 176$	
7	
1 00111 010	
$-$ 7-15 1.010 $\Rightarrow -\frac{5}{1024}$	
-8	
0 00000 111	
$+$ 0-15+1 0.111 $\Rightarrow \frac{7}{2^{14}}$	
-14	
1 11100 000	
$-$ 28-15 1.000 $\Rightarrow -8192$	
13	
0 10111 100	
$+$ 23-15 1.100 $\Rightarrow 384$	
8	

Bias $= 2^{4-1}-1 = 7$

1 0111 0010	
$-$ 7-7 1.0010 $\Rightarrow -\frac{9}{8}$	
0	
0 1110 0110	
$+$ 14-7 1.0110 $\Rightarrow 176$	
7	
1 0000 0101	
$-$ 0-7+1 0.0101 $\Rightarrow -\frac{5}{1024}$	
-6	
0 0000 0001	OR 0 0000 0000 $\Rightarrow 0$
$+$ 0-7+1 0.0001 $\Rightarrow \frac{1}{2048}$	
-6	
1 1110 1111	OR 1 1111 0000 $\Rightarrow -\infty$
$-$ 14-7 1.1111 $\Rightarrow -248$	
7	
0 1111 0000 $\Rightarrow \infty$	
$+$	

Recitation #4

2.88) A) This would evaluate to TRUE.
$(float)x == (float)dx$ Doubles have a greater precision than float, so it might have to be truncated, but it should truncate to the same value.

B) This would NOT evaluate to TRUE always.
$dx - dy == (double)(x-y)$ The ints might overflow before being cast to a double in $(x-y)$ while the left side would not overflow with the same numbers. Their results would differ.

C) This would NOT evaluate to TRUE always.
$(dx+dy)+dz == dx+(dy+dz)$ IF $dx = -dy$, and dx is a very large number while dz is a very small number we could end up with the comparison: $dz == 0$ due to rounding.

D) This would NOT evaluate to TRUE always
$(dx*dz)*dz == dx*(dy*dz)$ This is a similar truncation / lose of precision problem as before.

E) This would NOT evaluate to TRUE always.
$dx/dx == dz/dz$ If either dx or dz is 0 we get a comparison between NaN and a number.

2) 3.1415

3/2 ⇒ 1 R 1	0.1415 * 2 ⇒	0.2830
1/2 ⇒ 0 R 1	0.283 * 2 ⇒	0.566
3 ⇒ 11	0.566 * 2 ⇒	1.132
	0.132 * 2 ⇒	0.264
	0.264 * 2 ⇒	0.528
	0.528 * 2 ⇒	1.056
	...	↳ 0.1415 ≈ 0.001001

$s = 0$

$\oplus$ 3.1415 ∝ 11.001001 Bias $= 2^{4-1} - 1 \Rightarrow 7$ $1 = e - 7 \Rightarrow e = 8$

Exp = 1

$1.100\,1001 \times 2^1$

0	1000	100

Recitation #5

Bias $= 2^{8-1} - 1 = 127$

1) 3.1415

$3/2 \Rightarrow 1\ R\ 1$
$1/2 \Rightarrow 0\ R\ 1$
$\hookrightarrow 3 = 11$

$0.1415 * 2 \Rightarrow 0.2830$
$0.283 * 2 \Rightarrow 0.566$
$0.566 * 2 \Rightarrow 1.132$
$0.132 * 2 \Rightarrow 0.264$
$0.264 * 2 \Rightarrow 0.528$
$0.528 * 2 \Rightarrow 1.056$
...
$\hookrightarrow$ The values we want

3.1415 $\approx$ 0 10000000 1001 0010 0001 1100 1010 110

2) Through similar means as #1:

0 10011011 0001 1101 1110 0111 1000 010

3) Exp from #1 = 2 while Exp from #2 = 155 - 128 = 27. 27 - 2 = 25, and we only have 23 (24 with implied 1) bits to represent our float. We completely lose 3.1415 to 299792458.

0 10011011 0001 1101 1110 0111 1000 010

4) As explained in #3

5) No, a double has 53 bits to represent numbers, so we could fit both 3.1415 and 299792458 into it without losing presition.

Recitation #6

3.64) A) 8 off %ebp is a pointer to where the function is to fill in its results
12 off %ebp is s1.a
16 off %ebp is s1.p

B) The first field points to the location for the returned structure
The next two fields are s1.a and s1.p
The next two fields are where the result of word_sum is stored
(both s2.sum and s2.diff)

C) The general strategy is to pass the argument structure on the stack, just as any argument is passed. The callee accesses the fields with offsets relative to %ebp.

D) The general strategy is for the caller to allocate space in its own stack for the result structure, and then it passes a pointer to this structure as a hidden first argument to the function.

3.65) Create a system of equations with the knowledge we obtain from the assembly code:

$$\left.\begin{aligned} B + e_1 &= 12 \\ B + e_1 + 4 + 2B + e_2 &= 36 \\ 2AB + e_3 &= 92 \end{aligned}\right\}$$ Where e_1, e_2, and e_3 are potential places of padding

We also know:

$$e_1 \in \{0, 1, 2, 3\}$$
$$e_2, e_3 \in \{0, 2\}$$

And that:

$$B \in \{9, 10, 11, 12\}$$

If we rewrite eq #2:

$$3B + e_1 + e_2 = 32$$

We can rule out $B = 11$ and $B = 12$.
Plugging into the other equations we see that the only answer that meets all of our constraints is:

$$\boxed{A = 5 \qquad B = 9}$$