Kelvin Abrokwa-Johnson
14 March 2016
CS 304

Homework 1

1.

a) The code fails to comply with the C standard because it attempts to shift beyond the word size. In many systems the shift amount will be applied *mod* the word size so an attempt to shift by 32 with result in a $32 \mod 32 = 0$ shift.

b)

```
int int_size_is_32() {
        int set_msb = 1 << 31;
        int beyond_msb = 2 << 31;
        return set_msb && !beyond_msb;
}
```

c)

```
int int_size_is_32() {
        int i = 1 << 15;
        i = i << 15; // 0 if less than 32 bit
        int j = 4  << 15;
        j = j << 15;
        return i && !j;
}
```

2.

a) The code is incorrect because it doesn't perform the appropriate sign extension.

b)

```
int xbyte(packed_t word, int bytenum) {
        word = word << ((3 - bytenum) << 3);
        return word >> 24;
}
```

3.

a) The conditional test always succeeds because *sizeof*() returns type *size_t* which is an unsigned data type. So the subtraction will result in an unsigned number which will always be less than or equal to 0.

b)

```
void copy_int(int val, void *buf, int maxbytes) {
        // assumes sizeof(size_t) <= sizeof(int)
        if (maxbytes >= (int)sizeof(val))
                memcpy(buf, (void *) &val, sizeof(val));
}
```

4.

a) $(x << 4) + x$

b) $x - (x << 3)$

c) $(x << 6) - (x << 2)$

d) $(x << 4) - (x << 7)$

5.

a) $\sim 0 << k$

b)

```
int create_val(int i, int j) {
        r = 0;
        for (int i = j + 1; i =< j + k; i++) {
                r += 1 << i;
        }
        return i;
}
```

6.

a) $(x < y) == (-x > -y)$:
When $x = TMin_w$ the first condition will evaluate to true and the second will be false. So this will return 0.

b) $((x + y) << 4) + y - x == 17 * y + 15 * x$:
We know that a bit shift is equivalent to multiplication by powers of 2 and since multiplication is distributive $((x + y) << 4) == (x << 4) + (y << 4)$. So we can replace the left side of the equality with $(y << 4) + y + (x << 4) - x$. On the right side we can rewrite the multiplications as sums of powers of 2. So $17 * y$ becomes $)y << 4) + y$ and $15 * x$ becomes $x << 4 - x$. We now see that the left side if equivalent to the right side.

c) $\sim x + \sim y + 1 == \sim (x + y)$:
When $x = y = \sim 0$ (all bits set to 1), the left side of the expression $\sim x + \sim y + 1$ evaluates to 1 but the left side of the equation $\sim (x + y)$ evaluates to ~ 0 (all bits set to 1).

d) $(ux - uy) == -(unsigned)(y - x)$:
This is true because the casting does not affect th order of the bits just how the numbers are treated

e) $((x >> 2) << 2) <= x$:
In any case (where the most significant bit of x is 0 or 1) when we shift left and then right we will fill in 0's on the left side and any added bits will overflow on the right side. So this expression is always true.

7.

description	Hex	M	E	V	D
-0	$0x8000$	0	-14	0	-0.0
smallest value > 2	$0x8$	1	2	$2 + \frac{1}{2^{10}}$	$2 + \frac{1}{2^{10}}$
512	$0x7C00$	0	31	512	512.0
largest denormalized	$0x3FF$	1023	-14	0.999023	0.9990234375
$-\infty$	$0xFF0000$	-	-	$-\infty$	$-\infty$
number with hex $0x3BB0$	$0x3BB0$	1110110000	14	14.921875	14.921875

8.

a) $(float)x == (float)dx$:
True because the casting does not affect the bits or get rid of them in this case

b) $dx - dy == (double)(x - y)$:
False when $y = 0$

c) $(dx + dy) + dz == dx + (dy + dz)$:
True because Abelian groups have associative addition in general

d) $(dx * dy) * dz == dx * (dy * dz)$:
False when $x = y = z = INT\_MAX$

e) $dx/dx == dz/dz$:
This does not work in the case of 0 division

9.

```
float fpwr2(int x)
{
        /* Result exponent and fraction */
        unsigned exp, frac;
        unsigned u;
        if (x < -23){
                /* Too small. Return 0.0 */
                exp = 0;
                frac = 0;
        } else if (x < 0){
                /* Denormalized result */
                exp = 0;
                frac = 1 << x;
        } else if (x < 23){
                /* Normalized result. */
                exp = 1 < x;
                frac = 0;
        } else {
                /* Too big. Return +oo */
                exp = 0xFF;
                frac = 0;
        }
        /* Pack exp and frac into 32 bits */
        u = exp << 23 | frac;
        /* Return as float */
        return u2f(u);
}
```

10.

a) fractional binary of $0x40490FDB = 01000000010010010000111111011011$

b) fractional binary of $\frac{22}{7} = 01000000010010010010010010010010$

c) They diverge at the 17^{th} digit