Ali Mirabzadeh

CS 33

HW1

2.71

a. The function should return unsigned char whereas it is returning int.

Also, it doesn’t check for the range of bytenum to be 0-3 as it is required.

b.

int xbyte(packed\_t word, int bytenum)

{

int extractByte;

// (3-bytenum) << 3: to move to the most-significant byte

extractByte = word << ((3-bytenum)<<3)

return extractByte >> 24 // extend it to 32-bit signed int

}

2.82

1. (x<y) == (-x>-y)

**False**. If we have x and y both be zero the above condition doesn’t yield to 1.

1. ((x+y)«4) + y-x == l7 \* y + 15\*x

**This always yields to 1** since (x+y)«4 is equal to 16x +16 y so 16x + 16y +y -x = 17y +15x which is the right hand side

1. ~x+~y+1 == ~(x+y)

We know for every number that x +~x =-1 or ~x = -1 -x

=> ~x + ~y +1 = (-1 -x) + (-1-y) +1 = -1 –(x+y) = ~(x+y)

**Therefore, it always yields to 1**

1. (ux-uy) ==-(unsigned) (y-x)

**This yields to 1** since the left-hand side is an arithmetic operation on an unsigned and then right-hand sign is casted to unsigned so the result of both sides would be the same.

1. ((x » 2) « 2) <= x

Since we are shifting two bits right and left, we only need to be worried about the first and last two bits. If those first and last two bits are not 1s then LHS is the same as the RHS. Now, in the case where the 2 most significant bits are 1, then the shifts will result in the same expression as well because it is an arithmetic right shift. Any other case will result in a lower number than the original because we either drop of some of the more significant bits (or least significant bits) or add more 1's at the left most end of the bit in which it will result in a more negative number; hence, the LHS is smaller than RHS.

**This always yields to 1.**