Homework 1

Leslie Liang

204625818

2.71

A) This code only returns the byte that we want as indicated by bytenum. However, it doesn't actually sign extend that byte to a 32 int. This is clearly noticed by the fact that you return some bit pattern and you &'ed it with 0xFF.

B) One correct solution will be to extend my wanted bytenum all the way to the right (this essentially means that it will destroy all of the other bits to the left of it). Finally, we would want to sign extend this guy so we would have to shift right by 24(24 because we shifted all the way to the left, and I need to shift back by 24 to get rid of the bits on my right).

The following code would do the trick:

int xbyte( packed\_t word, int bytenum)

{

int foo = word << ( (3-bytenum) << 3); //(3-bytenum would give me the

correct ammount of shift to the left)

return (foo >> 24); //essentially all the way to the right which destroy the bits on the left and sign extends to 32 bit

}

2.82

For simplicity, I would do everything in w=4 bits. This would extend to 32 bit case.

A) (x < y) == (-x > -y)

This x is Tmin, this expression would be false.

-8 < -7 ?= -8 > -7

If we were to negate Tmin, we would get Tmin, and Tmin is obviously less than any other number.

B) ((x+y) << 4) +y-x == 17\*y + 15\*x

This expression is true because if we were to expand the left hand side, it would be equal to the right hand side.

(x+y)\*16 + y -x = 16x +16y +y -x = 15x + 17y == 15x + 17y (RHS)

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

-x-1 + (-y-1) = -x-y-1 (LHS)

~(x+y) == -(x+y) -1 = -x-y-1 == -x-y-1 (RHS) yay!

Expression is true!

D) (ux-uy) == -(unsigned) (y-x)

This is also true!

If we distribute the negative over to (y-x), we would get (x-y). Now, if we cast is to unsigned, we would get (ux-uy). In addition, integer arithmetic is the same whether it's two's complement or unsigned.

E) ( (x >> 2) << 2 ) <= x

This C expression is also true. This is because if we shift x to the right and then shift it back to the left, the two bits that were originally there would become 0's. As a result, the left hand side will always be less than or equal to the right hand side.

For example: Consider 1011 >> 2 = 1110. Then if we shift it to the left by 2, we would get 1110 << 2 = 1000. That number will always be less than 1011.

Now, consider the case where LHS is equal to RHS.

Consider 0100 >> 2 = 1101. 1101 << 2 = 0100.

0100 will always be equal to 0100. So in that case the LHS of the expression is equal to the RHS of the expression.