CS33 HW#1

2.71) int xbyte(packed_t word, int bytenum){
return (word >> (bytenum<<3))&0xFF
}

a) By using & with 0xFF, the most significant bit (along with the most significant 24 bits) will always be 0, therefore this function doesn't function when trying to return any negative number

b) 1111 1101 → -253, subtract everything except 2

int xbyte(packed_t word, int bytenum){
int x = word << (3-bytenum<<3); #shift desired byte to MSB
return x >> 24; #shift new MSB to last 8 bits
}

2.82) c) (x<y) == (-x>-y)

|Tmin| > Tmax!

x = 1000... → Tmin

y = 0111 → Tmax

x<y ✓

-x = 0111...+0001 = 1000... → Tmin

-y = 1000...+0001 = 100...1

-x > -y is not true

returns 0 when x=Tmin, y=Tmax

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

(x+y)×2^4 + y - x = 17×y + 15×x

16x+16y+y-x = 17y+15x

17y+15x = 17y+15x

returns 1: left shift by k = multiply by 2^k

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

x = 111...1, y = 111...1

~x = 000...0, ~y = 000...0

~x+~y+1 = 00...01

x+y = 011...110

~(x+y) = 100...001

~~return 0 when x=y=-1~~

returns 1: distributing the complement + overflow

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

returns 1 always, the bit pattern on both sides is the exact same

e) ((x>>2)<<2) <= x

returns 1 always, there are two cases:

- MSB of x is 1, in which case the bit pattern is unchanged and x == x
- MSB of x is 0, in which case the right shift may truncate the 1s and 2s place before the left shift fills them with 0s, resulting in x possibly decreasing