Jason Tran
605975912

4/9/23
CS33 HW #1

Problem 2.71:

#1. Since word is unsigned and right shifting with a unsigned value in any place will be just adding in 0s (logical). And when you (& 0xFF), the most left bit will always be 0, which means if the original is negative then it wouldn't be accurate.

#2.

```
int xbyte(packed_t word, int bytenum)
 {
   int s_word = (int) word;

   return (s_word  << ((3 - bytenum) << 3) )  >>  24;
}
```

Problem 2.82:

A) **Can Yield 0:** If X is tmin and Y is **anything** bigger than it and you negate both, then X will stay tmin (overflow) which means it cannot be bigger than Y.

B) **Always Yields 1:** If we simplify the left expression, left shifting by 4 is the same as multiplying by 2^4 or 16. So we would get 16x+16y+y-x. Simplifying further we would get 15x+17y or 17y+15x. Which is exactly the right expression.

C) **Always Yields 1:** When you negate: ~x = -x - 1. Therefore, you can change the left expression to -x-1+-y-1+1 then to -x-y-1. Then you can change the right to -(x+y) -1, which is -x-y-1, exactly like the left expression.

D) **Always Yields 1:** In the right expression when you change it to unsigned it doesnt change its bit representation so it could be simplified as - (uy-ux), then you can simplify it to -uy+ux or ux-uy, which is the left expression.

E) **Always Yields 1:** If you right shift by 2, the msb will either be 0 or 1, then left shift by 2 will make the most right bit always 0. If the most right bit is always 0 from the left shift. This means that it can never be bigger than the original value, but can be the same or smaller (since if the left most bit turned one it would be negative).