Sadhana Vadrevu

CS 33 Reinman

Homework 1

**Problem 2.71**

1. The code doesn’t process negative signed bytes correctly. Because word is an unsigned integer, we shift the desired byte to the right using a logical shift. This means the sign of the signed byte-long integer doesn’t get carried over, causing an error.
2. int xbyte (packed\_t word, int bytenum) {

int shiftedleft = word << ((3-bytenum) << 3); *//shift desired byte all the way to the left*

return shiftedleft >> 24; *//then use arithmetic right shift (shiftedleft is a signed int) to shift byte to left-most spot and maintain sign*

}

**Problem 2.82**

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

This is not always true. Assume that the first expression is true and yields 1. If x = Tmin, -x also equals Tmin, because the range of numbers wraps back around after -Tmin-1. So -x would be still be less than or equal to every other number in the range, and the second expression would yield 0, causing the entire expression to always yield 0.

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

This is always true. ((x+y) << 4) + y-x == (x+y) \* 24 + y-x == (16\*x + 16\*y) + y-x == 17\*y + 15\*x

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

Always true. ~x + ~y + 1 == ~x – y == -1 + ~x + 1 -y == -1 – x – y == -1 – (x+y) == -1 + ~(x+y) + 1 == ~(x+y)

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

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

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

Always true. All subtractions happen the same way bit-wise, and the bits are interpreted the same way because of the unsigned casting.

1. ((x >> 2) << 2) <= x

Always true. When you shift x to the right arithmetically and then shift back, you are essentially causing the last two digits to become 0. If x had two 0s in those last two positions initially, then x remains the same after the shifts. However, if the bits in those positions were 1s, then making them 0 would cause the value to decrease.