Maeneka Grewal – 204901948

CS33

Homework #1

2.71:

A. Because you’re returning a 32 bit signed int, the function needs to account for signed values that are stored in the byte. For instance, if the byte 2 looks like this: 1000 0001, this number is -127. What the current code does is shift this byte to the lowest significant byte position. It then replaces the previous 24 bits with 0’s, so the complete 32 bit would be: 0000 0000 0000 0000 0000 0000 1000 0001. This signed int is *not* -127 because the most significant bit is 0, therefore this number never even became negative. So, the correct code should account for negatives and make sure that all the 0’s are actually 1’s when the number is negative.

B.

int xbyte(packed\_t word, int bytenum)

{

int howMuch2Shift = (3 – bytenum) << 3;

int nowInMSB = word << howMuch2Shift;

int shiftDown = nowInMSB >> 24;

return shiftDown;

}

2.82:

A. No, it does not always yield 1. If X is the INT MIN (aka super negative), then making it negative would make it a huge positive number that would exceed the INT MAX because |Tmin| = Tmax+1. Therefore, x would wrap around. If y were a negative number, the expression (-x>-y) is then false. For example, (-int min > -1) is not true.

B. Yes, it does always yield 1. The way a left shift works is that u<<k is the same thing as u\*(2^k). So (x+y)<<4 is the same thing as (x+y)\*(2^4) aka 16(x+y). Then you do 16(x+y)+y-x and you obviously get 17y+15x by basic algebra.

C. No, it does not always yield 1. A simple way to prove this is just to choose random values for x and y and see if the expression is true or false. I chose X = -4 and Y is 3. X is (1100) and Y is (0011). ~X = (0011) which is the number 3. ~Y is (1100) which is the number -4. (3+-4)+1 = 0. Now we can evaluate the right side of the expression. ~(-4+3) = ~-1. -1 in signed binary is (1111). So the ~ of this is (0000), aka 0. 0 == 0. I tried it with a ton of different numbers and it always worked. Mathematically, the ~x is (-x-1) and ~y is (-y-1) always. So (~x) + (~y) + 1 is saying (-x-1) + (-y-1) + 1 which is (-x-y-1). The right hand side is ~(x+y) which is –(x+y)-1 which is (-x-y-1) so boom they’re equal.

D. Yes, it does always yield 1. When you make an unsigned number negative, it becomes 2^32 – that original number. So ux – uy is really ux + (2^32 – uy). –(unsigned)(y-x) is really 2^32 – uy + ux Those are equal!

E.. Yes, it does always yield 1. Remember that u>>k is u/(2^k) and u <<k is u\*(2^k). So x>>2 is x/4. (x/4)<<2 is (x/4)\*4 which is x.