Ethan Wong

Professor Reinman

CS33

Discussion IL – Eric Kong

Homework 1

2.71

a) The provided function does not work properly with negative values due to the masking (& 0xFF). The given function shifts the target byte over to the right and sign fills the bits to the left with 0s because of the mask with 0xFF. This is incorrect because if the value was negative, it would lose its negative sign due to the masking and become a different value. Basically, this function uses an unsigned extend instead of a signed extend.

b) Fixed version of the function:

typedef unsigned packed\_t;

int xbyte(packed\_t word, int bytenum)

{

return (int)(word << ((3 – bytenum) << 3)) >> 24;

}

This function will shift the target byte all the way to the left. Doing an arithmetic right-shift by 24 ensures that the bits to the left are then filled with the value of the most significant bit of the target byte.

2.82

a) It is possible for this expression to yield 0. For instance, if x had a value of Tmin, the second half of the expression would be incorrect. The negative of Tmin is equal to Tmax + 1 – the same value as Tmin. This causes the second half of the expression to fail because there Tmin is never going to be greater than another value.

b) This expression will always yield 1. No matter what values are used for x and y, the result will be 1. The expression can be rewritten like this to prove this statement:

16 \* (x + y) + y – x == 17 \* y + 15 \* x

16x + 16y + y – x == 17y + 15x

17y + 15x == 17y + 15x

c) This expression will always yield 1. No matter what values are used for x and y, the result will be 1. The expression can be rewritten like this to prove this statement:

First, ~x = -x -1

Secondly, ~y = -y – 1

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

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

-x – y -1 == -x – y – 1

d) This expression will always yield 1. No matter what values are used for x and y, the result will be 1. The expression can be rewritten like this to prove this statement:

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

ux – uy == (unsigned) (-(y-x))

ux – uy == (unsigned) (-y + x)

ux – uy == -uy + ux

ux – uy == ux – uy

e) This expression will always yield 1. No matter what value is used for x, the result will be 1. The expression can be rewritten like this to prove this statement:

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

((x ÷ 4) \* 4) <= x

x <= x

As an example, I will demonstrate what will happen if the bit vector 1000 is used.

Initial value : 1000

After right shift : 1110

After left shift : 1000

1000 <= 1000

When the bit vector 1111 is used:

Initial value : 1111

After right shift: 1111

After left shift : 1100

1100 <= 1111