Esther Li

604892225

4/13/19

CS 33 Homework 1

2.71

A)

/\* Failed attempt at xbyte \*/

int xbyte(packed\_t word, int bytenum)

{

return (word >> (bytenum << 3)) & 0xFF;

}

This code is incorrect because word was defined as a 32-bit unsigned, so after the function executes left and right shifts, an unsigned integer will be returned. Since we want to preserve the integer’s sign by sign extension, we need to first cast it to a signed int. This function also does not correctly isolate the designated byte. For example, if bytenum was 0, word would be shifted right by 0 bits, which would not produce the correct integer for the 0th byte.

B)

/\*

-first cast word to a signed int

-left shift by (3 – bytenum) \* 8 to isolate designated byte

-right shift by 24 to sign extend since we only care about preserving the first 8 bits

\*/

Int xbyte(packed\_t word, int bytenum)

{

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

}

2.82

A) False, the expression does not always yield 1. If x = TMin (1000…0000) and y = 1 (0000…0001), multiplying x by -1 (1111…1111) still results in x = TMin (1000…000). It is thus impossible for x, a negative int, to be both less than 1 and greater than -1.

B) True, the expression always yields 1. The expression (x + y) << 4 is equivalent to the expression (x + y) \* 16, so the left side of the equation evaluates to 16 \* x + 16 \* y + y – x, which simplifies to 17 \* y + 15 \* x.

C) True, the expression always yields 1. The left side of the equation ~x + ~y + 1 can be rewritten as (–x – 1) + (–y – 1) + 1 due to the observation that ~x = –x – 1, which can then be simplified to –x – y – 1. We can rewrite –x –y – 1 as –(x + y) – 1, then using the same observation that ~x = –x – 1, the left side of the equation becomes ~(x + y), matching the right side.

D) True, the expression always yields 1. The right side of the equation first casts (y – x) to an unsigned int, but then casts it back to a signed int due to the – sign. The bit representations of (ux – uy) and –(unsigned)(y – x) are equivalent, and any carry bit is discarded by both sides in the same way as TAdd and UAdd have identical bit-level behavior.

E) True, the expression always yields 1. The right shift (x >> 2) preserves the sign of the int through duplicating the most significant bit to fill the 2 leftmost bits. The left shift by 2 fills the 2 rightmost bits with 0, so the result should be less than the original value if those 2 bits were not 0, or equal to the original value if those 2 bits were 0’s to begin with.