2.71

A. Since word itself is unsigned. Right shifting it will still get an unsigned word, which performs logical shift and does not have sign extension. For example, if the extracted byte is zero or negative number, the result will have difference between unsigned and signed.

B.

int xbyte(packed\_t word, int bytenum)

{

int left\_shift = 24 - (bytenum << 3);

int sign = word << left\_shift;

return sign >> 24;

}

2.82

A. No. Let x = TMIN, y = 0. x < y is true, but -x > -y is false since -TMIN = -1, which is less than -0 = 0. Thus, it yields 0.

B. Yes. Since left shifting a signed value is the same as multiply it, and the rounding rule is universal for both positive and negative numbers.

C. Yes. Using the formula -a = ~a + 1. Left = -x – 1 -y – 1 + 1 = -(x + y) – 1. Right = - (x+y) – 1. Thus, left = right, which always yields 1.

D. Yes. Whatever x and y are signed or unsigned, their binary representations are the same. Thus, the implementations of subtraction and minus sign are the same between signed and unsigned.

E. Yes. Right shifting a signed number, whatever it is positive or negative, will potentially make the number smaller compared to divide it directly, since the right shifting will extract the lower terms that were shifted outside the range. Left shifting is the same as multiply.