[2.71]

(A)

This code does not work for negative numbers because masking with 0xFF makes the first 24 bits 0 in all cases; for negative numbers, we want the first 24 bits to be 1s.

(B)

```
typedef unsigned packed_t;
int xbyte(packed_t word, int bytenum) {
    return (word << ((3-bytenum) << 3)) >> 24;
}
```

[2.82]

(A) False.

Let x = t_min and y = 1, where indeed x < y. Then -x = ~x+1 = t_min, and -y = -1, so -y > -x, so x < y does not imply -x > -y.

(B) True.

((x+y)<<4) + y - x
= 16(x+y) + y - x
= 17y + 15x

(C) True.

Recall that -n = ~n+1, so ~n = -n-1.
~x + ~y + 1
= -x - 1 - y - 1 + 1
= -x - y - 1
= -(x+y) - 1
= ~(x+y)

(D) True.

The bit-wise operations are identical for signed and unsigned numbers, so the unsigned cast on ux, uy, and -(y-x) does not have any effect on the expression. Thus, we are really evaluating whether (ux-uy) = -(x-y), which is true by the properties of modular arithmetic.

(E) True.

The last two bits of the number are replaced by 0b00 because left shifts are logical. Since 0b00 is the smallest the last two bits of a number can be, this expression holds.