**Problem 2.64 Solution:**

This problem is very simple, but it reinforces the idea of using different bit patterns as masks.

*code/data/bits.c*

```
1 /* Return 1 when any even bit of x equals 1; 0 otherwise.  Assume w=32 */
2 int any_even_one(unsigned x) {
3     /* Use mask to select even bits */
4     return (x&0x55555555) != 0;
5 }
```

*code/data/bits.c*

**Problem 2.73 Solution:**

Here is the solution.

*code/data/bits.c*

```
1 /* Addition that saturates to TMin or TMax */
2 int saturating_add(int x, int y) {
3     int sum = x + y;
4     int wm1 = (sizeof(int)<<3)-1;
5     /* In the following we create "masks" consisting of all 1's
6        when a condition is true, and all 0's when it is false */
7     int xneg_mask = (x >> wm1);
8     int yneg_mask = (y >> wm1);
9     int sneg_mask = (sum >> wm1);
10    int pos_over_mask = ~xneg_mask & ~yneg_mask & sneg_mask;
11    int neg_over_mask = xneg_mask & yneg_mask & ~sneg_mask;
12    int over_mask = pos_over_mask | neg_over_mask;
13    /* Choose between sum, INT_MAX, and INT_MIN */
14    int result =
15        (~over_mask & sum) |
16        (pos_over_mask & INT_MAX) | (neg_over_mask & INT_MIN);
17    return result;
18 }
```

*code/data/bits.c*

Logically, this code is a straightforward application of the overflow rules for two's complement addition. Avoiding conditionals, however, requires expressing the conditions in terms of masks consisting of all zeros or all ones.

**Problem 2.81 Solution:**

These "C puzzle" problems are a great way to motivate students to think about the properties of computer arithmetic from a programmer's perspective. Our standard lecture on computer arithmetic starts by showing a set of C puzzles. We then go over the answers at the end.

A. `(x>y) == (-x<-y)`. No, Let `x` = $TMin_{32}$, `y` = 0.

B. `((x+y)<<5) + x-y == 31*y+33*x`. Yes, from the ring properties of two's complement arithmetic.

C. `~x+~y == ~(x+y)`. No, let `x` = 0, `y` = 0.

D. `(int) (ux-uy) == -(y-x)`. Yes. Due to the isomorphism between two's complement and unsigned arithmetic.

E. `((x >> 1) << 1) <= x`. Yes. Right shift rounds toward minus infinity.