

## CSI 333 – Programming at the Hardware-Software Interface

### C Program Examples Using Bit Masks

#### Problem 1:

Recall that a variable of type `short` is stored using 2 bytes (i.e., 16 bits) on `itsunix` machines. Assume for convenience that we number the 16 bits from RIGHT TO LEFT using integers 0 through 15. (Thus, the rightmost bit is at position 0 and the leftmost bit is at position 15.) Write a C function `bits_check` with the following declaration.

```
int bits_check (short val)
```

The function should examine the bits in positions 4 and 8 of the parameter `val`. If the bit in position 4 is 0 and the bit in position 8 is 1, then the function must return the value 1; otherwise, the function must return the value 0.

#### Solution to Problem 1:

This problem can be solved in two ways. The first approach uses just bit masks and the second one uses right shifting.

Approach 1: Use appropriate masks to test the bits in positions 4 and 8. The C code for the function based on this approach is given below.

```
int bits_check (short val) {

    short mask1 = 0x0010; /* To check the bit in position 4. */
    short mask2 = 0x0100; /* To check the bit in position 8. */

    short temp1, temp2; /* Temporaries. */

    temp1 = val & mask1; /* If the bit in position 4 is 0, then temp1 */
                        /* will be 0; otherwise, it will be non-zero. */

    temp2 = val & mask2; /* If the bit in position 8 is 1, then temp2 */
                        /* will be non-zero; otherwise, it will be zero. */

    if ((temp1 == 0) && (temp2 != 0)) /* Here we must use "&&" */
        return 1;
    else
        return 0;
} /* End of bits_check. */
```

**Approach 2:** Use appropriate right shifts and just check the bit in position 0. The C code for the function based on this approach is given below.

```
int bits_check (short val) {

    #define SHIFT1  4
    #define SHIFT2  8

    short mask = 0x0001; /* To check the bit in position 0. */

    short temp1, temp2; /* Temporaries. */

    temp1 = (val >> SHIFT1) & mask; /* If the bit in position 4 is 0, then */
                                     /* temp1 will be 0. */
    temp2 = (val >> SHIFT2) & mask; /* If the bit in position 8 is 1, then */
                                     /* temp2 will be non-zero. */

    if ((temp1 == 0) && (temp2 != 0)) /* Here, we must use "&&" */
        return 1;
    else
        return 0;
} /* End of bits_check. */
```

---

### **Problem 2:**

Recall that a variable of type `char` is stored using one byte (i.e., 8 bits). Assume for convenience that we number the 8 bits from RIGHT TO LEFT using integers 0 through 7. (Thus, the rightmost bit is at position 0 and the leftmost bit is at position 7.) Write a C function `rotate_left` with the following declaration.

```
void rotate_left(char *x)
```

The function must ROTATE the bit sequence stored in `*x` one place to the left; that is, it should shift the bits in `*x` one place to the left and fill the vacated spot (i.e., bit position 0) with the bit that “falls off” the left end (i.e., the bit in position 7 before the shift).

For example, if the char variable `y` initially contains the bit pattern 10101111, then after the call

```
rotate_left(&y);
```

`y` must contain the bit pattern 01011111.

### **Solution to Problem 2:**

**Idea:** If the bit that falls off (i.e., the bit in position 7) is 0, then we need just a left shift since the vacated position is automatically filled with 0. If the bit in position 7 is 1, then we need to insert a 1 in position 0 after the shift. The C code for the function is given on the next page.

```
void rotate_left(char *x) {

    char mask1 = 0x80; /* Mask to check the bit in position 7 */
    char mask2 = 0x01; /* Mask to introduce a 1 in position 0 */
    char temp;

    temp = (*x) << 1; /* Left shift *x by 1 bit and store result in temp. */
                    /* Note that this doesn't modify *x. */

    if (*x & mask1) {
        /* The bit in position 7 of *x is a 1. So, we must insert */
        /* a 1 in position 0. */
        *x = temp | mask2;
    }
    else { /* The bit in position 7 of *x is 0. */
        *x = temp;
    }
} /* End of rotate_left */
```

---