PES, Section 2.6
Shift ops

1. What is the value (in hex) of 0xAAAA << 3?

   1010 1010 1010 1010 << 3
   ~~101~~0 1010 1010 1010 000
   0101 0101 0101 0000
   0x5550

2. What is the value of 21 >> 2?

   The key is to remember that this is the decimal value 21, not the hex value 0x21!

   One approach is to interpret >> 2 as integer division (quotient) by 4.
   21 / 4 = 5

   Alternately, we can convert 21 to binary and perform the shift. Using 8 bits (i.e., a char), we can represent 21 as 0001 0101 (16 + 4 + 1).

   0001 0101 >> 2
   00 0001 01~~01~~
   0000 0101
   5

3. Write a single RIMS-compatible C-language statement that copies the values of A7…A5 to B2…B0, inverts the values of A4…A2 and copies them to B7…B5, and copies the values of A1…A0 to B4…B3.

```
B =   ( (A & 0xE0) >> 5) |
      ( ((~A) & 0x1C) << 3) |
      ( (A & 0x03) << 3);
```

4. A parking lot has eight spaces, each with a sensor connected to RIM input A7, A6, ..., or A0. A RIM input being 1 means a car is detected in the corresponding space. Spaces A7 and A6 are reserved handicapped parking. Write a RIM C program that:

(1) Sets B0 to 1 if both handicapped spaces are full, and
(2) Sets B7..B5 equal to the number of available non-handicapped spaces.

The first part can be accomplished by masking and shifting

```
unsigned char full = ((A & 0x80) >> 7) & ((A & 0x40) >> 6);
```

The second part is best handled in several steps:

```
unsigned char cnt = ((A & 0x20) >> 5) +
                    ((A & 0x10) >> 4) +
                    ((A & 0x08) >> 3) +
                    ((A & 0x04) >> 2) +
                    ((A & 0x02) >> 1) +
                    (A & 0x01);
```

Now, we need to shift cnt left to put the 3 bits in positions B7..B5, while writing to B0 at the same time:

```
B = (cnt << 5) | full;
```

Note that it is not possible to write to B0 in one statement, followed by B7…B5 in the next statement, e.g.:

```
B0 = full;
B = (cnt << 5);
```

The second statement would overwrite the value of B0.

5. Binary coded decimal (BCD) is encodes decimal (base-10) numbers in which the value of each digit (0-9) is represented by a 4-bits (values in the range 10-15 are not allowed). BCD takes advantage of the fact that any one decimal numeral can be represented by a four bit pattern. The most obvious way of encoding digits is "natural BCD" (NBCD), where each decimal digit is represented by its corresponding four-bit binary value, as shown in the following table. This is also called "8421" encoding.

Decimal	BCD 8421
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

With 8 bits of I/O, RIMS can express a 2-digit BCD number in the range 0-99. Assume that A3-A0 and B3-B0 represent the lower digits, and A7-A4 and B7-B4 represent the upper digits.

Write a short sequence of RIMS-compatible C language statements that interprets the value of A as a 2-digit BCD number adds 1 to it, and outputs the value to B. (assume that 99 + 1 = 00). Assume that illegal BCD inputs (i.e., hex values in the range A-F) cannot occur as inputs.

```
unsigned char lower = A & 0x0F;
unsigned char upper = A & 0xF0 >> 4;
unsigned char carry = 0;

if( lower < 9 )
    lower++;
else {
    lower = 0;

    // carry propagates from lower-order BCD digit to upper-order BCD digit
    if( upper < 9 ) upper++;
    else upper = 0;
}

B = (upper << 4) | lower;
```