Practice Problem 2.1 (solution page 179)

Perform the following number conversions:

A. 0x25B9D2 to binary

B. binary 1010111001001001 to hexadecimal

C. 0xA8B3D to binary

D. binary 1100100010110110010110 to hexadecimal

A. 11 0101 1011 1001 1101 0010

B. 0xAE49

C. 1010 1000 1011 0011 1101

D. 0x322D96

Practice Problem 2.2 (solution page 179)

Fill in the blank entries in the following table, giving the decimal and hexadecimal representations of different powers of 2:

n	2^n (decimal)	2^n (hexadecimal)
5	32	0x20
23	8,388,608	0x800000
15	32,768	0x8000
13	8192	0x2000
12	4096	0x1000
6	64	0x40
8	256	0x100

$2^{23} = 2^3 \cdot (2^4)^5$

$2^{15} = 2^3 \cdot (2^4)^3$

$2 \cdot (2^4)^3 = 2^{13}$

Practice Problem 2.3 (solution page 180)

A single byte can be represented by 2 hexadecimal digits. Fill in the missing entries in the following table, giving the decimal, binary, and hexadecimal values of different byte patterns:

Decimal	Binary	Hexadecimal
0	0000 0000	0x00
158	10011110	0x9D
76	01001100	0x4C
145	10010001	0x91
174	1010 1110	0xAE
60	0011 1100	0x3C
241	1111 0001	0xF1

Decimal	Binary	Hexadecimal
113	0111 0101	0x75
159	1011 1101	0xBD
245	1111 0101	0xF5

Practice Problem 2.4 (solution page 180)

Without converting the numbers to decimal or binary, try to solve the following arithmetic problems, giving the answers in hexadecimal. *Hint:* Just modify the methods you use for performing decimal addition and subtraction to use base 16.

A. 0x605c + 0x5 = 0x6061

B. 0x605c − 0x20 = 0x603c

C. 0x605c + 32 = 0x608e

D. 0x60fa − 0x605c = 0x9e

A.
```
  605c
+    5
------
  6061
```

D.
```
  60fa
- 605c
------
    9e
```

Practice Problem 2.5 (solution page 180)

Consider the following three calls to `show_bytes`:

```
int a = 0x12345678;
byte_pointer ap = (byte_pointer) &a;
show_bytes(ap, 1); /* A. */
show_bytes(ap, 2); /* B. */
show_bytes(ap, 3); /* C. */
```

Indicate the values that will be printed by each call on a little-endian machine and on a big-endian machine:

A. Little endian: 78 Big endian: 12

B. Little endian: 78 56 Big endian: 12 34

C. Little endian: 78 56 34 Big endian: 12 34 56

Practice Problem 2.6 (solution page 181)

Using `show_int` and `show_float`, we determine that the integer 2607352 has hexadecimal representation `0x0027C8F8`, while the floating-point number 3510593.0 has hexadecimal representation `0x4A1F23E0`.

A. Write the binary representations of these two hexadecimal values.

B. Shift these two strings relative to one another to maximize the number of matching bits. How many bits match?

C. What parts of the strings do not match?

A. ① 0x0027C8F8 ⇒ 0000 0000 0010 0111 1100 1000 1111 1000
② 0x4A1F23E0 ⇒ 0100 1010 0001 1111 0010 0011 1110 0000

B. ① 0000 0000 0010 0111 1100 1000 1111 1000
②>>2 0001 0010 1000 0111 1100 1000 1111 1000
⇒ 28 matchs

Practice Problem 2.7 (solution page 181)

What would be printed as a result of the following call to `show_bytes`?

```
const char *m = "mnopqr";
show_bytes((byte_pointer) m, strlen(m));
```

Note that letters 'a' through 'z' have ASCII codes `0x61` through `0x7A`.

⇒ 60 6E 6F 70 71 72

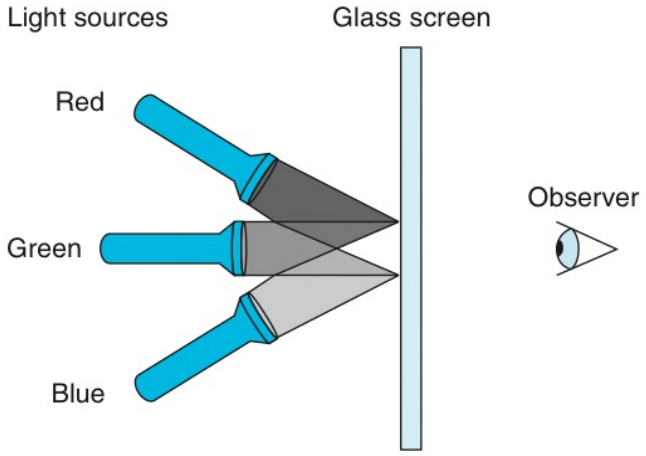
Practice Problem 2.8 (solution page 181)

Fill in the following table showing the results of evaluating Boolean operations on bit vectors.

Operation	Result
a	[01001110]
b	[11100001]
~*a*	10110001
~*b*	00011110
a & *b*	01000000
a | *b*	11101111
a ^ *b*	10101111

Practice Problem 2.9 (solution page 182)

Computers generate color pictures on a video screen or liquid crystal display by mixing three different colors of light: red, green, and blue. Imagine a simple scheme, with three different lights, each of which can be turned on or off, projecting onto a glass screen:



We can then create eight different colors based on the absence (0) or presence (1) of light sources R, G, and B:

R	G	B	Color
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White

A

Black – White
Blue – Yellow
Green – Magenta
Cyan – Red.

Each of these colors can be represented as a bit vector of length 3, and we can apply Boolean operations to them.

A. The complement of a color is formed by turning off the lights that are on and turning on the lights that are off. What would be the complement of each of the eight colors listed above?

B. Describe the effect of applying Boolean operations on the following colors:

Blue | Green = Cyan
Yellow & Cyan = Green
Red ^ Magenta = Blue

011
010
001

Practice Problem 2.10 (solution page 182)

As an application of the property that $a \hat{} a = 0$ for any bit vector a, consider the following program:

```
1   void inplace_swap(int *x, int *y) {
2       *y = *x ^ *y;  /* Step 1 */
3       *x = *x ^ *y;  /* Step 2 */
4       *y = *x ^ *y;  /* Step 3 */
5   }
```

As the name implies, we claim that the effect of this procedure is to swap the values stored at the locations denoted by pointer variables `x` and `y`. Note that unlike the usual technique for swapping two values, we do not need a third location to temporarily store one value while we are moving the other. There is no performance advantage to this way of swapping; it is merely an intellectual amusement.

Starting with values a and b in the locations pointed to by `x` and `y`, respectively, fill in the table that follows, giving the values stored at the two locations after each step of the procedure. Use the properties of $\hat{}$ to show that the desired effect is achieved. Recall that every element is its own additive inverse (that is, $a \hat{} a = 0$).

Step	*x	*y
Initially	a	b
Step 1	a	$a \hat{} b$
Step 2	$a \hat{} a \hat{} b = b$	$a \hat{} b$
Step 3	b	$a \hat{} b \hat{} b = a$

Practice Problem 2.11 (solution page 182)

Armed with the function `inplace_swap` from Problem 2.10, you decide to write code that will reverse the elements of an array by swapping elements from opposite ends of the array, working toward the middle.

You arrive at the following function:

```
1  void reverse_array(int a[], int cnt) {
2      int first, last;
3      for (first = 0, last = cnt-1;
4           first <= last;
5           first++,last--)
6           inplace_swap(&a[first], &a[last]);
7  }
```

When you apply your function to an array containing elements 1, 2, 3, and 4, you find the array now has, as expected, elements 4, 3, 2, and 1. When you try it on an array with elements 1, 2, 3, 4, and 5, however, you are surprised to see that the array now has elements 5, 4, 0, 2, and 1. In fact, you discover that the code always works correctly on arrays of even length, but it sets the middle element to 0 whenever the array has odd length.

A. For an array of odd length `cnt` $= 2k + 1$, what are the values of variables `first` and `last` in the final iteration of function `reverse_array`?

B. Why does this call to function `inplace_swap` set the array element to 0?

C. What simple modification to the code for `reverse_array` would eliminate this problem?

A. first == last == k.

B. For some pointer a,
inplace_swap(a, a)
↝ a = a^a (a==0)

C. for (...
~~first <= last~~ ⇒ first < last.
...
)