
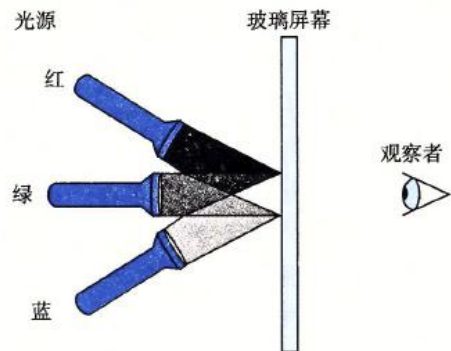


作业 1

练习题 2.9、2.14、2.15、2.18、2.21、2.23

 **练习题 2.9** 通过混合三种不同颜色的光(红色、绿色和蓝色)，计算机可以在视频屏幕或者液晶显示器上产生彩色的画面。设想一种简单的方法，使用三种不同颜色的光，每种光都能打开或关闭，投射到玻璃屏幕上，如图所示：



那么基于光源 R(红)、G(绿)、B(蓝)的关闭(0)或打开(1)，我们就能够创建 8 种不同的颜色：

R	G	B	颜色	R	G	B	颜色
0	0	0	黑色	1	0	0	红色
0	0	1	蓝色	1	0	1	红紫色
0	1	0	绿色	1	1	0	黄色
0	1	1	蓝绿色	1	1	1	白色

这些颜色中的每一种都能用一个长度为 3 的位向量来表示，我们可以对它们进行布尔运算。

A. 一种颜色的补是通过关掉打开的光源，且打开关闭的光源而形成的。那么上面列出的 8 种颜色每一种的补是什么？

B. 描述下列颜色应用布尔运算的结果：

蓝色 | 绿色 =  
黄色 & 蓝绿色 =  
红色 ^ 红紫色 =

A. 黑色-白色，蓝色-黄色，绿色-红紫色，蓝绿色-红色。

B. 蓝色 (001) | 绿色 (010) = 蓝绿色 (011)

黄色 (110) & 蓝绿色 (011) = 绿色 (010)

红色 (100) ^ 红紫色 (101) = 蓝色 (001)



**练习题 2.14** 假设  $x$  和  $y$  的字节值分别为  $0x66$  和  $0x39$ 。填写下表，指明各个 C 表达式的字节值。

表达式	值	表达式	值
$x \& y$	$0x20$	$x \&\& y$	$0x01$
$x   y$	$0x7F$	$x    y$	$0x00$
$\sim x   \sim y$	$0xDF$	$!x    !y$	$0x00$
$x \& !y$	$0x00$	$x \&\& \sim y$	$0x01$

$x$  和  $y$  对应的二进制数分别为  $0110\ 0110$  和  $0011\ 1001$

$x \& y = 0010\ 0000 = 0x20$ ,  $x \&\& y = 1 = 0x01$

$x | y = 0111\ 1111 = 0x7F$ ,  $x || y = 1 = 0x01$

$\sim x | \sim y = 1101\ 1111 | 1100\ 0110 = 1101\ 1111 = 0xDF$ ,  $!x || !y = 0 || 0 = 0 = 0x00$

$x \& !y = 0000\ 0000 = 0x00$ ,  $x \&\& \sim y = 0110\ 0110 \&\& 1100\ 0110 = 1 = 0x01$

**练习题 2.15** 只使用位级和逻辑运算，编写一个 C 表达式，它等价于  $x==y$ 。换句话说，当  $x$  和  $y$  相等时它将返回 1，否则就返回 0。

两个相同的二进制数进行异或操作得 0，则取  $!(x^y)$  等价于  $x==y$ 。



**练习题 2.18** 在第 3 章中，我们将看到由反汇编器生成的列表，反汇编器是一种将可执行程序文件转换回可读性更好的 ASCII 码形式的程序。这些文件包含许多十六进制数字，都是用典型的补码形式来表示这些值。能够认识这些数字并理解它们的意义（例如它们是正数还是负数），是一项重要的技巧。

在下面的列表中，对于标号为 A~I（标记在右边）的那些行，将指令名（sub、mov 和 add）右边显示的（32 位补码形式表示的）十六进制值转换为等价的十进制值。

```

4004d0: 48 81 ec e0 02 00 00    sub    $0x2e0,%rsp          A.
4004d7: 48 8b 44 24 a8          mov    -0x58(%rsp),%rax      B.
4004dc: 48 03 47 28             add    0x28(%rdi),%rax       C.
4004e0: 48 89 44 24 d0          mov    %rax,-0x30(%rsp)      D.
4004e5: 48 8b 44 24 78          mov    0x78(%rsp),%rax       E.
4004ea: 48 89 87 88 00 00 00    mov    %rax,0x88(%rdi)       F.
4004f1: 48 8b 84 24 f8 01 00    mov    0x1f8(%rsp),%rax      G.
4004f8: 00
4004f9: 48 03 44 24 08          add    0x8(%rsp),%rax
4004fe: 48 89 84 24 c0 00 00    mov    %rax,0xc0(%rsp)       H.
400505: 00
400506: 48 8b 44 d4 b8          mov    -0x48(%rsp,%rdx,8),%rax I.

```

A. 736, B. -88 C. 40 D. -48 E. 120 F. 136 G. 504 H. 192 I. -72


 **练习题 2.21** 假设在采用补码运算的 32 位机器上对这些表达式求值，按照图 2-19 的格式填写下表，描述强制类型转换和关系运算的结果。

表 达 式	类 型	求 值
<code>-2147483647-1 == 2147483648U</code>	无符号数	1
<code>-2147483647-1 &lt; 2147483647</code>	有符号数	1
<code>-2147483647-1U &lt; 2147483647</code>	无符号数	0
<code>-2147483647-1 &lt; -2147483647</code>	有符号数	1
<code>-2147483647-1U &lt; -2147483647</code>	无符号数	1

当两个操作数之中有一个是无符号数，运算之前把有符号操作数都转换为无符号数。

1. 无符号数 `0x80000000 == 0x80000000` 即 `2147483648 == 2147483648`
2. 有符号数 `0x80000000 < 0x7FFFFFFF` 即 `-2147483648 < 2147483647`
3. 无符号数 `0x80000000 < 0x7FFFFFFF` 即 `2147483648 < 2147483647`
4. 有符号数 `0x80000000 < 0x80000001` 即 `-2147483648 < -2147483647`
5. 无符号数 `0x80000000 < 0x80000001` 即 `2147483648 < 2147483649`

 **练习题 2.23** 考虑下面的 C 函数：

```
int fun1(unsigned word) {
    return (int) ((word << 24) >> 24);
}

int fun2(unsigned word) {
    return ((int) word << 24) >> 24;
}
```

假设在一个采用补码运算的机器上以 32 位程序来执行这些函数。还假设有符号数值的右移是算术右移，而无符号数值的右移是逻辑右移。

A. 填写下表，说明这些函数对几个示例参数的结果。你会发现用十六进制表示来做会更方便，只要记住十六进制数字 8 到 F 的最高有效位等于 1。

w	fun1(w)	fun2(w)
0x00000076	0x00000076	0x00000076
0x87654321	0x00000021	0x00000021
0x000000C9	0x000000C9	0xFFFFFC9
0xEDCBA987	0x00000087	0xFFFFF87

B. 用语言来描述这些函数执行的有用的计算。

A. `fun1` 是逻辑移位，`fun2` 是算术移位

B. `fun1` 直接取参数的低 8 位得到返回值，返回值是 0~255 之间的整数

`fun2` 取参数的低 8 位并进行符号扩展得到返回值，返回值是 -128~127 之间的整数



**练习题 2.25** 考虑下列代码，这段代码试图计算数组 `a` 中所有元素的和，其中元素的数量由参数 `length` 给出。

```
1  /* WARNING: This is buggy code */
2  float sum_elements(float a[], unsigned length) {
3      int i;
4      float result = 0;
5
6      for (i = 0; i <= length-1; i++)
7          result += a[i];
8      return result;
9  }
```

当参数 `length` 等于 0 时，运行这段代码应该返回 0.0。但实际上，运行时会遇到一个内存错误。请解释为什么会发生这样的情况，并且说明如何修改代码。

参数 `length` 是无符号数，当参数 `length` 等于 0 时，计算 `length-1 = 0 - 1`，根据无符号整数运算法得到最大的无符号整数，可见对任何 `i` 使用无符号数比较都满足 `i <= length - 1`，则函数进入 `for` 循环，进行对数组 `a` 的非法元素的访问，从而引发内存错误。



**练习题 2.26** 现在给你一个任务，写一个函数用来判定一个字符串是否比另一个更长。前提是你要用字符串库函数 `strlen`，它的声明如下：

```
/* Prototype for library function strlen */
size_t strlen(const char *s);
```

最开始你写的函数是这样的：

```
/* Determine whether string s is longer than string t */
/* WARNING: This function is buggy */
int strlonger(char *s, char *t) {
    return strlen(s) - strlen(t) > 0;
}
```

当你在一些示例数据上测试这个函数时，一切似乎都是正确的。进一步研究发现在头文件 `stdio.h` 中数据类型 `size_t` 是定义成 `unsigned int` 的。

- A. 在什么情况下，这个函数会产生不正确的结果？
- B. 解释为什么会出现这样不正确的结果。
- C. 说明如何修改这段代码好让它能可靠地工作。

- A. 当字符串 `s` 比 `t` 短时，应该返回 0，但函数会返回 1
- B. 由 `strlen` 的函数声明可知，它的返回值是无符号数，当字符串 `s` 比 `t` 短时，`strlen(s) - strlen(t)` 的差表示为一很大的无符号数，返回值 `strlen(s) - strlen(t) > 0` 为 1
- C. 语句改为 `return strlen(s) > strlen(t)`