2.22　这个练习很具体地说明了符号扩展如何保持一个补码表示的数值。

A.　[1011]：　$-2^3+2^1+2^0$ = $-8+2+1$ = -5

B.　[11011]：　$-2^4+2^3+2^1+2^0$ = $-16+8+2+1$ = -5

C.　[111011]：　$-2^5+2^4+2^3+2^1+2^0$ = $-32+16+8+2+1$ = -5

2.23　这些函数的表达式是常见的程序“习惯用语”，可以从多个位字段打包成的一个字中提取值。它们利用不同移位运算的零填充和符号扩展属性。请注意强制类型转换和移位运算的顺序。在 fun1 中，移位是在无符号 word 上进行的，因此是逻辑移位。在 fun2 中，移位是在把 word 强制类型转换为 int 之后进行的，因此是算术移位。

A.

w	fun1(w)	fun2(w)
0x00000076	0x00000076	0x00000076
0x87654321	0x00000021	0x00000021
0x000000C9	0x000000C9	0xFFFFFFC9
0xEDCBA987	0x00000087	0xFFFFFF87

B. 函数 fun1 从参数的低 8 位中提取一个值，得到范围 0～255 的一个整数。函数 fun2 也从这个参数的低 8 位中提取一个值，但是它还要执行符号扩展。结果将是介于－128～127 的一个数。

2.24　对于无符号数来说，截断的影响是相当直观的，但是对于补码数却不是。这个练习让你使用非常小的字长来研究它的属性。

十六进制		无符号		补码	
原始数	截断后的数	原始数	截断后的数	原始数	截断后的数
0	0	0	0	0	0
2	2	2	2	2	2
9	1	9	1	-7	1
B	3	11	3	-5	3
F	7	15	7	-1	-1

正如等式(2.9)所描述的，这种截断无符号数值的结果就是发现它们模 8 的余数。截断有符号数的结果要更复杂一些。根据等式(2.10)，我们首先计算这个参数模 8 后的余数。对于参数 0～7，将得出值 0～7，对于参数－8～－1 也是一样。然后我们对这些余数应用函数 $U2T_3$，得出两个 0～3 和－4～－1 序列的反复。

2.25　设计这个问题是要说明从有符号数到无符号数的隐式强制类型转换很容易引起错误。将参数 length 作为一个无符号数来传递看上去是件相当自然的事情，因为没有人会想到使用一个长度为负数的值。停止条件 i<=length-1 看上去也很自然。但是把这两点组合到一起，将产生意想不到的结果！

因为参数 length 是无符号的，计算 0－1 将使用无符号运算，这等价于模数加法。结果得到 $UMax$。≤比较同样使用无符号数比较，而因为任何数都是小于或者等于 $UMax$ 的，所以这个比较总是为真！因此，代码将试图访问数组 a 的非法元素。

有两种方法可以改正这段代码，其一是将 length 声明为 int 类型，其二是将 for 循环的测试条件改为 i<length。

2.26　这个例子说明了无符号运算的一个细微的特性，同时也是我们执行无符号运算时不会意识到的属性。这会导致一些非常棘手的错误。

A. 在什么情况下，这个函数会产生不正确的结果？当 s 比 t 短的时候，该函数会不正确地返回 1。

B. 解释为什么会出现这样不正确的结果。由于 strlen 被定义为产生一个无符号的结果，差和比较都采用无符号运算来计算。当 s 比 t 短的时候，strlen(s)-strlen(t)的差会为负，但是变成了一个很大的无符号数，且大于 0。