```
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.
```

2.2.4　有符号数和无符号数之间的转换

C 语言允许在各种不同的数字数据类型之间做强制类型转换。例如，假设变量 x 声明为int，u 声明为 unsigned。表达式 (unsigned)x 会将 x 的值转换成一个无符号数值，而 (int)u 将 u 的值转换成一个有符号整数。将有符号数强制类型转换成无符号数，或者反过来，会得到什么结果呢？从数学的角度来说，可以想象到几种不同的规则。很明显，对于在两种形式中都能表示的值，我们是想要保持不变的。另一方面，将负数转换成无符号数可能会得到 0。如果转换的无符号数太大以至于超出了补码能够表示的范围，可能会得到 $TMax$。不过，对于大多数 C 语言的实现来说，对这个问题的回答都是从位级角度来看的，而不是数的角度。

比如说，考虑下面的代码：

```
1       short    int    v  = -12345;
2       unsigned short uv = (unsigned short) v;
3       printf("v = %d, uv = %u\n", v, uv);
```

在一台采用补码的机器上，上述代码会产生如下输出：

```
v = -12345, uv = 53191
```

我们看到，强制类型转换的结果保持位值不变，只是改变了解释这些位的方式。在图 2-15 中我们看到过，−12 345 的 16 位补码表示与 53 191 的 16 位无符号表示是完全一样的。将 short 强制类型转换为 unsigned short 改变数值，但是不改变位表示。

类似地，考虑下面的代码：

```
1       unsigned u = 4294967295u;   /* UMax */
2       int      tu = (int) u;
3       printf("u = %u, tu = %d\n", u, tu);
```

在一台采用补码的机器上，上述代码会产生如下输出：

```
u = 4294967295, tu = -1
```

从图 2-14 我们可以看到，对于 32 位字长来说，无符号形式的 4 294 967 295($UMax_{32}$)和补码形式的−1 的位模式是完全一样的。将 unsigned 强制类型转换成 int，底层的位表示保持不变。

对于大多数 C 语言的实现，处理同样字长的有符号数和无符号数之间相互转换的一般规则是：数值可能会改变，但是位模式不变。让我们用更数学化的形式来描述这个规则。我们定义函数 $U2B_w$ 和 $T2B_w$，它们将数值映射为无符号数和补码形式的位表示。也就是说，给定 $0 \leqslant x \leqslant UMax_w$ 范围内的一个整数 x，函数 $U2B_w(x)$会给出 x 的唯一的 w 位无符号表示。相似地，当 x 满足 $TMin_w \leqslant x \leqslant TMax_w$，函数 $T2B_w(x)$会给出 x 的唯一的 w 位补码表示。

现在，将函数 $T2U_w$ 定义为 $T2U_w(x) \doteq B2U_w(T2B_w(x))$。这个函数的输入是一个