```
6      je      .L2
7      shrq    $2, %rdi
8      call    rfun
9      addq    %rbx, %rax
10   .L2:
11     popq    %rbx
12     ret
```

A. rfun 存储在被调用者保存寄存器%rbx 中的值是什么？

B. 填写上述 C 代码中缺失的表达式。

3.8　数组分配和访问

C 语言中的数组是一种将标量数据聚集成更大数据类型的方式。C 语言实现数组的方式非常简单，因此很容易翻译成机器代码。C 语言的一个不同寻常的特点是可以产生指向数组中元素的指针，并对这些指针进行运算。在机器代码中，这些指针会被翻译成地址计算。

优化编译器非常善于简化数组索引所使用的地址计算。不过这使得 C 代码和它到机器代码的翻译之间的对应关系有些难以理解。

3.8.1　基本原则

对于数据类型 T 和整型常数 N，声明如下：

T A[N];

起始位置表示为 x_A。这个声明有两个效果。首先，它在内存中分配一个 $L \cdot N$ 字节的连续区域，这里 L 是数据类型 T 的大小(单位为字节)。其次，它引入了标识符 A，可以用 A 来作为指向数组开头的指针，这个指针的值就是 x_A。可以用 0～N-1 的整数索引来访问该数组元素。数组元素 i 会被存放在地址为 $x_A + L \cdot i$ 的地方。

作为示例，让我们来看看下面这样的声明：

```
char    A[12];
char   *B[8];
int     C[6];
double *D[5];
```

这些声明会产生带下列参数的数组：

数组	元素大小	总的大小	起始地址	元素 i
A	1	12	x_A	x_A+i
B	8	64	x_B	x_B+8i
C	4	24	x_C	x_C+4i
D	8	40	x_D	x_D+8i

数组 A 由 12 个单字节(char)元素组成。数组 C 由 6 个整数组成，每个需要 8 个字节。B 和 D 都是指针数组，因此每个数组元素都是 8 个字节。

x86-64 的内存引用指令可以用来简化数组访问。例如，假设 E 是一个 int 型的数组，而我们想计算 E[i]，在此，E 的地址存放在寄存器%rdx 中，而 i 存放在寄存器%rcx 中。然后，指令

```
movl (%rdx,%rcx,4),%eax
```

会执行地址计算 x_E+4i，读这个内存位置的值，并将结果存放到寄存器%eax 中。允许的