3.32 追踪此等级上的程序的执行有助于理解过程调用和返回的很多方面。可以明确看到调用时控制是怎么传给过程的以及返回时调用函数如何继续执行的。还可以看到参数通过寄存器%rdi 和%rsi 传递，结果通过寄存器%rax 返回。

指令			状态值(指令开始执行前)					描述
标号	PC	指令	%rdi	%rsi	%rax	%rsp	*%rsp	
M1	0x400560	callq	10	—	—	0x7fffffffe820	—	调用 first(10)
F1	0x400548	lea	10	—	—	0x7fffffffe818	0x400565	first 的入口
F2	0x40054c	sub	10	11	—	0x7fffffffe818	0x400565	
F3	0x400550	callq	9	11	—	0x7fffffffe818	0x400565	调用 last(9,11)
L1	0x400540	mov	9	11	—	0x7fffffffe810	0x400555	last 的入口
L2	0x400543	imul	9	11	9	0x7fffffffe810	0x400555	
L3	0x400547	retq	9	11	99	0x7fffffffe810	0x400555	从 last 返回 99
F4	0x400555	repz repq	9	11	99	0x7fffffffe818	0x400565	从 first 返回 99
M2	0x400565	mov	9	11	99	0x7fffffffe820	—	继续执行 main

3.33 由于是多种数据大小混合在一起，这道题有点儿难。

让我们先描述第一种答案，再解释第二种可能性。如果假设第一个加(第 3 行)实现*u+=a，第二个加(第 4 行)实现 v+=b，然后我们可以看到 a 通过% edi 作为第一个参数传递，把它从 4 个字节转换成 8 个字节，再加到%rdx 指向的 8 个字节上。这就意味着 a 必定是 int 类型，u 一定是 long * 类型。还可以看到参数 b 的低位字节被加到了%rcx 指向的字节。这就意味着 v 一定是 char * ，但是 b 的类型是不确定的——它的大小可以是 1、2、4 或 8 字节。注意到返回值为 6 就能解决这种不确定性，这个返回值是 a 和 b 大小的和。因为我们知道 a 的大小是 4 字节，所以可以推断出 b 一定是 2 字节的。

该函数的一个加了注释的版本解释了这些细节：

```
   int procprobl(int a, short b, long *u, char *v)
   a in %edi, b in %si, u in %rdx, v in %rcx
1  procprob:
2    movslq  %edi, %rdi        Convert a to long
3    addq    %rdi, (%rdx)      Add to *u (long)
4    addb    %sil, (%rcx)      Add low-order byte of b to *v
5    movl    $6, %eax          Return 4+2
6    ret
```

此外，我们可以看到如果以它们在 C 代码中出现相反的顺序在汇编代码中计算这两个和，这段汇编代码同样合法。这会导致交换参数 a 和 b，参数 u 和 v，得到如下原型：

```
int procprob(int b, short a, long *v, char *u);
```

3.34 这个例子展示了被调用者保存寄存器的使用，以及保存局部数据的栈的使用。

A. 可以看到第 9～14 行将局部值 a0～a5 分别保存进被调用者保存寄存器%rbx、%r15、%r14、%r13、%r12 和%rbp。

B. 局部值 a6 和 a7 存放在栈中相对于栈指针偏移量为 0 和 8 的地方(第 16 和 18 行)。

C. 在存储完 6 个局部变量之后，这个程序用完了所有的被调用者保存寄存器，所以剩下的两个值保存在栈上。

3.35 这道题给了一个检查递归函数代码的机会。要学的一个很重要的内容就是，递归代码与我们看到的其他函数的结构一模一样。栈和寄存器保存规则足以让递归函数正确执行。

A. 寄存器%rbx 保存参数 x 的值，所以它可以被用来计算结果表达式。

B. 汇编代码是由下面的 C 代码产生而来的：