到栈上，而参数 7 位于栈顶。通过栈传递参数时，所有的数据大小都向 8 的倍数对齐。参数到位以后，程序就可以执行 call 指令将控制转移到过程 Q 了。过程 Q 可以通过寄存器访问参数，有必要的话也可以通过栈访问。相应地，如果 Q 也调用了某个有超过 6 个参数的函数，它也需要在自己的栈帧中为超出 6 个部分的参数分配空间，如图 3-25 中标号为“参数构造区”的区域所示。

作为参数传递的示例，考虑图 3-29a 所示的 C 函数 proc。这个函数有 8 个参数，包括字节数不同的整数(8、4、2 和 1)和不同类型的指针，每个都是 8 字节的。

```
void proc(long  a1, long  *a1p,
          int   a2, int   *a2p,
          short a3, short *a3p,
          char  a4, char  *a4p)
{
    *a1p += a1;
    *a2p += a2;
    *a3p += a3;
    *a4p += a4;
}
```

a) C代码

```
    void proc(a1, a1p, a2, a2p, a3, a3p, a4, a4p)
    Arguments passed as follows:
      a1  in %rdi          (64 bits)
      a1p in %rsi          (64 bits)
      a2  in %edx          (32 bits)
      a2p in %rcx          (64 bits)
      a3  in %r8w          (16 bits)
      a3p in %r9           (64 bits)
      a4  at %rsp+8        ( 8 bits)
      a4p at %rsp+16       (64 bits)
1   proc:
2     movq    16(%rsp), %rax    Fetch a4p   (64 bits)
3     addq    %rdi, (%rsi)      *a1p += a1  (64 bits)
4     addl    %edx, (%rcx)      *a2p += a2  (32 bits)
5     addw    %r8w, (%r9)       *a3p += a3  (16 bits)
6     movl    8(%rsp), %edx     Fetch a4    ( 8 bits)
7     addb    %dl, (%rax)       *a4p += a4  ( 8 bits)
8     ret                       Return
```

b) 生成的汇编代码

图 3-29　有多个不同类型参数的函数示例。参数 1～6 通过寄存器传递，而参数 7～8 通过栈传递

图 3-29b 中给出 proc 生成的汇编代码。前面 6 个参数通过寄存器传递，后面 2 个通过栈传递，就像图 3-30 中画出来的那样。可以看到，作为过程调用的一部分，返回地址被压入栈中。因而这两个参数位于相对于栈指针距离为 8 和 16 的位置。在这段代码中，我们可以看到根据操作数的大小，使用了 ADD 指令的不同版本：a1(long)使用 addq，a2(int)使用 addl，a3(short)使用 addw，而 a4(char)使用 addb。请注意第 6 行的 movl 指令从内存读入 4 字节，而后面的 addb 指令只使用其中的低位一字节。