其中一些关于机器代码和它的反汇编表示的特性值得注意：

- x86-64 的指令长度从 1 到 15 个字节不等。常用的指令以及操作数较少的指令所需的字节数少，而那些不太常用或操作数较多的指令所需字节数较多。
- 设计指令格式的方式是，从某个给定位置开始，可以将字节唯一地解码成机器指令。例如，只有指令 pushq %rbx 是以字节值 53 开头的。
- 反汇编器只是基于机器代码文件中的字节序列来确定汇编代码。它不需要访问该程序的源代码或汇编代码。
- 反汇编器使用的指令命名规则与 GCC 生成的汇编代码使用的有些细微的差别。在我们的示例中，它省略了很多指令结尾的‘q’。这些后缀是大小指示符，在大多数情况中可以省略。相反，反汇编器给 call 和 ret 指令添加了‘q’后缀，同样，省略这些后缀也没有问题。

生成实际可执行的代码需要对一组目标代码文件运行链接器，而这一组目标代码文件中必须含有一个 main 函数。假设在文件 main.c 中有下面这样的函数：

```
#include <stdio.h>

void multstore(long, long, long *);

int main() {
    long d;
    multstore(2, 3, &d);
    printf("2 * 3 --> %ld\n", d);
    return 0;
}
long mult2(long a, long b) {
    long s = a * b;
    return s;
 }
```

然后，我们用如下方法生成可执行文件 prog：

```
linux> gcc -Og -o prog main.c mstore.c
```

文件 prog 变成了 8 655 个字节，因为它不仅包含了两个过程的代码，还包含了用来启动和终止程序的代码，以及用来与操作系统交互的代码。我们也可以反汇编 prog 文件：

```
linux> objdump -d prog
```

反汇编器会抽取出各种代码序列，包括下面这段：

```
    Disassembly of function sum multstore binary file prog
1   0000000000400540 <multstore>:
2     400540:  53                    push   %rbx
3     400541:  48 89 d3              mov    %rdx,%rbx
4     400544:  e8 42 00 00 00        callq  40058b <mult2>
5     400549:  48 89 03              mov    %rax,(%rbx)
6     40054c:  5b                    pop    %rbx
7     40054d:  c3                    retq
8     40054e:  90                    nop
9     40054f:  90                    nop
```

这段代码与 mstore.c 反汇编产生的代码几乎完全一样。其中一个主要的区别是左边