汇编代码文件包含各种声明，包括下面几行：

```
multstore:
  pushq   %rbx
  movq    %rdx, %rbx
  call    mult2
  movq    %rax, (%rbx)
  popq    %rbx
  ret
```

上面代码中每个缩进去的行都对应于一条机器指令。比如，pushq 指令表示应该将寄存器%rbx 的内容压入程序栈中。这段代码中已经除去了所有关于局部变量名或数据类型的信息。

如果我们使用“-c”命令行选项，GCC 会编译并汇编该代码：

```
linux> gcc -Og -c mstore.c
```

这就会产生目标代码文件 mstore.o，它是二进制格式的，所以无法直接查看。1368 字节的文件 mstore.o 中有一段 14 字节的序列，它的十六进制表示为：

```
53 48 89 d3 e8 00 00 00 00 48 89 03 5b c3
```

这就是上面列出的汇编指令对应的目标代码。从中得到一个重要信息，即机器执行的程序只是一个字节序列，它是对一系列指令的编码。机器对产生这些指令的源代码几乎一无所知。

旁注　如何展示程序的字节表示

要展示程序(比如说 mstore)的二进制目标代码，我们用反汇编器(后面会讲到)确定该过程的代码长度是 14 字节。然后，在文件 mstore.o 上运行 GNU 调试工具 GDB，输入命令：

```
(gdb) x/14xb multstore
```

这条命令告诉 GDB 显示(简写为‘x’)从函数 multstore 所处地址开始的 14 个十六进制格式表示(也简写为‘x’)的字节(简写为‘b’)。你会发现，GDB 有很多有用的特性可以用来分析机器级程序，我们会在 3.10.2 节中讨论。

要查看机器代码文件的内容，有一类称为反汇编器(disassembler)的程序非常有用。这些程序根据机器代码产生一种类似于汇编代码的格式。在 Linux 系统中，带‘-d’命令行标志的程序 OBJDUMP(表示“object dump”)可以充当这个角色：

```
linux> objdump -d mstore.o
```

结果如下(这里，我们在左边增加了行号，在右边增加了斜体表示的注解)：

```
1  Disassembly of function multstore in binary file mstore.o
2  0000000000000000 <multstore>:
3  Offset   Bytes              Equivalent assembly language
4     0:    53                 push   %rbx
5     1:    48 89 d3           mov    %rdx,%rbx
6     4:    e8 00 00 00 00     callq  9 <multstore+0x9>
7     9:    48 89 03           mov    %rax,(%rbx)
8     c:    5b                 pop    %rbx
9     d:    c3                 retq
```

在左边，我们看到按照前面给出的字节顺序排列的 14 个十六进制字节值，它们分成了若干组，每组有 1～5 个字节。每组都是一条指令，右边是等价的汇编语言。