x86-64 代码是由 GCC 编译器产生的。Y86-64 代码与之类似，但有以下不同点：

- Y86-64 将常数加载到寄存器(第 2～3 行)，因为它在算术指令中不能使用立即数。
- 要实现从内存读取一个数值并将其与一个寄存器相加，Y86-64 代码需要两条指令(第 8～9 行)，而 x86-64 只需要一条 addq 指令(第 5 行)。
- 我们手工编写的 Y86-64 实现有一个优势，即 subq 指令(第 11 行)同时还设置了条件码，因此 GCC 生成代码中的 testq 指令(第 9 行)就不是必需的。不过为此，Y86-64 代码必须用 andq 指令(第 5 行)在进入循环之前设置条件码。

图 4-7 给出了用 Y86-64 汇编代码编写的一个完整的程序文件的例子。这个程序既包括数据，也包括指令。伪指令(directive)指明应该将代码或数据放在什么位置，以及如何对齐。这个程序详细说明了栈的放置、数据初始化、程序初始化和程序结束等问题。

```
1   # Execution begins at address 0
2           .pos 0
3           irmovq stack, %rsp      # Set up stack pointer
4           call main               # Execute main program
5           halt                    # Terminate program
6
7   # Array of 4 elements
8           .align 8
9   array:
10          .quad 0x000d000d000d
11          .quad 0x00c000c000c0
12          .quad 0x0b000b000b00
13          .quad 0xa000a000a000
14
15  main:
16          irmovq array,%rdi
17          irmovq $4,%rsi
18          call sum                # sum(array, 4)
19          ret
20
21  # long sum(long *start, long count)
22  # start in %rdi, count in %rsi
23  sum:
24          irmovq $8,%r8           # Constant 8
25          irmovq $1,%r9           # Constant 1
26          xorq %rax,%rax          # sum = 0
27          andq %rsi,%rsi          # Set CC
28          jmp     test            # Goto test
29  loop:
30          mrmovq (%rdi),%r10      # Get *start
31          addq %r10,%rax          # Add to sum
32          addq %r8,%rdi           # start++
33          subq %r9,%rsi           # count--.  Set CC
34  test:
35          jne    loop             # Stop when 0
36          ret                     # Return
37
38  # Stack starts here and grows to lower addresses
39          .pos 0x200
40  stack:
```

图 4-7　用 Y86-64 汇编代码编写的一个例子程序。调用 sum 函数来计算一个具有 4 个元素的数组的和