

在这个程序中，以“.”开头的词是汇编器伪指令(assembly directives)，它们告诉汇编器调整地址，以便在那儿产生代码或插入一些数据。伪指令 .pos 0(第2行)告诉汇编器应该从地址0处开始产生代码。这个地址是所有 Y86-64 程序的起点。接下来的一条指令(第3行)初始化栈指针。我们可以看到程序结尾处(第40行)声明了标号 stack，并且用一个 .pos 伪指令(第39行)指明地址 0x200。因此栈会从这个地址开始，向低地址增长。我们必须保证栈不会增长得太大以至于覆盖了代码或者其他程序数据。

程序的第8~13行声明了一个4个字的数组，值分别为

```
0x000d000d000d000d, 0x00c000c000c000c0
0x0b000b000b000b00, 0xa000a000a000a000
```

标号 array 表明了数组的起始，并且在8字节边界处对齐(用 .align 伪指令指定)。第16~19行给出了“main”过程，在过程中对那个四字节数组调用了 sum 函数，然后停止。

正如例子所示，由于我们创建 Y86-64 代码的唯一工具是汇编器，程序员必须执行本来通常交给编译器、链接器和运行时系统来完成任务。幸好我们只用 Y86-64 来写一些小的程序，对此一些简单的机制就足够了。

图4-8是YAS的汇编器对图4-7中代码进行汇编的结果。为了便于理解，汇编器的输出结果是ASCII码格式。汇编文件中有指令或数据的行上，目标代码包含一个地址，后面跟着1~10个字节的值。

我们实现了一个指令集模拟器，称为 YIS，它的目的是模拟 Y86-64 机器代码程序的执行，而不用试图去模拟任何具体处理器实现的行为。这种形式的模拟有助于在有实际硬件可用之前调试程序，也有助于检查模拟硬件或者在硬件上运行程序的结果。用 YIS 运行例子的目标代码，产生如下输出：

```
Stopped in 34 steps at PC = 0x13. Status 'HLT', CC Z=1 S=0 O=0
```


```
Changes to registers:
```

```
%rax: 0x0000000000000000    0x0000abcdabcdabcd
%rsp: 0x0000000000000000    0x0000000000000200
%rdi: 0x0000000000000000    0x0000000000000038
%r8:  0x0000000000000000    0x0000000000000008
%r9:  0x0000000000000000    0x0000000000000001
%r10: 0x0000000000000000    0x0000a000a000a000
```

```
Changes to memory:
```

```
0x01f0: 0x0000000000000000    0x0000000000000055
0x01f8: 0x0000000000000000    0x0000000000000013
```

模拟输出的第一行总结了执行以及 PC 和程序状态的结果值。模拟器只打印出在模拟过程中被改变了的寄存器或内存中的字。左边是原始值(这里都是0)，右边是最终的值。从输出中我们可以看到，寄存器 %rax 的值为 0xabcdabcdabcdabcd，即传给函数 sum 的四元素数组的和。另外，我们还能看到栈从地址 0x200 开始，向下增长，栈的使用导致内存地址 0x1f0~0x1f8 发生了变化。可执行代码的最大地址为 0x090，所以数值的入栈和出栈不会破坏可执行代码。

 **练习题 4.3** 机器级程序中常见的模式之一是将一个常数值与一个寄存器相加。利用目前已给出的 Y86-64 指令，实现这个操作需要一条 irmovq 指令把常数加载到寄存器，然后一条 addq 指令把这个寄存器值与目标寄存器值相加。假设我们想增加一条新指令 iaddq，格式如下：