```
4       movq    %rsp, %rdi      Save stack pointer
5       pushq   $0xabcd         Push test value
6       popq    %rsp            Pop to stack pointer
7       movq    %rsp, %rax      Set popped value as return value
8       movq    %rdi, %rsp      Restore stack pointer
9       ret
```

我们发现函数总是返回 0xabcd。这表示 popq %rsp 的行为是怎样的？还有什么其他 Y86-64 指令也会有相同的行为吗？

旁注　正确了解细节：x86 模型间的不一致

练习题 4.7 和练习题 4.8 可以帮助我们确定对于压入和弹出栈指针指令的一致惯例。看上去似乎没有理由会执行这样两种操作，那么一个很自然的问题就是“为什么要担心这样一些吹毛求疵的细节呢?”

从下面 Intel 关于 PUSH 指令的文档[51]的节选中，可以学到关于这个一致的重要性的有用的教训：

对于 IA-32 处理器，从 Intel 286 开始，PUSH ESP 指令将 ESP 寄存器的值压入栈中，就好像它存在于这条指令被执行之前。(对于 Intel 64 体系结构、IA-32 体系结构的实地址模式和虚 8086 模式来说也是这样。)对于 Intel® 8086 处理器，PUSH SP 将 SP 寄存器的新值压入栈中(也就是减去 2 之后的值)。(PUSH ESP 指令。Intel 公司。50。)

虽然这个说明的具体细节可能难以理解，但是我们可以看到这条注释说明的是当执行压入栈指针寄存器指令时，不同型号的 x86 处理器会做不同的事情。有些会压入原始的值，而有些会压入减去后的值。(有趣的是，对于弹出栈指针寄存器没有类似的歧义。)这种不一致有两个缺点：

- 它降低了代码的可移植性。取决于处理器模型，程序可能会有不同的行为。虽然这样特殊的指令并不常见，但是即使是潜在的不兼容也可能带来严重的后果。
- 它增加了文档的复杂性。正如在这里我们看到的那样，需要一个特别的说明来澄清这些不同之处。即使没有这样的特殊情况，x86 文档就已经够复杂的了。

因此我们的结论是，从长远来看，提前了解细节，力争保持完全的一致能够节省很多的麻烦。

4.2　逻辑设计和硬件控制语言 HCL

在硬件设计中，用电子电路来计算对位进行运算的函数，以及在各种存储器单元中存储位。大多数现代电路技术都是用信号线上的高电压或低电压来表示不同的位值。在当前的技术中，逻辑 1 是用 1.0 伏特左右的高电压表示的，而逻辑 0 是用 0.0 伏特左右的低电压表示的。要实现一个数字系统需要三个主要的组成部分：计算对位进行操作的函数的组合逻辑、存储位的存储器单元，以及控制存储器单元更新的时钟信号。

本节简要描述这些不同的组成部分。我们还将介绍 HCL(Hardware Control Language，硬件控制语言)，用这种语言来描述不同处理器设计的控制逻辑。在此我们只是简略地描述 HCL，HCL 完整的参考请见网络旁注 ARCH:HCL。

旁注　现代逻辑设计

曾经，硬件设计者通过描绘示意性的逻辑电路图来进行电路设计(最早是用纸和笔，后来是用计算机图形终端)。现在，大多数设计都是用硬件描述语言(Hardware Description