**4 Processor Architecture**

# 英语

|  |  |
| --- | --- |
| 1. dwarf使……相形见绌 2. manufacturer制造商 3. retrieval检索 4. embody体现 5. concise简洁的 | 1. envision设想 2. while同时 3. intrinsic固有的 4. monolithic整体式的 |

**|** *The instructions supported by a particular processor and their byte-level encodings are known as its instruction set architecture (ISA)*

*Each manufacturer produces processors of ever-growing performance and complexity, but the different models remain compatible at the ISA level.*

*The ISA provides a conceptual layer of abstraction between compiler writers, who need only know what instructions are permitted and how they are encoded, and processor designers, who must build machines that execute those instructions.*

指令集架构在编译器编写者和处理器设计者之间提供一层抽象概念层

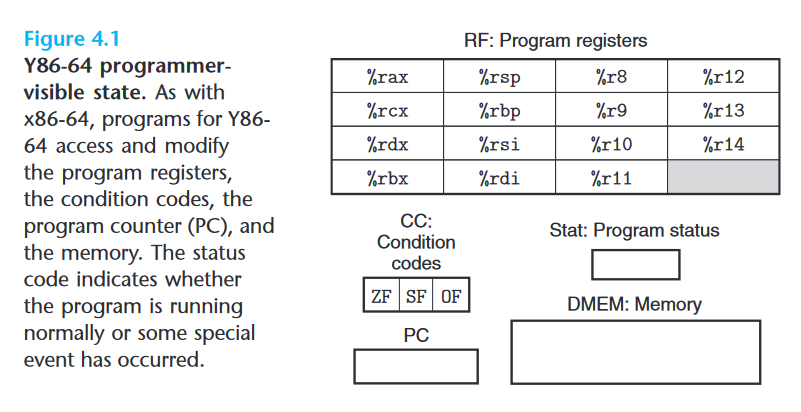
编译器编写者只需要知道架构内存在哪些指令以及这些指令是怎么被编码的

处理器设计者则必须要设计执行这些指令的机器

# 4.1 *The Y86-64 Instruction Set Architecture*

**|** 可以将Y86-64视为X86-64的精简版，包括状态的不同组成部分，指令集以及指令集的编码格式、编程惯例和异常处理机制

## 4.1.1 *Programmer-Visible State*



Y86-64中的每条指令都可以读或者修改处理器的一些状态，这些状态称为“程序员注释1可见状态”

Y86-64的处理器状态包括：15个64位的寄存器、ZF、SF、OF三个状态码、PC、Stat、主存

15个寄存器包括%rax、%rcx、%rdx、%rbx、%rsp、%rbp、%rsi、%rdi、%r8~%r14

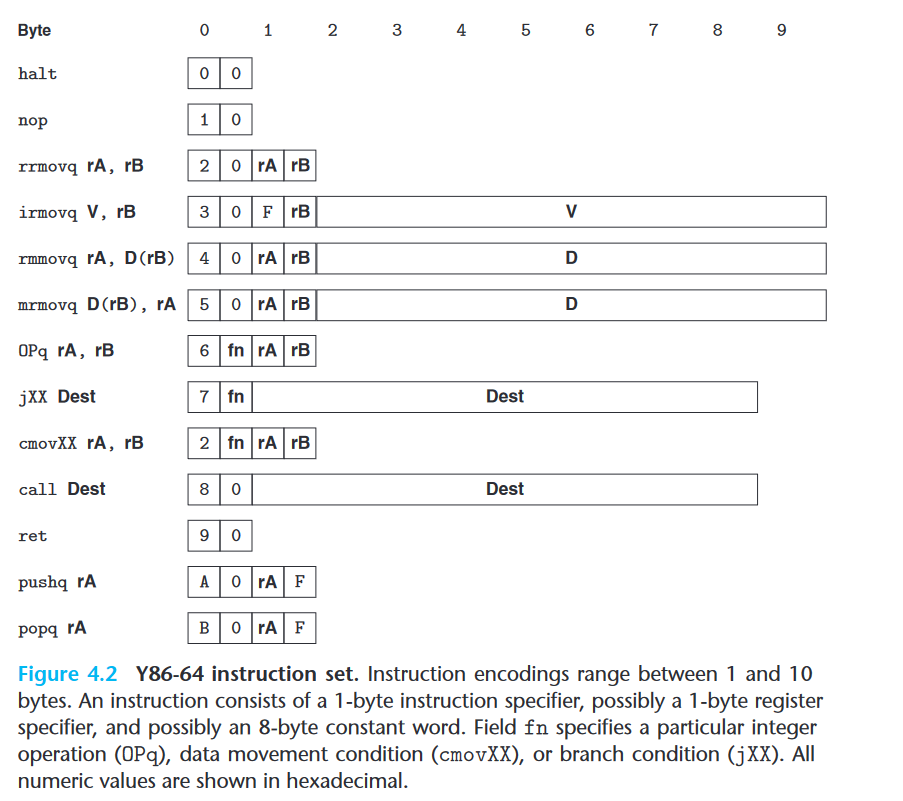
ZF、OF、SF三个用于整数算术、逻辑运算效果的指令

PC存储着当前正在执行的指令的地址

Stat状态码表示程序执行的整体状态，可以表示当前程序是在正常操作执行还是发生某种异常

主存在概念上是存储程序和数据的大的字节数组，Y86-64使用虚拟存储，由操作系统和硬件合作将虚拟地址翻译为物理地址

## 4.1.2 *Y86-64 Instructions*



Y86-64所支持的指令集如上图所示，指令所操作的对象是8字节整数

1. 将movq指令按照源和目的操作数的形式划分为irmovq、rmmovq、mrmovq、rrmovq

其中源操作数可以是i立即数、r寄存器、m存储器(指令助记符的第一个字符)，目的操作数可以是r寄存器、m存储器(指令助记符的第二个字符)

存储器操作数的内存引用方式是简单的基址和偏移量形式，不支持变址和缩放的内存引用形式

同x86-64，不支持内存和内存之间的数据直接传递，也不支持将立即数直接传送到内存

1. OPq指令表示基于整数的四种运算指令：addq、subq、andq、xorq

不像x86-64允许整数运算指令直接操作存储器数据，OPq指令仅仅在寄存器数据上操作

OPq指令可以设置SF、ZF、OF条件码位

1. jXX指令表示7种跳转指令：jmp、jle、jl、je、jne、jge、jg

跳转指令执行跳转的条件同x86-64

1. cmovXX表示6种条件传送指令：cmovle、cmovl、cmove、cmovne、cmovge、cmovg

cmovXX指令的格式同rrmovq，是寄存器和寄存器之间的数据传送

1. call和ret

call指令将返回地址入栈，并将目的地址送PC

ret指令则取返回地址送PC

1. pushq和popq

同x86-64的pushq、popq

pushq先将%rsp减8然后入栈数据，popq先取%rsp内存引用上的数据，然后加%rsp 8

1. halt指令暂停指令的执行

x86-64中有类似的指令hlt，但是并不允许应用程序执行该条指令，因为它会暂停整个系统的操作

在Y86-64中，halt指令的执行会使得处理器停止，设置状态码为HLT

## 4.1.3 *Instruction Encoding*

[行内引用](https://www.wolai.com/9YFhQHtAXYeR1vApt5a9Ad#bfeH8u7JiqiB1dwofAHX2K)Figure4.2也展示出了Y86-64指令集的位级编码：

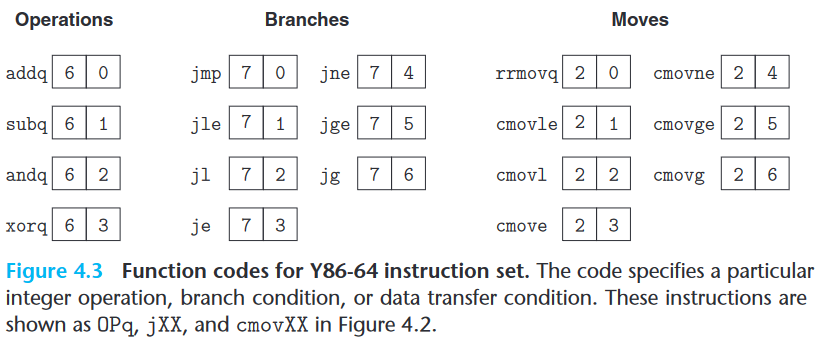
* 1. 指令编码长度范围在1字节到10字节之间
  2. 每条指令都以1字节的指令指示符开始，可能会有1字节的寄存器指示符、8字节的常量
  3. fn字段域表示了一个特定的整数运算或者是条件传送或者是分支指令

### *Instruction specifier*

每条指令都有一字节的指令指示符来表示当前指令类型。这一字节被分为两个4位部分：高4位表示代码，低4位表示功能

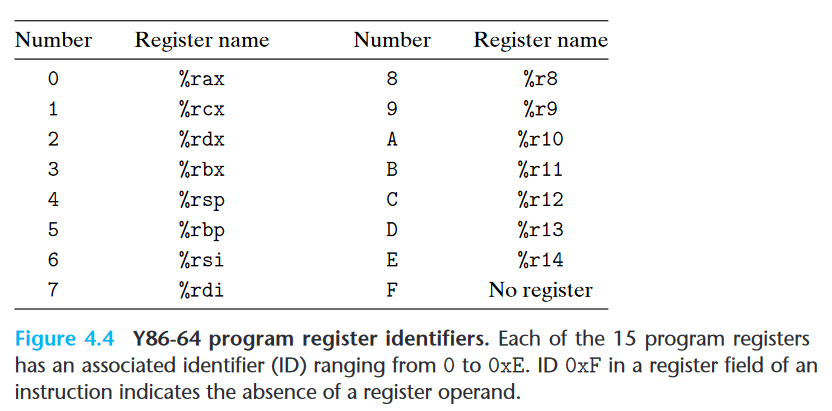
代码字段的值范围在0~0xB之间

功能字段仅用于区分一组共享代码字段的指令集合的某个指令，如下



值得注意的是rrmovq与cmovXX具有同样的格式，可以被视为jmp类的cmov，它的功能字段值恰好为0

### *Register specifier*



寄存器指示符的编码是从0x0~0xE。Y86-64和X86-64寄存器编号是匹配的(0xF为%r15)

**在Y86-64中的0xF表示不访问寄存器**，如irmovq、popq、pushq这些只需要一个寄存器操作数的指令，另一个寄存器操作数字段的值即为0xF

### 8B Constant

8B的常量字可以作为irmovq中的立即数、rmmovq和mrmovq中的内存引用的位移以及分支和调用的目的地。这里需要注意的是Y86-64中的分支和调用目的地址以绝对地址给出——X86-64中是采用PC相对寻址。且与X86-64一样，所有整数采用小端编码

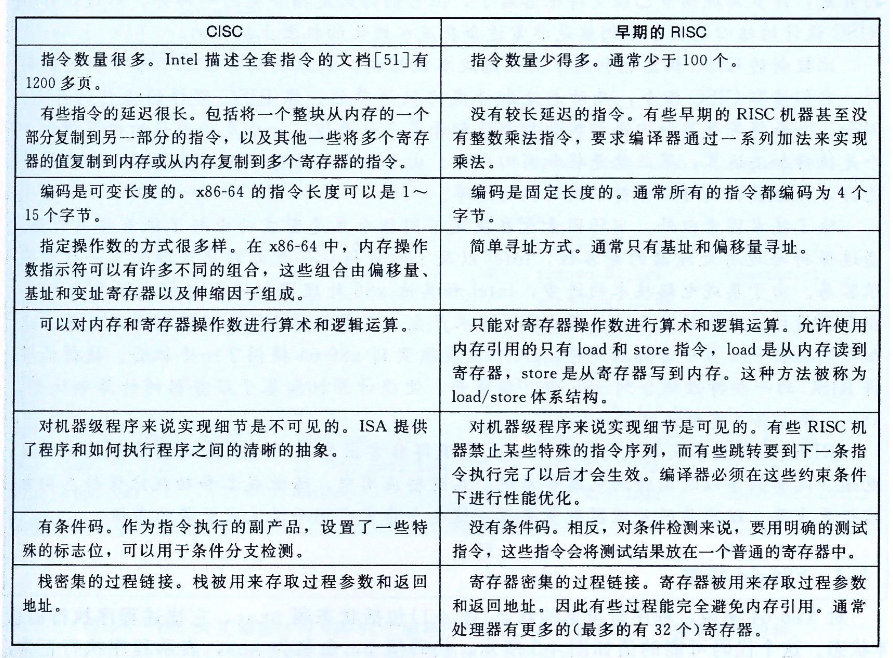
### 示例

例如指令“rmmovq %rsp,0x123456789abcd(%rdx)”的机器编码为"0x4042cdab896745230100"

✨*One important property of any instruction set is that the byte encodings must have a unique interpretation. An arbitrary sequence of bytes either encodes a unique instruction sequence or is not a legal byte sequence.*

Y86-64就是这样，只要知道指令指示符这一字节我们就能决定其他附加字节的长度和含义。但是如果不知道一段代码序列的起始位置，我们就不能准确地确定怎样将序列划分成单独的指令

### RISC and CISC

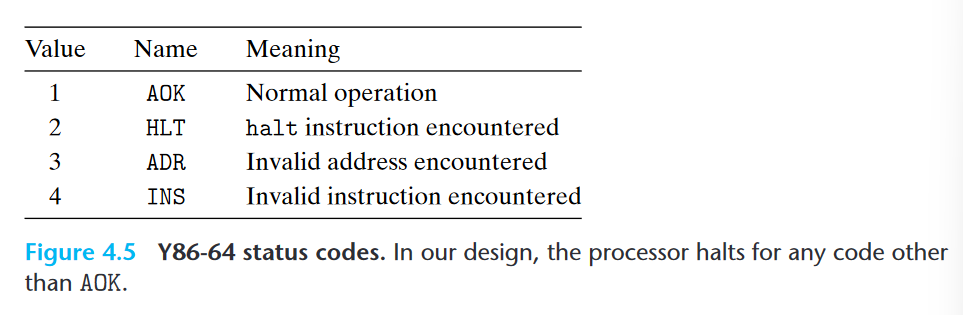


而Y86-64指令集既有CISC指令集的属性也有RISC指令集的属性。和CISC一样，它有条件码，长度可变的指令，并用栈来保存返回地址。和RISC一样的是，采用load-store机制和规则编码，通过寄存器来传参——Y86-64可以看作是采用了CISC x86-64指令集，但利用RISC的一些规则进行了简化

RISC和CISC的发展趋势是相结合的，而且各在不同的市场上表现得很出色

## 4.1.4 *Y86-64 Exception*

在4.1.1[行内引用](https://www.wolai.com/9YFhQHtAXYeR1vApt5a9Ad#8avVKS5oFQqkz6CNBJZa4z)中讲到，程序员的可见状态中包含一个Stat状态码位，其值如下：



值为1时，表示AOK，即表示程序正常执行

值为2时，表示HLT，是由指令halt引起的，表示处理器暂停执行指令

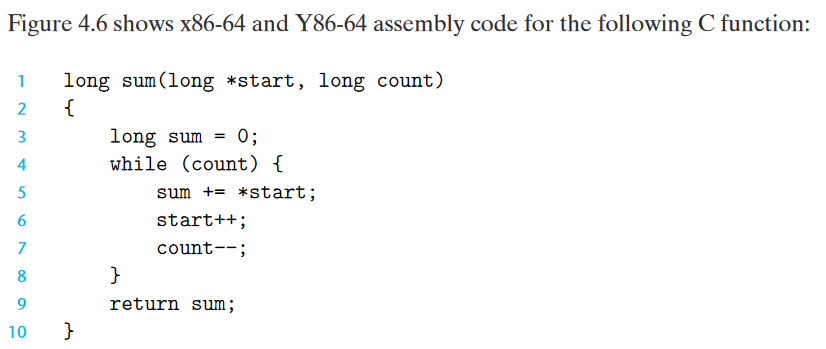
值为3时，表示ADR，即处理器在获取指令或读取或写入数据时尝试读取或写入无效的内存地址

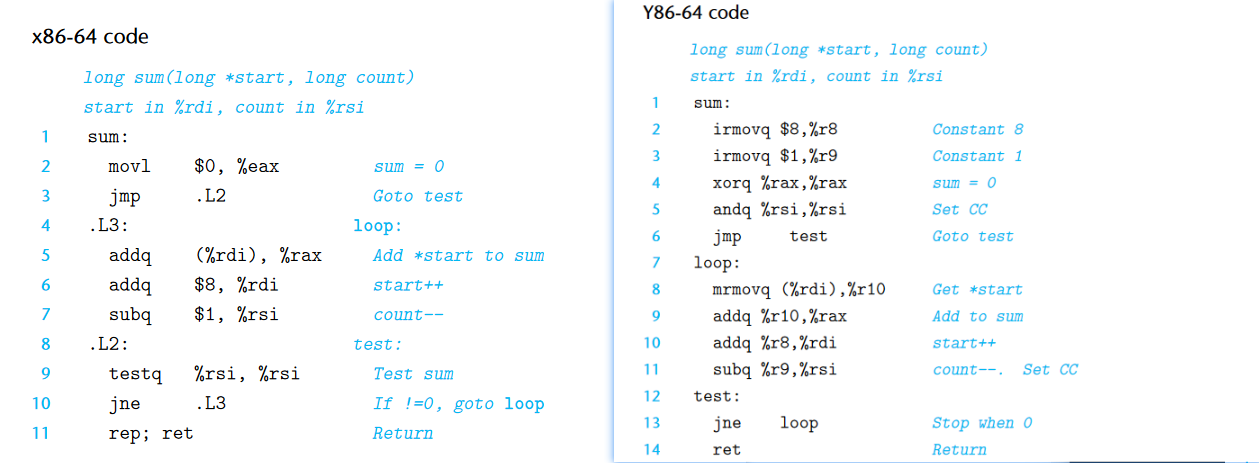
根据具体的实现细节限制最大的地址范围，超出此范围的地址的访问均会触发ADR异常

值为4时，表示INS，即无效的指令

## 4.1.5 *Y86-64 Procedure*

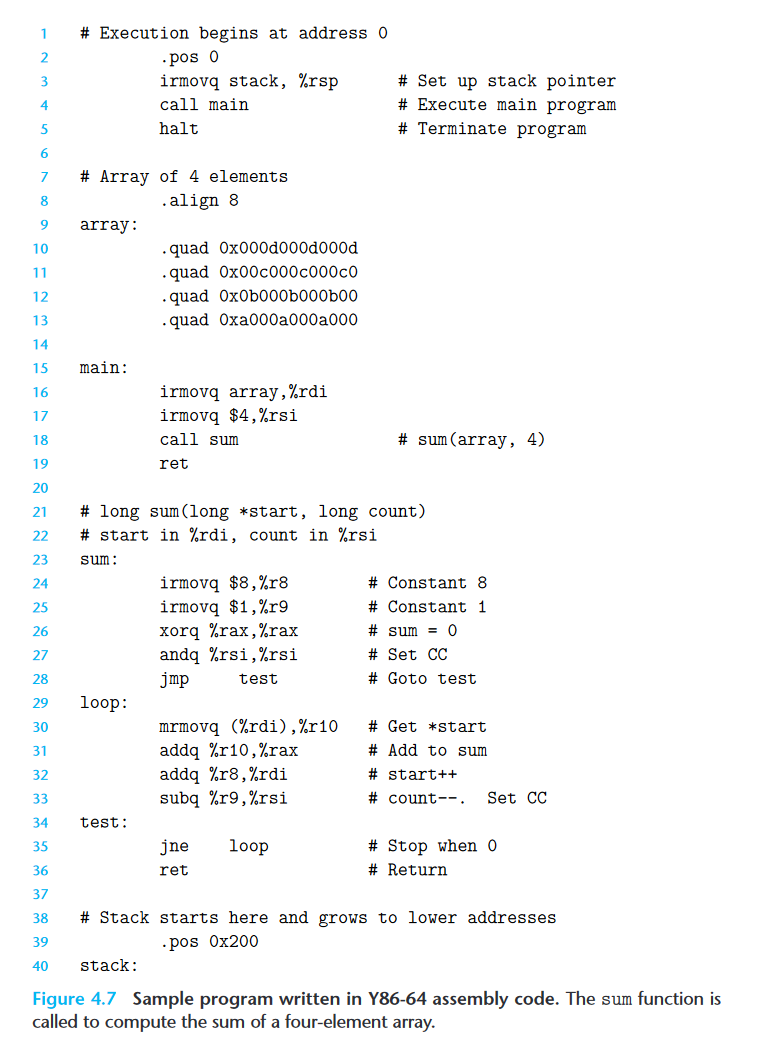
对于下列C函数，观察它的x86-64汇编代码和Y86-64汇编代码：





可以看到以下几点不同：

1. Y86-64是load-store架构，所有的算术运算均是寄存器和寄存器之间的，不存在寄存器和立即数运算，因此需要2-3行的汇编代码irmovq；也不存在存储器操作数和寄存器之间的运算，因此需要第8行的mrmovq

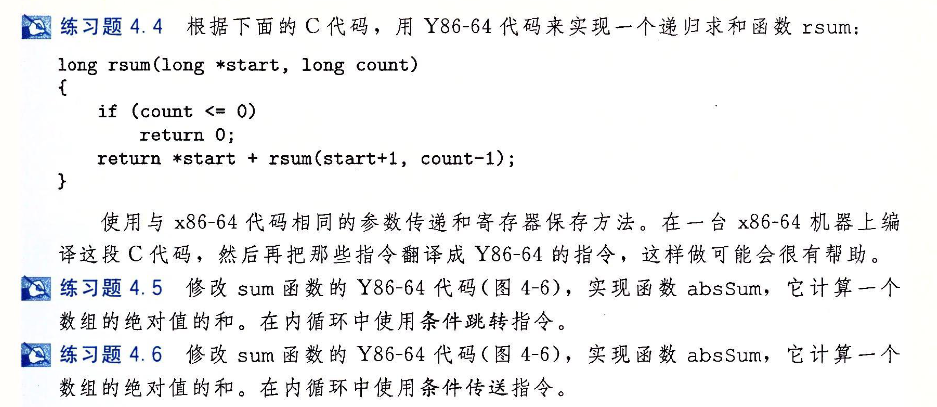


其中"."开头的是汇编器伪指令，告诉汇编器调整生成代码的地址或插入一些数据字

.pos 0表示汇编器应该从地址0开始生成代码——所有Y86-64汇编程序均以该伪指令开始

.align 8表示汇编器插入数据的对齐偏移量为8B

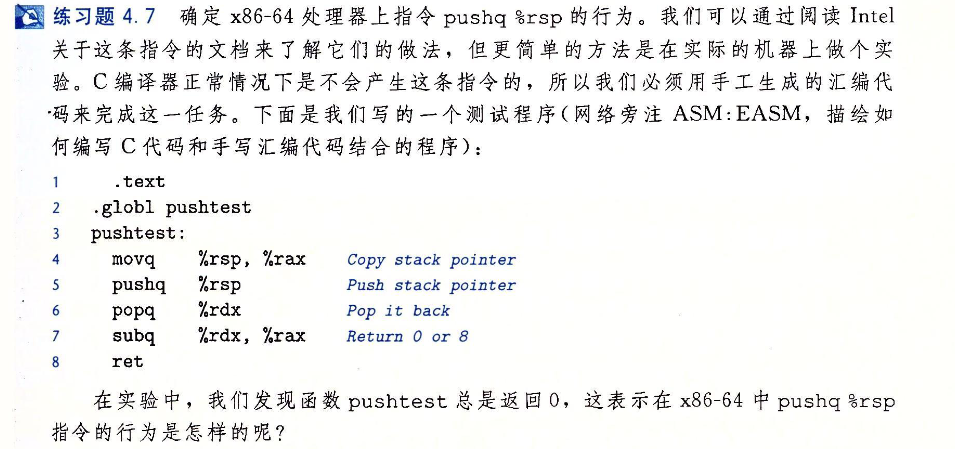
在实验的过程中会用到yas和yis



## 4.1.5 *Some Y86-64 Instruction Details*

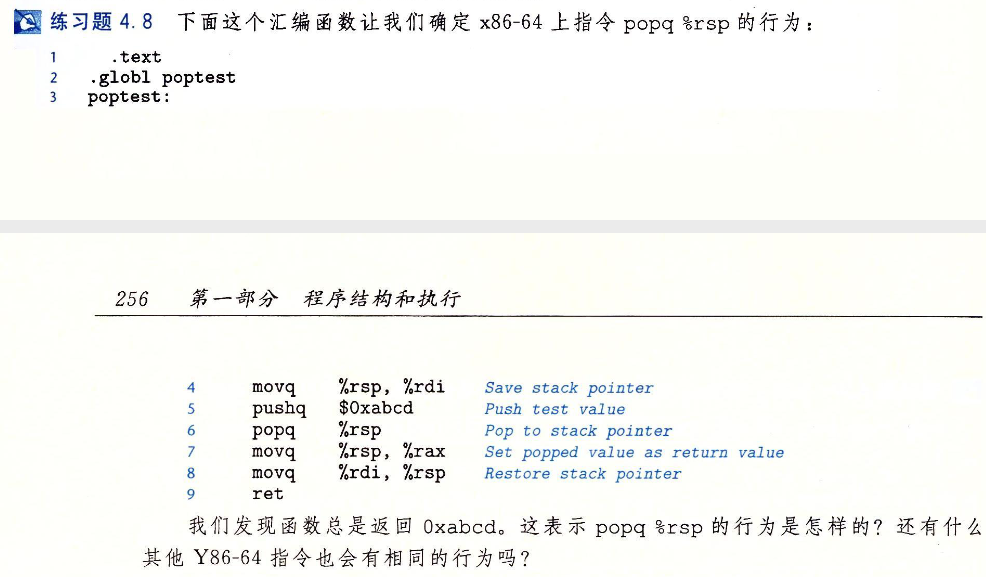
需要注意两种特别的指令的组合

1. pushq %rsp是存放原来的%rsp还是减8之后的%rsp



总返回0说明，**push %rsp保存的是原来的%rsp即未减8的**

1. popq %rsp是恢复栈中的值还是原来的%rsp+8



返回的总是0xabcd，表明**popq %rsp得到的是栈中的值**

# 4.2 *Logic Design and the Hardware Control Language HCL*

[注释1] 使用汇编语言或者机器语言编写程序的程序员