```
subq $8,%rsp            Decrement stack pointer
movq %rbp,(%rsp)        Store %rbp on stack
```

它们之间的区别是在机器代码中 pushq 指令编码为 1 个字节，而上面那两条指令一共需要 8 个字节。图 3-9 中前两栏给出的是，当%rsp 为 0x108,%rax 为 0x123 时，执行指令 pushq %rax 的效果。首先%rsp 会减 8，得到 0x100，然后会将 0x123 存放到内存地址 0x100 处。

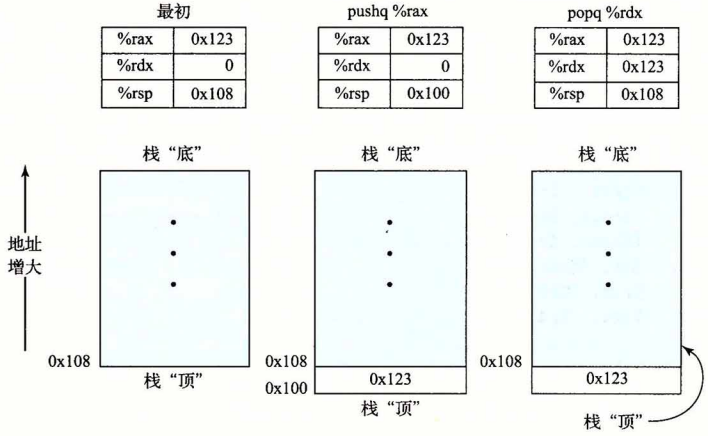

图 3-9　栈操作说明。根据惯例，我们的栈是倒过来画的，因而栈“顶”在底部。x86-64 中，栈向低地址方向增长，所以压栈是减小栈指针(寄存器%rsp)的值，并将数据存放到内存中，而出栈是从内存中读数据，并增加栈指针的值

弹出一个四字的操作包括从栈顶位置读出数据，然后将栈指针加 8。因此，指令 popq %rax 等价于下面两条指令：

```
movq (%rsp),%rax        Read %rax from stack
addq $8,%rsp            Increment stack pointer
```

图 3-9 的第三栏说明的是在执行完 pushq 后立即执行指令 popq %rdx 的效果。先从内存中读出值 0x123，再写到寄存器%rdx 中，然后，寄存器%rsp 的值将增加回到 0x108。如图中所示，值 0x123 仍然会保持在内存位置 0x100 中，直到被覆盖(例如被另一条入栈操作覆盖)。无论如何，%rsp 指向的地址总是栈顶。

因为栈和程序代码以及其他形式的程序数据都是放在同一内存中，所以程序可以用标准的内存寻址方法访问栈内的任意位置。例如，假设栈顶元素是四字，指令 movq 8(%rsp),%rdx 会将第二个四字从栈中复制到寄存器%rdx。

3.5　算术和逻辑操作

图 3-10 列出了 x86-64 的一些整数和逻辑操作。大多数操作都分成了指令类，这些指令类有各种带不同大小操作数的变种(只有 leaq 没有其他大小的变种)。例如，指令类 ADD 由四条加法指令组成：addb、addw、addl 和 addq，分别是字节加法、字加法、双字加法和四字加法。事实上，给出的每个指令类都有对这四种不同大小数据的指令。这些