这 4 种类型是很有帮助的。

两个内存传送指令中的内存引用方式是简单的基址和偏移量形式。在地址计算中，我们不支持第二变址寄存器(second index register)和任何寄存器值的伸缩(scaling)。

同 x86-64 一样，我们不允许从一个内存地址直接传送到另一个内存地址。另外，也不允许将立即数传送到内存。

- 有 4 个整数操作指令，如图 4-2 中的 OPq。它们是 addq、subq、andq 和 xorq。它们只对寄存器数据进行操作，而 x86-64 还允许对内存数据进行这些操作。这些指令会设置 3 个条件码 ZF、SF 和 OF(零、符号和溢出)。
- 7 个跳转指令(图 4-2 中的 jXX)是 jmp、jle、jl、je、jne、jge 和 jg。根据分支指令的类型和条件代码的设置来选择分支。分支条件和 x86-64 的一样(见图 3-15)。
- 有 6 个条件传送指令(图 4-2 中的 cmovXX)：cmovle、cmovl、cmove、cmovne、cmovge 和 cmovg。这些指令的格式与寄存器-寄存器传送指令 rrmovq 一样，但是只有当条件码满足所需要的约束时，才会更新目的寄存器的值。
- call 指令将返回地址入栈，然后跳到目的地址。ret 指令从这样的调用中返回。
- pushq 和 popq 指令实现了入栈和出栈，就像在 x86-64 中一样。
- halt 指令停止指令的执行。x86-64 中有一个与之相当的指令 hlt。x86-64 的应用程序不允许使用这条指令，因为它会导致整个系统暂停运行。对于 Y86-64 来说，执行 halt 指令会导致处理器停止，并将状态码设置为 HLT(参见 4.1.4 节)。

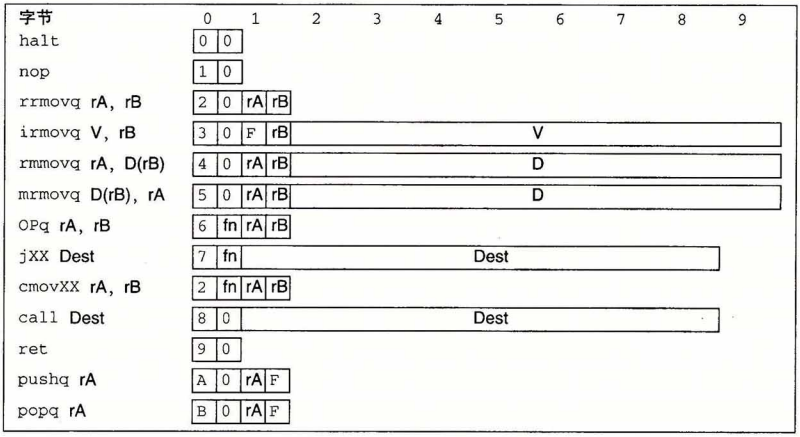

图 4-2　Y86-64 指令集。指令编码长度从 1 个字节到 10 个字节不等。一条指令含有一个单字节的指令指示符，可能含有一个单字节的寄存器指示符，还可能含有一个 8 字节的常数字。字段 fn 指明是某个整数操作(OPq)、数据传送条件(cmovXX)或是分支条件(jXX)。所有的数值都用十六进制表示

4.1.3　指令编码

图 4-2 还给出了指令的字节级编码。每条指令需要 1～10 个字节不等，这取决于需要哪些字段。每条指令的第一个字节表明指令的类型。这个字节分为两个部分，每部分 4