

是，有一组标准的编程规范控制着如何使用寄存器来管理栈、传递函数参数、从函数的返回值，以及存储局部和临时数据。我们会在描述过程的实现时（特别是在 3.7 节中），讲述这些惯例。

3.4.1 操作数指示符


大多数指令有一个或多个操作数(operand)，指出执行一个操作中要使用的源数据值，以及放置结果的目的位置。x86-64 支持多种操作数格式(参见图 3-3)。源数据值可以以常数形式给出，或是从寄存器或内存中读出。结果可以存放在寄存器或内存中。因此，各种不同的操作数的可能性被分为三种类型。第一种类型是立即数(immediate)，用来表示常数值。在 ATT 格式的汇编代码中，立即数的书写方式是‘\$’后面跟一个用标准 C 表示法表示的整数，比如，\$-577 或 \$0x1F。不同的指令允许的立即数值范围不同，汇编器会自动选择最紧凑的方式进行数值编码。第二种类型是寄存器(register)，它表示某个寄存器的内容，16 个寄存器的低位 1 字节、2 字节、4 字节或 8 字节中的一个作为操作数，这些字节数分别对应于 8 位、16 位、32 位或 64 位。在图 3-3 中，我们用符号 r_a 来表示任意寄存器 a ，用引用 $R[r_a]$ 来表示它的值，这是将寄存器集合看成一个数组 R ，用寄存器标识符作为索引。

第三类操作数是内存引用，它会根据计算出来的地址(通常称为有效地址)访问某个内存位置。因为将内存看成一个很大的字节数组，我们用符号 $M_b[Addr]$ 表示对存储在内存中从地址 $Addr$ 开始的 b 个字节值的引用。为了简便，我们通常省去下标 b 。

如图 3-3 所示，有多种不同的寻址模式，允许不同形式的内存引用。表中底部用语法 $Imm(r_b, r_i, s)$ 表示的是最常用的形式。这样的引用有四个组成部分：一个立即数偏移 Imm ，一个基址寄存器 r_b ，一个变址寄存器 r_i 和一个比例因子 s ，这里 s 必须是 1、2、4 或者 8。基址和变址寄存器都必须是 64 位寄存器。有效地址被计算为 $Imm + R[r_b] + R[r_i] \cdot s$ 。引用数组元素时，会用到这种通用形式。其他形式都是这种通用形式的特殊情况，只是省略了某些部分。正如我们将看到的，当引用数组和结构元素时，比较复杂的寻址模式是很有用的。

类型	格式	操作数值	名称
立即数	$\$Imm$	Imm	立即数寻址
寄存器	r_a	$R[r_a]$	寄存器寻址
存储器	Imm	$M[Imm]$	绝对寻址
存储器	(r_a)	$M[R[r_a]]$	间接寻址
存储器	$Imm(r_b)$	$M[Imm + R[r_b]]$	(基址 + 偏移量) 寻址
存储器	(r_b, r_i)	$M[R[r_b] + R[r_i]]$	变址寻址
存储器	$Imm(r_b, r_i)$	$M[Imm + R[r_b] + R[r_i]]$	变址寻址
存储器	(r_i, s)	$M[R[r_i] \cdot s]$	比例变址寻址
存储器	$Imm(r_i, s)$	$M[Imm + R[r_i] \cdot s]$	比例变址寻址
存储器	(r_b, r_i, s)	$M[R[r_b] + R[r_i] \cdot s]$	比例变址寻址
存储器	$Imm(r_b, r_i, s)$	$M[Imm + R[r_b] + R[r_i] \cdot s]$	比例变址寻址

图 3-3 操作数格式。操作数可以表示立即数(常数)值、寄存器值或是来自内存的值。比例因子 s 必须是 1、2、4 或者 8

 练习题 3.1 假设下面的值存放在指明的内存地址和寄存器中：