当然，在 C 程序中包含汇编代码使得这些代码与某类特殊的机器相关(例如 x86-64)，所以只应该在想要的特性只能以此种方式才能访问到时才使用它。

3.3　数据格式

由于是从 16 位体系结构扩展成 32 位的，Intel 用术语“字(word)”表示 16 位数据类型。因此，称 32 位数为“双字(double words)”，称 64 位数为“四字(quad words)”。图 3-1 给出了 C 语言基本数据类型对应的 x86-64 表示。标准 int 值存储为双字(32 位)。指针(在此用 char * 表示)存储为 8 字节的四字，64 位机器本来就预期如此。x86-64 中，数据类型 long 实现为 64 位，允许表示的值范围较大。本章代码示例中的大部分都使用了指针和 long 数据类型，所以都是四字操作。x86-64 指令集同样包括完整的针对字节、字和双字的指令。

C 声明	Intel 数据类型	汇编代码后缀	大小(字节)
char	字节	b	1
short	字	w	2
int	双字	l	4
long	四字	q	8
char*	四字	q	8
float	单精度	s	4
double	双精度	l	8

图 3-1　C 语言数据类型在 x86-64 中的大小。在 64 位机器中，指针长 8 字节

浮点数主要有两种形式：单精度(4 字节)值，对应于 C 语言数据类型 float；双精度(8 字节)值，对应于 C 语言数据类型 double。x86 家族的微处理器历史上实现过对一种特殊的 80 位(10 字节)浮点格式进行全套的浮点运算(参见家庭作业 2.86)。可以在 C 程序中用声明 long double 来指定这种格式。不过我们不建议使用这种格式。它不能移植到其他类型的机器上，而且实现的硬件也不如单精度和双精度算术运算的高效。

如图所示，大多数 GCC 生成的汇编代码指令都有一个字符的后缀，表明操作数的大小。例如，数据传送指令有四个变种：movb(传送字节)、movw(传送字)、movl(传送双字)和 movq(传送四字)。后缀‘l’用来表示双字，因为 32 位数被看成是“长字(long word)”。注意，汇编代码也使用后缀‘l’来表示 4 字节整数和 8 字节双精度浮点数。这不会产生歧义，因为浮点数使用的是一组完全不同的指令和寄存器。

3.4　访问信息

一个 x86-64 的中央处理单元(CPU)包含一组 16 个存储 64 位值的通用目的寄存器。这些寄存器用来存储整数数据和指针。图 3-2 显示了这 16 个寄存器。它们的名字都以%r 开头，不过后面还跟着一些不同的命名规则的名字，这是由于指令集历史演化造成的。最初的 8086 中有 8 个 16 位的寄存器，即图 3-2 中的%ax 到%bp。每个寄存器都有特殊的用途，它们的名字就反映了这些不同的用途。扩展到 IA32 架构时，这些寄存器也扩展成 32 位寄存器，标号从%eax 到%ebp。扩展到 x86-64 后，原来的 8 个寄存器扩展成 64 位，标号从%rax 到%rbp。除此之外，还增加了 8 个新的寄存器，它们的标号是按照新的命名规则制定的：从%r8 到%r15。