分的大小必须与指令最后一个字符('b'，'w'，'l'或'q')指定的大小匹配。大多数情况中，MOV 指令只会更新目的操作数指定的那些寄存器字节或内存位置。唯一的例外是 movl 指令以寄存器作为目的时，它会把该寄存器的高位 4 字节设置为 0。造成这个例外的原因是 x86-64 采用的惯例，即任何为寄存器生成 32 位值的指令都会把该寄存器的高位部分置成 0。

下面的 MOV 指令示例给出了源和目的类型的五种可能的组合。记住，第一个是源操作数，第二个是目的操作数：

```
1    movl $0x4050,%eax          Immediate--Register, 4 bytes
2    movw %bp,%sp               Register--Register,  2 bytes
3    movb (%rdi,%rcx),%al       Memory--Register,    1 byte
4    movb $-17,(%rsp)           Immediate--Memory,   1 byte
5    movq %rax,-12(%rbp)        Register--Memory,    8 bytes
```

图 3-4 中记录的最后一条指令是处理 64 位立即数数据的。常规的 movq 指令只能以表示为 32 位补码数字的立即数作为源操作数，然后把这个值符号扩展得到 64 位的值，放到目的位置。movabsq 指令能够以任意 64 位立即数值作为源操作数，并且只能以寄存器作为目的。

图 3-5 和图 3-6 记录的是两类数据移动指令，在将较小的源值复制到较大的目的时使用。所有这些指令都把数据从源(在寄存器或内存中)复制到目的寄存器。MOVZ 类中的指令把目的中剩余的字节填充为 0，而 MOVS 类中的指令通过符号扩展来填充，把源操作的最高位进行复制。可以观察到，每条指令名字的最后两个字符都是大小指示符：第一个字符指定源的大小，而第二个指明目的的大小。正如看到的那样，这两个类中每个都有三条指令，包括了所有的源大小为 1 个和 2 个字节、目的大小为 2 个和 4 个的情况，当然只考虑目的大于源的情况。

指令	效果	描述
MOVZ　S，R	R←零扩展(S)	以零扩展进行传送
movzbw		将做了零扩展的字节传送到字
movzbl		将做了零扩展的字节传送到双字
movzwl		将做了零扩展的字传送到双字
movzbq		将做了零扩展的字节传送到四字
movzwq		将做了零扩展的字传送到四字

图 3-5　零扩展数据传送指令。这些指令以寄存器或内存地址作为源，以寄存器作为目的

指令	效果	描述
MOVS　S，R	R←符号扩展(S)	传送符号扩展的字节
movsbw		将做了符号扩展的字节传送到字
movsbl		将做了符号扩展的字节传送到双字
movswl		将做了符号扩展的字传送到双字
movsbq		将做了符号扩展的字节传送到四字
movswq		将做了符号扩展的字传送到四字
movslq		将做了符号扩展的双字传送到四字
cltq	%rax←符号扩展(%eax)	把%eax 符号扩展到%rax

图 3-6　符号扩展数据传送指令。MOVS 指令以寄存器或内存地址作为源，以寄存器作为目的。cltq 指令只作用于寄存器%eax 和%rax