地址空间的概念非常通用，并且在很多场合中出现。比如电话号码，在美国和很多其他国家，一个本地电话号码通常是一个7位的数字。因此，电话号码的地址空间是从0 000 000到9 999 999，虽然一些号码并没有被使用，比如以000开头的号码。随着手机、调制解调器和传真机数量的增长，这个空间变得越来越不够用了，从而导致需要使用更多位数的号码。Pentium的I/O端口的地址空间从0到16 383。IPv4的地址是32位的数字，因此它们的地址空间从0到$2^{32}-1$（也有一些保留数字）。

地址空间可以不是数字的。一套“.com”的互联网域名也是地址空间。这个地址空间是由所有包含2~63个字符并且后面跟着“.com”的字符串组成的，组成这些字符串的字符可以是字母、数字和连字符。到现在你应该已经明白地址空间的概念了。它是很简单的。

比较难的是给每个程序一个自己的地址空间，使得一个程序中的地址28所对应的物理地址与另一个程序中的地址28所对应的物理地址不同。下面我们将讨论一个简单的方法，这个方法曾经很常见，但是在有能力把更复杂（而且更好）的机制运用在现代CPU芯片上之后，这个方法就不再使用了。

基址寄存器与界限寄存器

这个简单的解决办法使用一种简单的*动态重定位*。它所做的是简单地把每个进程的地址空间映射到物理内存的不同部分。从CDC 6600（世界上最早的超级计算机）到Intel 8088（原始IBM PC的心脏），所使用的经典办法是给每个CPU配置两个特殊硬件寄存器，通常叫做*基址寄存器*和*界限寄存器*。当使用基址寄存器和界限寄存器时，程序装载到内存中连续的空闲位置且装载期间无须重定位，如图3-2c所示。当一个进程运行时，程序的起始物理地址装载到基址寄存器中，程序的长度装载到界限寄存器中。在图3-2c中，当第一个程序运行时，装载到这些硬件寄存器中的基址和界限值分别是0和16 384。当第二个程序运行时，这些值分别是16 384和32 768。如果第三个16KB的程序被直接装载在第二个程序的地址之上并且运行，这时基址寄存器和界限寄存器里的值会是32 768和16 384。

每次一个进程访问内存，取一条指令，读或写一个数据字，CPU硬件会在把地址发送到内存总线前，自动把基址值加到进程发出的地址值上。同时，它检查程序提供的地址是否等于或大于界限寄存器里的值。如果访问的地址超过了界限，会产生错误并中止访问。这样，对图3-2c中第二个程序的第一条指令，程序执行

```
JMP 28
```

指令，但是硬件把这条指令解释成为

```
JMP 16412
```

所以程序如我们所愿地跳转到了CMP指令。在图3-2c中第二个程序的执行过程中，基址寄存器和界限寄存器的设置如图3-3所示。

使用基址寄存器和界限寄存器是给每个进程提供私有地址空间的非常容易的方法，因为每个内存地址在送到内存之前，都会自动先加上基址寄存器的内容。在很多实际系统中，对基址寄存器和界限寄存器会以一定的方式加以保护，使得只有操作系统可以修改它们。在CDC 6600中就提供了对这些寄存器的保护，但在Intel 8088中则没有，甚至没有界限寄存器。但是，Intel 8088提供了多个基址寄存器，使程序的代码和数据可以被独立地重定位，但是没有提供引用地址越界的预防机制。

使用基址寄存器和界限寄存器重定位的缺点是，每次访问内存都需要进行加法和比较运算。比较可以做得很快，但是加法由于进位传递时间的问题，在没有使用特殊电路的情况下会显得很慢。

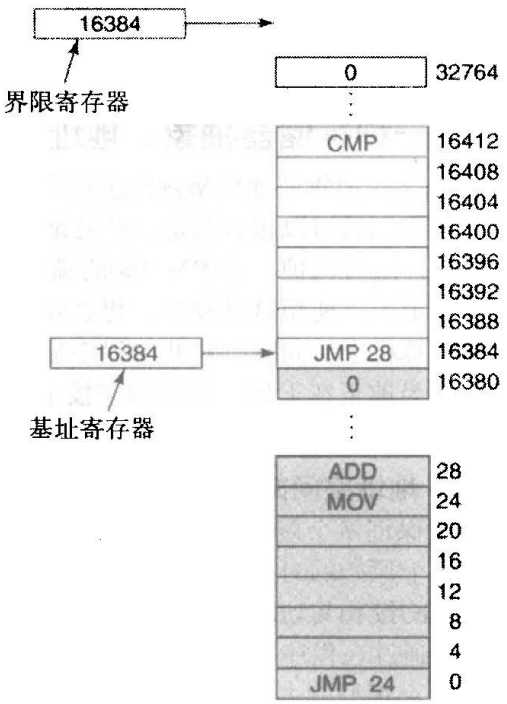

图3-3 基址寄存器和界限寄存器可用于为每个进程提供一个独立的地址空间