行一条指令，这样CPU不会受到存储器的限制），充分大，并且非常便宜。但是目前的技术无法同时满足这三个目标，于是出现了不同的处理方式。存储器系统采用一种分层次的结构，如图1-9所示。顶层的存储器速度较高，容量较小，与底层的存储器相比每位成本较高，其差别往往是十亿数量级。

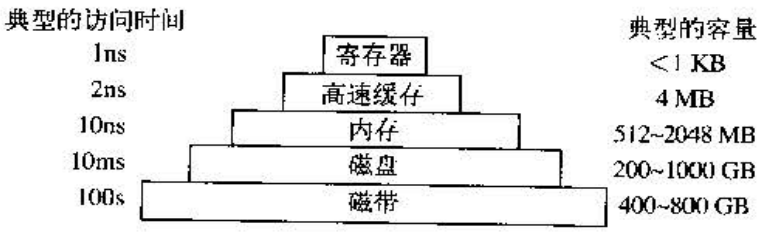

图1-9　典型的存储层次结构，图中的数据是非常粗略的估计

存储器系统的顶层是CPU中的寄存器。它们用与CPU相同的材料制成，所以和CPU一样快。显然，访问它们是没有时延的。其典型的存储容量是，在32位CPU中为32×32 位，而在64位CPU中为64×64 位。在这两种情形下，其存储容量都小于1 KB。程序必须在软件中自行管理这些寄存器（即决定如何使用它们）。

下一层是高速缓存，它多数由硬件控制。主存被分割成高速缓存行（cache line），其典型大小为64个字节，地址0至63对应高速缓存行0，地址64至127对应高速缓存行1，以此类推。最常用的高速缓存行放置在CPU内部或者非常接近CPU的高速缓存中。当某个程序需要读一个存储字时，高速缓存硬件检查所需要的高速缓存行是否在高速缓存中。如果是，称为高速缓存命中，缓存满足了请求，就不需要通过总线把访问请求送往主存。高速缓存命中通常需要两个时钟周期。高速缓存未命中就必须访问内存，这要付出大量的时间代价。由于高速缓存的价格昂贵，所以其大小有限。有些机器具有两级甚至三级高速缓存，每一级高速缓存比前一级慢且容量更大。

缓存在计算机科学的许多领域中起着重要的作用，并不仅仅只是RAM的缓存行。只要存在大量的资源可以划分为小的部分，那么，这些资源中的某些部分就会比其他部分更频繁地得到使用，通常缓存的使用会带来性能上的改善。操作系统一直在使用缓存。例如，多数操作系统在内存中保留频繁使用的文件（的一部分），以避免从磁盘中重复地调取这些文件。相似地，类似于

```
/home/ast/projects/minix3/src/kernel/clock.c
```

的长路径名转换成文件所在的磁盘地址的结果，也可以放入缓存，以避免重复寻找地址。还有，当一个Web 页面（URL）的地址转换为网络地址（IP地址）后，这个转换结果也可以缓存起来以供将来使用。还有许多其他的类似的应用。

在任何缓存系统中，都有若干需要尽快考虑的问题，包括：

1) 何时把一个新的内容放入缓存。

2) 把新内容放在缓存的哪一行上。

3) 在需要时，应该把哪个内容从缓存中移走。

4) 应该把新移走的内容放在某个较大存储器的何处。

并不是每个问题的解决方案都符合每种缓存处理。对于CPU缓存中的主存缓存行，每当有缓存未命中时，就会调入新的内容。通常通过所引用内存地址的高位计算应该使用的缓存行。例如，对于64字节的4096缓存行，以及32位地址，其中6～17位用来定位缓存行，而0～5位则用来确定缓存行中的字节。在这个例子中，被移走内容的位置就是新数据要进入的位置，但是在有的系统中未必是这样。最后，当将一个缓存行的内容重写进主存时（该内容被缓存后，可能会被修改），通过该地址来惟一确定需重写的主存位置。

缓存是一种好方法，所以现代CPU 中设计了两个缓存。第一级或称为L1缓存总是在CPU中，通常用来将已解码的指令调入CPU的执行引擎。对于那些频繁使用的数据字，多数芯片安排有第二个L1缓存。典型的L1缓存大小为16KB。另外，往往还设计有二级缓存，称为L2缓存，用来存放近来所使用过若干兆字节的内存字。L1和L2缓存之间的差别在于时序。对L1缓存的访问，不存在任何延时，而对L2缓存的访问，则会延时1或2个时钟周期。

在多核芯片中。设计师必须确定缓存的位置。在图1-8a中，一个L2缓存被所有的核共享。Intel多核芯片采用了这个方法。相反，在图1-8b中，每个核有其自己的L2缓存。AMD采用这个方法。不过每种策略都有自己的优缺点。例如，Intel的共享L2缓存需要有一种更复杂的缓存控制器，而AMD的方式在设法保持L2缓存一致性上存在困难。