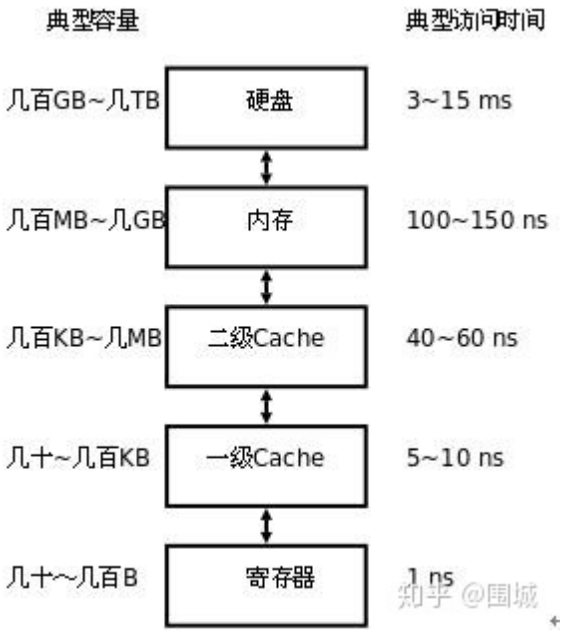
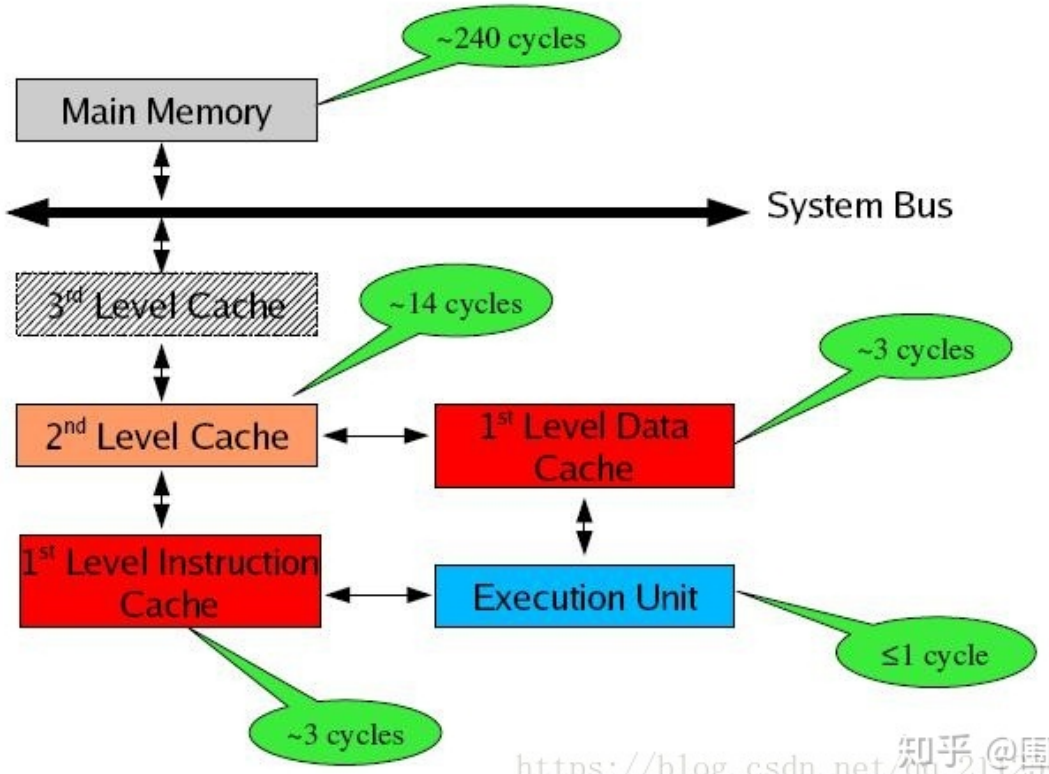
计算机缓存Cache以及Cache Line详解

1.计算机存储体系简介

存储器是分层次的，离CPU越近的存储器，速度越快，每字节的成本越高，同时容量也因此越小。寄存器速度最快，离CPU最近，成本最高，所以个数容量有限，其次是高速缓存（缓存也是分级，有L1，L2等缓存），再次是主存（普通内存），再次是本地磁盘。



寄存器的速度最快，可以在一个时钟周期内访问，其次是高速缓存，可以在几个时钟周期内访问，普通内存可以在几十个或几百个时钟周期内访问。



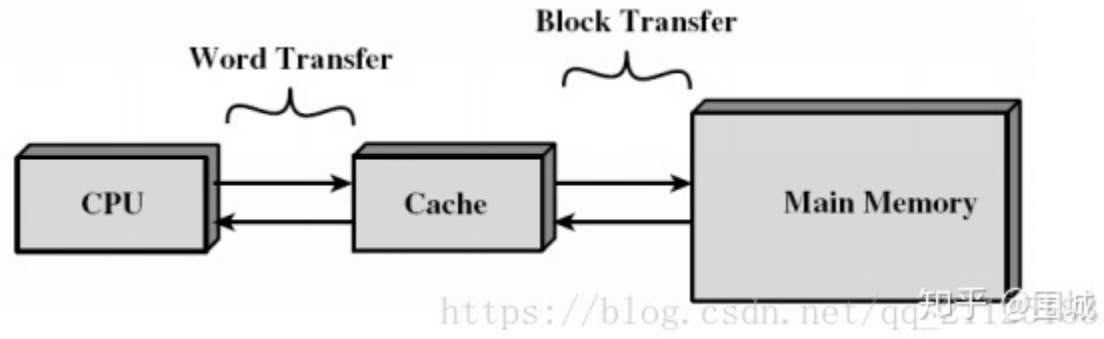
存储器分级，利用的是局部性原理。我们可以以经典的阅读书籍为例。我在读的书，捧在手里（寄存器），我最近频繁阅读的书，放在书桌上（缓存），随时取来读。当然书桌上只能放有限几本书。我更多的书在书架上（内存）。如果书架上没有的书，就去图书馆（磁盘）。我要读的书如果手里没有，那么去书桌上找，如果书桌上没有，去书架上找，如果书架上没有去图书馆去找。可以对应寄存器没有，则从缓存中取，缓存中没有，则从内存中取到缓存，如果内存中没有，则先从磁盘读入内存，再读入缓存，再读入寄存器。

2.计算机缓存 Cache

本系列的文章重点介绍缓存cache。了解如何获取cache的参数，了解缓存的组织结构。

2.1 Cache 概述

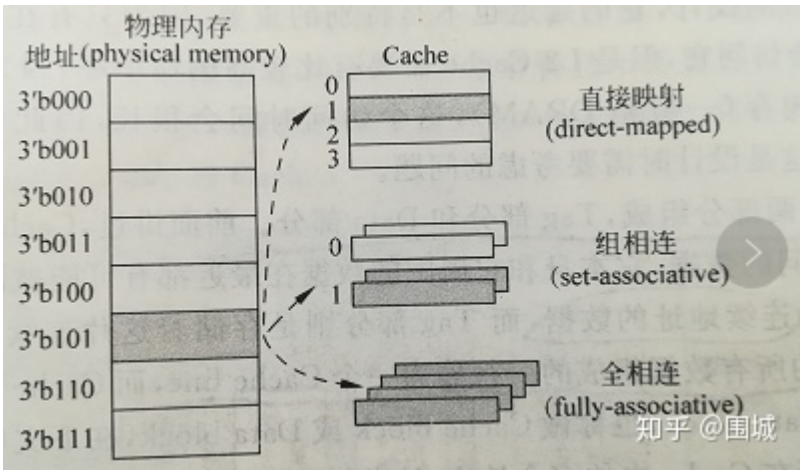
cache，中译名高速缓冲存储器，其作用是为了更好的利用局部性原理，减少CPU访问主存的次数。简单地说，CPU正在访问的指令和数据，其可能会被以后多次访问到，或者是该指令和数据附近的内存区域，也可能会被多次访问。因此，第一次访问这一块区域时，将其复制到cache中，以后访问该区域的指令或者数据时，就不用再从主存中取出。



cache分成多个组，每个组分成多个行，linesize是cache的基本单位，从主存向cache迁移数据都是按照linesize为单位替换的。比如linesize为32Byte，那么迁移必须一次迁移32Byte到cache。 这个linesize比较容易理解，想想我们前面书的例子，我们从书架往书桌搬书必须以书为单位，肯定不能把书撕了以页为单位。书就是linesize。当然了现实生活中每本书页数不同，但是同个cache的linesize总是相同的。

所谓8路组相连（ 8-way set associative）的含义是指，每个组里面有8个行。

我们知道，cache的容量要远远小于主存，主存和cache肯定不是一一对应的，那么主存中的地址和cache的映射关系是怎样的呢？
拿到一个地址，首先是映射到一个组里面去。如何映射？取内存地址的中间几位来映射。
举例来说，data cache: 32-KB, 8-way set associative, 64-byte line size
Cache总大小为32KB，8路组相连（每组有8个line），每个line的大小linesize为64Byte,OK，我们可以很轻易的算出一共有32K/8/64=64 个组。
对于32位的内存地址，每个line有2^6 = 64Byte，所以地址的【0，5】区分line中的那个字节。一共有64个组。我们取内存地址中间6为来hash查找地址属于那个组。即内存地址的【6，11】位来确定属于64组的哪一个组。组确定了之后，【12，31】的内存地址与组中8个line挨个比对，如果【12，31】为与某个line一致，并且这个line为有效，那么缓存命中。
OK，cache分成三类，
1 **直接映射高速缓存**，这个简单，即每个组只有一个line，选中组之后不需要和组中的每个line比对，因为只有一个line。
2 **组相联高速缓存**，这个就是我们前面介绍的cache。 S个组，每个组E个line。
3 **全相联高速缓存**，这个简单，只有一个组，就是全相联。不用hash来确定组，直接挨个比对高位地址，来确定是否命中。可以想见这种方式不适合大的缓存。想想看，如果4M 的大缓存　linesize为32Byte，采用全相联的话，就意味着4*1024*1024/32 = 128K 个line挨个比较，来确定是否命中，这是多要命的事情。高速缓存立马成了低速缓存了。



描述一个cache需要以下参数 ：

1 cache分级，L1 cache, L2 cache, L3 cache,级别越低，离 c p u 越近

2 cache的容量

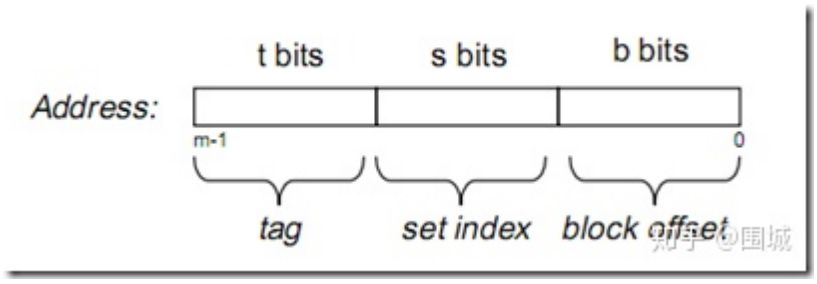
3 cache的linesize

4 cache 每组的行个数.

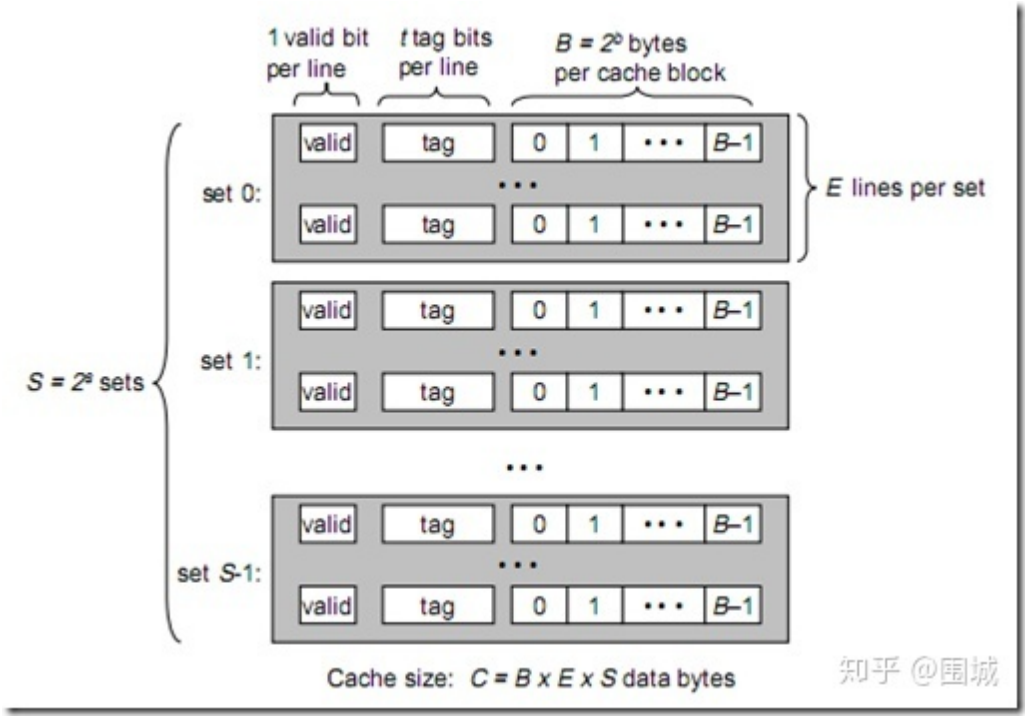
2.2 Cache 结构

假设内存容量为M，内存地址为m位：那么寻址范围为000…00~FFF…F(m位)

倘若把内存地址分为以下三个区间：



tag, set index, block offset三个区间有什么用呢？再来看看Cache的逻辑结构吧：



将此图与上图做对比，可以得出各参数如下：

B = 2^b

S = 2^s

现在来解释一下各个参数的意义：

一个cache被分为S个组，每个组有E个cacheline，而一个cacheline中，有B个存储单元，现代处理器中，这个存储单元一般是以字节(通常8个位)为单位的，也是最小的寻址单元。因此，在一个内存地址中，中间的s位决定了该单元被映射到哪一组，而最低的b位决定了该单元在cacheline中的偏移量。valid通常是一位，代表该cacheline是否是有效的(当该cacheline不存在内存映射时，当然是无效的)。tag就是内存地址的高t位，因为可能会有多个内存地址映射到同一个cacheline中，所以该位是用来校验该cacheline是否是CPU要访问的内存单元。

当tag和valid校验成功是，我们称为cache命中，这时只要将cache中的单元取出，放入CPU寄存器中即可。

当tag或valid校验失败的时候，就说明要访问的内存单元(也可能是连续的一些单元，如int占4个字节，double占8个字节)并不在cache中，这时就需要去内存中取了，这就是cache不命中的情况(cache miss)。当不命中的情况发生时，系统就会从内存中取得该单元，将其装入cache中，与此同时也放入CPU寄存器中，等待下一步处理。注意，以下这一点对理解linux cache机制非常重要：

3.计算机缓存行 ChaceLine

高速缓存其实就是一组称之为缓存行(cache line)的固定大小的数据块，其大小是以突发读或者突发写周期的大小为基础的。

每个高速缓存行完全是在一个突发读操作周期中进行填充或者下载的。即使处理器只存取一个字节的存储器，高速缓存控制器也启动整个存取器访问周期并请求整个数据块。缓存行第一个字节的地址总是突发周

期尺寸的倍数。缓存行的起始位置总是与突发周期的开头保持一致。

当从内存中取单元到cache中时，会一次取一个cacheline大小的内存区域到cache中，然后存进相应的cacheline中。

例如：我们要取地址 (t, s, b) 内存单元，发生了cache miss，那么系统会取 (t, s, 00…000) 到 (t, s, FF…FFF)的内存单元，将其放入相应的cacheline中。

下面看看cache的映射机制：

> 当E=1时， 每组只有一个cacheline。那么相隔2^(s+b)个单元的2个内存单元，会被映射到同一个cacheline中。(好好想想为什么?)
> 当1<E<C/B时，每组有E个cacheline，不同的地址，只要中间s位相同，那么就会被映射到同一组中，同一组中被映射到哪个cacheline中是依赖于替换算法的。
> 当E=C/B，此时S=1，每个内存单元都能映射到任意的cacheline。带有这样cache的处理器几乎没有，因为这种映射机制需要昂贵复杂的硬件来支持。

不管哪种映射，只要发生了cache miss，那么必定会有一个cacheline大小的内存区域，被取到cache中相应的cacheline。

现代处理器，一般将cache分为2~3级，L1, L2, L3。L1一般为CPU专有，不在多个CPU中共享。L2 cache一般是多个CPU共享的，也可能装在主板上。L1 cache还可能分为instruction cache, data cache. 这样CPU能同时取指令和数据。