第4季

cache基础知识

本节课主要内容

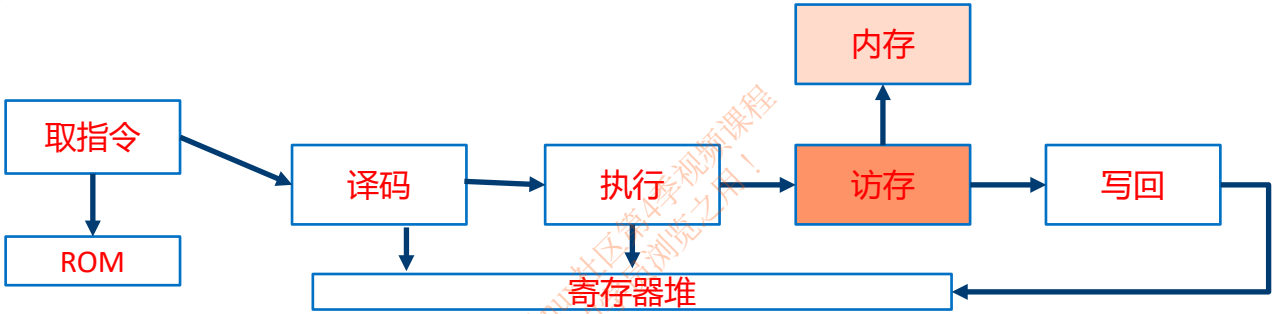
- 本章主要内容
  - Cache基础知识
  - Cache工作原理
  - Cache映射方式
  - 虚拟cache与物理cache
  - 重名和同名问题
  - 缓存策略

技术手册：

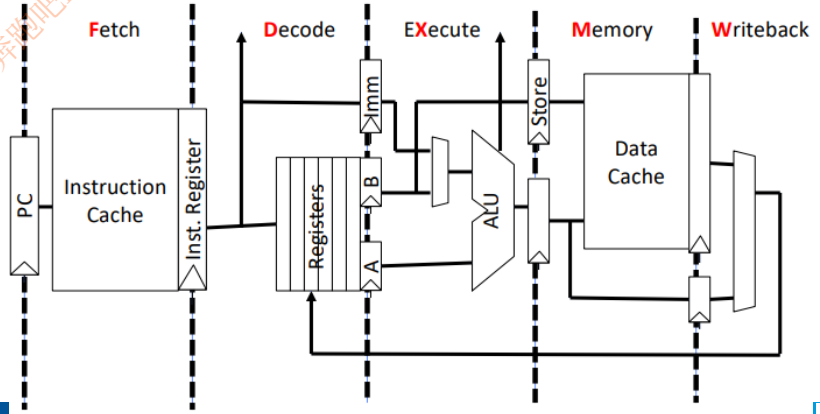
1. The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Document Version 20211203
2. SiFive U74-MC Core Complex Manual, 21G2.01.00

本节课主要讲解书上第11章内容

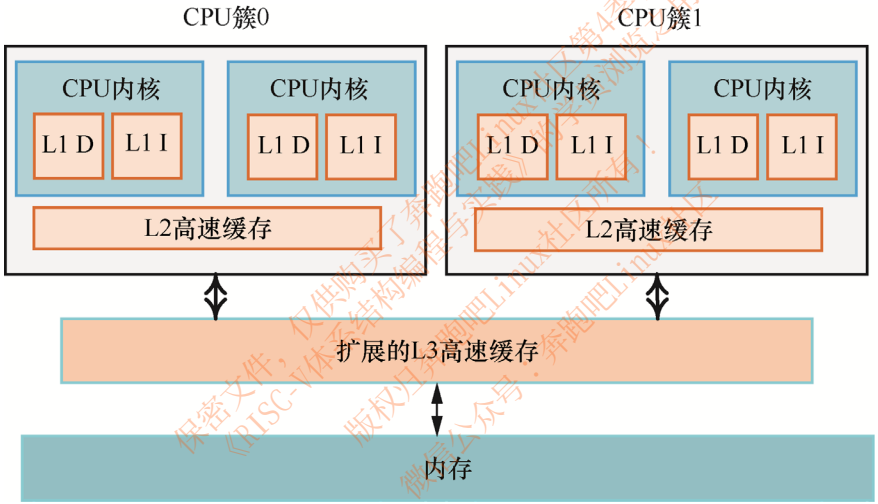
为什么需要cache?



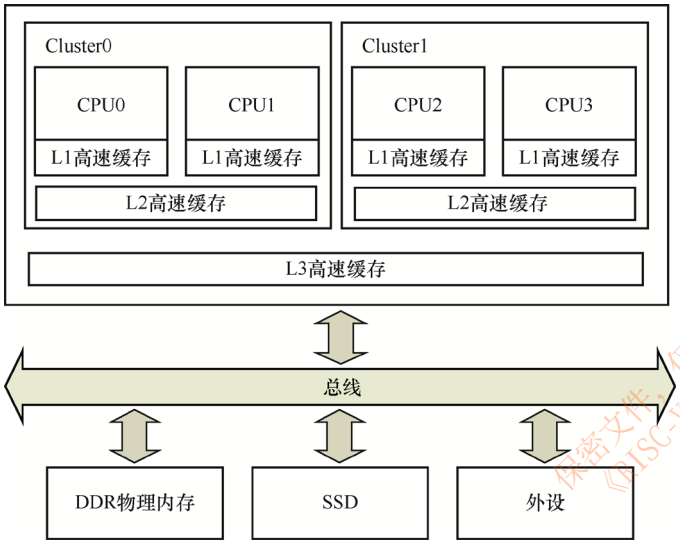
解决CPU访问速度与内存访问速度的不匹配问题



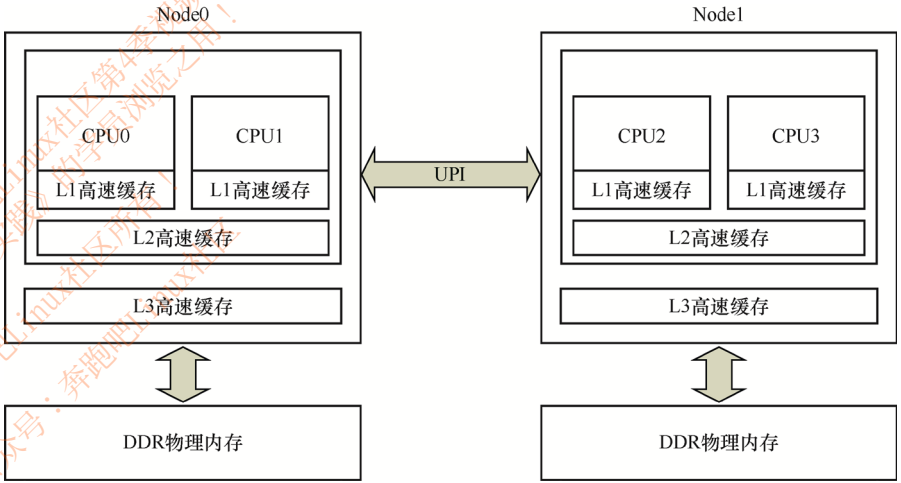
经典处理器的cache架构



cache的访问延时



UMA体系结构

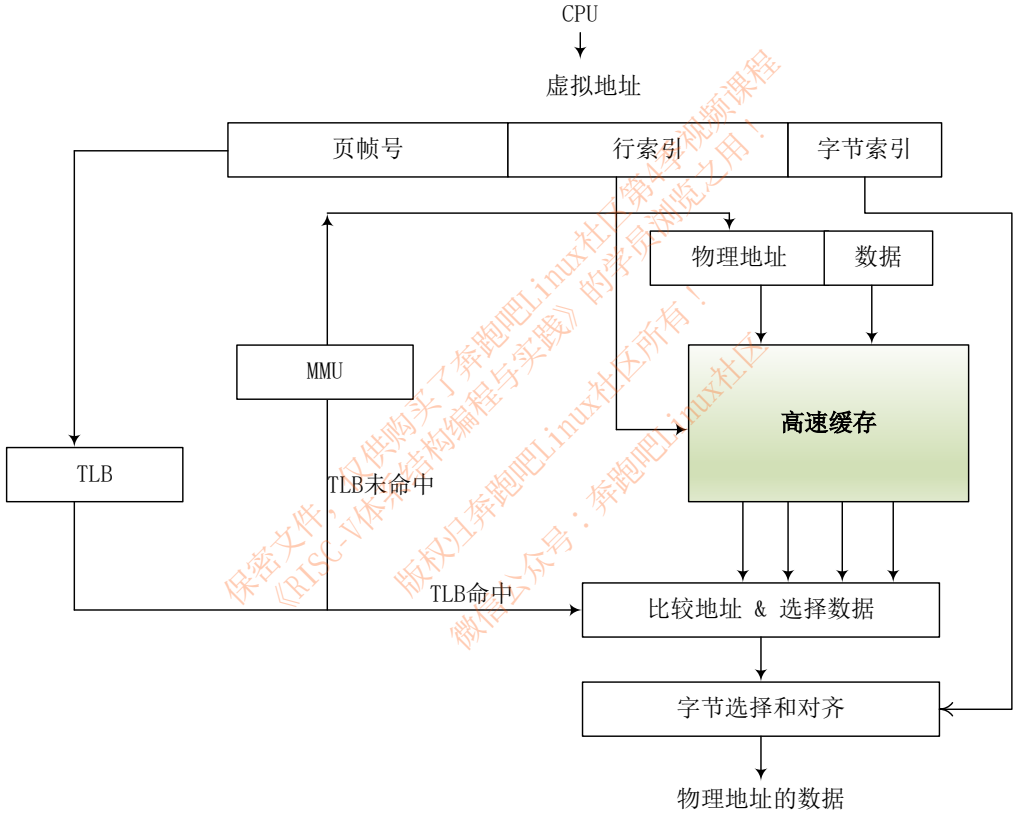


NUMA体系结构

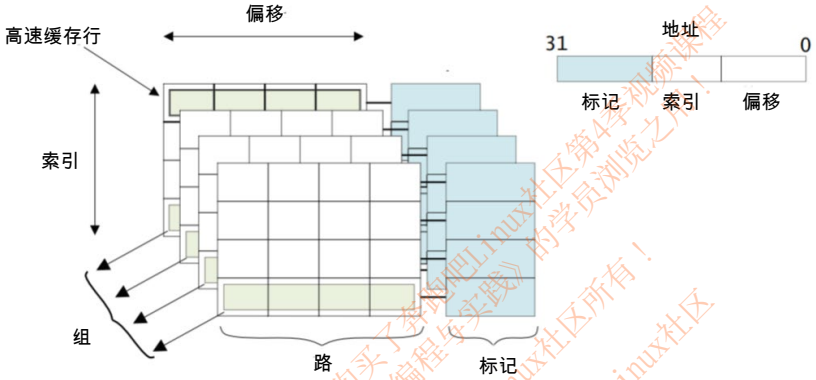
表 11.1　某服务器 CPU 访问各级内存设备的延时

访问类型	访问延时
L1 高速缓存命中	约 4 个时钟周期
L2 高速缓存命中	约 10 个时钟周期
L3 高速缓存命中（高速缓存行没有共享）	约 40 个时钟周期
L3 高速缓存命中（和其他 CPU 共享高速缓存行）	约 65 个时钟周期
L3 高速缓存命中（高速缓存行被其他 CPU 修改过）	约 75 个时钟周期
访问远端的 L3 高速缓存	100～300 个时钟周期
访问本地 DDR 物理内存	约 60 ns
访问远端内存节点的 DDR 物理内存	约 100 ns

经典的cache架构



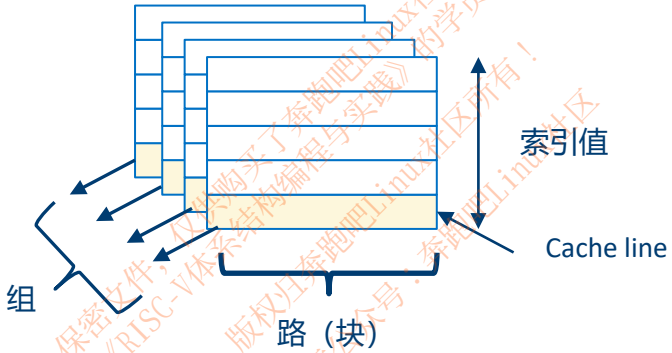
Cache内部架构图

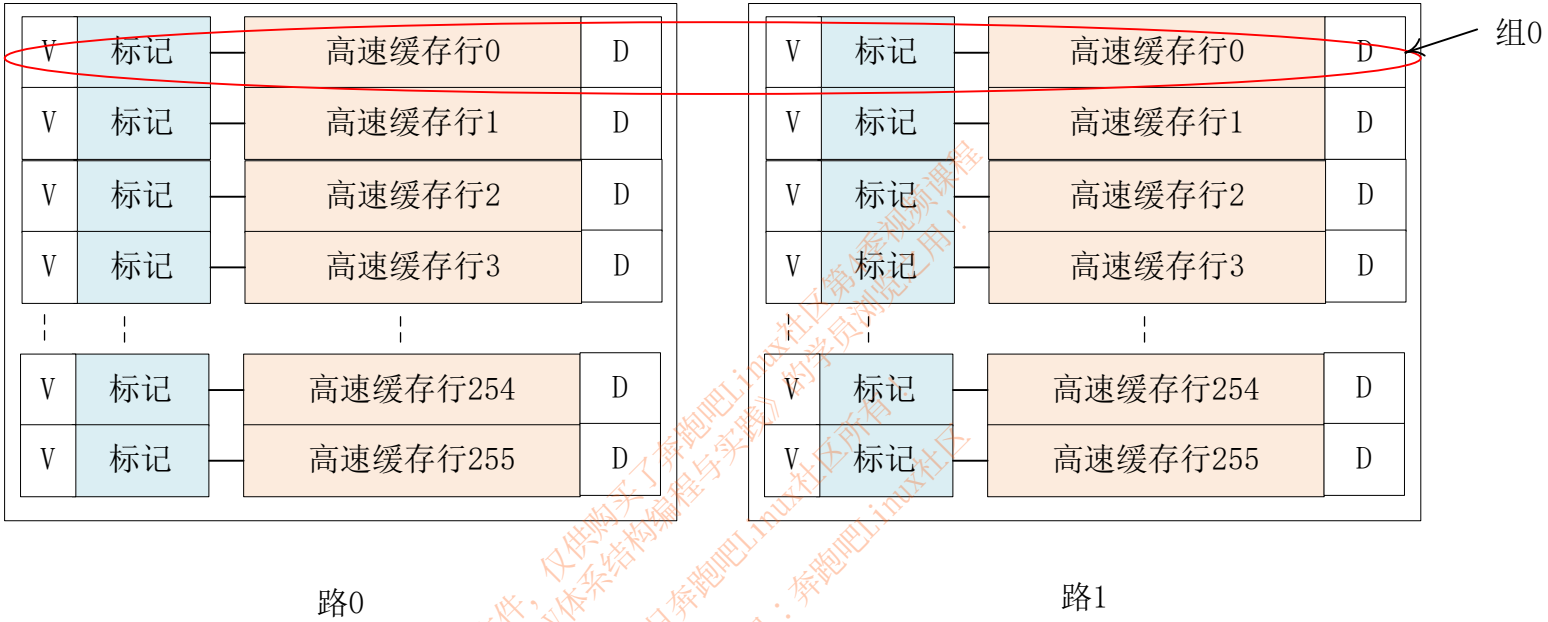


- 高速缓存行：高速缓存中最小的访问单元
- 索引（index）域：用于索引和查找是在高速缓存中的哪一行。
- 标记（tag）：高速缓存地址编码的一部分，通常是高速缓存地址的高位部分，用来判断高速缓存行缓存数据的地址是否和处理器寻址地址一致。
- 偏移（offset）：高速缓存行中的偏移。处理器可以按字（word）或者字节（Byte）来寻址高速缓存行的内容。

- 组（set）：相同索引域的高速缓存行组成一个组。
- 路（way）：在组相联的高速缓存中，高速缓存被分成大小相同的几个块。

访问cache的地址

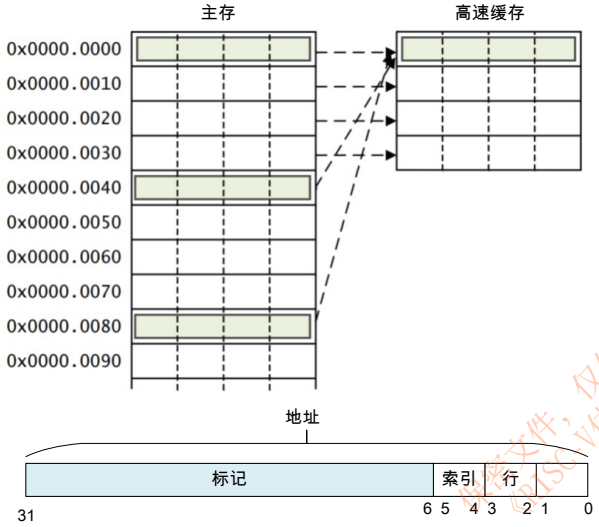




一个2路组相联高速缓存，它一共由2路组成，每一路都有256个高速缓存行。路0和路1中相同索引号对应的高速缓存行组成一组，例如，路0中的高速缓存行0和路1中的高速缓存行0构成组0，它一共有256组。

Cache映射方式 – 直接映射

- 当每个组只有一行高速缓存行时，称为直接映射高速缓存（direct-mapping）

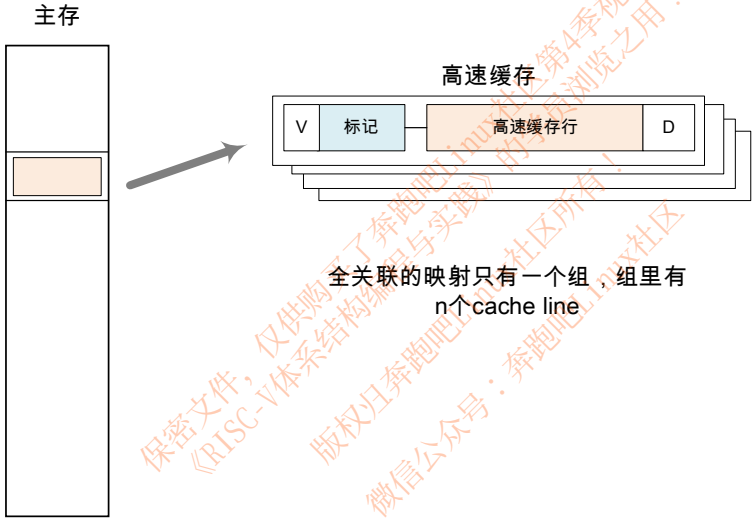


例子：假设在下面的代码片段中，result、data1和data2分别指向0x00、0x40和0x80地址，它们都会使用同一个高速缓存行。

```
void add_array(int *data1, int *data2, int *result, int size)
{
  int i;
  for (i=0 ; i<size ; i++) {
      result[i] = data1[i] + data2[i];
  }
}
```

Cache映射方式 – 全关联

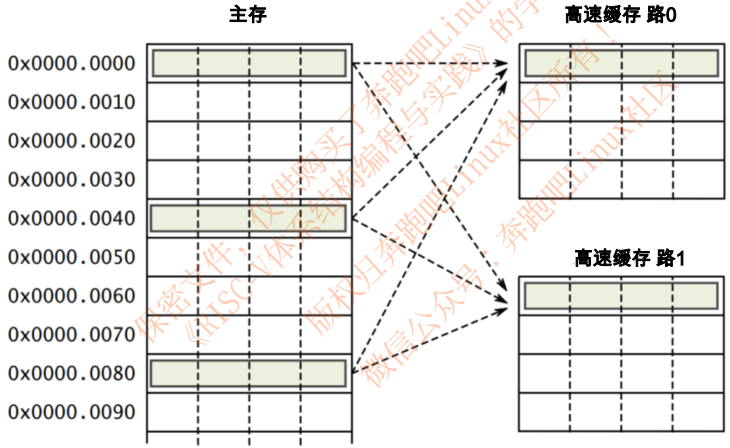
- 当cache只有一个组，即主存中只有一个地址与n个cache line对应，称为全关联



Cache映射方式 –组相联

- 直接映射和全关联直接的做一个折中 - 组相联（set associative）

  - ✓ 一个2路组相联的高速缓存为例，每一路包括4个高速缓存行，那么每个组有两个高速缓存行可以提供高速缓存行替换。
  - ✓ 减小高速缓存颠簸



2路组相联的映射关系

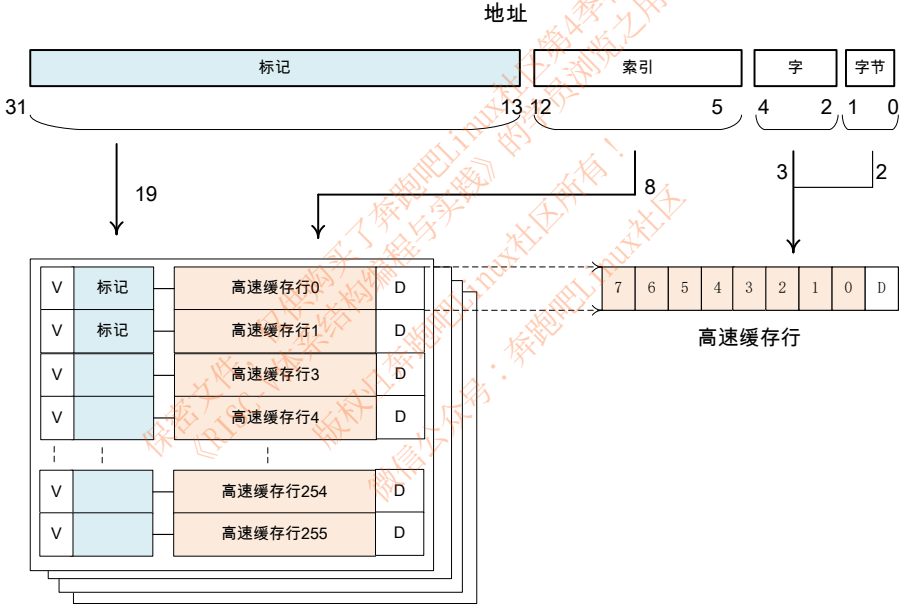
- 举个例子：32KB大小的4路组相联的cache，其中cache line为32字节，请画出cache的映射结构图

高速缓存的总大小为32KB，并且是4路（way），所以每一路的大小为8KB：

way_size = 32 / 4 = 8（KB）

高速缓存行的大小为32字节，所以每一路包含的高速缓存行数量为：

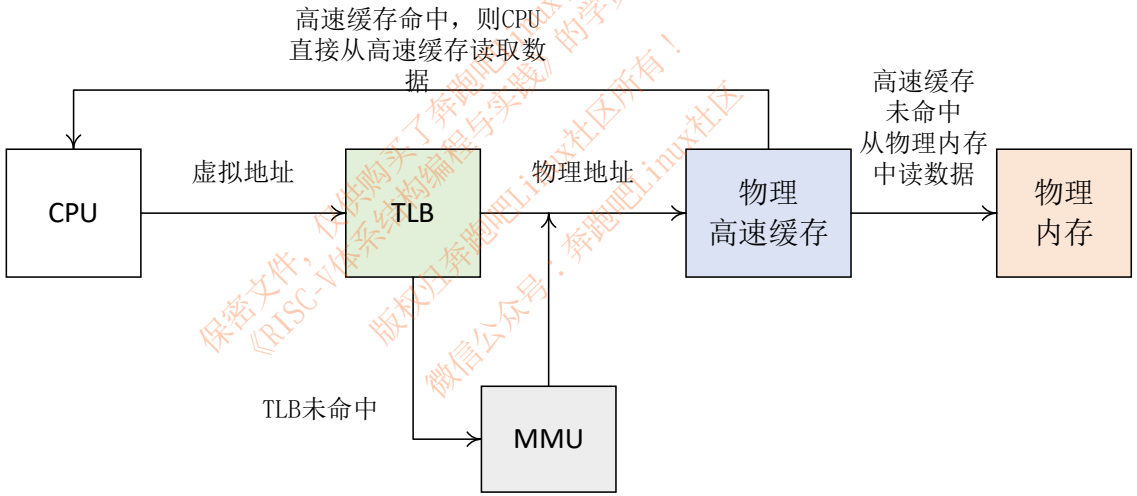
num_cache_line = 8KB/32B = 256



V=有效位　D=脏位

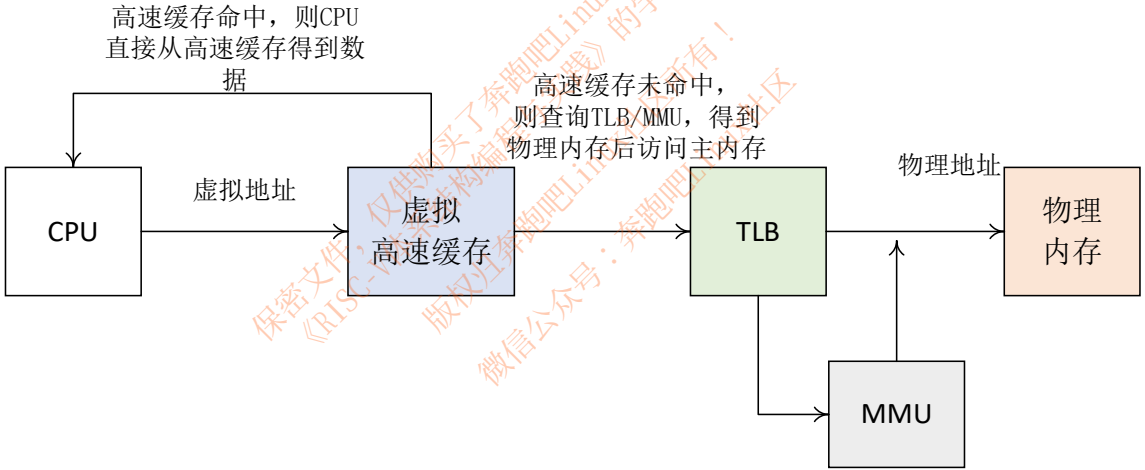
物理高速缓存

- 物理高速缓存：当处理器查询MMU和TLB得到物理地址之后，使用物理地址去查询高速缓存。
- 缺点：处理器在查询MMU和TLB后才能访问高速缓存，增加了流水线的延迟



虚拟高速缓存

- 虚拟高速缓存：处理器使用虚拟地址来寻址高速缓存。
- 缺点：会引入不少问题：
  - ✓ 重名（Aliasing）问题
  - ✓ 同名（Homonyms）问题

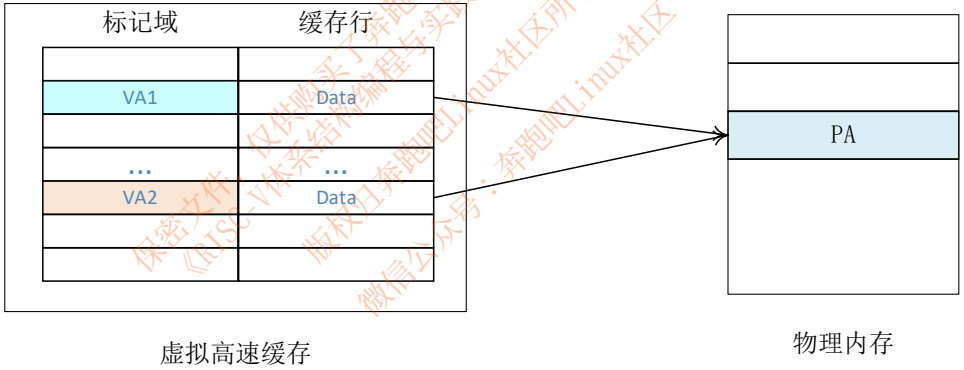


重名（Aliasing）问题

- 在操作系统中，多个不同的虚拟地址有可能映射相同的物理地址。由于采用虚拟高速缓存架构，那么这些不同的虚拟地址会占用高速缓存中不同的高速缓存行，但是它们对应的是相同的物理地址。

- 举个例子：VA1和VA2都映射到PA，在cache中有两个cache line缓存了VA1和VA2。

  当程序往VA1写入数据时，VA1对应的高速缓存行以及PA的内容会被更改，但是VA2还保存着旧数据。这样一个物理地址在虚拟高速缓存中就保存了两份数据，这样会产生歧义。



同名（Homonyms）问题

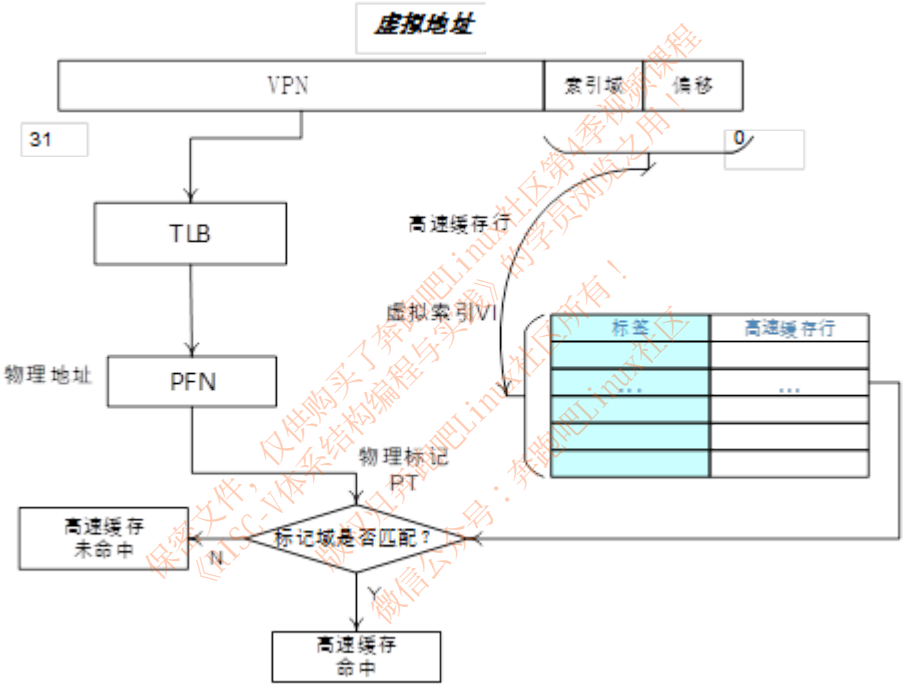
- 相同的虚拟地址对应着不同的物理地址，因为操作系统中不同的进程会存在很多相同的虚拟地址，而这些相同的虚拟地址在经过MMU转换后得到不同的物理地址，这就产生了同名问题。

- 同名问题最常见的地方是进程切换。当一个进程切换到另外一个进程时，新进程使用虚拟地址来访问高速缓存的话，新进程会访问到旧进程遗留下来的高速缓存，这些高速缓存数据对于新进程来说是错误和没用的。解决办法是在进程切换时把旧进程遗留下来的高速缓存都置为无效，这样就能保证新进程执行时得到一个干净的虚拟高速缓存。

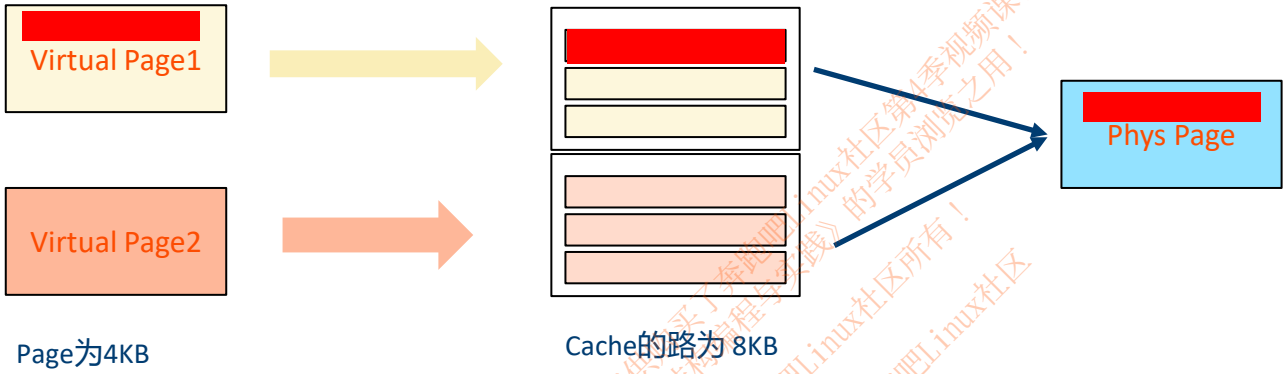
高速缓存分类

- ✓ VIVT（Virtual Index Virtual Tag）：使用虚拟地址的索引域和虚拟地址的标记域，相当于是虚拟高速缓存。
- ✓ PIPT（Physical Index Physical Tag）：使用物理地址索引域和物理地址的标记域，相当于是物理高速缓存。
- ✓ VIPT（Virtual Index Physical Tag）：使用虚拟地址索引域和物理地址的标记域。

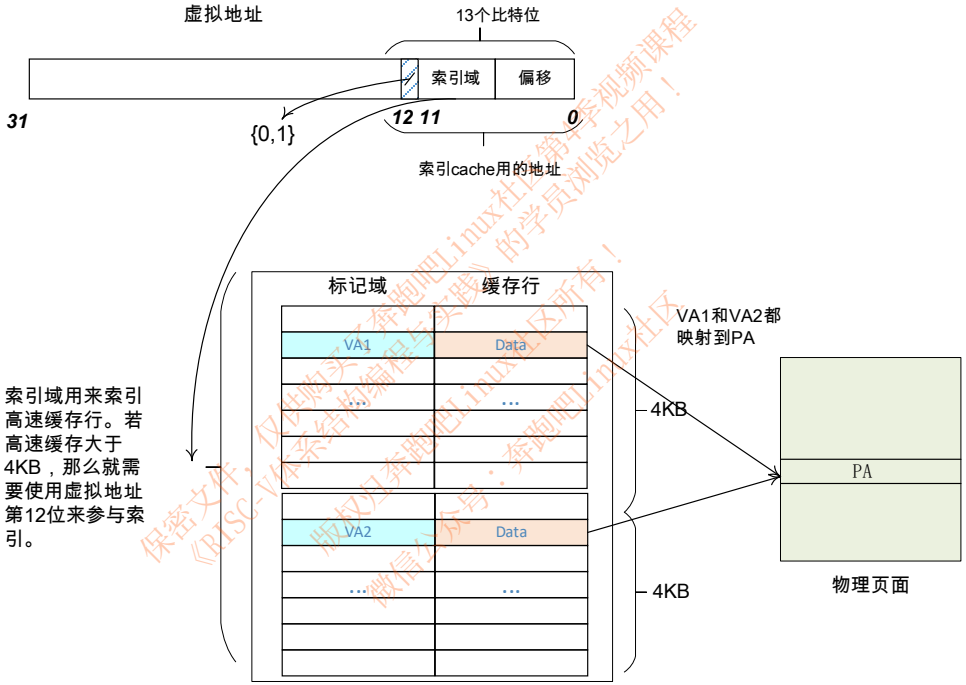
VIPT工作过程



VIPT别名问题

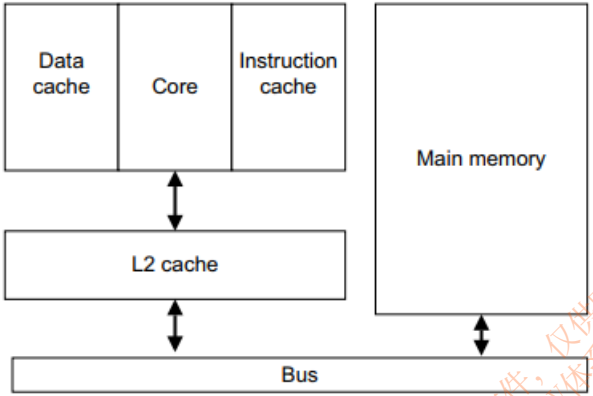


VIPT别名问题

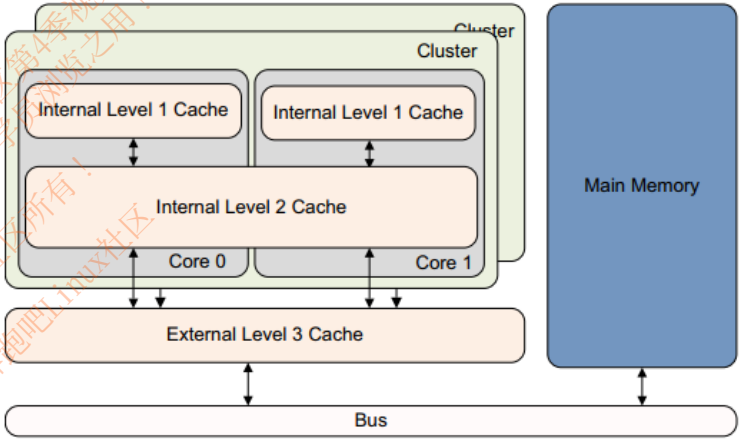


高速缓存的路（way）大小为8KB

Cache层级



两级cache的系统



三级cache的系统

Cache的策略（Cache policies）

- Cache策略包括：
  - Cacheable/non-cacheable
  - Cacheable细分：
    - ✓ Read/write-allocate
    - ✓ Write-Back cacheable、write-through cacheable
    - ✓ Shareability

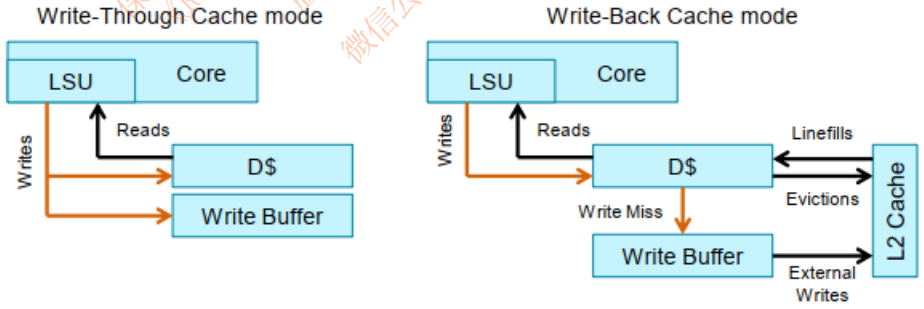
Cache的策略（Cache policies）

- Cache分配策略
  - ✓ Write allocation: 当write miss的时候才分配一个新的cache line
  - ✓ Read allocation：当read miss的时候才分配一个新的cache line

- Cache回写策略：
  - ✓ Write-back: 回写操作仅仅更新到cache，并没有马上更新会内存。(cache line is marked as dirty)
  - ✓ Write through：回写操作会直接更新cache和内存。

Write Back和Write Through

- WT写直通模式。
  - ✓ 进行写操作时，数据同时写入当前的高速缓存、下一级高速缓存或主存储器中。
  - ✓ 直写模式可以降低高速缓存一致性的实现难度，其最大的缺点是消耗比较多的总线带宽。
- 有些处理器把WT模式看成Non-cacheable，如Arm的Cortex-A72
- WB模式回写模式
  - ✓ 在进行写操作时，数据直接写入当前高速缓存，而不会继续传递，当该高速缓存行被替换出去时，被改写的数据才会更新到下一级高速缓存或主存储器中。该策略增加了高速缓存一致性的实现难度，但是有效降低了总线带宽需求。
  - ✓ Cache line变成Dirty data



CMO cache维护指令

- RISC-V中的CMO（Cache Management Operation，高速缓存管理操作）扩展指令集提供了对高速缓存进行管理的指令。
- 高速缓存的管理：
  - ✓ 失效（invalidate）操作：使某个高速缓存行失效，并丢弃高速缓存上的数据。
  - ✓ 清理（clean）操作：把标记为脏的某个高速缓存行写回下一级高速缓存中或者内存中，然后清除高速缓存行中的脏位。这使高速缓存行的内容与下一级高速缓存或者内存中的数据保持一致。
  - ✓ 冲刷（flush）操作：这是一种混合的操作，即清理并使其失效（clean and invalidate），它会先执行清理操作，然后使高速缓存行失效。
  - ✓ 清零（zero）操作

- CBO.CLEAN指令：对一个指定地址的cache line执行清理操作

```
cbo.clean rs1
```

- CBO.FLUSH指令：对一个指定地址的cache line执行冲刷操作

```
cbo.flush  rs1
```

- CBO.INVAL：使一个指定地址的cache line无效

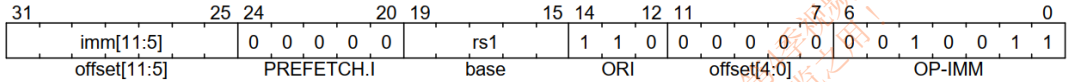
```
cbo.inval  rs1
```

- CBO.ZERO：对一个指定地址的cache line执行清零操作

```
cbo.zero  rs1
```

- PREFETCH.I：用于预取指令cache的数据

```
prefetch.i offset(base)
```



- PREFETCH.R：用于在读操作中预取数据cache的数据

```
prefetch.r offset(base)
```

- PREFETCH.W：用于在写操作中预取数据cache的数据

```
prefetch.w offset(base)
```

RISC-V架构与cache相关的内容 1

- RISC-V处理器中的 内存区域属性 与 cache实现，与具体CPU core RTL的实现相关
  - ✓ 平头哥C910处理器支持Cacheable，bufferable，Shareable的页面，在PTE页表项中设置
  - ✓ U74通过PMA来设定内存区域是否需要cacheable。
    - Memory Port： Atomics+LR/SC, Data Cacheable, Instruction Cacheable, Instruction Speculation
    - Peripheral Port： Atomics, Instruction Cacheable
    - System port： Instruction Cacheable

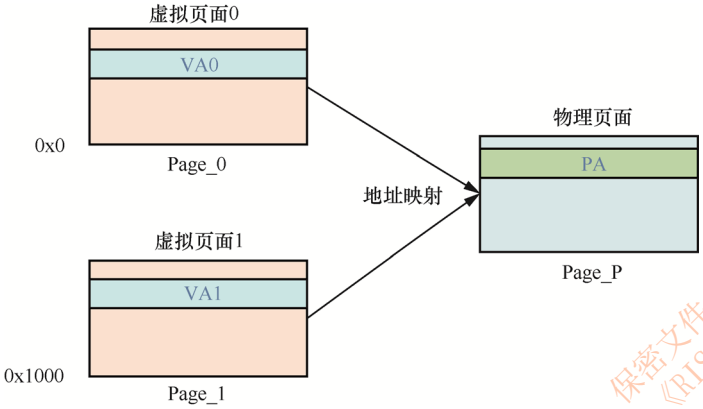
RISC-V架构与cache相关的内容 2

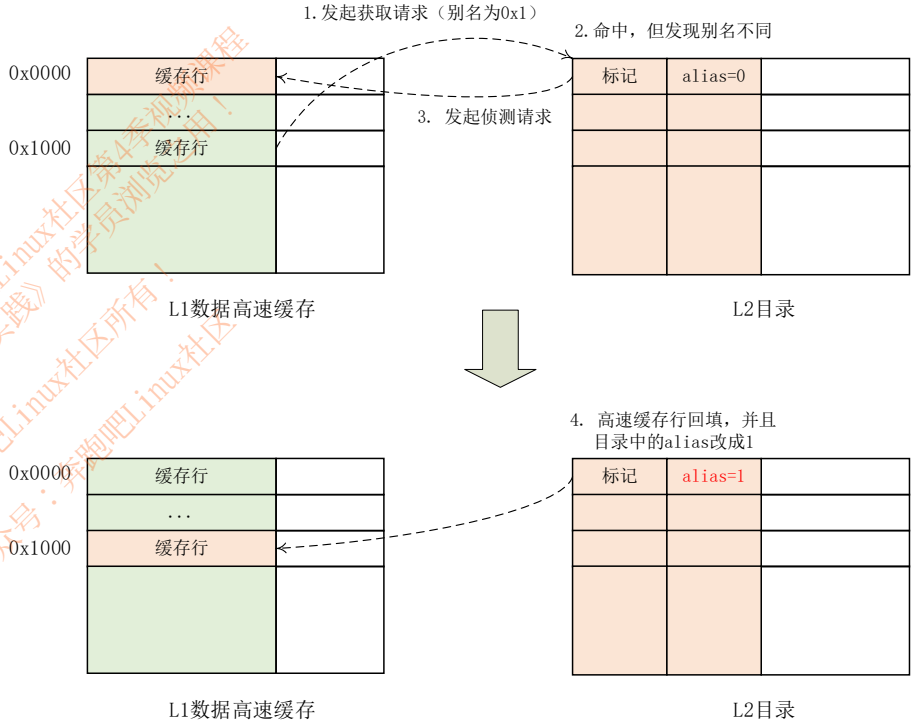
- RISC-V体系结构规范约定处理器不允许出现类似于高速缓存别名（aliasing）的缓存一致性问题

> *Some ISAs architecturally expose* virtually indexed, physically tagged *caches, in that accesses to the same physical address via different virtual addresses might not be coherent unless the virtual addresses lie within the same cache set. Implicitly, this specification does not permit such behavior to be architecturally exposed.*

- 香山处理器在RTL里面解决了VIPT别名问题
  - ✓ 由L2高速缓存保证一个物理缓存行在上层高速缓存中最多只有一种别名位，并且回填正确的数据到上层高速缓存行中。

香山处理器解决VIPT别名问题





RISC-V架构与cache相关的内容 3

- RISC-V架构手册里没有约定缓存一致性是 软件实现还是 硬件实现，不过一般SoC都是采用硬件实现。