MIPS Cache

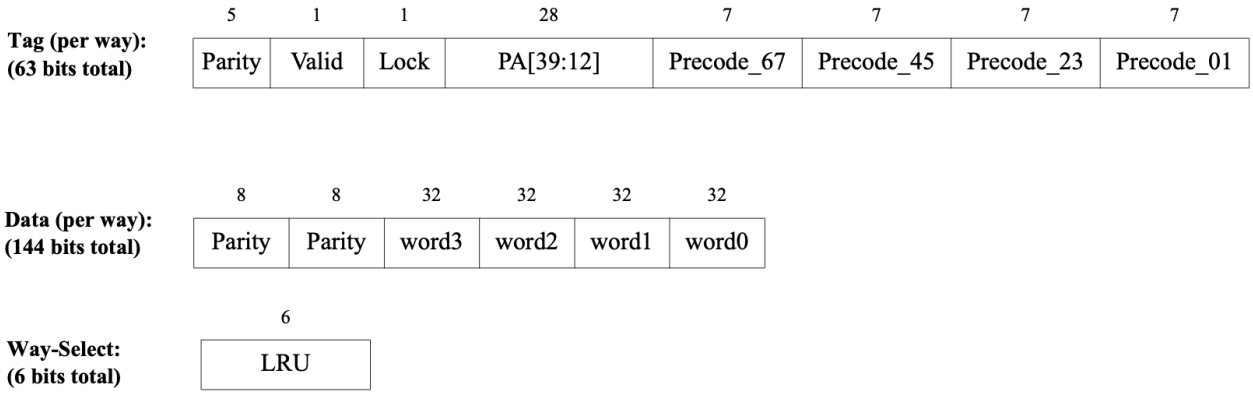
P6600核心包含三个缓存：L1指令缓存、L1数据缓存和共享的L2缓存。

L1指令缓存通过两个64位数据路径连接到指令获取单元（IFU），允许每个周期最多进行四次指令获取。L1数据缓存包含两个64位数据路径，允许每个周期最多进行两次数据读/写操作。L2缓存嵌入在一致性管理器（CM2）内，并通过可配置的128位或256位OCP接口与外部存储器进行通信。

Attribute	L1 Instruction Cache	L1 Data Cache	L2 Cache
Size[1]	32 KB or 64 KB	32 KB or 64 KB	512 KB 1 MB, 2 MB, 4 MB, or 8 MB
Line Size	32-byte	32-byte	32-byte
Number of Cache Sets	256 or 512	256 or 512	2048, 4096, 8192, 16384, or 32768
Associativity	4 way	4 way	8 way

L1 ICache

L1指令缓存由三个主要部分组成：**标签、数据和路径选择**。它使用**VIPT**。每个缓存集合有4路信息，并使用一个单一的有效位来标识整个32字节的数据线是否在缓存中。缓存还使用最近最少使用（LRU）算法来决定替换哪一路的数据。



Cache Aliasing

在P6600核心中，指令缓存是**VIPT**。由于单一路的Cache的大小大于最小的TLB页面大小，因此存在虚拟别名（virtual aliasing）的可能性。这意味着如果一个物理地址可能存在于缓存内的多个索引中。**这种虚拟别名只在缓存大小大于16 KB时才会出现。**

硬件开启方式

P6600核心的**Config7IAR**位总是被设置，以指示存在指令缓存虚拟别名硬件。核心允许一个物理地址通过不同的虚拟地址访问时位于多个索引中。当由于CACHE或SYNCI指令发出无效请求时，核心会逐个检查给定物理地址可能的所有别名位置。

硬件可以通过Config7IVAD位进行启用和禁用。清除此位时，用于消除指令缓存虚拟别名的硬件将被启用，这样虚拟别名就会在硬件中被管理，无需软件干预。

预编码和奇偶校验

P6600核心的L1指令缓存有一些额外的预编码（precode）位，这些位让issue能快速检测到分支和跳转指令，这些预编码位标明在一个64位的issue大小内转移指令的类型和位置。

L1指令缓存还包括16个奇偶校验位，用于128位数据的每个字节，以及标签数组对每个标签有5个奇偶校验位，用于预编码字段和物理标签、锁定和有效位。LRU数组没有任何奇偶校验。

替换策略

P6600核心的L1指令缓存使用LRU策略来管理替换，但会排除任何已锁定的路。

在出现miss时，选定行的标签和路径选择条目的锁定和LRU位可用于确定将被选择的路径。LRU字段在路径选择数组中会按照以下规则更新：

- 缓存命中：与之关联的路径更新为最近使用的，其他路径的相对顺序不变。
- 缓存refill：被填充的路径更新为最近使用的。
- CACHE指令：LRU位的更新取决于要执行的操作类型，包括各种情况如索引无效、索引加载标签等。

如果所有路径都是有效的，那么锁定的路径都将从LRU替换的中排除。对于未锁定的路径，使用LRU位来识别最近最少使用的路径，并选择该路径进行替换。

Cache Line Locking

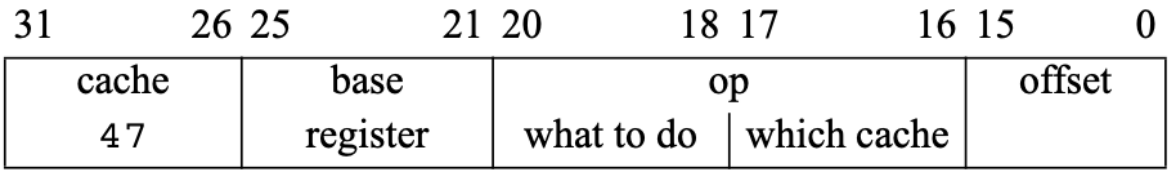
...

Software Cache Management

在P6600架构中，L1指令缓存并不完全是Coherent，因此有时需要操作系统进行干预。CACHE指令是这种干预的基础构建块，用于**正确处理DMA数据和缓存初始化**。CACHE指令在写入指令时也有作用。

在P6600架构中，CACHE指令以 cache op, addr 的形式编写，其中 addr 是一个地址格式，与加载/存储指令相同。缓存操作是特权指令，只能在内核模式下运行。但是，SYNCI指令可以在用户模式下工作。可以简化某些缓存管理任务。CACHE和SYNCI指令为系统提供了高度的控制能力。

CACHE指令的17:16位选择要处理的缓存：00：L1 I-cache，01：L1 D-cache，10：reserved， 11：L2 cache



CACHE指令有三种变体，它们在如何选择要操作的缓存行方面有所不同：

- 命中型缓存操作（Hit-type）：提供一个地址（就像加载/存储一样），该地址在缓存中查找。如果该位置在缓存中（即“命中”），则在该行上执行缓存操作。如果该位置不在缓存中，什么也不会发生。
- 地址型缓存操作（Address-type）：提供某个内存数据的地址，该地址就像一个缓存访问一样被处理 - 如果缓存之前是无效的，数据将从内存中获取。
- 索引型缓存操作（Index-type）：使用地址的尽可能多的低位来选择缓存行中的字节，然后选择缓存行内的一路。需要知道Config1寄存器中缓存的大小，以准确了解字段边界的位置。

MIPS64规范允许CPU设计者选择是从虚拟地址还是物理地址获取索引。对于索引型操作，MIPS推荐使用kseg0地址，这样虚拟地址和物理地址是相同的，也避免了缓存别名（aliasing）的可能。

CP0 Register Interface

不同的CP0寄存器用于指令缓存操作。具体地，这些CP0寄存器包括Config1、CacheErr、ITagLo、ITagHi、IDataLo和IDataHi。

例如在Config1寄存器中：

- IS字段（位24:22）指示指令缓存每路有多少个集合。P6600的L1指令缓存支持每路256个集合，用于配置32 KB的缓存，或每路512个集合，用于配置64 KB的缓存。
- IL字段（位21:19）指示指令缓存的行大小。P6600的L1指令缓存支持固定的32字节行大小，该字段的默认值为4。
- IA字段（位18:16）指示指令缓存的集合关联度。P6600的L1指令缓存固定为4路集合关联，该字段的默认值为3。

其他的就不一一介绍，等到需要使用的时候再查

Cache初始化

关于初始化部分的代码进行了简单的理解，下面的部分代码的解释：

在使用之前，缓存必须被初始化为一个已知的状态，即所有的缓存项都必须被置为无效。这个代码示例初始化缓存，找到缓存集的总数，然后使用缓存指令遍历缓存Set，使每个缓存Set无效。

```
LEAF (init_icache)
// For this Core there is always an L1 instuction cache // The IS field determines how many sets there are
// IS = 2 there are 256 sets
// IS = 3 there are 512 sets
// $11 set to line size, will be used to increment through the cache tags
li $11, 32 # Line size is always 32 bytes.
```

该指令缓存的行大小总是32字节，有4个way，并且大小可以是32 KB或64 KB。通过Config1寄存器的IS字段（每路集数）来确定缓存的大小，该字段可以有两个值：0x2表示32 KB缓存，0x3表示64 KB缓存。

```
mfc0  $10, $16, 1
ext   $12, $10, 22, 3
li    $14, 2
# Read C0_Config1
# Extract IS
# Used to test against
```

首先进行一系列检查和设置，以确定缓存的大小和迭代值，如果检查结果为真，代码将使用分支延迟槽（总是会被执行）来为32 KB缓存设置集合迭代值为256，然后跳转到 Isets_done 。如果检查结果为假，代码假定缓存大小为64 KB。此时，代码在分支延迟槽中仍然将迭代值设置为256，但随后会跳过并再次将其设置为512，以适应64 KB的缓存。

```
beq $14, $12, Isets_done # if IS = 2
li $12, 256 # sets = 256
li $12, 512 # else sets = 512 Skipped if branch taken
```

然后，通过虚拟地址设置一个索引进入缓存，该地址随后被转换为物理地址。之后，代码使用Move to Coprocessor Zero（MTC0）指令清除标签寄存器，从而使集合自由并可根据需要进行填充。然后，代码使用缓存指令进行索引存储标签操作，然后通过增加虚拟地址来初始化缓存的下一个集合。最后，代码进行循环维护，逐一减少循环计数器，并检查是否已经到达零。如果没有，它将回到标签一并继续执行。

```
// clear the lock bit, valid bit, and the LRF bit
mtc0  $0, $28     # Clear C0_ITagLo to invalidate entry

next_icache_tag:
cache 0x8, 0($14) # Index Store tag Cache op

# Loop maintenance
add   $12, -1             # Decrement set counter
bne   $12, $0, next_icache_tag # Done yet?
add $14, $11 # Increment line address by line size

done_icache:
# Modify return address to kseg0 which is cacheable
# (for code linked in kseg1.)
# Debugging consideration; leave commented during debugging
# ins r31_return_addr, $0, 29, 1
jr  r31_return_addr
nop
```