4. Cache Organization and Coherency

The processor implements a two-level cache structure consisting of separate primary instruction and data caches and a joint secondary cache.

Each cache is two-way set associative and uses a **write back protocol**; that is, two cache blocks are assigned to each set (as shown in Figure 4-1), and a cache store writes data into the cache instead of writing it directly to memory. Some time later this data is independently written to memory.

A write-invalidate cache coherency protocol (described later in this chapter) is supported through a set of cache states and external coherency requests.

4.1 Primary Instruction Cache

The processor has an on-chip 32-Kbyte primary instruction cache (also referred to simply as the *instruction cache*), which is a subset of the secondary cache. Organization of the instruction cache is shown in Figure 4-1.

The instruction cache has a fixed block size of 16 words and is two-way set associative with a **least-recently-used** (LRU) replacement algorithm.†

The instruction cache is indexed with a virtual address and tagged with a physical address.

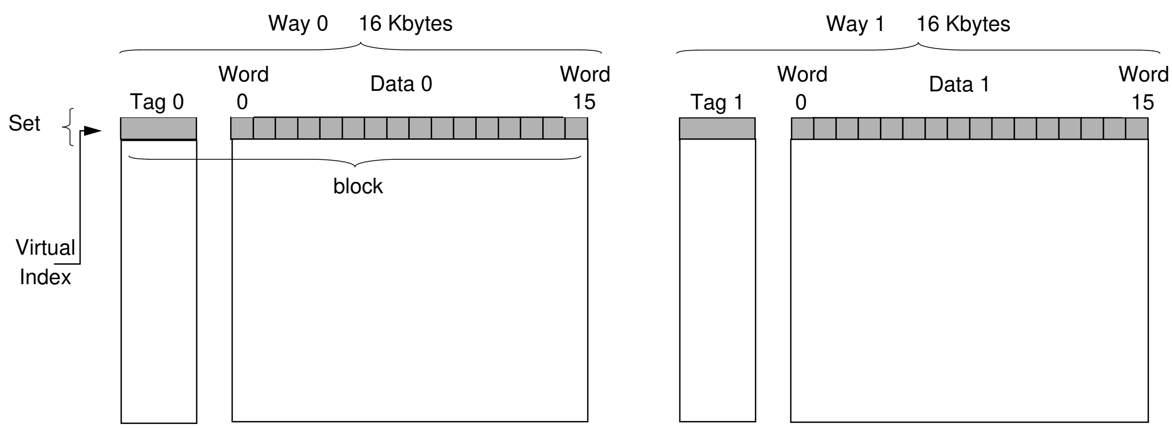

Figure 4-1 Organization of Primary Instruction Cache

Each instruction cache block is in one of the following two states:

- *Invalid*
- *Valid*

† The precise implementation of the LRU algorithm is affected by the speculative execution of instructions.

An instruction cache block can be changed from one state to the other as a result of any one of the following events:

- a primary instruction cache read miss
- subset property enforcement
- any of various CACHE instructions
- external intervention exclusive and invalidate requests

These events are illustrated in Figure 4-2, which shows the primary instruction cache state diagram.

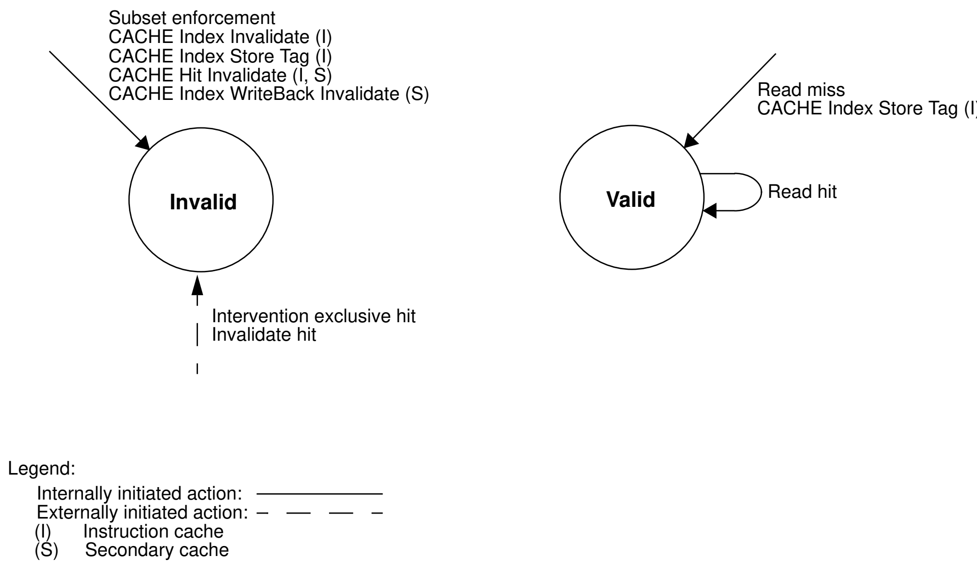

Figure 4-2 Primary Instruction Cache State Diagram

4.2 Primary Data Cache

The processor has an on-chip 32-Kbyte primary data cache (also referred to simply as the *data cache*), which is a subset of the secondary cache. The data cache uses a fixed block size of 8 words and is two-way set associative (that is, two cache blocks are assigned to each set, as shown in Figure 4-3) with an LRU replacement algorithm.†

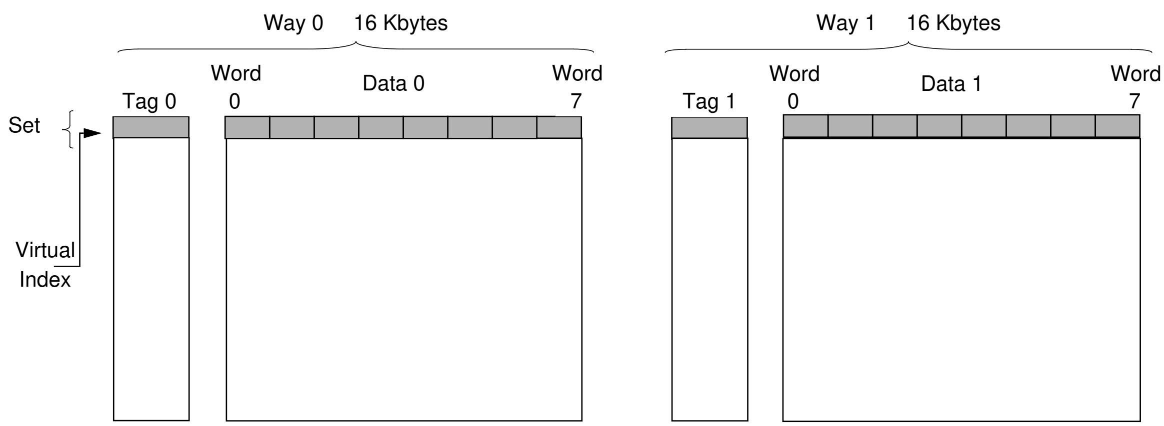

Figure 4-3 Organization of Primary Data Cache

The data cache uses a write back protocol, which means a cache store writes data into the cache instead of writing it directly to memory. Sometime later this data is independently written to memory, as shown in Figure 4-4.

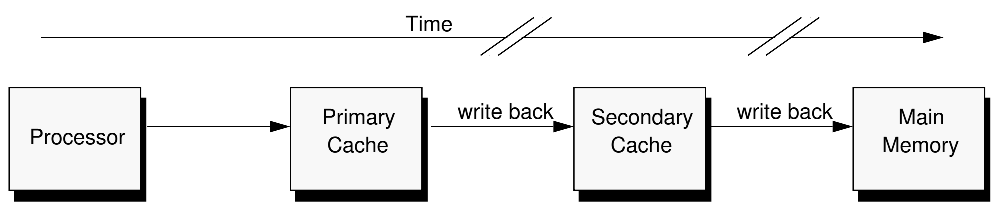

Figure 4-4 Write Back Protocol

Write back from the primary data cache goes to the secondary cache, and write back from the secondary cache goes to main memory, through the system interface. The primary data cache is written back to the secondary cache before the secondary cache is written back to the system interface.

† The precise implementation of the LRU algorithm is affected by the speculative execution of instructions.

The data cache is indexed with a virtual address and tagged with a physical address. Each primary cache block is in one of the following four states:

- *Invalid*
- *CleanExclusive*
- *DirtyExclusive*
- *Shared*

A primary data cache block is said to be *Inconsistent* when the data in the primary cache has been modified from the corresponding data in the secondary cache. The primary data cache is maintained as a subset of the secondary cache where the state of a block in the primary data cache always matches the state of the corresponding block in the secondary cache.

A data cache block can be changed from one state to another as a result of any one of the following events:

- primary data cache read/write miss
- primary data cache write hit
- subset enforcement
- a CACHE instruction
- external intervention shared request
- intervention exclusive request
- invalidate request

These events are illustrated in Figure 4-5, which shows the primary data cache state diagram.

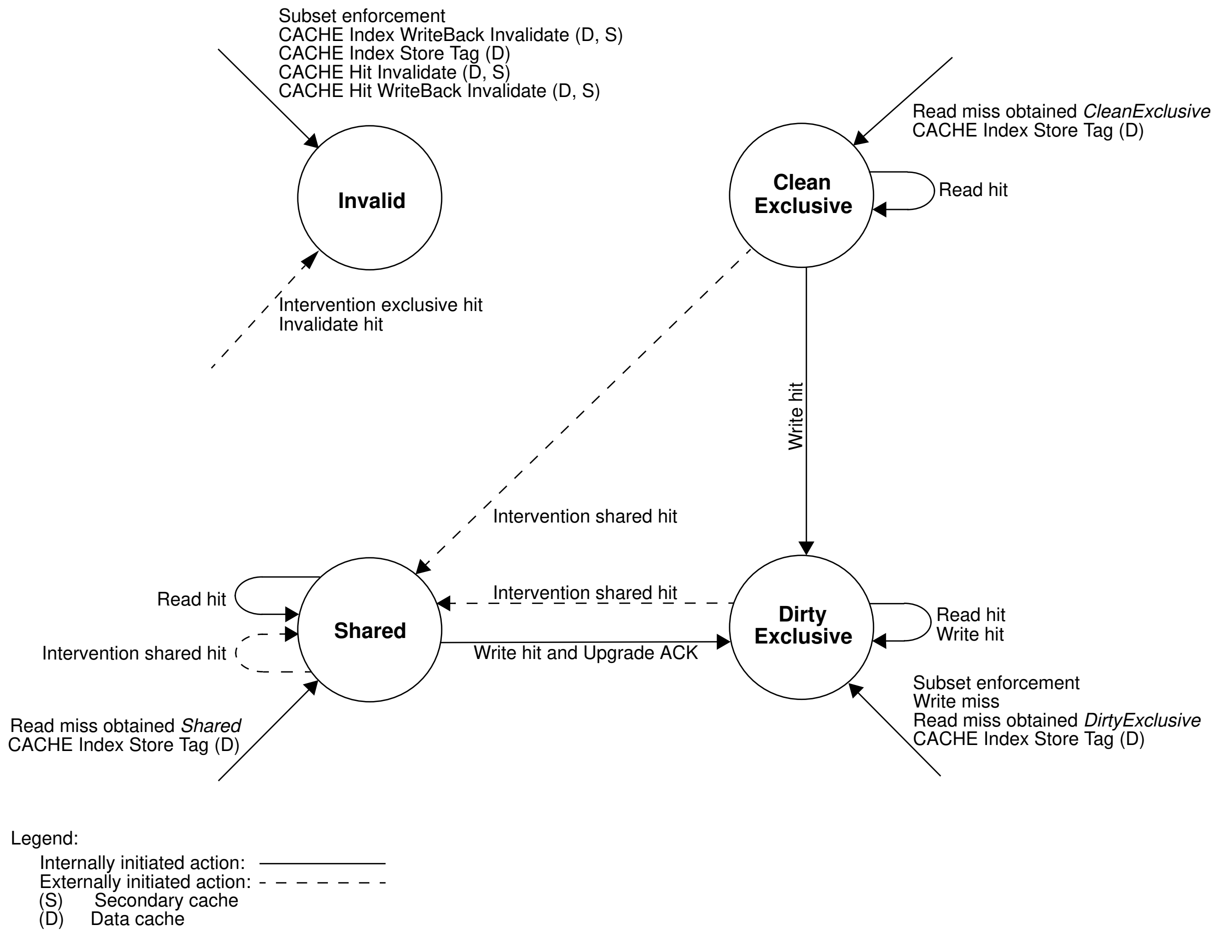

Figure 4-5 Primary Data Cache State Diagram

4.3 Secondary Cache

The R10000 processor must have an external secondary cache, ranging in size from 512 Kbytes to 16 Mbytes, in powers of 2, as set by the **SCSize** mode bit. The **SCBlkSize** mode bit selects a block size of either 16 or 32 words.

The secondary cache is two-way set associative (that is, two cache blocks are assigned to each set, as shown in Figure 4-6) with an LRU replacement algorithm.†

The secondary cache uses a write back protocol, which means a cache store writes data into the cache instead of writing it directly to memory. Some time later this data is independently written to memory.

The secondary cache is indexed with a physical address and tagged with a physical address.

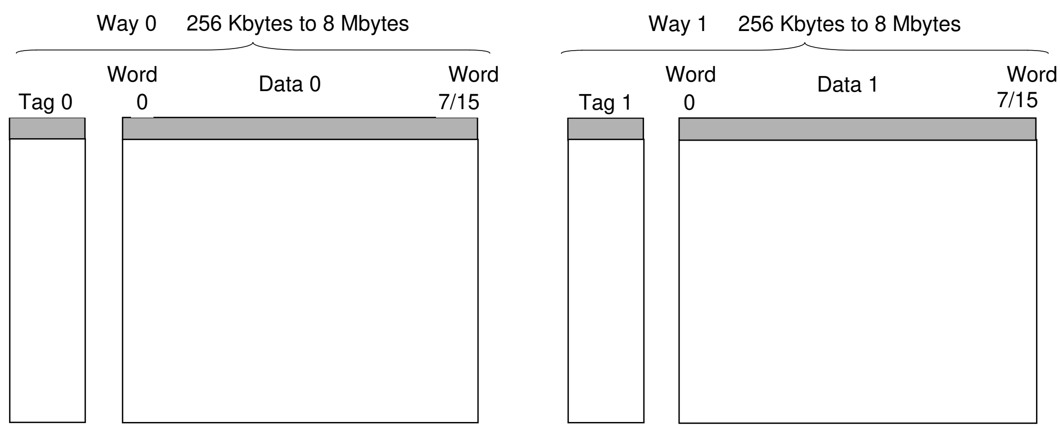

Figure 4-6 Organization of Secondary Cache

Each secondary cache block is in one of the following four states:

- *Invalid*
- *CleanExclusive*
- *DirtyExclusive*
- *Shared*

† The precise implementation of the LRU algorithm is affected by the speculative execution of instructions.

A secondary cache block can be changed from one state to another as a result of any of the following events:

- primary cache read/write miss
- primary cache write hit to a *Shared* or *CleanExclusive* block
- secondary cache read miss
- secondary cache write hit to a *Shared* or *CleanExclusive* block
- a CACHE instruction
- external intervention shared request
- intervention exclusive request
- invalidate request

These events are illustrated in Figure 4-7, which shows the secondary cache state diagram.

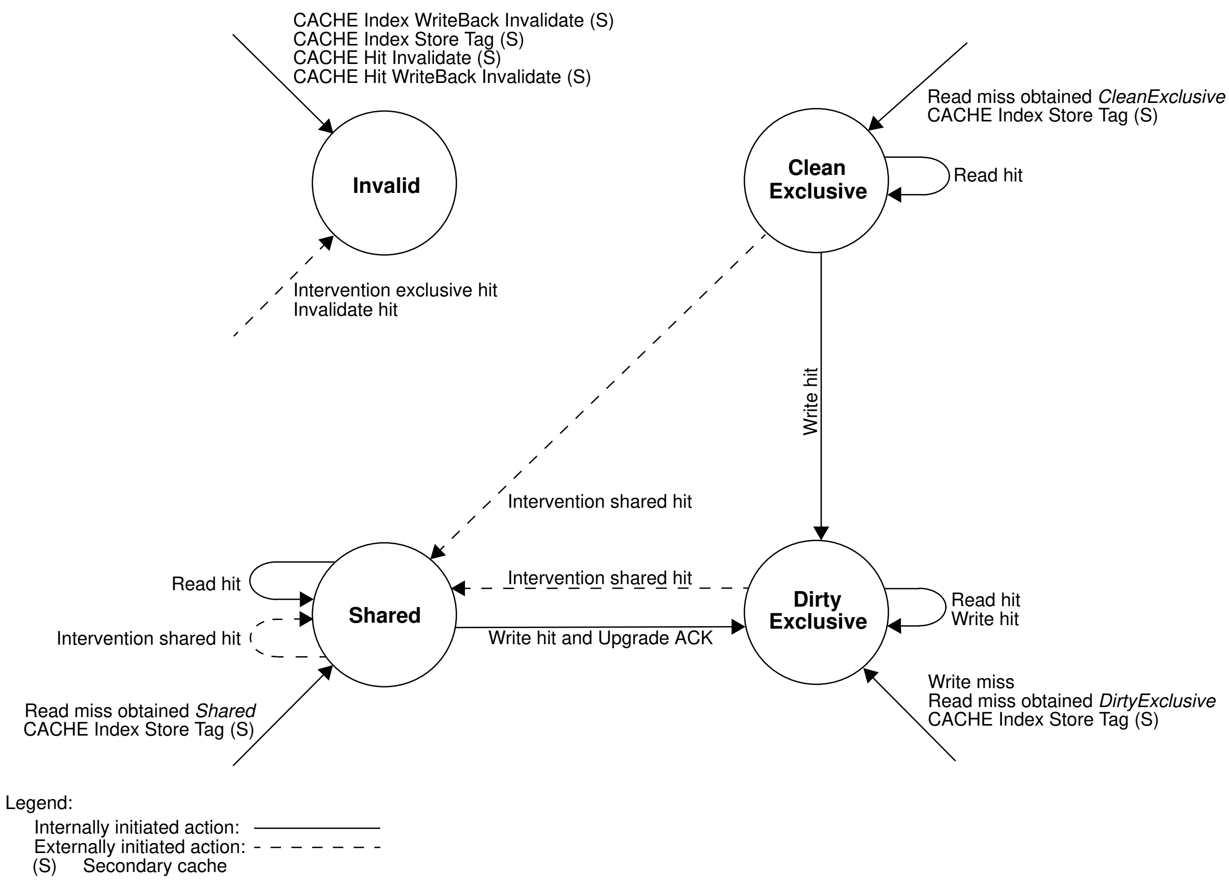

Figure 4-7 Secondary Cache State Diagram

4.4 Cache Algorithms

The behavior of the processor when executing load and store instructions is determined by the cache algorithm specified for the accessed address. The processor supports five different cache algorithms:

- uncached
- cacheable noncoherent
- cacheable coherent exclusive
- cacheable coherent exclusive on write
- uncached accelerated

Cache algorithms are specified in three separate places, depending upon the access:

- the cache algorithm for the mapped address space is specified on a per-page basis by the 3-bit cache algorithm field in the TLB
- the cache algorithm for the *kseg0* address space is specified by the 3-bit *K0* field of the CP0 *Config* register
- the cache algorithm for the *xkphys* address space is specified by **VA[61:59]**

Table 4-1 presents the encoding of the 3-bit cache algorithm field used in the TLB; *EntryLo0* and *EntryLo1* registers; CP0 *Config* register *K0* field for the *kseg0* address space; and **VA[61:59]** for the *xkphys* address space.

Table 4-1 Cache Algorithm Field Encodings

Value	Cache Algorithm
0	Reserved
1	Reserved
2	Uncached
3	Cacheable noncoherent
4	Cacheable coherent exclusive
5	Cacheable coherent exclusive on write
6	Reserved
7	Uncached accelerated

Descriptions of the Cache Algorithms

This section describes the cache algorithms listed in Table 4-1.

Uncached

Loads and stores under the *Uncached* cache algorithm bypass the primary and secondary caches. They are issued directly to the System interface using processor double/single/partial-word read or write requests.

Cacheable Noncoherent

Under the *Cacheable noncoherent* cache algorithm, load and store secondary cache misses result in processor noncoherent block read requests. External agents containing caches need not perform a coherency check for such processor requests.

Cacheable Coherent Exclusive

Under the *Cacheable coherent exclusive* cache algorithm, load and store secondary cache misses result in processor coherent block read exclusive requests. Such processor requests indicate to external agents containing caches that a coherency check must be performed and that the cache block must be returned in an *Exclusive* state.

Cacheable Coherent Exclusive on Write

The *Cacheable coherent exclusive on write* cache algorithm is similar to the *Cacheable coherent exclusive* cache algorithm except that load secondary cache misses result in processor coherent block read shared requests. Such processor requests indicate to external agents containing caches that a coherency check must be performed and that the cache block may be returned in either a *Shared* or *Exclusive* state.

Store hits to a *Shared* block result in a processor upgrade request. This indicates to external agents containing caches that the block must be invalidated.

Uncached Accelerated

The R10000 processor implements a new cache algorithm, *Uncached accelerated*. This allows the kernel to mark the TLB entries for certain regions of the physical address space, or certain blocks of data, as uncached while signalling to the hardware that data movement optimizations are permissible. This permits the hardware implementation to gather a number of uncached writes together, either a series of writes to the same address or sequential writes to all addresses in the block, into an uncached accelerated buffer and then issue them to the system interface as processor block write requests. The *uncached accelerated* algorithm differs from the *uncached* algorithm in that block write gathering is not performed.

There is no difference between an uncached accelerated load and an uncached load. Only word or doubleword stores can take advantage of this mode.

Stores under the *Uncached accelerated* cache algorithm bypass the primary and secondary caches. Stores to identical or sequential addresses are gathered in the uncached buffer, described in Chapter 6, the section titled “Uncached Buffer.”

Completely gathered uncached accelerated blocks are issued to the System interface as processor block write requests. Incompletely gathered uncached accelerated blocks are issued to the System interface using processor double/single-word write requests; this is also described in Chapter 6, the section titled “Uncached Buffer.”

4.5 Relationship Between Cached and Uncached Operations

Uncached and uncached accelerated load and store instructions are executed in order, and non-speculatively. Such accesses are buffered in the uncached buffer by the processor until they can be issued to the System interface.

All uncached and uncached accelerated accesses retain program order within the uncached buffer. The processor continues issuing cached accesses while uncached accesses are queued in the uncached buffer.

NOTE: Cached accesses do not probe the uncached buffer for conflicts.

Buffered uncached stores prevent a SYNC instruction from graduating. However buffered uncached accelerated stores do not prevent a SYNC instruction from graduating. The processor continues issuing cached accesses speculatively and out of order beyond a SYNC instruction that is waiting to graduate.

An uncached load may be used to guarantee that the uncached buffer is flushed of all uncached and uncached accelerated accesses.

A SYNC instruction and the **SysGblPerf*** signal may be used to guarantee that all cache accesses and uncached stores have been globally performed as described in Chapter 6, the section titled “SysGblPerf* Signal.”

An uncached load followed by a SYNC instruction may be used to guarantee that all cache accesses, uncached accesses, and uncached accelerated accesses have been globally performed.

4.6 Cache Algorithms and Processor Requests

The cache algorithm determines the type of processor request generated for secondary cache load misses, secondary cache store misses, and store hits. Table 4-2 presents the relationship between the cache algorithm and processor requests.

Table 4-2 Cache Algorithms and Processor Requests

Cache Algorithm	Load Miss	Store Miss	Store Hit
Uncached	Double/single/partial-word read	Double/single/partial-word write	NA
Cacheable noncoherent	Noncoherent block read	Noncoherent block read	Upgrade if *Shared**
Cacheable coherent exclusive	Coherent block read exclusive	Coherent block read exclusive	Upgrade if *Shared**
Cacheable coherent exclusive on write	Coherent block read shared	Coherent block read exclusive	Upgrade if *Shared*
Uncached accelerated	Double/single/partial-word read	Gather identical or sequential double/single-word stores in the uncached buffer. Block write for completely gathered blocks. Double/single-word write for incompletely gathered blocks. Partial-word write for partial-word stores.	NA

* Should not occur under normal circumstances. Most systems return the *Exclusive* state for a cacheable noncoherent line; therefore, the *Shared* state is not normal.

4.7 Cache Block Ownership

The processor requires cache blocks to have a single owner at all times. The owner is responsible for providing the current contents of the cache block to any requestor.

The processor uses the following ownership rules:

- The processor assumes ownership of a cache block if the state of the cache block becomes *DirtyExclusive*. For a processor block read request, the processor assumes ownership of the block after receiving the last doubleword of a *DirtyExclusive* external block data response and an external ACK completion response. For a processor upgrade request, the processor assumes ownership of the block after receiving an external ACK completion response.
- The processor gives up ownership of a cache block if the state of the cache block changes to *Invalid*, *CleanExclusive*, or *Shared*.
- *CleanExclusive* and *Shared* cache blocks are always considered to be owned by memory.