System Interface

12

The System interface allows the processor to access external resources needed to satisfy cache misses and uncached operations, while permitting an external agent access to some of the processor internal resources.

In the R4000MC configuration, the System interface also provides the processor with mechanisms to maintain the cache coherency of shared data, while providing an external agent the mechanisms to maintain system-wide multiprocessor cache coherency.

This chapter describes the System interface from the point of view of both the processor and the external agent.

12.1 Terminology

The following terms are used in this chapter:

- An *external agent* is any logic device connected to the processor, over the System interface, that allows the processor to issue requests.
- A *system event* is an event that occurs within the processor and requires access to external system resources.
- *Sequence* refers to the precise series of requests that a processor generates to service a system event.
- *Protocol* refers to the cycle-by-cycle signal transitions that occur on the System interface pins to assert a processor or external request.
- *Syntax* refers to the precise definition of bit patterns on encoded buses, such as the command bus.

12.2 System Interface Description

The R4000 processor supports a 64-bit address/data interface that can construct systems ranging from a simple uniprocessor with main memory to a multiprocessor system with caches and complete cache coherency. The System interface consists of:

- 64-bit address and data bus, **SysAD**
- 8-bit SysAD check bus, **SysADC**
- 9-bit command bus, **SysCmd**
- eight handshake signals:
  - **RdRdy***, **WrRdy***
  - **ExtRqst***, **Release***
  - **ValidIn***, **ValidOut***
  - **IvdAck***, **IvdErr***

The processor uses the System interface to access external resources such as cache misses and uncached operations. In the case of R4000MC, the System interface also supports multiprocessor cache coherency.

Interface Buses

Figure 12-1 shows the primary communication paths for the System interface: a 64-bit address and data bus, **SysAD(63:0)**, and a 9-bit command bus, **SysCmd(8:0)**. These **SysAD** and the **SysCmd** buses are bidirectional; that is, they are driven by the processor to issue a processor request, and by the external agent to issue an external request (see Processor and External Requests, in this chapter, for more information).

A request through the System interface consists of:

- an address
- a System interface command that specifies the precise nature of the request
- a series of data elements if the request is for a write, read response, or update.

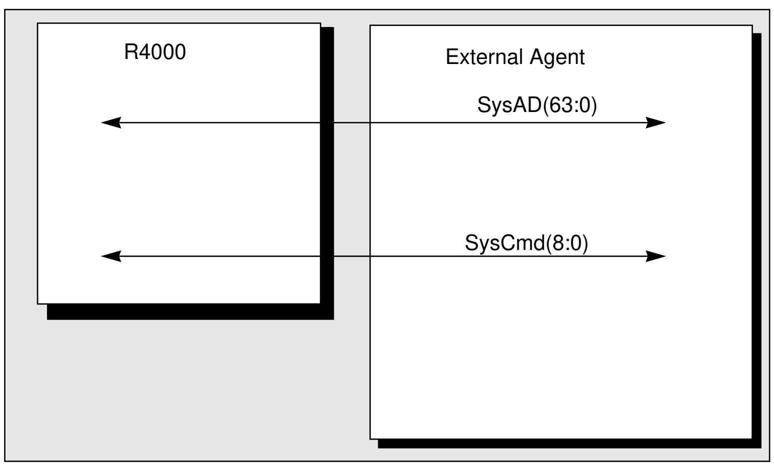

Figure 12-1 System Interface Buses

Address and Data Cycles

Cycles in which the **SysAD** bus contains a valid address are called *address cycles*. Cycles in which the **SysAD** bus contains valid data are called *data cycles*. Validity is determined by the state of the **ValidIn*** and **ValidOut*** signals (described in Interface Buses, in this chapter).

The **SysCmd** bus identifies the contents of the **SysAD** bus during any cycle in which it is valid. The most significant bit of the **SysCmd** bus is always used to indicate whether the current cycle is an address cycle or a data cycle.

- During address cycles [**SysCmd(8)** = 0], the remainder of the **SysCmd** bus, **SysCmd(7:0)**, contains a *System interface command* (the encoding of System interface commands is detailed in System Interface Commands and Data Identifiers, in this chapter).
- During data cycles [**SysCmd(8)** = 1], the remainder of the **SysCmd** bus, **SysCmd(7:0)**, contains a *data identifier* (the encoding of data identifiers is detailed in System Interface Commands and Data Identifiers, in this chapter).

Issue Cycles

There are two types of processor issue cycles:

- processor read, invalidate, and update request issue cycles
- processor write request issue cycles.

The processor samples the signal **RdRdy*** to determine the issue cycle for a processor read, invalidate, or update request; the processor samples the signal **WrRdy*** to determine the issue cycle of a processor write request.

As shown in Figure 12-2, **RdRdy*** must be asserted two cycles prior to the address cycle of the processor read/invalidate/update request to define the address cycle as the issue cycle.

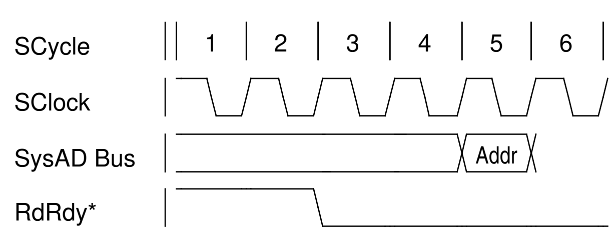

Figure 12-2 State of RdRdy Signal for Read, Invalidate, or Update Requests*

As shown in Figure 12-3, **WrRdy*** must be asserted two cycles prior to the first address cycle of the processor write request to define the address cycle as the issue cycle.

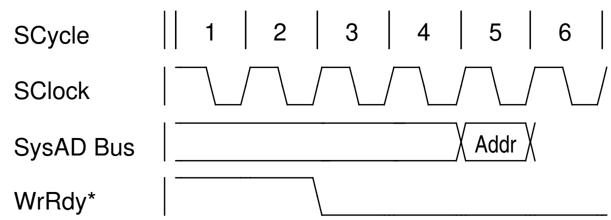

Figure 12-3 State of WrRdy Signal for Write Requests*

The processor repeats the address cycle for the request until the conditions for a valid issue cycle are met. After the issue cycle, if the processor request requires data to be sent, the data transmission begins. There is only one issue cycle for any processor request.

The processor accepts external requests, even while attempting to issue a processor request, by releasing the System interface to slave state in response to an assertion of **ExtRqst*** by the external agent.

Note that the rules governing the issue cycle of a processor request are strictly applied to determine the action the processor takes. The processor either:

- completes the issuance of the processor request in its entirety before the external request is accepted, or
- releases the System interface to slave state without completing the issuance of the processor request.

In the latter case, the processor issues the processor request (provided the processor request is still necessary) after the external request is complete. The rules governing an issue cycle again apply to the processor request.

Handshake Signals

The processor manages the flow of requests through the following eight control signals:

- **RdRdy***, **WrRdy*** are used by the external agent to indicate when it can accept a new read (**RdRdy***) or write (**WrRdy***) transaction.
- **ExtRqst***, **Release*** are used to transfer control of the **SysAD** and **SysCmd** buses. **ExtRqst*** is used by an external agent to indicate a need to control the interface. **Release*** is asserted by the processor when it transfers the mastership of the System interface to the external agent.
- The R4000 processor uses **ValidOut*** and the external agent uses **ValidIn*** to indicate valid command/data on the **SysCmd**/**SysAD** buses.
- **IvdAck***, **IvdErr*** are used in multiprocessor systems; they are asserted by the external agent to indicate the successful completion (**IvdAck***) or the unsuccessful completion **(IvdErr*)** of a pending processor invalidate or update request.†

† When using the R4000SC processor, **IvdAck*** and **IvdErr*** must be connected to Vcc.

12.3 System Interface Protocols

Figure 12-4 shows the System interface operates from register to register. That is, processor outputs come directly from output registers and begin to change with the rising edge of **SClock.**†

Processor inputs are fed directly to input registers that latch these input signals with the rising edge of **SClock**. This allows the System interface to run at the highest possible clock frequency.

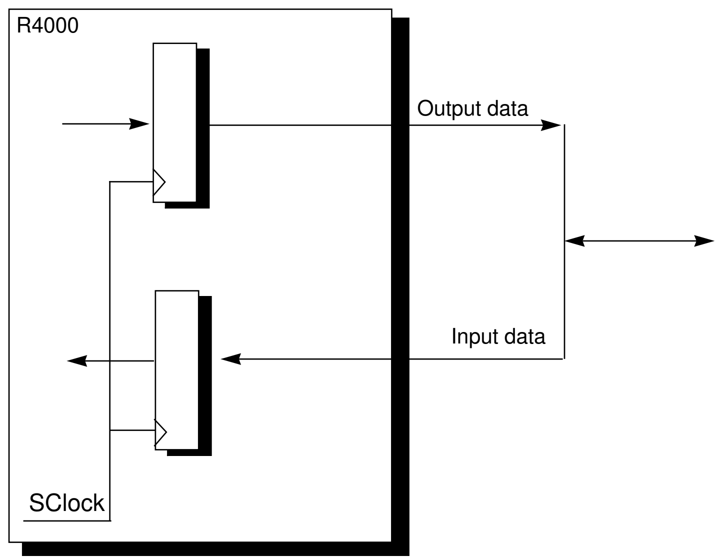

Figure 12-4 System Interface Register-to-Register Operation

Master and Slave States

When the R4000 processor is driving the **SysAD** and **SysCmd** buses, the System interface is in *master state*. When the external agent is driving the **SysAD** and **SysCmd** buses, the System interface is in *slave state*.

In master state, the processor asserts the signal **ValidOut*** whenever the **SysAD** and **SysCmd** buses are valid.

In slave state, the external agent asserts the signal **ValidIn*** whenever the **SysAD** and **SysCmd** buses are valid.

† **SClock** is an internal clock used by the processor to sample data at the System interface and to clock data into the processor System interface output registers; see Chapter 10 for more details.

Moving from Master to Slave State

The System interface remains in master state unless one of the following occurs:

- The external agent requests and is granted the System interface (external arbitration).
- The processor issues a read request or completes the issue of a cluster (uncompelled change to slave state).

External Arbitration

The System interface must be in slave state for the external agent to issue an external request through the System interface. The transition from master state to slave state is arbitrated by the processor using the System interface handshake signals **ExtRqst*** and **Release***. This transition is described by the following procedure:

1. An external agent signals that it wishes to issue an external request by asserting **ExtRqst***.
2. When the processor is ready to accept an external request, it releases the System interface from master to slave state by asserting **Release*** for one cycle.
3. The System interface returns to master state as soon as the issue of the external request is complete.

This process is described in External Arbitration Protocol, later in this chapter.

Uncompelled Change to Slave State

An *uncompelled* change to slave state is the transition of the System interface from master state to slave state, initiated by the processor when a processor read request is pending. **Release*** is asserted automatically after a read request or cluster (see Clusters, later in this chapter, for a definition of a cluster). An uncompelled change to slave state occurs either during or some number of cycles after the issue cycle of a read request, or either during or some number of cycles after the last cycle of the last request in a cluster.

The uncompelled release latency depends on the state of the cache, the presence or absence of a secondary cache, and the secondary cache parameters (see Release Latency, in this chapter). After an uncompelled change to slave state, the processor returns to master state at the end of the next external request. This can be a read response, or some other type of external request.

An external agent must note that the processor has performed an uncompelled change to slave state and begin driving the **SysAD** bus along with the **SysCmd** bus. As long as the System interface is in slave state, the external agent can begin an external request without arbitrating for the System interface; that is, without asserting **ExtRqst***.

After the external request, the System interface returns to master state.

Whenever a processor read request is pending, after the issue of a read request or after the issue of all of the requests in a cluster, the processor automatically switches the System interface to slave state, even though the external agent is not arbitrating to issue an external request. This transition to slave state allows the external agent to return read response data.

12.4 Processor and External Requests

There are two broad categories of requests: *processor requests* and *external requests*. These two categories are described in this section.

When a system event occurs, the processor issues either a single request or a series of requests—called *processor requests*—through the System interface, to access an external resource and service the event. For this to work, the processor System interface must be connected to an external agent that is compatible with the System interface protocol, and can coordinate access to system resources.

An external agent requesting access to processor caches or to a processor status register generates an *external request*. This access request passes through the System interface. System events and request cycles are shown in Figure 12-5.

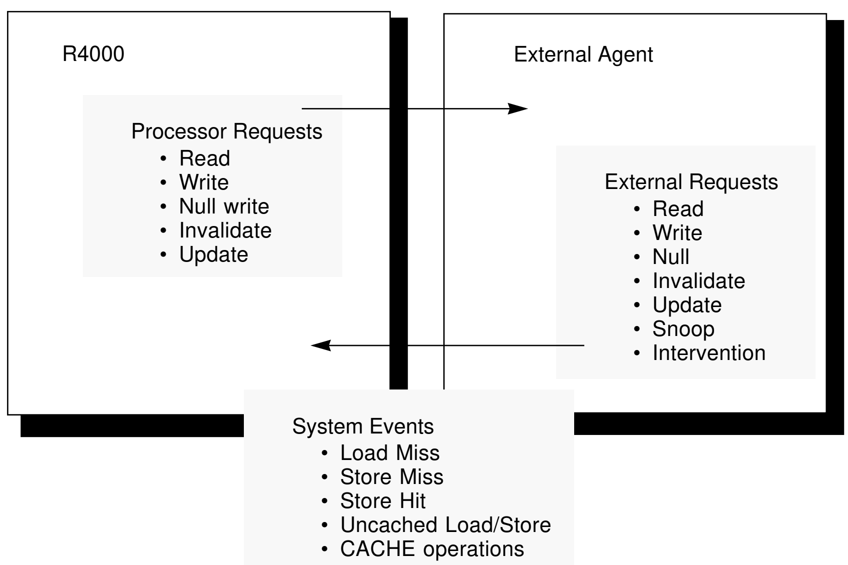

Figure 12-5 Requests and System Events

Rules for Processor Requests

The following rules apply to processor requests.

- After issuing a processor read request, either individually or as part of a cluster, the processor cannot issue a subsequent read request until it has received a read response.
- After issuing a processor update request, or after a potential update request becomes compulsory, the processor cannot issue a subsequent request until it has received an acknowledge for the update request.
- After the processor has issued a write request, the processor cannot issue a subsequent request until at least four cycles after the issue cycle of the write request. This means back-to-back write requests with a single data cycle are separated by two unused system cycles, as shown in Figure 12-6.

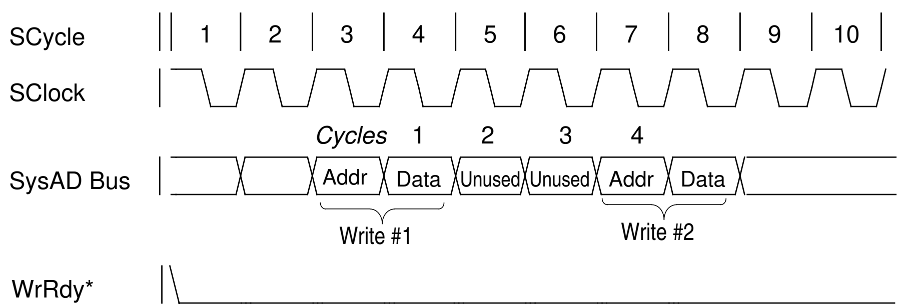

Figure 12-6 Back-to-Back Write Cycle Timing

Processor Requests

A processor request is a request or a series of requests, through the System interface, to access some external resource. As shown in Figure 12-7, processor requests include read, write, null write, invalidate, and update. This section also describes clusters.

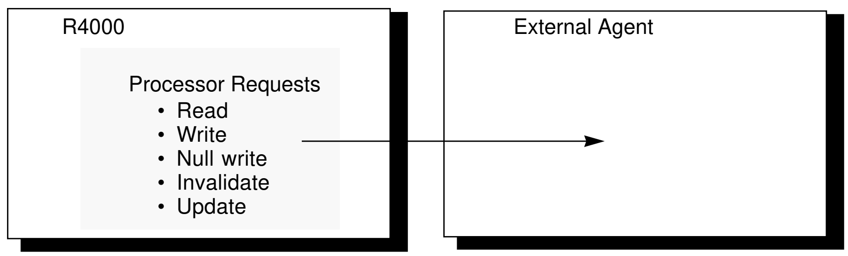

Figure 12-7 Processor Requests

Read request asks for a block, doubleword, partial doubleword, word, or partial word of data either from main memory or from another system resource.

Write request provides a block, doubleword, partial doubleword, word, or partial word of data to be written either to main memory or to another system resource.

Null write request indicates that an expected write has been cancelled as a result of an external request.

Invalidate request specifies a line in every other cache in the system that must be marked invalid.

Update request provides a block, doubleword, partial doubleword, word, or partial word of data that must be transferred to every other cache in the system.

Table 12-1 lists the processor requests that each type of R4000 can issue.

Table 12-1 Supported Processor Requests

Request	R4000PC	R4000SC	R4000MC
Processor Read	X	X	X
Processor Write	X	X	X
Processor Null Write		X	X
Processor Invalidate			X
Processor Update			X

Processor requests are managed by the processor in two distinct modes: *secondary-cache mode* and *no-secondary-cache mode* (see Chapter 11 for a description of these two modes), which are programmable through the boot-time mode control interface described in Chapter 9.

The permissible modes of operation are dependent on the following processor package configurations; if not programmed correctly, the behavior of the processor is undefined.

- An R4000PC must be programmed to run in no-secondary-cache mode.
- An R4000SC or R4000MC can be programmed to run in either secondary-cache or no-secondary-cache mode.

In no-secondary-cache mode, the processor issues requests in a strict sequential fashion; that is, the processor is only allowed to have one request pending at any time. For example, the processor issues a read request and waits for a read response before issuing any subsequent requests. The processor submits a write request only if there are no read requests pending.

The processor has the input signals **RdRdy*** and **WrRdy*** to allow an external agent to manage the flow of processor requests. **RdRdy*** controls the flow of processor read, invalidate, and update requests, while **WrRdy*** controls the flow of processor write requests. Processor null write requests must always be accepted and cannot be delayed by either **RdRdy*** or **WrRdy***. The processor request cycle sequence is shown in Figure 12-8.

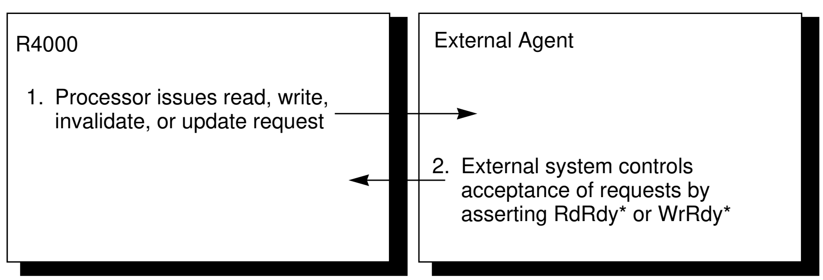

Figure 12-8 Processor Request

Processor Read Request

When a processor issues a read request, the external agent must access the specified resource and return the requested data. (Processor read requests are described in this section; external read requests are described in External Requests, later on in this chapter.)

A processor read request can be split from the external agent's return of the requested data; in other words, the external agent can initiate an unrelated external request before it returns the response data for a processor read. A processor read request is completed after the last word of response data has been received from the external agent.

Note that the data identifier (see System Interface Commands and Data Identifiers, in this chapter) associated with the response data can signal that the returned data is erroneous, causing the processor to take a bus error.

Processor read requests that have been issued, but for which data has not yet been returned, are said to be *pending*. A read remains pending until the requested read data is returned.

In secondary-cache mode, the external agent must be capable of accepting a processor read request followed by a potential update request any time all three of the following conditions are met:

- There is no processor read request pending.
- There is no unacknowledged processor update request.
- The signal **RdRdy*** has been asserted for two or more cycles.

In no-secondary-cache mode, the external agent must be capable of accepting a processor read request any time the following two conditions are met:

- There is no processor read request pending.
- The signal **RdRdy*** has been asserted for two or more cycles.

Processor Write Request

When a processor issues a write request, the specified resource is accessed and the data is written to it. (Processor write requests are described in this section; external write requests are described in External Requests, later on in this chapter.)

A processor write request is complete after the last word of data has been transmitted to the external agent.

In secondary-cache mode, the external agent must be capable of accepting a processor write request any time all three of the following conditions are met:

- There is no processor read request pending.
- There is no unacknowledged processor update request that is compulsory.
- The signal **WrRdy*** has been asserted for two or more cycles.

In no-secondary-cache mode, the external agent must be capable of accepting a processor write request any time the following two conditions are met:

- No processor read request is pending.
- The signal **WrRdy*** has been asserted for two or more cycles.

Processor Invalidate Request

An invalidate request notifies all processors that the specified cache line must be marked invalid in all caches in the system. Invalidate requests can only be used in a multiprocessing system.

When a processor issues an invalidate request, the specified resource is accessed and the line is marked invalid. (Processor invalidate requests are described in this section; external invalidate requests are described in External Requests, later on in this chapter.)

A processor invalidate request requires a completion acknowledge by either the invalidate acknowledge signal **IvdAck*** or the invalidate error signal **IvdErr***, unless the invalidate is canceled by the external agent. A processor invalidate request that has been submitted, but for which the processor has not yet received an acknowledge or a cancellation, is said to be *unacknowledged*. When the processor invalidate request fails (**IvdErr*** is asserted), the issuing processor takes a bus error on the store instruction that generated the failed request. Figure 12-10 shows a sample processor invalidate/update request cycle.

Invalidate cancellation is signaled to the processor during external invalidate, update, snoop, and intervention requests; **IvdErr*** signals a processor invalidate request has failed.

A completion acknowledge for processor invalidate requests is signaled through the System interface on dedicated pins, and this acknowledgment can occur in parallel with processor and external requests.

State changes in the external system are not instantaneously reflected in the caches of every processor, which means an external agent can discover that a processor request for an invalidate cannot be completed. For example, a processor store can hit on a shared cache line and issue an invalidate to the external agent. However, before the external agent can transmit the invalidate to the rest of the system another invalidate for the same cache line can be received by the external agent. If this occurs, the processor cache does not reflect the current state of the system and the processor invalidate must not be transmitted to the system; instead, the external agent must cancel the processor unacknowledged invalidate. Figure 12-9 shows this cancellation cycle.

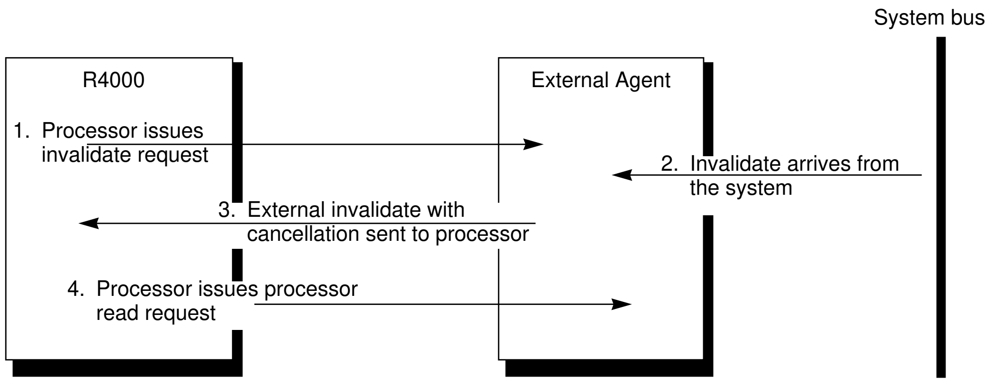

Figure 12-9 Cancelling an Invalidate Request

The steps shown in Figure 12-9 are described below:

1. The processor issues an invalidate on a store hit to a shared line in its cache.
2. An invalidate request, coming from the system bus, is received by the processor's external agent targeting the same cache line.
3. The external invalidate invalidates the cache line, and the processor invalidate request is cancelled.
4. The processor re-examines the state of the cache line and discovers the cache line which was target of the store is now invalid. The processor issues a processor read request to service the store miss.

Processor Update Request

An update request notifies all processors that a specified cache line in all caches throughout a multiprocessor system must be replaced with modified data. An update request can only be used in a multiprocessing system.

When a processor issues an update request, the specified resource is accessed and the line is updated. (Processor update requests are described in this section; external update requests are described in External Requests, later on in this chapter.)

A processor update request requires a completion acknowledge by either the invalidate acknowledge signal **IvdAck*** or the invalidate error signal **IvdErr*** (shown in Figure 12-10), unless the update is canceled by the external agent. A processor update request that has been submitted, but for which the processor has not yet received an acknowledge or a cancellation, is said to be *unacknowledged*. When the processor update request fails (**IvdErr*** is asserted), the issuing processor takes a bus error on the store instruction that generated the failed request. Figure 12-10 shows a sample processor invalidate/update request cycle.

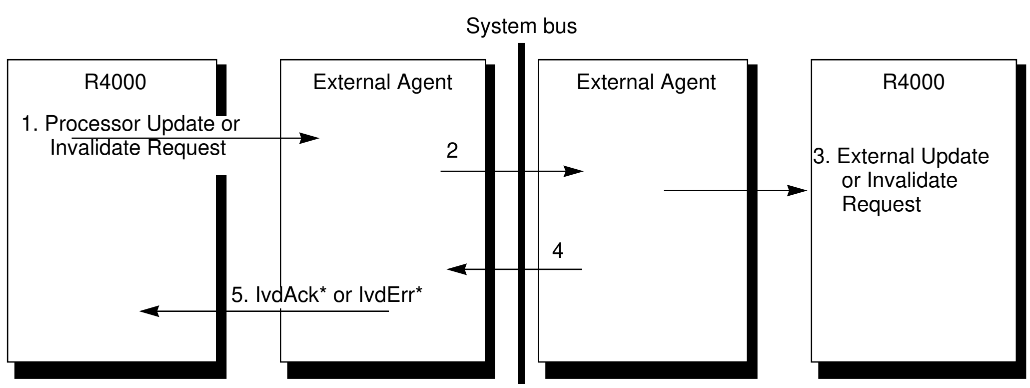

Figure 12-10 Processor Update/Invalidate Acknowledge Cycle

Update cancellation is signaled to the processor during external invalidate, update, snoop, and intervention requests; **IvdErr*** signals a processor update request has failed.

Since a completion acknowledge for processor update requests is signaled through the System interface on dedicated pins, this acknowledgment can occur in parallel with processor and external requests.

Clusters

A cluster consists of a single processor read request, followed by one or two additional processor requests that are issued while the initial read request is pending.

The processor supports three types of clusters:

- a processor read request, followed by a write request
- a processor read request, followed by potential update request
- a processor read request, followed by a potential update request, followed by a write request.

In secondary-cache mode, the processor issues individual requests (as in no-secondary-cache mode), or cluster requests. All requests in the cluster must be accepted before the response to the read request that began the cluster can be returned to the processor.

Potential update requests within a cluster can be disabled through the boot-time mode control interface.

Read With Write Forthcoming Request as Part of a Cluster

The processor signals that it is issuing a cluster containing a processor write request by issuing a *read-with-write-forthcoming* request, instead of starting the cluster with an ordinary read request. The read-with-write-forthcoming request is identified by a bit in the command for processor read requests.

The external agent must accept all requests that form the cluster before it can respond to the read request that began the cluster. The behavior of the processor is undefined if the external agent returns a response to a processor read request before accepting all of the requests of the cluster.

Potential Update as Part of a Cluster

Potential updates are identified by setting a bit in the processor update command. A processor potential update request is any update request that is issued while a processor read request is pending.

Once the processor issues a read request, a potential update request follows, regardless of the state of **RdRdy***. Potential update requests do not obey the **RdRdy*** flow control rules for issuance, but rather issue with a single address cycle regardless of the state of **RdRdy***.

A processor potential update request remains potential until the read response to the pending processor read request which began the cluster is received by the external agent.

- If the read response data is returned in one of the shared states—*shared* or *dirty shared*—the potential update becomes compulsory and is no longer potential. A compulsory update must receive an acknowledge either by the signal **IvdAck*** or **IvdErr***.
- If the read response data is returned in one of the exclusive states—*clean exclusive* or *dirty exclusive*—the potential update is nullified and the processor neither expects nor requires an acknowledge.

Write Request as Part of a Cluster

A write request that is part of a cluster obeys the **WrRdy*** timing rules for issuing, as shown earlier in Figure 12-3.

Null Write Request as Part of a Cluster

Since the processor accepts external requests between the issue of a read-with-write-forthcoming request that begins a cluster and the issue of the write request that completes a cluster, it is possible for an external request to eliminate the need for the write request in the cluster. For example, if the external agent issued an external invalidate request that targeted the cache line the processor was attempting to write back, the state of the cache line would be changed to invalid and the write back for the cache line would no longer be needed. In this event, the processor issues a processor null write request after completing the external request to complete the cluster.

Processor null write requests do not obey the **WrRdy*** flow control rules for issuance, rather they issue with a single address cycle regardless of the state of **WrRdy***. Any external request that changes the state of a cache line from dirty exclusive or dirty shared to clean exclusive, shared, or invalid obviates the need for a write back of that cache line.

External Requests

External requests include read, write, invalidate, update, snoop, intervention, and null requests, as shown in Figure 12-11. External invalidate, update, snoop and intervention requests, as a group, are referred to as *external coherence requests*. This section also includes a description of read response, a special case of an external request.

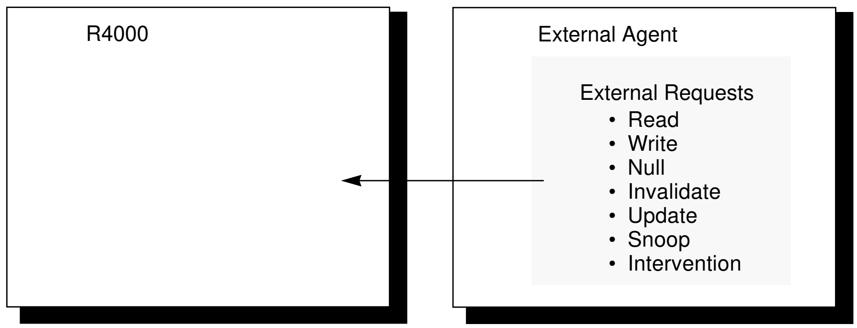

Figure 12-11 External Requests

Read request asks for a word of data from the processor's internal resource.

Write request provides a word of data to be written to the processor's internal resource.

Invalidate request specifies a cache line, in the primary and secondary caches of the processor, that must be marked invalid.

Update request provides a doubleword, partial doubleword, word, or partial word of data to be written to the processor's primary and secondary caches.

Snoop request checks the processor's secondary cache to see if a valid copy of a particular cache line exists. If a valid copy exists, the processor returns the state of the cache line at the specified physical address in the secondary cache, and can modify the state of the cache line.

Intervention request requires the processor to return the state of the secondary cache line at the specified physical address. Under certain conditions related to the state of the cache line and the nature of the intervention request, the contents of the primary and secondary cache line can be returned. The state of the line can also be modified by this request.

Null request requires no action by the processor; it provides a mechanism for the external agent to either return control of the secondary cache to the processor, or return the System interface to the master state without affecting the processor.

Table 12-2 lists the external requests that each type of R4000 can receive (an X indicates the request is supported on that model).

Table 12-2 Supported External Requests

Request Type	R4000PC	R4000SC	R4000MC
External Read	X	X	X
External Write	X	X	X
External Null (System interface)	X	X	X
External Null (Secondary Cache)		X	X
External Invalidate			X
External Update			X
External Snoop			X
External Intervention			X

The processor controls the flow of external requests through the arbitration signals **ExtRqst*** and **Release***, as shown in Figure 12-12. The external agent must acquire mastership of the System interface before it is allowed to issue an external request; the external agent arbitrates for mastership of the System interface by asserting **ExtRqst*** and then waiting for the processor to assert **Release*** for one cycle.

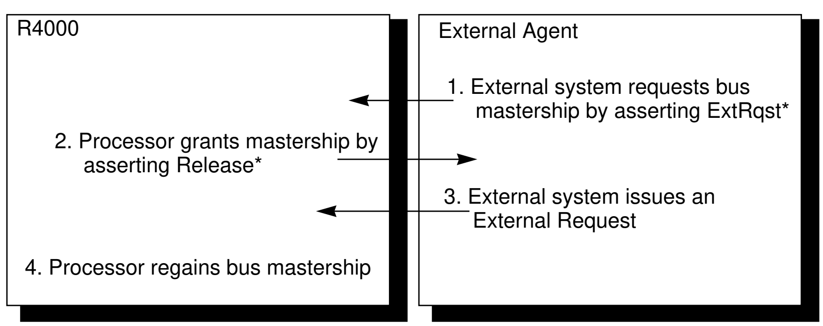

Figure 12-12 External Request

Mastership of the System interface always returns to the processor after an external request is issued. The processor does not accept a subsequent external request until it has completed the current request. The processor accepts external requests between the issue of a processor read request, or a processor read request followed by a potential update request and the issue of a processor write request within a cluster.

If there are no processor requests pending, the processor decides, based on its internal state, whether to accept the external request, or to issue a new processor request. The processor can issue a new processor request even if the external agent is requesting access to the System interface.

The external agent asserts **ExtRqst*** indicating that it wishes to begin an external request. The external agent then waits for the processor to signal that it is ready to accept this request by asserting **Release***. The processor signals that it is ready to accept an external request based on the criteria listed below.

- The processor completes any processor request or processor request cluster that is in progress.
- While waiting for the assertion of **RdRdy*** to issue a processor read request, the processor can accept an external request if the request is delivered to the processor one or more cycles before **RdRdy*** is asserted.
- While waiting for the assertion of **WrRdy*** to issue a processor write request, the processor can accept an external request provided the request is delivered to the processor one or more cycles before **WrRdy*** is asserted.
- If waiting for the response to a read request after the processor has made an uncompelled change to a slave state, the external agent can issue an external request before providing the read response data.

External Read Request

In contrast to a processor read request, data is returned directly in response to an external read request; no other requests can be issued until the processor returns the requested data. An external read request is complete after the processor returns the requested word of data.

The data identifier (see System Interface Commands and Data Identifiers in this chapter) associated with the response data can signal that the returned data is erroneous, causing the processor to take a bus error.

> **NOTE:** The processor does not contain any resources that are readable by an external read request; in response to an external read request the processor returns undefined data and a data identifier with its *Erroneous Data* bit, **SysCmd(5)**, set.

External Write Request

When an external agent issues a write request, the specified resource is accessed and the data is written to it. An external write request is complete after the word of data has been transmitted to the processor.

The only processor resource available to an external write request is the *Interrupt* register.

External Invalidate Request

When an external agent issues an invalidate request, the specified resource is accessed and the line is marked invalid. An external invalidate request is considered to be complete after the request has been transmitted.

External Update Request

When an external agent issues an update request, the specified resource is accessed and the line is replaced. An external update request is considered complete after the request has been transmitted.

External Snoop Request

An external snoop request makes the processor inspect the secondary cache to see if the cache contains a valid version of the specified cache line. If the valid cache line is present, the processor reports the cache line state and can modify this state.

An external snoop request is complete after the processor returns the state of the specified cache line.

External Intervention Request

When an external agent issues an intervention request, the specified secondary cache line is inspected. Upon inspection, the cache line state is reported and/or modified. Under certain circumstances the specified cache line may also be retrieved. The external intervention request is complete after one of the following occurs:

- the processor returns the state of the specified cache line
- the processor returns the last word of data for the specified cache line.

Read Response

A *read response* returns data in response to a processor read request, as shown in Figure 12-13. While a read response is technically an external request, it has one characteristic that differentiates it from all other external requests—it does not perform System interface arbitration. For this reason, read responses are handled separately from all other external requests, and are simply called read responses.

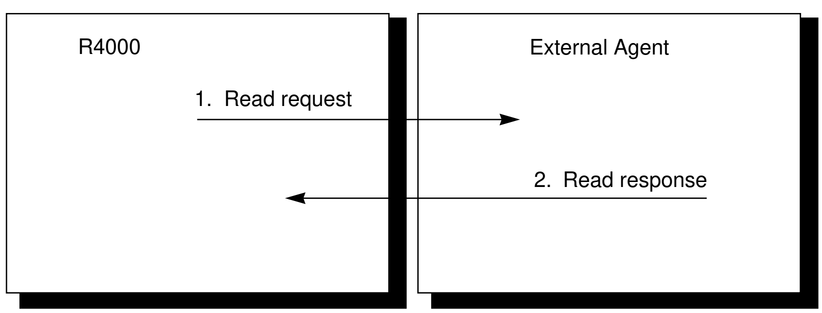

Figure 12-13 Read Response

12.5 Handling Requests

This section details the *sequence*, *protocol*, and *syntax* (See Terminology, in this chapter, for definitions of these terms) of both processor and external requests. The following system events are discussed:

- load miss in secondary-cache mode and no-secondary-cache mode
- store miss in secondary-cache mode and no-secondary-cache mode
- store hit
- uncached loads/stores
- CACHE operations
- load linked store conditional.

Load Miss

When a processor load misses in both the primary and secondary caches, before the processor can proceed it must obtain the cache line that contains the data element to be loaded from the external agent.

If the new cache line replaces a current dirty exclusive or dirty shared cache line, the current cache line must be written back before the new line can be loaded in the primary and secondary caches.

The processor examines the coherency attribute (cache coherency attributes are described in Chapter 11) in the TLB entry for the page that contains the requested cache line, and executes one of the following requests:

- If the coherency attribute is *exclusive*, the processor issues a coherent read request that also requests exclusivity.
- If the coherency attribute is *sharable* or *update*, the processor issues a coherent read request.
- If the coherency attribute is *noncoherent*, the processor issues a noncoherent read request.

Table 12-3 shows the actions taken on a load miss to primary and secondary caches.

Table 12-3 Load Miss to Primary and Secondary Caches

Page Attribute (Write-back policy)	Processor Configuration	State of Data Cache Line Being Replaced			
		No-Secondary-Cache Mode		Secondary-Cache Mode	
		Clean/Invalid	Dirty	Clean/Invalid	Dirty
Noncoherent	All R4000 models	NCR	NCR/W	NCR	NCR-W
Exclusive (read and write invalidate)	R4000SC R4000MC	N/A	N/A	R_{Ex}	R_{Ex}-W
Shareable (write invalidate)	R4000MC	N/A	N/A	R	R-W
Update (write update)	R4000MC	N/A	N/A	R	R-W

NCR.................. Processor noncoherent block read request

NCR/W .... Processor noncoherent block read request followed by processor block write request

NCR-W .... *Cluster*: Processor noncoherent block read request with write forthcoming followed by processor block write request

R........................ Processor coherent block read request

R-W .... *Cluster*: Processor coherent block read request with write forthcoming followed by processor block write request

R_{Ex} .... Processor coherent block read request with exclusivity

R_{Ex}-W............... *Cluster*: Processor coherent block read request with exclusivity and write forthcoming followed by processor block write request

Secondary-Cache Mode

In secondary-cache mode, if the current cache line does not have to be written back and the coherency attribute for the page that contains the requested cache line is not exclusive, the processor issues a coherent block read request for the cache line that contains the data element to be loaded.

If the current cache line needs to be written back and the coherency attribute for the requested cache line is sharable or update, the processor issues a cluster. The cluster consists of a coherent block read-with-write-forthcoming request for the cache line that contains the data element to be loaded, followed by a block write request for the current cache line.

If the current cache needs to be written back and the coherency attribute for the page containing the requested cache line is exclusive, the processor issues a cluster consisting of an exclusive read-with-write-forthcoming request, followed by a write request for the current cache line.

Table 12-3 lists these actions.

No-Secondary-Cache Mode

In no-secondary-cache mode, if the cache line must be written back on a load miss, the read request is issued and completed before the write request is handled. The processor takes the following steps:

1. The processor issues a noncoherent read request† for the cache line that contains the data element to be loaded.
2. The processor then waits for an external agent to provide the read response.

If the current cache line must be written back, the processor issues a write request to save the dirty cache line in memory.

† Only noncoherent and uncached attributes are supported in no-secondary-cache mode.

Store Miss

When a processor store misses in both the primary and secondary caches, the processor must obtain, from the external agent, the cache line that contains the target location of the store. The processor examines the coherency attribute in the TLB entry for the page (TLB page coherency attributes are listed in Chapter 4) that contains the requested cache line to see if the cache line is being maintained with either a write invalidate or a write update cache coherency protocol.

The processor then executes one of the following requests:

- If the coherency attribute is either *sharable* or *exclusive*, a write invalidate protocol is in effect, and a coherent block read that requests exclusivity is issued.
- If the coherency attribute is *update*, a write update protocol is in effect and a coherent block read request is issued.
- If the coherency attribute is *noncoherent*, a noncoherent block read request is issued.

Table 12-4 shows the actions taken on a store miss to primary and secondary caches.

Table 12-4 Store Miss to Primary and Secondary Caches

Page Attribute (Write-back Policy)	Processor Configuration	State of Data Cache Line Being Replaced					
		No-Secondary-Cache Mode		Secondary-Cache Mode			
		Clean/ Invalid	Dirty	Clean/ Invalid		Dirty	
Noncoherent	All R4000 models	NCR	NCR/ W	NCR		NCR-W	
Exclusive (write invalidate)	R4000SC R4000MC	N/A	N/A	R_{Ex}		R_{Ex}-W	
Shareable (write invalidate)	R4000MC	N/A	N/A	R_{Ex}		R_{Ex}-W	
Update (write update)	R4000MC	N/A	N/A	Dis(1) R/U	En(2) R-PU	Dis(1) R-W/U	En(2) R-PU-W

NCR ....Processor noncoherent block read request

NCR/W.............Processor noncoherent block read request followed by processor block write request

NCR-W..............*Cluster*: Processor noncoherent block read request with write forthcoming followed by processor block write request

R_{Ex}......................Processor coherent block read request with exclusivity

R_{Ex}-W ....*Cluster*: Processor coherent block read request with exclusivity and write forthcoming followed by processor block write request

R/U....................Processor coherent block read request followed by processor update request (if read response data is shared or dirty shared)

R-PU ....*Cluster*: Processor coherent block read request followed by processor potential update request

R-PU-W ....*Cluster*: Processor coherent block read request followed by processor potential update request, followed by processor block write request

R-W/U ....*Cluster*: Processor coherent block read request with write forthcoming followed by processor block write request, followed by processor update request (if read response data is shared or dirty shared)

Dis(1) ....Potential update disable [**Modebit(20)**: **PotUpdDis** = 1]

En(2)....................Potential update enable [**Modebit(20)**: **PotUpdDis** = 0]

Secondary-Cache Mode

In secondary-cache mode, if the new cache line replaces a current cache line that is in either the dirty exclusive or dirty shared state, the current cache line must be written back before the new line can be loaded in the primary and secondary caches. The processor requests issued are a function of the page attributes listed below.

Noncoherent Page Attribute

If the current cache line must be written back, and the coherency attribute for the requested cache line is noncoherent, the processor issues a cluster consisting of a noncoherent block read-with-write-forthcoming request for the cache line that contains the store target location, followed by a block write request for the current cache line.

If the current cache line does not need to be written back and the coherency attribute for the page that contains the requested cache line is noncoherent, the processor issues a noncoherent block read request for the cache line that contains the store target location.

Sharable or Exclusive Page Attribute

If the current cache line must be written back and the coherency attribute for the page that contains the requested cache line is sharable or exclusive, the processor issues a cluster consisting of a coherent block read request with exclusivity and write forthcoming, followed by a processor block write request for the current cache line.

If the current cache line does not need to be written back and the coherency attribute for the page that contains the requested cache line is sharable or exclusive, the processor issues a coherent block read request that also requests exclusivity.

Update Page Attribute

If the current cache line must be written back and the coherency attribute for the page that contains the requested cache line is update, and potential updates are enabled, the processor issues a cluster consisting of a coherent block read-with-write-forthcoming request, followed by a potential update request, followed by a write request for the current cache line.

If the current cache line does not need to be written back, the coherency attribute for the page that contains the requested cache line is update, and potential updates are enabled, the processor issues a cluster consisting of a read request, followed by a potential update request.

In an update protocol, the cache line requested by a processor coherent read request can be returned in a shared state; the processor then has to issue an update request before it can complete a store instruction. A potential update issued with a read request in a cluster allows the external agent to anticipate the read response on the system bus. If the read response is in a shared state, the required update is quickly transmitted to the rest of the system. This provides the processor with the acknowledge and allows the processor to complete the store instruction as rapidly as possible.

Without the potential update request, the response data must be returned to the processor. If the line is returned in the shared or dirty shared state, the processor issues an update request, which must then be forwarded to the system bus before an acknowledge can be returned to the processor.

Note that potential updates behave as if they have not yet been issued by the processor. Potential updates are not subject to cancellation, and do not require an acknowledge. When a potential update is nullified, the processor behaves as if no update request was ever issued; when a potential update becomes compulsory, the processor behaves as if it had issued an update request at that instant.

Compulsory Update: If the processor issues a cluster that contains a potential update, and the response data for the read request is returned with an indication that it must be placed in the cache in either a shared or dirty shared state, the potential update then becomes compulsory. Once a potential update becomes compulsory, it is subject to cancellation, and the processor requires an acknowledge for the update request. The external agent must forward the update to the system, then signal the acknowledge to the processor when the update is complete. The processor will not complete the store until it has received an acknowledge for the update request.

Nullifying a Potential Update: If the processor issues a cluster that contains a potential update, and the response data for the read request is returned in either a clean exclusive or dirty exclusive state, the potential update is nullified. Once a potential update has been nullified, the external agent must discard the update. The processor does not wait for or expect an acknowledge to a potential update that has been nullified. It is not correct to assert either **IvdAck*** or **IvdErr*** in this situation.

If the read response data is returned in either the clean exclusive or dirty exclusive state, the processor cannot issue an update request. It is assumed that the external agent will take the appropriate action to change the state of the cache line to invalid in other caches.

An external request indicating processor update cancellation can be issued when a processor read is not pending or when compulsory update is unacknowledged. Processor state is undefined if a cancellation is signaled on an external coherence request to the processor when a processor read is pending, or there is no unacknowledged compulsory update.

No-Secondary-Cache Mode

The processor issues a read request for the cache line that contains the data element to be loaded, then awaits the external agent to provide read data in response to the read request. Then, if the current cache line must be written back, the processor issues a write request for the current cache line.

In no-secondary-cache mode, if the new cache line replaces a current cache line whose *Write back* (*W*) bit is set, the current cache line moves to an internal write buffer before the new cache line is loaded in the primary cache.

Store Hit

This section describes store hits in both secondary-cache and no-secondary-cache mode.

Secondary-Cache Mode

When the processor hits in the secondary cache, on a line that is marked either shared or dirty shared, the processor must issue an update or invalidate request and then wait to receive an acknowledge, before the store is complete. The processor checks the coherency attribute in the TLB for the page containing the cache line that is target of the store, to determine if the cache line is managed by either a write invalidate or write update cache coherency protocol.

- If the coherency attribute is sharable or exclusive, a write invalidate protocol is in effect, and the processor issues an invalidate request. The processor cannot complete the store until the external agent signals an acknowledge for this invalidate request.
- If the coherency attribute is update, a write update protocol is in effect, and the processor issues an update request. The processor cannot complete the store until the external agent signals an acknowledge for this update request.

No-Secondary-Cache Mode

In no-secondary-cache mode, all lines are set to the dirty exclusive state. This means store hits cause no bus transactions.

Uncached Loads or Stores

When the processor performs an uncached load, it issues a noncoherent doubleword, partial doubleword, word, or partial word read request. When the processor performs an uncached store, it issues a doubleword, partial doubleword, word, or partial word write request.

External requests have a higher priority than uncached stores. When using the uncached store buffer on an R4400 processor, it is possible for the external agent to receive cached and uncached stores out of program order, as the example below illustrates. Figure 12-14 shows a cached and uncached store instruction sequence:

```
SW  r2, (r3)                    # uncached store
SW  r4, (r5)                    # cached store
```

Figure 12-14 R4400 Processor Cached and Uncached Store Sequence

Referring to Figure 12-14, suppose an external intervention or snoop is issued to the R4400 processor while the uncached store is still in the store buffer (the uncached store data has not yet been stored off-chip). The cached store from Figure 12-14 has hit in the primary cache and is in the tag check (TC) stage of the pipeline (see Chapter 3 for a description of the pipeline stages). In this case, the external agent sees the state of the internal caches after the cached store but *before* the result of the uncached store is available off the chip. Figure 12-15 shows how a SYNC instruction can force the uncached store to occur before the cached store.

```
SW  r2, (r3)                    # uncached store
SYNC
SW  r4, (r5)                    # cached store
```

Figure 12-15 R4400 Processor Cached and Uncached Stores, Using SYNC

CACHE Operations

The processor provides a variety of CACHE operations to maintain the state and contents of the primary and secondary caches. During the execution of the CACHE operation instructions, the processor can issue either write requests or invalidate requests.

Load Linked Store Conditional Operation

Generally, the execution of a Load Linked Store Conditional instruction sequence is not visible at the System interface; that is, no special requests are generated due to the execution of this instruction sequence.

There is, however, one situation in which the execution of a Load Linked Store Conditional instruction sequence is visible, as indicated by the *link address retained* bit during a processor read request, as programmed by the **SysCmd(2)** bit. This situation occurs when the data location targeted by a Load Linked Store Conditional instruction sequence maps to the same cache line to which the instruction area containing the Load Linked Store Conditional code sequence is mapped. In this case, immediately after executing the Load Linked instruction, the cache line that contains the link location is replaced by the instruction line containing the code. The link address is kept in a register separate from the cache, and remains active as long as the *link* bit, set by the Load Linked instruction, is set.

The *link* bit, which is set by the load linked instruction, is cleared by a change of cache state for the line containing the link address, or by a Return From Exception.

In order for the Load Linked Store Conditional instruction sequence to work correctly, all coherency traffic targeting the link address must be visible to the processor, and the cache line containing the link location must remain in a shared state in every cache in the system. This guarantees that a Store Conditional executed by some other processor is visible to the processor as a coherence request, changing the state of the cache line containing the link location.

To accomplish this, a read request issued by the processor, causing the cache line containing the link location to be replaced. In the mean time, the *link address retained* bit is set, indicating the link address is being retained. This informs the external agent that, although the processor has replaced this cache line, the processor must still see any coherence traffic that targets this cache line.

Any snoop or intervention request that targets a cache line which is not present in the cache—but for which the snoop or intervention address matches the current link address while the link bit is set—returns an indication that the cache line is present in the cache in a shared state. This is consistent with the coherency model, since the processor never returns data, in response to an intervention request, for a cache line that is in the shared state. The shared response guarantees that the cache line containing the link location remains in a shared state in all other processor's caches, and therefore that any other processor attempting a store conditional to this link location must issue a coherence request in order to complete the store conditional.

For more information, refer to Chapter 11, or see the specific Load Linked and Store Conditional instructions described in Appendix A.