APPLICATION NOTE

MIPS R4000 Synchronization Primitives

Duk Chun & Shabbir Latif
Technology Products Group
MIPS Technologies, Inc.
2011 N. Shoreline Blvd.
Mountain View, CA 94039

Publication No.: AP004
Publication Date: April, 1993

The R4000 /R4400 is available from the following manufacturers:

Integrated Device Technology, Inc. (Attention - RISC Microprocessor Marketing)
2975 Stender Way
Santa Clara, CA 95052-8015
Tel: (408) 727 6116

LSI Logic Corporation (Attention - MIPS Division)
1551 McCarthy Blvd.
Milpitas, CA 95035
Tel: (408) 433 8000

NEC Corporation (Attention - Microcomputer Division)
401 Ellis St.
Mountain View, CA 94039
Tel: (415) 960-6000

Performance Semiconductor Corporation (Attention - Microprocessor Marketing)
610 E. Weddell Drive
Sunnyvale, CA 94089
Tel: (408) 734 9000

Siemens Components Inc. (Attention - Integrated Circuit Division)
10950 Tantau Ave.
Cupertino, CA 95014
Tel: (408) 777-4500

Toshiba Corporation
9740 Irvine Blvd.
Irvine, CA 92713
(714) 583-3000

Table of Contents

Introduction

1

In a multiprocessing system, it is essential to have a way in which two or more processors working on a common task can each execute programs without corrupting the other's sub-tasks. Sometimes, it is also necessary to restrict the number of processors that can access a particular section of memory (for example, in the case when multiple licensees are allowed to access the same code). Synchronization, an operation that guarantees an orderly access to shared memory, must be implemented for a properly functioning multiprocessing system.

There are many synchronization techniques in existence, but for the sake of simplicity, only two widely used methods will be discussed in this application note. One allows only a single processor to access some part of shared data or code (from now on referred to as "critical section") and another one allows a fixed number of processors to access a critical section. The general description of these techniques is given in chapter: 2. Examples are given in chapter: 4 to show how these techniques can be implemented using *Load Link* and *Store Conditional* instructions, in conjunction with other coherency mechanisms and protocols provided by the R4000 family. Moreover, the functions of *Load Link Address Register* and *Load Link Retained Bit* are discussed in chapter: 5 & 6. These are the two mechanisms, provided by R4000, to support the implementation of the above techniques in special cases.

General description of synchronization techniques

2

Test-and-Set

Test-and-set is a typical way to achieve synchronization where only one processor is allowed to access a critical section. In general, this technique involves using a variable called the *semaphore* and assigning values to this variable for the "lock-off" or "lock-on" state. *Semaphore* can be interpreted as a lock to some critical section. Each processor checks if the lock is off; if so, it tries to lock it by modifying the variable appropriately. Caution: the system must guaranty that only one processor, at a time, can get the ownership to modify the variable and lock it or unlock it. Once a processor gets the lock, it is then allowed to modify restricted data or access the critical section. After it is done, it gets out of the critical section and modifies the *semaphore* to the "lock-off" state so that other processors can get a chance to access it.

*Figure 2-1 i*llustrates a test-and-set synchronization procedure which uses a *semaphore*.

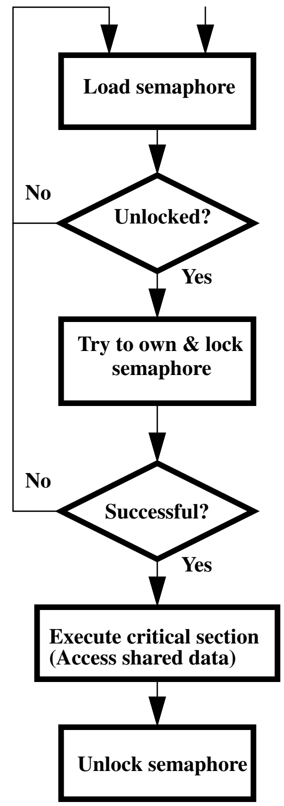


Figure 2-1: Synchronization with test-and-set

Counter

If N processors (N>1) are to be allowed to have permission to access a critical section or access limited resources, a common synchronization technique uses a variable as a *counter* to count number of limited resources or number of permits available. Each processor checks if a permit is free to be used; if available, it tries to modify the *counter* and thus obtain permission. Again it is critical that only one processor, at a time, is allowed to get the ownership and modify the *counter*. All processors after the *N*th processor must wait until a permit becomes available--when any one of the *N* processor exits the critical section and modifies the *counter* to reflect the new status. The flowchart in *Figure 2-2* shows synchronization using a *counter*.

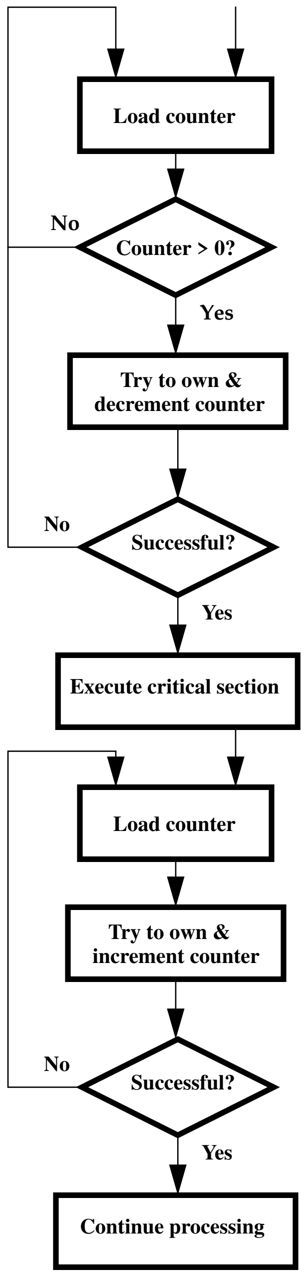


Figure 2-2: Synchronization using a counter

LL and SC

3

MIPS II instructions *Load Linked* (LL) and S*tore Conditional* (SC)[1], in conjunction with the cache coherency mechanism and protocol, provide synchronization support for R4000 processors. The two instructions work very much like their simple counterparts load and store. The LL instruction, in addition to doing a simple load, has the side effect of setting a user transparent bit called the *load link bit(LLbit)*. The *LLbit* forms a breakable link between the LL instruction and a subsequent SC instruction. The SC performs a simple store if and only if the *LLbit* is set when the store is executed. If the *LLbit* is not set, then the store will fail to execute. The success or failure of the SC is indicated in the target register of the store after the execution of the instruction. The target register is loaded with 1 in case of a successful store or it is loaded with 0 if the store was unsuccessful. The *LLbit* is reset upon occurrence of any event that even has potential to modify the lock-variable (like *semaphore* or *counter*) while the sequence of code between LL and SC is being executed. The most obvious case where the link will be broken is when an invalidate occurs to the cache line which was the subject of the load. In this case, some other processor successfully completed a store to that shared line. In general, the link will be broken if following events occur while the sequence of code between LL and SC is being executed:

1. External Update to the cache line containing the lock-variable.
2. External Invalidate to the cache line containing the lock-variable.
3. Intervention or Snoop invalidating cache the line containing the lock-variable.
4. Upon completion of an ERET (return from exception)

The most important features of the LL and SC primitives are:

1. They provide a mechanism for generating all of the common synchronization primitives including test-and-set, counters, sequencers, etc. with no additional overhead.
2. They operate in a fashion so that bus traffic is generated only when the state of the cache line changes; locked words stay in the cache until another processor takes ownership of that cache line.

1. While LL & SC are MIPS II instructions, for MIPS III instructions LLD & SCD also operate similarly on double words.

For additional information regarding LL and SC instructions, cache-coherency mechanism and protocols, please refer to the *MIPS R4000 Microprocessor User's Manual*.

Examples Using LL and SC

4

In this chapter, it is shown how the LL and SC instructions can be used to implement the techniques discussed previously. The flowchart shows general methodology and an example of implementation code is listed next to the corresponding flow symbol, with comments next to the code line. Possible actions taken by the processor that are not as a direct result of the instructions are written in parenthesis.

Implementation of Test-and-Set Technique

Figure 4-1 shows how test-and-set can be implemented using LL and SC instructions.

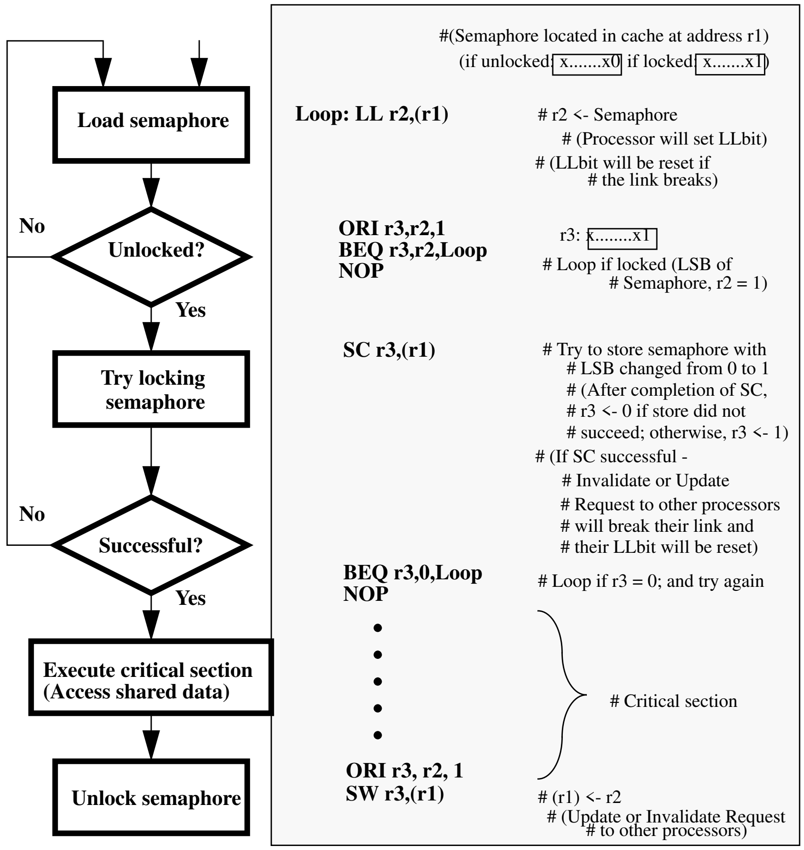
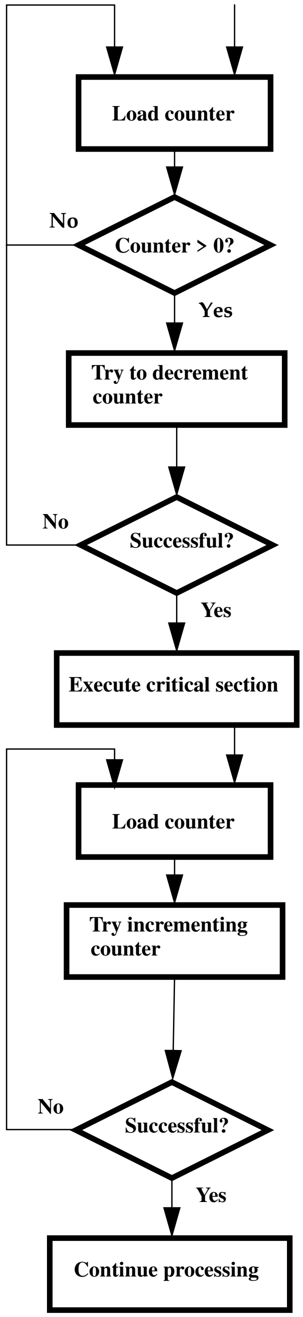


Figure 4-1: Test-and-set using LL and SC

Implementation of Counter Technique

Synchronization using a *counter* is shown in *Figure 4-2*.




```
                    #(Counter located in cache
                    # at address r1)
Loop1: LL r2,(r1)   # r2 <- Counter
                    # (Processor will set LLbit)
                    # (LLbit will be reset if
                    # the link breaks)
     BLEZ r2,Loop1  # Loop if no permit available
     NOP            # r2 < or = 0)
     SUB r3,r2,1    # Decrement the permit available
     SC r3,(r1)     # and try to own it
                    # (After completion of SC,
                    # r3 <- 0 if store did not
                    # succeed; otherwise, r3 <- 1)
                    # (If SC successful -
                    # Invalidate or Update
                    # Request to other processors
                    # will break their link and
                    # their LLbit will be reset)
     BEQ r3,0,Loop1 # Loop if r3 = 0 and try again
     NOP
       •
       •            # Critical section
       •
       •
Loop2: LL r2,(r1)   # r2 <- Counter
                    # (Processor will set LLbit)
                    # (LLbit will be reset if
                    # the link breaks)
     ADDr3,r2,1     # Increment the permit No.
     SC r3,(r1)     # and try to release the permit
                    # (After completion of SC,
                    # r3 <- 0 if store did not
                    # succeed; otherwise, r3 <- 1)
                    # (If SC successful -
                    # Invalidate or Update
                    # Request to other processors
                    # will break their link and
                    # their LLbit will be reset)
     BEQ r3,0,Loop2 # Loop if r3 = 0 and try again
     NOP
```

Figure 4-2: Counter using LL and SC

For both synchronization mechanisms to work properly, the cache line which contains the word addressed by (r1) in the LL instruction, must not be uncached or noncoherent. If the line is not in the cache, then it will be brought in from another processor or the memory. If the line is in the cache of only one processor it could exist as *clean exclusive*; but, unless more than one processor has the line, the synchronization question doesn't arise. However, as soon as another processor will request a line it will be changed to *shared* state.

In the implementation of the Test and Set technique, shown in figure: 4-1, the "logic 1" in the LSB of the *semaphore* at location (r1) represents that the "lock is ON"; a "logic 0" in the same location represents that the "lock is OFF". One of the scenario could be that the locked *semaphore* in the cache of Processor #1 is in the *shared* state; this means some other processor owns the lock. If the processor #1 is trying to lock the *semaphore*, it will keep checking the *semaphore* and wait for the owner, say processor #2, to unlock it. When unlocked by the processor #2, the processor #1 will lock it by writing the "logic 1" in LSB, using SC instruction. If the processor #1's cache-line, containing the *semaphore*, is found locked and in the dirty *shared* state, means, either processor #1 or#2 could own the lock. If the processor #1, erroneously, didn't unlock the cell after executing SC, it might loop forever because it owns the lock and the cache-line; so no other processor can modify it. However, to avoid this problem, one could include in the *semaphore* the number of the processor which owns the lock. The processor #1 would then check not only if the *semaphore* is unlocked but if it is locked by processor #1. In this case, the processor, trying for the lock, would not loop if it owned the lock.

Between the LL and SC instruction, while the processor #1 is trying to lock the variable, if another processor, say #2, succeeds in locking it (i.e. the SC in the processor #2 succeeds), an "Invalidate Request" or an "Update Request" will come from the processor #2. This will cause the LLBit in the processor #1 to be reset and the SC in the processor #1 will not succeed. Regardless of whether the state of the *semaphore* is *shared* or *dirty shared*, if the SC succeeds, because LL-Bit stayed in logical 1 state, then the processor #1 (the processor with successful SC) will send an Invalidate or an Update to other processors and the state of the cache line in the processor #1 will be modified depending on the page attribute. The LLBit will guaranty that only one processor, at a time, modify the *semaphore* used as a lock. The processor will check the success of the SC by reading the target register used by the SC instruction; since the content of the LL-Bit is transferred into this register after the completion of SC. If the processor finds that the content of the target register r3 is not 0, it enters the critical section of the code.

Load Link Address (LLAddr) Register

5

During the time the LL/SC sequence is being executed, the processor must have access to the address of the target line of the LL/SC instruction; to support externally initiated coherency operations to that line. Typically, this line will be in the cache; however, it is possible that this line could be replaced during the time after the LL instruction executed and before the SC is executed; for example, if the instruction area containing a LL/SC sequence is mapped to the data location targeted by the LL and SC instructions used in that sequence. In such a case, the processor, after performing the Write-Back and Invalidate or Update, will save the address of the targeted line into a register called *"Load Link Address (LLAddr) Register"* and will consider it retained in a *shared* state. Thus, if the LL-Bit is set and the target line is not found in the cache, the processor will always compare the content of the LLAddr Register with the address of any external request received. If necessary, it will reset the LL-Bit and if there is an Intervention or Snoop to this target line, the processor will return an indication that the cache line is present in the cache in a *shared* state. A shared indication, even though the processor does not have the data, is consistent 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 that contains the link location will remain in a *shared* state in all other processor's caches. Thus, any other processor attempting a *store conditional* to this link location must issue a coherence request in order to complete the *store conditional* instruction.

Link Address Retained Bit

6

If a system is designed in which the external agent keeps track of the cache coherency (for example, using a directory based protocol or using a duplicate set of secondary tags), it would be necessary to have some mechanism by which the external agent is notified that a line is retained in a *shared* state, even if it is not in the cache. The R4000 notifies the external agent of the above situation during the read request it issues, as a result of the miss on the cache line that contains the link location, while the link bit is set. During this request, bit #2 on the SysCmd bus, assigned as *Link Address Retained Bit*, is set to 1. The external agent interprets this bit as the indication that the line being replaced in the cache is actually retained in the LLAdr Register as *shared* and the processor must see any coherent traffic that targets this cache line.

Summary

7

In any multiprocessing system, synchronous operations are needed to prevent one processor from corrupting other's task or to restrict fixed number of processors to access a section of code or data. R4000 provides *Load Link (LL)* and *Store Conditional (SC)* instructions and the coherency mechanism to support such synchronous operations. Two techniques were discussed as examples to show how these features provided by R4000 can be used. "Test and Set technique", which uses *semaphore*, provides a way to restrict only one processor to access a section of the shared memory and the "Counter" technique provides a way to allow a maximum number of processors to access shared data or code.

R4000 uses an internal *Load Link Address (LLAdr) Register* to support synchronous operations even if the cache line, which is the target of the LL and SC instructions, was replaced by another line during the operation of the load link store conditional sequence.

R4000 also provides an indication to the external agent, using *Link Address Retained Bit*, that a linked cache line, which was replaced in the cache, is retained as *shared* cache line in the LLAdr Register. This allows the external agent to use coherency schemes like "directory based protocol" or schemes using dual secondary tags.

Reader's Comments

Please FAX your comments about this document to:

Anjaneya Thakar: Fax: (415) 390-6170

Address: P.O. Box 7311, MS 952, Mountain View, CA 94039-7311

Areas of Improvement:

Errors:

Reader Information:

Name:

Company:

Address:

Phone: FAX:

Thank you for your feedback.