称为测试并加锁（Test and Set Lock），它将一个内存字lock读到寄存器RX中，然后在该内存地址上存一个非零值。读字和写字操作保证是不可分割的，即该指令结束之前其他处理器均不允许访问该内存字。执行TSL指令的CPU将锁住内存总线，以禁止其他CPU在本指令结束之前访问内存。

着重说明一下，锁住存储总线不同于屏蔽中断。屏蔽中断，然后在读内存字之后跟着写操作并不能阻止总线上的第二个处理器在读操作和写操作之间访问该内存字。事实上，在处理器1上屏蔽中断对处理器2根本没有任何影响。让处理器2远离内存直到处理器1完成的惟一方法就是锁住总线，这需要一个特殊的硬件设施（基本上，一根总线就可以确保总线由锁住它的处理器使用，而其他的处理器不能用）。

为了使用TSL指令，要使用一个共享变量lock来协调对共享内存的访问。当lock 为0时，任何进程都可以使用TSL指令将其设置为1，并读写共享内存。当操作结束时，进程用一条普通的move指令将lock的值重新设置为0。

这条指令如何防止两个进程同时进入临界区呢？解决方案如图2-25所示。假定（但很典型）存在如下共4条指令的汇编语言子程序。第一条指令将lock 原来的值复制到寄存器中并将lock 设置为1，随后这个原来的值与0相比较。如果它非零，则说明以前已被加锁，则程序将回到开始并再次测试。经过或长或短的一段时间后，该值将变为0（当前处于临界区中的进程退出临界区时），于是过程返回，此时已加锁。要清除这个锁非常简单，程序只需将0存入lock即可，不需要特殊的同步指令。

```
enter_region:
    TSL REGISTER,LOCK      | 复制锁到寄存器并将锁设为1
    CMP REGISTER,#0        | 锁是零吗?
    JNE enter_region       | 若不是零，说明锁已被设置，所以循环
    RET                    | 返回调用者，进入了临界区

leave_region:
    MOVE LOCK,#0           | 在锁中存入0
    RET                    | 返回调用者
```

图2-25 用TSL指令进入和离开临界区

现在有一种很明确的解法了。进程在进入临界区之前先调用enter_region，这将导致忙等待，直到锁空闲为止，随后它获得该锁并返回。在进程从临界区返回时它调用 leave_region，这将把lock设置为0。与基于临界区问题的所有解法一样，进程必须在正确的时间调用enter_region和 leave_region，解法才能奏效。如果一个进程有欺诈行为，则互斥将会失败。

一个可替代TSL的指令是XCHG，它原子性地交换了两个位置的内容，例如，一个寄存器与一个存储器字。代码如图2-26所示，而且就像可以看到的那样，它本质上与TSL的解决办法一样。所有的Intel x86 CPU在低层同步中使用XCHG指令。

```
enter_region:
    MOVE REGISTER,#1       | 在寄存器中放一个1
    XCHG REGISTER,LOCK     | 交换寄存器与锁变量的内容
    CMP REGISTER,#0        | 锁是零吗?
    JNE enter_region       | 若不是零，说明锁已被设置，因此循环
    RET                    | 返回调用者，进入临界区

leave_region:
    MOVE LOCK,#0           | 在锁中存入0
    RET                    | 返回调用者
```

图2-26 用XCHG指令进入和离开临界区

### 2.3.4 睡眠与唤醒

Peterson解法和TSL或XCHG解法都是正确的，但它们都有忙等待的缺点。这些解法在本质上是这样的：