非常感谢你的指正！你说得完全正确，我的表述不够精确。你的教科书上的定义是标准的、准确的。让我来重新梳理和解释这两个关键指令，并澄清它们与自旋锁的关系。

你的书本定义是经典的，我们来以此为基础。

---

1. Test-and-Set 指令

你的书本中的 `TS` 函数精确地描述了 Test-and-Set 指令的语义：

```
boolean TS(boolean *lock) {
    boolean old;
    old = *lock;  // "Test": 测试锁的当前状态
    *lock = TRUE; // "Set": 无论它之前是什么，都将其设置为TRUE（占用状态）
    return old;   // 返回测试到的旧状态
}
```

- 核心特征：
  1. 它读取一个内存位置的值。
  2. 它写入一个固定的值（通常是 `1`）到那个内存位置。
  3. 它返回该内存位置的旧值。
- 原子性：最关键的是，这读-改-写三步操作是由一条不可中断的硬件指令完成的。
- 用途：`TS` 是实现自旋锁最直观的指令之一。返回值是关键：
  - 如果返回 `FALSE`，表示锁之前是空闲的，你成功地获取了它（因为你把它设为了 `TRUE`）。
  - 如果返回 `TRUE`，表示锁之前就是被占用的，你这次获取失败。

用 Test-and-Set 实现自旋锁：

```
// 自旋锁数据结构
typedef int spinlock_t;
#define FALSE 0
#define TRUE 1

void spin_lock(spinlock_t *lock) {
    // 一直尝试TS指令，直到它返回FALSE（表示我们成功获取了锁）
    while (TS(lock) == TRUE) {
        // 自旋等待...
    }
}

void spin_unlock(spinlock_t *lock) {
    *lock = FALSE; // 释放锁很简单，就是将其设置为FALSE
}
```

---

2. Swap 指令（Exchange / XCHG）

你的书本中的 `swap` 函数也精确地描述了 Swap 或 Exchange 指令的语义：

```
void swap(boolean *a, boolean *b) {
    boolean temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

- 核心特征：原子地交换两个操作数（通常是一个内存位置和一个寄存器）的值。
- 原子性：同样的，这个交换操作是一条硬件指令，不会被中断。
- 用途：它同样可以非常高效地实现自旋锁。思路是：准备一个本地变量，其值为 `TRUE`（表示“我想占用”），然后不断地用它去和锁变量的值做交换。

用 Swap 指令实现自旋锁：

```
void spin_lock(spinlock_t *lock) {
    boolean my_val = TRUE; // 我准备好了“占用”的值

    // 如果锁是空闲的(FALSE)，交换后my_val会变成FALSE，lock会变成TRUE（被我占用）
    // 如果锁是占用的(TRUE)，交换后my_val会变成TRUE，lock还是TRUE（没变化）
    while (my_val == TRUE) {
        swap(lock, &my_val); // 原子地交换锁和本地变量的值
    }
    // 循环退出的唯一条件：my_val变成了FALSE。
    // 这意味着在某次交换中，我们从锁那里拿到了FALSE，同时把TRUE塞给了锁。
    // 这就标志着我们成功获取了锁！
}

void spin_unlock(spinlock_t *lock) {
    *lock = FALSE;
}
```

这个实现可能比 `TS` 的实现更难理解一些，但功能是完全等效的。

核心区别与总结

特性	Test-and-Set （**TS**）	Swap （**XCHG**）
操作	读 → 写固定值 → 返回旧值	交换两个操作数的值
实现锁的逻辑	检查返回值。如果返回 `FALSE` 就成功了。	检查本地变量。如果本地变量变为 `FALSE` 就成功了。
直观性	更直观。“测试并设置”这个名字直接体现了其用途。	稍绕一些。需要理解“通过交换来试探”的思路。

它们的共同点和最终效果：
两者都是最基本的原子读-改-写指令。它们的存在是为了解决同一个核心问题：将“检查值”和“更新值”这两个操作合并为一个不可分割的原子操作，从而避免竞态条件。

在实现自旋锁时，`TS` 和 `Swap` 在功能和性能上是等效的。不同的CPU架构可能更倾向于提供其中一种或两种都提供。x86系列的 `XCHG` 指令非常著名，而许多其他架构

也提供类似 `TS` 功能的指令。

与现代指令的关系

你之后会学到 `Compare-and-Swap`，它是一个更强大、更通用的原语。可以认为：

- `Test-and-Set` 是 `Compare-and-Swap` 的一个特例。
  - `TS(lock)` 等价于 `CAS(lock, expected=任何值, new_value=TRUE)`，但它不关心旧值是什么，它总是设置为 `TRUE` 并返回旧值。
- `Compare-and-Swap` 是条件版本的写操作：“只有当你现在的值是我期望的，我才会修改你”。这使它不仅能实现锁，还能实现复杂的无锁算法。

总结：
你的书本定义是完全正确的。感谢你的指正，这让我们能更准确地进行讨论。`Test-and-Set` 和 `Swap` 是两种略有不同但功效相同的原子指令，它们是构建自旋锁和许多操作系统同步原语的硬件基础。