Busy-Wait Solutions with H/W Support

Where are we?

- Disable Interrupts
  - **Effectively stops scheduling other activities.**
- Busy-wait/spinlock Solutions
  - **Pure software solutions**
  - **Integrated hardware-software solutions**
- Blocking Solutions

- Complications arose because we had atomicity only at the granularity of a machine instruction, and what a machine instruction could do was limited.

- Can we provide specialized instructions in hardware to provide additional functionality (with an instruction still being atomic)?

Specialized Instructions

- Bool Test&Set(bool)
- Swap (bool, bool)

machine instructions

- Note that these are machine/assembly instructions, and are thus atomic.
- Parameters are passed by reference.

Test&Set

```
Atomic bool Test&Set(bool x) {
  bool temp;

     temp = x;
    x = TRUE;
    return (temp);
}
```

- Note that “=x” and “x=“ would have required at least 1 machine instruction each without this specialized instruction.

Using Test&Set()

not guarantee Bounded waiting

```
Bool lock = FALSE;

Enter_CS() {
  while (Test&Set(lock))
      ;
}

Exit_CS() {
  lock = FALSE;
}
```

NOTE: This solution does not guarantee bounded Waiting.

EXERCISE: Enhance this Solution for bounded waiting

how

Bool lock = FALSE

flag[0]

Swap()

```
Atomic Swap(bool a, bool b) {
    temp = a;
    a = b;
    b = temp;
}
```

- Again, all this is done atomically!

Using swap()

```
Bool lock;

Enter_cs() {
    key = TRUE;
    while (key == TRUE) swap(key,lock);
}
Exit_cs() {
    lock = FALSE;
}
```

每43

① if lock == true
after enter while, swap(key, lock)
key = T, lock = T, and keep looping until
other set lock = False

② if lock = False.
after enter while, swap(key, lock).
key = F, lock = T, it will grap the lock
and enter CS

```
Test & set (bool x) {
    bool temp = x;
        x = True;
        return temp; }

swap (bool a, bool b) {
    bool temp = a;
        a = b;
        b = temp;
}
```