Go - Race condition

Suppose this Go program:

```go
package main

import (
    "fmt"
    "time"
)

func rest(number *int, with int) {
    time.Sleep(500 * time.Microsecond)
    *number = (*number) - with
    fmt.Printf("Current value after rest %d: %d\n", with, *number)
}

func sum(number *int, with int) {
    time.Sleep(500 * time.Microsecond)
    *number = (*number) + with
    fmt.Printf("Current value after sum %d: %d\n", with, *number)
}
func main() {
    initialNumber := 100
    fmt.Printf("Initial number: %d\n", initialNumber)
    go rest(&initialNumber, 2)
    go sum(&initialNumber, 5)

    var exit string
    fmt.Scan(&exit)
}
```

Notice that I have added a sleep at the beginning of the two goroutines to force an async state.

Here we have two possibles overleaping, both with this initial conditions:

```
initialNumber := 100
```

What's race condition

The race condition happens when a threads depends in some manner of another thread. And because the time and order of execution of the thread's code in a CPU isn't deterministic we could got different results executing the same code.

In the code above, we have a code that can give us different results, why? It's because we have a race condition. We share a pointer that could be modified by two goroutines in a non-deterministic time.

Let me explain that with these examples:

Overleap #1

rest(*initialValue, 2)	**sum(*initialValue, 5)**
`100 - 2 , prints (98)`	
	`98 + 5, prints (103)`

I this first case, the code is executed as we written it. So first the rest of 2 is executed then the value of the pointer to `initialNumber` changed to 98, then we add 5 to the value and the result is 103.

Overleap #2

rest(*initialValue, 2)	**sum(*initialValue, 5)**
	`100 + 5, prints (105)`
`105 - 2 , prints (103)`	

In this second case the execution is inverse, first we add 5 to the initial value, and after the goroutine rest 2. And the result is different from the first one.

We can't determine how it will work in each execution.