1. Maschinenbefehle zur Konstruktion von Schlössern:

a) Erläutern Sie die Details im Assemblercode zur Realisierung der Funktion `TestAndSet` (Seite 33 im Buch; Hinweis: Fußnote 21).

b) Implementieren Sie ein Schloss mit `FetchAndAdd`. Erläutern Sie die Rolle des Maschinenbefehls `XADD` dabei.

2. Gegeben sei ein unteilbarer Maschinenbefehl `Swap` mit folgender Semantik:

```
func Swap (a, b *bool) {
  *a, *b = *b, *a
}
```

a) Konstruieren Sie ein Eintritts- und Austrittsprotokoll zum gegenseitigen Ausschluss mehrerer Prozesse unter Verwendung von `Swap`. Begründen Sie die Korrektheit Ihrer Lösung.

b) Erläutern Sie die Verwendung durch die Angabe eines Prozesses in Go aus einem selbstgewählten typischen Beispiel.

3. Im folgenden sind vier Schlossalgorithmen zur Realisierung der Eintrittsprotokolle zweier Prozesse in einen kritischen Abschnitt gegeben:

```
func Lock1() {
  for e2 { Null() }
  e1 = true
}
```

```
func Lock1() {
  for f == 2 { Null() }
}
```

```
func Lock1() {
  e1 = true
  f = 2
  for e2 && f == 2 { Null() }
}
```

```
func Lock1() {
  e1 = true
  for e2 { Null() }
}
```

Die entsprechenden Funktionen `Lock2()` sind dual (d. h. durch Vertauschen von 1 und 2) definiert.

a) Welche Rolle spielen die definierten globalen Variablen?

b) Wie sollten diese Variablen initialisiert werden?

c) Welche dieser Protokolle sind fehlerhaft, welche sind korrekt und warum?

4. Untersuchen Sie die Tauglichkeit zur Sperrsynchronisation der folgenden Implementierungen von `Lock/Unlock` (mit `var a, interested [2]bool; var favoured uint`, initial `a[i] == true` und `interested[i] == false` für `i < 2`, `favoured < 2` sowie `p < 2` beim Aufruf):

a)
```
func Lock (p uint) {
  for {
    a[1-p] = ! a[p]
    if a[p] && ! a[1-p] { break }
  }
}
```

```
func Unlock (p uint) {
  a[1-p] = true
}
```

b)
```
func Lock (p uint) {
  for {
    interested[p] = true
    if interested[1-p] {
```

```
func Unlock (p uint) {
  interested[p] = false
}
```

```
        interested[p] = false
      }
      if interested[p] { break }
    }
  }
```

c)
```
func Lock (p uint) {
  interested[p] = true
  for interested[1-p] && favoured != p {
    Null()
  }
}
```

```
func Unlock (p uint) {
  interested[p] = false
}
```