Praktikum zu Rechnerarchitektur

In diesem Praktikum werden Sie in die Programmierumgebung MARS (MIPS Assembler and Runtime Simulator) eingeführt. Es handelt sich um eine einfache Umgebung, in der Sie MIPS-Assemblersprachprogramme entwickeln können. Mit dem MARS-Simulator können Sie MIPS-Assemblerprogramme bearbeiten, assemblieren, ausführen und interagieren. Die Assemblerprogramme sind Textdateien mit der Erweiterung ".asm", zum Beispiel "lab1.asm".

MARS ist eine Software, die in der Programmiersprache Java geschrieben wurde. Auf den meisten Macs, Windows- und Linux-Computern ist Java nicht installiert, aber Sie können das sog. Java-Runtime-Environment kostenlos herunterladen, z.B. unter
https://www.heise.de/download/product/java-runtime-environment-jre-627

Aufgabe 1

Überführen Sie folgende einfache C-Anweisungen in ein MIPS-Assembler-Programm (Bitte beachten: die C-Anweisungen wurden kompakt notiert, um Platz zu sparen):

```
long  a, b, c;
a = 10; b = 1;
c = a + 0;
b = b * 2;
c = c + b;
```

– Assembler-Programme werden strukturiert formuliert. Die Spalten werden durch Tabulatoren realisiert.
  - Spalte 1: Vereinbarung von symbolischen Namen/Label
  - Spalte 2: mnemotechnische Abkürzung des Maschinenbefehls
  - Spalte 3: Operandenspezifikation des Maschinenbefehls
  - Spalte 4: abgetrennt durch ein # folgt bis zu Zeilenende ein Kommentar

```
26          # Loop to compute each Fibonacci number using the previous two Fib. numbers.
27  loop:   lw   $s3, 0($s0)      # Get value from array F[n-2]
28          lw   $s4, 4($s0)      # Get value from array F[n-1]
29          add  $s2, $s3, $s4    # F[n] = F[n-1] + F[n-2]
```

– für jede Variable ist im Datensegment Speicherplatz zu reservieren. Gleichzeitig kann der Speicherplatz mit einem Wert vorbelegt werden. Dazu ist anzugeben
  - ein symbolischer Name (Label), mit dem der Speicherplatz (und damit der Variablenwert) angesprochen werden kann
  - wie viel Byte für die Aufnahme der Werte zu reservieren sind (`long` = 4 Byte)
  - mit welchem Wert der reservierte Speicherplatz vorbelegt werden soll

die Notierung erfolgt mittels sog. Assembler-Direktiven:

`.align n`	Ausrichten des nächsten Datums auf eine Adresse 2^n
`.ascii str`	die Zeichenfolge `str` wird im Speicher abgelegt, ohne Nullterminierung
`.asciiz str`	wie `.ascii` mit Nullterminierung
`.byte b1,..., bn`	speichere die n Werte in aufeinander folgende Speicherzellen, analog `.half`, `.word`, `.float`, and `.double`
`.data`	nachfolgende Einträge werden im Datensegment abgelegt
`.space n`	allokiere `n` Leer-Bytes im Datensegment
`.text`	Beginn des Textsegmentes

– notieren Sie Text- und Datensegment

```
1  .text   # notiert wird die Folge von Maschinenbefehle
2          # z.B. ein nop-Befehl (no/null operation)
3          # in dieser dürfen keine Speicherplätze für Variablen reserviert werden
4          nop
5
6
7
8  .data   # notiert werden Reservierungsanweisungen für Daten
9
```

– vereinbaren und initialisieren Sie den Speicherplatz für die C-Variablen im Datensegment

– notieren Sie im Textsegment die Maschinenbefehle zu obiger Anweisungssequenz

separat finden Sie eine Übersicht über die Maschinenbefehle, u.a.

add	R_d, R_s, RI	addition	$R_d \leftarrow R_s + RI$
addi	R_d, R_s, imm	addition immediate	$R_d \leftarrow R_s + imm$
sub	R_d, R_s, RI	subtract	$R_d \leftarrow R_s - RI$

dabei bezeichnet R_d ein Destination-, R_s ein Source- und RI ein weiteres Operandenregister oder ein konstanter Integer-Wert.

Bedenken Sie, dass der MIPS-Rechner eine Load&Store-Architektur besitzt, d.h. bevor eine Operation auf Daten vorgenommen werden kann, müssen die Daten zu erst in Register geladen werden, dann können sie verarbeitet werden, wobie das Ergebnis wieder nur in einem Register gespeichert werden kann. Erst danach kann das Ergebnis in den Speicher zurückgeschrieben werden.

– assemblieren Sie Ihr Programm (Überführung in Maschinencode) und simulieren Sie die Ausführung, beachten Sie die Auswirkung der Befehlsausführung auf die Register.

Beachten: für eine Assemblierung ist die vorherige Speicherung des Assemblerprogramms erforderlich.

Aufgabe 2

a. Schreiben Sie ein MIPS-Assemblerprogramm, das zwei Register mit Konstanten (frei Wahl) belegt und dann diese Register multipliziert. Wo steht das Ergebnis?

b. Der syscall-Befehl wird zum Aufrufen von (Betriebs)Systemdiensten verwendet. Die Systemdienste ermöglichen den Zugriff auf die Benutzerkonsole und andere externe Geräte. Der auszuführende Dienst ist eine Nummer, die im Register $v0 abzulegen ist (s. MARS-Help – Syscalls). Gängige Dienste sind: Dienst 1 gibt eine Zahl aus, die in $a0 gespeichert ist; Dienst 4 druckt eine Zeichenkette, die ab der Speicheradresse, die in $a0 steht, beginnt; Dienst 5 liest eine Zahl, die über die Tastatur eingegeben wird, ein und speichert sie in $v0; Dienst 10 hält das Programm an bzw. beendet es.

Verwenden Sie Rahmen aus der Datei `mips2.asm`, um nun zwei Zahlen über Tastatur einzulesen und zu addieren (oder multiplizieren).