# Übungsblatt zum Umgang mit MARS[1] — MIPS Assembler and Runtime Simulator



## 1 Vorbemerkung

Mit diesem Übungsblatt sollen Sie lernen, mit MARS, einem MIPS Assembler and Runtime Simulator, umzugehen. Da MARS ein Java-Programm ist, muss als Voraussetzung Java J2SE 1.5 (oder spätere Version) von *http://java.sun.com/* installiert sein. Auf den Rechnern in den Übungsräumen ist die erforderliche Java-Umgebung bereits eingerichtet.
Einige Dokumentation zur MIPS32-Architektur finden Sie als `.pdf` Files auf der Webseite zu dieser Lehrveranstaltung.
Arbeiten Sie bitte die folgenden Übungen nacheinander am Rechner durch!

## 2 MARS Starten

### 2.1 Download MARS

Zunächst muss das Java-Archiv *Mars.jar* von der Webseite für diese Lehrveranstaltung oder von *http://courses.missouristate.edu/KenVollmar/MARS/* heruntergeladen werden.

### 2.2 Starten unter Linux

Das Kommado zum Starten von MARS lautet

```
java -jar Mars.jar
```

oder

```
java -jar <path>/Mars.jar
```

wenn der File *Mars.jar* nicht in Ihrer *present working directory* liegt, sondern in der Directory `<path>`.

## 3 Mars Oberfläche

Nach erfolgreichem Start von MARS öffnen sich mehrere Fenster:

- Das Hauptfenster in dem Sie *Edit* oder *Execute* auswählen können.
- Ein Fenster für Messages unterhalb des Hauptfensters.
- Ein Fenster mit den Inhalten der Register des Prozessors.

### 3.1 Erstellen eines MIPS-Programms

**Entfernen Sie zunächst im Menüpunkt "Settings" die Markierung bei "Permit extended (pseudo) instructions and formats"!**

Wenn Sie im Haupfenster *Edit* wählen, können Sie eine beliebige Datei erstellen, indem Sie im Menü *File→New* wählen oder den entsprechenden Button in der Buttonleiste.

Erstellen Sie jetzt eine Datei `prog1.asm` mit folgendem Inhalt:

```
     .text
     addi  $8,$0,0
loop:
     addi  $8,$8,1
     j     loop
```

Speichern Sie Ihre Datei über das Menü oder den entsprechenden Button in der Buttonleiste unter dem Namen `prog1.asm`! Bei der Datei handelt es sich um ein sehr einfaches MIPS-Programm:

**.text** (Assembler-Direktive)
: Der Assembler soll für die folgenden Befehle die Instruktionswörter nacheinander in der Text-Sektion, also im Instruktionsspeicher ab Adresse `0x00400000` generieren.

**addi $8, $0, 0** (Maschinenbefehl)
: Das Register GPR[8] wird auf Null gesetzt. Was passiert genau?

**loop:** (Assembler-Direktive)
: Die Adresse der folgenden Instruktion wird mit dem Label `loop` markiert.

**addi $8, $8, 1** (Maschinenbefehl)
: Der Inhalt des Registers GPR[8] (Operand 2) wird als 32-bit Zahl aufgefasst und mit dem Wert 1 (Operand 3) addiert; das 32-bit Ergebnis wird in Register GPR[8] (Operand 1) geschrieben.

**j loop** (Maschinenbefehl)
: Es wird zur Instruktion an der Marke `loop` gesprungen.

### 3.2 Assemblieren eines Programms

Wenn Sie die Datei `prog1.asm` erfolgreich abgespeichert haben, können Sie das Programm über den Menüeintrag *Run→Assemble* oder den entsprechenden Button in das maschinenlesbare Format übersetzen. Das funktioniert natürlich nur, wenn Ihr Programm syntaktisch fehlerfrei ist. Ansonsten produziert der Assembler Fehlermeldungen im *Message* Fenster, und Sie müssen Ihr Programm korrigieren.

Beachten Sie, dass beim Assemblieren eines Programms kein Maschinenbefehl ausgeführt wird! Lediglich die Assembler-Direktiven werden „ausgeführt", und das Ergebnis der Assemblierung ist die Übersetzung der symbolischen Maschinenbefehle in Instruktionswörter an bestimmten Adressen.

Wenn die Assemblierung erfolgreich war, wird im Hauptfenster automatisch von *Edit* zu *Execute* gewechselt, und Sie sehen, an welchen Adressen (Spalte *Address*) der Assembler für die Befehle aus Ihrem symbolischen Programm (Spalte *Source*) maschinenlesbare Wörter (Spalte *Code*) erzeugt hat.

[1] MARS © 2003 – 2014 Pete Sanderson and Kenneth Vollmar

### 3.3 Ausführen eines Programms

Sie können Ihr maschinenlesbares Programm vom Simulator jetzt in zwei Moden ausführen lassen:

**Single Step**
: Über den Menüpunkt *Run→Step* oder den entsprechenden Button wird genau eine Instruktion ausgeführt, nämlich die, deren Adresse im Register PC steht. Sie können sich nach jedem Schritt die Inhalte der Register anschauen. Interessant sind bei `prog1.asm` aber nur GPR[8] und PC. Die meisten Register können auch „per Hand "geändert werden, um die Fehlersuche zu erleichtern.

**Go**
: Über den Menüpunkt *Run→Go* oder den entsprechenden Button läuft das Programm bis ein Fehler auftritt oder bis Sie es angehalten haben.

Weiterhin sind folgende Befehle möglich:

**Reset**
: Wenn das Programm gerade nicht ausgeführt wird, können Sie den Prozessor über den Menüpunkt *Run→Reset* oder den entsprechenden Button in den ursprünglichen Zustand zurücksetzen.

**Pause**
: Während das Programm ausgeführt wird, können Sie es über den Menüpunkt *Run→Pause* oder den entsprechenden Button unterbrechen.

**Stop**
: Während das Programm ausgeführt wird, können Sie es über den Menüpunkt *Run→Stop* oder den entsprechenden Button beenden.

Starten Sie Ihr Programm, lassen Sie es eine Weile laufen (*Run→Go*), unterbrechen Sie es nach zehn Sekunden (*Run→Pause*), und schauen Sie sich die Inhalte der Register an! Wieviele Instruktionen pro Sekunde werden ungefähr simuliert?

Ändern Sie den Inhalt des Registers GPR[8] auf den Wert `0x7ff00000`, und führen Sie das Programm für einige Instruktionen schrittweise aus!

Lassen Sie Ihr Programm jetzt wieder laufen (*Run→Go*)! Was passiert nach einer Weile? Begründung!

### 3.4 Erstellen eines Programms mit einem externen Editor

Sie können ein Assembler-Programm auch mit einem anderen als dem internen MARS-Editor erstellen. Der Name der Datei muss das Suffix `.asm` haben und kann über den Menupunkt *File→Open* oder den entsprechenden Button geöffnet werden.

### 3.5 Ein weiteres MIPS-Programm

Erstellen Sie mit dem MARS-internen oder einem externen Editor folgendes MIPS-Programm (`prog2.asm`):

```
     .text
     ori   $8,$0,0xffff
     addi  $9,$0,-16384
loop:
     add   $8,$8,$9
     j     loop
```

Übersetzen Sie das Programm und führen Sie es schrittweise aus! Beobachten Sie, wie sich die Inhalte der Register nach jeder Instruktion ändern! Machen Sie sich klar, was exakt bei jeder Instruktion passiert! Lassen Sie das Programm dann über den Menüpunkt *Run→Go* eine Weile laufen! Wenn ein Runtime-Fehler auftritt, erklären Sie warum! Nehmen Sie, wenn nötig, die MIPS-Handbücher zu Hilfe!

■