# Projekt 1

Compiler Design Lab
Prof. Dr. Sebastian Hack
Simon Moll, M. Sc.

## MIPS-Potpourri (10 Punkte)

In diesem Projekt werden Sie mehrere Aufgaben in MIPS-Assembler bearbeiten. Zu jeder Aufgabe gibt es ein eigenes Verzeichnis, indem Sie jeweils eine `main.s` vorfinden. Diese beinhaltet die Funktion `main`, die den Einstiegspunkt für Ihr Programm darstellt. Passend zu jedem Unterprogramm, welches Sie schreiben werden, gibt es eine entsprechende Datei, in der Sie die geforderte Funktionalität implementieren müssen.

Achten Sie darauf, dass sich Ihre Lösung ausschließlich in der jeweiligen Datei zu der entsprechenden Aufgabe befindet. Zum Beispiel ist die Lösung für Aufgabe 1 in die Datei `number2text/number2text.s` einzutragen. Das Testsystem wird Änderungen an anderen Dateien (an `main.s` zum Beispiel) sowie von Ihnen hinzugefügte Dateien beim Testen ignorieren. Selbstverständlich können Sie andere Dateien für Ihre eigenen Tests verändern. Erzeugen Sie in den Abgabedateien auch keine Ausgaben, die von der Aufgabenstellung nicht gefordert sind. Andernfalls wird das Testsystem Ihre Abgabe als falsch bewerten.

Beachten Sie, dass Sie die Einstellungen *Assemble all files in directory* und *Initialize Program Counter to global 'main' if defined* im Menü *Settings* in MARS aktivieren müssen, um die Datei `main.s` zu übersetzen, und den Programmeinstiegspunkt auf die Marke `main` zu setzen. Jede `main.s` Datei enthält einen Testfall, der ihr Programm mit sinnvollen Eingaben aufruft. Sie können das Projekt mit Git unter folgender URL beziehen:

`https://prog2scm.cdl.uni-saarland.de/git/project1/<username>`

Ersetzen Sie `<username>` in der URL durch Ihren Benutzernamen von der Webseite. Die Abgabe erfolgt per Einbuchung in das Git-Depot. Wir betrachten ihre letzte Einbuchung bis einschließlich 10.05.2016, 23:59 Uhr als Abgabe. Sie werden per E-mail über die Ergebnisse der regelmäßigen Tests informiert. Zusätzlich werden die Testergebnisse auf die Website hochgeladen. Die Testergebnisse können Ihnen helfen Ihr Projekt zu verbessern. Buchen Sie deshalb früh und regelmäßig in das Git-Depot ein! Viel Erfolg!

## 1 Zahl zu Text (2 Punkte)

Schreiben Sie ein Unterprogramm `number2text` (Datei `number2text/number2text.s`), das eine vorzeichenlose, 32 Bit breite Zahl in ihre Textdarstellung zu einer gegebenen Basis umwandelt. Ihr Unterprogramm erhält folgende Argumente:

1. Die Adresse genau hinter dem Ende des Zielpuffers.
2. Die umzuwandelnde Zahl
3. Die Basis im Intervall $[2, 36]$, in die die Zahl konvertiert werden soll

Der Rückgabewert des Unterprogramms ist die Adresse des ersten Zeichens (höchstwertige Ziffer) der Zeichenkette. Die Zeichenkette ist mit NUL (Wert 0) abzuschließen. Sie können davon ausgehen, dass der Zielpuffer ausreichend groß ist. Verwenden Sie die Zeichen $0, \ldots, 9, A, \ldots, Z$ in ASCII-Kodierung zur Darstellung der Ziffern. Ein Beispiel:

- Endadresse `0x1000`, Zahl `42`, Basis `16`
- Pufferinhalt `50, 65, 0` (`"2A\0"`), Rückgabewert `0x0FFD`

## 2 Text zu Zahl (2 Punkte)

Schreiben Sie ein Unterprogramm `text2number` (Datei `text2number/text2number.s`), das die Textdarstellung einer vorzeichenlosen Zahl zu einer gegebenen Basis in die Zahl umwandelt. Ihr Unterprogramm erhält folgende Argumente:

1. Die Adresse des ersten Zeichens der Textdarstellung.
2. Die Basis im Intervall $[2, 36]$, in die die Zahl konvertiert werden soll

Der Rückgabewert des Unterprogramms ist die Zahl. Die Zeichenkette ist mit NUL (Wert 0) abgeschlossen. Sie können davon ausgehen, dass 32 Bit für die Speicherung der Zahl ausreichend sind. Die Textdarstellung verwendet die Zeichen $0, \ldots, 9, A, \ldots, Z$ in ASCII-Kodierung zur Darstellung der Ziffern. Ein Beispiel:

- Startadresse `0x1000`, Basis `16`
- Eingabe `50, 65, 0` (`"2A\0"`), Rückgabewert `42`

# 3 Destruktives Listenreversieren (3 Punkte)

Wir betrachten die Darstellung von einfach verketteten Listen im Speicher, wie Sie sie in der Vorlesung kennengelernt haben. Schreiben Sie ein Unterprogramm `listrev` (Datei `listrev/listrev.s`), das solche Listen reversiert. Das Unterprogramm erhält als erstes Argument die Adresse des ersten Elements der Liste. Ihr Unterprogramm soll die Eingabeliste modifizieren.

Jedes Listenelement besteht aus der Adresse des nächsten Elements gefolgt von Nutzdaten. Das letzte Element ist dadurch gekennzeichnet, dass die Nachfolgeradresse 0 ist. Das Unterprogramm soll die Adresse des neuen ersten (vormals letzten) Elements zurückgeben.

# 4 Text entpacken (3 Punkte)

In dieser Aufgabe geht es darum, eine bereits gepackte Zeichenkette zu entpacken und sie auszugeben. Unter einer gepackten Zeichenkette verstehen wir in dieser Aufgabe eine 6 Bit breite Kodierung pro Zeichen. Sie können davon ausgehen, dass die Zeichenkette immer mit der Bitfolge 000000 endet. Schreiben Sie ein Unterprogramm `decode` (Datei `decode/decode.s`) welches die beschriebene Kodierung dekodiert und ausgibt. Die einzelnen Werte werden wie folgt auf Zeichen abgebildet:

| Wert | Zeichen |
|---|---|
| 0 | *Ende einer Zeichenkette* |
| 1 | *Leerzeichen* |
| $[2-27]$ | A–Z |
| $[28-53]$ | a–z |
| $[54-63]$ | 0–9 |

Beachten Sie, dass es an den 32-Bit Grenzen (Ende eines Wortes) zu Überlappung kommen wird. Dies bedeutet, dass der erste Teil eines Zeichens in dem aktuellen Wort enthalten und der zweite Teil des selben Zeichens im nächsten Wort enthalten ist. Im folgenden Beispiel ist die Zeichenkette `Hallo Welt` in der gepackten Darstellung gezeigt:

| 6 | 6 | 6 | 6 | 6 | 2 |
|---|---|---|---|---|---|
| 001001 | 011100 | 100111 | 100111 | 101010 | 00 |
| 4 | 6 | 6 | 6 | 6 | 4 |
| 0001 | 011000 | 100000 | 100111 | 101111 | 0000 |
| 2 | 6 | 6 | 6 | 6 | 6 |
| 00 | 000000 | 000000 | 000000 | 000000 | 000000 |

# Public Tests selber ausführen

Sie können die public Tests für alle Teilaufgaben selber ausführen. Führen Sie dafür in Ihrem Projektordner `./run_test.py` aus. Falls Sie das Projekt nicht auf der VM bearbeiten, achten Sie darauf, dass MARS auf Ihrem System mit dem Befehl `mars` gestartet werden kann. Ansonsten bricht das Skript mit einem Fehler ab.

## Hinweise

- Überlegen Sie sich Randfälle für mögliche Eingaben, mit denen Ihre Programme auch zurechtkommen müssen.
- Überlegen Sie sich beim Entwurf von Schleifen zuerst den „Induktionsschritt“, bevor Sie den Anfangs-/Endfall bearbeiten.
- Nutzen Sie, dass die Ziffern und Buchstaben in ASCII-Kodierung aufeinander folgen.
- Beachten Sie die MIPS-Aufrufkonventionen.
- Sie müssen alle public-Tests bestehen, um Punkte für das Projekt zu erhalten.

## Lernziele

- Arbeiten mit Zahlen in verschiedenen Basen
- Schleifen
- Arbeiten mit Datenstrukturen im Speicher
- Aufruf von Unterprogrammen, Aufrufkonventionen