Technische Informatik
Dr. Henrik Brosenne
M. Sc. Maria Kosche

Wintersemester 2023/24
Georg-August-Universität Göttingen
Institut für Informatik

Übungsblatt 1

Maschinensprache

Hinweise zur Abgabe der Lösungen:

Die Abgabe von Lösungen zu allen Übungsblättern ist grundsätzlich freiwillig.

Die Bearbeitungen der Aufgaben können Sie freiwillig und in geeigneter Form in der Stud.IP-Veranstaltung über das **Vips-Modul** zum entsprechenden Aufgabenblatt hochladen. Sie erhalten dann entsprechend ein kurzes Feedback zu Ihrer Abgabe. Sie können Ihre Bearbeitungen gerne mit LaTeX formatieren, es ist aber auch der direkte Upload von Text oder der Upload von Text- und Bilddateien in gängigen Formaten möglich.

Die Aufgaben können auch in Kleingruppen bearbeitet und abgegeben werden. Wenn Sie nicht alleine abgeben, achten Sie bitte darauf, dass Sie sich selbstständig **vor** der Abgabe im Vips-Modul in einer sogenannten Übungsgruppe zusammenschließen. Hierzu können Sie sich in einer der dort vorhandenen Übungsgruppen gemeinsam eintragen. Nur so erhalten alle Zugang zu Feedback und Kommentaren zu der entsprechenden Abgabe.

Am Ende von Übungsblatt 1 finden Sie Hinweise zu den **Pseudocode-Konventionen** für die Übungen.

Aufgabe 1 – Arithmetik

Mit dieser Aufgabe sollen Sie sich zunächst mit der Entwicklungsumgebung vertraut machen, die in der Vorlesung vorgestellt wurde. Wir empfehlen die Verwendung von JSLinux unter folgender Adresse:

`https://bellard.org/jslinux/vm.html?cpu=riscv32&url=https://bellard.org/jslinux/buildroot-riscv32.cfg`

Um ein Home-Verzeichnis zu bekommen, können Sie sich wie im Emulator angegeben anonym und kostenlos einen Account anlegen. Sie können Dateien, die Sie zuvor auf ihrem Rechner gespeichert haben, direkt über die Website in Ihr Home-Verzeichnis hochladen und Dateien aus dem Emulator auf Ihren Rechner herunterladen.

Eine weitere Möglichkeit, Code zu simulieren und Schritt für Schritt auszuführen, ist mit dem Risc-V Simulator Ripes möglich. Das Projekt ist unter `https://github.com/mortbopet/Ripes` zu finden und es existieren bereits binaries für Linux, Windows und Mac zum Herunterladen auf der *Releases Page* des Projekts.

Schreiben Sie Assemblercode, der zu einem Dreieck, von dem zwei Winkel bekannt sind, den dritten Winkel berechnet. Orientieren Sie sich dabei an dem in der Vorlesung vorgestellten Assemblercode für die Berechnung des Umfangs eines Vierecks.

Aufgabe 2 – Ablaufsteuerung

Diese Aufgabe können Sie als reine Textaufgabe bearbeiten. Integrieren Sie die Assemblercode-Ausschnitte in ihre Textabgabe, Sie müssen nicht für jede Teilaufgabe eine einzelne Assemblercode-Datei hochladen.

1. (a) Wie realisiert man in Assembler eine bedingte Anweisung, d.h. einen Codeabschnitt, der nur durchlaufen wird, wenn eine bestimmte Bedingung erfüllt ist?

   (b) Geben Sie Assembler für folgenden Pseudocode an, der den Betrag von `a` berechnet.

```
if (a < 0)
  a = -a;
```

2. (a) Wie realisiert man in Assembler eine Auswahl von Alternativen, d.h. zwei Codeabschnitte, von denen jeweils einer durchlaufen wird, abhängig davon, ob eine bestimmte Bedingung erfüllt ist oder nicht?

   (b) Geben Sie Assembler für folgenden Pseudocode an, der den Abstand `dist` von `a` und `b` berechnet. Es gilt $\texttt{dist} = |\texttt{a} - \texttt{b}|$.

```
if (a > b)
  dist = a - b;
else
  dist = b - a;
```

3. (a) Wie realisiert man in Assembler eine Schleife, die solange Wiederholungen durchführt, wie eine bestimmte Bedingung erfüllt ist?

   (b) Geben Sie Assembler für folgenden Pseudocode an, der einen Näherungswert `k` für den Logarithmus von `x` zur Basis 2 liefert. Es gilt $2^{\texttt{k}-1} \leq \texttt{x} < 2^{\texttt{k}}$.

```
k = 0;
while (x > 0) {
  x = x/2;
  k = k + 1;
}
```

4. (a) Wie realisiert man in Assembler eine Schleife, die eine vorgegebene Anzahl an Wiederholungen durchführt?

   (b) Geben Sie Assembler für folgenden Pseudocode an, der die `k`-te Potenz `z` von `x` berechnet, es gilt $\texttt{z} = \texttt{x}^{\texttt{k}}$.

```
z = 1;
for (i = 0; i < k; i = i+1)
  z = z*x;
```

Aufgabe 3 – Daten durchlaufen

1. Betrachten Sie folgende Daten.

```
.data
list:   .word   2, -5, 3, -90, 300, -54, -23, -35, 120,
last:   .word   -54
```

   Schreiben Sie Assemblercode mit dem die Liste, deren erstes Element `list` und letztes Element `last` ist, durchlaufen wird und dabei die Anzahl der Elemente gezählt wird.

2. Betrachten Sie folgende Daten.

```
.data
list:   .word   2, -5, 3, -90, 300, -54, -23, -35, 120, -54
len:    .word   10
```

   (a) Schreiben Sie Assemblercode, mit dem die Liste `list` der Länge `len` durchlaufen wird und dabei die negativen Zahlen gezählt werden.

   (b) Schreiben Sie Assemblercode, mit dem die Liste `list` der Länge `len` durchlaufen wird und dabei das Minimum und Maximum der Liste bestimmt wird.

Aufgabe 4 – big-/little-endian

Big-Endian und Little-Endian sind Bezeichnungen für Ordnungen bezüglich der Byte-Reihenfolge, die für die Speicherung an Adressen verwendet wird. Bei Big-Endian wird das höchstwertige Byte zuerst gespeichert, also an der kleinsten Speicheradresse. Bei Little-Endian ist es genau andersherum und das kleinstwertige Byte kommt zuerst. Die Begriffe zeigen also jeweils an, welches Ende eines Wortes zuerst im Speicher abgelegt wird:

Big End → Höchstwertiges Byte zuerst,

Little End → Kleinstwertiges Byte zuerst.

1. Welche Möglichkeiten hat man, um mit einem RV32I-Programm zu ermitteln, welche Byte-Reihenfolge auf einem Risc-V-Prozessor verwendet wird?
2. Schreiben Sie ein RV32I-Programm, das ermittelt, welche Byte-Reihenfolge die zugrundeliegende Architektur verwendet. Das Programm gibt als Ergebnis „big-endian“ oder „little-endian“ aus.

Pseudocode

- Variablen können beliebige Namen haben. Mögliche Typen für Variablen sind `int32`/`int16`/`int8` und `pointer`, d.h. ganze Zahlen mit 32-/16-/8-Bit und 32-Bit-Adressen.

  Variablen werden durch Angabe des Typs gefolgt vom Namen der Variable angelegt.

- Die üblichen Operatoren Addition `+`, Subtraktion `-`, Multiplikation `*`, ganzzahlige Division `/` und Rest der ganzzahligen Division `%` stehen zur Verfügung.

  Mit Variablen, Konstanten und diesen Operatoren können arithmetische Ausdrücke gebildet werden.

- Für eine Adresse (z.B. `pointer p`) liefern `load32`/`load16`/`load8` den `int32`-/`int16`-/`int8`-Wert, der ab dieser Adresse gespeichert ist (z.B. `int32 v = load32(p)`).

- Weiterhin stehen die Vergleichsoperatoren für den Test auf Gleichheit `==`, Ungleichheit `!=`, Größer `>`, Kleiner `<`, Größer-gleich `>=` und Kleiner-gleich `<=` zur Verfügung.

  Mit Variablen, Konstanten und diesen Operatoren werden Bedingungen, die die Wahrheitswerte *wahr* und *falsch* annehmen können, gebildet.

- Durch den Zuweisungsoperator `=` kann einer Variablen ein Wert zugewiesen werden. Dies kann auch gleich beim Anlegen der Variable erfolgen.

- Anweisungen werden mit einem Semikolon `;` abgeschlossen.

- Mehrere Anweisungen werden durch das Einschließen in geschweifte Klammern `{}` zu einem Block, der wie eine Anweisung behandelt wird, zusammengefasst.

- Die Anweisung

```
if (condition)
  if_instruction
else
  else_instruction
```

  erwartet eine Bedingung `condition`. Ist der Vergleich *wahr*, wird die Anweisung (der Block) `if_instruction`, sonst wird die Anweisung (der Block) `else_instruction` ausgeführt.

- Die Anweisung

```
while (condition)
  instruction
```

  führt solange die Anweisung (den Block) `instruction` aus wie die Bedingung `condition` den Wert *wahr* hat.

- Die Anweisung

```
do
  instruction
while (condition);
```

führt solange die Anweisung (den Block) `instruction` aus wie die Bedingung `condition` den Wert *wahr* hat.

Der Unterscheid zu `while` ist, dass die Anweisung (der Block) `instruction` erst einmal ausgeführt wird bevor die Bedingung überprüft wird.

- Die Anweisung

```
for (expressionInit; condition; expressionLoop)
  instruction
```

führt zuerst den Ausdruck `expressionInit` aus (z.B. das Anlegen einer Variable). Anschließend wird solange die Anweisung (der Block) `instruction`, gefolgt vom Ausdrucks `expressionLoop` ausgeführt, wie die Bedingung `condition` den Wert *wahr* hat.