

Übungsblatt 10 (Block C – 2)

(16 Punkte)

**Abgabe bis spätestens Mittwoch, 10. Januar 2018, 16:00 Uhr.
 Besprechung ab Montag, 15. Januar 2018.**

Um die Aufgaben zu lösen, sollten Sie den in der Vorlesung vorgestellten MARS Simulator installieren und mit ihm arbeiten. Sie finden die Software unter:

<http://courses.missouristate.edu/KenVollmar/MARS/index.htm>

Installieren Sie den Simulator auf Ihrem Rechner.

10.1 Adressierungsarten (2 Punkte)

Gegeben sei eine abstrakte Maschine und folgender Auszug aus der Speicherbelegung einiger Register und Speicherzellen.

Register	Wert	Speicherzelle	Wert
Register 1	12	Speicherzelle 12	7
Register 2	36	Speicherzelle 24	95
Register 3	-12	Speicherzelle 36	24
Register 4	67	Speicherzelle 48	36
		Speicherzelle 68	42
		Speicherzelle 96	-5

Geben Sie jeweils an, welcher Wert durch folgende Adressierungsbefehle geladen wird. Die Adressierungsart ist jeweils vorgegeben und die Parameter (Konstanten / Adressen bzw. Register) sind in Klammern angegeben.

Geben Sie den dazugehörigen MIPS-Befehl an. Das Ergebnis soll jeweils im Register \$t0 stehen. Die Register 1 bis 4 und die Speicherzellen 12 bis 96 seien schon entsprechend vorbelegt.

- a. Unmittelbare Adressierung (36)
- b. Direkte Adressierung (24)
- c. Registeradressierung (Register 4)
- d. Register-indirekte Adressierung (Register 1)

10.2 Stackprogrammierung (6 Punkte)

In dieser Aufgabe sollen Sie die Realisierung von Hochsprachen-Funktionen in Assembler betrachten und dabei den Stackaufbau und die Parameterübergabe üben¹. Damit Funktionen unabhängig voneinander übersetzt werden können, gibt es Standards zur Übergabe von Parametern und zum Umgang mit Registern und dem Stack.

Nutzen Sie für diese Aufgabe die MIPS-Aufrufkonvention. Die Register sind nach dieser Konvention wie folgt bezeichnet:

Register-Nummer	Register-Name	Funktion	Sicherung
\$0	\$zero	Ist immer 0	Undefiniert
\$1	\$at	Reserviert für Assembler	Undefiniert
\$2 – \$3	\$v0 – \$v1	Rückgabewerte von Funktionen	Undefiniert
\$4 – \$7	\$a0 – \$a3	Die ersten 4 Parameter für Funktionen	Undefiniert
\$8 – \$15, \$24 – \$25	\$t0 – \$t9	Temporaries (Lokale Puffer-Register)	Caller-Saved
\$16 – \$23	\$s0 – \$s7	Saved (Persistente Puffer-Register)	Callee-Saved
\$26 – \$27	\$k0 – \$k1	Kernel, reserviert für Betriebssystem	Undefiniert
\$28	\$gp	Global Pointer (Adresse des .data-Bereichs)	Undefiniert
\$29	\$sp	Stack Pointer (Adresse des obersten Stack-Elements)	Undefiniert
\$30	\$fp	Frame Pointer (Basis des aktuellen Stackframes)	Undefiniert
\$31	\$ra	Return Address (Rücksprungadresse)	Undefiniert

Tabelle 1: MIPS-Registerkonvention

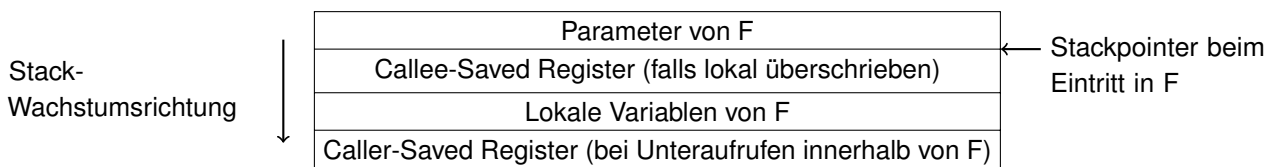
Für diese Aufgabe benötigen Sie nur die Register aus den grau markierten Zeilen der Register-Konvention.

Die Register \$t0 – \$t9 sind *Caller-Saved*, werden also vom Aufrufer gesichert und können innerhalb der Funktion ohne Rücksicht verwendet werden. Die Register \$s0 – \$s7 sind *Callee-Saved*, sie müssen deshalb von der Funktion auf dem Stack gesichert und zum Ende der Funktion wieder auf ihren ursprünglichen Wert gesetzt werden, sofern sie benutzt werden. Alle anderen Register sind für besondere Verwendungen reserviert (siehe Spalte „Funktion“).

Rückgabewerte von Funktionen werden über \$v0 – \$v1 an den Aufrufer zurückgegeben. Die ersten 4 Parameter einer Funktion werden in den Registern \$a0 – \$a3 übergeben, alle weiteren Parameter werden über den Stack übergeben. Es ist also nötig, dass eine Funktion weiß, wie viele Parameter sie erwartet, damit sie weiß, wie viele Argumente noch auf dem Stack liegen.

Das Register \$sp zeigt immer auf das oberste Element des Stacks. Immer wenn etwas auf den Stack gelegt oder davon genommen wird, muss dieses Register explizit angepasst werden (+/- Byte-Größe der entfernten/hinzugelegten Elemente). Der Stack wächst in Richtung der absteigenden Adressen, d. h. wenn Sie Elemente hinzufügen, müssen Sie den Stackpointer dekrementieren.

Der Stack-Abschnitt (Stack-Frame) einer Funktion *F* folgt diesem Aufbau (Adressen von oben nach unten absteigend):



Nach Abschluss einer Funktion *muss* das Stackframe komplett abgebaut (vom Stack genommen worden) sein, inklusive der eigenen Parameter.

¹siehe auch Skript, S. 19 – 25: <http://ls12-www.cs.tu-dortmund.de/daes/media/documents/teaching/courses/ws1314/rs/script/rs2.pdf>

- b. Vervollständigen Sie das Programm, so dass die Funktionsaufrufe von „f1“ und „f2“ korrekt gemäß der oben beschriebenen Aufrufkonvention abgewickelt wird. Dabei können Sie über den Inhalt bzw. das Verhalten der Funktionen keine Annahme machen, außer, dass sie ebenfalls der MIPS-Aufrufkonvention genügt. Weiterhin wissen Sie, dass „f1“ fünf 4-Byte-Integer als Parameter erwartet und einen 4-Byte-Integer zurück gibt und dass „f2“ einen Parameter erwartet und einen 4-Byte-Integer zurück gibt.

```

.text
.globl main
main:

-----

-----

-----

lw $t0, zahl1
lw $t1, zahl2
lw $t2, zahl3
lw $t3, zahl4

-----

-----

-----

-----

-----

-----

-----

-----

-----

jal f1

-----

-----

-----

-----

li $v0, 10
syscall

```

- c. Testen Sie Ihr Programm mit dem MIPS-Simulator.

10.3 Assemblerprogrammierung (4 Punkte)

Mit dem folgenden Assemblerprogramm soll die Quersumme QS einer Dezimalzahl (wert) berechnet und unter „quer“ im Speicher abgelegt werden. Ergänzen Sie die fehlenden Operationen gemäß den Kommentaren.

```

.data
wert: .word 1495           # Eingabewert.
quer: .word 0             # Ergebnis der Quersumme QS.
.text
.globl main
main:
    _____           # lade den Eingabewert in Register $2.
    _____           # Reg [4] für die QS initialisieren.
loop: _____         # wenn Reg[2] = 0 -> ende.
    _____         # in Reg[3] steht eine 10 (dezimal).
    _____         # wert / 10 und den Teil hinter dem
    _____         # Komma in Reg [3] laden
    _____         # und in Reg [4] aufaddieren.
    _____         # Restzahl vor dem Komma in Reg [2] laden.
    _____         # bei loop weiterrechnen.
ende: _____         # Ergebnis in quer ablegen.

    li $2,10           # Programmende.
    syscall

```

10.4 Belegungsverfahren (4 Punkte)

- Die Zeichenkette „Pause“ sei in 8-Bit-ASCII in einem Speicher ab der Adresse $0xE8E8$ abgelegt. Die Zeichen wurden dabei wortweise, d. h. in 32-Bit-Blöcken, mit dem Belegungsverfahren „little endian“ vom Prozessor in den Speicher geschrieben. Welches Zeichen enthält das Byte mit der Adresse $0xE8EB$?
- Wie wird auf das Speicherbelegungsverfahren „big endian“ hardwaremäßig umgeschaltet?

Hinweise:

Die Abgaben sollen bis Mittwoch, 10. Januar 2018, 16:00 Uhr in die Briefkästen in der Otto-Hahn-Straße 12 eingeworfen werden.

Die Briefkästen finden Sie in der ersten Etage der Otto-Hahn-Straße 12 am Übergang zum Erdgeschoss der Otto-Hahn-Straße 14. Die Briefkästen sind mit dem Namen der Veranstaltung, der Gruppennummer sowie der Zeit der Übung gekennzeichnet. Für Rechnerstrukturen sind dies die Briefkästen mit den Nummern 20 bis 32.

Schreiben Sie unbedingt Ihren **Namen**, Ihre **Matrikelnummer** und Ihre **Gruppennummer** rechts oben auf Ihre Abgabe. Sie dürfen als Team mit bis zu zwei weiteren Personen abgeben. Geben Sie dann nur eine einzige Lösung ab und schreiben Sie alle Namen und Matrikelnummern des Teams auf die gemeinsame Abgabe.

Heften Sie die Abgabe bitte zusammen (Tacker oder notfalls Büroklammer). Bitte die Abgabe **nicht falten** und **keine Schnellhefter oder Umschläge** abgeben.

Es gibt insgesamt 12 Übungsblätter, die in 3 Blöcke (A, B, C) aufgeteilt sind. In jedem Block müssen Sie 30 Punkte von 64 möglichen Punkten erreichen, um zur Prüfung zugelassen zu werden.

HelpDesk Rechnerstrukturen:

Neben den Übungen bieten wir dieses Jahr auch einen speziellen RS Help Desk an. Der Help Desk kann euch bei der Bearbeitung der Übungsaufgaben, der Klausurvorbereitung oder sonstigen vorlesungsrelevanten Problemen helfen. Weitere Information finden Sie auf der Webseite zur Vorlesung.