georg.von-der-brueggen [☺] tu-dortmund.de
ulrich.gabor [☺] tu-dortmund.de
pascal.libuschewski [☺] tu-dortmund.de
olaf.neugebauer [☺] tu-dortmund.de

Übung zur Vorlesung
Rechnerstrukturen
Wintersemester 2015

Übungsblatt 10 (Block C - 2)

(16 Punkte)

Abgabe bis spätestens Mittwoch, 13. Januar 2016, 16:00 Uhr
Besprechung ab Montag, 18. Januar 2016

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 (4 Punkte)

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

Register	Wert
Register 1	12
Register 2	36
Register 3	-12
Register 4	68

Speicherzelle	Wert
Speicherzelle 12	6
Speicherzelle 24	96
Speicherzelle 36	24
Speicherzelle 48	36
Speicherzelle 68	42
Speicherzelle 96	-5

Geben Sie jeweils an, welcher Wert jeweils 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 (4 Punkte)

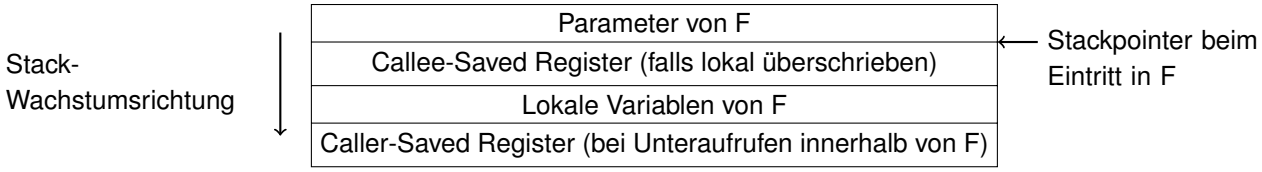
In dieser Aufgabe sollen Sie die Implementierung von Hochsprachen-Funktionen in Assembler betrachten und dabei den Stackaufbau und die Parameterübergabe unter Verwendung von Caller-Saved- und Callee-Saved-Registern üben. Nutzen Sie für diese Aufgabe die im folgenden gegebenen MIPS-Aufrufkonvention. Die 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	Erste 4 Parameter für Funktionen	Undefiniert
$8-$15, $24-$25	$t0-$t9	Temporaries (Lokale Puffer-Register)	Caller-Saved
$16-$23	$s0-$s9	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-Registerkovention

Für diese Aufgabe benötigen Sie nur die Register aus den grau markierten Zeilen der Register-Konvention. Die Register $s0-$s7 sind *Callee-Saved* wohingegen $t0-$t9 *Caller-Saved* sind. Alle anderen Register sind für besondere Verwendungen reserviert (siehe Feld „Funktion“). Rückgabewerte von Funktionen werden über $v0-$v1 an den Aufrufer zurückgegeben. Die ersten 4 Parameter einer Funktion (hier von der später definierten Funktion f1) werden in den Registern $a0-$a3 übergeben, alle weiteren Parameter werden über den Stack übergeben. Daher ist es 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 hinzugelegten/entfernten 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. Das vorgegebene Programm (s.u.) soll

$$ergebnis = zahl1 \cdot f1(zahl1, zahl2, zahl3, zahl4, f2(zahl5))$$

berechnen und das Ergebnis in „ergebnis“ speichern. Benutzen Sie, um das Programm lesbarer zu machen, in dieser Aufgabe nur die Register-Namen in Ihrem Code (s.o.), nicht die Register-Nummern. Ändern Sie nichts am vorgegebenen Code.

Der Code für f2 ist vorgegeben:

```
f2:
    li $t0, 5             # Lade Konstante 5.
    add $v0, $a0, $t0     # Addiere Konstante auf Parameter.
                          # Speicher in Rückgaberegister.
    jr $ra                # Rücksprung an Adresse $ra.
```

a. Schreiben Sie nun die Funktion „f1“. „f1“ soll die Summe seiner Eingabewerte berechnen und zurückgeben.

```
# Beachten Sie, dass ein Parameter im Stack übergeben wurde.
# Sie dürfen nur die $t-Register verwenden und das Rückgaberegister $v0.

f1:
        _______________
        _______________
        _______________
        _______________
        _______________
        _______________
        _______________
        jr $ra
```

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 Operanden 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 initalisieren.

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)

a. Die Zeichenkette „Pause“ sei in 8-Bit-ASCII in einem Speicher ab der Adresse F8F8 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 F8FB?

b. Wie wird auf das Speicherbelegungsverfahren „big endian“ hardwaremäßig umgeschaltet?

Hinweise:
Die Abgaben sollen bis Mittwoch, 13. Januar 2016, 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 Zeit und Ort der Übung gekennzeichnet.

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 zusammen. (Tacker oder notfalls Büroklammer). Falten Sie aber nicht ihre Abgabe. Stecken Sie die Abgabe nicht in einen Umschlag. Benutzen Sie den richtigen Briefkasten. Dazu benötigen Sie ihre Gruppennummer.

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.