

Rechnerstrukturen im WS 2013/2014 Übungsblatt 10 (Block C-2)

Aufgabe 1 (Fehlersuche) (4 Punkte)

Sie haben ein Programm zur iterativen Berechnung der Fakultät erhalten. Leider haben sich 6 Fehler eingeschlichen (syntaktische und semantische Fehler), die Sie finden und beseitigen sollen. Die Anzahl der Programmzeilen soll dabei erhalten bleiben.

Hinweis: $n! = n * (n-1) * (n-2) * \dots * 1$ und $0! = 1$.

a) Geben Sie die Fehler und das korrigierte Programm an.

b) Das Ergebnis in Reg 3 soll in einem anderen Programmteil als Zweierkomplementzahl interpretiert werden. Wie groß (dezimal) darf die Eingabe „ein“ sein, damit Reg 3 als richtiges Ergebnis verwendet werden kann?

```
.data
ein:      .word 5   #Eingabewert vom User (z.B. 5!=120=78H)
erg:      .word 1   #Ergebnis für den User (am Programmende)

.text
.global main
main:     lw $2,ein          #Eingabe „holen“
          li $3,0           #vorbelegen, in $3 könnte ja sonstwas stehen
          bneq $2,$0,fertig #0! gibt keine Schleife
jump:     mul $3,$3,$2      #mul erg mit zähler
          subi $2,1         #zählen
          bgt $2,1, jump    #Schleifenende, mul mit 1 muss nicht sein
fertig:   sw erg, $3        #Fakultät nach Berechnung in erg
          li $2,$10        #Programmende
          syscall
```

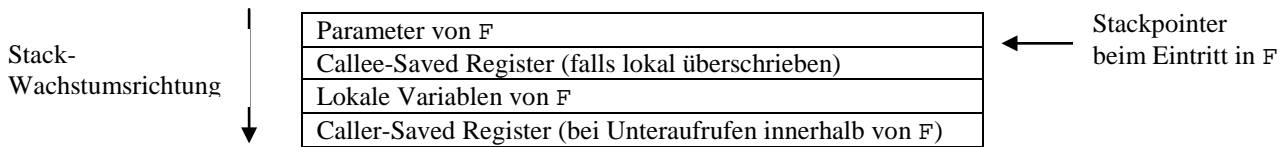
Aufgabe 2 (Stackprogrammierung) (4 Punkte)

In dieser Aufgabe sollen Sie die Implementierung von Hochsprachen-Funktionen in Assembler betrachten und dabei 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	Sicherungsmodus
\$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-\$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

Für diese Aufgabe benötigen Sie nur die grau markierten Zeilen der Register-Konvention. Die Register \$s0-\$s7 sind Callee-Saved, \$t0-\$t9 sind Caller-Saved. 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 func) werden in den Registern \$a0-\$a3 übergeben, alle weiteren Parameter werden über den Stack übergeben. Daher ist es nötig, dass eine Funktion weiß, wieviele Parameter sie erwartet, damit sie weiß, wieviele 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 *mus*s das Stackframe komplett abgebaut (vom Stack genommen worden sein), inklusive der eigenen Parameter. Das vorgegebene Programm (s.u.) soll

ergebnis = zahl1 * f1(zahl1, zahl2, zahl3, zahl4, f2())

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.

- a) 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 zurückgibt und dass „f2“ keine Parameter erwartet und einen 4-Byte-Integer zurückgibt.

```
.data
zahl1: .word 2
zahl2: .word 2
zahl3: .word 3
zahl4: .word 4
ergebnis: .space 4 # Korrekter Ergebniswert: 32(dez)

.text
.globl main
main:
    .
    .

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

    .
    .
    .
    .
    .
    .
    .

    jal f1

    .
    .

    li $v0, 10
    syscall
```

- b) 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

```

c) Testen Sie Ihr Programm mit der unten gegebenen Implementierung von „f2“.

```

f2:
    li $v0, 5
    jr $ra

```

Aufgabe 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 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

```

Aufgabe 4 (Belegungsverfahren) (4 Punkte)

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

Die Abgaben sollen bis Mittwoch den 17. Dezember 2014 um 18.00 Uhr in die Briefkästen in der Otto-Hahn-Strasse 12 eingeworfen werden. Bitte Name (bei einem 3er-Team alle), Matrikel- und Gruppennummer oben auf der ersten Seite der Lösungen angeben.