Tutoriumsblatt 7
Rechnerarchitektur im SoSe 2020

Zu den Modulen K

Aufgabe T21: Test des MIPS Simulators

(– Pkt.)

Für diese Aufgabe sollten Sie sich mit dem MIPS-Simulator SPIM vertraut machen. Sie können einen MIPS-Simulator von der Vorlesungshomepage herunterladen.

- Laden Sie sich das Assemblerprogramm `simple.s` von der Rechnerarchitektur–Homepage herunter und speichern Sie es in Ihrem Home–Verzeichnis ab.
- Starten Sie Ihren Simulator.
- Laden Sie das Programm `simple.s` in den Simulator und führen Sie es aus. Dabei sollte eine Konsole erscheinen, über die die Ein- und Ausgabe erfolgt.

Beantworten Sie nun folgende Fragen.

a. Welches Ergebnis liefert das Programm für die Eingabefolge "6, 7, 8, 9, 0"(d.h. nach Start des Programms erfolgt über die Konsole die Eingabe "6", gefolgt von *Enter*, dann die Eingabe "7", gefolgt von *Enter*, usw.)?

b. Die folgenden Kommentare beschreiben Teile des Programms `simple.s`. Ordnen Sie den Kommentaren jeweils die minimale Anzahl an Codezeilen zu, die benötigt werden, um das beschriebene Verhalten im Code darzustellen, und geben Sie die Zeilennummer(n) dieser Zeile(n) an!

   i) `str1` wird auf der Konsole ausgegeben.

   ii) Es wird eine Zahl von der Konsole eingelesen.

   iii) Das Programm wird beendet.

   iv) Eine Zählvariable wird um den Wert 1 erhöht.

c. In welchem Wertebereich müssen sich die eingegebenen Zahlen befinden, damit keine Fehlerbehandlung stattfindet (= damit das Label *error* nicht angesprungen wird).

d. Welche mathematische Funktion berechnet das Programm?

```
        .data
str1:   .ascii "Geben Sie beliebig viele Zahlen ein.\n"
        .asciiz "Eingabe von 0 beendet die Eingabe und gibt das Ergebnis aus.\n"
askstr: .asciiz "\n?-> "
errstr: .asciiz "Die Eingegebene Zahl ist ungültig. Bitte probieren Sie es erneut\n"
answstr:.asciiz "Das Ergebnis lautet: "
str2:   .asciiz "\n\n"

        .text
main:   li      $s0, 0
        li      $s1, 0
        li      $s2, 3

        li      $v0, 4
        la      $a0, str1
        syscall

loop:   li      $v0, 4
        la      $a0, askstr
        syscall

        li      $v0, 5
        syscall
        beqz    $v0, exit
        li      $t2, 103
        bgt     $v0, $t2, error
        li      $t2, 5
        blt     $v0, $t2, error
        addi    $s1, $s1, 1
        rem     $t2, $v0, $s2
        mul     $t2, $t2, $s1
        add     $s0, $s0, $t2

        j       loop

error:  li      $v0, 4
        la      $a0, errstr
        syscall
        j       loop

exit:   li      $v0, 4
        la      $a0, answstr
        syscall

        li      $v0, 1
        move    $a0, $s0
        syscall

        li      $v0, 4
        la      $a0, str2
        syscall

        li      $v0, 10
        syscall

```

Aufgabe T22: Umsetzung Boolescher Ausdrücke

(– Pkt.)

Übersetzen Sie folgendes Pseudocodefragment in MIPS-Code. Gehen Sie davon aus, dass der Wert der Variablen `a` bereits in das Register `$t0` geladen wurde.

```
1 IF (a < 0) OR (a > 99) THEN
2     a := a - 10;
3 ELSE
4     a := a - 1;
5 END;
```

Bedenken Sie dabei insbesondere: Der Ausdruck `a > 99` wird nur dann ausgewertet, wenn `a < 0` fehlgeschlagen ist.

Aufgabe T23: SPIM Programmieraufgabe

(– Pkt.)

Erstellen Sie ein **vollständiges** SPIM–Programm, das folgendes durchführt:

- Es werden zwei positive Integer–Zahlen von der Konsole eingelesen.
- Es wird der Durchschnitt dieser beiden Zahlen auf eine Nachkommastelle genau berechnet.
- Das Ergebnis der Berechnung wird ausgegeben.

Tipp: Programmieren Sie diejenigen Schritte, die Sie auch beim handschriftlichen Dividieren durchführen!

Beachten Sie hierbei folgendes:

- Verwenden Sie nur die **unten aufgeführten Befehle**.
- Verwenden Sie für die Vorkommazahl das Register `$s0` und für die Nachkommazahl das Register `$s1`, ansonsten nur die temporären Register.
- **Kommentieren** Sie ihr Programm sinnvoll!
- Sowohl die Eingabe als auch die Ausgabe soll mit einem Anweisungstext versehen werden, wie z.B. *"Geben Sie die 1. Zahl ein: "*, etc.

Überblick über die wichtigsten SPIM Assemblerbefehle		
Befehl	Argumente	Wirkung
`add`	`Rd, Rs1, Rs2`	Rd := Rs1 + Rs2 (mit Überlauf)
`sub`	`Rd, Rs1, Rs2`	Rd := Rs1 - Rs2 (mit Überlauf)
`addu`	`Rd, Rs1, Rs2`	Rd := Rs1 + Rs2 (ohne Überlauf)
`subu`	`Rd, Rs1, Rs2`	Rd := Rs1 - Rs2 (ohne Überlauf)
`addi`	`Rd, Rs1, Imm`	Rd := Rs1 + Imm
`addiu`	`Rd, Rs1, Imm`	Rd := Rs1 + Imm (ohne Überlauf)
`div`	`Rd, Rs1, Rs2`	Rd := Rs1 DIV Rs2
`rem`	`Rd, Rs1, Rs2`	Rd := Rs1 MOD Rs2
`mul`	`Rd, Rs1, Rs2`	Rd := Rs1 × Rs2
`b`	`label`	unbedingter Sprung nach `label`
`j`	`label`	unbedingter Sprung nach `label`
`jal`	`label`	unbed.Sprung nach `label`, Adresse des nächsten Befehls in $ra
`jr`	`Rs`	unbedingter Sprung an die Adresse in `Rs`
`beq`	`Rs1, Rs2, label`	Sprung, falls `Rs1` = `Rs2`
`beqz`	`Rs, label`	Sprung, falls `Rs` = `0`
`bne`	`Rs1, Rs2, label`	Sprung, falls `Rs1` ≠ `Rs2`
`bnez`	`Rs1, label`	Sprung, falls `Rs1` ≠ `0`
`bge`	`Rs1, Rs2, label`	Sprung, falls `Rs1` ≥ `Rs2`
`bgeu`	`Rs1, Rs2, label`	Sprung, falls `Rs1` ≥ `Rs2`
`bgez`	`Rs, label`	Sprung, falls `Rs` ≥ `0`
`bgt`	`Rs1, Rs2, label`	Sprung, falls `Rs1` > `Rs2`
`bgtu`	`Rs1, Rs2, label`	Sprung, falls `Rs1` > `Rs2`
`bgtz`	`Rs, label`	Sprung, falls `Rs` > `0`
`ble`	`Rs1, Rs2, label`	Sprung, falls `Rs1` ≤ `Rs2`
`bleu`	`Rs1, Rs2, label`	Sprung, falls `Rs1` ≤ `Rs2`
`blez`	`Rs, label`	Sprung, falls `Rs` ≤ `0`
`blt`	`Rs1, Rs2, label`	Sprung, falls `Rs1` < `Rs2`
`bltu`	`Rs1, Rs2, label`	Sprung, falls `Rs1` < `Rs2`
`bltz`	`Rs, label`	Sprung, falls `Rs` < `0`
`not`	`Rd, Rs1`	Rd := ¬ Rs1 (bitweise Negation)
`and`	`Rd, Rs1, Rs2`	Rd := Rs1 & Rs2 (bitweises UND)
`or`	`Rd, Rs1, Rs2`	Rd := Rs1 | Rs2 (bitweises ODER)
`syscall`		führt Systemfunktion aus
`move`	`Rd, Rs`	Rd := Rs
`la`	`Rd, label`	Adresse des Labels wird in Rd geladen
`lb`	`Rd, Adr`	Rd := MEM[Adr]
`lw`	`Rd, Adr`	Rd := MEM[Adr]
`li`	`Rd, Imm`	Rd := Imm
`sw`	`Rs, Adr`	MEM[Adr] := Rs (Speichere ein Wort)
`sh`	`Rs, Adr`	MEM[Adr] MOD 2^{16} := Rs (Speichere ein Halbwort)
`sb`	`Rs, Adr`	MEM[Adr] MOD 256 := Rs (Speichere ein Byte)

Funktion	Code in `$v0`	Funktion	Code in `$v0`
`print_int`	1	`read_float`	6
`print_float`	2	`read_double`	7
`print_double`	3	`read_string`	8
`print_string`	4	`sbrk`	9
`read_int`	5	`exit`	10