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.)? 8

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. 14, 15, 16
   ii) Es wird eine Zahl von der Konsole eingelesen. 22, 23
   iii) Das Programm wird beendet. 54
   iv) Eine Zählvariable wird um den Wert 1 erhöht. 29

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?

(c) Error when bgt or blt $\Rightarrow$ $(\$v0=input > 103)$ or $(\$v0=input < 5)$

$\Rightarrow$ $5 \leq \$v0 \leq 103$

(d) $1 \cdot (\$v0 \% 3) + 2(\$v0 \% 3) + 3(\$v0 \% 3) + \ldots$

$\Rightarrow \sum_{i=1}^{\uparrow} i\,(x_i \bmod 3)$, wobei x_i die i-te Eingabe

```
1           .data
2   str1:   .ascii "Geben Sie beliebig viele Zahlen ein.\n"
3           .asciiz "Eingabe von 0 beendet die Eingabe und gibt das Ergebnis aus.\n"
4   askstr: .asciiz "\n?-> "
5   errstr: .asciiz "Die Eingegebene Zahl ist ungültig. Bitte probieren Sie es erneut\n"
6   answstr:.asciiz "Das Ergebnis lautet: "
7   str2:   .asciiz "\n\n"
8
9           .text
10  main:   li      $s0, 0
11          li      $s1, 0
12          li      $s2, 3
13
14          li      $v0, 4
15          la      $a0, str1
16          syscall
17
18  loop:   li      $v0, 4
19          la      $a0, askstr
20          syscall
21
22          li      $v0, 5
23          syscall
24          beqz    $v0, exit
25          li      $t2, 103
26          bgt     $v0, $t2, error
27          li      $t2, 5
28          blt     $v0, $t2, error
29  $s1++ <-addi    $s1, $s1, 1
30          rem     $t2, $v0, $s2
31          mul     $t2, $t2, $s1
32          add     $s0, $s0, $t2
33
34          jump    loop
35
36  error:  li      $v0, 4
37          la      $a0, errstr
38          syscall
39          j       loop
40
41  exit:   li      $v0, 4
42          la      $a0, answstr
43          syscall
44
45          li      $v0, 1
46          move    $a0, $s0
47          syscall
48
49          li      $v0, 4
50          la      $a0, str2
51          syscall
52
53          li      $v0, 10
54          syscall
55
```

Main Programm

load 0 into $s0

" into $s1

SysCall

load str1 into args $a0

Print str1

5 = read integer

Read int into $v0

Branch equal zero ($v0 = 0 then spring)

Branch greater than ($v0 > $t2) → 103 ⇒ Error

Branch less than ($v0 < $t2) → 3 ⇒ Error

dest

3

Modulo ⇒ $t2 = $v0 % $s2

Multiply ⇒ $t2 = ($v0 % 3) * $s1 → Zahlwar

$s0 += $t2

Print error

offer another chance to enter

Print integer

Ergebnis

Print $s0

10 ⇒ Exit Programm

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.

```
# a bereits in $t0 geladen

# if ( a < 0 || a > 90 )
        bltz    $t0, true
        li      $t1, 99
        bgt     $t0, $t1, true
# else-case: a --
        sub     $t0, $t0, 1
        b       exit

# if-case  : a - 10
true:   sub     $t0, $t0, 10

exit:  # To be continued, since pseudocode no exit
```

→ can just use like that

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.

Vorüberlegungen:

A=23, B=56

23+56 = 79

79 ÷ 2 = 39

$$\begin{array}{r|l} & 39{,}5 \\ 2 & 79 \\ \hline & 6 \\ \hline & 19 \\ & 18 \\ \hline & 10 \\ & 10 \\ \hline & 0 \end{array}$$

```
        .data
line1:  .asciiz "Geben Sie die 1. Zahl ein: "
line2:  .asciiz "Geben Sie die 2. Zahl ein: "
erg:    .asciiz "Das Ergebnis lautet: "
komma:  .asciiz ","

        # Registerbelegungen
        # $t0 = A
        # $t1 = B
        # $t2 = Summe (von A und B)
        # $t3 = 2
        # $t4 = Rest der Division
        # $t5 = 10
        # $s0 = Vorkommazahl
        # $s1 = Nachkommazahl

        .text
main:
    1)  ################################################
        # Einlesen der 2 Werte
        ################################################

        li      $v0, 4          # 4: print_str:
        la      $a0, line1      # Adresse der 1. auszugebende Zeile nach $a0
        syscall                 # ausgeben

        li      $v0, 5          # 5: read_int
        syscall                 # integer einlesen
        move    $t0, $v0        # eingelesenen Wert in Register $t0 verschieben (A)

        li      $v0, 4          # 4: print_str:
        la      $a0, line2      # Adresse der 2. auszugebende Zeile nach $a0
        syscall                 # ausgeben

        li      $v0, 5          # 5: read_int
        syscall                 # integer einlesen
        move    $t1, $v0        # eingelesenen Wert in Register $t1 verschieben (B)

        ################################################
    2)  # Berechnung des Durchschnitts
        ################################################

        add     $t2, $t0, $t1   # t2/Summe = $t0/A + $t1/B
        li      $t3, 2          # Den Wert 2 in Register $t0 laden
        div     $s0, $t2, $t3   # $s0/Vorkomma  = $t2/Summe DIV $t3/2  (whole number divide)
        rem     $t4, $t2, $t3   # $t4/Rest = $t2/Summe MOD $t3/2

      { li      $t5, 10         # Den Wert 10 in Register $t5 laden
        mul     $t6, $t4, $t5   # Eine Null an den Rest haegen (= x/10) }

        div     $s1, $t6, $t3   # den mit 0 erweiterten Rest wieder durch 2 teilen = NKS

        ################################################
    3)  # Ausgabe des Ergebnisses
        ################################################

        li      $v0, 4          # 4: print_str
```

79 ÷ 2 = 39,5

$s0

$t4

39,5
2|79
6
19
18
10
10
0

```
59          la      $a0, erg        # Adresse des Ergebnis-Texts nach $a0
60          syscall                 # ausgeben
61
62          li      $v0, 1          # 1: print_int
63          move    $a0, $s0        # Vorkommastelle nach £a0
64          syscall
65
66          li      $v0, 4          # 4: print_str
67          la      $a0, komma      # Adresse des Komma-Texts nach $a0
68          syscall
69
70          li      $v0, 1          # 1: print_int
71          move    $a0, $s1        # Nachkommastelle nach $a0
72          syscall
73
74          li      $v0, 10         # Systemaufrufnr. 10 = EXIT
75          syscall
```

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