

# 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.

- 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
- 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!
  - `str1` wird auf der Konsole ausgegeben. 14, 15, 16
  - Es wird eine Zahl von der Konsole eingelesen. 22, 23
  - Das Programm wird beendet. 54
  - Eine Zählvariable wird um den Wert 1 erhöht. 29
- In welchem Wertebereich müssen sich die eingegebenen Zahlen befinden, damit keine Fehlerbehandlung stattfindet (= damit das Label *error* nicht angesprungen wird).
- Welche mathematische Funktion berechnet das Programm?

(c) Error when bgt or blt  $\Rightarrow$   $(\$10 = \text{input} > 103) \text{ or } (\$10 = \text{input} < 5)$   
 $\Rightarrow 5 \leq \$10 \leq 103$

(d)  $1 \cdot (\$10 \% 3) + 2 \cdot (\$10 \% 3) + 3 \cdot (\$10 \% 3) + \dots$   
 $\Rightarrow \sum_{i=1}^n i \cdot (x_i \bmod 3)$ , 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 Main Programm
10 main: li $s0, 0 load 0 into $0
11      li $s1, 0 " info $1
12      li $s2, 3
13
14      (syscall Print str1
15      li $v0, 4
16      la $a0, str1 ← load str1 into arg $a0)
17      syscall
18
19 loop: li $v0, 4
20      la $a0, askstr
21      syscall
22
23      (5 = read integer Read int into $v0
24      li $v0, 5)
25      beqz $v0, exit → Branch equal zero ($v0 = 0 then spring)
26      bgt $v0, $t2, error → Branch greater than ($v0 > $t2) → 103 ⇒ Error
27      blt $v0, $t2, error → Branch less than ($v0 < $t2) → 3 ⇒ Error
28      $s1++ ← dest addi $s1, $s1, 1 3
29      rem $t2, $v0, $s2 → Modulo ⇒ $t2 = $v0 % $s2
30      mul $t2, $t2, $s1 → Multiply ⇒ $t2 = ($v0 % 3) * $s1
31      add $s0, $s0, $t2 → $s0 += $t2 → Zahlur
32
33      jump loop
34
35 error: (Print error
36      li $v0, 4
37      la $a0, errstr)
38      syscall
39      j loop → offer another chance to enter
40
41 exit: li $v0, 4
42      la $a0, ansWSTR
43      syscall
44
45      (Print Integer Ergebnis Print $s0
46      li $v0, 1
47      move $a0, $s0)
48      syscall
49
50      li $v0, 4
51      la $a0, str2
52      syscall
53
54      li $v0, 10 ⇒ Exit Programm
55      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.

```
# a bereits in $t0 geladen
```

```
# if ( a < 0 || a > 99 )
```

```
    bltz    $t0, true
```

```
    li      $t1, 99
```

```
    bgt     $t0, $t1, true
```

```
# else-case: a --
```

```
    sub     $t0, $t0, 1
```

```
    b       exit
```

*→ can just use like that*

```
# if-case : a - 10
```

```
true:  sub     $t0, $t0, 10
```

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

## 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.

Vorüberlegen:

$A=23, B=56$

$23+56=79$

$79 \div 2 = 39,5$

$$\begin{array}{r} 39,5 \\ 2 \overline{) 79} \\ \underline{6} \phantom{0} \\ 19 \\ \underline{18} \\ 10 \\ \underline{10} \\ 0 \end{array}$$

### 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	$Rd := Rs1 \text{ DIV } Rs2$
rem	Rd, Rs1, Rs2	$Rd := Rs1 \text{ MOD } Rs2$
mul	Rd, Rs1, Rs2	$Rd := Rs1 \times Rs2$
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 $Rs1 \neq Rs2$
bnez	Rs1, label	Sprung, falls $Rs1 \neq 0$
bge	Rs1, Rs2, label	Sprung, falls $Rs1 \geq Rs2$
bgeu	Rs1, Rs2, label	Sprung, falls $Rs1 \geq Rs2$
bgez	Rs, label	Sprung, falls $Rs \geq 0$
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 $Rs1 \leq Rs2$
bleu	Rs1, Rs2, label	Sprung, falls $Rs1 \leq Rs2$
blez	Rs, label	Sprung, falls $Rs \leq 0$
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	$Rd := \neg Rs1$ (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	$Rd := \text{MEM}[\text{Adr}]$
lw	Rd, Adr	$Rd := \text{MEM}[\text{Adr}]$
li	Rd, Imm	$Rd := Imm$
sw	Rs, Adr	$\text{MEM}[\text{Adr}] := Rs$ (Speichere ein Wort)
sh	Rs, Adr	$\text{MEM}[\text{Adr}] \text{ MOD } 2^{16} := Rs$ (Speichere ein Halbwort)
sb	Rs, Adr	$\text{MEM}[\text{Adr}] \text{ MOD } 256 := Rs$ (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