Ludwig-Maximilians-Universität München
Institut für Informatik
Lehrstuhl für Mobile und Verteilte Systeme
Prof. Dr. Claudia Linnhoff-Popien

mobile and distributed systems group

Online-Hausarbeit 5a
Rechnerarchitektur im SoSe 2020

Abgabetermin: Geben Sie Ihre Lösung im Uni2Work bis zur Deadline am 07.07.2020, 18:59:00 Uhr, ab! Sollten Sie nachweislich Internetprobleme haben, die eine Abgabe bis 18:59:00 Uhr nicht ermöglichen, so geben Sie bitte bis 23:59:59 Uhr ab und schreiben uns parallel dazu eine E-Mail, wo Sie um eine verlängerte Abgabe bitten und Ihre Umstände erklären.

Ankündigungen: Wir sind darauf aufmerksam gemacht worden, dass die Online-Hausarbeit 5 zu SPIM: Caesar-Verschlüsselung bei Stack Overflow eingestellt und beantwortet wurde.
In der Abwägung aller zu berücksichtigenden Aspekte hat Frau Linnhoff folgendes beschlossen: Die Online-Hausarbeit 5 wird nicht bewertet, stattdessen gibt es diese Woche eine Online-Hausarbeit 5a zur gleichen Thematik. Die verbleibenden Online-Hausarbeiten 6, 7 und 8 verschieben sich um je eine Woche.

Bitte fügen Sie die folgende Selbständigkeitserklärung vollständig und unterschrieben Ihrer Abgabe hinzu.

Selbstständigkeitserklärung

Hiermit versichere ich, dass die abgegebene Lösung alleinig durch mich angefertigt wurde und ohne die Hilfe Dritter entstanden ist. Insbesondere habe ich keine Lösungen von Dritten teilweise oder gänzlich abgegeben.

Matrikelnummer, Name

Ort, Datum

Unterschrift

OH11: Assemblerprogrammierung des MIPS-Prozessors

(10 Pkt.)

Bearbeiten Sie die folgende Aufgabe zum Thema Assemblerprogrammierung unter SPIM.
Hinweis: Eine Übersicht der SPIM-Befehle finden Sie am Ende.
Im Folgenden soll ein MIPS-Assembler Programm vervollständigt werden, welches für ein im Datensegment des Programms gegebenes Wort die Anzahl der Vokale ermittelt und eine Kopie des Wortes, welche dann keine Vokale mehr enthält, im Datensegment speichert. Abschließend soll das Programm die Anzahl der Vokale sowie das Wort, das keine Vokale mehr enthält, auf der Konsole ausgeben.
Zur Erläuterung: Angenommen, es ist das Wort "Worte" gegeben. Das Programm soll dann "Worte enthaelt: 2 Vokale." und "Worte ohne Vokale ist: Wrt" auf der Konsole ausgeben.

a. Geben Sie für jeden der folgenden Kommentare die Nummer der Code-Zeile an, zu dem er am besten passt.

   (i) Der aktuelle Buchstabe des Strings `needle` wird in ein Register geladen.

   (ii) Die Anzahl der gezählten Vokale wird auf der Konsole ausgegeben.

   (iii) Erhöhe die Anzahl der gezählten Vokale um eins.

   (iv) Führe einen Sprung durch, sofern alle Buchstaben des Strings `needle` betrachtet wurden.

b. Ergänzen Sie den unten angegebenen Coderahmen um insgesamt **6 Zeilen Code**, so dass das Programm wie beschrieben funktioniert. Tragen Sie Ihre Lösung unter den mit "`# Ihre Loesung:`" markierten Stellen direkt in den folgenden Coderahmen ein:

```
1  .data
2  needle: .asciiz "Rechnerarchitektur"
3  result: .space 18
4  ascii_vocals: .asciiz "aeiouAEIOU"
5  string1: .asciiz "Rechnerarchitektur enthaelt: "
6  string2: .asciiz " Vokale."
7  string3: .asciiz "\nRechnerarchitektur ohne Vokale ist: "
8  nline: .asciiz "\n"
9
10 .text
11 main:   li $t0, 0        # Zaehler fuer die Anzahl der Vokale
12         li $t1, 0        # Index des aktuell betrachteten Zeichens von needle
13         li $t2, 0        # Index des aktuell betrachteten Vokals in ascii_vocals
14         li $t3, 0        # Index des naechsten freien Speicherplatzes in result
15
16
17 n_loop: lb $t4, needle($t1)
18         beqz $t4, end
19
20 vocs:   ###################################
21         # Fuegen Sie hier Ihre Loesung ein #
22         ###################################
23
24
25
26
27
28
29
30
31
```

```
32          ######################
33          # Ende Ihrer Loesung #
34          ######################
35          j vocs
36
37  vocal:  addi $t0, $t0, 1
38          li $t2, 0
39          addi $t1, $t1, 1
40          j n_loop
41
42
43  # Der Coderahmen geht auf der folgenden Seite weiter!
44
45  save:   ###################################
46          # Fuegen Sie hier Ihre Loesung ein #
47          ###################################
48
49
50
51
52
53
54
55
56          ######################
57          # Ende Ihrer Loesung #
58          ######################
59
60  reset:  add $t1, $t1, 1
61          li $t2, 0
62          j n_loop
63
64
65
66  end:    li $v0, 4
67          la $a0, string1
68          syscall
69
70          li $v0, 1
71          move $a0, $t0
72          syscall
73
74          li $v0, 4
75          la $a0, string2
76          syscall
77
78          li $v0, 4
79          la $a0, string3
80          syscall
81
82          li $v0, 4
83          la $a0, result
84          syscall
85
86          li $v0, 10
87          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 $\times$ 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 := 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