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

mobile and distributed systems group

Online-Hausarbeit 5
Rechnerarchitektur im SoSe 2020

Abgabetermin: Geben Sie Ihre Lösung im Uni2Work bis zur Deadline am 30.06.2020, 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.

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

OH10: SPIM: Caesar-Verschlüsselung

(10 Pkt.)

Bearbeiten Sie die folgende Aufgabe zum Thema Assemblerprogrammierung unter SPIM.
Im Folgenden soll ein MIPS-Assembler Programm vervollständigt werden, welches einen gegebenen Text mittels der **Caesar-Verschluesselung** in einen Geheimtext umwandelt. Bei der Caesar-Verschluesslung wird jeder Buchstabe im zu verschluesselnden Text um eine vorher festgelegte Distanz im Alphabet verschoben. Ist z.B. die Distanz 3, so wird der Buchstabe A zum Buchstaben D, der Buchstabe B zum Buchstaben E, ..., der Buchstabe Z zum Buchstaben C.
Das folgende MIPS-Assembler Programm erwartet als Nutzereingabe die Distanz, um die die Buchstaben verschoben werden soll und verschluesselt dann einen gegebenen Text.

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

(i) Der aktuelle Buchstabe wird in ein Register geladen.

(ii) Der Text mit Anfangsadresse in a0 wird auf der Konsole ausgeben.

(iii) Führe einen Sprung durch, sofern ein Wert größer ist, als der Wert der ASCII-Darstellung des Buchstaben „Z“.

(iv) Führe einen Sprung durch, sofern alle zu verschlüsselnden Buchstaben 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
3  shift_text: .asciiz "Um wieviele Stellen soll der Text verschoben werden: "
4  string1: .asciiz "Der verschluesselte Text lautet:  "
5  secret: .asciiz "GEHEIMNIS"
6  string_a: .asciiz "A"
7  string_z: .asciiz "Z"
8
9  result: .space 9
10
11 .text
12 main:
13         # t0  - Zum Zwischenspeichern der Position des aktuell betrachteten Buchstabens
14         # t1 - Gibt die Laenge des Geheimworts an
15         # t2 - ASCII Wert des Buchstaben A (65)
16         # t3 - ASCII - WERT des Buchstaben Z (90)
17         li  $t0, 0
18         li  $t1, 9
19         lb  $t2, string_a
20         lb  $t3, string_z
21
22         la $a0, shift_text
23         li $v0, 4
24         syscall
25
26         li $v0, 5
27         syscall
28
29         move $s1, $v0
30
31 loop:   bge $t0, $t1, end
32         lb  $t4, secret($t0)
33
34 caesar: ###################################
35         # Fuegen Sie hier Ihre Loesung ein #
36         ###################################
37
38
39
40         ######################
41         # Ende Ihrer Loesung #
42         ######################
43         bgt $t4, $t3, cadd
```

```
44
45 save:   ####################################
46         # Fuegen Sie hier Ihre Loesung ein #
47         ####################################
48
49
50
51
52
53
54         ########################
55         # Ende Ihrer Loesung #
56         ########################
57         j loop
58
59 cadd:   ####################################
60         # Fuegen Sie hier Ihre Loesung ein #
61         ####################################
62
63
64
65
66
67
68
69         ########################
70         # Ende Ihrer Loesung #
71         ########################
72         j save
73
74
75 end:    la $a0, string1
76         li $v0, 4
77         syscall
78
79         la $a0, result
80         li $v0, 4
81         syscall
82
83         li $v0, 10
84         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
sltu	Rd, Rs1, Rs2	Rd := 1 if Rs1 < Rs2 else 0; Rs1, RS2 sind unsigned integers
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