



Technische Informatik II im SS 2005

Musterlösungen zum 5. Übungsblatt

Prof. Dr. U. Brinkschulte

Geb. 40.28, D-76131 Karlsruhe

Email: brinks@ira.uka.de

Dr.-Ing. T. Asfour

Telefon: +49-721-608-7379

Fax: +49-721-608-8270

Email: asfour@ira.uka.de

<http://i61www.ira.uka.de/users/asfour/TI>

Lösung 1

1. Laden von 1111 0000 0011 1101 0000 1001 0000 1001 ins Register \$s0:

```
lui $s0, 1111 0000 0011 1101 # load upper immediate
ori $s0, 0000 1001 0000 1001
```

oder auch

```
lui $s0, 1111 0000 0011 1101
addi $s0, $s0, 0000 1001 0000 1001
```

2. Die 2 niedrigstwertigen Bits einer Wortadresse haben den Wert 0
3. Register- und Speicherinhalte nach der Ausführung:

Registersatz		Hauptspeicher	
Register	Inhalt	Adresse	Inhalt
\$t0	0x10	\$0x20	0x22
\$t1	0x40	\$0x24	0x30
\$t2	0x16	\$0x28	0x40
\$t3	0x20	\$0x2C	0x50
\$t4	0x30	\$0x30	0x30

Lösung 2

Das Programm ermittelt das maximale Element im Array a[36, 20, 27, 15, 1, 62, 41]. Zur Ermittlung des maximalen Elements wird eine if-then-Schleife benutzt.

```
.data
a:      .word 36, 20, 27, 15, 1, 62, 41
n:      .word 7
max:    .word 0

.text
```

```

        .globl main

main:    li $t0, 0           # Array-Index i=0 und in $t0 speichern.
        li $s0, 0           # max=0 und in $s0 ablegen
        lw $s1, n           # Anzahl der Array-Elemente in $s1

m1:      bge $t0, $s1, m3
        mul $t1, $t0, 4      # i auf Wortgrenze skalieren
        lw $t2, a($t1)       # Lade a[i] ins Register $t2
        ble $t2, $s0, m2     # if a[i] <= max, dann "then-Teil"
        move $s0, $t2        # "then-Teil": max = a[i]
m2:      addi $t0, $t0, 1     # Array-Index i inkrementieren
        b m1
m3:      move $a0, $s0        # Ende der Schleife
        li $v0, 1
        syscall
        li $v0, 10
        syscall

```

Lösung 3

1. (a) Funktion des Programmstücks:
Addiert alle ungeraden Zahlen, die kleiner oder gleich n sind.
- (b) Wert im Register $\$v0$ nach Abarbeitung des Programmstücks:

Wenn $\$a0 = 9$ dann $\$v0 = 25$

Wenn $\$a0 = 10$ dann $\$v0 = 25$

2.

Pseudobefehle	Echte Befehle
move \$t5, \$t3	addi \$t5, \$t3, 0
clear \$t5	add \$t5, \$zero, \$zero
bgt \$t5, \$t3, marke	slt \$at, \$t3, \$t5 bne \$at, \$zero, marke
bge \$t5, \$t3, marke	slt \$at, \$t3, \$t5 bne \$at, \$zero, marke beq \$t5, \$t3, marke

3. (a) `lw $s1, 100($s2)`:
Laden des Wortes (32-Bit) mit der Adresse ($100 + \text{Inhalt des Registers } \$s2$) ins Register $\$s1$

(b) `sw $s1, 100($s2):`

Speichern des Wortes im Register `$s1` an der Adresse (`100 + Inhalt des Registers $s2`)

(c) `jal mystery:`

Unbedingter Sprung zur Marke `mystery` und Speicherung der Adresse des nächsten Befehls (Rücksprungadresse) im Register `$ra`

Lösung 4

1. Registerinhalte:

Register	Inhalt
<code>\$t1</code>	<code>\$t1 = 0x0000 0008</code>
<code>\$t2</code>	<code>\$t2 = 0x0000 000C</code>
<code>\$t3</code>	<code>\$t3 = 0x0000 0010</code>
<code>\$t4</code>	<code>\$t4 = 0x0000 0013</code>

2. MIPS-Code zur Speicherung der Adresse von `vec` im Register `$s0`:

```
la $s0, vec
```

3. Programmschleife zur Ausgabe der ersten fünf Elemente aus `vec`:

```
.data
vec:  .word 8, 12, 16, 19, 2002, 0, 0, 0, 0, 0
leer: .asciiz " "

.text

    la  $s0, vec          # Adresse von vec in $a0
    addi $t0, $0, 0        # Zähler = 0
    addi $t1, $0, 5        # zum Beenden der Schleife
    add  $t2, $0, $s0

loop: lw  $a0, 0($t2)      # Element aus vec in $a0 laden
      li  $v0, 1           # print_int
      syscall
      la  $a0, leer
      li  $v0, 4           # print-str (Leerzeichen)
      syscall
      addi $t0, $t0, 1     # Zähler inkrementieren
      addi $t2, $t2, 4     # Zeiger auf vec inkrementieren
      bne $t0, $t1, loop
```

Lösung 5

Berechnung von n ueber k

```

.data
cr_string: .ascii "\n"          # Sonderzeichen "neue Zeile"
eingabeN:  .ascii "n: "
eingabeK:  .ascii "k: "
result_str: .ascii "n ueber k = "
error_str: .ascii "n < k ist nicht erlaubt!"

.text
.globl main
main:      subu $sp, $sp, 8      # Stack Frame ist 8 Bytes
           sw $ra, 0($sp)       # Sichern der Ruecksprungadresse
           sw $fp, 4($sp)       # Sichern des alten Frame-Pointers
           addu $fp, $sp, 8      # neuen Frame-Pointer definieren

           la $a0, eingabeN     # n holen
           li $v0, 4
           syscall
           li $v0, 5
           syscall
           move $s0, $v0        # n nach $s0

           la $a0, eingabeK     # k holen
           li $v0, 4
           syscall
           li $v0, 5
           syscall
           move $s1, $v0        # k nach $s1

           blt $s0, $s1, error   # wenn n < k, dann zu error

           li $s3, 1             # Zaehlerprodukt in $s3
           li $s4, 1             # Nennerprodukt in $s4
label1:    beqz $s1, label2      # wenn k = 0, dann zu fertig
           mul $s4, $s4, $s1
           subu $s1, $s1, 1
           mul $s3, $s3, $s0
           subu $s0, $s0, 1
           b label1
label2:    la $a0, result_str    # Ausgabe des Ergebnisses
           li $v0, 4
           syscall
           divu $a0, $s3, $s4
           li $v0, 1
           syscall
           b fertig

error:     la $a0, error_str     # n < k
           li $v0, 4
           syscall

fertig:    lw $ra, 0($sp)        # Ruecksprungadresse wiederherstellen
           lw $fp, 4($sp)        # Frame-Pointer wiederherstellen
           addu $sp, $sp, 8      # Stack-Frame loeschen
           jr $ra

```