



Technische Informatik II im SS 2007

Musterlösungen zum 10. Übungsblatt

Prof. Dr. J. Henkel

Am Zirkel 2, Geb. 20.20
D-76131 Karlsruhe

Dr.-Ing. T. Asfour

Telefon: +49-721-608-7379
Fax: +49-721-608-8270
Email: asfour@ira.uka.de
<http://ti.ira.uka.de>

Lösung 1

Ausgabe eines ASCII-Zeichens entsprechend einer Integer-Zahl

```
.data
cr_string: .asciiz "\n"          # Sonderzeichen "neue Zeile"
asc_str:   .ascii "ASCII-Zeichen \"
char:      .space 1
           .asciiz "\"\n"
eingabe_str: .asciiz "Integer-Zahl zwischen 32 und 126 ein: "
error_str:  .asciiz "Integer-Zahl liegt nicht im ASCII-Bereich!\n"

.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, eingabe_str    # Eingabe holen
          li $v0, 4
          syscall
          li $v0, 5
          syscall
          sb $v0, char

          blt $v0, 32, error     # Wenn kleiner als 32, dann zu error
          bgt $v0, 126, error    # Wenn groesser als 126, dann zu error
          la $a0, asc_str       # Ausgabe des Ergebnisses
          b fertig

error:    la $a0, error_str      # Integer-Zahl nicht im ASCII-Bereich
fertig:   li $v0, 4
          syscall

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

Lösung 2

Verschlusselung eines Strings

```
.data
sp_string: .asciiz " "
eingabe_str: .asciiz "Bitte geben Sie eine Zeichenkette (max. 40 Zeichen) ein: "
key_str:     .asciiz "Bitte geben Sie eine Zahl zwischen 0 und 255 ein: "
crypt_str:   .space 40
           .byte 0
```

```

        .text

# Prozedur: Ausgabe eine Integer-Zahl mit Leerzeichen
print_int:    li $v0, 1
              syscall
              la $a0, sp_string
              li $v0, 4
              syscall
              jr $ra

# Start des Hauptprogrammes

        .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, eingabe_str # Eingabe holen
              li $v0, 4
              syscall
              la $a0, crypt_str
              li $a1, 40
              li $v0, 8
              syscall

              la $a0, key_str     # Schluessel holen
              li $v0, 4
              syscall
              li $v0, 5
              syscall
              move $s2, $v0       # Schluessel in $s2 sichern

              move $s0, $0        # $s0 := 0
label1:      lbu $s1, crypt_str + 0($s0)
              beq $s1, 10, fertig # Wenn letztes Zeichen, dann zu fertig
              xor $a0, $s1, $s2   # exklusiv-oder von Zeichen und Schluessel
              jal print_int
              addu $s0, $s0, 1    # naechstes Zeichen
              b label1

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

```

Die Zeichenkette lautet: Mikroprozessoren, Speicher, Peripherie

Lösung 3

1. (a) Funktion des Programmst $\tilde{A}\frac{1}{4}$ cks:
Addiert alle ungeraden Zahlen, die kleiner oder gleich n sind.
- (b) Wert im Register $\$v0$ nach Abarbeitung des Programmst $\tilde{A}\frac{1}{4}$ cks:

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 + 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

Berechnung trigonometrischer Funktionen

```
.data
cr_string: .asciiz "\n"           # Sonderzeichen "Neue Zeile"
sp_string: .asciiz " "           # Leerzeichen
enter_str: .asciiz "Bitte geben Sie ein Zahl ein: "
result1_str: .asciiz "Ergebnis der Sinus-Berechnung: "
result2_str: .asciiz "Ergebnis der Cosinus-Berechnung: "
result3_str: .asciiz "Ergebnis der Tangens-Berechnung: "
error_str:  .asciiz "Tangens-Funktion nicht definiert!"
```

```
.text
```

Prozedur: Ausgabe einer Double-Zahl in \$f12 mit CR

```
print_dbl: li $v0, 3
           syscall
           la $a0, cr_string
           li $v0, 4
           syscall
           jr $ra
```

Prozedur: Fakultät von \$a0, Ergebnis in \$f0

```
fakultaet: subu $sp, $sp, 20      # Stack Frame ist 20 Bytes
           s.d $f4, 0($sp)       # FP4 sichern
           s.d $f6, 8($sp)       # FP6 sichern
           sw $fp, 16($sp)       # Sichern des alten Frame-Pointers
           addu $fp, $sp, 20      # neuen Frame-Pointer definieren
```

```

        li.d $f0, 1.0          # Ergebnisvariable
        li.d $f4, 1.0          # Double-Zaehler
        li.d $f6, 1.0          # Konstante 1

label1:    beqz $a0, label2
           mul.d $f0, $f0, $f4  # Ergebnis := Ergebnis * Double-Zaehler
           add.d $f4, $f4, $f6  # Double-Zaehler um 1 erhoehen
           subu $a0, $a0, 1     # $a0 um 1 vermindern
           b label1

label2:    l.d $f4, 0($sp)      # FP4 laden
           l.d $f6, 8($sp)      # FP6 laden
           lw $fp, 16($sp)      # Sichern des alten Frame-Pointers
           addu $sp, $sp, 20    # Stack Frame loeschen
           jr $ra

# Prozedur: Exponent $f0 hoch $a0, Ergebnis in $f0

exponent:  subu $sp, $sp, 12    # Stack Frame ist 12 Bytes
           s.d $f2, 0($sp)      # FP2 sichern
           sw $fp, 8($sp)      # Sichern des alten Frame-Pointers
           addu $fp, $sp, 12    # neuen Frame-Pointer definieren

label4:    li.d $f2, 1.0        # Ergebnis $f2 mit 1 initialisieren
           beqz $a0, label3     # wenn $a0 = 0, dann zu label3 (fertig!)
           mul.d $f2, $f2, $f0  # Ergebnis := Ergebnis * $f0
           subu $a0, $a0, 1     # $a0 um 1 vermindern
           b label4            # zurueck zu label2
label3:    mov.d $f0, $f2       # Ergebnis nach $f0

           l.d $f2, 0($sp)      # FP2 laden
           lw $fp, 8($sp)      # Sichern des alten Frame-Pointers
           addu $sp, $sp, 12    # Stack Frame loeschen
           jr $ra

# Prozedur: Wenn $a0=1, dann Sinus von $f0
#           Wenn $a0=0, dann Cosinus von $f0

co_sinus:  subu $sp, $sp, 40    # Stack Frame ist 40 Bytes
           s.d $f2, 0($sp)      # FP2 sichern
           s.d $f4, 8($sp)      # FP4 sichern
           s.d $f8, 16($sp)     # FP8 sichern
           s.d $f12, 24($sp)    # FP12 sichern
           sw $ra, 32($sp)      # Sichern der Ruecksprungadresse
           sw $fp, 36($sp)      # Sichern des alten Frame-Pointers
           addu $fp, $sp, 40    # neuen Frame-Pointer definieren

           mov.d $f2, $f0       # Eingabe $f0 in $f2 sichern

           li.d $f4, 0.0        # Ergebnisvariable $f4 initialisieren
           move $s0, $a0        # Integer-Zaehler $s0 initialisieren

loop1:     move $a0, $s0
           jal fakultaet        # Fakultaeet von $s0 berechnen
           mov.d $f8, $f0       # und in $f8 zwischenspeichern

           mov.d $f0, $f2
           move $a0, $s0
           jal exponent         # $f0 := Eingabe hoch Integer-Zaehler
           div.d $f8, $f0, $f8   # $f8 := Eingabe^$s0 / $s0!
           mov.d $f12, $f4      # altes Ergebnis sichern

           andi $s1, $s0, 2      # das zweit-niederwertigste Bit
                                   # von $s0 ausmaskieren
           beqz $s1, loop3       # wenn (-1)^$s0 positiv, dann addiere
           sub.d $f4, $f4, $f8   # sonst subtrahiere
           b loop2

loop3:     add.d $f4, $f4, $f8
loop2:     addu $s0, $s0, 2      # Integer-Zaehler um 2 erhoehen
           c.eq.d $f12, $f4      # Vergleiche altes und neues Ergebnis
           bc1f loop1           # wenn verschieden, dann zu loop1
           mov.d $f0, $f4       # Ergebnis $f4 in $f0 sichern

           l.d $f2, 0($sp)      # FP2 wiederherstellen

```

```

        l.d $f4, 8($sp)      # FP4 wiederherstellen
        l.d $f8, 16($sp)    # FP8 wiederherstellen
        l.d $f12, 24($sp)   # FP12 wiederherstellen
        lw $ra, 32($sp)     # Wiederherstellen der Ruecksprungadresse
        lw $fp, 36($sp)     # Wiederherstellen des Frame-Pointers
        addu $sp, $sp, 40   # Stack-Frame loeschen
        jr $ra

# Start des Hauptprogrammes

        .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, enter_str
        li $v0, 4
        syscall            # Eingabeaufforderung
        li $v0, 7
        syscall            # Eingabezahl nach $f0 holen
        mov.d $f4, $f0     # Eingabe sichern

        li $a0, 1
        jal co_sinus       # Sinus-Funktion berechnen
        la $a0, result1_str
        li $v0, 4
        syscall            # Ergebnisstring
        mov.d $f12, $f0
        jal print_dbl      # Ergebnis ausgeben
        mov.d $f2, $f0     # Ergebnis sichern

        mov.d $f0, $f4
        move $a0, $0
        jal co_sinus       # Cosinus-Funktion berechnen
        la $a0, result2_str
        li $v0, 4
        syscall            # Ergebnisstring
        mov.d $f12, $f0
        jal print_dbl      # Ergebnis ausgeben

        li.d $f6, 0.0
        c.eq.d $f0, $f6
        bclt undefined    # Wenn $f0 = 0, dann zu undefined
        div.d $f12, $f2, $f0 # Tangens-Funktion berechnen
        la $a0, result3_str
        li $v0, 4
        syscall            # Ergebnisstring
        jal print_dbl      # Ergebnis ausgeben
        b fertig

undefined: la $a0, error_str
        li $v0, 4
        syscall            # Tangens nicht definiert

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

```