Karlsruhe Institute of Technology
Institut für Anthropomatik und Robotik (IAR)

Rechnerorganisation im WS 2017/18

Musterlösungen zum 5. Übungsblatt

Prof. Dr. Wolfgang Karl
Haid-und-Neu-Str. 7

Dr.-Ing. Ömer Terlemez
Adenauerring 2, Geb. 50.20

Email: ti@ira.uka.de
Web: http://ti.ira.uka.de

Lösung 1 (6 Punkte)

1. Bei einem Pseudobefehl handelt es sich um ein Mnemonik, das vom Assemblierer nicht auf einen Maschinenbefehl abgebildet wird. Es findet eine Ersetzung durch einen oder mehrere andere Befehle statt. Der Befehlsumfang kann mit dieser Technik für den Programmierer einfach erweitert werden, ohne dass die Komplexität der CPU erhöht wird. 2 P.

   Pseudobefehle sind üblicherweise direkte Spezialfälle allgemeinerer Befehle (z.B. `b`, `clear` und `neg`) und Befehle wie `li`, die aus technischen Gründen nicht als ein Befehl codiert werden können. Das Problem des `li` Befehls liegt in der festen Breite eines Befehlswortes (32 Bit). Diese feste Länge verhindert das Speichern des Opcodes und der 32 Bit langen Konstanten in einem Befehlswort.

2. Eine Assemblerdirektive wird vom Assemblierer nicht zu unmittelbaren Maschinenbefehlen übersetzt, sondern beeinflusst das Verhalten des Assemblierers selbst. 2 P.

   Die Direktive `.data` sorgt hierbei dafür, dass die folgenden Anweisungen im Datensegment des Programms abgelegt werden (das Codesegment wird mit `.text` ausgewählt).

   Die Direktive `.float 20.12` reserviert 4 Byte für eine Gleitkommazahl mit einfacher Genauigkeit (*single precision*) und initialisiert den Speicherbereich mit dem Wert 20.12.

3. In den `$tX`-Registern werden vorwiegend temporäre Variablen abgelegt, während die `$sX`-Register zur längerfristigen Speicherung von Werten verwendet werden. 2 P.

   Entsprechend besagt die Aufrufkonvention, dass der Wert der `$tX`-Register von Funktionen beliebig überschrieben werden darf.

   Vor dem Verändern eines `$sX`-Register muss hingegen der alte Wert gesichert und vor dem Rücksprung wieder hergestellt werden. Das Speichern erfolgt typischerweise auf dem Stack.

   Es handelt sich hierbei um eine Konvention der Programmierer, die keine Auswirkungen auf das Hardware-Design hat. Auf Hardwareebene sind die Register identisch behandelte General Purpose Register.

Lösung 2 (4 Punkte)

```
        .text
        .globl main

main:   addi $v0, $zero, 5   # Read number
        syscall
        add $s0, $zero, $v0

        sub $a0, $zero, $s0  # Output negated value
        addi $v0, $zero, 1
        syscall

        add $a0, $zero, $s0  # Output absolute value
        bgez $s0, skip
        sub $a0, $zero, $s0
skip:   addi $v0, $zero, 1
        syscall

        addi $t0, $zero, 3   # Output tripled value
        mult $s0, $t0
        mflo $a0
        addi $v0, $zero, 1
        syscall

        li $v0, 10           # Exit
        syscall
```

Lösung 3 (7 Punkte)

1. i.) Funktion des Programmstücks: 2 P.
Addiert alle ungeraden Zahlen, die kleiner oder gleich n sind.

ii.) Wert im Register `$v0` nach Abarbeitung des Programmstücks:

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

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

2. 2 P.

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. i.) `lw $s1, 100($s2)`: 3 P.
Laden des Wortes (32-Bit) mit der Adresse (100 + Inhalt des Regsites `$s2`) ins Register `$s1`

ii.) `sw $s1, 100($s2)`:
Speichern des Wortes im Register `$s1` an der Adresse (100 + Inhalt des Regsites `$s2`)

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

Lösung 4 (6 Punkte)

Siehe `blatt07_vollkommene_zahl_musterloesung.s`.

```
          .data

          # String literals for output
prompt:   .asciiz "Bitte Zahl eingeben: "
yes:      .asciiz "Zahl ist vollkommen!"
no:       .asciiz "Zahl ist nicht vollkommen!"
newline:  .asciiz "\n"

          .text
          .globl main

          # Register usage:
          # $s0: Number
          # $s1: Currently tested divisor
          # $s2: Current sum of all divisors

          # print prompt string (syscall 4)
main:     li   $v0, 4
          la   $a0, prompt
          syscall

          # Read number from user (syscall 5)
          li   $v0, 5
          syscall
          # store read number in $s0
          move $s0, $v0

          li   $s1, 1 # initialize divisor
          li   $s2, 0 # initialize sum

          # jump to outputno: if number is
          # smaller than 1, otherwise proceed
          ble  $s0, $s1, outputno

          # Check all numbers up to $s0
          # calculate $s0 % $s1 (modulus)
loop:     rem  $t0, $s0, $s1
          # branch to nofactor: if modulus != 0
          bnez $t0, nofactor
          # print divisor
          li   $v0, 1
          move $a0, $s1
          syscall
          li   $v0, 4
          la   $a0, newline
          syscall
```

```
          # add current divisor to sum
          add  $s2, $s2, $s1
          # increase current divisor
nofactor: addi $s1, $s1, 1
          # branch to loop: unless current divisor
          # is equal to $s0
          blt  $s1, $s0, loop

          # jump to outputno: if number is
          # not perfect, otherwise proceed
          bne  $s0, $s2, outputno

          # Number is perfect (load yes: string)
          la   $a0, yes
          # jump to finish: label for
          # result output
          j    finish

outputno: # Number is not perfect (load no: string)
          la   $a0, no

finish:   # Output string (syscall 4)
          li   $v0, 4
          syscall

          jr   $ra
```

Lösung 5 (6 Punkte)

Siehe `blatt07_pi_bbp_musterloesung.s`.

```
#
# Karlsruher Institut fuer Technologie
# Institut fuer Anthropomatik und Robotik (IAR)
# Vorlesung Rechnerorganisation
#
# Autor: James Bond
# Matrikelnummer: 007
# Tutoriumsnummer: 42
# Name des Tutors: TI
#


        .data
str:    .asciiz "PI: "
zero:   .double 0
one:    .double 1
two:    .double 2
three:  .double 3
four:   .double 4
ff:     .double 16

        .text
        .globl main

main:
        l.d $f12, zero  # pi
        li  $s3,  10     # counter

        l.d $f2, one    # 8k+x
        l.d $f4, one    # 16^k

loop:
        # 4 / (8k+1)
        l.d $f0, four
        div.d $f6, $f0, $f2
        l.d $f0, three
        add.d $f2, $f2, $f0

        # 2 / (8k+4)
        l.d $f0, two
        div.d $f8, $f0, $f2
        l.d $f0, one
        add.d $f2, $f2, $f0
        # 4/(8k+1) - 2/(8k+4)
        sub.d $f6, $f6, $f8
```

```
# 1 / (8k+5)
l.d $f0, one
div.d $f8, $f0, $f2
add.d $f2, $f2, $f0
# 4/(8k+1) - 2/(8k+4) - 1/(8k+5)
sub.d $f6, $f6, $f8

# 1 / (8k+6)
div.d $f8, $f0, $f2
l.d $f0, three
add.d $f2, $f2, $f0
# 4/(8k+1) - 2/(8k+4) - 1/(8k+5) - 1/(8k+6)
sub.d $f6, $f6, $f8

# 1/16^k * (4/(8k+1) - 2/(8k+4) - 1/(8k+5) - 1/(8k+6))
div.d $f6, $f6, $f4
add.d $f12, $f12, $f6

# 16^k
l.d $f0, ff
mul.d $f4, $f4, $f0

subi $s3, $s3, 1
bgez $s3, loop


la $a0, str
li $v0, 4
syscall

# print_double
li $v0, 3
syscall

li $v0, 10              # exit
syscall
```