Aufgabe 25

Die jeweils verwendeten Adressierungsarten sind als Kommentare vor den entsprechenden Speicherzugriffen beschrieben, die Registerbelegungen in den Kommentaren rechts neben den Befehlen:

```
# Aufgabe 25: Adressrechnung
#
# von: Stephan Brumme, nach Vorlage
#
# zuletzt geaendert: 24.Mai 2000

        .data

wert:   .word   20,4,27,99,9


        .text
main:
        # direkte Adressierung
        lw      $t1,wert              # $t1 enthaelt: 20

        # Register-indirekte Adressierung
        li      $t3,0x10010000        # $t3 enthaelt: 0x10010000
        lw      $t4,($t3)             # $t4 enthaelt: 20

        # direkte Adressierung
        lw      $t1,wert+4            # $t1 enthaelt: 4

        # indexierte Adressierung
        lw      $t2,wert($t1)         # $t2 enthaelt: 4
        lw      $t2,wert+8($t1)       # $t2 enthaelt: 99

        li      $v0,10                # Programm beenden
        syscall
```

Auf das Datum „9“ kann in den drei verschiedenen Adressierungsarten wie folgt zugegriffen werden (Ergebnis steht dann jeweils in `$t0`):

a) Register-indirekt:
```
li $t1,0x10010010
lw $t0,($t1)
```

b) direkt:
```
lw $t0,wert+16
```

c) indexiert:
```
li $t1,16
lw $t0,wert($t1)
```

Aufgabe 26

Zuerst der Quellcode eines kleinen Assemblerprogramms, mit dem sowohl Aufgabe a), als auch Aufgabe b) bearbeitet wurde:

```
# Aufgabe 26: Ladebefehle
#
# von: Stephan Brumme, nach Vorlage
#
# zuletzt geaendert: 24.Mai 2000

        .data

a_wert1:        .word 0xffffff00
a_wert2:        .word 0x000000ff

b_wort:         .word 0
b_halbwort:     .half 0
b_bytes:        .byte 0,0
```

```
        .text
main:


# Aufgabe a
        lb      $t0,a_wert1           # Bytes aus Wert1/2 laden
        lb      $t1,a_wert2

        lbu     $t2,a_wert1           # vorzeichenlose Bytes aus Wert1/2 laden
        lbu     $t3,a_wert2

        lh      $t4,a_wert1           # Halfwords aus Wert1/2 laden
        lh      $t5,a_wert2

        lhu     $t6,a_wert1           # vorzeichenlose Halfwords aus Wert1/2 laden
        lhu     $t7,a_wert2


# Aufgabe b
        li      $s0,20                # 20 als
        sw      $s0,b_wort            # Word,
        sh      $s0,b_halbwort        # Halfword
        sb      $s0,b_bytes           # und Byte abspeichern


        li      $v0,10                # Programm beenden
        syscall
```

a) Mein Programm erzeugt in den Register `$t0` bis `$t7` folgende Veränderungen:

Bytezugriff, vorzeichenbehaftet	`$t0`	`0x00000000`
	`$t1`	`0xffffffff`
Bytezugriff, vorzeichenlos	`$t2`	`0x00000000`
	`$t3`	`0x000000ff`
Halfwordezugriff, vorzeichenbehaftet	`$t4`	`0xffffff00`
	`$t5`	`0x000000ff`
Halfwordzugriff, vorzeichenlos	`$t6`	`0x0000ff00`
	`$t7`	`0x000000ff`

Sie begründen sich in der unterschiedlich Behandlung des Zweierkomplements: Besitzt die Zahl, die eingelesen wird und weniger als 32 Bit hat, an ihrer höchstwertigsten Bitstelle eine 1, so wird sie als negative Zahl aufgefaßt. Das Register, in das sie geladen wird, besteht aber aus 32 Bit und demzufolge müssen die restlichen Bits aufgefüllt werden: bei negativen Zahlen mit Einsen, bei positiven mit Nullen. Der Grundsatz $K_2(z_{8/16})=K_2(z_{32})$ bleibt also stets erhalten.
Der vorzeichenlose Zugriff auf den Datenspeicher füllt die führenden Bits generell mit Nullen auf, da er nur von positiven Zahlen ausgeht. Der Grundsatz $K_2(z_{8/16})=K_2(z_{32})$ kann dadurch nicht mehr erfüllt werden, er gilt nur noch für positive $z_{8/16}$.
Die mathematischen Zusammenhänge werden deutlicher, wenn die Hexadezimalwerte in vorzeichenbehaftete Dezimalwerte umgerechnet werden:

a_wert1	a_wert2	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7
-256	255	0	-1	0	255	-256	255	65280	255

Der Vergleich von `a_wert1` bzw. `a_wert2` mit `$t0`, `$t1`, `$t4` und `$t5` zeigt jedoch auf, dass diese nicht immer übereinstimmen. Dies begründet sich in der Tatsache, dass die unteren 8 Bits von `a_wert1` bzw. `a_wert2` erst im Zusammenhang mit den oberen 24 Bit die gemeinte Zahl ergeben. Das jeweils unterste Byte enthält scheinbar gänzlich andere Vorzeicheninformationen, somit liefert der Zugriff mit dem Befehl `lb` ein falsches Ergebnis.

b) Wenn eine Zahl, deren Darstellung mehr Bits benötigt, als der Speicherbefehl bewegen kann, so werden nur die niederwertigen Bits in den Datenspeicher geschrieben. Ersetzt man in obigem Programm die Zahl 20 durch 70000 (=`0x00011170`), so ist weder der `sh`- noch der `sb`-Befehl in der Lage, diese Zahl, die min. 17 Bits zur Darstellung benötigt, korrekt in den Speicher zu schieben. Im Ergebnis steht unter

`b_halbwort` dann `0x1170`=4464 und unter `b_bytes 0x70`=112.
Ein Zugriff mittels `sw` auf eine Speicherzelle, die im Datenspeicher über eine `.byte`-Anweisung bereitgestellt wurde, ändert den dortigen Speicherinhalt nicht.

Aufgabe 27

Ich habe mich bemüht, den Quellcode derart ausführlich zu kommentieren, dass ich ihn einfach unverändert abdrucke:

```
# Aufgabe 27: Berechnung der Fakultaet
#
# von: Stephan Brumme
#
# zuletzt geaendert: 24.Mai 2000

       .data

request:.asciiz       "\nBitte geben Sie die Zahl ein, von der die Fakultaet ermittelt werden
soll: "
result1:.asciiz "Die Fakultaet von "
result2:.asciiz " ist "
result3:.asciiz "!.\n"


       .text
main:
       la     $a0,request           # Eingabeaufforderung
       li     $v0,4
       syscall

       li     $v0,5                 # n einlesen
       syscall

       # Hier beginnt jetzt die eigentliche Berechnung der Fakultaet

       # Grundlage ist die Formel n!=1*2*3*...*n
       # In einer Schleife wird $t0 als Zaehler von 1 bis n durchlaufen
       # und zu einem temporaeren Zwischenspeicher $t1 multipliziert
       # d.h. $t1=1*2*3*...*$t0=$t0!  (mit $t0<=n)
       # wenn $t0=n erreicht ist, dann ist $t1=n!
       # (n steht im Register $t2)

       # in Pseudocode:
       #    $t0=$t1=1;
       #    while ($t0<=n)
       #    {
       #       $t1*=$t0;
       #       $t0++;
       #    }

       move   $t2,$v0
       li     $t1,1                 # 0!=1
       li     $t0,1                 # bei 1! anfangen
loop:
       bgt    $t0,$t2,done          # inverse Bedingung von while fuehrt zum Beenden der
Schleife
       mul    $t1,$t1,$t0           # $t1*=$t0
       addi   $t0,$t0,1             # $t0++
       j      loop                  # naechster Schleifendurchlauf

done:
       la     $a0,result1           # Ausgabe "Die Fakultaet von "
       li     $v0,4
       syscall

       move   $a0,$t2               # Ausgabe n
       li     $v0,1
       syscall

       la     $a0,result2           # Ausgabe " ist "
       li     $v0,4
       syscall
```

```
        move    $a0,$t1             # Ausgabe n!
        li      $v0,1
        syscall

        la      $a0,result3         # Ausgabe "!."
        li      $v0,4
        syscall


        li      $v0,10              # Programm beenden
        syscall
```

Aufgabe 28

Wiederum ist Quellcode ausführlich kommentiert, so dass er unverändert abgedruckt wird:

```
# Aufgabe 28: Notendatenbank
#
# von: Stephan Brumme
#
# zuletzt geaendert: 24.Mai 2000


# ############################################################################################ #
# Ich gehe davon aus, dass der Benutzer nur gueltige Punktzahlen eingibt, d.h. er weder #
# negative Werte noch Zahlen >70 verwendet.                                                    #
# Vor- und Nachname sind jeweils auf 31 Zeichen begrenzt                                       #
# ############################################################################################ #


        .data

# Eingabeaufforderungen
ask_firstname: .asciiz "\nVorname: "
ask_familyname:       .asciiz "\nNachname: "
ask_points:    .asciiz "\nErreichte Punktzahl "

# Ausgabetexte
print_name:    .asciiz "\nStudent(in) "
print_space:   .asciiz " "
print_points:  .asciiz " erreichte mit "
print_mark:    .asciiz " Punkten die Note "

# Notenbezeichnungen
mark_sehrgut:         .asciiz "'sehr gut'"
mark_gut:             .asciiz "'gut'"
mark_befriedigend:    .asciiz "'befriedigend'"
mark_ausreichend:     .asciiz "'ausreichend'"
mark_ungenuegend:     .asciiz "'ungenuegend'"
mark_nochmal:         .asciiz "'noch einmal bitte'"

# speichert die Daten
# Groesse ermittelt sich mittels:
# Anzahl_Studenten*(Laenge_Vorname+Laenge_Nachname+Punkte_32Bit)
              .align 2              # sonst gibt Fehlermeldung wegen Misalignment
database:     .space 272            # 4*(32+32+4)=4*68



# ############################################################################

        .text
main:
        la      $a0,database          # Startadresse der Datenbank laden

        # das Unterprogramm bewegt automatisch $a0 auf den naechsten Datensatz weiter
        jal     ask                   # ersten Studenten in Datenbank eintragen
        jal     ask                   # zweiter
        jal     ask                   # dritter
        jal     ask                   # vierter
```

```
        la      $a0,database

        # das Unterprogramm bewegt automatisch $a0 auf den naechsten Datensatz weiter
        jal     print                   # ersten Studenten auswerten und ausgeben
        jal     print                   # zweiter
        jal     print                   # dritter
        jal     print                   # vierter


        li      $v0,10                  # Programm beenden
        syscall


# ##############################################################################################
# ASK:
# Unterprogramm, das Vornamen, Nachnamen und erreichte Punktzahl eines Studenten einliest
# Parameter: $a0 = Adresse, wo die Daten gespeichert werden
#                  Zeiger $a0 wird weiterbewegt, zeigt am Ende des UP auf naechste
#                  freie Speicherzelle
ask:
        move    $t0,$a0                 # $t0 ist jetzt der Zeiger auf die Speicherzelle,
                                        # wo die die Daten eingetragen werden können, da
                                        # $a0 stets anderweitig benutzt werden muss

        la      $a0,ask_firstname       # nach Vornamen fragen
        li      $v0,4
        syscall

        move    $a0,$t0                 # Vornamen einlesen
        li      $a1,32
        li      $v0,8
        syscall

        add     $t0,$t0,32              # Zeiger um 32 Bytes weiterbewegen

        la      $a0,ask_familyname      # nach Nachnamen fragen
        li      $v0,4
        syscall

        move    $a0,$t0                 # Nachnamen einlesen
        li      $a1,32
        li      $v0,8
        syscall

        add     $t0,$t0,32              # Zeiger um 32 Bytes weiterbewegen

        la      $a0,ask_points          # nach Punktzahl fragen
        li      $v0,4
        syscall

        li      $v0,5                   # Punktzahl einlesen
        syscall
        sw      $v0,0($t0)              # und im Datensatz abspeichern
        add     $t0,$t0,4               # Zeiger um 4 Bytes (=32 Bit) weiterbewegen

        move    $a0,$t0                 # $a0 enthaelt wieder den Zeiger

        jr      $ra                     # UP verlassen


# ##############################################################################################
# PRINT:
# Unterprogramm, das fuer einen Studenten die Punktzahl auf dem Bildschirm auswertet
# und einer Note zuordnet
# Parameter: $a0 = Adresse des Datensatzes fuer den betreffenden Studenten
#                  Zeiger $a0 wird weiterbewegt, zeigt am Ende des UP auf naechsten
#                  Datensatz
print:
        move    $t0,$a0                 # $t0 ist jetzt der Zeiger auf die Speicherzelle,
                                        # wo die die Daten eingetragen werden können, da
                                        # $a0 stets anderweitig benutzt werden muss

        la      $a0,print_name          # einleitenden Satz ausgeben: "\nStudent(in) "
        li      $v0,4
```

```
        syscall

        move    $a0,$t0              # Vornamen ausgeben
        li      $v0,4
        syscall

        add     $t0,$t0,32           # Zeiger um 32 Bytes auf Nachnamen weiterbewegen

        la      $a0,print_space      # Leerzeichen ausgeben
        li      $v0,4
        syscall

        move    $a0,$t0              # Nachnamen ausgeben
        li      $v0,4
        syscall

        add     $t0,$t0,32           # Zeiger um 32 Bytes auf Punktzahl weiterbewegen

        la      $a0,print_points     # ueberleitende Worte ausgeben: " erreichte mit "
        li      $v0,4
        syscall

        lw      $t1,($t0)            # Punktzahl laden
        add     $t0,$t0,4            # Zeiger um 4 Bytes (=32 Bit) auf naechsten Datensatz
weiterbewegen

        move    $a0,$t1              # Punktzahl ausgeben
        li      $v0,1
        syscall

        la      $a0,print_mark       # ueberleitende Worte ausgeben: " Punkten die Note "
        li      $v0,4
        syscall


        li      $t2,19               # if ($t1<=19)
        ble     $t1,$t2,print_0_19   #    goto print_0_19
        li      $t2,29               # if ($t1<=29)
        ble     $t1,$t2,print_20_29  #    goto print_20_29
        li      $t2,39               # if ($t1<=39)
        ble     $t1,$t2,print_30_39  #    goto print_30_39
        li      $t2,49               # if ($t1<=49)
        ble     $t1,$t2,print_40_49  #    goto print_40_49
        li      $t2,59               # if ($t1<=59)
        ble     $t1,$t2,print_50_59  #    goto print_50_59

                                     # jetzt muss $t1>=60 sein
print_60_70:
        la      $a0,mark_sehrgut     # Adresse von 'sehr gut' laden
        j       print_printmark
print_50_59:
        la      $a0,mark_gut         # Adresse von 'gut' laden
        j       print_printmark
print_40_49:
        la      $a0,mark_befriedigend # Adresse von 'befriedigend' laden
        j       print_printmark
print_30_39:
        la      $a0,mark_ausreichend  # Adresse von 'ausreichend' laden
        j       print_printmark
print_20_29:
        la      $a0,mark_ungenuegend  # Adresse von 'ungenuegend' laden
        j       print_printmark
print_0_19:
        la      $a0,mark_nochmal     # Adresse von 'noch einmal bitte' laden

print_printmark:
        li      $v0,4                # geladene Zeichenkette (d.h. Note) ausgeben
        syscall

        move    $a0,$t0              # $a0 enthaelt wieder den Zeiger

        jr      $ra                  # UP verlassen


# That's all folks.
```