Eine Anmerkung vorweg:
Die Programme auf der Diskette sind entsprechend den Aufgabennummern benannt. Für Aufgabe 17 ist dies die Datei „Aufgabe17.s“. Alle Dateien befindet sich im Stammverzeichnis und sind mit der Windowsversion von PC-SPIM bearbeitet worden.
Unter „RechnerarchitekturX.doc“, wobei X für die jeweilige Übungsblattnummer steht, findet sich dieser Text als MS-Word-Dokument. Bei eventuellen Fragen oder Diskettenfehlern bin ich unter stbrumme@rz.uni-potsdam.de zu erreichen.

Aufgabe 17

Das Programm lädt in Register $2 die Adresse des Datenwortes *mem2*. Anschließend werden die Register $3 bis $9 in die Speicherzellen der Datenworte *mem3* bis *mem9* geschrieben, wobei deren alter Inhalt verloren geht. Nach drei No-Operation-Befehlen, die den Zustand des Rechners (vom Befehlszähler abgesehen) nicht verändern, geht das Programm in eine Endlosschleife über. Der letzte Befehl, erneut NOP, wird daher nie ausgeführt.
Das erste Word befindet sich unter dem Namen *mem2* an der Adresse 0x10010000 im Datenspeicher. 4 Bytes dahinter liegt *mem3*, weitere 4 Bytes darauf folgt *mem4* usw. . Die Adresse von *mem6* ist daher 0x10010010. Der im Programm verwendete Speicherzugriff ergibt sich durch die Benutzung der Basisadresse 0x1001000 (entspricht *mem2*) und einer Konstanten als Offset. Somit sind für den Simulator die Benennungen *mem3* bis *mem9* überflüssig, sie dienen höchstens der besseren Lesbarkeit für den Programmierer. Problematisch wird die Angelegenheit, wenn es, aus welchen Gründen auch immer, notwendig ist, eine Datenspeicherzelle *mem3b* einzuführen, die genau zwischen *mem3* und *mem4* liegen soll (die Zahlen 3,3b und 4 sind hier nur beispielhaft verwendet worden und können beliebig ersetzt werden). Die Offsetmethode macht es nun notwendig, alle Befehle ab `sw $4,8($2)` umzuschreiben, da sie sonst auf falsche Speicherzelle verweisen. Hätte man stattdessen jedesmal die Befehlskombination `la $4,mem4; sw $4,0($2)` verwendet, könnte man einfach die entsprechenden Zeilen für *mem3b* in den Quellcode einfügen, ohne sich um die anderen Speicherzugriffe noch extra kümmern zu müssen. Allerdings ist diese zweite, von mir vorgeschlagene Methode Code-intensiver, da sie doppelt soviele Befehle benötigt.
Als dritte Möglichkeit könnte $2 nach jedem Speicherzugriff um 4 erhöht. In der Adreßermittlungsformel wird damit aus Adresse=Basis+(Offset+4) dann Adresse=(Basis+4)+Offset, was nach dem Assoziativgesetz der Addition äquivalent ist.

```
# Aufgabe 17
#
# von: Stephan Brumme, nach Vorlage
## zuletzt geaendert: 13.Mai 2000


    .text 0x400000

main:
    la      $2, mem2
    sw      $3, 4($2)
    sw      $4, 8($2)
    sw      $5,12($2)
    sw      $6,16($2)
    sw      $7,20($2)
    sw      $8,24($2)
    sw      $9,28($2)

    nop
    nop
    nop

ende:
    j       ende
    nop


    .data

mem2:   .word 4
mem3:   .word 8
mem4:   .word 16
mem5:   .word 32
mem6:   .word 64
```

```
mem7:   .word 128
mem8:   .word 256
mem9:   .word 512
```

Aufgabe 18

Ich habe die Aufgabenteile a und b in einem Programm zusammengefaßt, die umfangreichen Kommentare sollten zur Programmerklärung ausreichen. Im Schriftstil Arial (unterstrichen) habe ich lediglich kurz den jeweiligen Anfang der Aufgabenteile markiert:

```
# Aufgabe 18
#
# von: Stephan Brumme
#
# zuletzt geaendert: 12.Mai 2000


        .text 0x400000
        .globl main

main:
```

Aufgabe a)

```
        la      $s1,int             # Speicherplatz fuer eingegebene Integers
        li      $s0,5               # Anzahl einzugebender Integerwerte

loop:                               # in Schleife 5 ints einlesen
        la      $a0,request_int             # Eingabeaufforderung Integer
        li      $v0,4
        syscall

        li      $v0,5               # Integer von Konsole einlesen
        syscall
        sw      $v0,0($s1)          # im Datensegment ablegen

        addiu   $s1,$s1,4           # auf naechste Speicherzelle im Datensegment zeigen
        addi    $s0,$s0,-1          # Zaehler $s0 um 1 dekrementieren
        bgtz    $s0,loop            # wenn $s0>0 dann noch nicht fertig
```

Aufgabe b)

```
        la      $a0,request_str     # Eingabeaufforderung Zeichenkette
        li      $v0,4
        syscall

        la      $a0,string1         # String einlesen und bei string1 abspeichern
        li      $a1,21
        li      $v0,8
        syscall

        la      $a0,request_str     # Eingabeaufforderung Zeichenkette
        li      $v0,4
        syscall

        la      $a0,string2         # String einlesen und bei string1 abspeichern
        li      $a1,21
        li      $v0,8
        syscall


        li      $v0,10              # Programm beenden
        syscall


        .data

# Eingabeaufforderungen
request_int:   .asciiz "\nBitte geben Sie eine Integerzahl ein: "
request_str:   .asciiz "\nBitte geben Sie eine Zeichenkette ein: "

# Speicherplatz fuer die 5 Integerwerte
int:           .word 0,0,0,0,0
```

```
# Zeichenketten speichern: je 20 Zeichen + 1 Zeichen fuer Nullterminierung
string1:       .space 21
string2:       .space 21
```

Aufgabe 19

Wiederum habe ich alle drei Teilaufgaben zusammen in einem lauffähigen Programm gelöst:

```
# Aufgabe 19
#
# von: Stephan Brumme
#
# zuletzt geaendert: 13.Mai 2000


       .text 0x00400000
       .globl main

main:
```

Aufgabe a)

```
       li     $a0,1                # Fall 1 pruefen
       beq    t0_1                 # wenn er zutrifft, entsprechende Ausgabe vorbereiten
       li     $a0,2                # Fall 2 pruefen
       beq    t0_2                 # wenn er zutrifft, entsprechende Ausgabe vorbereiten
       li     $a0,3                # Fall 3 pruefen
       beq    t0_3                 # wenn er zutrifft, entsprechende Ausgabe vorbereiten
       li     $a0,4                # Fall 4 pruefen
       beq    t0_4                 # wenn er zutrifft, entsprechende Ausgabe vorbereiten

       la     $a0,str_t0_sonst     # Fall "sonst" ist eingetreten, Zeichenkette "sonst"
laden
       j      a_done               # zur Zeichenkettenausgabe springen

t0_1:
       la     $a0,str_t0_1         # Zeichenkette "Fall 1" laden
       j      a_done               # zur Zeichenkettenausgabe springen
t0_2:
       la     $a0,str_t0_2         # Zeichenkette "Fall 2" laden
       j      a_done               # zur Zeichenkettenausgabe springen
t0_3:
       la     $a0,str_t0_3         # Zeichenkette "Fall 3" laden
       j      a_done               # zur Zeichenkettenausgabe springen
t0_4:
       la     $a0,str_t0_4         # Zeichenkette "Fall 4" laden

a_done:
       li     $v0,4                # die jeweilige Zeichenkette ausgeben
       syscall
```

Aufgabe b)

```
       li     $a0,27               # mit 27 vergleichen
       bne    $t0,$a0,t0_not27     # ist $t0=27 ? wenn nicht, dann zu t0_not27 springen

       la     $a0,str_t0_27        # $t0 ist 27, entsprechende Zeichenkette laden
       j      b_done               # zur Zeichenkettenausgabe springen

t0_not27:
       la     $a0,str_t0_not27     # $t0 ist ungleich 27, entsprechende Zeichenkette laden

b_done:
       li     $v0,4                # die jeweilige Zeichenkette ausgeben
       syscall
```

Aufgabe c)

```
       li     $t0,0                # k=0
       li     $t1,1                # i=1, Startwert der Schleife
       li     $t2,27               # i_stop=27, Abbruchbedingung der Schleife

loop:
       add    $t0,$t0,$t1          # k:=k+i
       addi   $t1,$t1,2            # i:=i+2
       ble    $t1,$t2              # ist Abbruchbedingung der Schleife erfuellt ?
```

```
                                        # wenn nein, dann naechster Schleifendurchlauf mittels
Sprung zu "loop"



        li      $v0,10                  # Programm beenden
        syscall



        .data

str_t0_1:       .asciiz "Erster Fall\n"
str_t0_2:       .asciiz "Zweiter Fall\n"
str_t0_3:       .asciiz "Dritter Fall\n"
str_t0_4:       .asciiz "Vierter Fall\n"
str_t0_sonst:   .asciiz "\n"
str_t0_27:      .asciiz "Register $t0 = 27\n"
str_t0_not27:   .asciiz "Register $t0 ist ungleich 27\n"
```

Aufgabe 20

In C++-ähnlichem Pseudocode sieht die Problemlösung folgendermaßen aus (ich gehe davon aus, dass der eingegebene Preis keine Pfennigbeträge enthält):

```
int Preis;
cout << "\nBitte geben Sie den Preis ein: DM "
cin >> Preis;
if (Preis>=10)
{
        if (Preis<50)
                Preis-=2;
        else
                Preis-=5;
}
cout << "\nDas Produkt kostet abzgl. Rabatt: DM " << Preis;
```

In Assembler kann man diese Vorgehensweise fast 1:1 umsetzen, lediglich sind mehrere Sprünge und „Hilfsregister“ für die `if`-Anweisungen einzubauen, die in C++ der Compiler automatisch generiert.

```
# Aufgabe 20
#
# von: Stephan Brumme
#
# zuletzt geaendert: 12.Mai 2000


        .text 0x00400000
        .globl main

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

        li      $v0,5                   # Preis von Konsole einlesen
        syscall

        move    $s0,$v0                 # Preis in $s0 speichern

        li      $s1,10
        blt     $s0,$s1,done            # kein Rabatt bei weniger als 10 DM

        li      $s1,50
        bge     $s0,$s1,above50         # >=50 DM, verzweigen

        addi    $s0,$s0,-2              # 10<=x<50, also 3 DM Rabatt
        j       done                    # zur Ausgabe springen

above50:
        addi    $s0,$s0,-5              # 5 DM Rabatt abziehen
```

```
done:
        la      $a0,answer              # preiserklaerender Text
        li      $v0,4
        syscall

        move    $a0,$s0                 # rabattierter Preis
        li      $v0,1
        syscall


        li      $v0,10                  # Programm beenden
        syscall


        .data

# Eingabeaufforderung
request:        .asciiz "\nBitte geben Sie den Preis ein: DM "
answer:         .asciiz "\nDas Produkt kostet abzgl. Rabatt: DM "
```