Informatik 3 – Übung 03 – Georg Kuschk

3.1) MIPS

```
#Aufgabe 3.1.a)
#Georg Kuschk

.data

##### Text-Variablen initialisieren #####
str_txt1:   .asciiz     "Bitte Zahl eingeben : "
str_txt2:   .asciiz "Eingabeende"
str_txt3:   .asciiz     "\n\nAusgabe : "
str_txt4:   .asciiz " "

.text

main:
            ##### Initialisierung #####
            li          $s0,0                   #Counter fuer die Anzahl der eingegebenen Zahlen

input:
            ##### Eingabeanfang anzeigen #####
            li          $v0,04                  #print_string "Bitte Zahl eingeben : "
            la          $a0,str_txt1
            syscall


            ##### Zahl einlesen #####
            li          $v0,05                  #read_int
            syscall

            beqz        $v0,input_finished      #ist die eingegebene zahl =0 dann den Input beenden

            ##### ansonsten die eingelesene zahl auf den stack schreiben #####
            addi        $sp,-4                  #Stackpointer dekrementieren
            sw          $v0,4($sp)              #Zahl auf den stack schreiben
            addi        $s0,1                   #Counter inkrementieren

            b           input                   #naechste Zahl einlesen


input_finished:

            ##### Eingabeende anzeigen #####
            li          $v0,04                  #print_string "Eingabeende"
            la          $a0,str_txt2
            syscall

            ##### Ausgabeanfang anzeigen #####
            li          $v0,04                  #print_string "\n\nAusgabe : "
            la          $a0,str_txt3
            syscall

output:
            ##### Zahlen ausgeben #####

            li          $v0,01                  #print_int
            lw          $a0,4($sp)              #zahl vom Stack holen
            addi        $sp,4                   #Stackpointer aktualisieren
            syscall
            addi        $s0,-1                  #Counter dekrementieren

            li          $v0,04                  #die auszugebenden Zahlen durch ein Leerzeichen trennen
            la          $a0,str_txt4
            syscall

            bnez        $s0,output              #pruefen, ob schon alle Zahlen ausgegeben wurden
                                                #wenn nicht, dann die naechste Zahl ausgeben


            ##### auf Tastendruck zum Beenden warten #####
            li          $v0,05                  #read_int
            syscall


exit:
            li          $v0,10                  #exit program
            syscall
```

```
#################################################################################################
#################################################################################################
#################################################################################################
#Aufgabe 3.1.b)
#Georg Kuschk

.data

##### Text-Variablen initialisieren #####
str_txt1:   .asciiz     "Bitte ersten  Operanden eingeben : "
str_txt2:   .asciiz     "\nBitte zweiten Operanden eingeben : "
str_txt3:   .asciiz "\nArt der Operation definieren {+,-,*,/} : "

str_txt4:   .asciiz "\nErgebnis : "
str_txt5:   .asciiz "\nUngueltiges Operationssymbol!\n"
str_txt6:   .asciiz "\nKann Division durch 0 nicht ausfuehren!\n"

str_input:  .space 64                               #Speicher fuer die Eingabe des Operationssysmbols

.text

main:

            ##### Ersten Operanden einlesen #####
            li      $v0,04                          #print_string "Bitte ersten  Operanden eingeben : "
            la      $a0,str_txt1
            syscall

            li      $v0,05                          #read_int
            syscall
            move    $t0,$v0                         #ersten Operanden in $t0 speichern

next_operation:
            ##### Zweiten Operanden einlesen #####
            li      $v0,04                          #print_string "Bitte zweiten Operanden eingeben : "
            la      $a0,str_txt2
            syscall

            li      $v0,05                          #read_int
            syscall
            move    $t1,$v0                         #zweiten Operanden in $t1 speichern



            ##### Operations-Art einlesen #####
            li      $v0,04                          #print_string "Art der Operation definieren {+,-,*,/} : "
            la      $a0,str_txt3
            syscall


            li      $v0,08                          #read_string - EingabeString in str_input speichern
            la      $a0,str_input                   #Adresse des EingabeStrings laden
            li      $a1,64                          #maximale Laenge des Eingabestrings
            syscall

            lb      $t2,str_input($zero)            #nur erstes Zeichen des eingegebenen Strings auswerten

            beq     $t2,'+',addition                # + -> springe zur Addition
            beq     $t2,'-',subtraction             # - -> springe zur Subtraktion
            beq     $t2,'*',multiplication          # * -> springe zur Multiplikation
            beq     $t2,'/',division                # / -> springe zur Division


            ##### Falls kein gueltiges Operationssymbol eingegeben wurde, dies anzeigen und fortfahren #####
            li      $v0,04                          #print_string "\nUngueltiges Operationssymbol!\n"
            la      $a0,str_txt5
            syscall
            j       output                          #mit dem Zwischenergebnis in $t0 die naechste Operation durchfuehren



addition:
            add     $t0,$t0,$t1                     #Addition ueber den zwei Operanden ausfuehren
            j       output                          #zur Ergebnisausgabe springen
subtraction:
            sub     $t0,$t0,$t1                     #Subtraktion ueber den zwei Operanden ausfuehren
            j       output                          #zur Ergebnisausgabe springen
multiplication:
            mul     $t0,$t0,$t1                     #Multiplikation ueber den zwei Operanden ausfuehren
            j       output                          #zur Ergebnisausgabe springen
division:
            beqz    $t1,divnull                     #Division durch Null abfangen

            div     $t0,$t0,$t1                     #wenn Divisor ungleich Null, dann Ergebnis berechnen
            j       output                          #zur Ergebnisausgabe springen

divnull:                                            #Division durch Null anzeigen
            li      $v0,04                          #print_string "\nKann Division durch 0 nicht ausfuehren!\n"
            la      $a0,str_txt6
            syscall
            j       next_operation                  #den zweiten Operanden nochmal neu eingeben lassen
```

```
output:
        ##### Ergebnis anzeigen #####
        li      $v0,04                  #print_string "\nErgebnis : "
        la      $a0,str_txt4
        syscall

        li      $v0,01                  #print_int    (in $t0 steht das Ergebnis der Operation)
        move    $a0,$t0
        syscall

        ##### und Ergebnis auf den Stack schreiben #####
        addi    $sp,-4                  #Stackpointer dekrementieren
        sw      $t0,4($sp)              #Ergebnis auf den stack schreiben

        j       next_operation          #und wieder einen neuen Operanden fuer die naechste Operation einlesen




##### ueberfluessig, da programm nicht terminiert #####

        ##### auf Tastendruck zum Beenden warten #####
#       li      $v0,05                  #read_int
#       syscall

#exit:
#       li      $v0,10                  #exit program
#       syscall
```

3.2) Adressierungsarten :

immediate : Bei der unmittelbaren Adressierung (immediate) ist kein Speicherzugriff erforderlich. Der zu ladende Wert wird im Befehl explizit angegeben.

Bsp: li $1 , 20 ($1 := 20;)
(lädt den Integer-Wert 20 in das Register $1)

direkt : Bei der direkten Adressierung wird die Adresse der Speicherzelle als Operand angegeben, in der sich der zu ladende Wert befindet

Bsp: lw $1, 20 ($1 := mem[20];)
(liest den Wert der Speicherzelle 20 und schreibt diesen in das Register $1)

indirekt : Bei der indirekten Adressierung steht die Adresse der Speicherzelle mit dem zu ladenden Wert in dem Register, welches als Operand angegeben ist

Bsp: lw $a0 , 4($sp) ($1 := mem[($sp)])
(die „obersten“ 4Bytes des Stacks stellen die Adresse der Speicherzelle dar, in der sich der zu ladende Wert befindet)