Informatik 3 – Übung 06 – Georg Kuschk

6.3) MIPS

```
#Aufgabe 6.3)
#Georg Kuschk

#ACHTUNG :      Da laut Forum davon ausgegangen werden soll, dass der Eingabewert,
#               falls er denn kleiner gleich 10 Stellen         besitzt,
#               mit maximal 32Bit darstellbar ist, wird nicht noch extra ueberprueft
#               ob der Eingabewert zwischen 4294967296 und 9999999999 liegt
#
#               Ebenso werden eventuell fuehrende Nullen nicht entfernt, sonder als
#               Dezimalstelle mitgezaehlt


#               Belegung der globalen Variablen :
#               $s1     :       die Anzahl der auf dem Stack befindlichen Zahlen


.data
str_txt1:   .asciiz     "\nEingabe : "
str_txt2:   .asciiz "\nFehler! - Keine gueltige Eingabe !"
str_txt3:   .asciiz "\nProgrammende - letztes Rechenergebnis : "
str_txt4:   .asciiz "Noch keine Zahl auf den Stack geschrieben !"
str_txt5:   .asciiz "\nKeine 2 Operanden fuer die Operation auf dem Stack vorhanden !"
str_txt6:   .asciiz "\nKann Division durch Null nicht ausfuehren !"
str_txt7:   .asciiz "\nErgebnis der Operation : "
str_txt8:   .asciiz "\nWeniger als 2 Code-Worte auf dem Stack !"
str_txt9:   .asciiz "\nDistanz des Codes : "

str_input: .space   64


.text


main:
        ########################## Initialisierung der globalen Variablen ###################
        li      $s1,0               #in $s1 steht die Anzahl der auf dem Stack befindlichen Zahlen



loop_input:
        li      $v0,04              #print_string "Eingabe : "
        la      $a0,str_txt1
        syscall




        ########################## Zeichenkette einlesen ###################################
        li      $v0,08              #read_string - EingabeString in str_input einlesen
        la      $a0,str_input       #Adresse des EingabeStrings laden
        li      $a1,64              #maximale Laenge des EingabeStrings
        syscall                     #schliesslich die Funktion (read_string) aufrufen


        ########################## Zeichenkette auswerten ##################################

        ##### Prüfen, ob 'q' eingegeben wurde #####
        li      $t1,0
        li      $s0,1
        lb      $t1,str_input($s0)          #pruefen, ob die Zeichenkette nur ein Zeichen enthaelt (2.Byte->Wert=10)
        bne     $t1,10,check_number         #wenn nicht, dann nicht als Operationssymbol erkennen

        lb      $t1,str_input($zero)        #1. Zeichen der Zeichenkette in $t1 laden und auswerten
        bne     $t1,'q',check_distance      #wenn das Zeichen ungleich 'q' ist, dann als Nächstes
                                            #prüfen, ob das Zeichen ='h' ist

                                            #wenn Zeichen ='q' :
        li      $v0,04                      #print_string "\nProgrammende - letztes Rechenergebnis : "
        la      $a0,str_txt3
        syscall

        bne     $s1,$zero,stack_print       #befinden sich auf dem Stack keine eingegebenen Zahlen
                                            #oder Rechenergebnisse so wird eine Meldung angezeigt,
```

```
                                        #ansonsten wird der letzte Wert vom Stack geholt und ausgegeben

        li      $v0,04                  #print_string "Noch keine Zahl auf den Stack geschrieben !"
        la      $a0,str_txt4
        syscall
        j       exit_prog               #und zum Programm-Ende springen


        ###Falls sich auf dem Stack Werte befand, so wird der "oberste" hier ausgegeben
stack_print:
        li      $v0,01                  #print_int
        lw      $a0,4($sp)              #letzten Wert vom Stack holen
        addi    $sp,4                   #Stackpointer aktualisieren
        syscall
        addi    $s1,-1                  #Zahlen-Counter dekrementieren
        j       exit_prog               #und zum Programm-Ende springen









check_distance:
        ##### Prüfen, ob 'h' eingegeben wurde #####
        bne     $t1,'h',check_symbol    #1. Zeichen der Zeichenkette befindet sich noch in $t1
                                        #wenn das Zeichen ungleich 'h' ist, dann als Nächstes
                                        #prüfen, ob ein Operationssymbol vorliegt

        blt     $s1,2,no_words          #Prüfen, ob mindestens zwei Worte auf dem Stack liegen

        #######  An dieser Stelle ist die Eingabe 'h' ; also die Distanz des Codes berechnen #####
        #######  Die Codewoerter w1 und w2, deren Distanz zu berechnen ist, werden in t3 und t4 stehen

        #######  weitere Variablen :
        #######  $s1     :       die Anzahl der auf dem Stack befindlichen Zahlen
        #######  $s2     :       maximale Distanz zweier Codewoerter
        #######  $s3     :       gibt an, das wievielte Wort auf dem Stack gerade mit den anderen Woerten
        #######                  "unter" ihm verglichen wird = Nr von w1
        #######  $s4     :       gibt an, mit dem wievielten Wort auf dem Stack  $s3 gerade
        #######                  verglichen wird = Nr von w2

        li      $s2,0
        li      $s3,1


dist_outer_loop:
        sll     $t5,$s3,2               #berechnen der Stack-Adresse des Codewortes w1 (zwischenspeichern in $t5)
        add     $sp,$sp,$t5             #Stackpointer zu w1 verschieben
        lw      $t3,($sp)               #Codewort w1 vom Stack lesen und in $t3 speichern
        sub     $sp,$sp,$t5             #Stackpointer wieder auf das erste Stack-Element zurueckverschieben

        add     $s4,$s3,1               #w2 liegt ein Wort tiefer im Stack

start_dist_inner_loop:
        sll     $t6,$s4,2               #berechnen der Stack-Adresse des Codewortes w2 (zwischenspeichern in $t6)
        add     $sp,$sp,$t6             #Stackpointer zu w2 verschieben
        lw      $t4,($sp)               #Codewort w2 vom Stack lesen und in $t4 speichern
        sub     $sp,$sp,$t6             #Stackpointer wieder auf das erste Stack-Element zurueckverschieben

        li      $t0,0                   #$t0 gibt an, das wievielte Bit gerade verglichen wird
        li      $t1,1                   #$t1 besitzt die Wertigkeit des gerade zu vergleichenden Bits (=2^$t0)
        li      $t2,0                   #$t2 zaehlt die Distanz der beiden aktuellen Codewoerter


dist_inner_loop:
        and     $t5,$t3,$t1             #pruefen, ob das aktuelle Bit im 1.Codewort gesetzt ist
        and     $t6,$t4,$t1             #pruefen, ob das aktuelle Bit im 2.Codewort gesetzt ist

        beq     $t5,$t6,bit_equal       #pruefen, ob die beiden obigen Befehle das selbe Ergebnis lieferten
                                        #d.h. ob das aktuelle Bit in beiden Codewoertern gleich ist

        add     $t2,$t2,1               #wenn nicht, dann die Distanz um 1 erhoehen

bit_equal:
        beq     $t0,31,end_inner_loop   #wenn gerade das letzte (das 32.) Bit geprueft wurde, den Vergleich beenden

        add     $t0,$t0,1               #ansonsten den Bearbeitungs-Zeiger auf das naechsthoehere Bit verschieben
        sll     $t1,$t1,1               #naechste 2er-Potenz durch shift-left berechnen
        j       dist_inner_loop         #und die Schleife mit dem naechsten Bit neu durchlaufen
```

```
end_inner_loop:
        ##### die Distanz der beiden Code-Woerter steht nun in $t2

        ble     $t2,$s2,no_max_dist     #ist die Distanz >= der bisherigen maximalen Distanz,
        add     $s2,$t2,0               #dann die soeben berechnete Distanz als die maximale Distanz setzen
no_max_dist:



        beq     $s4,$s1,w2_is_last      #pruefen, ob w2 das letzte Stack-Element ist
        add     $s4,$s4,1               #wenn nicht, dann ist w2 nun ein Wort tiefer im Stack
        j       start_dist_inner_loop   #und wieder die Distanz vom neuen w2 mit w1 bestimmen
w2_is_last:



        add     $s3,$s3,1
        sne     $t0,$s3,$s1             #pruefen, ob w1 das vorletzte Stack-Element ist
        sub     $s3,$s3,1

        beq     $t0,$zero,w1_is_last    #wenn ja, dann wurden alle noetigen n(n-1)/2 Vergleiche durchgefuehrt
                                        #und die maximale Distanz zwischen 2 Code-Woertern wurde bestimmt

        add     $s3,$s3,1               #wenn nicht, dann ist w1 nun ein Wort tiefer im Stack
        j       dist_outer_loop         #und wieder die Distanz vom neuen w1 mit allen
                                        #tiefer im Stack liegenden Code-Woertern bestimmen
w1_is_last:

        ##### Distanz des Codes befindet sich nun in $s2
        ##### Diese nun auf die Konsole ausgeben

        li      $v0,04                  #print_string "\nDistanz des Codes : "
        la      $a0,str_txt9
        syscall

        li      $v0,01                  #print_int (maximale Distanz zwischen 2 Code-Woertern)
        add     $a0,$s2,0
        syscall

        sll     $s1,$s1,2               #wie in der Aufgabenstellung gefordert, nun den Stack leeren
        add     $sp,$sp,$s1
        li      $s1,0                   #Anzahl der eingegebenen Werte auf 0 setzen

        j       loop_input              #und wieder neue Zeichenkette erfragen

no_words:
        li      $v0,04                  #print_string "\nWeniger als 2 Code-Worte auf dem Stack !"
        la      $a0,str_txt8
        syscall
        j       loop_input              #und wieder neue Zeichenkette erfragen









check_symbol:
        ##### Prüfen, ob ein Operationssymbol +,-,*,/ eingegeben wurde #####
        lb      $t1,str_input($zero)    #1. Zeichen der Zeichenkette in $t1 laden und auswerten

        lb      $t1,str_input($zero)    #1. Zeichen der Zeichenkette in $t1 laden und auswerten

        beq     $t1,'+',addition        # + -> springe zur Addition
        beq     $t1,'-',subtraction     # - -> springe zur Subtraktion
        beq     $t1,'*',multiplication  # * -> springe zur Multiplikation
        beq     $t1,'/',division        # / -> springe zur Division


        j       check_number            #wurde kein gueltiges Operationssymbol eingegeben, pruefen,
                                        #ob die Zeichenkette eine Zahl ist
```

```
addition:
        blt     $s1,2,no_ops            #Prüfen, ob mindestens zwei Operanden auf dem Stack liegen
        lw      $t0,8($sp)              #1.Operanden vom Stack holen holen
        lw      $t1,4($sp)              #2.Operanden vom Stack holen holen
        addi    $sp,8                   #Stackpointer aktualisieren
        addi    $s1,-2                  #Zahlen-Counter um die zwei soeben gelesenen Werte verringern
        add     $t0,$t0,$t1             #Addition ueber den zwei Operanden ausfuehren
        j       op_success              #Ergebnis berechnet
subtraction:
        blt     $s1,2,no_ops            #Prüfen, ob mindest ens zwei Operanden auf dem Stack liegen
        lw      $t0,8($sp)              #1.Operanden vom Stack holen holen
        lw      $t1,4($sp)              #2.Operanden vom Stack holen holen
        addi    $sp,8                   #Stackpointer aktualisieren
        addi    $s1,-2                  #Zahlen-Counter um die zwei soeben gelesenen Werte verringern
        sub     $t0,$t0,$t1             #Subtraktion ueber den zwei Operanden ausfuehren
        j       op_success              #Ergebnis berechnet
multiplication:
        blt     $s1,2,no_ops            #Prüfen, ob mindestens zwei Operanden auf dem Stack liegen
        lw      $t0,8($sp)              #1.Operanden vom Stack holen holen
        lw      $t1,4($sp)              #2.Operanden vom Stack holen holen
        addi    $sp,8                   #Stackpointer aktualisieren
        addi    $s1,-2                  #Zahlen-Counter um die zwei soeben gelesenen Werte verringern
        mul     $t0,$t0,$t1             #Multiplikation ueber den zwei Operanden ausfuehren
        j       op_success              #Ergebnis berechnet
division:
        blt     $s1,2,no_ops            #Prüfen, ob mindestens zwei Operanden auf dem Stack liegen
        lw      $t0,8($sp)              #1.Operanden vom Stack holen holen
        lw      $t1,4($sp)              #2.Operanden vom Stack holen holen
        addi    $sp,8                   #Stackpointer aktualisieren
        addi    $s1,-2                  #Zahlen-Counter um die zwei soeben gelesenen Werte verringern

        beqz    $t1,divnull             #Division durch Null abfangen
        div     $t0,$t0,$t1             #wenn Divisor ungleich Null, dann Ergebnis berechnen
        j       op_success              #Ergebnis berechnet
divnull:                                #Division durch Null anzeigen
        li      $v0,04                  #print_string "\nKann Division durch Null nicht ausfuehren !"
        la      $a0,str_txt6
        syscall
        j       loop_input              #bei Division durch Null gleich wieder neue Zeichenkette erfragen


op_success:
        ###bei erfolgreich ausgefuehrter Operation das Ergebnis (liegt in $t0) auf den Stack schreiben
        addi    $sp,-4                  #Stackpointer dekrementieren
        sw      $t0,4($sp)              #Ergebnis auf den Stack schreiben
        add     $s1,$s1,1               #Zahlen-Counter inkrementieren

        ###und auch noch auf der Konsole ausgeben
        li      $v0,04                  #print_string "\nErgebnis der Operation : "
        la      $a0,str_txt7
        syscall
        li      $v0,01                  #print_int
        add     $a0,$t0,0               #Ergebnis der letzten Rechnung der Funktion als Argument uebergeben
        syscall
        j       loop_input              #und wieder neue Zeichenkette erfragen

no_ops:
        li      $v0,04                  #print_string "\nKeine 2 Operanden fuer die Operation auf dem Stack vorhanden !"
        la      $a0,str_txt5
        syscall
        j       loop_input              #und wieder neue Zeichenkette erfragen
```

```
check_number:
        ##### Prüfen, ob die Eingabe eine gueltige, vorzeichenlose Dezim alzahl ist #####
        li      $s0,0                   #$s0 zaehlt die Anzahl der Ziffern
input_evaluation:
        lb      $t1,str_input($s0)      #Zeichen des eingegebenen Strings einzeln in $t1 laden und auswerten

        beq     $t1,10,input_end        #Ende des Eingabestrings erreicht?
        blt     $t1,'0',input_error     #wenn nicht : ist das Zeichen eine Ziffer ('0'-'9')?
        bgt     $t1,'9',input_error

        add     $s0,$s0,1               #Zeichen ist eine Ziffer -> Anzahl der Ziffern inkrementieren
                                        #und den Integer-Wert der Ziffer auf den Stack schreiben

        addi    $sp,-1                  #Stackpointer dekrem entieren
        sub     $t1,$t1,'0'             #das gelesene Zeichen in entsprechende Ziffer umrechnen
        sb      $t1,1($sp)              #und die Ziffer auf den Stack schreiben

        j       input_evaluation        #da das Ende des Eingabestrings noch nicht erreicht ist,
                                        #das naechste Zeichen auswerten

input_end:
        beq     $s0,$zero,input_error   #hat die Zahl 0 Stellen ? -> dann Fehler
        bgt     $s0,10,input_error      #hat die Zahl > 10 Stellen ? -> dann Fehler

        ##### EingabeString wurde jetzt akzeptiert -> Binaerzahl berechnen
        ##### zur Erinnerung : Anzahl der Ziffem der Dezimalzahl steht in $s0 und muss an dieser Stelle >0 sein
        ##### und die Ziffern stehen in umgekehrter Reihenfolge auf dem Stack

        li      $t0,0                   #das Gesamtergebnis wird zunaechst in $t0 zwischengespeichert
        li      $t1,1                   #in $t1 stehen die gerade benoetigten 10er-Potenzen
dec2bin_loop:
        li      $t2,0                   #$t2 fuer Aufnahme eines Bytes vorbereiten
        lb      $t2,1($sp)              #Ziffer vom Stack holen
        addi    $sp,1                   #Stackpointer aktualisieren

        mul     $t2,$t2,$t1             #Ziffern-Wertigkeit entsprechend der Dezimal-Stelle berechnen
        add     $t0,$t0,$t2             #und zum Gesamtergebnis hinzuaddieren

        mul     $t1,$t1,10              #naechste 10er-Potenz berechnen
        sub     $s0,$s0,1               #Anzahl der noch abzuarbeitenden Stellen dekrementieren
        bgt     $s0,$zero,dec2bin_loop  #und wenn diese >0 ist, mit dem Verfahren fortfahren


        ##### Die bisher von dem Programm-Teil "check_number" auf den Stack geschriebenen Ziffern wieder entfernen
        add     $sp,$sp,$s0             #Stackpointer wieder um die Anzahl der auf den Stack geschriebenen
                                        #Bytes erhoehen

        #Alle Ziffern wurden abgearbeitet , die Zahl ist erkannt und steht nun in $t0 -> Zahl auf den Stack schreiben
        addi    $sp,-4                  #Stackpointer dekrementieren
        sw      $t0,4($sp)              #Zahl auf den Stack schreiben
        add     $s1,$s1,1               #Zahlen-Counter inkrementieren

        j       loop_input              #und wieder neue Zeichenkette erfragen




##### Hier ist das Sprungziel des Programms fuer eine ungueltige Eingabe
input_error:
        li      $v0,04                  #print_string "\nFehler! - Keine gueltige Eingabe !"
        la      $a0,str_txt2
        syscall

        ##### Die bisher von dem Programm-Teil "check_number" auf den Stac k geschriebenen Ziffern wieder entfernen
        add     $sp,$sp,$s0             #Stackpointer wieder um die Anzahl der auf den Stack geschriebenen
                                        #Bytes erhoehen

        j       loop_input              #und wieder neue Zeichenkette erfragen




exit_prog:

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


        li      $v0,10          #exit
        syscall
```