Ludwig-Maximilians-Universität München
Institut für Informatik
Lehrstuhl für Mobile und Verteilte Systeme
Prof. Dr. Linnhoff-Popien

mobile and distributed systems group

Übungsblatt 13
Rechnerarchitektur im Sommersemester 2023

Besprechung: Dieses Übungsblatt dient zur Vorbereitung auf die Klausur und wird nicht besprochen.

Ankündigungen: Am Montag dem 17.07. findet ein Sondertutorium um 18-20 Uhr für alle statt, bei welchem noch einmal gezielt Fragen zum Stoff gestellt werden können.

Aufgabe Ü1: Einfachauswahlaufgabe: Wiederholung

Für jede der folgenden Fragen ist eine korrekte Antwort auszuwählen („1 aus n“). Nennen Sie dazu in Ihrer Abgabe die jeweils ausgewählte Antwortnummer ((i), (ii), (iii) oder (iv)). Eine korrekte Antwort ergibt jeweils einen Punkt. Mehrfache Antworten oder eine falsche Antwort werden mit 0 Punkten bewertet.

1) Welche Dualzahl entspricht dem hexadezimalen Wert `C9`?

(i) `10000001`	(ii) `11001001`	(iii) `10111111`	(iv) `10101010`

2) Wie lautet eine der De Morganschen Regeln?

(i) $\overline{(a+b)} = \overline{a} \cdot \overline{b}$	(ii) $a \cdot 0 = 0$	(iii) $a + \overline{a} = 1$	(iv) $a{\cdot}(b{+}c) = (a{\cdot}b){+}(a{\cdot}c)$

3) Wie lautet die Belegung von `$t2` nach Ausführung des folgenden SPIM-Codes?

```
.data
var: .word 8, 32, 17, 4, 9

.text
main: lw $t1, var
      lw $t2, var+4($t1)
```

(i) `8`	(ii) `32`	(iii) `12`	(iv) `4`

4) Sei folgende Wahrheitstafel einer Booleschen Funktion $f : B^2 \to B$ gegeben.
Welcher Ausdruck entspricht **nicht** dieser Funktion?

i	x_1	x_2	$f(x_1, x_2)$
0	0	0	1
1	0	1	0
2	1	0	1
3	1	1	1

(i) $f(x_1, x_2) = \overline{\overline{(x_1 \cdot x_1)} \cdot x_2}$	(ii) $f(x_1, x_2) = x_1 + \overline{x}_2$	(iii) $f(x_1, x_2) = \overline{(\overline{x}_1 \cdot x_2)}$	(iv) $f(x_1, x_2) = (x_1 + \overline{x}_2) \cdot (\overline{x}_1 + \overline{x}_2)$

5) Ein Carry-Save-Addiernetz...			
(i) ...berechnet Teilergebnisse doppelt und selektiert eines davon.	(ii) ...dient der schnellen Addition von mehr als zwei Summanden.	(iii) ...lässt den endgültigen Übertrag (von rechts nach links) durch das Schaltnetz rieseln.	(iv) ...führt die Additionsoperation auf die Subtraktion zurück.

6) Was ist kein Schritt im Pipelining einer klassischen RISC CPU?			
(i) Instruction Fetch	(ii) Instruction Decode	(iii) Erase	(iv) Execute (ALU)

7) Um welche Art von Hazards handelt es sich, wenn Entscheidungen (z.B. Sprünge) erst basierend auf einem späteren Ausführungsschritt oder dem noch ausstehenden Ergebnis einer vorangegangenen Instruktion getroffen werden können und dadurch das Pipelining erschwert wird?			
(i) Control Hazards	(ii) Biohazards	(iii) Data Hazards	(iv) Structural Hazards

8) Angenommen ein Decoder hat `4` Eingänge. Wie viele Ausgänge muss der Decoder zur vollständigen Umsetzung seiner Funktionalität besitzen?			
(i) `2`	(ii) `4`	(iii) `8`	(iv) `16`

9) Wie lautet die Belegung von `$t2` nach Ausführung des folgenden SPIM-Codes?

```
.data
var: .word 12, 31, 9, 5, 4

.text
main: lw $t1, var
      lw $t2, var($t1)
```

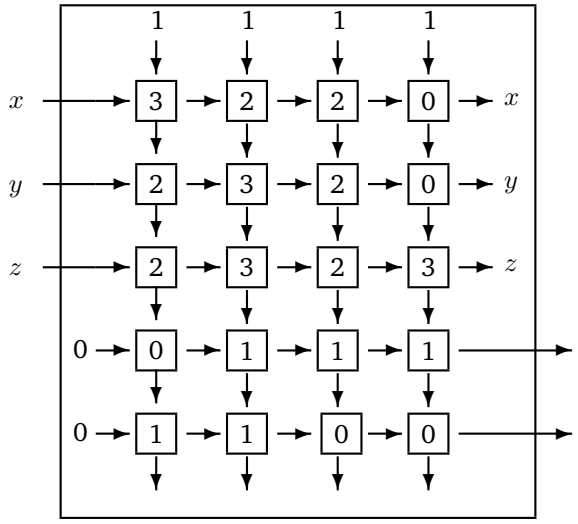
(i) `12`	(ii) `9`	(iii) `5`	(iv) `4`

10) Wie lautet das dezimale Ergebnis der Addition der folgenden in Zweierkomplementdarstellung (unter Verwendung von 8 Bit) gegebenen Binärzahlen?

	`00001111`
+	`00110001`
Übertrag	
Ergebnis	

(i) `110`	(ii) `64`	(iii) `32`	(iv) `-19`

11) Welche Boolesche Funktion realisiert folgendes PLA?



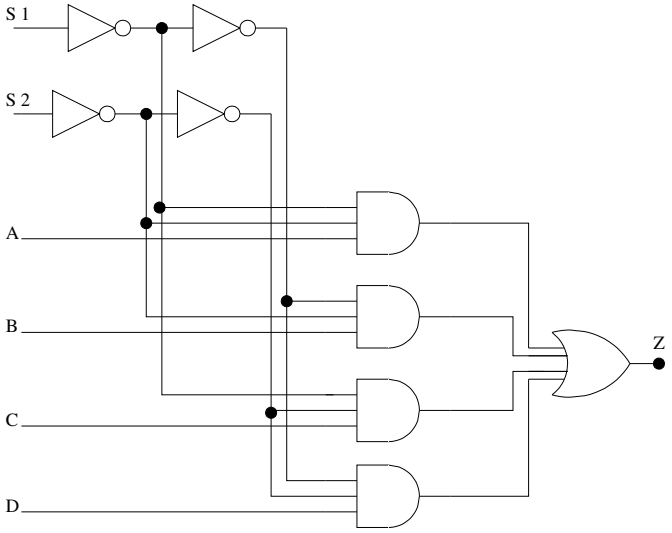
(i) $f(x,y,z) = (\overline{yz}+xyz+\overline{z}, \overline{x}yz+\overline{yz})$	(ii) $f(x,y,z) = (\overline{yz}, xyz+\overline{y})$	(iii) $f(x,y,z) = (\overline{yz}, \overline{x}yz+\overline{y}z)$	(iv) $f(x,y,z) = (x\overline{yz}+xyz+\overline{z}, \overline{x}yz+x\overline{y}\,\overline{z})$

12) Welche Belegung der beiden Eingänge S (Set) und R (Reset) eines SR-Latch ist unzulässig?

(i) $S=1, R=1$	(ii) $S=1, R=0$	(iii) $S=0, R=1$	(iv) $S=0, R=0$

Aufgabe Ü2: Schaltnetze

Betrachten Sie das folgende Schaltbild.



a. Stellen Sie die Kurzform der Funktionstabelle des obigen Schaltbildes auf. Tragen Sie Ihre Lösung in die folgende Tabelle ein:

S_1	S_2	Z

b. Beschreiben Sie das Schaltnetz mittels einer Booleschen Funktion für Z!

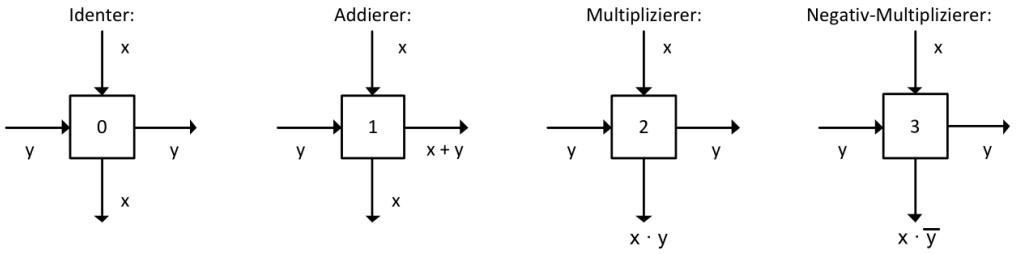
c. Ordnen Sie das Schaltnetz einem Ihnen bekannten Schaltungsbaustein höherer Ordnung zu (Name dieses Bausteins). Wozu werden diese Bausteine ganz allgemein benötigt?

Aufgabe Ü3: Programmierbare Logische Arrays

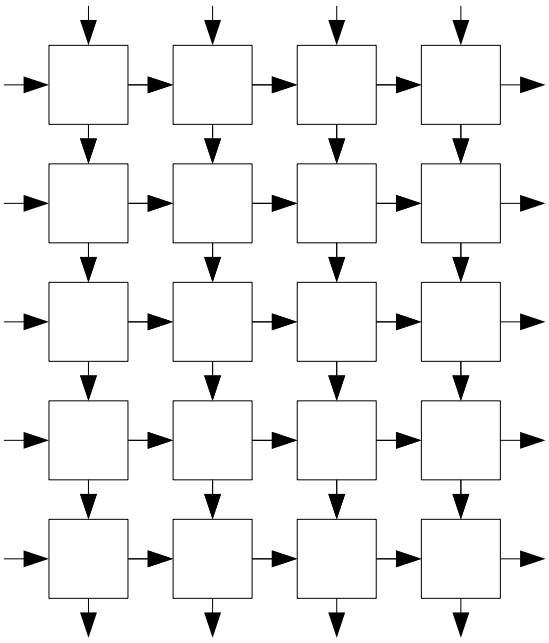
Gegeben sei die folgende boolesche Funktion:

$$f(x, y, z) = \left(xy + \bar{x}\bar{y}, \bar{x}\bar{y} + \bar{x}\bar{z} + xyz \right)$$

Realisieren Sie diese Funktion mittels eines *normierten PLAs*. Verwenden Sie ausschließlich Bausteine der folgenden Typen (0 bis 3):



Tragen Sie dazu jeweils die Typ-Nummer des verwendeten Bausteins in die folgende Vorlage ein. **Kennzeichnen Sie zudem die Und- und die Oder-Ebene. Markieren Sie gesperrte und neutralisierte Eingänge sowohl durch anlegen des entsprechenden Werts**, als auch durch die **explizite Beschriftung mit den Worten „gesperrt“ bzw. „neutralisiert“**. **Beschriften Sie** in das PLA **eingehende Pfeile** mit der jeweils anliegenden logischen Funktion. **Beschriften Sie** anschließend **alle** aus dem PLA **ausgehenden Pfeile** mit der jeweils anliegenden logischen Funktion.



Aufgabe Ü4: Optimierung von Schaltnetzen

a. Gegeben sei die Wahrheitstabelle einer partiellen Booleschen Funktion $g(x_1, x_2, x_3, x_4)$. Undefinierte Ausgaben sind mit einem D gekennzeichnet:

	x_1	x_2	x_3	x_4	$g(x_1, x_2, x_3, x_4)$
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	D
5	0	1	0	1	D
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	D
13	1	1	0	1	1
14	1	1	1	0	D
15	1	1	1	1	0

Minimieren Sie die Funktion g unter Verwendung eines Karnaugh-Diagramms. Beachten Sie dabei die **Don't Care** Argumente und fassen Sie dabei möglichst viele Felder zusammen. Geben Sie abschließend die minimierte Funktion in disjunktiver Form an.

Aufgabe Ü5: Zahlendarstellung

Beantworten Sie die folgenden Fragen im Bezug auf die Dualdarstellung von Ganzzahlen und Gleitkommazahlen:

a. Geben Sie die größte und die kleinste Dezimalzahl samt ihrer Binärdarstellung an, die jeweils unter Verwendung von 11 Bit in der Zweierkomplementdarstellung darstellbar sind.

b. Geben Sie die Zweierkomplementdarstellung der beiden Dezimalzahlen -59 und 128 unter Verwendung von 9 Bit an. Berechnen Sie danach die Summe (-59 + 128). Hat bei Ihrer Addition ein Überlauf (Overflow) stattgefunden? Begründen Sie kurz Ihre Antwort.

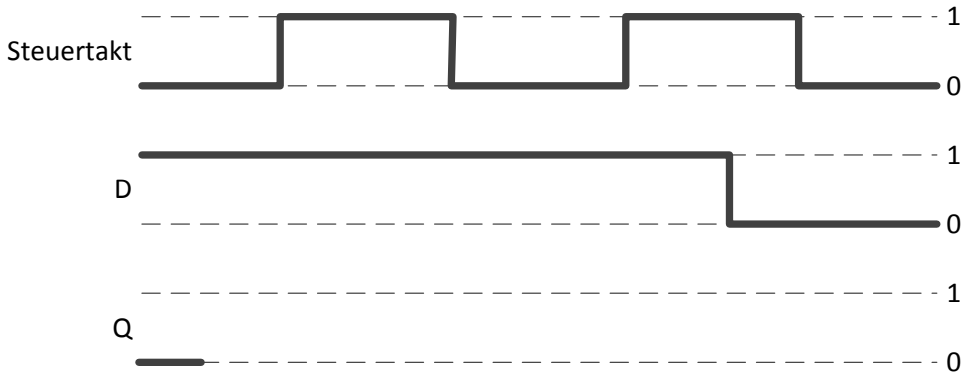
c. Erläutern Sie kurz, warum man bei der Darstellung einer Gleitkommazahl nach dem Standard IEEE 754 die *Bias-Notation* verwendet?

d. Wandeln Sie folgende Zahl, die in Gleitkommadarstellung (IEEE 754) gegeben ist, in ihre Dezimaldarstellung um. Sie dürfen das Ergebnis auch in Bruchdarstellung angeben.

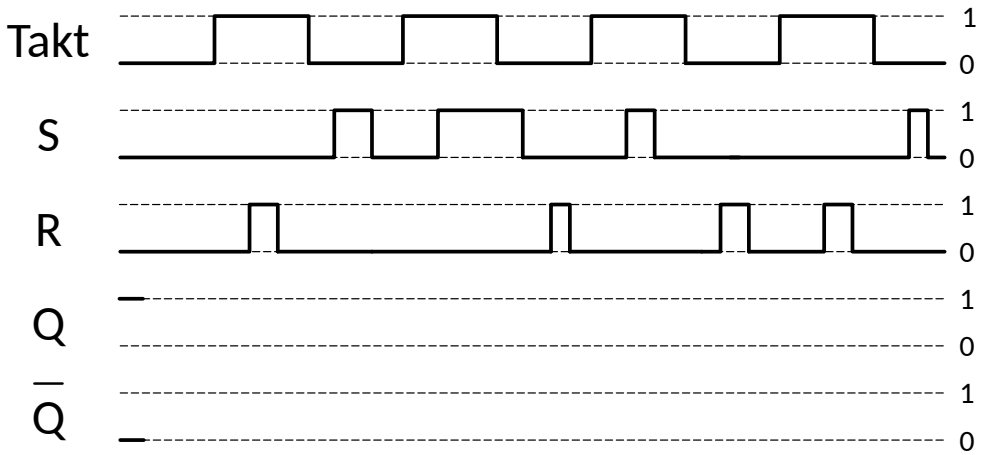
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S	Exponent								Significand																						

Aufgabe Ü6: Flip-Flop-Schaltungen

a. Betrachten Sie das folgende Impulsdiagramm einer D-Flip-Flop-Schaltung. Nehmen Sie an, dass der D-Flip-Flop bei steigender Flanke schaltet. Gehen sie davon aus, dass der Baustein ohne Zeitverzögerung schaltet. Vervollständigen Sie das folgende Impulsdiagramm!

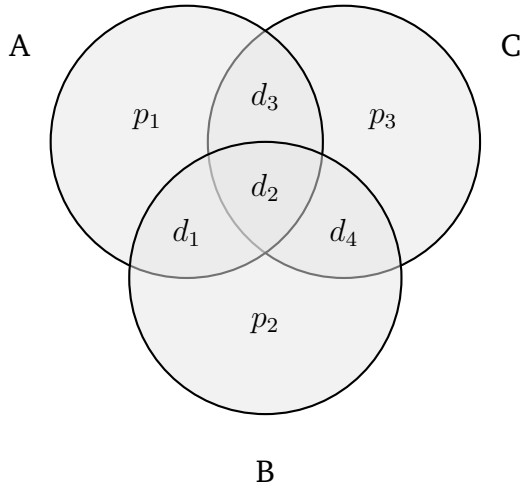


b. Ergänzen Sie ausgehend von einer SR-Latch-Schaltung das nachfolgende Impulsdiagramm für die Ausgänge Q und $\bar{Q}$. Gehen Sie zur Vereinfachung davon aus, dass sich die Pegel von Q und $\bar{Q}$ des Bausteins ohne Zeitverzögerung in Abhängigkeit vom Takt und den Signalen S und R ändern.

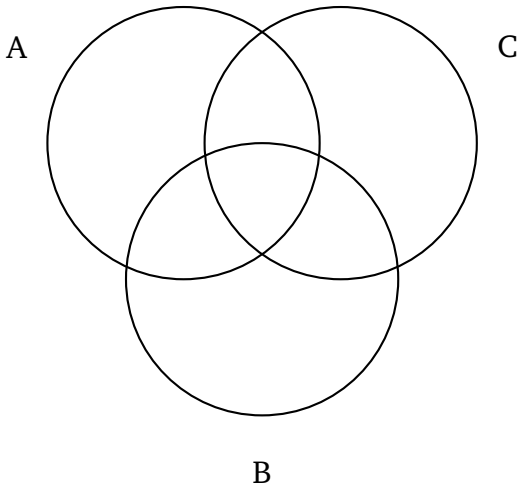


Aufgabe Ü7: Fehlererkennungscodes

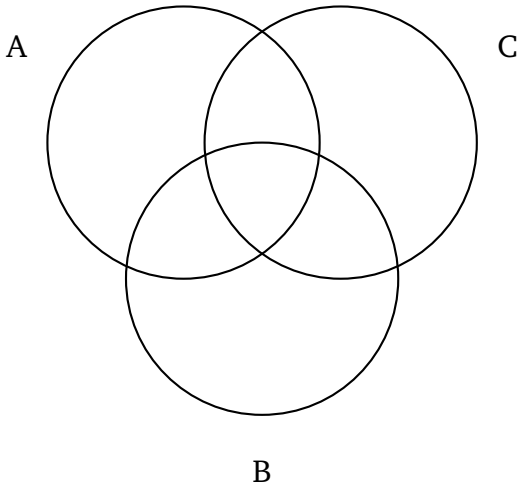
Wir gehen von folgender Struktur der Code-Wörter $d_1d_2d_3d_4p_1p_2p_3$ aus. Wobei $d_i(i \in \{1,2,3,4\})$ für das jeweilige Datenbit und $p_j(j \in \{1,2,3\})$ für das jeweilige Prüf- bzw. Paritätsbit steht. Die Paritätsbits zur Fehlererkennung bzw. Fehlerkorrektur für ein Datenwort $d_1d_2d_3d_4$ können anschaulich mit Hilfe eines Venn-Diagramms berechnet werden, in welchem sich die Bits wie folgt anordnen:



a. Berechnen Sie unter Verwendung des folgenden Venn-Diagramms die Prüfbits für das Datenwort `0110`. Verwenden Sie dazu **gerade Parität**. Tragen Sie zunächst die Datenbits in die für die Berechnung sinnvollen (Schnitt-)Mengen ein.



b. Gehen Sie nun davon aus, dass Sie ein mit dem zuvor beschriebenen Code codiertes Code-Wort `0001101` empfangen haben. Es wurde **gerade Parität** verwendet. Handelt es sich um ein gültiges Codewort? Falls nein, treffen Sie eine Aussage darüber, an welcher/welchen Stelle/Stellen mutmaßlich (ein) Bitfehler aufgetreten ist/sind. Verwenden Sie zur Berechnung das folgenden Venn-Diagramm. Korrigieren Sie (falls nötig) den/die Fehler **innerhalb** des Venn-Diagramms und geben Sie das (ggf. korrigierte) 4-Bit Datenwort an.


A
C
B

Aufgabe Ü8: Assemblerprogrammierung unter SPIM

Beantworten Sie die folgenden Fragen zum Thema Assemblerprogrammierung des MIPS-Prozessors.
Hinweis: Eine Übersicht zu den wichtigsten SPIM-Befehlen finden Sie am Ende des Klausurhefts.

a. Gegeben sei folgendes Programm in SPIM, in dem einige Kommentare eingefügt sind:

```
1  .data
2
3  x: .word 3, 7, 1, 2
4  y: .word 2, 1, 8, 4
5
6
7  .text
8  main:   li      $s0, 4          # Kommentar 1:
9
10         move    $t0, $zero      # Kommentar 2:
11
12         move    $t1, $zero      # Kommentar 3:
13
14
15 while:  bge     $t0, $s0, end   # Kommentar 4:
16
17         mul     $t2, $t0, 4     # Kommentar 5:
18
19         lw      $t3, x($t2)     # Kommentar 6:
20
21         lw      $t4, y($t2)     # Kommentar 7:
22
23         mul     $t5, $t3, $t4   # Kommentar 8:
24
25         add     $t1, $t1, $t5
26
27         addi    $t0, $t0, 1     # Kommentar 9:
28
29         j       while
30
31
32 end:    li      $v0, 1
33         move    $a0, $t1
34         syscall                 # Kommentar 10:
35         j       exit
36
37
38 exit:   li      $v0, 10
39         syscall                 # Programm beenden
```

Ordnen Sie in diesem Programm jeder Zeile, die mit "`# Kommentar <Nr>:`" versehen ist, den jeweils genau passenden der folgenden Kommentare zu. Tragen Sie dazu die Nummer des richtigen Kommentars aus der folgenden Liste in den obigen Coderahmen hinter der betreffenden Kommentarzeile ein!

Nr.	Kommentar	Nr.	Kommentar
(i)	`while j < 4`	(vi)	`Ergebnis initialisieren`
(ii)	`Ergebnis ausgeben`	(vii)	$temp = x_i \cdot y_i$
(iii)	`Lade` y_i	(viii)	`Mit Wortlänge multiplizieren`
(iv)	`j = j + 1`	(ix)	`j = 0`
(v)	`Lade` x_i	(x)	`Lade Länge des Arrays`

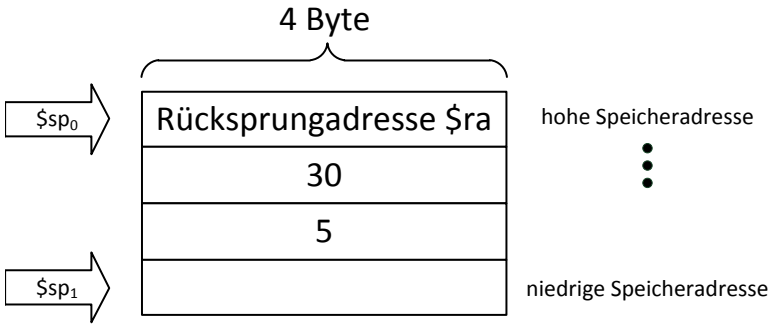
b. Beschreiben Sie kurz, was das oben angegebene Programm aus Teilaufgabe a) bewirkt!

c. Unabhängig von Teilaufgabe a) sei nun die folgende Befehlssequenz eines Unterprogramms gegeben:

```
1         move    $t0, $a0
2         move    $t1, $a1
3         move    $t2, $a2
4
5         add     $v0, $t0, $t1
6         add     $v0, $v0, $t2
7         div     $v0, $v0, 3
8
9         jr      $ra
```

Um welche Art der Wertübergabe handelt es sich beim Aufruf dieses Unterprogramms?

d. Unabhängig von Teilaufgabe a sei nun folgendes Stack-Layout gegeben:



Geben Sie eine Sequenz von SPIM Befehlen an, die dieses Layout erzeugt. Beachten Sie, dass Sie Platz auf dem Stack schaffen müssen und dass dies nach der MIPS-Konvention geschehen soll. Dabei bezeichnet $\$sp_0$ die Position (bzw. den Inhalt) des Stackpointers vor der von Ihnen gegebenen Befehlssquenz und $\$sp_1$ die Position (bzw. den Inhalt) des Stackpointers nach Ihrer Befehlssequenz.

Aufgabe Ü9: Assemblerprogrammierung unter SPIM II

Bearbeiten Sie die folgende Aufgabe zum Thema Assemblerprogrammierung unter SPIM.
Hinweis: Eine Übersicht der SPIM-Befehle finden Sie am Ende des Klausurhefts.

a. Im Folgenden soll ein MIPS-Assembler Programm vervollständigt werden, welches als Nutzereingabe einen Großbuchstaben zwischen A und Z entgegennimmt und als Ergebnis dessen Stelle im Alphabet auf der Konsole ausgibt. Das Alphabet wird zum Abgleich bereits im Datensegment ab der Markierung `alpha` abgelegt. Der vom Nutzer eingegebene Buchstabe wird an der Markierung `input` abgelegt. Der Abgleich, an welcher Stelle sich der eingegebene Buchstabe im Alphabet befindet, geschieht in einer Unterprozedur namens `search_cbr`, welche dann das Ergebnis an das Hauptprogramm zurückgibt. Nach Rückkehr aus dem Unterprozeduraufruf soll das Ergebnis auf der Konsole ausgegeben werden.

   Ergänzen Sie den unten angegebenen Coderahmen um insgesamt **6 Zeilen Code**, so dass die Adresse des Alphabet-Strings und die Adresse des vom Nutzer eingegebenen Strings als Argumente in den gemäß der MIPS-Konvention dafür vorgesehenen Registern an die Unterprozedur übergeben werden. Die Unterprozedur soll nur mit den ihr übergebenen Argumenten arbeiten. Die Rückgabe des Ergebnisses aus der Unterprozedur soll ebenfalls in einem gemäß der MIPS-Konvention dafür vorgesehenen Register erfolgen. Sie müssen sich hierbei nicht um das ebenfalls per Konvention vorgesehene Sichern von Registern kümmern. Es geht lediglich um die korrekte Übergabe der Argumente bzw. des Rückgabewertes und die korrekte Handhabung der Adressen. Tragen Sie Ihre Lösung unter den mit "`# Ihre Lösung:`" markierten Stellen direkt in den folgenden Coderahmen ein:

```
1  .data
2
3  alpha:      .asciiz "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
4  input:      .asciiz " "
5
6  info:       .asciiz "Geben Sie ein Zeichen ein: "
7  linebreak:  .asciiz "\n"
8  fehler:     .asciiz "Fehler!"
9
10 .text
11 main:
12         la      $a0, info
13         li      $v0, 4
14         syscall         # Eingabeaufforderung
15
16         li $v0, 8
17         la $a0, input
18         li $a1, 2
19         syscall         # Zeichen eigeben
20 # Ihre Loesung:
21
22
23
24
25
26
27
28
29
30         jal search_cbr
31
32
```

```
33  # Ihre Loesung:
34
35
36
37
38
39
40
41          la $a0, linebreak
42          li $v0, 4
43          syscall             # Zeilenumbruch ausgeben
44
45          move $a0, $s0
46          li $v0, 1
47          syscall             # Ergebnis ausgeben
48          j exit
49
50
51  search_cbr:
52  # Ihre Loesung:
53
54
55
56
57
58
59          blt $t2, 65, error # Fehlerfall Zeichen mit ASCII-Wert < 65
60          bgt $t2, 90, error # Fehlerfall Zeichen mit ASCII-Wert > 90
61
62  loop:
63          lb $t1, ($t0)       # Lade aktuelles Zeichen
64          beq $t1, $t2, leave_search_cbr
65          addi $t0, $t0, 1    # Adresse = Adresse + 1
66          j loop
67
68  leave_search_cbr:
69          sub $t0, $t0, $a0  # Zaehler um Adresse von "alpha" korrigieren
70          addi $t0, $t0, 1
71  # Ihre Loesung:
72
73
74
75
76
77
78          jr $ra
79
80  error:
81          la $a0, fehler
82          li $v0, 4
83          syscall
84          j exit
85
86  exit:
87          li $v0, 10
88          syscall
```

Aufgabe Ü10: Neighbouring-Problem mittels Quantenannealing

In dieser Aufgabe sollen Sie ein dem Graph-Coloring Problem sehr ähnliche Problemstellung bearbeiten. Gegeben seien drei Produkte (Gummibärchen 1, Gurken 2 & getrockneter Fisch 3) die in Regalen eines Supermarktes nebeneinander angeordnet werden sollen. Die Supermarkt Regale können als Anordnung in einem ungerichteten Graph verstanden werden. Regalplätze (Knoten: $\{A, B, C\}$) sind dabei nur mit ihrem nächsten Nachbarn verbunden. Bei der Zuordnung von Produkten ($\{1, 2, 3\}$) zu Knoten ist darauf zu achten, dass der getrocknete Fisch nicht benachbart mit einem unverpackten Produkt im Regal liegt.
Formulieren sie diese Problemstellung als QUBO. Es müssen auch „katastrophale“ Ereignisse bewertet werden, wie das Ereignis, dass sich ein Produkt gleichzeitig an zwei Regalplätzen befindet. Füllen Sie folgenden Matrix mit den Zahlenwerten $\{-2, 0, 5\}$, je nachdem, wie günstig eine Zustandskombination zu bewerten ist, so dass die Optimierung (Minimierung) mittels Quantenannealing stattfinden kann.

		1A	1B	1C	2A	2B	2C	3A	3B	3C
	1A									
	1B									
	1C									
	2A									
	2B									
	2C									
	3A									
	3B									
	3C									

Überblick über die wichtigsten SPIM Assemblerbefehle		
Befehl	Argumente	Wirkung
`add`	`Rd, Rs1, Rs2`	Rd := Rs1 + Rs2 (mit Überlauf)
`sub`	`Rd, Rs1, Rs2`	Rd := Rs1 - Rs2 (mit Überlauf)
`addu`	`Rd, Rs1, Rs2`	Rd := Rs1 + Rs2 (ohne Überlauf)
`subu`	`Rd, Rs1, Rs2`	Rd := Rs1 - Rs2 (ohne Überlauf)
`addi`	`Rd, Rs1, Imm`	Rd := Rs1 + Imm
`addiu`	`Rd, Rs1, Imm`	Rd := Rs1 + Imm (ohne Überlauf)
`div`	`Rd, Rs1, Rs2`	Rd := Rs1 DIV Rs2
`rem`	`Rd, Rs1, Rs2`	Rd := Rs1 MOD Rs2
`mul`	`Rd, Rs1, Rs2`	Rd := Rs1 $\times$ Rs2
`b`	`label`	unbedingter Sprung nach `label`
`j`	`label`	unbedingter Sprung nach `label`
`jal`	`label`	unbed.Sprung nach `label`, Adresse des nächsten Befehls in $ra
`jr`	`Rs`	unbedingter Sprung an die Adresse in `Rs`
`beq`	`Rs1, Rs2, label`	Sprung, falls `Rs1` $=$ `Rs2`
`beqz`	`Rs, label`	Sprung, falls `Rs` $=$ `0`
`bne`	`Rs1, Rs2, label`	Sprung, falls `Rs1` $\neq$ `Rs2`
`bnez`	`Rs1, label`	Sprung, falls `Rs1` $\neq$ `0`
`bge`	`Rs1, Rs2, label`	Sprung, falls `Rs1` $\geq$ `Rs2`
`bgeu`	`Rs1, Rs2, label`	Sprung, falls `Rs1` $\geq$ `Rs2`
`bgez`	`Rs, label`	Sprung, falls `Rs` $\geq$ `0`
`bgt`	`Rs1, Rs2, label`	Sprung, falls `Rs1` $>$ `Rs2`
`bgtu`	`Rs1, Rs2, label`	Sprung, falls `Rs1` $>$ `Rs2`
`bgtz`	`Rs, label`	Sprung, falls `Rs` $>$ `0`
`ble`	`Rs1, Rs2, label`	Sprung, falls `Rs1` $\leq$ `Rs2`
`bleu`	`Rs1, Rs2, label`	Sprung, falls `Rs1` $\leq$ `Rs2`
`blez`	`Rs, label`	Sprung, falls `Rs` $\leq$ `0`
`blt`	`Rs1, Rs2, label`	Sprung, falls `Rs1` $<$ `Rs2`
`bltu`	`Rs1, Rs2, label`	Sprung, falls `Rs1` $<$ `Rs2`
`bltz`	`Rs, label`	Sprung, falls `Rs` $<$ `0`
`not`	`Rd, Rs1`	Rd := $\neg$ Rs1 (bitweise Negation)
`and`	`Rd, Rs1, Rs2`	Rd := Rs1 & Rs2 (bitweises UND)
`or`	`Rd, Rs1, Rs2`	Rd := Rs1 | Rs2 (bitweises ODER)
`syscall`		führt Systemfunktion aus
`move`	`Rd, Rs`	Rd := Rs
`la`	`Rd, label`	Adresse des Labels wird in Rd geladen
`lb`	`Rd, Adr`	Rd := MEM[Adr]
`lw`	`Rd, Adr`	Rd := MEM[Adr]
`li`	`Rd, Imm`	Rd := Imm
`sw`	`Rs, Adr`	MEM[Adr] := Rs (Speichere ein Wort)
`sh`	`Rs, Adr`	MEM[Adr] MOD 2^{16} := Rs (Speichere ein Halbwort)
`sb`	`Rs, Adr`	MEM[Adr] MOD 256 := Rs (Speichere ein Byte)

Funktion	Code in `$v0`	Funktion	Code in `$v0`
`print_int`	1	`read_float`	6
`print_float`	2	`read_double`	7
`print_double`	3	`read_string`	8
`print_string`	4	`sbrk`	9
`read_int`	5	`exit`	10