Platznummer hier eintragen! ⇒

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR MOBILE UND VERTEILTE SYSTEME
PROF. DR. CLAUDIA LINNHOFF-POPIEN

Sommersemester 2015
Klausur
17.07.2015

Klausur zur Vorlesung Rechnerarchitektur

Beachten Sie die folgenden Hinweise:

- Tragen Sie oben rechts Ihre **Platznummer** ein! Tragen Sie unten Ihre persönlichen Daten ein.
- Zur Klausur sind **keine** Hilfsmittel erlaubt, bei Schwierigkeiten mit der deutschen Sprache darf ein Wörterbuch verwendet werden.
- Tragen Sie Ihre Lösungen **direkt in dieses Klausurheft** ein. Schreiben Sie nicht mit roter oder grüner Farbe und nicht mit Bleistift. Lösungswege und Rechnungen auf den karrierten Bögen (Schmierpapier) werden **nicht berücksichtigt**!
- Die Punkte für die einzelnen Aufgaben dienen nur als vorläufige Richtlinie.
- **Geben** Sie am Ende der Klausur Ihr Klausurheft, Ihre karierten Bögen und Ihre Platznummer **ab**!
- Um Ihre Klausur zu entwerten, streichen Sie bitte deutlich das **Deckblatt und alle Seiten des Klausurenheftes vor der Abgabe durch**. Die Klausur wird dann nicht korrigiert und auch nicht als Prüfungsversuch gewertet.
- Legen Sie Ihren amtlichen Lichtbild– und Studentenausweis gut sichtbar neben sich.

Bearbeitungszeit: 120 Minuten

Name:	
Vorname:	
Matrikelnummer:	

Bewertung

Aufgabe 1	max. 15 Pkt.		Pkt.
Aufgabe 2	max. 12 Pkt.		Pkt.
Aufgabe 3	max. 10 Pkt.		Pkt.
Aufgabe 4	max. 21 Pkt.		Pkt.
Aufgabe 5	max. 20 Pkt.		Pkt.
Aufgabe 6	max. 10 Pkt.		Pkt.
Aufgabe 7	max. 08 Pkt.		Pkt.
Aufgabe 8	max. 12 Pkt.		Pkt.
Aufgabe 9	max. 12 Pkt.	9a Pkt.	9b Pkt.
Summe max. 120 Pkt.			Pkt.

Hinweise zu den Multiple Choice Aufgaben:

- Pro Aussage müssen Sie entscheiden, ob die Aussage **richtig** oder **falsch** ist. Das entsprechende Kästchen markieren Sie bitte **deutlich** mit einem **X**.
- Es werden lediglich die Aussagen/Zeilen gewertet, die *genau ein* **X** enthalten. Für eine richtige Antwort gibt es dabei einen Punkt, für eine falsche Antwort wird ein Punkt abgezogen.
- Aussagen, bei denen entweder keine oder beide Alternativen gekennzeichnet sind, werden mit 0 Punkten gewertet.
- Undeutliche Kennzeichnungen werden im Zweifelsfall zu Ihrem Nachteil gewertet.
- Alle Aussagen sind innerhalb eines Blocks gleich gewichtet und werden jeweils mit 1 Punkt gewertet.
- Man kann auf einen Block im schlechtesten Fall insgesamt 0 Punkte erhalten.
- Ein Block von Aussagen kann *keine*, *eine*, oder *mehrere* richtige Aussagen enthalten.

Fragebeispiel – Hauptstädte:

Sind folgende Aussagen zum Thema **Hauptstädte** richtig (r) oder falsch (f)?		r	f
a	Berlin liegt in Deutschland.		
b	Paris liegt in Deutschland.		
c	London liegt nicht in Europa.		
d	Rom liegt in Spanien.		
e	Paris ist eine Hauptstadt		

Antwortbeispiel:

Sind folgende Aussagen zum Thema **Hauptstädte** richtig (r) oder falsch (f)?		r	f	Bewertung
a	Berlin liegt in Deutschland.	X		(korrekt → 1 Punkt)
b	Paris liegt in Deutschland.	X		(nicht korrekt → −1 Punkt)
c	London liegt nicht in Europa.			(nicht bearbeitet → 0 Punkte)
d	Rom liegt in Spanien.	X	X	(falsch bearbeitet → 0 Punkte)
e	Paris ist eine Hauptstadt	X		(korrekt → 1 Punkt)

Das Ergebnis wäre in diesem Fall: $(+1-1+0+0+1=)$ 1 Punkt (von maximal möglichen 5).

Die **korrekte** Lösung lautet:

Sind folgende Aussagen zum Thema **Hauptstädte** richtig (r) oder falsch (f)?		r	f	Bewertung
a	Berlin liegt in Deutschland.	X		(korrekt → 1 Punkt)
b	Paris liegt in Deutschland.		X	(korrekt → 1 Punkt)
c	London liegt nicht in Europa.		X	(korrekt → 1 Punkt)
d	Rom liegt in Spanien.		X	(korrekt → 1 Punkt)
e	Paris ist eine Hauptstadt	X		(korrekt → 1 Punkt)

Aufgabe 1: Multiple Choice

(15 Punkte)

Sind folgende Aussagen zum Thema **Von-Neumann-Architektur** richtig (r) oder falsch (f)?		**r**	**f**
a	Ein Grundprinzip des Von-Neumann-Rechners ist ein gemeinsamer Speicher für Daten und Programme.		
b	Der klassische Von-Neumann-Rechner besitzt eine MISD (Multiple Instruction – Single Data) Architektur.		
c	Der Von-Neumann-Flaschenhals bezeichnet die Schnittstelle zwischen Datenprozessor und Befehlsprozessor.		
d	Der Befehlszyklus einer CPU entsprechend der von-Neumann-Architektur unterscheidet zwischen der Fetch- und der Execution-Phase.		
e	Ein von-Neumann-konformer Speicher besteht aus unterschiedlich großen Zellen.		

Sind folgende Aussagen zum Thema **Fehlererkennung und -korrektur** richtig (r) oder falsch (f)?		**r**	**f**
a	Hamming-Codes nutzen Paritätsbits zur Fehlererkennung.		
b	Der Hamming-Algorithmus kann nur auf Datenwörter mit einer Länge von $l=2^x$ mit $x \in \mathbb{Z}$ angewendet werden.		
c	Zur Korrektur von Einzelbitfehlern in einem Codewort mit 16 Datenbits benötigt man 4 Prüfbits.		
d	Unter dem Hamming-Abstand versteht man den durchschnittlichen Abstand zwischen fehlerhaften Bits in einem Codewort.		
e	Für die Korrektur von d Einzelbitfehlern benötigt man Codewörter mit mindestens einem Abstand von 2d + 1.		

Sind folgende Aussagen zum Thema **Cache** richtig (r) oder falsch (f)?		**r**	**f**
a	Der Cache-Speicher ist in der Regel kleiner als der Hauptspeicher.		
b	In der Speicherhierarchie dient der Cache als Bindeglied zwischen CPU und Arbeitsspeicher.		
c	Der Vorteil des Cache-Speichers ergibt sich aus dem Lokalitätsprinzip.		
d	Der Zugriff auf Register ist immer langsamer als ein Zugriff auf den Cache-Speicher.		
e	Bei der Verwendung eines *Split-Cache* halbiert sich die Datenmenge, die pro Zeiteinheit geliefert wird.		

Aufgabe 2: Multiplexer

(12 Pkt.)

Bearbeiten Sie folgende Aufgaben:

a. Erläutern Sie in einem Satz die Funktionsweise eines Multiplexers.

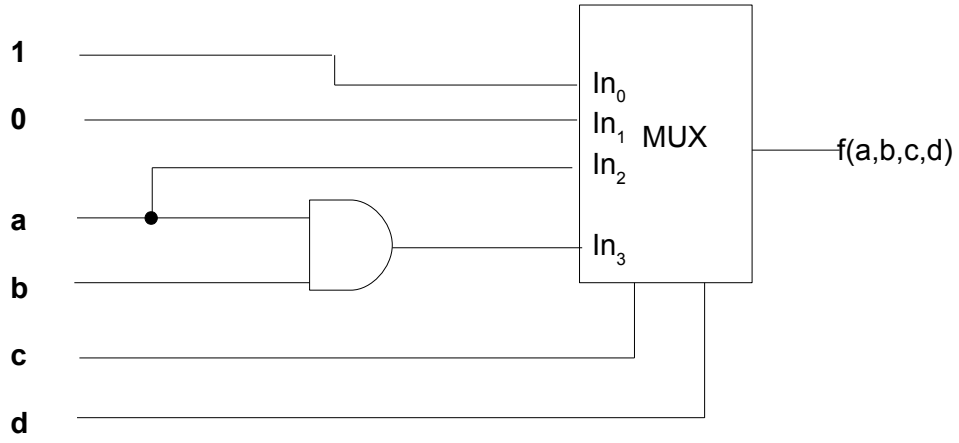
b. Wie viele Steuerleitungen werden für einen n-Eingaben Multiplexer benötigt?

c. Komplexere Multiplexer lassen sich durch das Verschalten mehrerer einfacher Multiplexer realisieren. Wie viele 2-Eingaben-Multiplexer sind notwendig, um einen 4-Eingaben-Multiplexer zu realisieren?

d. Stellen Sie die Kurzform der Funktionstabelle eines 4-Eingaben-Multiplexers mit den Eingangsleitungen In_0, In_1, In_2, In_3, den Steuerleitungen S_0, S_1 und der Ausgangsleitung Out auf. Tragen Sie Ihre Lösung in die folgende Tabelle ein:

S_0	S_1	Out

e. Gegeben sie folgendes Schaltnetz:



Leiten Sie direkt aus diesem Schaltnetz die Definition der realisierenden Funktion f(a,b,c,d) ab und geben Sie diese in **disjunktiver Form** an.
Der 4-Eingaben-Multiplexer soll dabei gemäß Ihrer aufgestellten Funktionstabelle aus der vorherigen Teilaufgabe d arbeiten!
Hinweis: Überlegen Sie sich zunächst, wie die Wahrheitstabelle für f(a,b,c,d) aussieht.

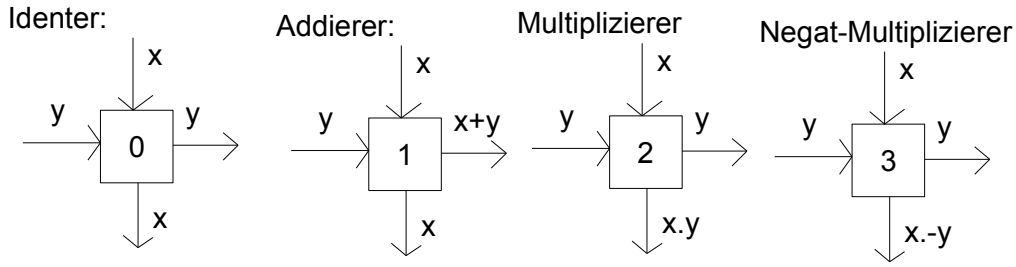
Aufgabe 3: Programmierbare Logische Arrays (PLAs)

(10Punkte)

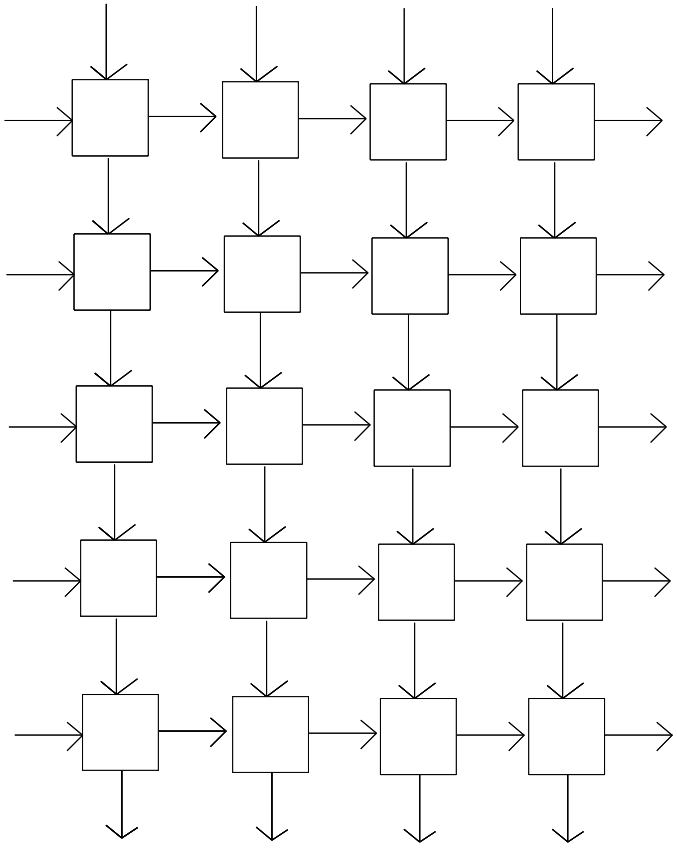
a. Gegeben sie die folgende boolesche Funktion:

$$f(a,b,c) = ab + \bar{b}c + ab\bar{c}$$

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. Beschriften Sie eingehende Pfeile mit der jeweils anliegenden logischen Funktion. Beschriften Sie ebenfalls ausgehende Pfeile, an denen das gewünschte Ergebnis anliegt mit der entsprechenden logischen Funktion.



Aufgabe 4: Optimierung von Schaltnetzen

(21 Pkt.)

a. Gegeben sei folgende Wahrheitstabelle einer Funktion $f(x_1,x_2,x_3,x_4)$

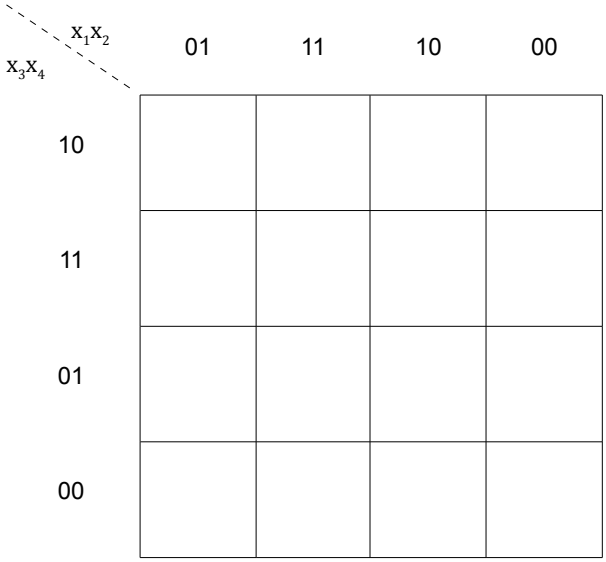
	x_1	x_1	x_1	x_1	$f(x_1,x_2,x_3,x_4)$
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	1

Leiten Sie aus dieser Wahrheitstabelle die Schaltfunktion ihrer vollständigen disjunktiven Normalfunktion (DNF) her.

b. Gegeben sei folgende Termdarstellung der Funktion $g(x_1,x_2,x_3,x_4)$:

$$g(x_1,x_2,x_3,x_4) = x_1x_2x_3x_4 + x_1\bar{x}_2x_3x_4 + x_1x_2x_3\bar{x}_4 + \bar{x}_2\bar{x}_3x_4 + \bar{x}_2x_3 + \bar{x}_1\bar{x}_2$$

Minimieren Sie die Funktion g unter Verwendung eines Karnaugh-Diagramms grafisch. Fassen Sie dabei möglichst viele Felder zusammen. Geben Sie abschließend die minimierte Funktion in disjunktiver Form an. Verwenden Sie für Ihre Lösung die folgende Vorlage:



Die minimierte Funktion lautet:

c. Gegeben sei eine minimierte Funktion j(a,b,c,d,e,f) in disjunktiver Form:

$$j(a,b,c,d,e,f) = aef + b\bar{e}f + c\bar{e}\bar{f} + d\bar{e}\bar{f}$$

(i) Welchen Ihnen bekannten logischen Baustein realisiert diese Funktion j(a,b,c,d,e,f)? Geben Sie bei Ihrer Antwort eine **genaue Bezeichnung** des logischen Bausteins an und nennen Sie dabei die jeweilige Aufgabe der Inputparameter a – f.

(ii) Entwerfen Sie nun das Schaltbild dieser Funktion j(a,b,c,d,e,f). Verwenden Sie dabei ausschließlich die elementaren Gatter *AND, OR, NOT* und tragen Sie Ihre Lösung in folgende Vorlage ein. Zur Vereinfachung Ihres Schaltbilds dürfen Sie auch gerne OR bzw. AND Gatter mit mehr als 2 Eingängen verwenden.

a ——

b ——

c ——

d ——

e ——

f ——

Aufgabe 5: Zahlendarstellung im Rechner

(20 Pkt.)

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

a. Erläutern Sie in einem Satz den Begriff *Zweierkomplement.*

b. Geben Sie eine Dezimalzahl an, die in 1er, 2er-Komlement und sign/magnitude-Darstellung durch dieselbe Bitfolge repräsentiert wird.

c. Wie lautet die **kleinste** Dezimalzahl, die in der Zweierkomplementdarstellung darstellbar ist, wenn 1 Byte inklusive des Vorzeichenbits verwendet wird?

d. Geben Sie die Zweierkomplementdarstellung der folgenden Dezimalzahl an. Verwenden Sie zur Darstellung 1 Byte (inklusive des Vorzeichenbits):
y = -94

e. Gegeben seien die beiden Binärzahlen u = 10011001 und v = 10110011 in ihrer Zweierkomplementdarstellung. Kann bei der **Subtraktion** der beiden Zahlen ein Überlauf stattfinden?
Begründen Sie Ihre Antwort **ohne** Berechnung!

f. **Addieren** Sie nun die beiden Zahlen u und v aus der vorherigen Teilaufgabe e! Der Rechenweg **muss** ersichtlich und nachvollziehbar sein!

g. Hat bei Ihrer Addition in der vorherigen Teilaufgabe f ein Überlauf (Overflow) stattgefunden? Begründen Sie kurz Ihre Antwort.

h. Gegeben sei nun die Dezimalzahl x = -9.125.
Wandeln Sie diese Dezimalzahl in ihre Gleitkommadarstellung nach IEEE 754 mit einfacher Genauigkeit (32-Bit) um und tragen Sie Ihr Ergebnis in die entsprechende Vorlage ein! Das niederwertigste Bit des Exponenten befindet sich dabei auf Platz 23, das niederwertigste Bit des Signifikanden befindet sich auf Platz 0.

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
S	Exponent								Signifikand																						

i. Wandeln Sie folgende Zahl, die in 32-Bit-Gleitkommadarstellung nach IEEE 754 gegeben ist, in ihre Dezimaldarstellung um! Das niederwertigste Bit des Exponenten befindet sich dabei auf Platz 23, das niederwertigste Bit des Signifikanden befindet sich auf Platz 0.

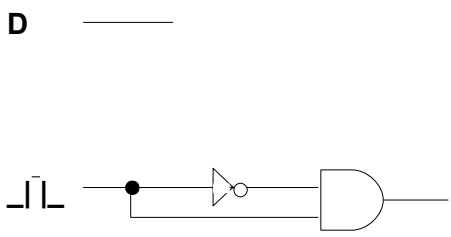
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	1	1	1	0	1	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
S	Exponent								Signifikand																						

Aufgabe 6: Flip-Flop-Schaltung

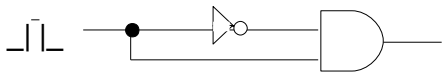
(10 Pkt.)

Bearbeiten Sie folgende Aufgaben:

a. Skizzieren Sie das Schaltwerk eines D-Flip-Flops! Benutzen Sie dazu nur die elementaren Gatter vom Typ AND, NOT und NOR und tragen Sie Ihre Lösung in die folgende Vorlage ein:

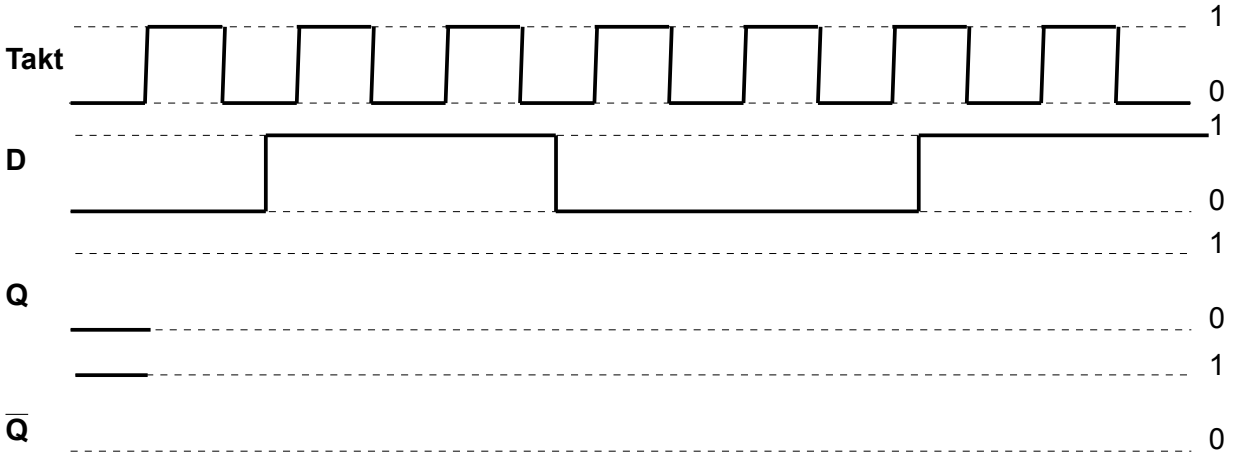


b. Gehen Sie von einem D-Flip-Flop mit folgendem Taktgeber aus:



Realisiert dieser Taktgeber den Zustandsübergang bei steigender oder fallender Flanke?

c. Gegeben sei das nachfolgende Impulsdiagramm eines D-Flip-Flops mit dem Taktgeber aus der vorherigen Teilaufgabe b. Vervollständigen Sie das folgende Impulsdiagramm für die Ausgänge Q und $\overline{Q}$ unter der Annahme, dass der Baustein ohne Zeitverzögerung schaltet:



d. Welches Problem ergibt sich beim D-Flip-Flop im Hinblick auf das Speichern über mehrere Takte hinweg?

Aufgabe 7: Fehlererkennung und -Korrektur

(8 Pkt.)

Als Schutz vor Speicherfehlern werden u.a. Codes zur Fehlererkennung und zur Fehlerkorrektur eingesetzt. Bearbeiten Sie dazu folgende Aufgaben:

a. Verwenden Sie den Hamming Algorithmus und kodieren Sie das folgende 8-Bit Daten-Wort in einem 12-Bit Hamming Code:

(i) 0110 0011

Verwenden Sie dazu **gerade Parität** und kennzeichnen Sie ihr Ihrem resultierenden Codewort alle eingesetzten Paritätsbits!

b. Gegeben sei das folgende 12-Bit Codewort, welches nach dem Hamming-Alrgorithmus enkodiert ist:

1101 0110 0100

Bearbeiten Sie davon ausgehend die folgenden Schritte:

(i) Verwenden Sie wieder gerade Parität und geben Sie alle fehlerhaften Paritätsbits an.

(ii) Identifizieren Sie (falls möglich) das fehlerhafte Datenbit.

(iii) Korrigieren Sie (falls möglich) den Fehler und geben Sie ein (ggf. korrigiertes) Codewort an.

(iv) Geben Sie das resultierende 8-Bit Datenwort an.

Aufgabe 8: Assemblerprogrammierung unter SPIM I

(12 Pkt.)

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  input:    .asciiz „Geben Sie eine Zahl ein:“
3  .text
4  main:
5       la   $a0, input
6       li   $v0, 4
7       syscall             # Kommentar 1:
8
9       li   $v0, 5
10      syscall             # Zahl einlesen
11
12      bltz $v0, error     # Kommentar 2:
13
14      move $t0, $v0       # Kommentar 3:
15
16      li   $t1, 1         # Kommentar 4:
17
18 while:
19      beqz $t0, output    # Kommentar 5:
20
21      mul  $t1, $t1, $t0  # Kommentar 6:
22
23      sub  $t0, $t0, 1    # Kommentar 7:
24
25      j    while          # Kommentar 8:
26
27 output:
28      li   $v0, 1         # Zahl ausgeben
29      move $a0, $t1       # Kommentar 9:
30      syscall
31
32      li   $v0, 10        #Beenden
33      syscall
34
35 error:
36      li   $t1, -2        # Kommentar 10:
37      j    output         # Sprung zu Marke
```

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)	Ergebnis initialisieren	(vi)	Nutzereingabe sichern
(ii)	Übergabe Ergebnis für Ausgabe	(vii)	Eingabe := Eingabe -1
(iii)	Schleife wiederholen	(viii)	Sprung, falls Eingabe < 0
(iv)	Speichere -2 als Ergebnis	(ix)	while Eingabe > 0
(v)	String ausgeben	(x)	Ergebnis := Ergebnis * Eingabe

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

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

```
1     li   $t0, 3
2     li   $t1, 5
3     addi $sp, $Sp, -12
4     sw   $fp, 12($sp)
5     sw   $ra, 8 ($sp)
6     sw   $t0, 8 ($sp)
7     sw   $t1, 4 ($sp)
```

Zeichnen Sie den Stack, wie er nach Ausführung dieser Befehlssequenz aussieht. Tragen Sie auch den Stackpointer in Ihre Skizze ein.

d. Welches Problem liegt nach der Abarbeitung der Befehlssequenz aus der vorherigen Teilaufgabe c vor?

Hinweise zur folgenden Aufgabe:
Aufgabe 9 ist eine Wahlaufgabe! Lösen Sie entweder 9a oder 9b

In § 17, Absatz 1, Satz 2 der Prüfungs- und Studienordnung der Ludwig-Maximilian-Universität München für den Bachelorstudiengang Informatik/Medieninformatik bzw. in vergleichbaren Inhalten der Nebenfachordnungen heißt es „Der oder dem Studierenden können Themen zur Auswahl gegeben werden; Ein Anspruch hierauf besteht nicht“

Sie haben im folgenden Fall die Wahl, entweder Aufgabe 9a oder Aufgabe 9b zulösen:

– *Aufgabe 9a: Assemblerprogrammierung unter SPIM II* oder

– *Aufgabe 9b: Anwendungen der Digitalisierung*

Beide Teilaufgaben ergeben die gleiche maximale Anzahl an Punkten.

Wenn Sie beide Aufgaben bearbeiten möchten, so wird bei der Korrektur diejenige Aufgabe gewertet, die eine höhere Punktzahl erreicht hat. Die Punkte der Aufgabe mit der geringeren Punktzahl verfallen.

Aufgabe 9a: Assemblerprogrammierung unter SPIM II

(12 Pkt.)

Bearbeiten Sie die folgende Aufgabe zum Thema Assemblerprogrammierung unter SPIM.
Hinweis: Eine Übersicht zu den wichtigsten SPIM-Befehlen finden Sie am Ende des Klausurhefts.

a. Im Folgenden soll ein MIPS-Assembler Programm vervollständigt werden, das eine Unterprozedur `triangle` aufruft, welche aus den Benutzereingaben a und b die Seitenlänge c^2 eines Dreiecks anhand des Satz von Pythagoras ($c^2 = a^2 + b^2$) berechnet. Die Unterprozedur bekommt dabei jeweils die Adresse der Eingaben und der Ausgabe als Argument übergeben und arbeitet daher nach dem Prinzip **Call-by-Reference**. Nach erfolgreicher Berechnung von c^2soll das Ergebnis an der übergebenen Adresse für die Ausgabe gespeichert werden. Die Unterprozedur `output` lädt dann den Wert von `ausgabe` und gibt das Ergebnis auf der Konsole aus.

Ergänzen Sie das folgende SPIM-Programm auf der nächsten Seite um insgesamt **4 Zeilen Code** und jeweils einen sinnvollen Kommentar, so dass die oben beschriebene Funktionalität vollständig und korrekt implementiert wird und das Programm mittels **Call-by-Reference** arbeitet! Tragen Sie Ihre Lösung unter den mit Ihre Lösung: markierten Stellen direkt in den folgenden Coderahmen auf der nächsten Seite ein:

```
1   .data
2   input_a:    asciiz „Seitenlaenge a:“
3   input_b:    asciiz „Seitenlaenge b:“
4   eingabe:    .word 0 0
5   ausgabe:    .word 0
6
7   .text
8   main:
9         li    $v0, 4
10        la    $a0, input_a      # print input_a String
11        syscall
12
13        li    $v0, 5            # read input a
14        syscall
15        sw    $v0, eingabe      # Eingabe [0]:=a
16
17        li    $v0, 4
18        la    $a0, input_b      # print input_b String
19        syscall
20
21        li    $v0, 5            # read input b
22        syscall
23        sw    $v0, eingabe+4    # Eingabe [1]:=b
24
25        la    $a0, eingabe      # Pass address of eingabe to Argument_1
26        la    $a1, ausgabe      # Pass address of ausgabe to Argument_2
27        jal   triangle          # Jump to label triangle
28        jal output        # Jump to label output
29
30  triangle:
31  #Ihre Lösung:
32
33
34
35
36
37
38        mul   $t0, $t0, $t0     # $t0 = a^2
39        mul   $t1,  $t1, $t1    # $t1 = b^2
40
41        add   $t2, $t0, $t1     # $t2 = a^2 + b^2
42
43  #Ihre Lösung:
44
45
46
47        jr    $ra               # Jump back to return address
48
49  output:
50  #Ihre Lösung:
51
52
53
54        li    $v0, 1            # print int
55        syscall
56
57        li    $v0, 10           # exit.
58        syscall
```

Aufgabe 9b: Anwendungen der Digitalisierung

(12 Pkt.)

Nennen Sie 4 Branchen, bei denen sich Ausprägungen der Digitalisierung beobachten lassen und geben Sie dazu jeweils 2 Beispiele an. Benutzen Sie für Ihre Antwort die folgende Vorlage:

Branche I:

Beispiel 1:

Beispiel 2:

Branche II:

Beispiel 1:

Beispiel 2:

Branche III:

Beispiel 3:

Beispiel 2:

Branche IV:

Beispiel 1:

Beispiel 2:

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 × 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 ≠ Rs2
bnez	Rs1, label	Sprung, falls Rs1 ≠ 0
bge	Rs1, Rs2, label	Sprung, falls Rs1 ≥ Rs2
bgeu	Rs1, Rs2, label	Sprung, falls Rs1 ≥ Rs2
bgez	Rs, label	Sprung, falls Rs ≥ 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 ≤ Rs2
bleu	Rs1, Rs2, label	Sprung, falls Rs1 ≤ Rs2
blez	Rs, label	Sprung, falls Rs ≤ 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 := ¬ Rs1 (bitweise Negation)
and	Rd, Rs1, Rs2	Rd := Rs1 & Rs2 (bitweises UND)
or	Rd, Rs1, Rs2	Rd := Rs1 | Rs2 (bitweises ODER)
xori	Rd, Rs1, Imm	Rd := Rs1 ↮ Imm (bitweises XOR)
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