

## 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

(20 Pkt.)

Sind folgende Aussagen zum Thema Addiernetze richtig (r) oder falsch (f)?		r	f
a	Ein Halbaddierer addiert zwei Bits unter Berücksichtigung eines möglichen Übertrags.	X	
b	Für das Resultat R der Addition zweier Bits x und y gilt: $R = \bar{x} \cdot y + x \cdot \bar{y}$	X	
c	Ein Halbaddierer liefert unter der Annahme einer minimalen Umsetzung schneller ein Ergebnis als ein Volladdierer	X	
d	Ein Ripple-Carry-Addiernetz zur Addition von zwei n-stelligen Dualzahlen benötigt zwingend n Volladdierer.		X
e	Die Idee eines Ripple-Carry-Addiernetzes ist die Zusammenfassung von Bitgruppen zur Beschleunigung der Berechnung.		X

Sind folgende Aussagen zum Thema Boolesche Algebra richtig (r) oder falsch (f)?		r	f
a	Es gilt: $X \cdot (Y + X) = (X \cdot Y) + X$ $X \cdot Y + X \cdot X$	X	
b	Es gibt $2^n$ Möglichkeiten eine n-stellige Boolesche Funktion zu definieren.		X
c	Eine der Regeln von de Morgan lautet: $\overline{X + Y} = \bar{X} \cdot \bar{Y}$	X	
d	Der NAND-Baustein reicht als einziges Schaltelement aus, um jede beliebige Boolesche Funktion durch eine Schaltung zu realisieren.	X	
e	Die Begriffe Schaltfunktion und Boolesche Funktion sind im Allgemeinen gleichbedeutend.	<del>X</del>	X

Sind folgende Aussagen zum Thema SPIM richtig (r) oder falsch (f)?		r	f
a	Der Stack wächst mit größer werdenden Hauptspeicheradressen in Richtung der Adresse 0.	X	
b	Programmcodem und Stack besitzen jeweils einen eigenen Speicher.		
c	Der Stackpointer \$sp zeigt nach Konvention auf das Wort, das zuletzt in den Stack geladen wurde.	X	
d	Wegen seiner dynamischen Zellbreite eignet sich der Stack zur Speicherung von Daten, für die die statische Breite der Register (32 Bit) nicht ausreicht.		
e	Beim Aufruf von Unterprogrammen wird der Program-Counter PC manipuliert.	X	

Sind folgende Aussagen zum Thema Fehlererkennung und -korrektur richtig (r) oder falsch (f)?		r	f
a	Der Hamming-Abstand zwischen zwei gleichlangen Codewörtern wird bestimmt, indem man bitweise AND anwendet und die 1er im Ergebnis zählt.		X
b	Wird ein einzelnes Bit eines Codewortes x invertiert, so hat das daraus resultierende Codewort die Hamming-Distanz 2 zum Codewort x.		X
c	Der Hamming-Algorithmus wird auf Codewörter angewandt, um diese zu komprimieren.		X
d	Zur Korrektur von Einzelbitfehlern in einem Codewort mit 8 Datenbits benötigt man 4 Prüfbits.		X
e	Falls Venn-Diagramme zur Visualisierung der Fehlererkennung verwendet werden, so werden in den Schnittmengen die Paritätsbits eingetragen.	X	

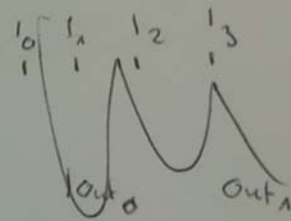
**Aufgabe 2: Encoder über programmierbare logische Arrays**

(26 Pkt.)

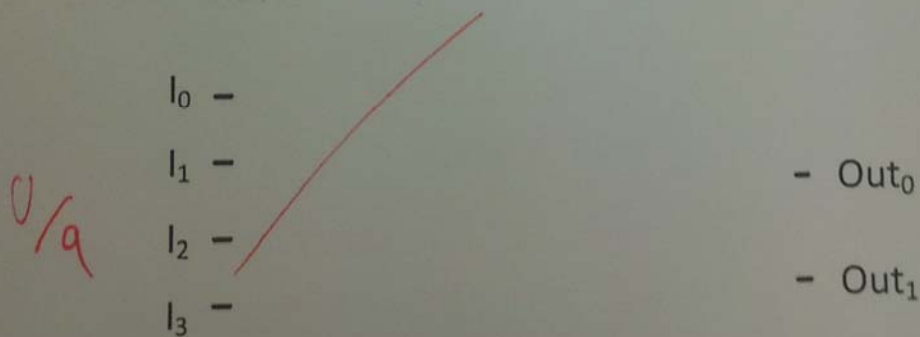
In dieser Aufgabe soll schrittweise ein 4-zu-2-Encoder durch ein Programmierbares logisches Array (PLA) realisiert werden. Ein Encoder besitzt die umgekehrte Funktionalität eines Dekoders. Er besitzt 4 Eingänge  $I_0, I_1, I_2, I_3$  und die zwei Ausgänge  $Out_0$  und  $Out_1$ . Es wird angenommen, dass stets genau einer der Eingänge mit einer 1 belegt ist. Ist ein Eingang  $I_i$  mit einer 1 belegt, so ist  $(Out_1, Out_0)$  die duale Darstellung der Dezimalzahl  $i$ . Bearbeiten Sie dazu folgende Teilaufgaben:

- a. Bestimmen Sie nun die Schaltfunktion des 4-zu-2-Encoders. Verwenden Sie dabei die Bezeichnungen gemäß der obigen Beschreibung.

$$\begin{aligned}
 & I_0 \ I_1 \ I_2 \ I_3 \\
 Out_0 &= \bar{I}_0 I_1 \bar{I}_2 \bar{I}_3 + \bar{I}_0 \bar{I}_1 I_2 \bar{I}_3 + \bar{I}_0 \bar{I}_1 \bar{I}_2 I_3 + \bar{I}_0 I_1 I_2 \bar{I}_3 \quad \checkmark \\
 Out_1 &= \bar{I}_0 \bar{I}_1 I_2 \bar{I}_3 + \bar{I}_0 I_1 \bar{I}_2 I_3 \quad \checkmark
 \end{aligned}$$



- b. Zeichnen Sie das Schaltnetz eines 4-zu-2-Encoders. Ergänzen Sie dazu folgende Vorlage gemäß der obigen Beschreibung zu dem entsprechenden Schaltnetz eines 4-zu-2-Encoders:



Hinweis: Die Erstellung der zugehörigen Wahrheitstabelle kann hierbei hilfreich sein.

$I_0$	$I_1$	$I_2$	$I_3$	$Out_1$	$Out_0$
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1



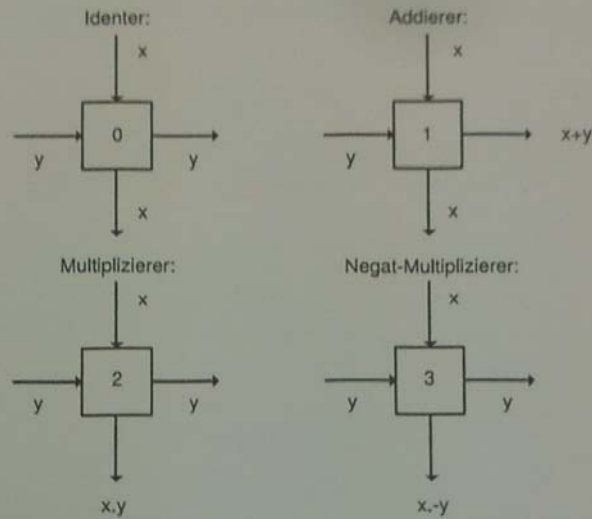
- c. Erläutern Sie kurz das Konzept von programmierbaren logischen Arrays (PLAs).

Bei einem PLA teilt man die benötigte Schaltung in die häufigsten verwendeten Bausteine auf. ~~Dadurch werden Kosten gespart~~

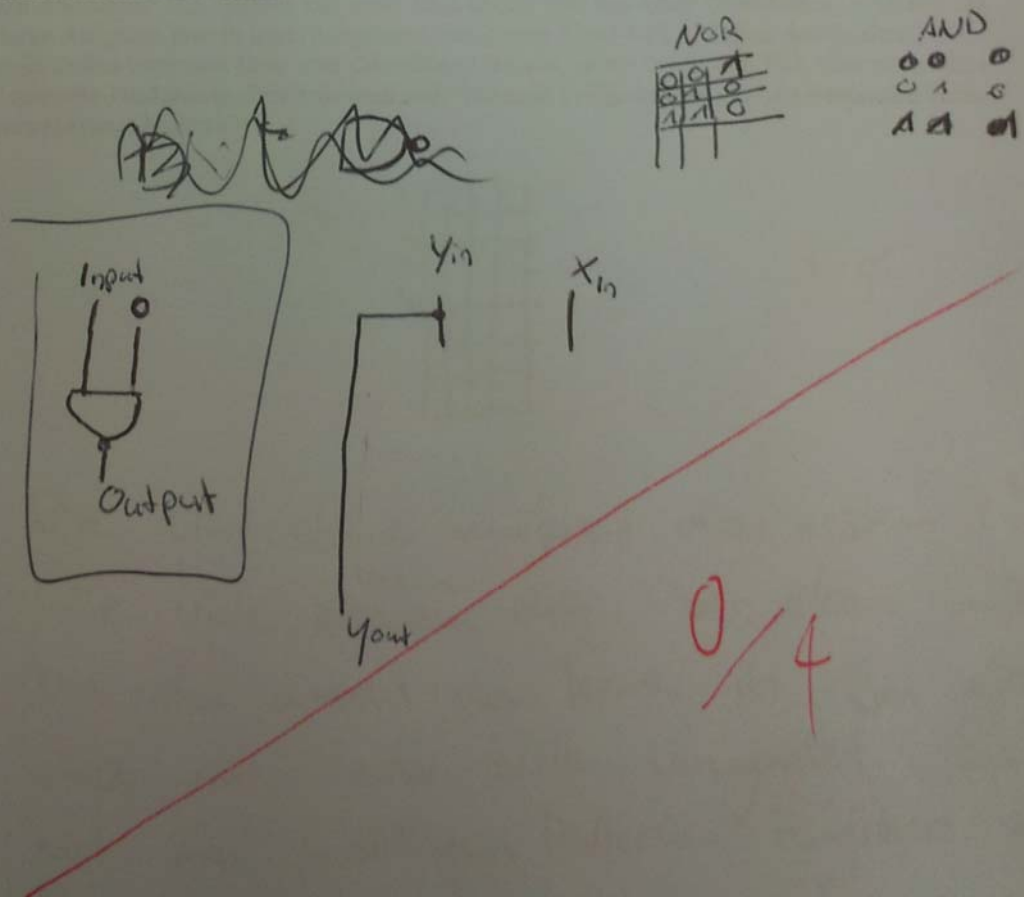
Danach setzt man diese Teile auf ein Gitter. Man achtet dabei UND- und ODER-Ebene. ~~Dadurch spart man meist Kosten~~ ein.

2/3

d. Ein PLA besitzt die vier folgenden Elemente:

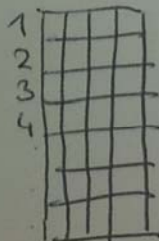


Zeichnen Sie zunächst die NOT-Schaltfunktion auf, wobei nur das NOR-Gatter verwendet werden darf. Auf Basis dieser Überlegung soll anschließend das Schaltbild für den Typ 2: Multiplizier-Baustein mittels des NOR-Gatters entwickelt werden. Verwenden Sie dazu ausschließlich das NOR-Gatter.



Platz zur Fortsetzung der Lösung von Teilaufgabe d

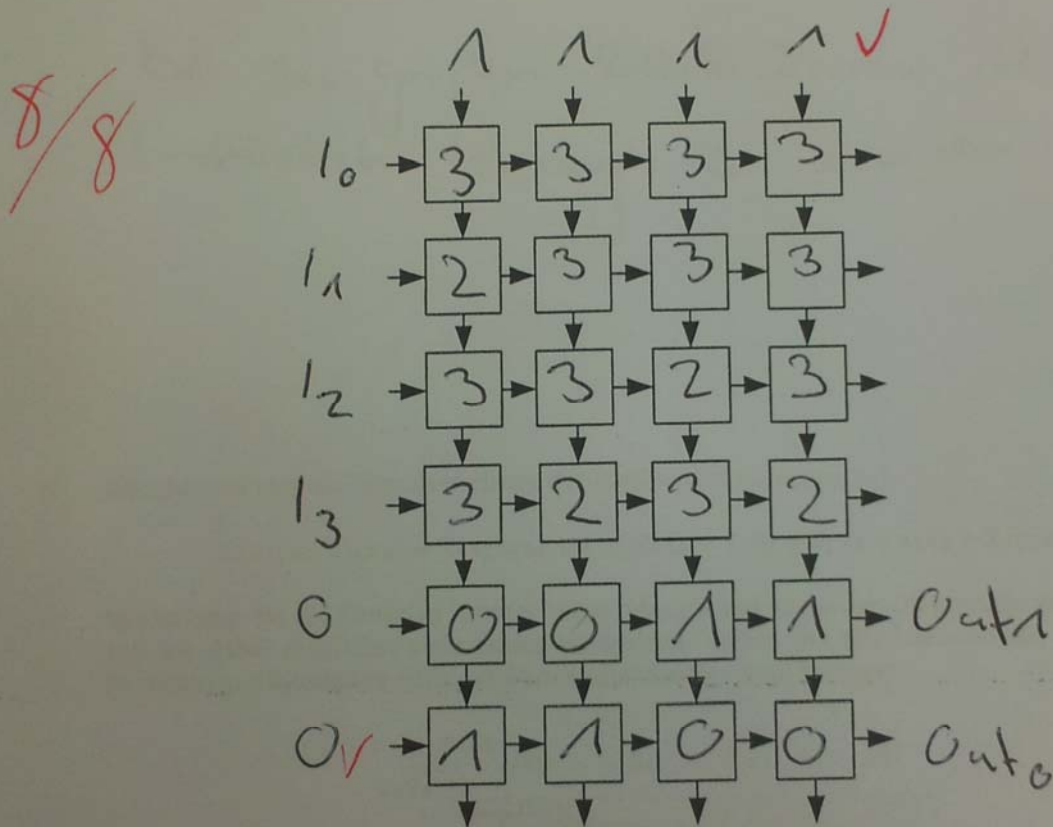
- e. Ein normierter PLA besteht aus einer Und-Ebene und aus einer Oder-Ebene. Erklären Sie deren Aufgaben jeweils kurz. Ausgehend von einem 7-mal-4-PLA (Zeilen-mal-Spalten): Wieviele Zeilen umfassen Und- und Oder-Ebene jeweils, wenn durch den PLA eine vierstellige Boolesche Funktion realisiert werden soll? Wieviele Produktterme darf die boolesche Funktion maximal besitzen?



3/5

Die Und-Ebene umfasst die ersten vier <sup>Zeilen</sup> ~~Spalten~~ <sup>✓</sup>  
 Wie viele <sup>Zeilen</sup> ~~Ebenen~~ die Oder-Ebene umfasst hängt  
 von der Anzahl der Terme ab. Bei einem  
 wird nur eine Zeile verwendet. Das final  
 mit einer 4-stelligen Booleschen <sup>✓</sup> Funktion hat ein  
 Limit von 3 <sup>✓</sup> Oder Ebenen / Termen

- f. Realisieren Sie die Schaltfunktion eines 4-zu-2-Encoders mittels eines normierten PLAs. Verwenden Sie ausschließlich Bausteine der in Teilaufgabe d dargestellten Typen 0 bis 3. Kennzeichnen Sie in Ihrer Skizze 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. Die Lösung soll in folgende Vorlage eingetragen werden:





## Aufgabe 3: Optimierung von Schaltnetzen

(14 Pkt.)

- a. Zur Optimierung von Schaltfunktionen wird neben Karnaugh-Diagrammen auch das Verfahren nach Quine-McCluskey eingesetzt werden. Erläutern Sie, in welchen Fällen eine Verwendung des Verfahrens nach Quine-McCluskey wesentlich vorteilhafter ist als die Verwendung von Karnaugh-Diagrammen.

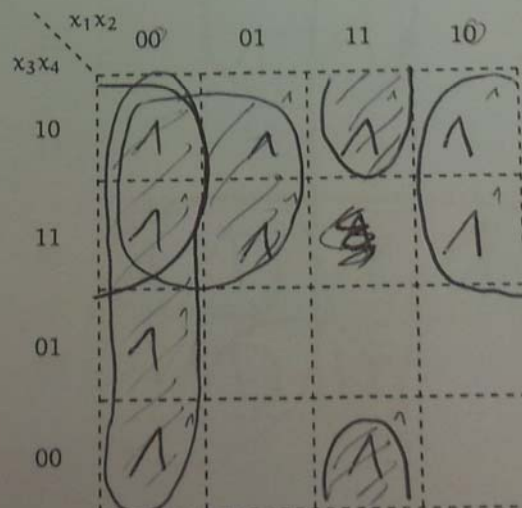
Bei zu großen Schalt Termen ist es ~~sehr~~ sehr unübersichtlich Karnaugh zu verwenden.  $\Rightarrow$

1/2

- b. Gegeben sei nun die Termdarstellung der Funktion  $h(x_1, x_2, x_3, x_4)$ :

$$h(x_1, x_2, x_3, x_4) = \bar{x}_1 x_2 x_3 x_4 + x_1 x_2 \bar{x}_3 \bar{x}_4 + \bar{x}_1 \bar{x}_2 + \bar{x}_2 x_3 + x_3 \bar{x}_4 + \bar{x}_1 x_2 x_3$$

Minimieren Sie die Funktion  $h$  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 folgende Vorlage:



5/6

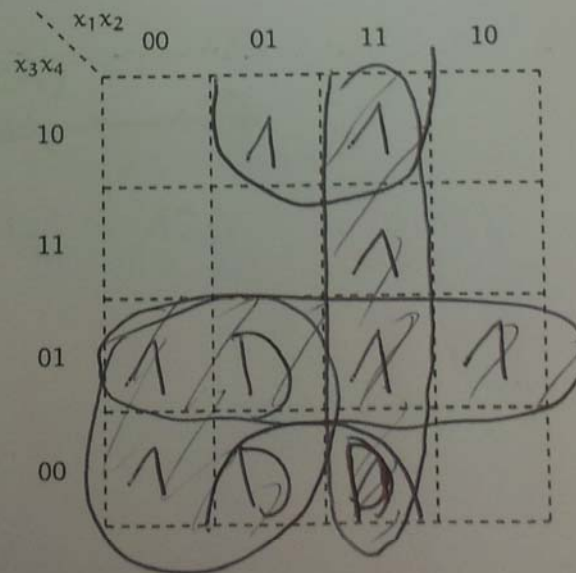
$$\bar{x}_1 \bar{x}_2 + x_1 x_2 \bar{x}_4 + \bar{x}_1 \bar{x}_3 + \bar{x}_2 x_3$$



- c. Gegeben sei nun 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	1
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	D
5	0	1	0	1	D
6	0	1	1	0	1
7	0	1	1	1	0
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	1

Minimieren Sie die Funktion  $g$  unter Verwendung eines Karnaugh-Diagramms grafisch. 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. Verwenden Sie folgende Vorlage:



$$x_1 x_2 + x_3 \overline{x_4} + \overline{x_1} \overline{x_3} + x_2 \overline{x_4}$$

5/6

11/19

**Aufgabe 4: Zahlendarstellung**

(17 Pkt.)

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

- a. Was versteht man unter der Speichereinheit "Wort"? Wie wird die Länge dieser Speichereinheit festgelegt?

Die Speichereinheit hängt davon ab man mit einfacher oder doppelter Präzision speichert.

Normal ist ein Wort: 16 Byte

bei doppelter Präzision 32 Byte

0/2

- b. Geben Sie die größte und die kleinste Dezimalzahl samt ihrer Binärdarstellung an, die jeweils unter Verwendung von 9 Bit in der Zweierkomplementdarstellung darstellbar ist.

1 2 4 8 16 32 64 128 256

$$100000001_2 = -127$$

~~$$100000001_2 = -127$$~~

$$011111111_2 = 126$$

25/4

32  
16  
48

- c. Gegeben seien die beiden Dezimalzahlen  $x = -116$  und  $y = 52$ . Geben Sie deren Binärdarstellung an und nutzen Sie dabei, sofern möglich, das Zweierkomplement. Verwenden Sie zur Darstellung jeweils 9 Bit.

$$00111000_2 = 116$$

$$11000100_2 = -116$$

$$000110100_2 = 52$$

~~$$11100100_2 = -52$$~~

3/3

- d. Berechnen Sie die Zweierkomplementdarstellung der Zahl  $z = x - y$  ( $x$  und  $y$  mit den Werten aus Aufgabe c). Der Rechenweg muss ersichtlich sein!

$$x - y$$

$$= x + (-y)$$

$$\begin{array}{r} 110001100 \\ + 111001100 \\ \hline 1101011000 \end{array}$$

2/3





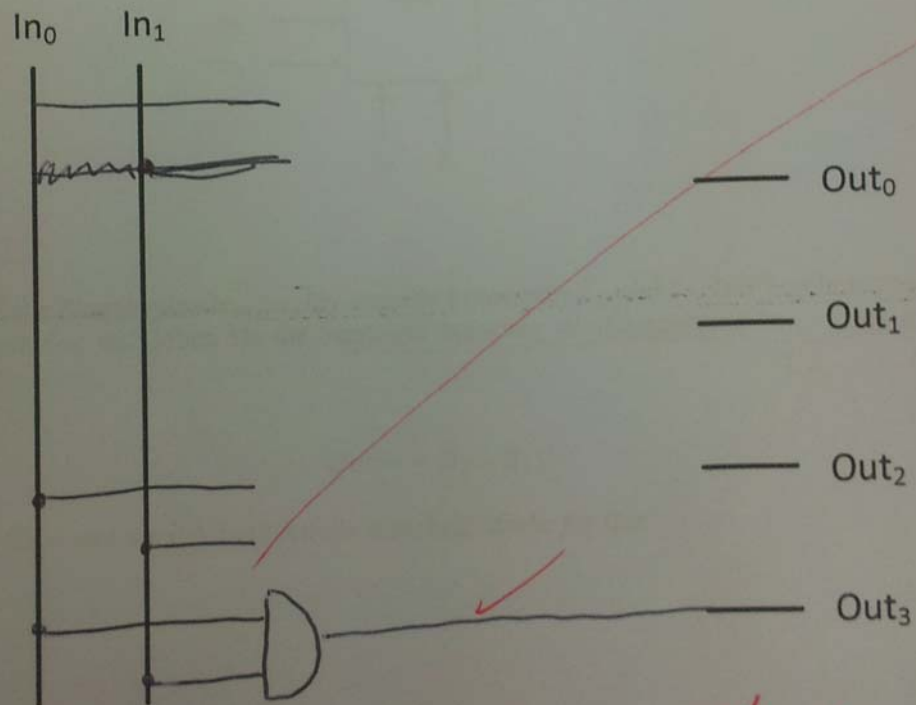
**Aufgabe 5: Logische Bausteine: Multiplexer und Decoder**

(23 Pkt.)

- a. Gegeben sei die folgende Wahrheitstabelle eines 2-zu-4-Decoders, der zwei Eingabepins  $In_0$  und  $In_1$  und vier Ausgabepins  $Out_0$ ,  $Out_1$ ,  $Out_2$  und  $Out_3$  besitzt:

$In_0$	$In_1$	$Out_0$	$Out_1$	$Out_2$	$Out_3$
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Zeichnen Sie das zugehörige Schaltnetz, indem Sie die folgende Vorlage ergänzen. Verwenden Sie dazu ausschließlich Bausteine vom Typ UND, ODER und NICHT:



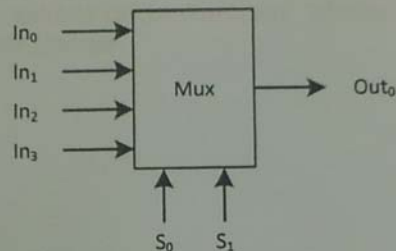
1/4 P

- b. Ausgehend von einem  $n$ -Eingaben-Multiplexer, wieviele Steuereingaben werden im Allgemeinen benötigt? Wieviele Parameter hat im Allgemeinen die zugehörige boolesche Funktion? Begründen Sie jeweils Ihre Antwort.

$n = 2^m$   $m$  Steuereingaben  $\Rightarrow$  noch  $m$  auf  $10x$

Die boolesche Funktion hat ? Parameter. Das 0/2P  
liegt daran das ~~ein Multiplexer aus  $n$  Multiplexern rekursiv aufgebaut sind.~~

- c. Ein 4-Eingaben-Multiplexer (4-Mux) wird auch durch folgendes Schaltsymbol abgekürzt:



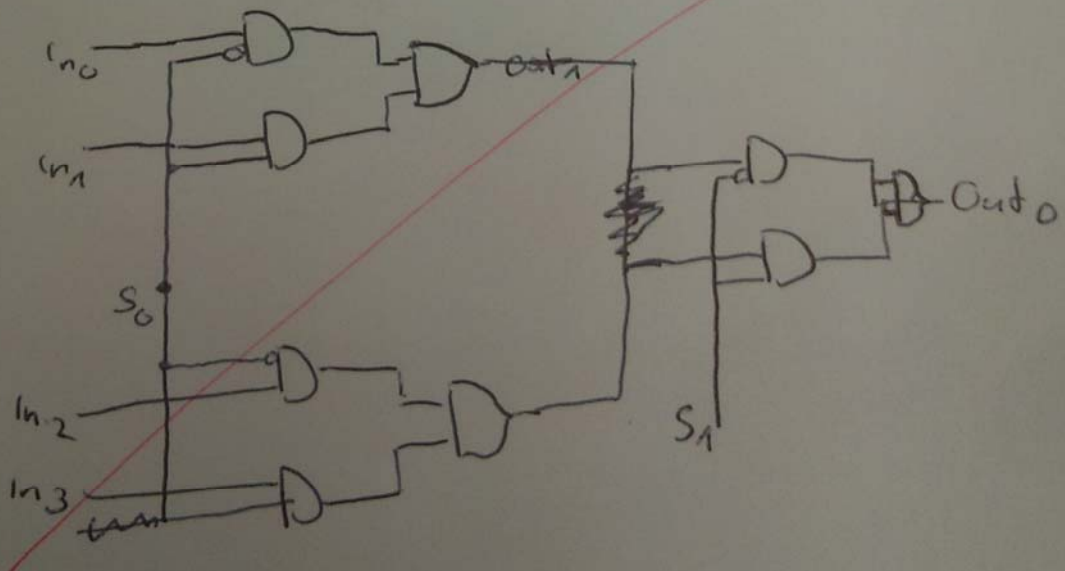
Er besitzt die Eingabepins  $In_0, \dots, In_3$  und die Steuerpins  $S_0$  und  $S_1$ . Das Ergebnis liegt am Ausgabepin  $Out$  an. Geben Sie die benötigte Belegung der Eingabepins  $In_0, \dots, In_3$  an, so dass gilt:

$\hookrightarrow$  Wo?

$$Out = \neg(S_0 \wedge S_1)$$

Hinweis: Überlegen Sie sich zunächst die Wahrheitstabelle für  $Out$ .

0/2P

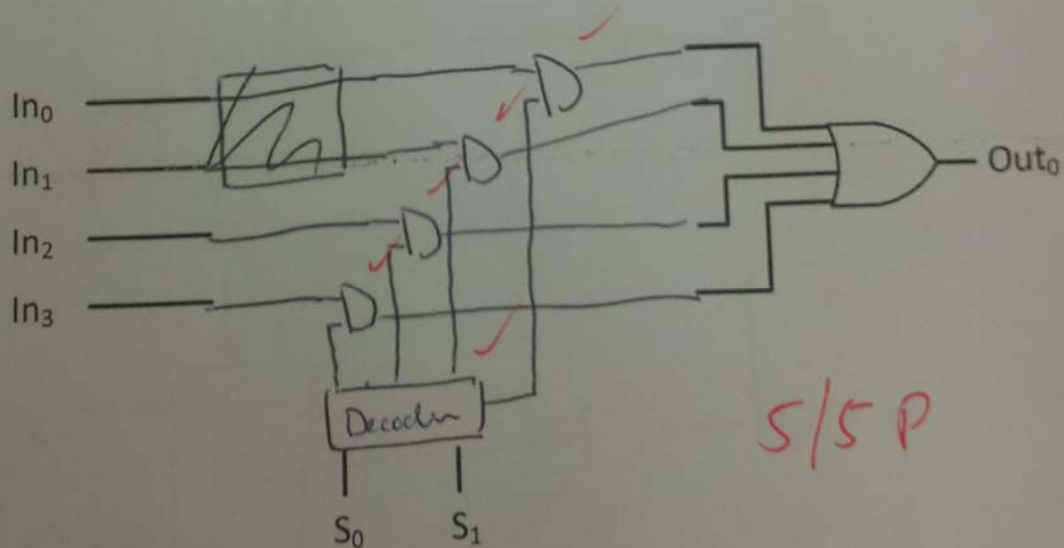


- d. Geben Sie allgemein die boolesche Schaltfunktion für den Ausgabepin Out eines 4-Eingaben-Multiplexers mit den Eingabepins  $In_0, In_1, In_2, In_3$  und den Steuerpins  $S_0, S_1$  an:

$$(In_0 \bar{S}_0 + In_1 S_0) \cdot \bar{S}_1 + S_1 \cdot (\bar{S}_0 In_2 + S_0 In_3)$$

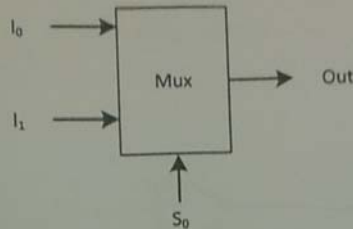
4/4 P

- e. Zeichnen Sie das Schaltnetz eines 4-Eingaben-Multiplexers. Verwenden Sie dabei ausschließlich die Bausteine vom Typ UND, ODER und NICHT sowie zwingend einen 2-zu-4-Decoder Baustein, der durch sein Schaltbild (nicht durch sein Schaltnetz!) dargestellt werden soll. Ergänzen Sie hierzu folgende Vorlage:

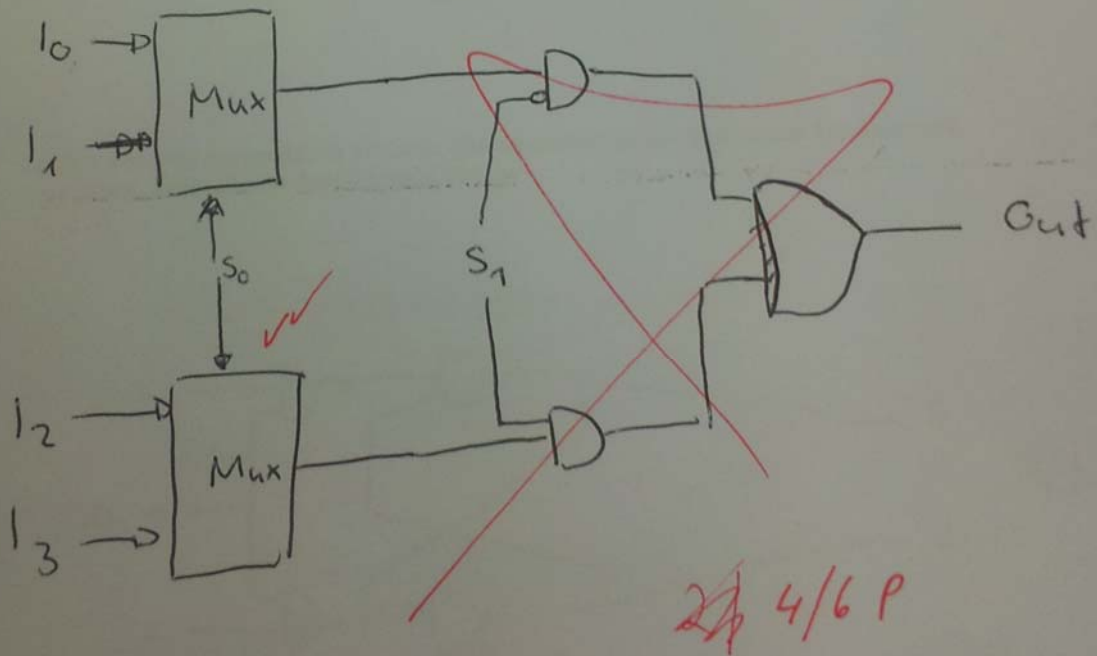


5/5 P

- f. Entwerfen Sie das Schaltnetz eines 4-Eingaben-Multiplexers, das lediglich aus 2-Eingaben-Multiplexern zusammengesetzt sein soll. Verwenden Sie hierfür das folgende Schaltsymbol:



Der zu entwerfende 4-Eingaben-Multiplexer besitzt die Eingänge  $I_0, I_1, I_2, I_3$ , die Steuerleitungen  $S_0$  und  $S_1$  sowie das Ausgangspint  $Out$ :



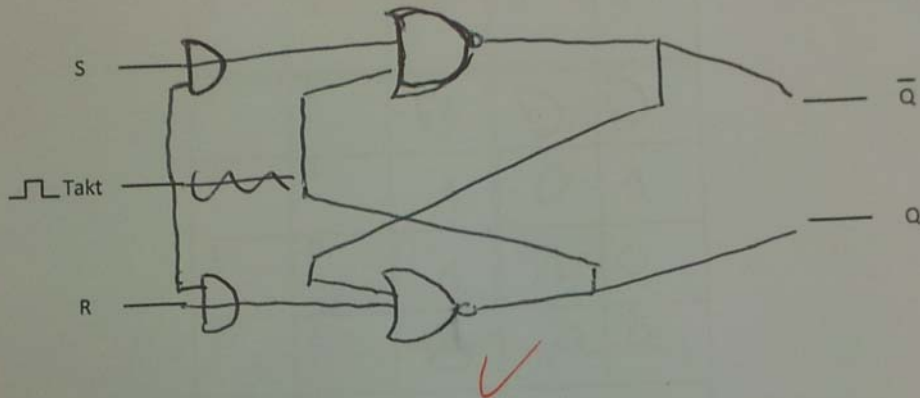


## Aufgabe 6: Schaltwerke und 1-Bit-Speicher

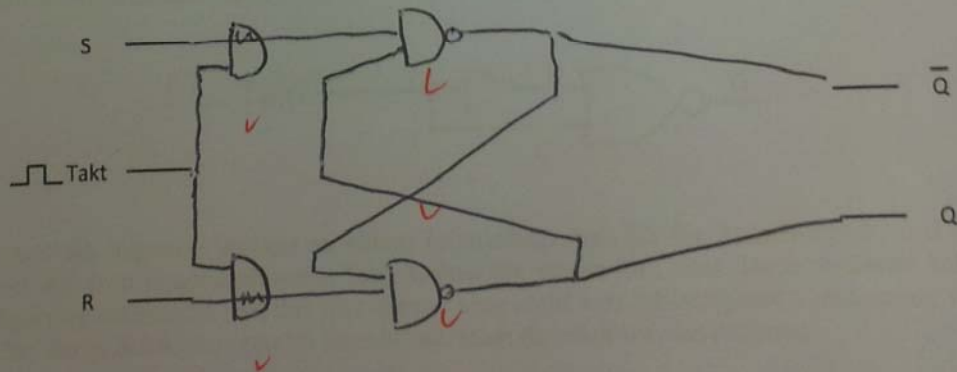
(18 Pkt.)

Bearbeiten Sie die folgenden Teilaufgaben zum Thema Schaltwerke und 1-Bit-Speicher:

- a. Zeichnen Sie das Schaltnetz eines getakteten SR-Latch, indem Sie folgende Vorlage ergänzen. Verwenden Sie dabei ausschließlich Bausteine vom Typ AND, OR, NOR und NOT:



- b. Zeichnen Sie nun ein getaktetes SR-Latch, das ausschließlich auf Gattern vom Typ NAND und AND basiert. Ergänzen Sie dazu folgende Vorlage:

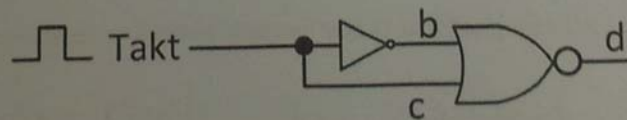


- c. Geben Sie die Zustandstabelle eines getakteten SR-Latch an. Geben Sie dabei die Belegung aller Ausgänge und Eingänge an, indem Sie folgende Tabelle ausfüllen. Verwenden Sie *Don't-Care-Argumente*, falls es keinen Unterschied für die Belegung der Ausgänge macht, ob an einem Eingang eine 0 oder 1 anliegt. Verwenden Sie zur Kennzeichnung solcher Belegungen in der Tabelle ein D. Verwenden Sie die Notation  $Q^*$  und  $\bar{Q}^*$ , um den alten Wert von Q bzw.  $\bar{Q}$  zu symbolisieren, falls dieser in diesem Zustand nicht explizit (0 oder 1) bekannt ist. Kennzeichnen Sie unzulässige Eingaben als solche. Einige Einträge sind schon vorgegeben:

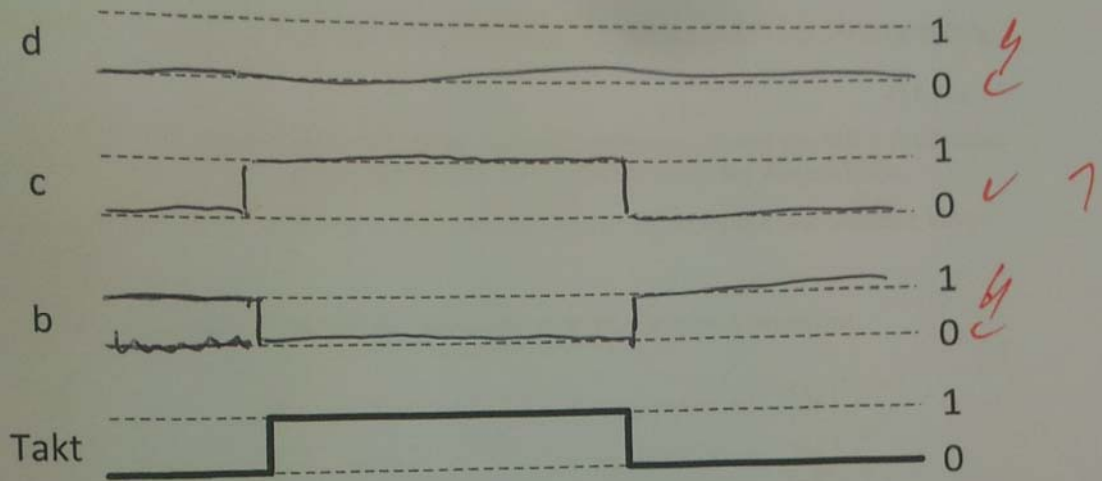
S	R	Takt	Q	$\bar{Q}$
0	0	<del>D</del>	<del>D</del>	<del>D</del>
0	1	1	0	1
1	0	1	1	0
1	1	<del>D</del>	<del>1</del>	<del>0</del>

7,5

- d. Gegeben sei folgendes Schaltnetz, welches einen Impulsgenerator realisiert, der aus Taktflanken kurze Impulse erzeugt:



Ergänzen Sie folgende Vorlage zu einem Impulsdiagramm für die Ausschnitte b, c, d basierend auf dem eingezeichnetem Takt. Gehen Sie davon aus, dass das NOR-Gatter keine Verzögerung verursacht und das NOT-Gatter eine nicht vernachlässigbare Verzögerung verursacht, deren Auswirkungen im Impulsdiagramm deutlich werden müssen:



- e. Für welche Art von D-Flip-Flops kann der aus Teilaufgabe d bekannte Impulsgenerator ohne weitere Modifikationen eingesetzt werden? Begründen Sie Ihre Antwort.

0

- f. Nennen Sie einen Vorteil und einen Nachteil von D-Latches gegenüber SR-Latches.

Vorteil: sie sind im Zustand  $S=R=\emptyset$  bestimmt.

0

~~SR~~  $\$$

$\frac{1}{1} = 1$

# Aufgabe 7: Assemblerprogrammierung des MIPS-Prozessors (SPIM)

(15 Pkt.)

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

- a. Gegeben sei folgendes Programm in MIPS-Assembler, wo schon einige Kommentare eingefügt sind:

```

1 .data
2 sMsg: .asciiz "\nAktueller Wert:\t: " # Für Ergebnisanzeige
3
4 .text
5 main: li      $t0, 10                # Kommentar Nr.: (vi) ✓
6
7 while: blez   $t0, elihw            # Kommentar Nr.: (ii) ✓
8         addi  $t0, $t0, -1          # Kommentar Nr.: (viii) ✓
9         addi  $sp, $sp, -4          # Kommentar Nr.: (v) ✓
10        sw    $t0, 4($sp)           # Kommentar Nr.: (vii) ✓
11        move  $a0, $t0              # Kommentar Nr.: (iii) ✓
12        jal   print                 # Unterprozeduraufruf
13        lw    $t0, 4($sp)           # Kommentar Nr.:
14        addi  $sp, $sp, 4           # Kommentar Nr.:
15        b     while                # Schleife wiederholen
16
17 elihw: li $v0, 10
18        syscall                     # EXIT des Programms
19
20 print: move   $t0, $a0              # Kommentar Nr.: (i) ✓
21         li    $v0, 4                # print_string
22         la    $a0, sMsg
23         syscall
24         move  $a0, $t0              # Kommentar Nr.: (iv) ✓
25         li    $v0, 1                # print_int
26         syscall
27         jr    $ra                  # Rücksprung

```

2/4,5

Ordnen Sie in diesem Programm jeder Zeile, die mit "# Kommentar Nr.:" versehen ist, den jeweils sinnvollsten der folgenden Kommentare zu. Ein Kommentar kann auch mehrfach benötigt werden. Schreiben Sie dazu in obigem Code die Nummer des sinnvollsten Kommentars aus folgender Auswahl hinter das jeweilige "# Kommentar Nr.:":

- (i) Schaffe Platz auf Stack
- (ii) while i > 0
- (iii) ~~übergebe Argument~~
- (iv) Sichere i auf Stack
- (v) Lade i vom Stack
- (vi) i := 10
- (vii) Sichere übergebenes Argument
- (viii) i := i - 1
- (ix) Setze Stackgröße zurück



- b. Geben Sie die letzten drei Zeilen an, die das Programm aus Teilaufgabe a) auf der Konsole ausgibt.

...  
Aktueller wert : 2 ✓

Aktueller wert : 1 ✓

1/1,5

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

```
1      li $t0, 5
2      li $t1, 5
3      li $t2, 5
4      addi $sp, -16
5      sw $t0, 12($sp)
6      sw $t1, 8($sp)
7      sw $t2, 4($sp)
```

Erläutern Sie, welchen Nachteil diese Befehlssequenz bezüglich des Stacks besitzt und geben Sie die dafür verantwortliche Zeile an. Welche Änderung wäre zur Behebung des Problems nötig?

0/2

- b. Geben Sie die letzten drei Zeilen an, die das Programm aus Teilaufgabe a) auf der Konsole ausgibt.

...  
Aktueller wert : 2 ✓

Aktueller wert : 1 ✓

1/1,5

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

```
1      li $t0, 5
2      li $t1, 5
3      li $t2, 5
4      addi $sp, -16
5      sw $t0, 12($sp)
6      sw $t1, 8($sp)
7      sw $t2, 4($sp)
```

Erläutern Sie, welchen Nachteil diese Befehlssequenz bezüglich des Stacks besitzt und geben Sie die dafür verantwortliche Zeile an. Welche Änderung wäre zur Behebung des Problems nötig?

/

0/2

- d. Im Folgenden soll ein MIPS-Assembler Programm vervollständigt werden, das eine Unterprozedur `fac` aufruft, welche die Fakultät des in `$a0` übergebenen Wertes berechnet und das Ergebnis in `$v0` zurückliefert. Die Berechnung erfolgt rekursiv. Ergänzen Sie folgenden Coderaum um insgesamt 4 Zeilen Code, so dass ein Programm mit gültiger Syntax und der beschriebenen Funktionalität entsteht. Nur an den mit Lösung markierten Stellen darf Code eingefügt werden:

```


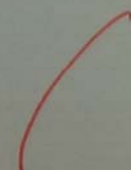
1  .data
2
3  .text
4  main:
5      li      $a0,    5           # $a0 := 5
6      jal     fac           # fac($a0)
7
8      move    $a0,     $v0       # Ergebnis ausdrucken
9      li      $v0,     1         # 1: print_int
10     syscall
11     li      $v0,     10        # EXIT des Programms
12     syscall
13
14 fac:   blez    $a0,     fertig   # if $a0 <= 0 return 1
15
16       addi    $sp,     $sp,     -8
17       sw      $a0,     8($sp)
18       sw      $ra,     4($sp)
19
20       addi    $a0,     $a0,     -1
21       jal     fac           # $v0 = fac($a0 - 1)
22
23       # Begin Lösung Teil 1)
24
25
26
27
28
29
30
31
32
33
34       # Ende Lösung Teil 1)
35
36       mul     $v0,     $v0,     $a0   # $v0 = fac($a0 - 1) * $a0
37       jr      $ra
38
39 fertig:
40
41
42
43       jr      $ra

```

0/7

- d. Im Folgenden soll ein MIPS-Assembler Programm vervollständigt werden, das eine Unterprozedur `fac` aufruft, welche die Fakultät des in `$a0` übergebenen Wertes berechnet und das Ergebnis in `$v0` zurückliefert. Die Berechnung erfolgt rekursiv. Ergänzen Sie folgenden Coderahmen um insgesamt 4 Zeilen Code, so dass ein Programm mit gültiger Syntax und der beschriebenen Funktionalität entsteht. Nur an den mit *Lösung* markierten Stellen darf Code eingefügt werden:

```
1  .data
2
3  .text
4  main:
5      li      $a0,    5           # $a0 := 5
6      jal     fac           # fac($a0)
7
8      move    $a0,    $v0        # Ergebnis ausdrucken
9      li      $v0,    1         # 1: print_int
10     syscall
11     li      $v0,    10
12     syscall                # EXIT des Programms
13
14 fac:  blez    $a0,    fertig    # if $a0 <= 0 return 1
15
16      addi    $sp,    $sp,    -8
17      sw      $a0,    8($sp)
18      sw      $ra,    4($sp)
19
20      addi    $a0,    $a0,    -1
21      jal     fac           # $v0 = fac($a0 - 1)
22
23      # Begin Lösung Teil 1)
24
25
26
27
28
29
30
31
32
33
34      # Ende Lösung Teil 1)
35
36      mul     $v0,    $v0,    $a0  # $v0 = fac($a0 - 1) * $a0
37      jr      $ra
38
39 fertig:
40
41
42
43      jr      $ra           # Lösung Teil 2)
```



0/2



## 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	$Rd := Rs1 \text{ DIV } Rs2$
rem	Rd, Rs1, Rs2	$Rd := Rs1 \text{ MOD } Rs2$
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)
xori	Rd, Rs1, Imm	$Rd := Rs1 \oplus Imm$ (bitweises XOR)
syscall		führt Systemfunktion aus
move	Rd, Rs	$Rd := Rs$
la	Rd, label	Adresse des Labels wird in Rd geladen
lb	Rd, Adr	$Rd := \text{MEM}[\text{Adr}]$
lw	Rd, Adr	$Rd := \text{MEM}[\text{Adr}]$
li	Rd, Imm	$Rd := Imm$
sw	Rs, Adr	$\text{MEM}[\text{Adr}] := Rs$ (Speichere ein Wort)
sh	Rs, Adr	$\text{MEM}[\text{Adr}] \text{ MOD } 2^{16} := Rs$ (Speichere ein Halbwort)
sb	Rs, Adr	$\text{MEM}[\text{Adr}] \text{ MOD } 256 := Rs$ (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