EvaExam	Prüfung Informationstechnik I+II	Electric Paper

Prof. Dr.-Ing. Birgit Vogel-Heuser

Bitte so markieren: ☐ ☒ ☐ ☐ ☐ Bitte verwenden Sie einen Kugelschreiber oder nicht zu starken Filzstift. Dieser Fragebogen wird maschinell erfasst.

Korrektur: ☐ ■ ☐ ☒ ☐ Bitte beachten Sie im Interesse einer optimalen Datenerfassung die links gegebenen Hinweise beim Ausfüllen.

Vorname: _____

Nachname: _____

Für die eindeutige Zuordung der Prüfung übertragen Sie bitte Ihre **Matrikelnummer (= Prüfungsteilnehmer-ID) mit vorangestellter "0"** gewissenhaft in die dafür vorgesehenen Felder. Alle Seiten sind vollständig individualisiert und nicht mit anderen Prüfungen tauschbar.

Prüfungsteilnehmer-ID für den Prüfungsbogen Nr.: 0:

0	☐	☐	☐	☐	☐	☐	☐	☐
1	☐	☐	☐	☐	☐	☐	☐	☐
2	☐	☐	☐	☐	☐	☐	☐	☐
3	☐	☐	☐	☐	☐	☐	☐	☐
4	☐	☐	☐	☐	☐	☐	☐	☐
5	☐	☐	☐	☐	☐	☐	☐	☐
6	☐	☐	☐	☐	☐	☐	☐	☐
7	☐	☐	☐	☐	☐	☐	☐	☐
8	☐	☐	☐	☐	☐	☐	☐	☐
9	☐	☐	☐	☐	☐	☐	☐	☐

1. G: Umrechnung zwischen Zahlensystemen

Überführen Sie die unten angegebenen Zahlen in die jeweils anderen Zahlensysteme.
Wichtig: Achten Sie genau auf die jeweils angegebene Basis!

Hinweis: Nur Lösungen innerhalb der angegebenen Ergebnisfelder werden gewertet! Lassen Sie etwaige leere Stellen frei.

1.1 $(101\ 110\ 011)_2 = (???)_{16}$

Ergebnis: 173 || Ergebnis: 0173 || Ergebnis: 00173

Ergebnis: ☐☐☐☐☐

1.2 $(356)_{10} = (???)_{12}$

Ergebnis: 258 || Ergebnis: 0258 || Ergebnis: 00258 ||

Ergebnis: ☐☐☐☐☐

1.3 $(6{,}03125)_{10} = (???)_2$

Ergebnis: 110,00001 || Ergebnis: 110,000010 || Ergebnis: 110,0000100 || Ergebnis: 110,00001000

Ergebnis: ☐☐☐,☐☐☐☐☐☐☐

2. G: IEEE 754 Gleitkommazahlen

Rechnen Sie die gegebene Gleitkommazahl (angelehnt an die IEEE 754 Darstellung) in eine Dezimalzahl um.

0	1	0	0	1	1	1	0	0	0
V	e (4 Bit)				M (5 Bit)				

Hinweis: Zwischenergebnisse und Nebenrechnungen werden nicht gewertet. Nur Lösungen innerhalb der angegebenen Ergebnisfelder werden gewertet! Lassen Sie etwaige leere Stellen frei.

2.1 Vorzeichen (V) ☑ V = + ☐ V = -
Hinweis: Nur Einfachnennung möglich.

2.2 Bias (B, Dezimalzahl)

Ergebnis: 7 || Ergebnis: 07 || Ergebnis: 007 || Ergebnis: 0007 || Ergebnis: 00007

Ergebnis:

2.3 Biased Exponent (e, Dezimalzahl)

Ergebnis: 9 || Ergebnis: 09 || Ergebnis: 009 || Ergebnis: 0009 || Ergebnis: 00009

Ergebnis:

2.4 Exponent ohne Vorzeichen (E, Dezimalzahl)

Ergebnis (ohne Vorzeichen): 2 || Ergebnis (ohne Vorzeichen): 02 || Ergebnis (ohne Vorzeichen): 002 || Ergebnis (ohne Vorzeichen): 0002 || Ergebnis (ohne Vorzeichen): 00002

Ergebnis (ohne Vorzeichen):

2.5 Vorzeichen des Exponenten E ☑ sign(E) = + ☐ sign(E) = -
Hinweis: Nur Einfachnennung möglich.

2.6 Mantisse (M_2, Dualzahl und Denormalisiert)

Ergebnis: 111,000 || Ergebnis: 111,0000 || Ergebnis: 111,00000 || Ergebnis: 111,000000 || Ergebnis: 111,0000000

Ergebnis: ,

2.7 Vollständige Dezimalzahl (Z, Dezimalzahl)

Ergebnis: 7, || Ergebnis: 7,0 || Ergebnis: 7,00 || Ergebnis: 7,000 || Ergebnis: 07, || Ergebnis: 07,0 || Ergebnis: 07,00 || Ergebnis: 07,000 || Ergebnis: 007, || Ergebnis: 007,0 || Ergebnis: 007,00 || Ergebnis: 007,000

Ergebnis: ,

3. G: Logische Schaltungen und Schaltbilder

Sie sind zuständig für die Überprüfung der Korrektheit von Schaltungen für die sicherheitstechnische Auslegung einer Anlage. Ein Mitarbeiter legt Ihnen Schaltung 1 und eine minimierte Schaltung 2 vor (siehe **Bild G-3.1**). Sie müssen überprüfen, ob beide Schaltungen das selbe Schaltungsverhalten haben.

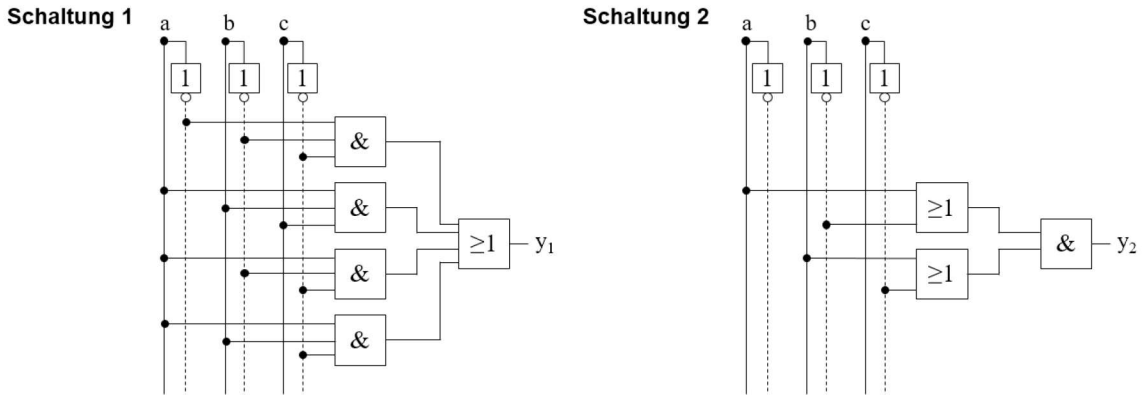


***Bild G-3.1**: Schaltungen 1 und 2*

Entscheiden Sie hierzu für die Eingaben in der Wahrheitstabelle (**Nummern 1 bis 8 in Bild G-3.2**), welche Ausgaben y_1 bzw. y_2 die Schaltungen produzieren und schreiben sie diese (0 oder 1) in die Lösungsfelder.

Hinweis: Nur Lösungen innerhalb der angegebenen Ergebnisfelder werden gewertet!

Nr.	a	b	c	y_1	y_2
1	0	0	0		
2	0	0	1		
3	0	1	0		
4	0	1	1		
5	1	0	0		
6	1	0	1		
7	1	1	0		
8	1	1	1		

***Bild G-3.2**: Wahrheitstabelle*

3.1 y_1 und y_2 für **Nr. 1 in Bild G-3.2**

y1 = 1, y2 = 1

y1 = [ ], y2 = [ ]

3.2 y_1 und y_2 für **Nr. 2 in Bild G-3.2**

y1 = 0, y2 = 0

y1 = [ ], y2 = [ ]

3.3 y_1 und y_2 für **Nr. 3 in Bild G-3.2**

y1 = 0, y2 = 0

y1 = [ ], y2 = [ ]

3.4 y_1 und y_2 für **Nr. 4 in Bild G-3.2**

y1 = 0, y2 = 0

y1 = [ ], y2 = [ ]

3.5 y_1 und y_2 für **Nr. 5 in Bild G-3.2**

y1 = 1, y2 = 1

y1 = [ ], y2 = [ ]

3.6 y_1 und y_2 für **Nr. 6 in Bild G-3.2**

y1 = 0, y2 = 0

y1 = [ ], y2 = [ ]

3.7 y_1 und y_2 für **Nr. 7 in Bild G-3.2**

y1 = 1, y2 = 1

y1 = [ ], y2 = [ ]

3.8 y_1 und y_2 für **Nr. 8 in Bild G-3.2**

y1 = 1, y2 = 1

y1 = [ ], y2 = [ ]

3.9 Ist das Schaltungsverhalten von Schaltung 1 und Schaltung 2 identisch?
Hinweis: Nur Einfachnennung möglich.

- [x] Das Schaltungsverhalten ist identisch.
- [ ] Das Schaltungsverhalten ist NICHT identisch.
- [ ] Es ist keine eindeutige Antwort möglich.

4. G: Normalformen und Minimierung

Gegeben ist folgende Wahrheitstabelle (**Bild G-4.1**) sowie das folgende leere KV-Diagramm (**Bild G-4.2**).

a	b	c	y
0	0	0	1
0	0	1	X
0	1	0	1
0	1	1	X
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

***Bild G-4.1**: Wahrheitstabelle*

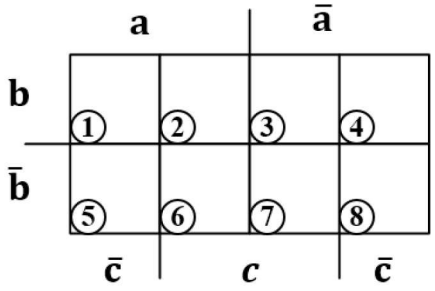


***Bild G-4.2**: KV-Diagramm*

Geben Sie für die im **KV-Diagramm (Bild G-4.2) angegebenen Felder 1 bis 8** an, welchen Wert die Ausgangsvariable y aus der Wahrheitstabelle annimmt. Die Ausgänge mit y = „X“ sind don‘t care bits.

Hinweis: Nur Lösungen innerhalb der angegebenen Ergebnisfelder werden gewertet!

4.1 Feld **Nr. 1 in Bild G-4.2**
y = 1
y =

4.2 Feld **Nr. 2 in Bild G-4.2**
y = 0
y =

4.3 Feld **Nr. 3 in Bild G-4.2**
y = X || y = x
y =

4.4 Feld **Nr. 4 in Bild G-4.2**
y = 1
y =

4.5 Feld **Nr. 5 in Bild G-4.2**
y = 1
y =

4.6 Feld **Nr. 6 in Bild G-4.2**
y = 1
y =

4.7 Feld **Nr. 7 in Bild G-4.2**
y = X || y = x
y =

4.8 Feld **Nr. 8 in Bild G-4.2**
y = 1
y =

4.9 Geben Sie die minimierte Form in DNF (*Disjunktiver Normalform*) Boolescher Algebra an.

y_min = b-quer v c-quer

5. G: Flip-Flops

Gegeben ist die folgende Master-Slave Flip-Flop Schaltung (MS-FF) (siehe **Bild G-5.1**).

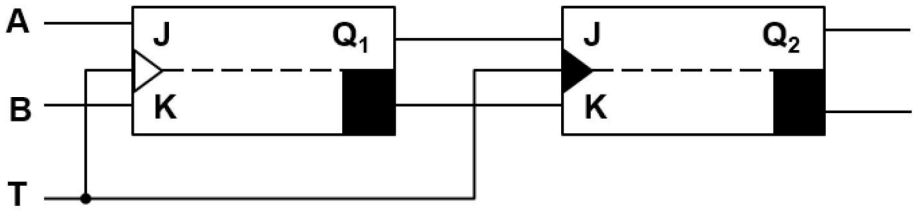


***Bild G-5.1**: Master-Slave Flip-Flop Schaltung (MS-FF)*

Bei t = 0 sind die Flip-Flops in folgendem Zustand:

$$Q_1(t=0) = Q_2(t=0) = 0$$

Analysieren Sie die Schaltung, indem Sie für die Eingangssignale A, B und T die zeitlichen Verläufe für Q_1 und Q_2 in die vorgegebenen Koordinatensysteme eintragen.

Hinweis: Signallaufzeiten können bei der Analyse vernachlässigt werden. Übertragen Sie anschließend Ihre Ergebnisse in die Ergebnisfelder. Nur Lösungen innerhalb der angegebenen Ergebnisfelder werden gewertet! **Tragen Sie bei einem Zustandswechsel immer den neuen Zustand ein.**

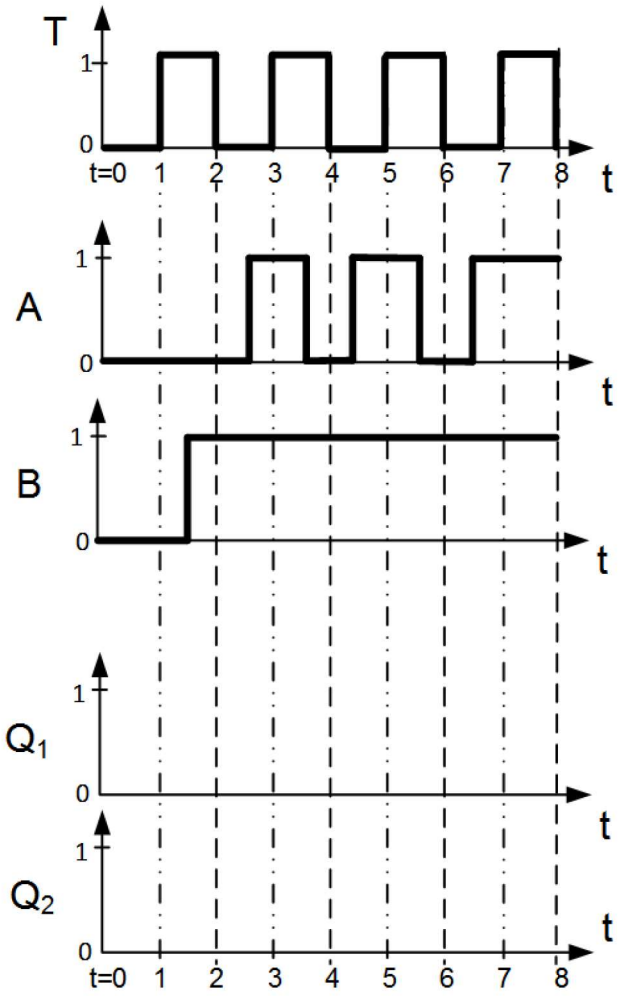


***Bild G-5.2**: Signallaufzeiten*

5.1 Q_1 und Q_2 bei **t = 1 in Bild G-5.2**
Q1 = 0, Q2 = 0
Q1 = [ ], Q2 = [ ]

5.2 Q_1 und Q_2 bei **t = 2 in Bild G-5.2**
Q1 = 0, Q2 = 0
Q1 = [ ], Q2 = [ ]

5.3 Q_1 und Q_2 bei **t = 3 in Bild G-5.2**
Q1 = 1, Q2 = 0
Q1 = [ ], Q2 = [ ]

5.4 Q_1 und Q_2 bei **t = 4 in Bild G-5.2**
Q1 = 1, Q2 = 1
Q1 = [ ], Q2 = [ ]

5.5 Q_1 und Q_2 bei **t = 5 in Bild G-5.2**
Q1 = 0, Q2 = 1
Q1 = [ ], Q2 = [ ]

5.6 Q_1 und Q_2 bei **t = 6 in Bild G-5.2**
Q1 = 0, Q2 = 1
Q1 = [ ], Q2 = [ ]

5.7 Q_1 und Q_2 bei **t = 7 in Bild G-5.2**
Q1 = 1, Q2 = 0
Q1 = [ ], Q2 = [ ]

5.8 Q_1 und Q_2 bei **t = 8 in Bild G-5.2**
Q1 = 1, Q2 = 1
Q1 = [ ], Q2 = [ ]

6. G: MMIX-Rechner

Im Registerspeicher eines MMIX-Rechners (**Bild G-6.1**) befinden sich die in **Bild G-6.2** gegebenen Werte. Es sollen nacheinander die vier Befehle (**Bild G-6.4**) abgearbeitet und das Ergebnis in dem Registerspeicher (**Bild G-6.2**) bzw. Datenspeicher (**Bild G-6.3**) abgelegt werden.

	0x_0	0x_1	…	0x_4	0x_5	…
	0x_8	0x_9	…	0x_C	0x_D	
0x0_	TRAP	FCMP	…	FADD	FIX	…
	FLOT	FLOT I		SFLOT	SFLOT I	
0x1_	FMUL	FCMPE	…	FDIV	FSQRT	…
	MUL	MUL I		DIV	DIV I	
0x2_	ADD	ADD I	…	SUB	SUB I	…
	2ADDU	2ADDU I		8ADDU	8ADDU I	
…	…	…	…	…	…	…
0x8_	LDB	LDB I	…	LDW	LDW I	…
	LDT	LDT I		LDO	LDO I	
0x9_	LDSF	LDSF I	…	CSWAP	CSWAP I	…
	LDVTS	LDVTS I		PREGO	PREGO I	
0xA_	STB	STB I	…	STW	STW I	…
	STT	STT I		STO	STO I	
…	…	…	…	…	…	…
0xE_	SETH	SETMH	…	INCH	INCMH	…
	ORH	ORMH		ANDNH	ANDNMH	
0xF_	JMP	JMP B	…	GETA	GETA B	…
	POP	RESUME		SYNC	SWYM	

***Bild G-6.1**: MMIX-Code-Tabelle*

Registerspeicher	
Adresse	Wert vor Befehlsausführung
…	…
$0x9D	0x00 00 00 00 00 00 00 03
$0x9E	0x00 00 00 00 00 00 01 01
$0x9F	0x00 00 00 00 0B 12 11 05
$0xA0	0x00 00 00 00 00 00 61 00
…	…

***Bild G-6.2**: Registerspeicher*

Datenspeicher
Adresse
…
0x0..0 61 FE
0x0..0 61 FF
0x0..0 62 00
0x0..0 62 01
0x0..0 62 02
0x0..0 62 03
0x0..0 62 04
…

***Bild G-6.3**: Datenspeicher*

	Maschinensprache	Assemblersprache	Befehlsbeschreibung
1	0x18 9D 9D 9E	①	
2		STW $0x9F, $0x9E, $0xA0	②
3	③		$0x9E=$0x9F -0x07
4	④		

***Bild G-6.4**: Maschinensprache – Assemblersprache – Befehlsbeschreibung*

Bearbeiten Sie nun folgende Fragen zu den Befehlen und zu Änderungen, die sich durch die Befehle ergeben.

Hinweis: Kreuzen Sie die richtigen Antworten an oder schreiben Sie diese in die Lösungsfelder.

6. G: MMIX-Rechner [Fortsetzung]

6.1 Geben Sie für den in **Nr. 1 in Bild G-6.4** angegebenen Befehl an, wie dieser in Assemblersprache formuliert ist.
Hinweis: Nur Einfachnennung möglich.

- ☐ MUL $0x9D 0x9D 0x9E
- ☑ MUL $0x9D $0x9D $0x9E
- ☐ MUL $0x9D $0x9D 0x9E
- ☐ MULI $0x9D $0x9D 0x9E

6.2 In welcher Zelle des Registerspeichers ergeben sich durch den in **Nr. 1 in Bild G-6.4** angegebenen Befehl Änderungen?
Hinweis: Nur Einfachnennung möglich.

- ☑ 0x9D
- ☐ 0x9E
- ☐ 0x9F
- ☐ 0xA0

6.3 Geben Sie die letzten vier Stellen der durch den in **Nr. 1 in Bild G-6.4** angegebenen Befehl geänderten Registerspeicherzelle nach der Befehlsausführung an. Nur Lösungen innerhalb der angegebenen Ergebnisfelder werden gewertet!

Ergebnis: 0303

Ergebnis: [ ][ ][ ][ ]

6.4 Wie lautet die Befehlsbeschreibung des in **Nr. 2 in Bild G-6.4** angegebenen Befehls?
Hinweis: Nur Einfachnennung möglich.

- ☑ M_2[$0x9E + $0xA0] = $0x9F
- ☐ M_4[$0x9E + $0xA0] = $0x9F
- ☐ $0x9F = M_2[$0x9E + $0xA0]
- ☐ $0x9F = M_4[$0x9E + $0xA0]

6.5 Welche Zelle des Speicherbereichs ändert sich durch die Ausführung des in **Nr. 2 in Bild G-6.4** angegebenen Befehls?
Hinweis: Nur Einfachnennung möglich.

- ☐ Registerspeicherzelle $0x9D
- ☐ Registerspeicherzelle $0x9E
- ☐ Registerspeicherzelle $0x9F
- ☐ Datenspeicherzelle 0x0..0 61 FF
- ☑ Datenspeicherzelle 0x0..0 62 00
- ☐ Datenspeicherzelle 0x0..0 62 01

6.6 Wie lautet die Schreibweise des in **Nr. 3 in Bild G-6.4** angegebenen Befehls in Maschinensprache?
Hinweis: Nur Einfachnennung möglich.

- ☐ 0x24 9E 9F 07
- ☐ 0x24 9F 9E 07
- ☐ 0x25 9F 9E 07
- ☑ 0x25 9E 9F 07

6.7 Durch den in **Nr. 4 in Bild G-6.4** angegebenen Befehl soll in der Datenspeicherzelle $0xA0 folgender Wert nach der Befehlsausführung erreicht werden:
0x00 00 12 34 00 00 00 00
Welcher Befehl eignet sich dafür? Vervollständigen Sie den Befehlscode in Maschinensprache. Nur Lösungen innerhalb der angegebenen Ergebnisfelder werden gewertet!

Ergebnis: 0xE1

Ergebnis: 0x[ ][ ]

6.8 Wie lauten die passenden Parameter für den in **Nr. 4 in Bild G-6.4** angegebenen Befehl, um das in Frage 6.7 genannte Ziel zu erreichen?
Hinweis: Nur Einfachnennung möglich.

- ☐ $0xA0 $0x12 $x34
- ☑ $0xA0 0x12 0x34
- ☐ $0x12 $0x34 0xA0
- ☐ 0x12 0x34 $0xA0

7. BS: Asynchrone Programmierung

Die zwei periodischen Prozesse PR2 und PR3 sowie der asynchrone Prozess PR1 sollen mit dem Verfahren der asynchronen Programmierung **präemptiv** auf einem Einkernprozessor eingeplant werden. Der Prozess PR1 besitzt die höchste, der Prozess PR3 die niedrigste Priorität. Die Ausführung wird durch einen Interrupt unterbrochen.
Tragen Sie in das unter der Abbildung (siehe **Bild BS-7.1**) angegebene leere Diagramm (siehe **Bild BS-7.2**) den Verlauf der Abarbeitung von Programmen und Interrupts ein. Kreuzen Sie in den darauffolgenden Fragen die aktiven Prozesse zu den in **Bild BS-7.2 angegebenen Zeitpunkten mit den Nummern 1 bis 6** an.

Hinweis: Es werden ausschließlich Ihre Antworten in den folgenden Fragen bewertet. Jeweils nur Einfachnennung möglich.

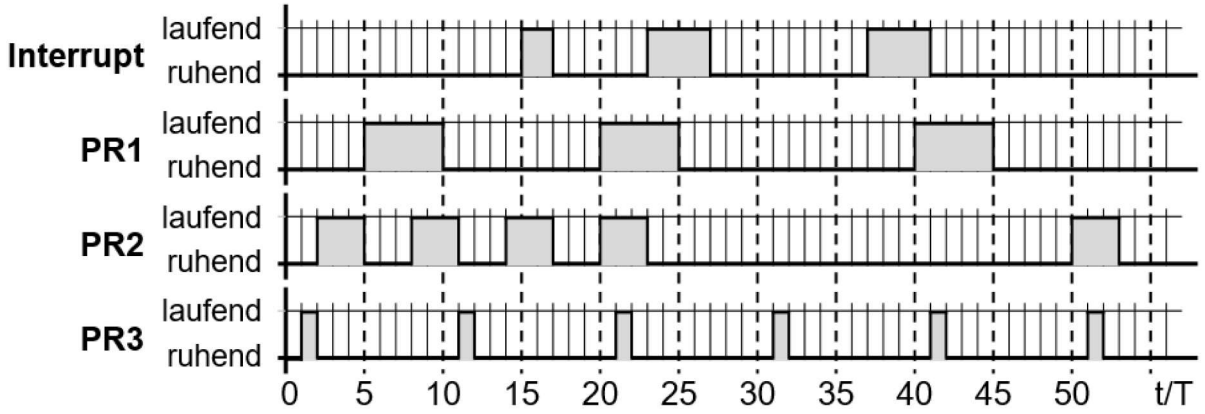


***Bild BS-7.1**: Asynchrone Programmierung*

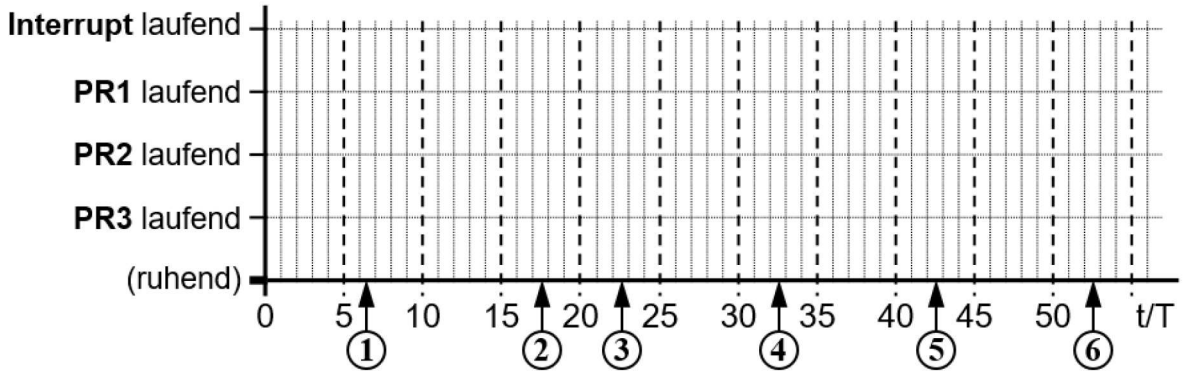


***Bild BS-7.2**: Verlauf der Abarbeitung von Programmen und Interrupts*

7.1	Prozess zum **Zeitpunkt Nr. 1 in Bild BS-7.2**	☐ Interrupt ☐ PR3	☑ PR1 ☐ ruhend	☐ PR2
7.2	Prozess zum **Zeitpunkt Nr. 2 in Bild BS-7.2**	☐ Interrupt ☐ PR3	☐ PR1 ☐ ruhend	☑ PR2
7.3	Prozess zum **Zeitpunkt Nr. 3 in Bild BS-7.2**	☐ Interrupt ☐ PR3	☑ PR1 ☐ ruhend	☐ PR2
7.4	Prozess zum **Zeitpunkt Nr. 4 in Bild BS-7.2**	☐ Interrupt ☑ PR3	☐ PR1 ☐ ruhend	☐ PR2
7.5	Prozess zum **Zeitpunkt Nr. 5 in Bild BS-7.2**	☐ Interrupt ☐ PR3	☑ PR1 ☐ ruhend	☐ PR2
7.6	Prozess zum **Zeitpunkt Nr. 6 in Bild BS-7.2**	☐ Interrupt ☐ PR3	☐ PR1 ☐ ruhend	☑ PR2

8. BS: Semaphoren

Gegeben ist die Anordnung von Semaphor-Operationen am Anfang und am Ende der Tasks A, B und C in **Bild BS-8.1**. Ermitteln Sie für die Fälle I und II in **Bild BS-8.2**, ob und in welcher Reihenfolge diese Tasks bei der angegebenen Initialisierung der Semaphor-Variablen ablaufen.

Hinweis: Die für die Ausführung eines Tasks notwendigen Semaphoren sollen nur im Block verwendet werden (z.B. Task A startet nur, wenn alle Semaphoren S2, S2, S3, S3 frei sind). Sind mehrere Tasks ablauffähig, gelten folgende Prioritäten: A: hoch, B: mittel, C: niedrig. P(Si) senkt Si um 1, V(Si) erhöht Si um 1.

Taskabarbeitung	Tasks A	Tasks B	Tasks C
↓	P(S2)		
	P(S2)	P(S1)	P(S3)
	P(S3)	P(S3)	
	P(S3)		
	…	…	…
	V(S3)	V(S2)	V(S1)
	V(S3)	V(S3)	V(S3)

***Bild BS-8.1**: Taskabarbeitung*

Fall I:

	Semaphoren		
Task	**S1**	**S2**	**S3**
Start	1	1	2

Fall II:

	Semaphoren		
Task	**S1**	**S2**	**S3**
Start	2	0	1

***Bild BS-8.2**: Initialisierung der Semaphor-Variablen für Fälle I und II*

Schreiben Sie nun die Reihenfolge der Tasks (in Großbuchstaben A, B und C) in die angegebenen Lösungsfelder, und machen Sie im Falle von Wiederholungen von Sequenzen keinen Strich über die Buchstaben.

Hinweis: Nur Lösungen innerhalb der angegebenen Ergebnisfelder werden gewertet!

8.1 Reihenfolge in **Fall I in Bild BS-8.2**

Ergebnis: BACBC

Ergebnis: | | | | | |

8.2 Reihenfolge in **Fall II in Bild BS-8.2**

Ergebnis: BBCBC

Ergebnis: | | | | | |

8.3 Welche der folgenden Aussagen für **Fälle I bzw. II in Bild BS-8.2** trifft zu?
Hinweis: Nur Einfachnennung möglich.

- ☐ In Fall I entsteht ein Deadlock, in Fall II ebenfalls.
- ☐ In Fall I wiederholt sich die gesamte Abfolge, in Fall II entsteht ein Deadlock.
- ☐ In Fall I sind nach einiger Zeit nur noch ein Teil der Tasks ablauffähig, in Fall II wiederholt sich die gesamte Folge.
- ☑ In Fall I wiederholt sich die gesamte Abfolge, in Fall II sind nach einiger Zeit nur noch ein Teil der Tasks ablauffähig.
- ☐ In Fall I wiederholt sich die gesamte Abfolge, in Fall II ebenfalls.

9. BS: Scheduling

Sechs Prozesse (P1 bis P6) sollen mit einem Einkernprozessor abgearbeitet werden. Das **Bild BS-9.1** zeigt die Zeiten, zu denen die Prozesse am Einkernprozessor eintreffen und die Ausführungszeiten der einzelnen Prozesse. Die Prozesse sollen zur Laufzeit mit unterschiedlichen Schedulingverfahren eingeplant werden. Alle Schedulingverfahren beginnen zum Zeitpunkt t = 0T. Für die Schedulingverfahren, bei denen Prioritäten berücksichtigt werden müssen, ist in **Bild BS-9.2** die entsprechende Prioritätenverteilung (Prioritäten 1 bis 4) gegeben.

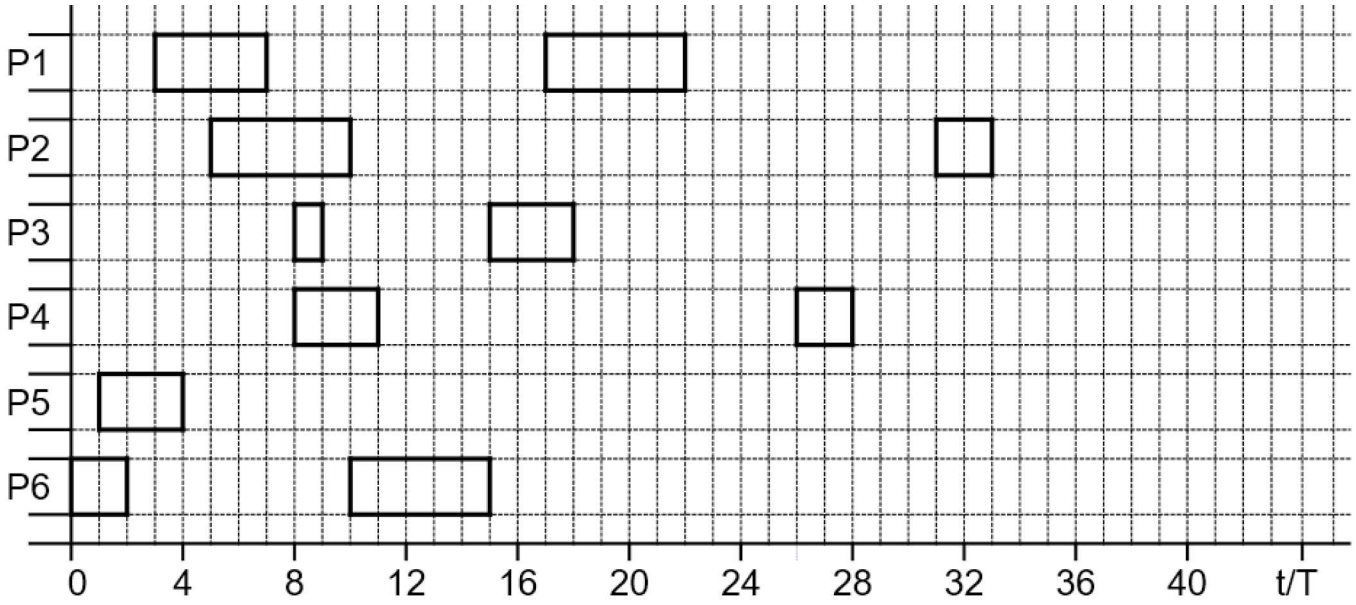


***Bild BS-9.1**: Sollzeitverlauf der Prozesse P1 bis P6*

Prozess	P1	P2	P3	P4	P5	P6
Priorität	1 (hoch)	2	2	3	3	4 (niedrig)

***Bild BS-9.2**: Prioritätenverteilung*

Im ersten Teil dieser Aufgabengruppe sollen die Prozesse **nicht-präemptiv** nach ihren Prioritäten eingeplant werden. Wenn keine andere Regel greift, gilt das Prinzip FIFO.
Tragen Sie den Verlauf der Prozessabarbeitung im nachfolgenden Schema ein. Kreuzen Sie in den darauffolgenden Fragen die aktiven Prozesse zu den in **Bild BS-9.3 angegebenen Zeitpunkten mit den Nummern 1 bis 4** an.

Hinweis: Es werden ausschließlich Ihre Antworten in den folgenden Fragen bewertet. Jeweils nur Einfachnennung möglich.

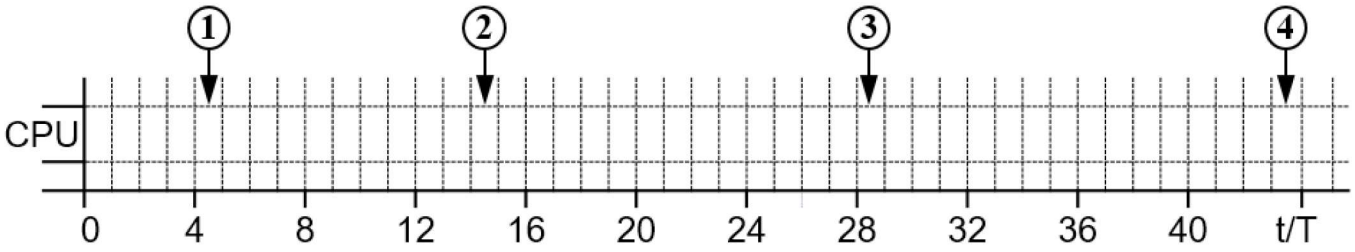


***Bild BS-9.3**: Aufgabenteil 1: nicht-präemptiv und FIFO*

9. BS: Scheduling [Fortsetzung]

9.1	Prozess zum **Zeitpunkt Nr. 1 in Bild BS-9.3**	☐ P1 ☐ P3 ☑ P5 ☐ ruhend	☐ P2 ☐ P4 ☐ P6
9.2	Prozess zum **Zeitpunkt Nr. 2 in Bild BS-9.3**	☐ P1 ☑ P3 ☐ P5 ☐ ruhend	☐ P2 ☐ P4 ☐ P6
9.3	Prozess zum **Zeitpunkt Nr. 3 in Bild BS-9.3**	☐ P1 ☐ P3 ☐ P5 ☐ ruhend	☐ P2 ☐ P4 ☑ P6
9.4	Prozess zum **Zeitpunkt Nr. 4 in Bild BS-9.3**	☐ P1 ☐ P3 ☐ P5 ☑ ruhend	☐ P2 ☐ P4 ☐ P6

Im zweiten Teil dieser Aufgabengruppe soll die Einplanung der Prozesse **präemptiv** mit festen Prioritäten erfolgen. Für jedes Prioritätslevel soll dabei ein eigenes Round-Robin-Verfahren angewendet werden. Für die Zeitscheiben soll angenommen werden, dass diese unendlich viele Schlitze besitzen und so zu jedem Zeitpunkt ausreichend freie Schlitze vorhanden sind. Die Zeitschlitze besitzen eine Länge von 3T. Restzeiten können nicht übersprungen werden.
Tragen Sie den Verlauf der Prozessabarbeitung im nachfolgenden Schema ein. Kreuzen Sie in den darauffolgenden Fragen die aktiven Prozesse zu den in **Bild BS-9.4 angegebenen Zeitpunkten mit den Nummern 1 bis 10** an.

Hinweis: Es werden ausschließlich Ihre Antworten in den folgenden Fragen bewertet. Jeweils nur Einfachnennung möglich.

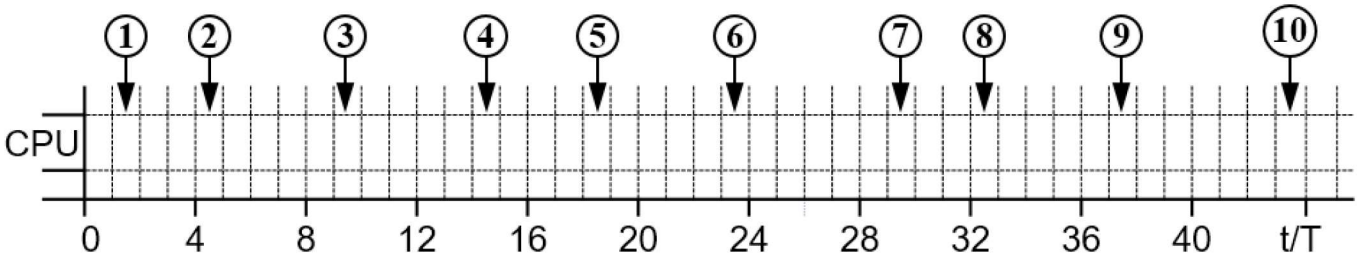


***Bild BS-9.4**: Aufgabenteil 2: nicht-präemptiv und FIFO*

9.5	Prozess zum **Zeitpunkt Nr. 1 in Bild BS-9.4**	☐ P1 ☐ P3 ☑ P5 ☐ ruhend	☐ P2 ☐ P4 ☐ P6
9.6	Prozess zum **Zeitpunkt Nr. 2 in Bild BS-9.4**	☑ P1 ☐ P3 ☐ P5 ☐ ruhend	☐ P2 ☐ P4 ☐ P6
9.7	Prozess zum **Zeitpunkt Nr. 3 in Bild BS-9.4**	☐ P1 ☐ P3 ☐ P5 ☐ ruhend	☑ P2 ☐ P4 ☐ P6
9.8	Prozess zum **Zeitpunkt Nr. 4 in Bild BS-9.4**	☐ P1 ☐ P3 ☐ P5 ☑ ruhend	☐ P2 ☐ P4 ☐ P6
9.9	Prozess zum **Zeitpunkt Nr. 5 in Bild BS-9.4**	☑ P1 ☐ P3 ☐ P5 ☐ ruhend	☐ P2 ☐ P4 ☐ P6
9.10	Prozess zum **Zeitpunkt Nr. 6 in Bild BS-9.4**	☐ P1 ☐ P3 ☐ P5 ☑ ruhend	☐ P2 ☐ P4 ☐ P6
9.11	Prozess zum **Zeitpunkt Nr. 7 in Bild BS-9.4**	☐ P1 ☐ P3 ☐ P5 ☐ ruhend	☐ P2 ☑ P4 ☐ P6
9.12	Prozess zum **Zeitpunkt Nr. 8 in Bild BS-9.4**	☐ P1 ☐ P3 ☐ P5 ☐ ruhend	☑ P2 ☐ P4 ☐ P6
9.13	Prozess zum **Zeitpunkt Nr. 9 in Bild BS-9.4**	☐ P1 ☐ P3 ☐ P5 ☐ ruhend	☐ P2 ☐ P4 ☑ P6
9.14	Prozess zum **Zeitpunkt Nr. 10 in Bild BS-9.4**	☐ P1 ☐ P3 ☐ P5 ☐ ruhend	☐ P2 ☐ P4 ☑ P6

10. BS: IEC 61131-3

Eine Zugübergang mit Vollschranken (*VS*) soll automatisch gesteuert werden, siehe **Bild BS-10.1**. Das Schließen der Schranken *VS* kommt nur dann in Frage, wenn die Weiche *X1* Richtung *„A"* zeigt. Darüber hinaus muss sowohl die Ampel *Y1* grün leuchten als auch der von links anfahrende Zug durch die Lichtschranke *L1* detektiert werden. Unmittelbar nach dem Bahnübergang dient Lichtschranke *L2* zum automatischen Öffnen der Schranken *VS*. Darüber Hinaus soll basierend auf dem Signal der Ampel und der ersten Lichtschranke *L1* ein *Alarm* ausgelöst werden, wenn der Zug rotes Licht missachtet. Eine Tabelle der Sensoren und Aktoren des Zugübergangs ist in **Bild BS-10.2** aufgezeigt.

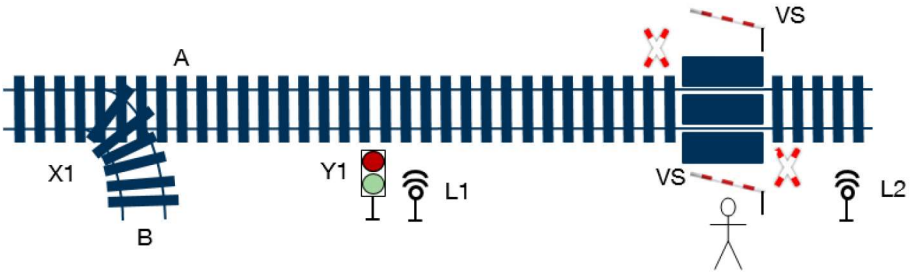


***Bild BS-10.1**: Skizze der Steuerungselemente des Zugübergangs*

Name	Typ	Datentyp	Beschreibung
X1	Aktor	String	Weiche Richtung „A“ → X1 = A, Richtung „B“ → X1 = „B“
Y1	Sensor	Bool	Ampel grün (true) oder rot (false)
L1, L2	Sensor	Bool	Zug auf Höhe Li detektiert (true) oder nicht (false)
VS	Aktor	Bool	Schranken öffnen (true) oder schließen (false)
Alarm	Sensor	Bool	Fehlerfall (true) oder nicht (false)

***Bild BS-10.2**: Sensoren und Aktoren des Zugübergangs*

Folgendes, unvollständiges, in **Funktionsbausteinsprache** implementiertes Programm (siehe **Bild BS-10.3**) ist als Basis für die folgenden Aufgaben gegeben. Beantworten Sie die nachfolgenden Fragen zu den **Lücken mit den Nummern 1 bis 12 in Bild BS-10.3** und kreuzen Sie die jeweils korrekte Antwort an.

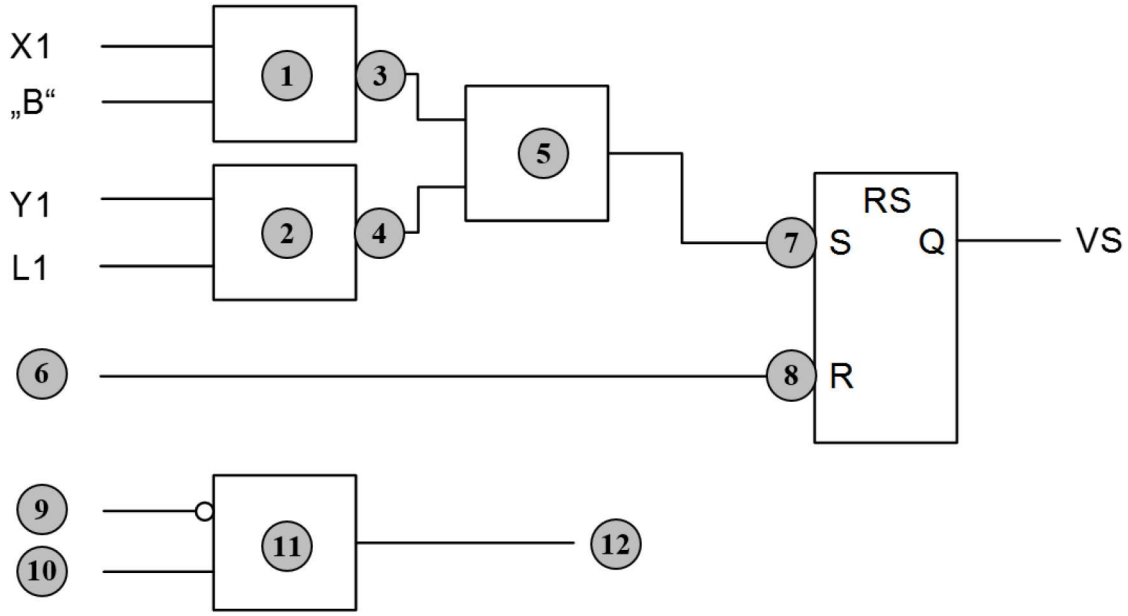


***Bild BS-10.3**: Steuerungsprogramm realisiert in Funktionsbausteinsprache*

Hinweis: Nutzen Sie die die Steuerungselemente (**Bild BS-10.1**) sowie die vorgegebenen Sensoren und Aktoren (**Bild BS-10.2**) zum Lösen der folgenden Aufgaben. Jeweils nur Einfachnennung möglich.

10. BS: IEC 61131-3 [Fortsetzung]

10.1 Lücke **Nr. 1 in Bild BS-10.3**
- ☐ AND
- ☐ OR
- ☐ XOR
- ☑ EQ

10.2 Lücke **Nr. 2 in Bild BS-10.3**
- ☑ AND
- ☐ OR
- ☐ XOR
- ☐ EQ

10.3 Lücke **Nr. 3 in Bild BS-10.3**
- ☑ Negation (--o)
- ☐ keine Negation (---)

10.4 Lücke **Nr. 4 in Bild BS-10.3**
- ☐ Negation (--o)
- ☑ keine Negation (---)

10.5 Lücke **Nr. 5 in Bild BS-10.3**
- ☑ AND
- ☐ OR
- ☐ XOR
- ☐ EQ

10.6 Lücke **Nr. 6 in Bild BS-10.3**
- ☐ X1
- ☐ Y1
- ☐ L1
- ☑ L2
- ☐ VS
- ☐ Alarm

10.7 Lücke **Nr. 7 in Bild BS-10.3**
- ☐ Negation (--o)
- ☑ keine Negation (---)

10.8 Lücke **Nr. 8 in Bild BS-10.3**
- ☐ Negation (--o)
- ☑ keine Negation (---)

10.9 Lücke **Nr. 9 in Bild BS-10.3**
- ☐ X1
- ☑ Y1
- ☐ L1
- ☐ L2
- ☐ VS
- ☐ Alarm

10.10 Lücke **Nr. 10 in Bild BS-10.3**
- ☐ X1
- ☐ Y1
- ☑ L1
- ☐ L2
- ☐ VS
- ☐ Alarm

10.11 Lücke **Nr. 11 in Bild BS-10.3**
- ☑ AND
- ☐ OR
- ☐ XOR
- ☐ EQ

10.12 Lücke **Nr. 12 in Bild BS-10.3**
- ☐ X1
- ☐ Y1
- ☐ L1
- ☐ L2
- ☐ VS
- ☑ Alarm

11. BS: Pearl

Im Folgenden sehen Sie das Zustandsdiagramm eines Rechenprozesses mit fünf Grundzuständen. Dieses weist allerdings hinsichtlich der Bezeichnung der Zustände und ihrer Übergänge die **Lücken mit den Nummern 1 bis 5** (siehe **Bild BS-11.1**) auf.

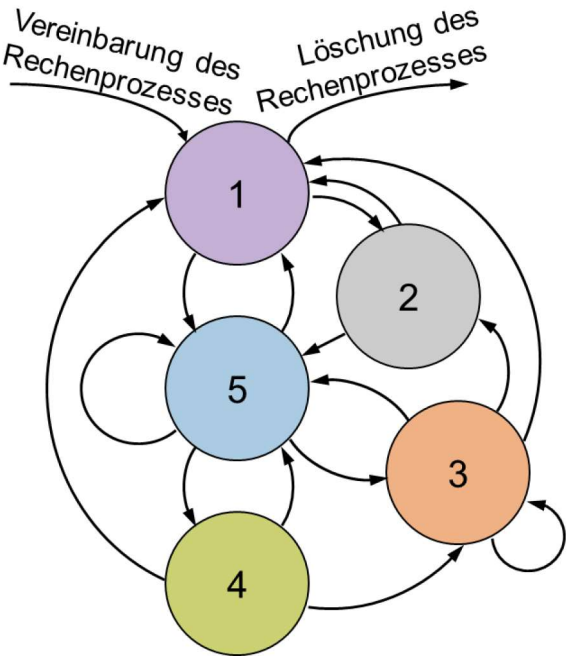


***Bild BS-11.1**: Zustandsdiagramm des Rechenprozesses*

11.1 Ordnen Sie die in **Bild BS-11.1 angegebenen Nummern 1 bis 5** den unten stehenden Bezeichnungen zu.
Hinweis: Jeweils nur Einfachnennung möglich.

	eingeplant (scheduled)	bereit (runnable)	ruhend (dormant)	laufend (running)	blockiert (suspended)
1	☐	☐	☑	☐	☐
2	☐	☐	☐	☐	☑
3	☑	☐	☐	☐	☐
4	☐	☑	☐	☐	☐
5	☐	☐	☐	☑	☐

11.2 Welche Beschreibung der Anweisung "SUSPEND" trifft zu? Bitte kreuzen Sie an.
Hinweis: Nur Einfachnennung möglich.
- ☐ Beendigungsanweisung
- ☑ Blockierungsanweisung
- ☐ Task kann durch diese Anweisung wieder in "bereit" versetzt werden
- ☐ Eingeplante Task wird wieder ausgeplant
- ☐ Vereinbarungsanweisung

12. MSE: Automaten

12.1 Welche Eigenschaft muss für einen deterministischen Automaten erfüllt sein? Bitte kreuzen Sie die richtige Antwort an.
Hinweis: Nur Einfachnennung möglich.

☐ Unendliche Anzahl von Zuständen ☐ Genau ein Endzustand ☑ Eindeutige Übertragungsfunktion

Betrachten Sie nun im Folgenden die in **Bild MSE-12.1 angegebenen Automaten mit den Nummern 1 bis 3** und beantworten Sie die dazugehörigen Fragen.

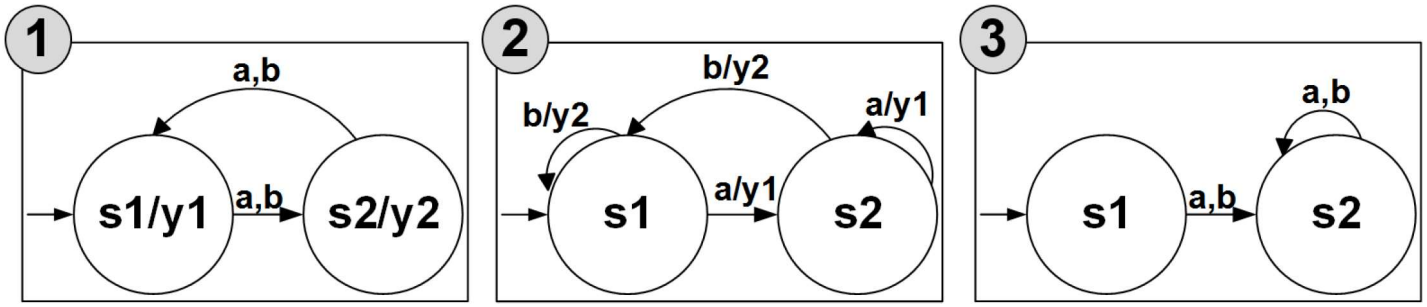


***Bild MSE-12.1**: Automaten 1, 2 und 3*

12.2 Bei welchem der Automaten in **Bild MSE-12.1** handelt es sich um einen Moore-Automaten?
Hinweis: Nur Einfachnennung möglich.

☑ Automat 1
☐ Automat 2
☐ Automat 3

12.3 Welchen der in **Bild MSE-12.1** angegebenen Automaten müssen Sie wählen, damit der Automat bei der Eingabe *b* unabhängig vom aktuellen Zustand immer in den Zustand *s2* wechselt?
Hinweis: Nur Einfachnennung möglich.

☐ Automat 1
☐ Automat 2
☑ Automat 3

12.4 **Automat Nr. 2 in Bild MSE-12.1** soll sich aktuell in Zustand *s1* befinden. Welche Eingabe müssen Sie tätigen, damit Sie die Ausgabesequenz *y1 y2 y2* erhalten? Bitte tragen Sie Ihre Antwort **in Großbuchstaben** in die gestrichelten Lösungsfelder ein.
Hinweis: Nur Lösungen innerhalb der angegebenen Ergebnisfelder werden gewertet!

Ergebnis: ABB

Ergebnis: [ ][ ][ ][ ][ ]

12.5 In welchem Zustand befindet sich der **Automat Nr. 2 in Bild MSE-12.1** nach der in Frage 12.4 angegebenen Ausgabesequenz *y1 y2 y2*? Bitte tragen Sie Ihre Antwort in die Lösungsfelder ein.
Hinweis: Nur Lösungen innerhalb der angegebenen Ergebnisfelder werden gewertet!

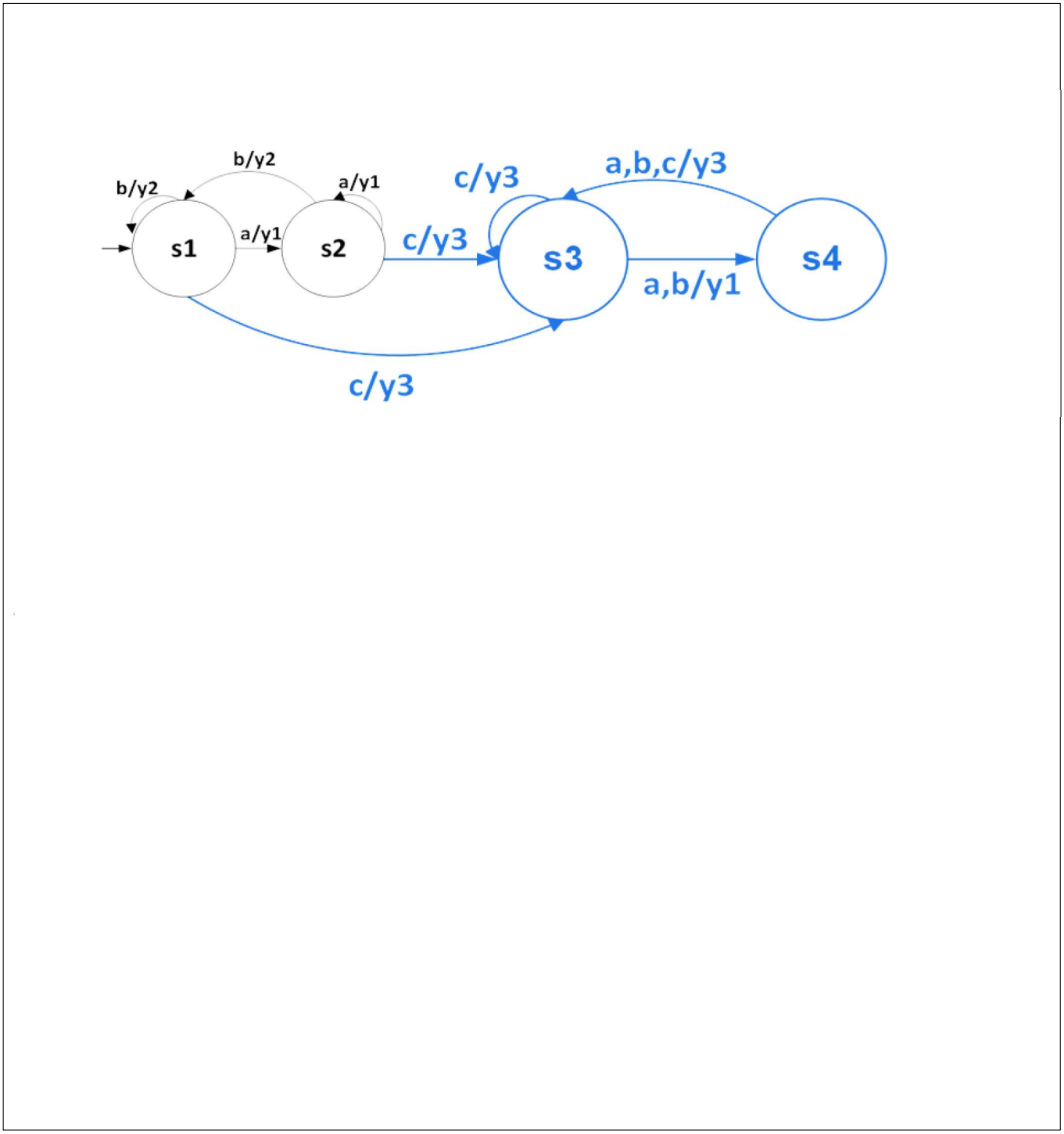
Ergebnis: S1 || Ergebnis: s1

Ergebnis: [ ][ ][ ][ ][ ]

12. MSE: Automaten [Fortsetzung]

12.6 **Automat Nr. 2 in Bild MSE-12.1** soll um die Zustände *s3* und *s4* sowie um das Eingabezeichen *c* erweitert werden. Unabhängig vom aktuellen Zustand soll der erweiterte Automat bei einer Eingabe *c* immer *y3* ausgeben und in den Zustand *s3* wechseln. Aus Zustand *s3* soll der Automat durch Eingabe von *a* oder *b* in den Zustand *s4* wechseln und *y1* ausgeben. Aus Zustand *s4* soll der Automat bei jeder Eingabe in den Zustand *s3* wechseln und *y3* ausgeben. Vervollständigen Sie den angegebenen Automaten gemäß den Angaben.

Hinweis: Ein Entfernen von Bestandteilen des Automaten ist nicht erlaubt.



13. MSE: Strukturierte Analyse/Real-Time (SA/RT)

Gegeben ist folgendes (Teil-)System eines industriellen Brotbackofens (siehe **Bild MSE-13.1**). Sobald ein neuer Brotlaib von der Lichtschranke *LS1* erkannt wird, befördert das Band *E1* den Brotlaib zum Backofen. Nachdem das Brot gebacken wurde, wird es durch Lichtschranke *LS2* erkannt. Sobald *LS2* das Brot erkannt hat, wird es durch Band *A1* abtransportiert. Das System soll in den folgenden Aufgaben mittels Strukturierter Analyse/ Real-Time (SA/RT) modelliert werden.

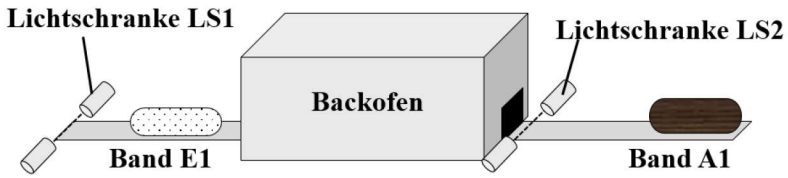
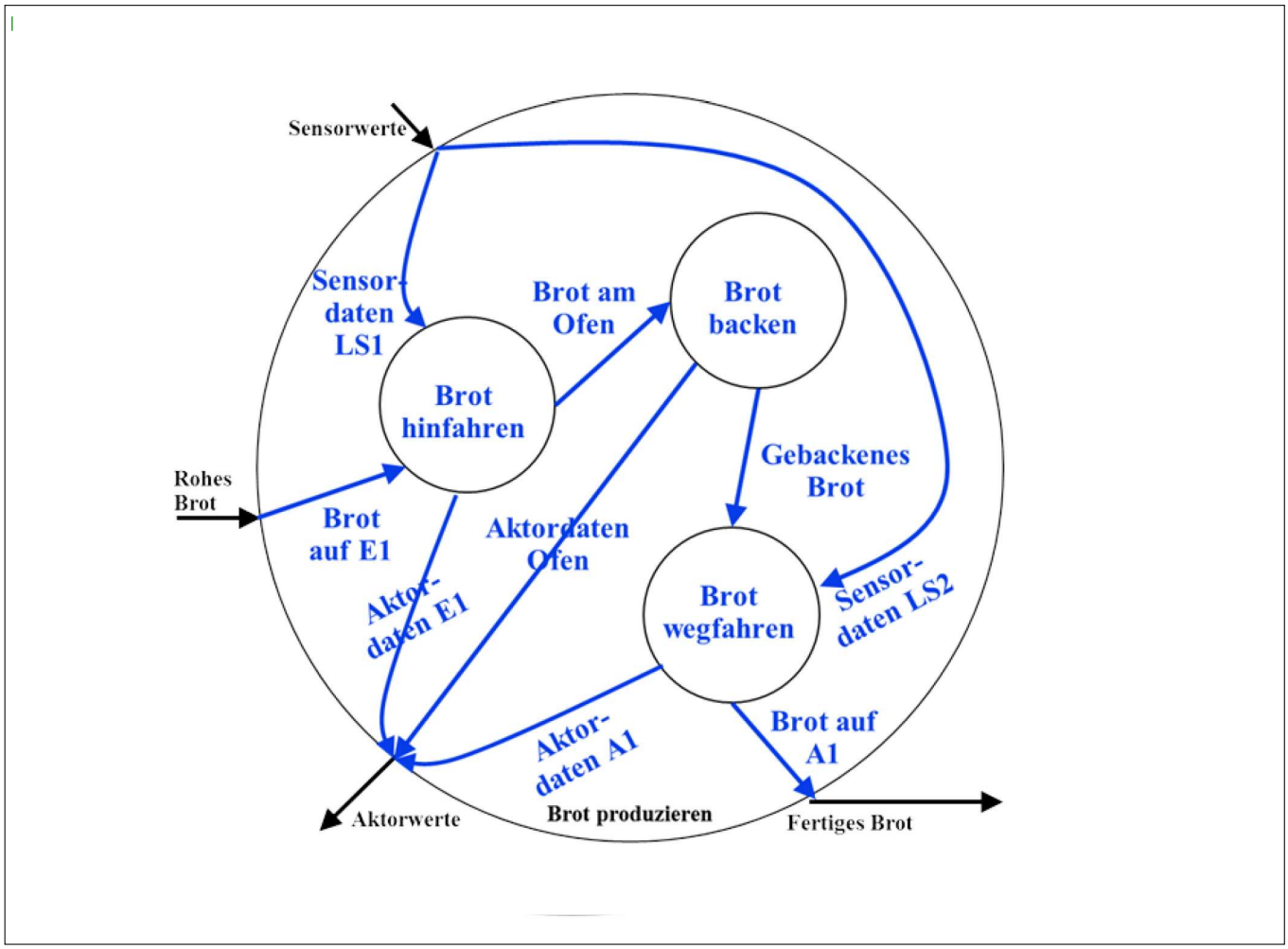


***Bild MSE-13.1**: Skizze des industriellen Brotbackofens*

13.1 Es soll das **Datenflussdiagramm** des Prozesses *Brot produzieren* (siehe **Bild MSE-13.1**) modelliert werden. Der Prozess besteht aus den Subprozessen *Brot hinfahren*, *Brot backen* und *Brot wegfahren*. Folgende Flüsse sollen dabei betrachtet werden:

Materialfluss: Am Anfang des Prozesses befindet sich das Brot auf *E1*. Sobald das Brot am Ofen ist wird es gebacken. Das gebackene Brot wird daraufhin abtransportiert und somit befindet sich das Brot auf *A1*. Das fertige Brot kann dann in den Verkauf gegeben werden.
Sensordaten: Von dem übergeordneten Prozess fließen die Sensordaten der Lichtschranke *LS1* (notwendig um Band *E1* zu bewegen) und der Lichtschranke *LS2* (notwendig um Band *A1* zu bewegen) zu den entsprechenden Subprozessen.
Aktordaten: Die Aktordaten setzen sich aus den Aktordaten der Bänder und den Aktordaten des Ofens zusammen.

Vervollständigen Sie folgendes Datenflussdiagramm entsprechend der Beschreibung.



13. MSE: Strukturierte Analyse/Real-Time (SA/RT) [Fortsetzung]

13.2 Zur Verfeinerung des Prozesses *Brot produzieren* (siehe **Bild MSE-13.1**) soll eine Antwortzeitspezifikation in Form eines **Timing-Diagramms** durchgeführt werden, um sicherzustellen, dass der Prozess korrekt durchgeführt wird. Die Werte der Sensoren bzw. Aktoren können jeweils TRUE (z. B. Brot erkannt) oder FALSE (z. B. Brot nicht erkannt) sein. Ergänzen Sie das Timing-Diagramm gemäß folgender Angaben (Werteverläufe und Zeitangaben):

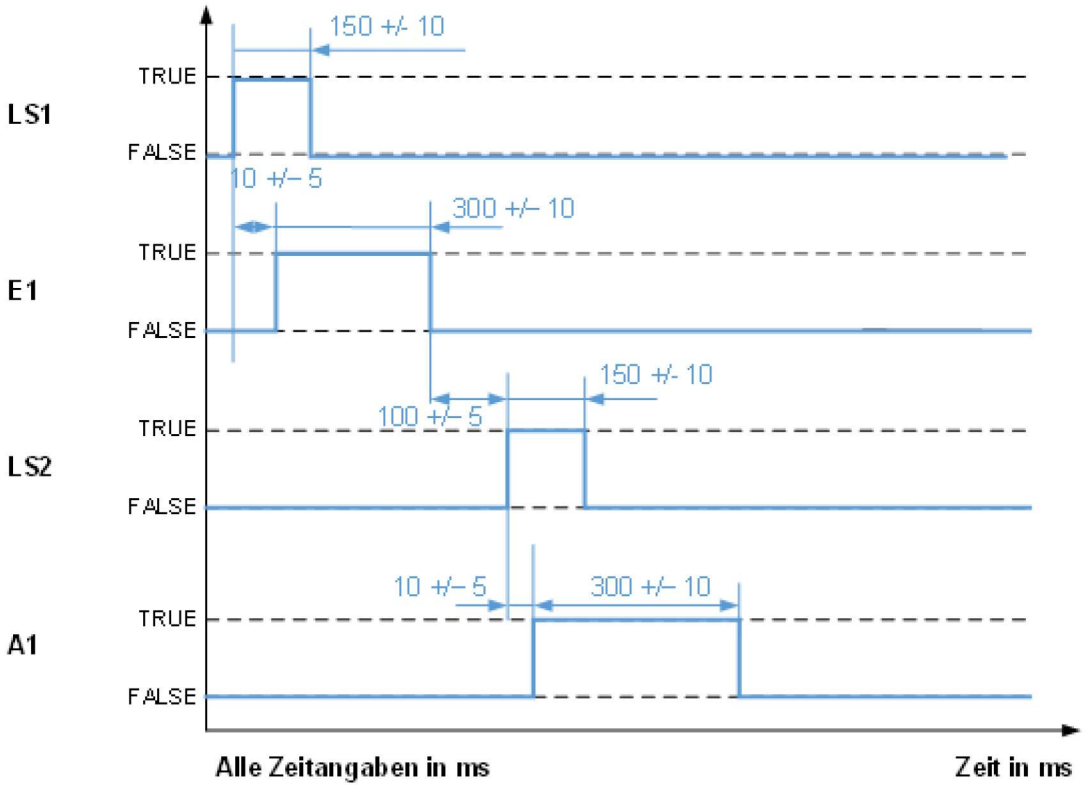
Sobald ein neues Brot erkannt wird, liefert die Lichtschranke *LS1* für 150 +/− 10 ms den Wert TRUE.

10 +/− 5 ms nach Beginn der Broterkennung durch *LS1* bewegt sich Band *E1* für 300 +/− 10 ms und schaltet dann wieder ab.

Nach Abschalten von E1 wird das Brot für 100 +/− 5 ms gebacken und wird dann direkt von Lichtschranke *LS2* erkannt. *LS2* liefert wiederum für 150 +/− 10 ms den Wert TRUE.

Ebenfalls 10 +/− 5 ms nach Beginn der Broterkennung durch *LS2* bewegt sich Band *A1* 300 +/− 10 ms und schaltet dann wieder ab.

Vervollständigen Sie folgendes Timing-Diagramm entsprechend der Beschreibung.

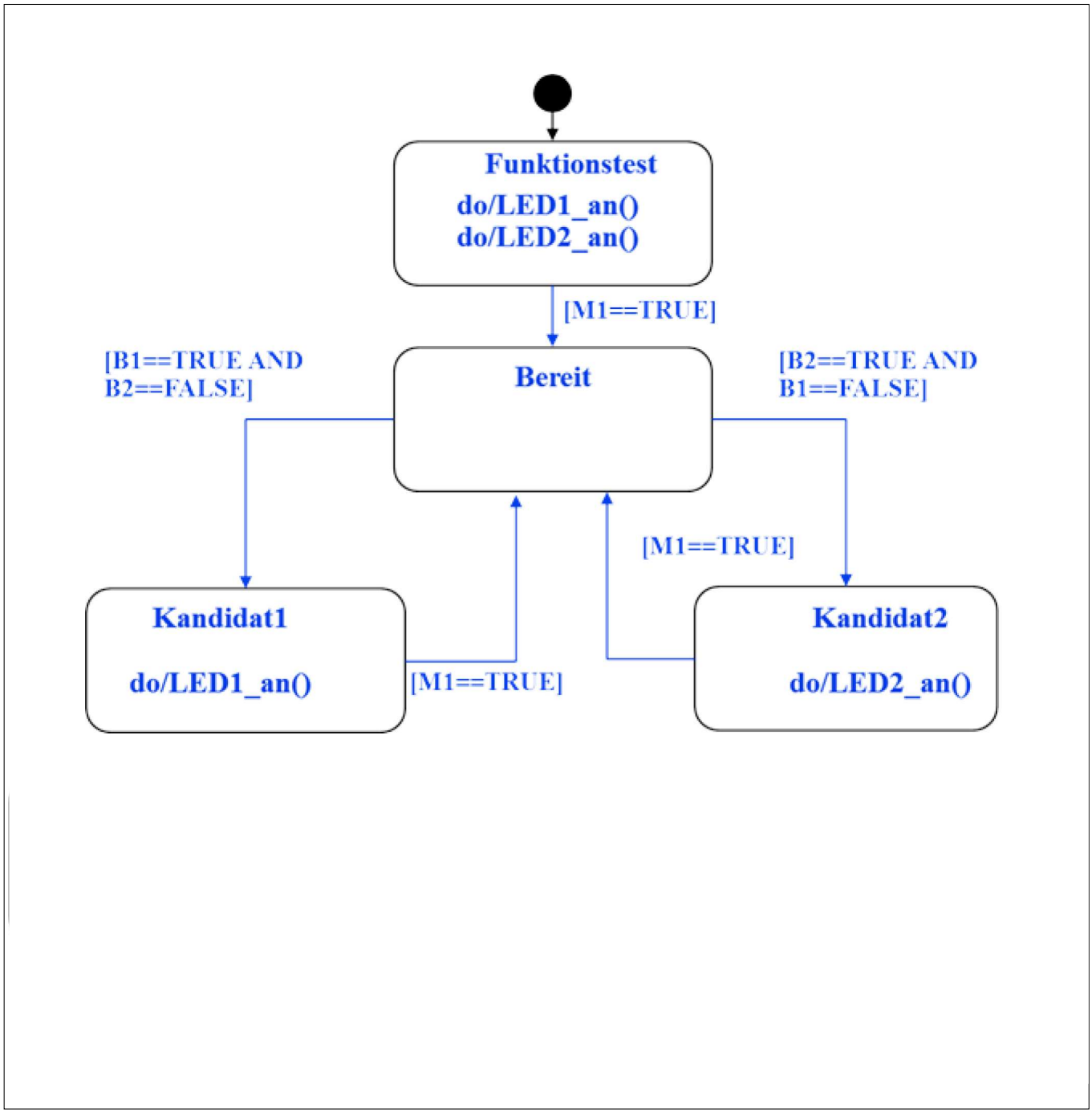


14. MSE: Zustandsdiagramm

Für ein Quizspiel soll ein Buzzersystem aufgebaut werden. Dieses besteht aus den 2 Buzzern (Tastern) *B1* und *B2* für die beiden Kandidaten und einem ‚Moderatortaster' *M1* zum Zurücksetzen des Systems. Die jeweilige Variable ist TRUE, wenn der Buzzer bzw. der Moderatortaster gedrückt wird, ansonsten ist die Variable FALSE (z.B. ist *B1* TRUE wenn Buzzer 1 gedrückt wird). Jeder Buzzer hat eine LED, um aufzuzeigen, welcher Kandidat als Erstes gedrückt hat. Die LEDs leuchten, solange die Aktionen *LED1_an* bzw. *LED2_an* ausgeführt werden.

Beim Starten geht das System zunächst in den Zustand *Funktionstest* über. In dem Zustand sollen die LEDs beider Buzzer leuchten. Dieser Zustand soll verlassen werden, wenn der Moderatortaster *M1* gedrückt wird. Das System geht dann in den Zustand *Bereit* über. Wird nun Buzzer *B1* gedrückt und *B2* nicht, wird in den Zustand *Kandidat1* gewechselt. Wird stattdessen *B2* gedrückt und *B1* nicht, wird in Zustand *Kandidat2* gewechselt. Im Zustand *Kandidat1* soll LED1 bzw. im Zustand *Kandidat2* soll LED2 solange leuchten, bis der jeweilige Zustand wieder verlassen wird. Ein Verlassen der Zustände *Kandidat1* und *Kandidat2* ist nur durch Drücken des Moderatortasters *M1* möglich. Hierdurch wird wieder in den Zustand *Bereit* gewechselt.

14.1 Vervollständigen Sie folgendes Zustandsdiagramm gemäß der Angaben.



15. C: Datentypen

Sie sollen folgende Variablen für die Beschreibung eines Fahrzeugs definieren, sodass *so wenig Speicher wie möglich* genutzt wird:
Variable baujahr: Baujahr des Fahrzeugs
Variable masse: Masse des Fahrzeugs in Kilogramm, auf zwei Nachkommastellen genau
Variable effklasse: Effizienzklasse des Fahrzeugs (A bis G)
Variable seriennr: 10-stellige, herstellerseitige Seriennummer des Fahrzeugs, die nur Ziffern enthält
Kreuzen Sie an, welchen Datentyp Sie jeweils verwenden würden. Jeweils nur Einfachnennung möglich.

15.1 Datentyp für **Variable baujahr**
- ☐ char
- ☐ long int
- ☐ double
- ☑ int
- ☐ float

15.2 Datentyp für **Variable masse**
- ☐ char
- ☐ long int
- ☐ double
- ☐ int
- ☑ float

15.3 Datentyp für **Variable effklasse**
- ☑ char
- ☐ long int
- ☐ double
- ☐ int
- ☐ float

15.4 Datentyp für **Variable seriennr**
- ☐ char
- ☑ long int
- ☐ double
- ☐ int
- ☐ float

16. C: Ein-/Ausgabe

Ergänzen Sie die in **Bild C-16.1 angegebenen Lücken mit den Nummern 1 bis 4** in der formatierten Ausgabe, sodass das **Baujahr** sowie die **Masse in Gramm ohne Nachkommastellen** ausgegeben werden. Kreuzen Sie an, welche Codebestandteile Sie einfügen. Jeweils nur Einfachnennung möglich.

```
__(1)__("Baujahr: __(2)__, Masse: __(3)__", __(4)__);
```

***Bild C-16.1**: Formatierte Ausgabe*

16.1 Lücke **Nr. 1 in Bild C-16.1**
- ☐ print
- ☐ println
- ☐ scanf
- ☑ printf
- ☐ scan
- ☐ scanln

16.2 Lücke **Nr. 2 in Bild C-16.1**
- ☐ baujahr
- ☑ %i
- ☐ %.0f
- ☐ masse
- ☐ %f
- ☐ %0.f

16.3 Lücke **Nr. 3 in Bild C-16.1**
- ☐ baujahr
- ☐ %i
- ☑ %.0f
- ☐ masse
- ☐ %f
- ☐ %0.f

16.4 Lücke **Nr. 4 in Bild C-16.1**
- ☐ baujahr, masse
- ☑ baujahr, masse*1000
- ☐ baujahr, masse/1000
- ☐ &baujahr, &masse
- ☐ &baujahr, &masse*1000
- ☐ &baujahr, &masse/1000

17. C: Boolesche Algebra

Bestimmen Sie das Ergebnis der nachfolgend angegebenen Ausdrücke im Dezimalsystem. Gegeben seien dafür folgende Variablen:
int-Variablen: A = 4, B = 22;
float-Variable: C = 0.923f;
char-Variable: D = 0x2B;
int-Zeiger: pi = &A;

Hinweis: Nur Lösungen innerhalb der angegebenen Ergebnisfelder werden gewertet!

17.1 Ausdruck **Nr. 1:** A && ((int) D)

1 || 01

17.2 Ausdruck **Nr. 2:** (A | B) - 2 * (*pi)

14

17.3 Ausdruck **Nr. 3:** B - (A & ((int) C)) != B

0 || 00

17.4 Ausdruck **Nr. 4:** (A / B) == ((int) C)

1 || 01

18. C: Kontrollstrukturen

Ihre Aufgabe ist es, die Qualitätskontrolle einer Fertigungsanlage für Schokoladenkugeln zu programmieren. Hierzu soll mittels einer einfachen Berechnung sichergestellt werden, dass die Schokoladenkugeln ein minimales Volumen (Variable *minVolumen*) nicht unterschreiten. Dazu wird anhand von *NUM* Schokoladenkugeln das Volumen jeder Kugel (Variable *aktuellesVolumen*) ermittelt und mit dem minimalen Volumen (Variable *minVolumen*) verglichen. Ist das Volumen der Kugel akzeptabel (d.h. ist ihr Volumen mindestens so groß wie das minimale Volumen), so wird an die entsprechende Stelle im Array *bewertung* eine *1* geschrieben. Ist das Volumen der Kugel nicht akzeptabel (d.h. unterschreitet ihr Volumen das minimale Volumen), so wird an die entsprechende Stelle im Array *bewertung* eine *0* geschrieben. Gleichzeitig soll das Durchschnittsvolumen aller Schokoladenkugeln (Variable *durchschnittVolumen*) sowie die Anzahl der akzeptablen Schokoladenkugeln ermittelt und durch die Ausgabefunktion *ausgabe(float durchschnittVolumen, int anzahlGut)* ausgegeben werden.

Hinweis: Verwenden Sie für Ihr C-Programm die angegebene Vorlage (**Bild C-18.1**). Kreuzen Sie für die jeweiligen in **Bild C-18.1 angegebenen Lücken mit den Nummern 1 bis 10** an, welchen C-Code Sie einfügen. Jeweils nur Einfachnennung möglich.

```
#include <stdio.h>
#include <math.h>

#define NUM 5

int main() {
   float radius[NUM] = {2, 1.8, 2.3, 1.8, 1.7};
   float bewertung[NUM] = {-1 , -1 , -1 , -1 , -1};
   float durchschnittVolumen = 0.0f, summeVolumen = 0.0f;
   float aktuellesVolumen = 0.0f, minVolumen = 20.0f, pi = 3.14159f;
   int i = 0, j = 0;

   (1) ( (2) ) {
        aktuellesVolumen = (3) ;

        (4) ( (5) )
          bewertung[i] = 0 ;
        (6) {
          bewertung[i] = 1 ;
          (7) ;
        }

        summeVolumen += aktuellesVolumen;
        (8) ;
   }

   durchschnittVolumen = (9) / (10) ;
   ausgabe(durchschnittVolumen, j);
   return 0;
}
```

***Bild C-18.1**: Qualitätskontrolle für die Fertigung von Schokoladenkugeln*

Hinweis: Das Volumen einer Kugel wird durch die angegebene Formel ermittelt. Verwenden Sie hierzu die Funktion *double pow(double x, double y)*, welche die Potenz x^y zurückgibt.	$$V = \frac{4}{3} \cdot \pi \cdot r^3$$

18. C: Kontrollstrukturen [Fortsetzung]

18.1 Lücke **Nr. 1 in Bild C-18.1**

- ☐ if
- ☐ else
- ☑ while
- ☐ do
- ☐ for
- ☐ case

18.2 Lücke **Nr. 2 in Bild C-18.1**

- ☐ i > NUM
- ☑ i < NUM
- ☐ i <= NUM
- ☐ j > NUM
- ☐ j < NUM
- ☐ j <= NUM

18.3 Lücke **Nr. 3 in Bild C-18.1**

- ☐ 4.0 / 3.0 * pi * pow (3, radius[i])
- ☑ 4.0 / 3.0 * pi * pow (radius[i], 3)
- ☐ 4.0 / 3.0 * pi * pow (3, radius+i)
- ☐ 4.0 / 3.0 * pi * pow (radius+i, 3)

18.4 Lücke **Nr. 4 in Bild C-18.1**

- ☑ if
- ☐ else
- ☐ while
- ☐ do
- ☐ for
- ☐ case

18.5 Lücke **Nr. 5 in Bild C-18.1**

- ☐ aktuellesVolumen > minVolumen
- ☐ aktuellesVolumen >= minVolumen
- ☐ aktuellesVolumen != minVolumen
- ☐ aktuellesVolumen <= minVolumen
- ☑ aktuellesVolumen < minVolumen

18.6 Lücke **Nr. 6 in Bild C-18.1**

- ☐ if
- ☑ else
- ☐ while
- ☐ do
- ☐ for
- ☐ case

18.7 Lücke **Nr. 7 in Bild C-18.1**

- ☐ i--
- ☐ i++
- ☐ j--
- ☑ j++

18.8 Lücke **Nr. 8 in Bild C-18.1**

- ☐ i--
- ☑ i++
- ☐ j--
- ☐ j++

18.9 Lücke **Nr. 9 in Bild C-18.1**

- ☐ i
- ☐ j
- ☑ summeVolumen
- ☐ durchschnittVolumen
- ☐ minVolumen
- ☐ aktuellesVolumen

18.10 Lücke **Nr. 10 in Bild C-18.1**

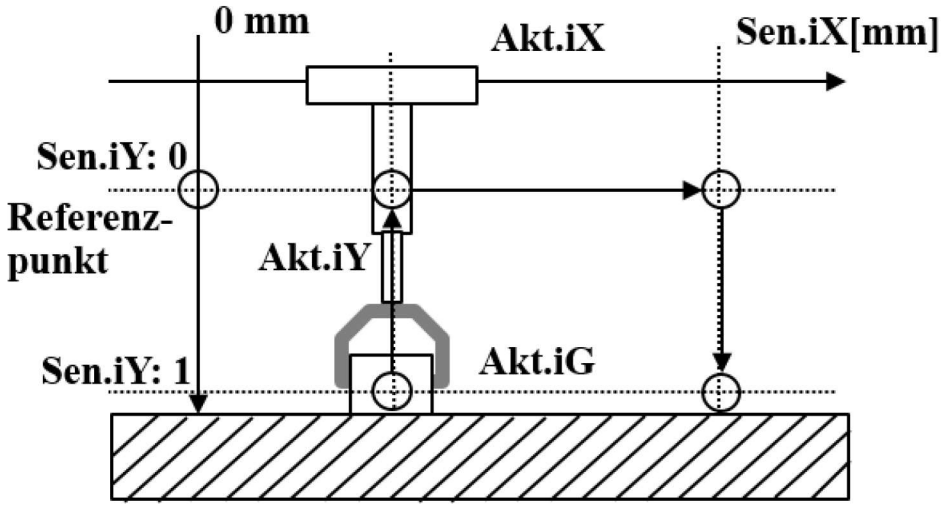
- ☑ i
- ☐ j
- ☐ summeVolumen
- ☐ durchschnittVolumen
- ☐ minVolumen
- ☐ aktuellesVolumen

19. C: Zyklische Programmierung einer Krananlage

Die Firma ISA GmbH stellt die Krananlage „flexCrane“ (siehe **Bild C-19.1, links**) her, welche ein eigenes Kommandosystem mit insgesamt 4 Befehlstypen (*iCmd* = { 1, 2, 3, 4 }) besitzt. Die Anlage kann vertikal in zwei Positionen (*Akt.iY = 0*: oben; *Akt.iY = 1*: unten), sowie kontinuierlich entlang der X-Achse horizontal nach rechts (*Akt.iX = +1*) oder links (*Akt.iX = -1*) verfahren. Weiterhin kann der Greifer des Krans die Zustände offen (*Akt.iG = 0*) und geschlossen (*Akt.iG = 1*) einnehmen. Die Kommandoschritte sind dazu im Struct-Array *crane* gespeichert (siehe **Bild C-19.1, rechts**), welches extern eingelesen wird. Diese werden vom Steuerungsprogramm schrittweise abgearbeitet. Zum Beginn muss hierfür ein Datentyp *COMMAND* definiert werden. Legen Sie in diesem die Datentypen *iCmd (int)* und *iParam (int)* an.

Eine Liste der Aktoren, Sensoren und Merkvariablen, die im Folgenden verwendet werden, finden Sie in **Bild C-19.2**.

Skizze der Krananlage



Kommandoschritte

	Array *crane*	
i	***iCmd***	***iParam***
0	1	0
1	2	+150
2	1	1
3	4	0

***Bild C-19.1**: Skizze der Krananlage und Kommandoschritte*

Aktoren	Akt.iY	Verfährt den Kran vertikal an die Position oben (0) oder unten (1)
	Akt.iX	Verfährt den Kran horizontal nach rechts (+1) oder nach links (-1)
	Akt.iG	Öffnet (0) und schließt (1) den Greifarm
Sensoren	Sen.iY	Diskrete vertikale Position des Kran: oben (0) / unten (1)
	Sen.iX	Kontinuierliche horizontale Position des Kran in mm zum Nullpunkt (int)
	Sen.iG	Zustand des Greifers: offen (0) / geschlossen (1)
Merkvariablen	crane[i].iCmd	Variable, welche das **Kommando** enthält {1,2,3,4}
	crane[i].iParam	Variable, welche abhängig vom Kommando die **Parameter** enthält
	i	Aktueller Rezeptschritt
	s	Speichert einen Weg (in mm) für die Verwendung als Streckenmessung
	active	Steuert die Ausführung des Programms (Start / Stopp)
	state	Speichert den aktuellen Zustand / Subzustand

***Bild C-19.2**: Aktoren, Sensoren und Merkvariablen*

19. C: Zyklische Programmierung einer Krananlage [Fortsetzung]

Im Folgenden sind die einzelnen Zustände der Krananlage (siehe **Bild C-19.3**) beschrieben:

Zustand 0: Zu Anfang steht der Kran am Referenzpunkt (*Sen.iX = 0, Sen.iG = 0, Sen.iY = 0*). Der Befehlszähler *i* wird um eins erhöht und geprüft, welcher Befehl vorliegt. Je nachdem wird von 0 zum Zustand 1 (Vertikal), 2 (Greifer), 3 (Horizontal) oder 4 (Ende) gewechselt.

Zustand 1/2 (Vertikal/Greifer): Zum Übergang vom Istzustand (*Sen.iY/Sen.iG*) zum Sollzustand (*iParam*), wird der Linear-Aktuator *Akt.iY* bzw. der Greifer *Akt.iG* entsprechend geschaltet. Nachdem der Istzustand den Wert von *iParam* eingenommen hat, wird in den Zustand 0 zurückgewechselt.

Zustand 3 (Links/Rechts): Zum Übergang vom Istzustand (*Sen.iX*) zum Sollzustand, wird der Linear-Aktuator *Akt.iX* nun entsprechend mit +1 (nach rechts) oder mit -1 (nach links) geschaltet, bis die in *iParam* angegebene Wegstrecke in mm gefahren wurde. Die zugehörige IF-Abfrage kann dabei mithilfe der Betragsfunktion *int abs(int x)* in einer Abfrage für beide Richtungen (positiver/negativer Verfahrweg) realisiert werden. Daraufhin wird der Aktor wieder auf 0 gesetzt und in den Zustand 0 gewechselt. Verwenden Sie zur Realisierung in Analogie zum Timer die Merkvariable *s*, um sich beim Start des Verfahrvorgangs die Ausgangsposition entlang der X-Achse zu merken.

Zustand 4: Nach Abarbeitung aller Befehle, soll der Kran in eine sichere Position gefahren werden, welche sich im Referenzpunkt befindet. Der Greifarm soll also bei *Sen.iY = 0, Sen.iX = 0* und der Greifer offen (*Sen.iG = 0*) sein. Sobald dieser Zustand erreicht wurde, soll das Programm durch Setzen der Variable *active* auf *0* beendet werden.

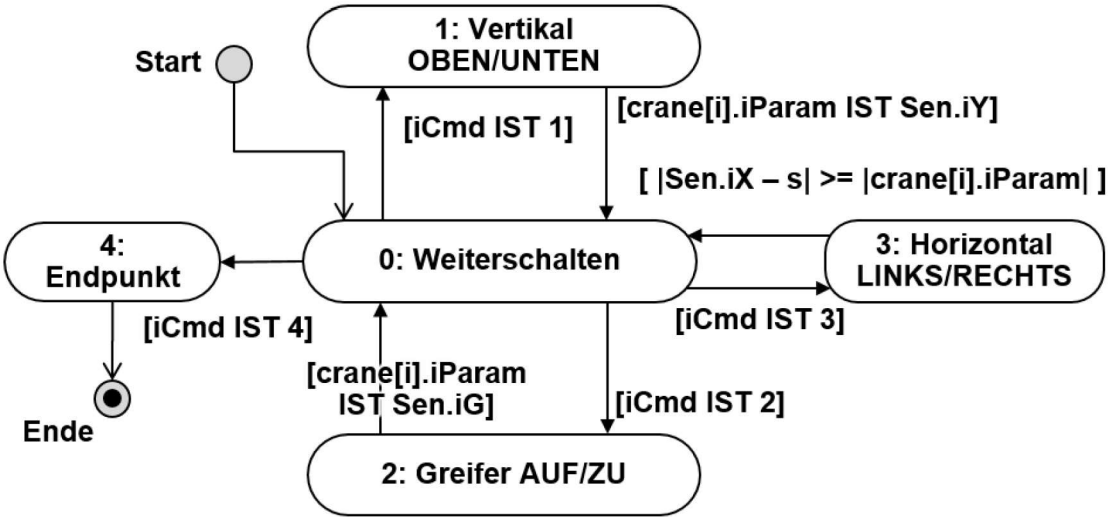


***Bild C-19.3**: Zustandsautomat der Krananlage*

Übertragen Sie das Zustandsdiagramm in **Bild C-19.3** und die oben beschriebene Funktionalität in den C-Code in den folgenden Aufgaben. Benutzen Sie als Basis die Skizze (**Bild C-19.1**) und die angegebenen Variablen (**Bild C-19.2**). Der Kopf des Programms mit Deklaration und Initialisierung der notwendigen Variablen ist bereits gegeben.

```
#include "eavar_flexcrane.h"
#include <stdlib.h>       // enthaelt Funktion int abs(int x)

struct SData Sen;         // Sensorvariablen
struct AData Akt;         // Aktorvariablen
unsigned int state=0;     // Zustandsvariable1 (Start in 0)
unsigned int iSteps=8;    // Max. Anzahl von Befehlen
unsigned int active=1;    // Programm laeuft solange true

int s=0;   // Speicher für Strecken-Messung (in mm)
int i=-1; // Initialisierung Kommandoschrittzähler
```

19. C: Zyklische Programmierung einer Krananlage [Fortsetzung]

19.1 **Krananlage (Teil 1)**: Bitte füllen Sie die Lücken aus.

```
typedef struct      // Typdefinition
{
        int iCmd;
        int iParam;
} COMMAND;

int main()
{
  COMMAND crane[iSteps];     //Structarray crane
Rezept_einlesen(&crane, iSteps);
while (active){     //Zyklische Ausführung
LeseSensoren(&Sen); //Sensorwerte einlesen
switch(   state    )
{
  case 0        : // Zustand 0 weiterschalten
    i++;
     state=crane[i].iCmd;                    //0->iCmd
    break;
   case 1        :  // Zustand 1 vertikal fahren
   Akt.iY=crane[i].iParam;   // Aktor setzen
   if(Sen.iY==crane[i].Param) state=0;          //1->0
   break;
   case 2:       : // Zustand 2 Greifer setzen
     Akt.iG=crane[i].iParam; // Aktor setzen
        if(Sen.iG==crane[i].iParam) state=0; //2->0
  break;
//(Fortsetzung naechste Seite)
```

19. C: Zyklische Programmierung einer Krananlage [Fortsetzung]

19.2 **Krananlage (Teil 2)**: Bitte füllen Sie die Lücken aus.

```
case 3:       : // Zustand 3 horizontal fahren
     if(!s)       // Messung und Aktor starten
     {    s=Sen.iX;       // Startposition merken
      // Je nach Vorzeichen rechts/links fahren
      if(crane[i].iParam>0)
          Akt.iX=1;
      else    Akt.iX=-1;          }
     // IF-Abfrage ob absoluter Weg bereits gefahren
     if(abs(Sen.iX-s)>=abs(crane[i].iParam))
          { Akt.iX=0;
          s=0;
            state=0;     }   // 3->0
     break;
case 4:       : // Zustand 4 Ende und Reset
// IF-Abfrage ob Reset-Zustand noch nicht erreicht
if(Sen.iY!=0 && Sen.iX>=0 && Sen.iG!=0)
     { Akt.iG=0;      // Aktoren setzen
       Akt.iY=0;
       Akt.iX=-1;  }
     active=0;          // Programm beenden
     break;       }
  SchreibeAktoren(&Akt); //Aktorwerte schreiben
} return 0;}
```

20. C: Sortieren von Zahlen mittels Swap Sort

Mittels des in einem Nassi-Shneidermann-Diagramm (siehe **Bild C-20.1**) angegebenen Sortieralgorithmus Swap Sort sollen integer-Zahlenfolgen aufsteigend (also kleinste Zahl zuerst) sortiert werden, welche ausschließlich aus ungleichen Zahlen bestehen; d.h. es befindet sich keine Zahl mehr als einmal im betreffenden Array mit der Größe *n*. Hierbei wird der Vorteil ausgenutzt, dass für jede Zahl im Array nach Zählung der restlichen Zahlen, welche kleiner als diese Zahl sind, die endgültige Stelle im Array bestimmt werden kann. Die Variable *k* dient hierbei als Zähler für die Anzahl der restlichen, kleineren Zahlen im Array.

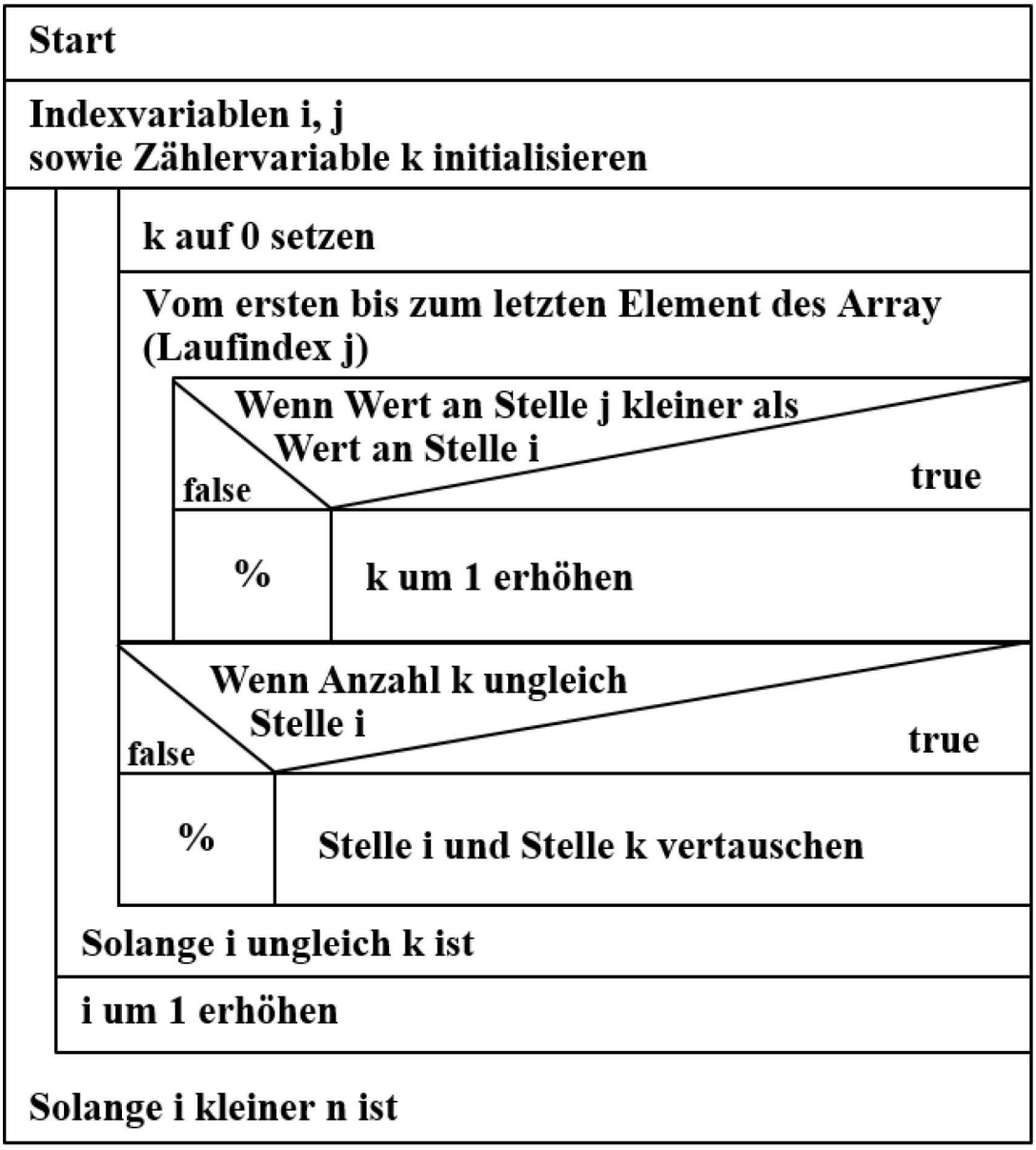


***Bild C-20.1**: Nassi-Shneidermann-Diagramm zum Sortieralgorithmus Swap Sort*

Zum weiteren Verständnis der Wirkungsweise des Swap-Sort-Algorithmus, ist im Folgenden ein Beispiel für eine Zahlenreihe aufgeführt. Die unterstrichenen Plätze sind endgültig und die Zahlen verbleiben an der Stelle.

Ausgangszustand:
9 7 2 4 1 (i=0) Inhalt: 9, Zählen aller Array-Inhalt welche kleiner als 9 sind:
k=4 also ungleich i => Tausche Stelle 0 und 4 (endgültiger Platz für 9 gefunden)

Sortiervorgänge:
1 7 2 4 **9** (i=0) Inhalt: 1, Zählen führt zu k=0 → k gleich i → kein Tausch → i um 1 erhöhen
1 7 2 4 **9** (i=1) Inhalt: 7, Zählen führt zu k=3 → k ungleich i → Tausch mit Stelle 3
1 4 2 **7 9** (i=1) Inhalt: 4, Zählen führt zu k=2 → k ungleich i → Tausch mit Stelle 2
1 2 **4 7 9** (i=1) Inhalt: 2, Zählen führt zu k=1 → k gleich i → kein Tausch → i um 1 erhöhen
1 2 4 7 9 → Erhöhen von i bis n, kein weiteres Tauschen, da alle Endpositionen erreicht.

20. C: Sortieren von Zahlen mittels Swap Sort [Fortsetzung]

Vervollständigen Sie nun unten die Funktion „swapsort“ entsprechend der zuvor im Nassi-Shneidermann-Diagramm (siehe **Bild C-20.1**) beschriebenen Funktionsweise. Der Funktion wird mittels *call-by-reference* ein Zeiger auf das Ziel-Array übergeben. Sie sortiert also direkt im Array vom Typ Integer und gibt keinen Rückgabewert zurück.

Die Sortierfunktion soll für ein Array mit einer Anzahl n Einträgen ausgelegt sein. Die dazugehörige integer Zahl soll mittels *call-by-value* an die Funktion übergeben werden.

Die Arrayelemente sollen *aufsteigend* sortiert werden.

Es sei des Weiteren bereits eine Funktion „*swap*“ implementiert, die nicht dargestellt ist. Die Funktion kann zwei Arrayelemente vom Typ Integer vertauschen. Verwenden Sie diese durch einen korrekten Funktionsaufruf mittels *call-by-reference*, indem Sie ihr die beiden Arrayelemente als Adresse übergeben.

20.1 Vervollständigen Sie, wie oben angegeben, die Lücken im Programm.

```
void  swapsort( int *iArray, int n )
{
    int i=0, j=0, k=0;          //Variablen init.
    do          //Aeussere Schleife
    {
        do          // Innere Schleife
        {
            k=0;          //Zaehler k zuruecksetzen
            for(j=0;j<n;j++)      //Zaehlen
            {
              if(iArray[j]<iArray[i]) k++;
            }
            // Tauschen falls ungleich
            if(k!=i) swap(&iArray[i],&iArray[k]);
        } while(k!=i);
        i++;          //Index erhoehen
    } while(i<n);
}
```

21. C: Zeiger

Gegeben Sei das in **Bild C-21.1** aufgezeigte, kurze C-Programm, welches ein Array mit vier Elementen *iAFeld* und die Zeigervariable *piZeiger* initialisiert. Die im C-Programm angegebenen Befehle führen jeweils zu einer Manipulation des Zeigers, sowie der Werte im Array. Geben Sie die veränderten Werte des Arrays nach Ausführen des C-Programms in den nachfolgenden Aufgaben an.

Hinweis: Nur Lösungen innerhalb der angegebenen Ergebnisfelder werden gewertet! Lassen Sie etwaige Stellen frei.

```
#include <stdio.h>
int main ()
{
   int *piZeiger = NULL;
   int iAFeld[4] = {7,4,9,2};

   iAFeld[3] = 0;
   piZeiger = &iAFeld[1];
   *(piZeiger++) = 8;
   *(piZeiger--) += *(piZeiger-1) * 3;
   *(--piZeiger) = iAFeld[0] | 12;

  return 0;
}
```

***Bild C-21.1**: C-Programm zur Zeigermanipulation*

21.1 **iAFeld[0]** nach Ausführung des C-Programms in **Bild C-21.1**

iAFeld[0]: 15

iAFeld[0]: ⋯

21.2 **iAFeld[1]** nach Ausführung des C-Programms in **Bild C-21.1**

iAFeld[1]: 08 || iAFeld[1]: 8

iAFeld[1]: ⋯

21.3 **iAFeld[2]** nach Ausführung des C-Programms in **Bild C-21.1**

iAFeld[2]: 33

iAFeld[2]: ⋯

21.4 **iAFeld[3]** nach Ausführung des C-Programms in **Bild C-21.1**

iAFeld[3]: 0 || iAFeld[3]: 00

iAFeld[3]: ⋯

22. MATLAB: Simulink 2D-Lookup-Table

Gegeben ist das in **Bild MATLAB-22.1** angegebene Motorkennfeld als Matrixvariable *X* mit der Dimension 8x6, welches in den MATLAB Workspace geladen wurde und den Verbrauch eines Dieselmotors in Gramm pro Sekunde (g/s) ausgibt. Der *Verbrauch* wird dabei entlang der zwei Eingangsparameter *Gaspedalstellung* (in %), sowie der aktuell eingestellten *Drehzahl* (in Umdrehungen pro Minute U/Min) bestimmt.

Hinweis: Nur Lösungen innerhalb der angegebenen Ergebnisfelder werden gewertet!

Breakpoints 1

Break-points 2	1 ⇩	2	3	4	5	6
1 ⇨	0	0	1000	2000	3000	4000
2	0	0	120	150	150	180
3	0.1000	0	120	150	150	190
4	0.2000	0	130	180	200	210
5	0.4000	0	120	210	230	210
6	0.6000	0	100	200	260	230
7	0.8000	0	50	170	250	220
8	1	0	0	150	200	180

Gaspedal-stellung [0–1]

Drehzahl [U/Min]

Verbrauch [g/s]

***Bild MATLAB-22.1**: Motorkennfeld als Matrixvariable X*

22. MATLAB: Simulink 2D-Lookup-Table [Fortsetzung]

Dazu wurde das in **Bild MATLAB-22.2** aufgezeigte Modell mittels einer 2D-Lookup-Table in Simulink erstellt.

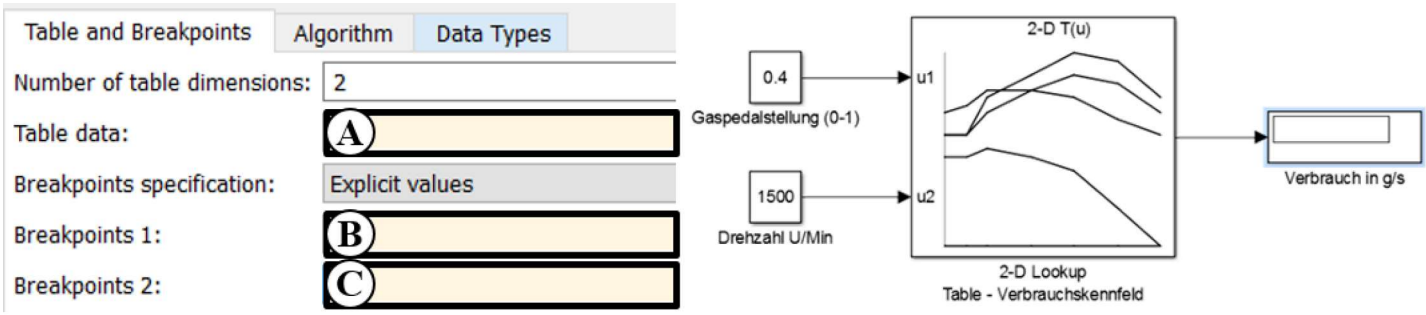


***Bild MATLAB-22.2**: 2D-Lookup-Table*

22.1 Berechnen Sie den Momentanverbrauch mittels linearer Interpolation, welcher sich wie in **Bild MATLAB-22.1** aus einer Gaspedalstellung von 0.4 und einer Drehzahl von 1500 U/Min ergibt.

Verbrauch: 165 g/s

Verbrauch: [ ] [ ] [ ] g/s

22.2 Welche mathematische Operation muss durchgeführt werden um den nach Ablauf der Simulation entstanden Gesamtverbrauch zu erhalten?
Hinweis: Nur Einfachnennung möglich.

☐ Addieren ☑ Integrieren ☐ Differenzieren

Weisen Sie in den folgenden Aufgaben anhand der in **Bild MATLAB-22.1** angegebenen Informationen zur Parametrierung der 2D-Lookup-Table den **Lücken A bis C in Bild MATLAB-22.2** den jeweils korrekten Ausdruck zu.

Hinweis: Jeweils nur Einfachnennung möglich.

22.3 Lücke **A in Bild MATLAB-22.2**

☐ X(1:end,1:end)	☑ X(2:end,2:end)	☐ X
☐ X(2:end,1)	☐ X(1:end)	☐ X(1:8,1:6)
☐ X(1,2:end)	☐ X(1:6,1)	

22.4 Lücke **B in Bild MATLAB-22.2**

☐ X(1:end,1:end)	☐ X(2:end,2:end)	☐ X
☑ X(2:end,1)	☐ X(1:end)	☐ X(1:8,1:6)
☐ X(1,2:end)	☐ X(1:6,1)	

22.5 Lücke **C in Bild MATLAB-22.2**

☐ X(1:end,1:end)	☐ X(2:end,2:end)	☐ X
☐ X(2:end,1)	☐ X(1:end)	☐ X(1:8,1:6)
☑ X(1,2:end)	☐ X(1:6,1)	

23. MATLAB: Steuerung einer Anlage zum Werkstücktrennen mit Stateflow

Eine Anlage zum Trennen von Werkstücken (WS) (siehe **Bild MATLAB-23.1**) soll von einem Zustandsautomaten in Stateflow gesteuert werden. Sie besitzt dazu vier Sensoren, sowie zwei Gatter-Aktoren (siehe **Bild MATLAB-23.2, rechts**). Weiterhin soll die in **Bild MATLAB-23.2, links** dargestellte Wahrheitstabelle umgesetzt werden, um die Werkstücke dem entsprechenden Kunden A, B oder in eine Schachtel für Ausschuss zuzuteilen. Die Sensoren sind int-Variablen welche die Werte 0 oder 1 einnehmen können und wie in **Bild MATLAB-23.2, rechts** angegeben belegt werden.

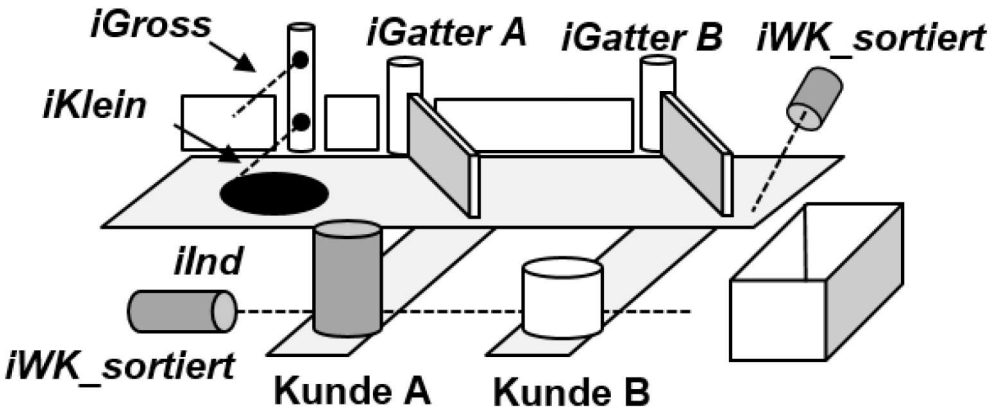


***Bild MATLAB-23.1**: Skizze der Anlage zum Trennen von Werkstücken (WS)*

Wahrheitstabelle

iGross	iKlein	iInd	Ausgang
0	1	0	Kunde B
0	1	1	Ausschuss
1	X	0	Ausschuss
1	X	1	Kunde A

X : Don‘t-care-Bit

Sensoren und Aktoren

Aktoren	iGatterA	Schaltet das Gatter A auf „Ausleiten“.
	iGatterB	Schaltet das Gatter B auf „Ausleiten“.
	iBand	Fährt das Laufband.
Sensoren	iKlein	Die Lichtschranke erkennt ein kleines WS.
	iGross	Die Lichtschranke erkennt ein hohes WS.
	iInd	Der Induktionssensor erkennt ein metallisches WS.
	iWK_sortiert	Eine der Lichtschranken erkennt ein WS an den Endlagen.

***Bild MATLAB-23.2**: Wahrheitstabelle sowie Sensoren und Aktoren der Werkstücktrennung*

Füllen Sie in den folgenden Aufgaben die **Lücken A bis D in Bild MATLAB-23.3** unter Zuhilfenahme der **Lösungsmöglichkeiten 1 bis 8 in Bild MATLAB-23.4**. Ordnen Sie hierzu die Nummer der jeweiligen Lösungsmöglichkeit zu.

Hinweis: Nur Lösungen innerhalb der angegebenen Ergebnisfelder werden gewertet!

23. MATLAB: Steuerung einer Anlage zum Werkstücktrennen mit Stateflow [Fortsetzung]

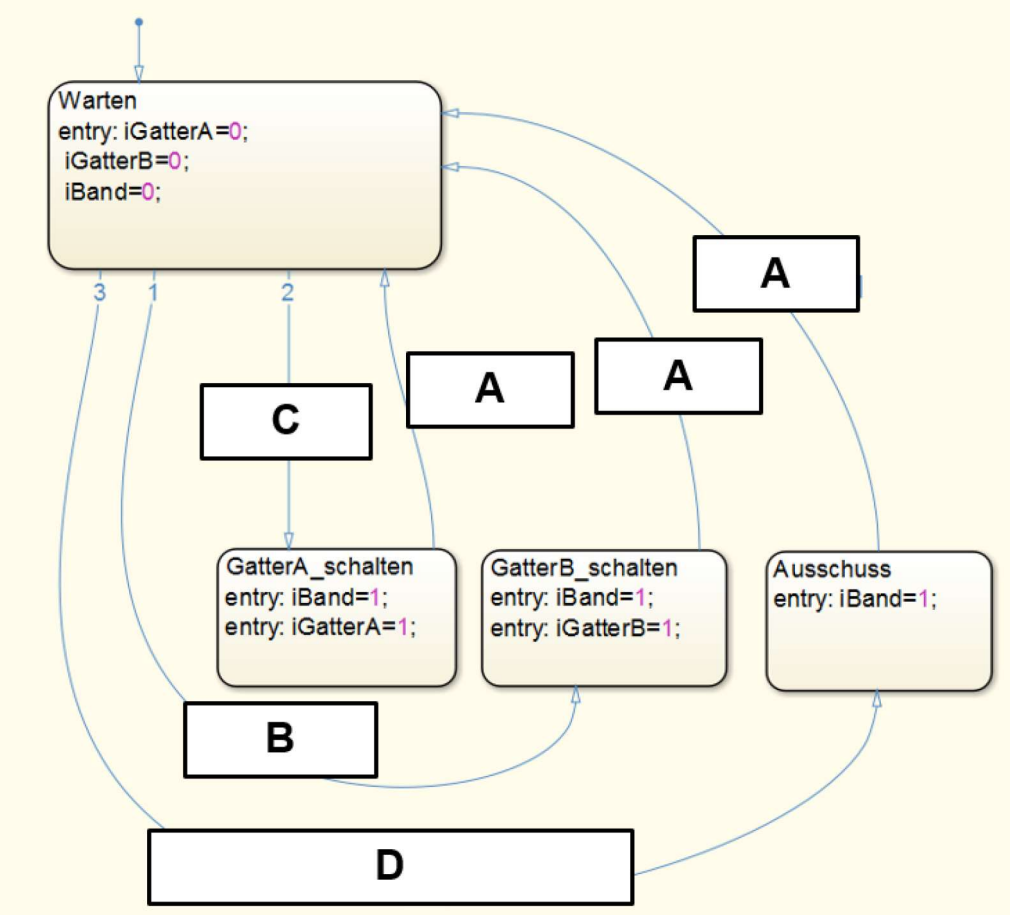


***Bild MATLAB-23.3**: Stateflow-Statechart*

23.1 Lücke **A in Bild MATLAB-23.3**

Lücke A: 2

Lücke A: ⬚

23.2 Lücke **B in Bild MATLAB-23.3**

Lücke B: 4

Lücke B: ⬚

23.3 Lücke **C in Bild MATLAB-23.3**

Lücke C: 3

Lücke C: ⬚

23.4 Lücke **D in Bild MATLAB-23.3**

Lücke D: 8 || 5 || Lücke D: 5 || 8

Lücke D: ⬚ || ⬚

1	iBand = 0;
2	iWK_sortiert
3	iGross && iInd
4	~iGross && iKlein && ~iInd
5	~iGross && iKlein && iInd
6	iGross || !iKlein && !iInd
7	!iWK_sortiert
8	iGross && ~iInd

***Bild MATLAB-23.4**: Lösungsmöglichkeiten*