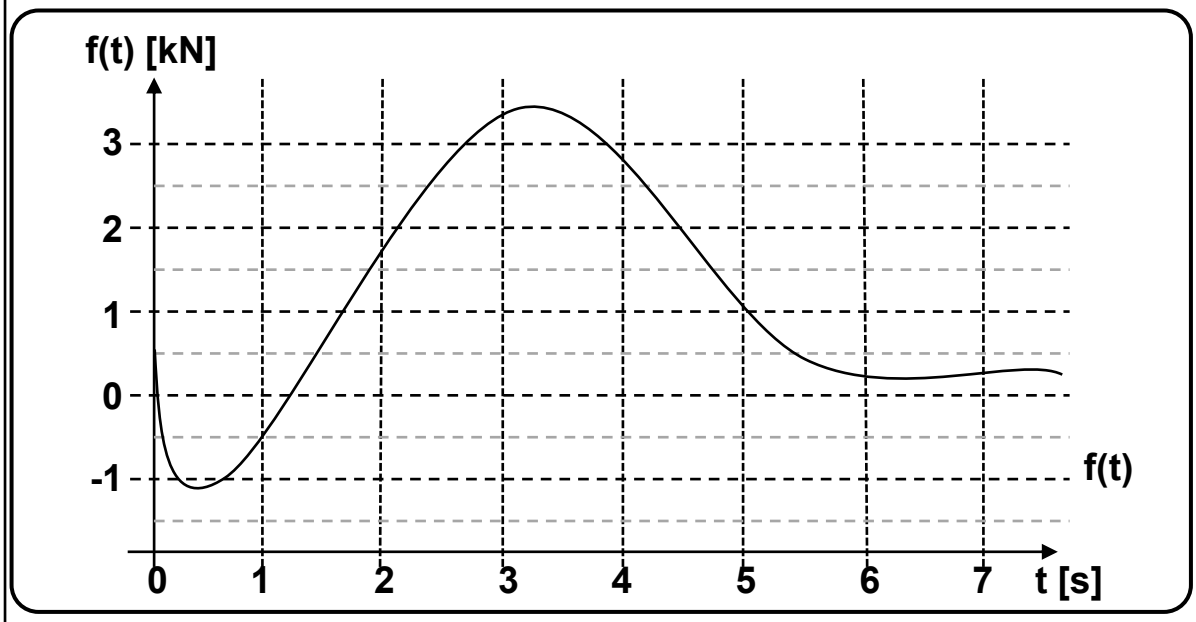
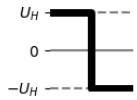
Aufgabe 1: Grundlagen der Informationstechnik und Digitaltechnik

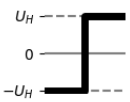
a) Gegeben ist das dargestellte, wert- und zeitkontinuierliche Signal f(t). Zeichnen Sie den **wert-** und **zeitdiskreten** Signalverlauf von f(t) für $0 \leq t \leq 7$ in das Schaubild ein. Die Diskretisierung erfolgt jede Sekunde [s] auf ganzzahlige kN, wobei alle **Nachkommastellen abgeschnitten** werden (**d.h. -2,8 → -2 und 2,8 → 2**).

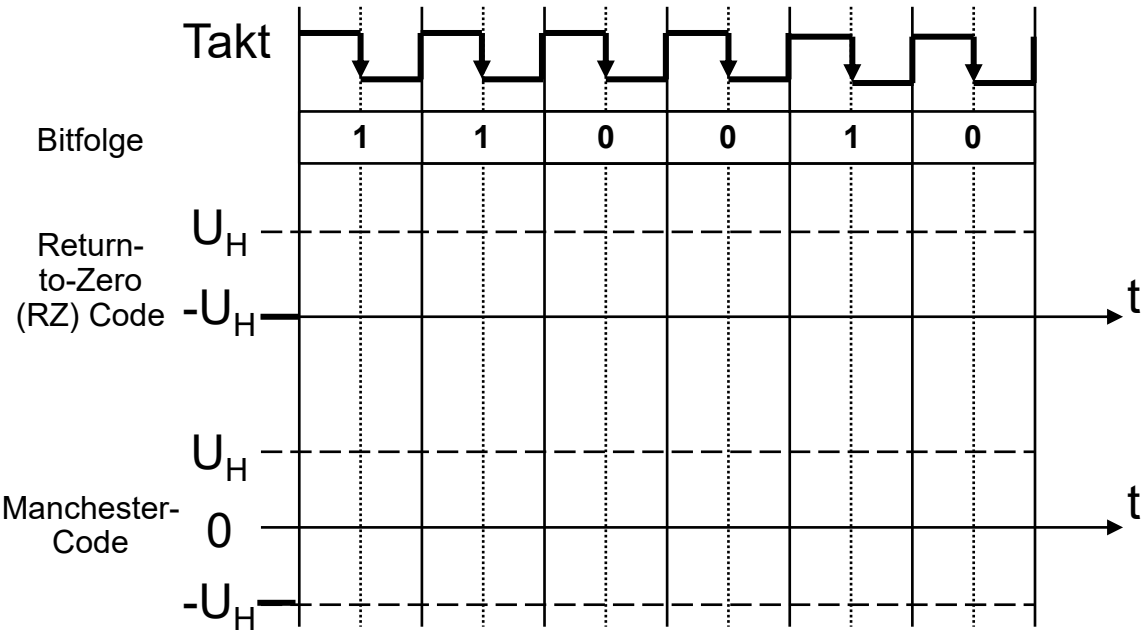


b) Sie versenden die **Bitfolge `110010`** auf einem seriellen Bussystem. Zeichnen Sie den resultierenden Leitungscode als Return-to-Zero Code und im Manchester-Code. Für beide Codes liegt **zu Beginn $-U_H$** an.

Hinweis:

Manchester-Code: **1**: 

0: 



Aufgabe 2: IEEE 754 Gleitkommadarstellung und Zahlensysteme

a) Rechnen Sie die Dezimalzahl $\mathbf{(-31{,}75)_{10}}$ in eine Gleitkommazahl (angelehnt an die IEEE 754 Darstellung) um, indem Sie die folgenden Textblöcke ausfüllen.

Hinweis: Ergebnisse und Nebenrechnungen außerhalb der dafür vorgesehenen Textblöcke werden nicht bewertet!

Vorzeichen V 1 Bit	Biased Exponent E 3 Bit	Mantisse M 6 Bit

Vorzeichenbit

Dezimalzahl $(31{,}75)_{10}$ als Binärzahl

Bias als Dezimalzahl

Exponent als Dezimalzahl

Biased Exponent als Dualzahl

Vollständige Gleitkommazahl $(-31{,}75)_{10}$ (nach obigem Schema)

Vorzeichen	**Biased Exponent**	**Mantisse**

b) Überführen Sie die unten gegebenen Zahlen in die jeweils anderen Zahlensysteme.
Hinweis: Achten Sie genau auf die jeweils angegebene Basis.

1 $\mathbf{(1011\ 1101)_2 = (\ \_\_\_\_\_\ )_8 = (\ \_\_\_\_\_\ )_{16}}$

2 $\mathbf{(424)_5 = (\ \_\_\_\_\_\ )_{10}}$

Aufgabe 3: Logische Schaltungen und Schaltbilder

a) Vervollständigen Sie das Schaltbild, um die vollständige disjunktive Normalform (DNF) passend zur Wahrheitstabelle (Tabelle 3.1) zu erhalten.

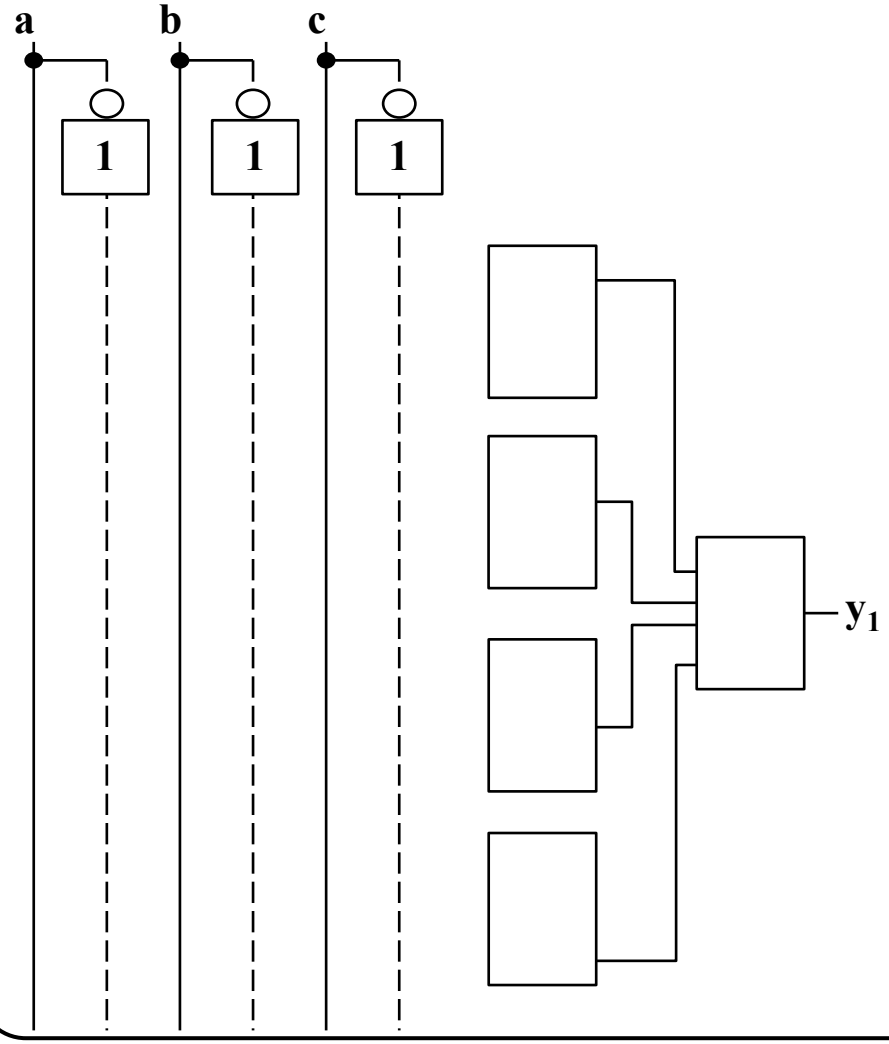


Tabelle 3.1:
Wahrheitstabelle

a	b	c	y_1
1	1	1	0
1	1	0	0
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	1
0	0	0	1

b) Stellen Sie für die Wahrheitstabelle (Tabelle 3.1) die zugehörige Disjunktive Normalform (DNF)-Gleichung auf.

c) Minimieren Sie folgende Disjunktive Normalform (DNF)-Gleichung:

$$y_2 = (a \vee \overline{a})bc \vee \overline{b}c$$

Aufgabe 4: FlipFlops

Gegeben ist folgende Primary-Secondary-(Master-Slave)-Flip-Flop-Schaltung (Bild 4.1):

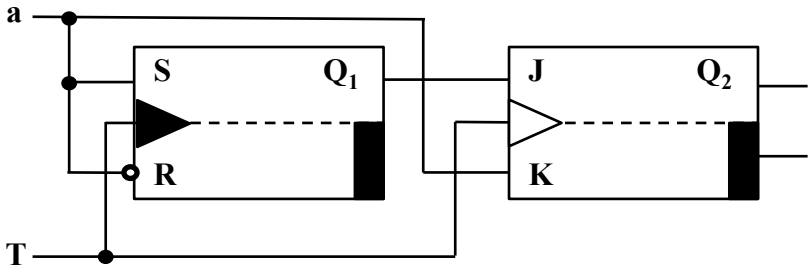
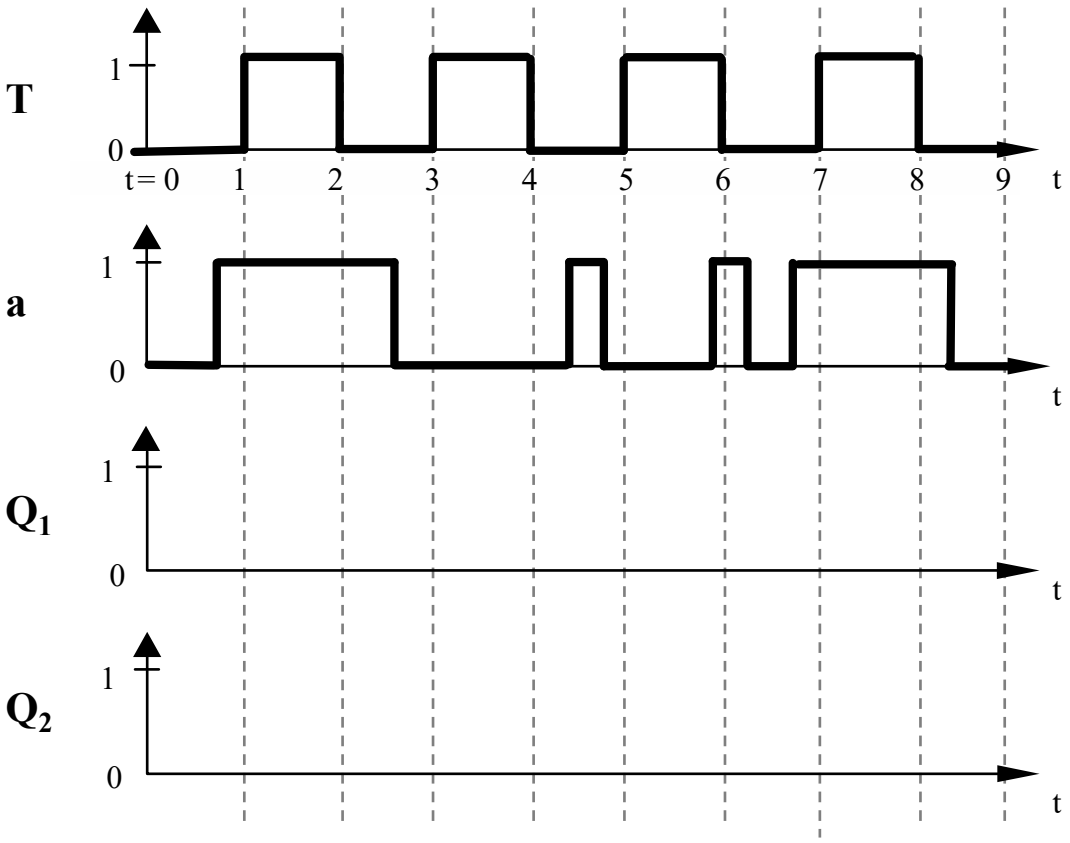


Bild 4.1: PS(MS)-FF

Bei $t = 0$ sind die Flip-Flops in folgendem Zustand: $Q_1 = Q_2 = 0$.

Analysieren Sie die Schaltung für den Zeitbereich $0 \leq t \leq 8$, indem Sie für die Eingangssignale a und T die zeitlichen Verläufe für Q_1 und Q_2 in die vorgegebenen Koordinatensysteme eintragen.

Hinweis: Signallaufzeiten können bei der Analyse vernachlässigt werden.



Aufgabe 5: Automaten

a) Gegeben ist der in Bild 5.1 gezeigte Automat. Kreuzen Sie für I. bis III. die jeweils zutreffende Aussage an.

Der in Bild 5.1 gezeigte Automat …

I. ( ) … ist ein Mealy-Automat
( ) … ist ein Moore-Automat

II. ( ) … ist deterministisch
( ) … ist nicht-deterministisch

III. ( ) … hat den Startzustand Z1
( ) … hat den Startzustand Z2
( ) … hat den Startzustand Z3

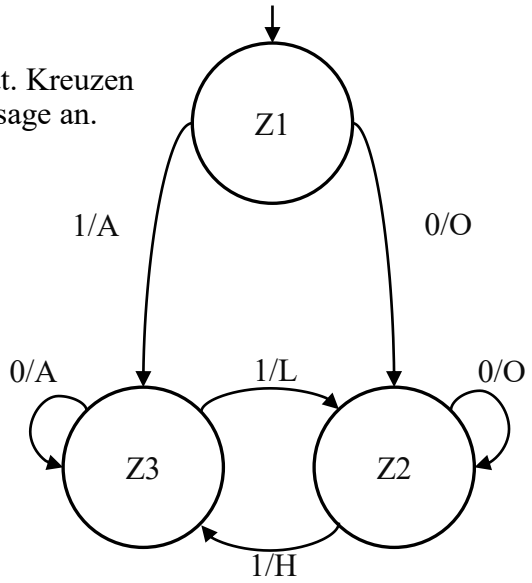


Bild 5.1: *Automat*

b) Erstellen Sie zu dem abgebildeten Automaten die zugehörige Übergangstabelle. Tragen Sie hierzu Zustand und Transition ein.

T \ Z	Z1	Z2	Z3
0			
1			

c) Nennen Sie eine Eingabe, für die Sie ausgehend von Zustand Z1 die Ausgabe „ALOHA“ erhalten.

d) Zeichnen Sie den Automaten aus Tabelle 5.1 mit Z1 als Startzustand.

Tabelle 5.1: *Übergangstabelle*

T \ Z	Z1, 1	Z2, 2	Z3, 3
0		Z1	Z2
1	Z2	Z3	

Aufgabe 6: MMIX – Assembler-Code

Der folgende Algorithmus, ein Auszug aus der MMIX-Codetabelle (Tabelle 6.1) und ein Auszug aus einem Registerspeicher (Tabelle 6.2) sind gegeben.

	0x_0	0x_1	…	0x_4	0x_5	…
	0x_8	0x_9		0x_C	0x_D	
…	…	…	…	…	…	…
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	

Tabelle 6.1: *MMIX-Code-Tabelle*

Algorithmus:

$$y = \frac{(a+b)^2 - 32}{c} \bullet b$$

Registerspeicher		
Adresse	Wert vor Befehlsausführung	Kommentar
…	…	…
$0x70	0x00 00 00 00 00 00 00 04	Nicht veränderbar
$0x71	0x00 00 00 00 00 00 A0 06	Nicht veränderbar
$0x72	0x00 00 00 00 00 00 EE F1	Variable a
$0x73	0x00 00 00 00 00 00 D0 20	Variable b
$0x74	0x00 00 00 00 00 00 A0 0C	Variable c
$0x75	0x00 00 00 00 00 00 00 BB	Variable y
$0x76	0x00 00 00 00 00 00 10 EF	Zwischen-ergebnis
…	…	…

Tabelle 6.2: *Registerspeicher*

Der Registerspeicher eines MMIX-Rechners enthält zunächst die in Tabelle 6.2 angegebenen Werte. In der zusätzlichen Spalte Kommentar ist angegeben, welche Daten die jeweilige Registerzelle enthält und wofür sie zu verwenden ist.

a) Führen Sie den gegebenen Algorithmus aus. Verwenden Sie dazu nur die in Tabelle 6.1 **umrandeten Befehlsbereiche**. **Speichern Sie die Zwischenergebnisse** nach jedem Befehl des Algorithmus in der Registerzelle mit dem Kommentar **Zwischenergebnis**. Das Endergebnis ist in der Registerzelle mit dem Kommentar **Variable y** zu speichern. Übersetzen Sie die Operationen in Assemblercode mit insgesamt **maximal 5 Befehlen**.

1	
2	
3	
4	
5	

b) Im Folgenden wird davon ausgegangen, dass der Inhalt des Datenspeichers dem Zustand in Tabelle 6.3 entspricht. Laden Sie ein **Tetra** von der Speicheradresse M[0x0…A0 0A] in die Variable a im Registerspeicher (Tabelle 6.2).

Hinweis: Für diese Teilaufgabe gilt die Big-Endian Adressierung.

Geben Sie den dafür benötigten Assembler-Befehl an. Nutzen Sie für die Adressierung **ausschließlich Werte des Registerspeichers** (Tabelle 6.2), keine Sofort-operanden:

Ab welcher Adresse wird aus dem Datenspeicher gelesen?

Wie lautet der 64-Bit-Wert der Variablen a nach dem Ladevorgang in Hexadezimalschreibweise?

Datenspeicher	
Adresse	Wert
…	…
M[0x00 … A0 05]	0xEE
M[0x00 … A0 06]	0xAA
M[0x00 … A0 07]	0x56
M[0x00 … A0 08]	0x01
M[0x00 … A0 09]	0x0C
M[0x00 … A0 0A]	0x3C
M[0x00 … A0 0B]	0x7D
M[0x00 … A0 0C]	0x8A
M[0x00 … A0 0D]	0x11
M[0x00 … A0 0E]	0x32
M[0x00 … A0 0F]	0x98
M[0x00 … A0 10]	0x0F
M[0x00 … A0 11]	0xC7
…	…

Tabelle 6.3: *Datenspeicher*

c) Übersetzen Sie den folgenden Befehl von Maschinensprache (hexadezimal) in Maschinensprache (binär) und in Assemblercode.

Hinweis: Nutzen Sie die Informationen aus Tabelle 6.1.

Maschinensprache (hexadezimal): **0x 2 1 7 2 A B F F**

Maschinensprache (binär):

Assemblercode:

d) Kreuzen Sie an, ob die folgenden Aussagen zur MMIX-Systemarchitektur wahr oder falsch sind.

	Wahr	**Falsch**
1. Eine tetraweise Zugriffsart auf den Datenspeicher entspricht der Codierung 0x3.	**( )**	**( )**
2. Die Auswahl der ALU-Operation ist eine Aufgabe der Funktionseinheit Steuerung.	**( )**	**( )**

Aufgabe 7: Round-Robin-Scheduling

Gegeben ist der folgende Soll-Verlauf der vier Tasks A, B, C und D (Bild 7.1). Die Einplanung der Tasks soll **präemptiv mit festen Prioritäten** erfolgen. Für jedes Prioritätslevel (Tabelle 7.2) 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 2s. Restzeiten können nicht übersprungen werden.

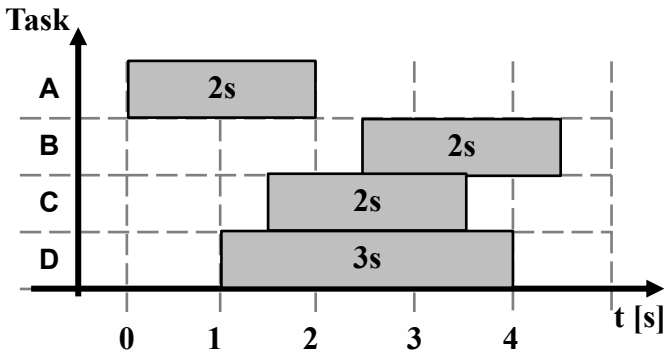


Bild 7.1: *Einplanung / Soll-Verlauf der Tasks*

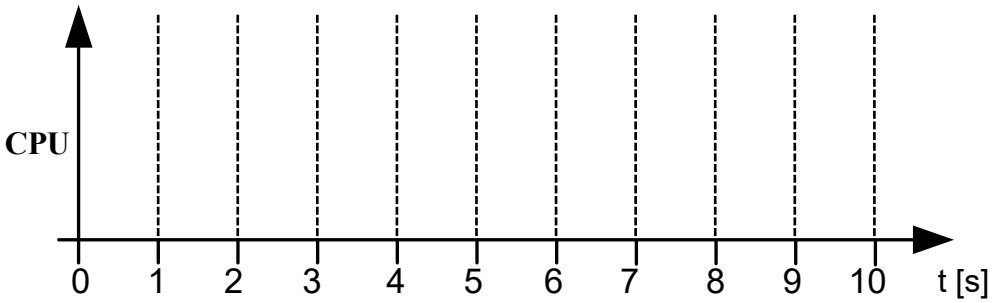
Task	Priorität	Deadline
A	3	t = 6 s
B	2	t = 9.5 s
C	1	t = 5.8 s
D	1	t =7 s

1 ist höchste Priorität

Tabelle 7.2: *Prioritätenverteilung und Deadline je Task*

a) Tragen Sie für jeden 1s-Zeitraum im Intervall $0 \leq t \leq 10$ ein, welcher Task auf der CPU läuft. Läuft kein Task auf der CPU, tragen Sie „-“ ein.

Ist-Verlauf der Tasks auf der CPU:



b) Verwenden Sie den Ist-Verlauf der Tasks aus Bild 7.3. Nennen Sie alle Tasks aus Tabelle 7.2, die die Deadline eingehalten haben.

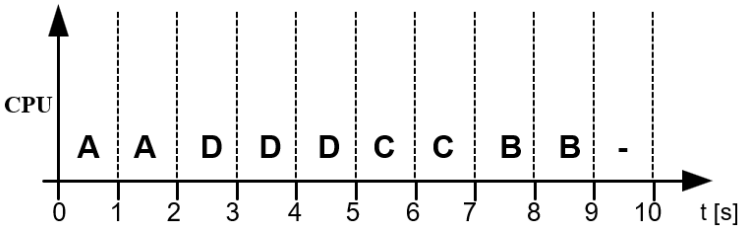


Bild 7.3: *Angenommener Ist-Verlauf der Tasks*

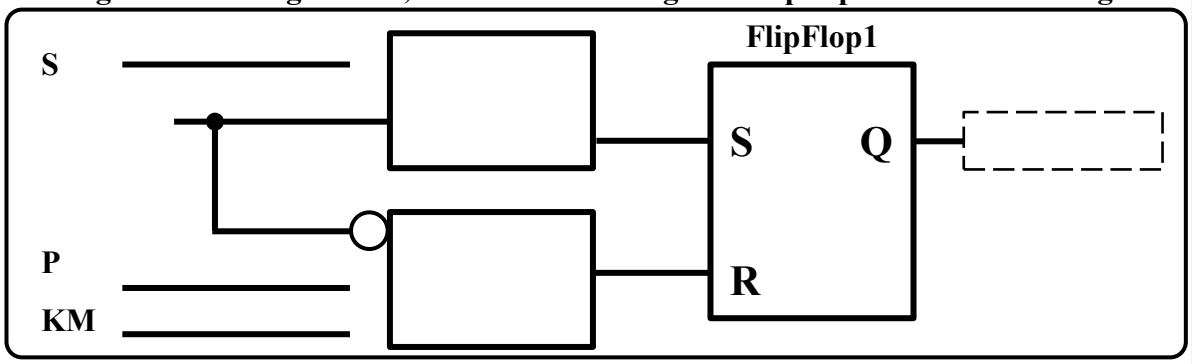
Aufgabe 8: IEC 61131-3 Funktionsbausteinsprache und Ablaufsprache

a) Ergänzen Sie das untenstehende Programm für die Steuerung eines Wasserkühlers in einem Produktionsprozess in IEC 61131-3 Funktionsbausteinsprache (FBS).

- Der Wasserkühler startet (**`WK=1`**), wenn der Sensor**taster (S)** betätigt wird und der Notaus inaktiv ist (**`Notaus=1`**). Stellen Sie sicher, dass ein dauerhaft gedrückter Sensortaster den Wasserkühler nicht mehrfach startet.
- Der Wasserkühler stoppt (**`WK=0`**), falls der Notaus aktiv ist (**`Notaus=0`**) oder der aktuelle Arbeitsvorgang beendet ist (**`P auf 0 geschaltet`**). Der Wasserkühler stoppt auch immer wenn die Warnleuchte für Kühlwassermangel (**`KM=1`**) an ist. Der Start des Wasserkühlers wird nicht behindert, wenn der Arbeitsvorgang noch nicht gestartet wird (`dauerhaft` **`P=0`**). Im Zweifel wird der Wasserkühler immer gestoppt.

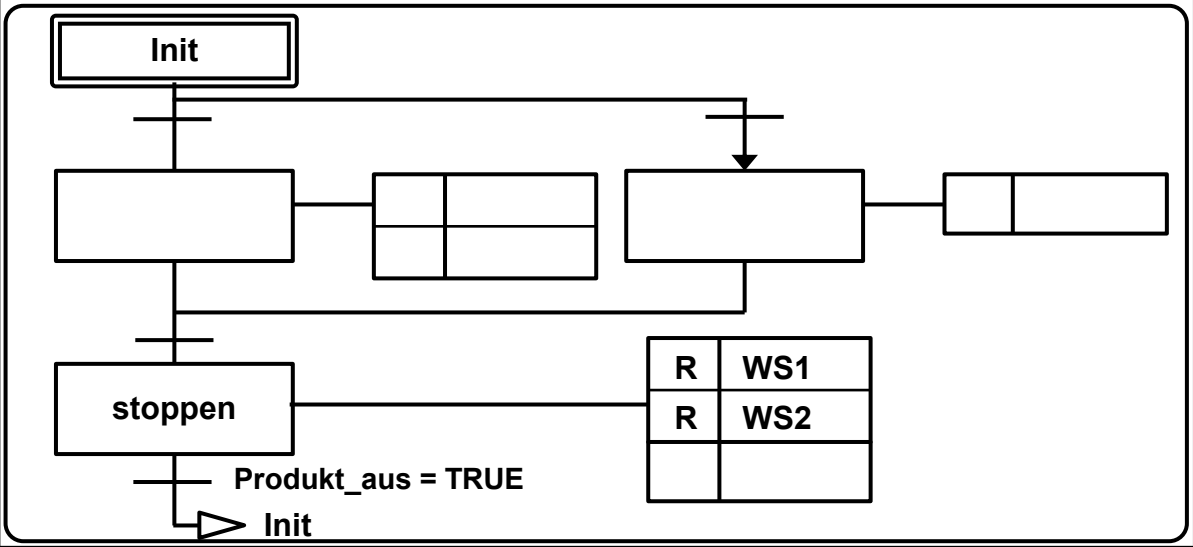
Hinweise:

- Signalverzögerungen im System sind zu vernachlässigen.
- Verwenden Sie **keine** Schaltglieder außer den in der Vorlage bereits vorhandenen.
- **Ergänzen Sie Negationen, Flankenerkennung und Flipflopart falls notwendig**.



b) Zwei Wasserkühler werden in ein automatisch gesteuertes Kühlsystem integriert. Stellen Sie das folgende Verhalten in IEC 61131-3 Ablaufsprache (AS) dar.

- Liegt die Temperatur **`T`** über oder gleich 90°C (**`T>=90`**), werden Wasserkühler 1 (**`WS1`**) und Wasserkühler 2 (**`WS2`**) eingeschaltet (Schritt **`kühlen_s`**). Liegt **`T`** unter 90°C (**`T<90`**), wird nur Wasserkühler 1 (**`WS1`**) eingeschaltet (Schritt **`kühlen_n`**). Sinkt **`T`** auf oder unter 40°C (**`T<=40`**), wird der Schritt **stoppen** aktiviert. Im Schritt **`stoppen`** werden beide Wasserkühler (**`WS1`** und **`WS2`**) ausgeschaltet. Nur im Schritt **`stoppen`** ist der **`Pusher`** für Werkstückausstoß aktiv. Nach Verlassen des Schritts **`stoppen`** wird der **`Pusher`** zurückgesetzt.



Aufgabe 9: Echtzeitprogrammiersprache PEARL

Vervollständigen Sie den untenstehenden Codeausschnitt einer Kaffeemaschine in PEARL gemäß den Kommentaren über den Lücken.

```
// Deklarieren Sie die Semaphor-Variable „Auslass“ mit dem Startwert 1.

_____________ Auslass _____________________________ ;

// Deklarieren Sie die Unterbrechung „Stopp_manuell“ für einen
// manuellen Stopp per Knopfdruck.

_____________________________________________________ ;

// Deklarieren Sie den Task „Milchschaum_erzeugen“ mit Priorität 2.
Milchschaum_erzeugen: _____________________________________ ;

// Überprüfen Sie den Status der Semaphor-Variable „Auslass“.
   _____________________________ Auslass ;

// Aktivieren Sie von nun an die Unterbrechung „Stopp_manuell“.

   _____________________________________________;

// Ist genügend Milch vorhanden (Milch_vorhanden = TRUE), wird der
// Task „Aufschaeumen“ aktiviert
_____ Milch_vorhanden _____________________________ Aufschaeumen;
   FIN;

// Bei Auftreten der Unterbrechung „Stopp_manuell“ wird der Task
// „Aufschaeumen“ gestoppt.

   _____________________________________________________

   _____________________________________________________ ;

// Der Task „Abfuellen“ wird nach 15 Sekunden ausgeführt.

   _____________________________________________ Abfuellen ;

    RELEASE Auslass ;
END;
```

Aufgabe 10: UML Use Case Diagramm

Wie in Bild 10.1 dargestellt, wird in einem **`Lagerhaus`** nach dem Eingang eines Kundenauftrags das entsprechende Produkt vom **`Lager`** zur **`Verpackungsstation`** transportiert. Dieser Prozess wird durch ein fahrerloses Transportsystem mit Greifarm (**`FTS-Greifarm`**) automatisiert.

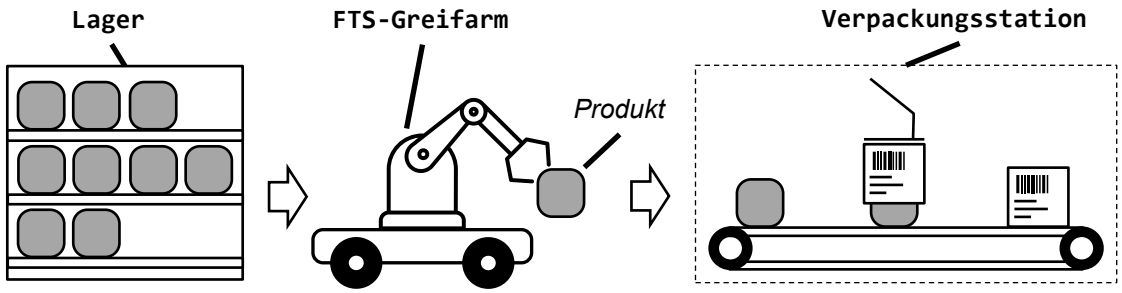


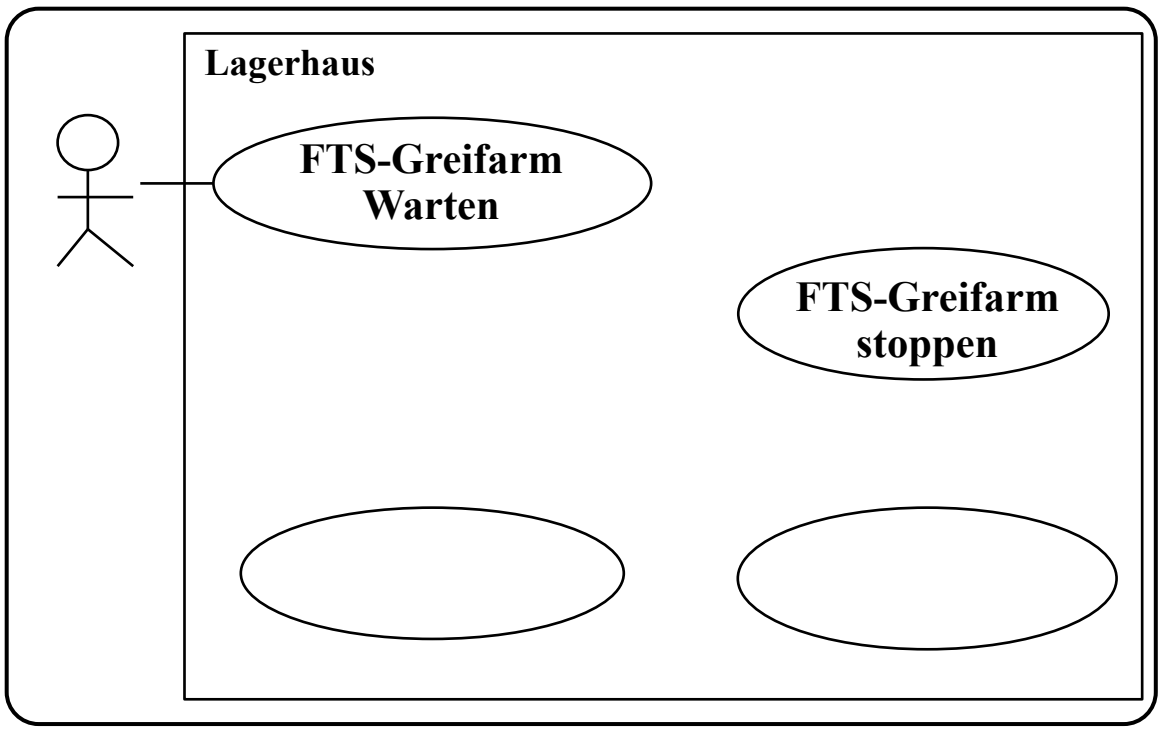
Bild 10.1: *Schematische Darstellung des automatisierten Transportprozesses im Lagerhaus*

Zur Gewährleistung des Transports von Produkten muss ein **`Arbeiter`** regelmäßig den **`FTS-Greifarm Warten`**.

Für den Wartungsvorgang muss der Arbeiter zuerst den **`FTS-Greifarm stoppen`**, um die weitere Bewegung des FTS-Greifarms zu verhindern. Danach muss der Arbeiter das vom FTS-Greifarm transportierte **`Produkt entfernen`**.

Um den Transportprozess zwischen dem Lager und der Verpackungsstation während der FTS-Greifarmwartung nicht zu unterbrechen, kann der Arbeiter *optional* das betroffene **`Produkt manuell transportieren`**.

Vervollständigen Sie das untenstehende UML Use Case Diagramm gemäß der oben beschriebenen Anwendungsfälle.



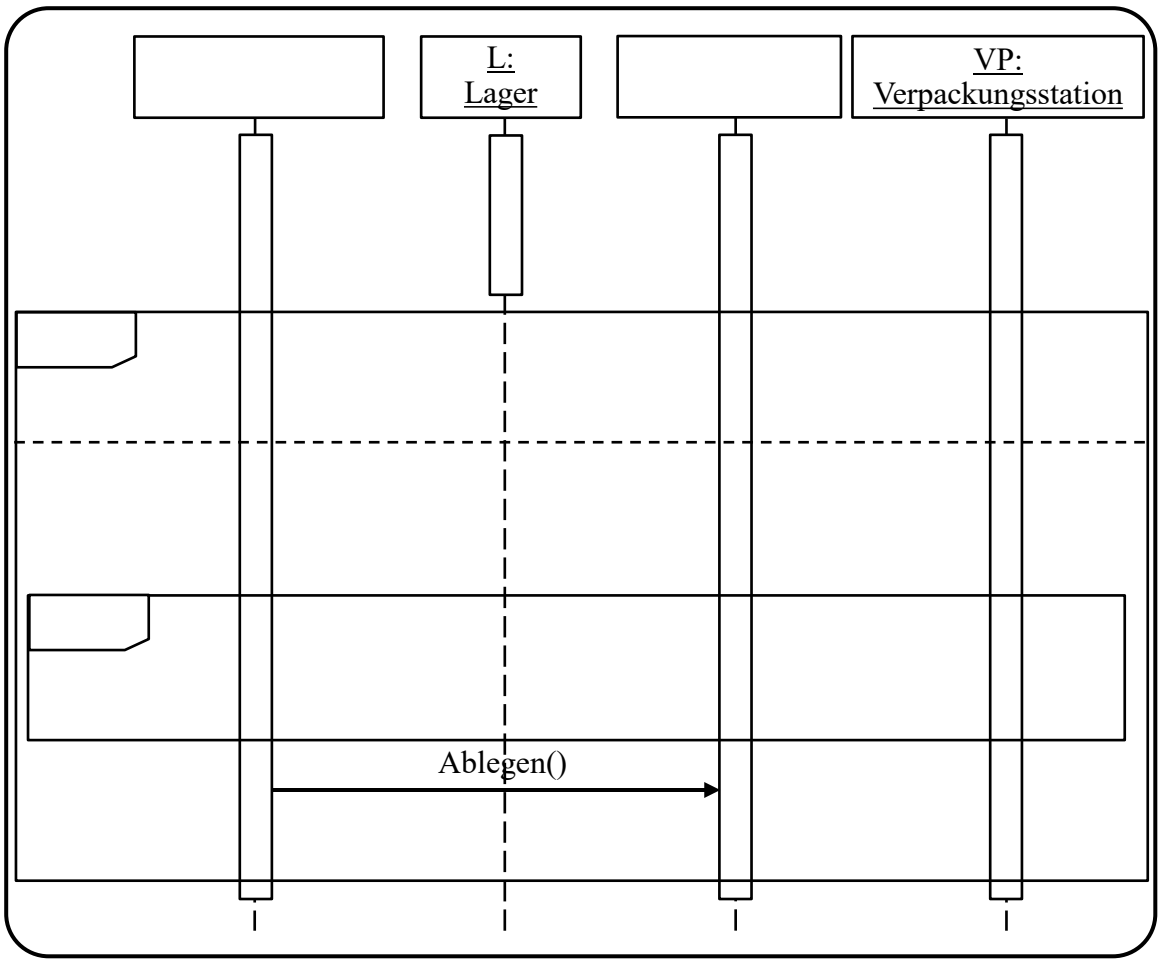
Aufgabe 11: UML-Sequenzdiagramm

Der in Bild 10.1 dargestellte Transportprozess zwischen dem Lager und der Verpackungsstation soll in einem Sequenzdiagramm modelliert werden. Hierzu werden Objekte der Klassen **`Versandsystem (Objekt VS)`**, **`Lager (Objekt L)`**, **`FTS-Greifarm (Objekt FG)`** und der **`Verpackungsstation (Objekt VP)`** benötigt.

Vervollständigen Sie das folgende Sequenzdiagramm entsprechend der untenstehenden Beschreibung. Achten Sie darauf, dass die Pfeilspitzen den benötigten Nachrichtentypen entsprechen.

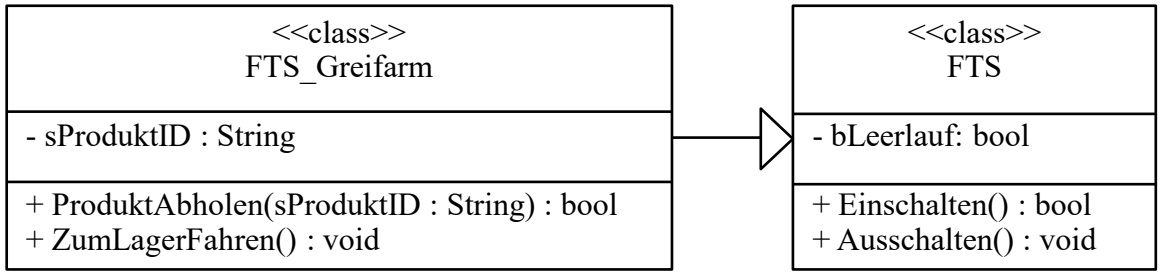
Das **`Versandsystem`** fragt zunächst das **`Lager`** nach der Produktverfügbarkeit ab (**`Abfragen()`**) und wartet auf dessen **`Antwort "Verfügbarkeit"`**.

- Wenn das Produkt nicht verfügbar ist (d.h. **`bVerfügbar==false`**), storniert das **`Versandsystem`** den Kundenauftrag durch eine Selbstnachricht (**`Stornieren()`**).
- Wenn das Produkt verfügbar ist (**`bVerfügbar==true`**), fordert das **`Versandsystem`** den **`FTS-Greifarm`** auf, das Produkt aus dem Lager abzuholen (**`Abholen()`**), und empfängt dessen **`Antwort "Abgeholt"`**.
- Anschließend fordert das **`Versandsystem`** die **`Verpackungsstation`** über die asynchrone Nachricht so lange auf, den Platz für das ankommende Produkt freizuhalten (**`Freihalten()`**), wie die Bedingung **`bStationFrei==false`** erfüllt ist.
- Danach fordert das **`Versandsystem`** den **`FTS-Greifarm`** auf, das Produkt an der Verpackungsstation abzulegen (**`Ablegen()`**) und wartet auf die entsprechende **`Antwort "Abgelegt"`**.



Aufgabe 12: Überführung eines UML-Klassendiagramms in C++-Code

Folgendes Klassendiagramm ist gegeben:



Ergänzen Sie die Klassendeklarationen der zwei dargestellten Klassen **FTS** und **FTS_Greifarm** in der Programmiersprache C++.
Hinweise:
Alle benötigten Header-Dateien sind bereits eingebunden.
Deklarieren Sie auch die default Konstruktoren und Destruktoren.

```
using namespace std;
class FTS {
private:_____________________________________
_____________________________________
_____________________________________
_____________________________________
_____________________________________
_____________________________________
};

class _________________ : _____________ FTS{
private:_____________________________________
_____________________________________
_____________________________________
_____________________________________
_____________________________________
_____________________________________
_____________________________________
};
```

Aufgabe 13: UML-Zustandsdiagramm

Im Folgenden wird der Prozessablauf (vgl. Bild 13.2) für die Verpackung eines Produktes in der Verpackungsstation (Bild 13.1) beschrieben.

Die Verpackungsstation befindet sich nach dem Start im Zustand **`Wartend`**, bis eine Steuerungsanweisung (**`command`**) mit Wert **`"start"`** erfolgt (**`command=="start"`**). Dann geht die Verpackungsstation in den Zustand **`ProduktAufnehmend`** über.

In dem Zustand **`ProduktAufnehmend`** wird das **`Förderband`** das Produkt dauerhaft nach rechts befördern (**`bFBVorwärts=1`**). Wenn das Produkt in der Mitte des Förderbandes ankommt, gibt die Lichtschranke_1 einen Messwert (**`bLS1`**) von **`TRUE`** zurück. Daraufhin wird das **`Förderband`** wieder gestoppt (**`bFBVorwärts=0`**) und die Verpackungsstation geht in den Zustand **`ProduktVerpackend`** über.

Bei Eintritt in den Zustand **`ProduktVerpackend`**, soll der **`Schließer`** die Verpackung an das Produkt schließen (**`bSchließerAn=1`**). Der Zustand **`ProduktVerpackend`** kann durch zwei Fälle verlassen werden:

Fall 1: Sollte die mit dem Scanner gescannte Etiketten-ID (**`iEtikettenID`**) mit der Kunden-ID (**`iKundenID`**) übereinstimmen, so wechselt die Verpackungsstation in den Zustand **`ProduktWeiterleitend`**.

Fall 2: Wenn nach **`2 Sekunden`** die gescannte Etiketten-ID noch nicht mit der Kunden-ID übereinstimmt, so wechselt die Verpackungsstation in den Zustand **`ProduktRücksendend`**. hierzu wird die Timer-Variable **`iTimer`** verwendet, welche die **`Zeit in Millisekunden`** zählt.

In dem Zustand **`ProduktRücksendend`** befördert das **`Förderband`** das Produkt **`3 Sekunden`** lang nach links (**`bFBRückwärts=1`**), danach wird das **`Förderband`** wieder gestoppt (**`bFBRückwärts=0`**) und die Verpackungsstation wechselt wieder in den Zustand **`Wartend`**.

In dem Zustand **`ProduktWeiterleitend`** befördert das **`Förderband`** das Produkt dauerhaft nach rechts (**`bFBVorwärts=1`**). Wenn das Produkt an dem Ende des Förderbandes ankommt, gibt die Lichtschranke_2 einen Messwert (**`bLS2`**) von **`TRUE`** zurück. Daraufhin wird das **`Förderband`** wieder angehalten (**`bFBVorwärts=0`**) und die Verpackungsstation wechselt wieder in den Zustand **`Wartend`**.

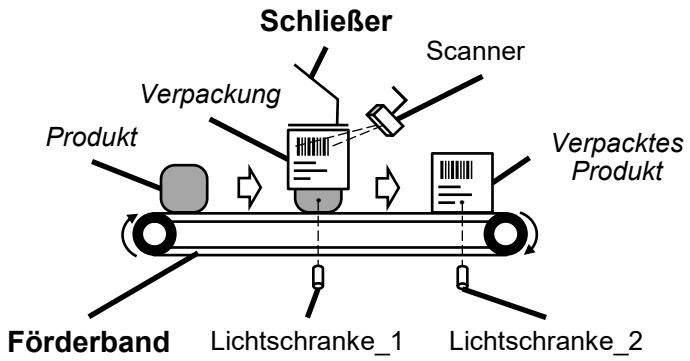


Bild 13.1: *Verpackungsstation im Detail*

Bild 13.2 zeigt ein Zustandsdiagramm, welches den zuvor beschriebenen Prozessablauf modelliert.

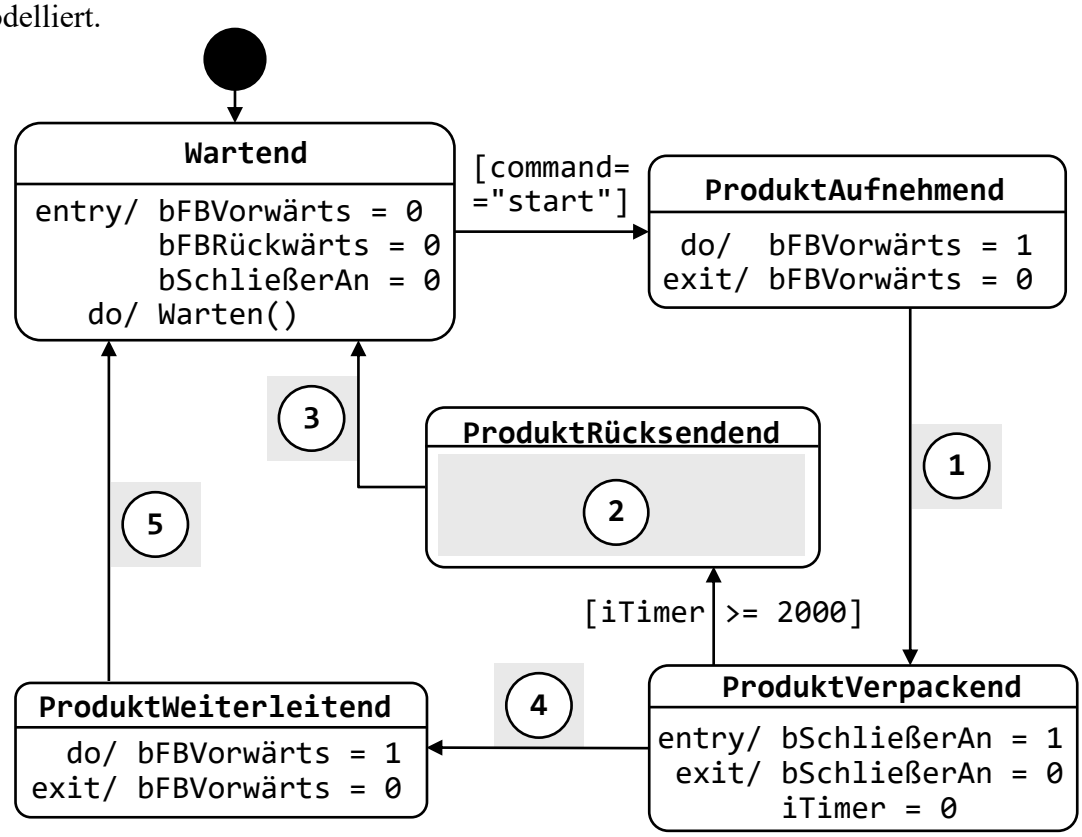


Bild 13.2: *Zustandsdiagramm des Verpackungsprozesses mit zu ergänzenden Lücken.*

Ergänzen Sie die durch die Ziffern ① bis ⑤ gekennzeichneten Lücken des Zustandsdiagramms in Bild 13.2, nach dem Beschreibungstext.

Geben Sie die korrekte Übergangsbedingung für Lücke ① an:

_____

Vervollständigen Sie Zustand **`Produktrücksendend`** in Lücke ②:

_____

_____

_____

Geben Sie die korrekte Übergangsbedingung für Lücke ③ an:

_____

Geben Sie die korrekte Übergangsbedingung für Lücke ④ an:

_____

Geben Sie die korrekte Übergangsbedingung für Lücke ⑤ an:

_____

Aufgabe 14: UML-Zustandsdiagramm zu C-Code

Implementieren Sie Teile des in Aufgabe 13 beschriebenen Zustandsdiagramms in der Programmiersprache C. Nutzen Sie hierfür die in Tabelle 14.1 vorgegebenen Variablen und Funktionen.

Typ	Name	Beschreibung
VARIABLEN	`int iState`	Variable für den aktuellen Zustand der Verpackungsstation mit folgender Zuordnung: iState = 0 : Wartend iState = 1 : ProduktAufnehmend **iState = 2 : ProduktVerpackend** *(im Folgenden betrachtet)* iState = 3 : ProduktRuecksendend iState = 4 : ProduktWeiterleitend
	`int iEtikettenID`	Variable, die die vom Scanner gescannten Adressinformationen angibt und als Bedingung für den Zustandswechsel verwendet wird.
	`int iKundenID`	Variable, die die Adressinformationen des Kunden im System angibt und als Bedingung für den Zustandswechsel verwendet wird.
	`unsigned int vplcZeit`	Zählvariable für die aktuelle Zeit der SPS in Millisekunden ab Programmstart.
	`unsigned int iTimer`	Zählvariable für Zeitmessung in Millisekunden.
FUNKTIONEN	`void SchliesserAn()`	Funktion zum Steuern des Schließers im Zustand Produktverpackend, um das Produkt zu verpacken.
	`void SchliesserAus()`	Funktion zum Steuern des Schließers im Zustand Produktverpackend, um den Schließer zurückzufahren.

Tabelle 14.1: *Vorgegebene Variablen und vorimplementierte Funktionen*

Vervollständigen Sie das folgende Programmgerüst in der Programmiersprache C gemäß den Kommentaren im Lösungsfeld und der Beschreibung des Verpackungsprozesses in Aufgabe 13. Verwenden Sie hierfür die in Tabelle 14.1 angegebenen Variablennamen und Funktionen, welche im Header namens **`Station.h`** bereits deklariert und implementiert sind.

Hinweis: Die Platzhalter /*ZUSTAENDE*/ enthalten spezifischen Code für Zustände des Zustandsautomaten und müssen nicht implementiert werden. Sie können für diese Aufgabe davon ausgehen, dass beim ersten Eintritt des Zustands **`ProduktVerpackend`** (**`iState=2`**), **`iTimer`** den Wert **`0`** hat.

```
#include <stdio.h>
// Benutzerdefinierte Bibliothek Station.h einbinden

_____________________________________________________________
iState = 0;           // Zustandsvariable iState
int main () {
// Zyklische Endlosausfuehrung

_____________________________________________________________
// Zustandsautomat

_____________________________________________________________
    {
     case 0: /* WARTEND */
     case 1: /* PRODUKTAUFNEHMEND */
     case 2:
_________________________// Schliesser anschalten
           // Abfragen ob die Adressen uebereinstimmen
           if_____________________________________________{
_____________________________________// Schliesser ausschalten
_____________________________________// iTimer zuruecksetzen
_____________________________________}// Wechsel in Zustand 4
           else{
           // iTimer auf aktuelle Zeit setzen, wenn er 0 ist

_____________________________________________________________
_____________________________________________________________
_____________________________________________________________
           // Abfragen ob 2 Sek. vergangen

_____________________________________________________________
_____________________________________// Schliesser ausschalten
_____________________________________// iTimer zuruecksetzen
_____________________________________// Wechsel in Zustand 3
_______________}

_____________________________________________________________
     case 3: /* PRODUKTRUECKSENDEND */
     case 4: /* PRODUKTWEITERLEITEND */
     }
 }
 return 0;
}
```

Aufgabe 15: Grundlagen in C

a) Deklarieren Sie in der Programmiersprache C eine Variable **prod** vom Typ float und initialisieren Sie sie mit dem Wert 0.1. Deklarieren Sie einen Pointer **ptr**, der auf die Variable **prod** verweist. Erhöhen Sie anschließend den Wert der Variablen **prod** um 1.2 mithilfe des Pointers **ptr**.

_____
_____
_____
_____
_____
_____

b) Vervollständigen Sie das folgende C-Programm, indem Sie die Lücken entsprechend den Kommentaren ausfüllen. Das Programm liest fünf Ganzzahlen vom Benutzer ein und speichert sie in einem Array. Dann werden nur die positiven Zahlen addiert. Am Ende wird die Summe der positiven Zahlen auf der Konsole ausgegeben. Eine Prüfung auf falsche Eingaben ist nicht erforderlich. Gehen Sie davon aus, dass der Benutzer mindestens eine positive Zahl eingibt.

```
#include <stdio.h>
int main()
{
    int zahlen[5];
    int summe = 0;
    int i = 0;
    printf("Geben Sie 5 Ganzzahlen ein:\n");
    // while-Schleife zur Eingabe der Zahlen
    while (_____) {
        scanf("%d", &zahlen[i]);
        // Wenn die Zahl groesser als null ist
        if (_____) {
            // Zahl zur Summe addieren
            _____
            _____
}
        // Zaehler erhoehen
        i++;
    }
    // Ausgabe der Summe als Ganzzahl auf der
    // Konsole (kein zusaetzlicher Text
    // Erforderlich)
    _____
_____
    return 0;
}
```

Aufgabe 16: Algorithmen

Den Produkten im Lagerhaus werden gemäß folgenden Regeln Produkt-IDs zugeteilt (siehe Bild 16.1).

Nummerierungsregeln der Produkt-IDs:

- Gültiger Bereich: $1 \leq posx \leq 3; 1 \leq posz \leq 15$
- Die Nummerierung beginnt bei 1 und endet bei 45.
- Die Nummerierung erfolgt aufsteigend (1, 2, 3, ... 45), von links nach rechts und von unten nach oben, wie in Bild 16.1 gezeigt.

Die Funktion `fkt` gibt die Produkt-ID (Typ int) basierend auf den übergebenen Koordinaten posx und posz (beide vom Typ int) zurück. Es wird davon ausgegangen, dass die Übergabeparameter der Funktion den richtigen Datentyp haben. Wenn posx oder posz außerhalb des gültigen Bereichs liegen (siehe oben), gibt die Funktion **-1** zurück.

Implementieren Sie die Funktion `fkt` in der Programmiersprache C. Vervollständigen Sie die Funktion gemäß den oben beschriebenen Regeln. Sie dürfen ausschließlich die Bibliothek `<stdlib.h>` verwenden.

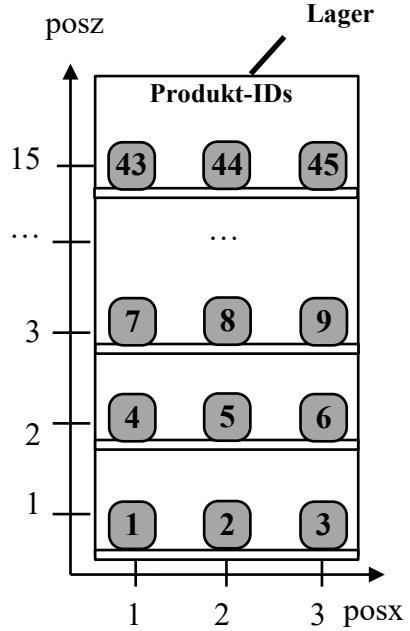


Bild 16.1 Produkt-IDs im Lager

```
#include <stdlib.h>
// Implementierung der Funktion fkt
```

Aufgabe 17: Datenstrukturen

Zur Sicherstellung von Menge und Gewicht der Produkte beim Transport zur Verpackungsstation soll eine einfache Datenbankfunktionalität in Form einer **doppelt verketteten Liste mit Listenkopf** implementiert werden.

Im Listenelement vom Typ **`PRODUKT`** sollen gespeichert werden:

- Das Gewicht (**`gewicht`**) eines Produkts als Gleitkommazahl vom Typ **`float.`**
- Das Material (**`material`**) als String mit maximal 20 Zeichen. Der End of String Character muss zusätzlich berücksichtigt werden.
- Ein Zeiger (**`pnext`**) auf das nachfolgende Listenelement.
- Ein Zeiger (**`pprev`**) auf das vorherige Listenelement.

Im Listenkopf vom Typ **`PRODUKTKOLLEKTION`** sollen gespeichert werden:

- Anzahl der Produkte (**`anzahl`**) als vorzeichenlose Ganzzahl vom Typ **`unsigned int`**.
- Gesamtgewicht der Produkte (**`gesamtgewicht`**) als Gleitkommazahl vom Typ **`float.`**
- Einen Zeiger (**`pfirst`**) auf das erste Listenelement vom Typ **`PRODUKT`**.

a) Implementieren Sie ein Listenelement vom Typ **`PRODUKT`** sowie den Listenkopf vom Typ **`PRODUKTKOLLEKTION`** gemäß der obigen Beschreibung in der Programmiersprache C.

```
_____{
_____
_____
_____
_____
_____
_____
} PRODUKT;
```

```
_____{
_____
_____
_____
_____
_____
_____
} PRODUKTKOLLEKTION;
```

b) Vervollständigen Sie die **Funktion produktHinzufuegen**, die ein neues Produkt (Listenelement vom Typ **PRODUKT**) hinter der **i**-ten Position der doppelt verketteten Liste einfügt. Dabei sollen im Listenkopf vom Typ **PRODUKTKOLLEKTION** die Anzahl der Produkte (**anzahl**) aktualisiert werden.

Die Funktion erhält als Übergabeparameter einen Zeiger (**data**) auf den Listenkopf vom Typ **PRODUKTKOLLEKTION** sowie die Positionsnummer **i** vom Typ **int**. Gehen Sie davon aus, dass die doppelt verkettete Liste nicht leer ist. Der Zeiger (**pnext**) des letzten Listenelements zeigt auf **NULL,** und der Zeiger (**pprev**) des Listenkopfes zeigt auf **NULL**.

```
// Neues Produkt hinter der i-ten Position der Liste
// einfuegen
void produktHinzufuegen(_____________________________){

//  Zuweisung des ersten Listenelements an die
//  Variable temp
_____________________________________________________
_____________________________________________________
// Iteration, bis das i-te Element gefunden wird
for (_____________________________________________){
// Verkettung zum naechsten Element
_____________________________________________________
_____________________________________________________
_____________________________________________________
_____________________________________________________
_____________________________________________________

}

// Dynamische Speicherzuweisung für das neue PRODUKT
// an die Variable neu
_____________________________________________________
_____________________________________________________
_____________________________________________________
_____________________________________________________
_____________________________________________________
// Neues Produkt einfuegen
// Wenn Einfuegen an Listenkopf
if(i==0){
_____________________________________________________
_____________________________________________________
_____________________________________________________
_____________________________________________________
_____________________________________________________

}
```

```
// Wenn das Produkt an eine andere Position
// eingefuegt wird
else {
    neu->pnext = temp->pnext;
    neu->pprev = temp;
    // Wenn es ein Element nach temp gibt, wird
    // dessen pprev auf das neue Produkt gesetzt,
    // damit die doppelte Verkettung korrekt bleibt
    if (temp->pnext != NULL) {
_____________________________________________________
_____________________________________________________
_____________________________________________________
_____________________________________________________
    }
    temp->pnext = neu;
}


// Aktualisieren des Listenkopfs
// Anzahl der Produkte im Listenkopf um 1 erhoehen
_____________________________________________________
_____________________________________________________
_____________________________________________________
_____________________________________________________
}
```