Klausur

Grundlagen der Datenverarbeitung/ Technische Informatik

21.09.04

Machen Sie bitte zu Beginn der Klausur hier unten die notwendigen Angaben. Lösen Sie nicht die Heftung der Klausur.

Wenn Sie die Heftung lösen, müssen Sie jedes Blatt oben mit Namen und Matrikelnummer versehen, und wir übernehmen keine Verantwortung für evt. verloren gegangene Blätter.
Reicht Ihnen der Platz nicht, benutzen Sie die leeren Rückseiten.
Bemühen Sie sich bitte um Eindeutigkeit und deutliche Lesbarkeit Ihrer Antworten.

Das letzte Blatt enthält den Befehlssatz. Es ist sinnvoll, dass Sie den Befehlssatz ohne Blättern vor Augen haben, wenn Sie die Programmieraufgabe lösen. Sobald Sie an der entsprechenden Stelle sind, sollten Sie das letzte Blatt vorsichtig abreißen, damit Sie es geeignet verwenden können.

Beachten Sie, dass es eine gute Prüfungsstrategie ist, die Punktzahl, die rechts neben der Bezeichnung der Gesamtaufgabe angegeben wird, als Minutenäquivalent zu behandeln. Versuchen Sie, die Bearbeitungszeit der Aufgabe ungefähr auf dieses Minutenäquivalent zu beschränken. Damit können Sie ein „Verbeißen“ in eine Aufgabe vermeiden, was in aller Regel zum Misserfolg führt.

1. Zahlensysteme	34
2. Schaltalgebra	29
3. Schaltnetze	27
4. Codierung	30
5. Schaltwerksynthese	42
6. Schaltwerkanalyse	30
7. Programmierung	40
	232

Name	
Vorname	
Matrikelnr.:	108 0
Unterschrift	

1. Zahlensysteme

1.1 Welche der folgenden Dezimalzahlen sind in den angegebenen Formaten ohne Genauigkeitsverlust darstellbar? Schreiben Sie in das Feld <nein>, wenn die Zahl nicht darstellbar ist, bzw. das Bitmuster der Zahl im Format, wenn sie darstellbar ist.

Dezimal-zahlen	x.xxx xxxx 2-Komplement	x.xxx xxxx Betrag mit Vorzeichen (1 = neg.)	xxxx.xxxx 2-Komplement	
- 1.5				1
$-2^{0} + 2^{-4}$				2
$-8 + 2^{-4}$				3
$-8 - 2^{-4}$				4
- 7.75				5

Sorgfältige Zwischenrechnungen, da nur das Ergebnis zählt, das dann oben angegeben wird:

1.2 Welche der folgenden Dezimalzahlen sind in den angegebenen Formaten ohne Genauigkeitsverlust darstellbar? Schreiben Sie in das Feld ein <nein>, wenn die Zahl nicht darstellbar ist, bzw. das Bitmuster der Zahl im Format, wenn sie darstellbar ist.

Dezimalzahlen	xxxx. x.xxx xxxx 2 Mantisse und Exponent im 2-Komplement, normalisiert	xxxx. x.xxx xxxx 2 Mantisse und Exponent als Betrag mit Vorzeichen, normalisiert
-2^{-8}		
-2^{8}		
7.1875		
10^{2}		
$2^{-9} + 2^{-15}$		

Sorgfältige Zwischenrechnungen, da nur das Ergebnis zählt, das dann oben angegeben wird:

1.3 Gegeben ist das Format x.xxx xxxx für Zahlen im 2-Komplement.

Welches vollständige Bitmuster ergibt sich, wenn man die beiden größten positiven Zahlen multipliziert. (Ergebnis mit Rechnung)

Berechnen Sie, wie der Bruch 10^{-1} in diesem Format dargestellt wird.

Bestimmen Sie den Dezimalwert der Differenz zwischen dem richtigen Wert und dem abgerundeten Wert, also den absoluten Fehler.

1.4 Gegeben ist folgendes Format: x.xxx xxxx $8^{xxxx.}$. Die Mantisse und der Exponent gelten als Oktal-Zahlen und Betrag mit Vorzeichen.

Stellen Sie die Dezimalzahl 128 normalisiert in diesem Format dar.

Zu dem Ergebnis soll die Zahl 0.000 0002 $8^{0007.}$ addiert werden. Berechnen Sie das normalisierte Ergebnis in diesem Format.

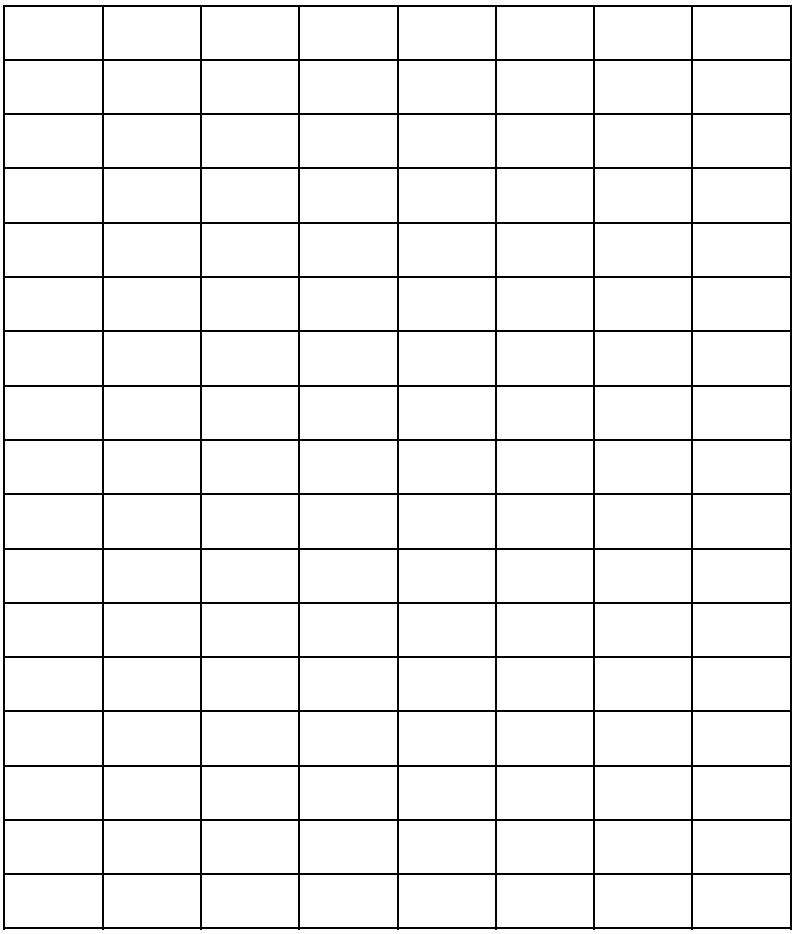
2. Schaltalgebra

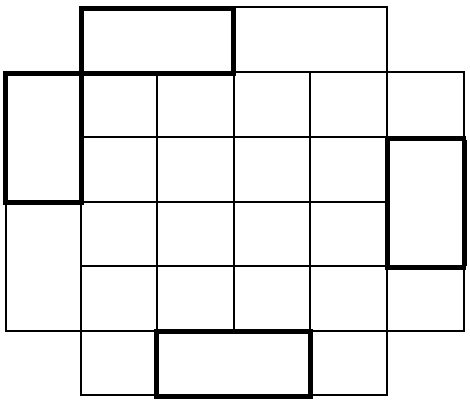
2.1 Minimieren Sie schaltalgebraisch folgende Funktion (Ergebnis ohne negierte Variable):

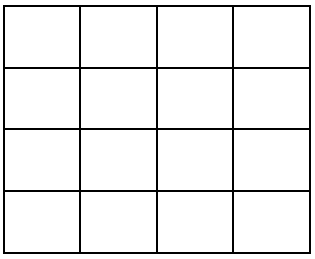
$$y = \overline{\overline{b}\,a\,\overline{a\,c}\,\overline{a\,b}\,\overline{b\,c}}$$

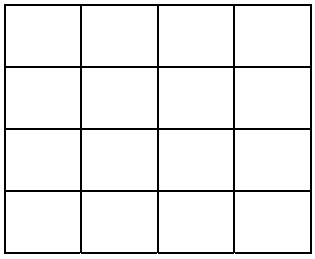
2.2 Es soll ein Schaltnetz entworfen werden, das zwei zweistellige ganze positive Dualzahlen (x_{11}, x_{10}) und (x_{21}, x_{20}) miteinander vergleicht. Das Schaltnetz soll drei Schaltvariable als Ergebnis liefern gemäß den Fällen: $x_1 > x_2$, $x_1 < x_2$, $x_1 = x_2$. Die Schaltvariablen sollen 1 sein, wenn die jeweilige Bedingung erfüllt ist.

Bestimmen Sie die Schalttabelle für jede der drei Schaltvariablen. Erst die Bedeutung der Spalten in die obere Zeile eintragen. Bestimmen Sie dann die **minimalen disjunktiven Normalformen** aus den KV-Diagrammen.









Geben Sie die minimalen DNFs der Schaltfunktionen an.

2.3 Formen Sie die folgende Funktion so um, dass sie nur noch Antivalenz-Verknüpfungen enthält (Negation zugelassen).

$$y = a \not\equiv b \not\equiv c \equiv d$$

2.4 Wie lautet die **minimale konjunktive** Normalform der Funktion?

$$y = a\,b\,c + \overline{a}\,\overline{b}\,\overline{c}$$

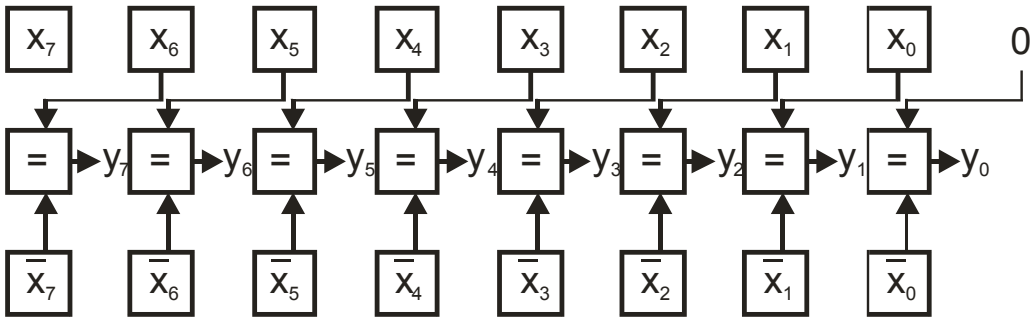
3. Schaltnetze

3.1 Ein Schaltnetz soll aus gegebenen 8-Bit-Wörtern gemäß folgender Zuordnung neue 8-Bit-Wörter erzeugen (b für beliebig):

d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	→	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
b	b	b	b	b	b	b	1		1	1	1	1	1	1	1	1
b	b	b	b	b	b	1	0		1	1	1	1	1	1	1	0
b	b	b	b	b	1	0	0		1	1	1	1	1	1	0	0
b	b	b	b	1	0	0	0		1	1	1	1	1	0	0	0
b	b	b	1	0	0	0	0		1	1	1	1	0	0	0	0
b	b	1	0	0	0	0	0		1	1	1	0	0	0	0	0
b	1	0	0	0	0	0	0		1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0		1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0

Geben Sie den Schaltplan des Schaltnetzes an, das diese Umwandlung ausführt. (Übliche Symbole für die logischen Standard-Gatter)

3.2 Gegeben ist folgendes Schaltnetz, das die Bits von 8 Bit Wörtern (x_7, .., x_0) und ihrer Inversen wie folgt verarbeitet (= bedeutet Äquivalenz).



Angenommen, dass es nur folgende Wörter für x gibt:

x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	0
1	1	1	1	1	1	0	0
1	1	1	1	1	0	0	0
1	1	1	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

1. Welche Bitmuster ergeben sich dann für die y_i?

y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0

2. Welche Bitmuster ergeben sich, wenn man $z_i = y_i \not\equiv x_i$ bildet?

z_7	z_6	z_5	z_4	z_3	z_2	z_1	z_0

3.3 Die in 3.2 gewonnenen Bitmuster (z_7, …, z_0) sind nützlich für die Bildung der 2-Komplemente der 8Bit-Wörter in 3.1.

Geben Sie das einstufige Schaltnetz an, das aus Datenwörtern (d_7, …, d_0) und den Wörtern (z_7, …, z_0) die Bitmuster der 2-Komplemente von (d_7, …, d_0) erzeugt.

4. Codierung

4.1 Bezeichnen Sie an der folgenden Kontrollmatrix eines gültigen Hamming-Codes die Prüf- und die Datenbits.

0	0	0	1	1	1	1	0	0	0	0
1	0	1	0	1	0	1	0	1	0	1
0	1	1	0	0	1	1	0	0	1	1
0	0	0	0	0	0	0	1	1	1	1

Geben Sie (entsprechend zu den von Ihnen gewählten Bezeichnungen) die Schaltfunktionen zur Bestimmung der Prüfbits aus den Datenbits an.

Ein Empfänger empfängt folgendes Wort:

0	1	1	1	0	1	0	1	1	1	1

Welche Werte erzeugen die Schaltfunktionen zur Bestimmung der Fehlersyndrom-Bits?

Wenn es eine betroffene Stelle gibt, markieren Sie diese und geben Sie an, was im Empfänger zu tun wäre. (Achten Sie genau darauf, welche Zeilen durch die Fehlersyndrom-Bits betroffen sind, sonst ordnen Sie die Position falsch zu.)

4.2 Beim Vergleich der Vor- und der Nachteile der Sicherung durch Blockübertragung mit Zeilen- und Spaltenparität und der Sicherung durch Hamming-Code wird folgende Aussage gemacht:

Unstrittig ist, dass bei einer gegebenen Datenwortlänge die Sicherung mit Hamming-Code weniger Prüfbits benötigt als die Blocksicherung.
Aber die zusätzlichen Prüfbits bei der Blocksicherung haben einen Vorteil: man kann die Zweifach-Fehler eindeutig erkennen und man kann damit in diesen Fällen eine falsche Korrektur vermeiden, die beim Hamming-Code eintreten würde.

Beweisen oder widerlegen Sie die Aussage über die Zweifachfehler bei der Blocksicherung dadurch, dass Sie hier alle möglichen Typen von Zweifach-Fehlern bei der Blocksicherung auf ihre sichere Erkennbarkeit prüfen.

Tatsächlich korrigiert man im Falle von Zweifachfehlern beim Hamming-Code falsch. Bestimmen Sie das Codewort (gemäß Hamming-Code in 4.1) zu folgendem Datenwort:

		1		0	1	0		1	1	1

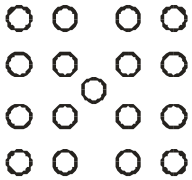
Dieses Codewort wurde ursprünglich in 4.1 gesendet. Markieren Sie die Stellen, die unterschiedlich sind zum empfangenen Codewort in 4.1. Bewerten Sie aus dieser Sicht die Korrektur in 4.1.

Sie sollen einen Vorschlag machen, wie man Hamming-codierte Wörter zusätzlich codieren kann, so dass man mit Sicherheit erkennt, ob ein Einfachfehler vorliegt oder nicht. Suchen Sie nach einer Lösung mit möglichst wenig zusätzlichen Prüfbits.

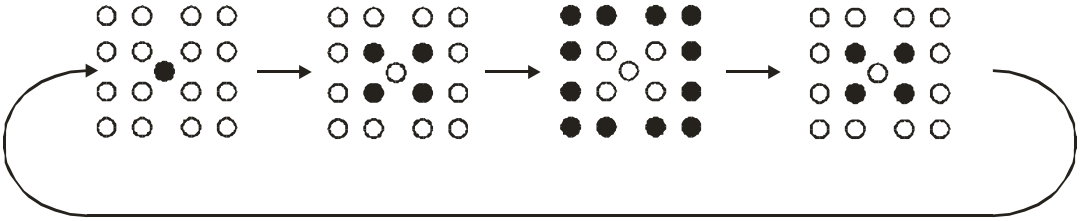
Unter welcher Bedingung darf man bei Ihrem Verfahren nur korrigieren, damit man die Korrektur bei Doppelfehlern mit Sicherheit ausschließen kann?

5. Schaltwerksynthese

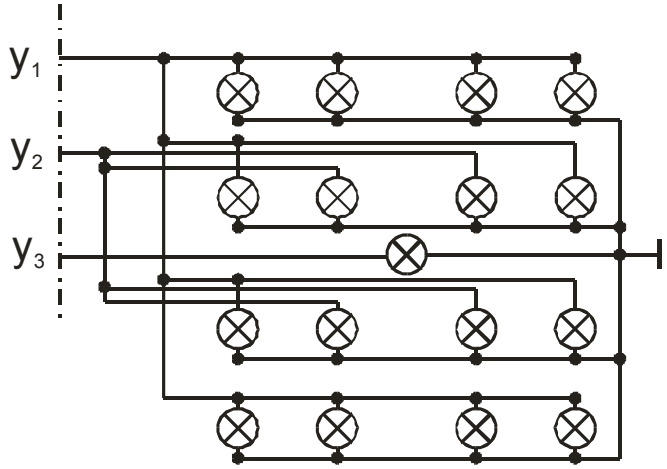
5.1 Gegeben ist folgende Matrix von Glühlampen:



Diese Glühlampen sollen in einer zyklischen Folge von Mustern leuchten:



Die Schnittstelle zur Ansteuerung der Lampen besteht aus drei binären Variablen:
$y_i = 1$ bedeutet, dass der Strompfad durch die Lampen der Gruppe durchgängig ist, d.h. dass die Lampen leuchten.



Es ist die Aufgabe, ein **synchrones** Schaltwerk mit **JK**-Flip-Flops zu entwickeln, das die angegebene Musterfolge erzeugt.

5.1.1 Bestimmen Sie die **Schaltfunktionen des Übergangsschaltnetzes in minimaler disjunktiver Normalform**.

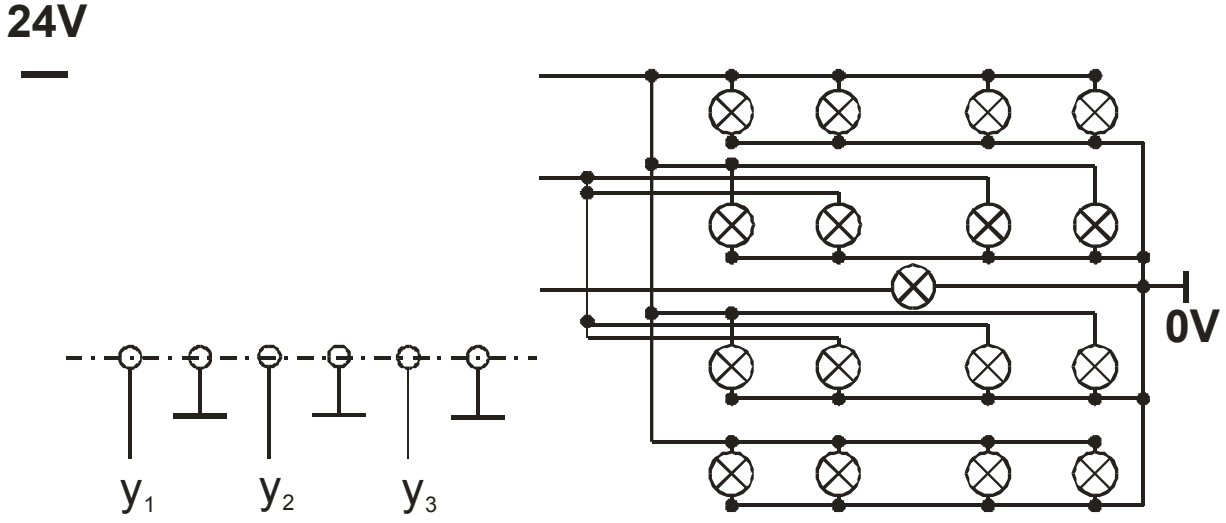
Ihre Entwicklungsmethode muss deutlich erkennbar sein. Dazu gehört:

- die Definition des Übergangsgraphen,
- die Angabe der notwendigen Zahl der Zustandsspeicher,
- die eindeutige Übergangstabelle,
- die deutlich erkennbare Minimierung.

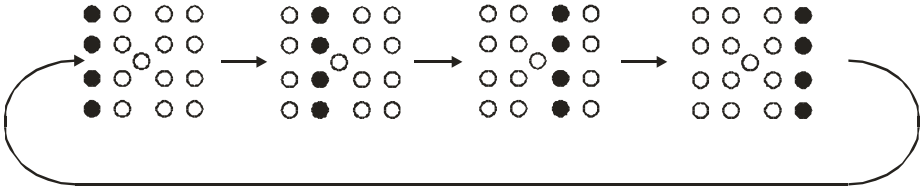
Wegen der geringen Komplexität wird die Punktzahl nur für ein richtiges Gesamtergebnis gegeben. Achten Sie deshalb auf Sorgfalt.

5.1.2 Geben Sie die Schaltfunktionen des Ausgabeschaltnetzes an.

5.1.3 Ergänzen Sie in der folgenden Skizze mit den bekannten Symbolen die Strompfade und die Schaltelemente, welche mit Hilfe der Schaltvariablen y_i die Strompfade durch die Lampen schließen.
(Ein Hinweis, der nur den praktischen Hintergrund betrifft: Die y_i werden in der Regel mit Gattern erzeugt, die nicht mit 24 V, sondern z.B. bei TTL-Gattern mit 5 V versorgt werden, was der wichtigste Grund für die Trennung der Steuer- und Licht-Stromkreise ist.)



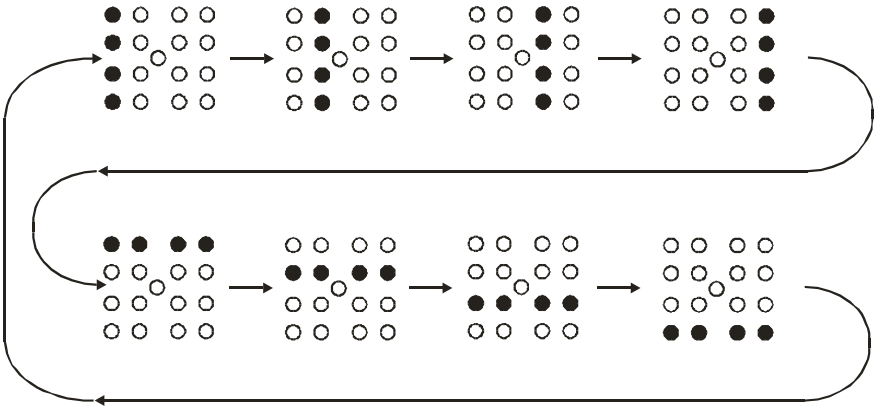
5.1.4 Es soll folgende Muster-Folge erzeugt werden. Welcher Teil vom vorausgehenden Entwurf könnte unverändert übernommen werden, welche Teile müssten geändert werden? Warum?



……unverändert, weil

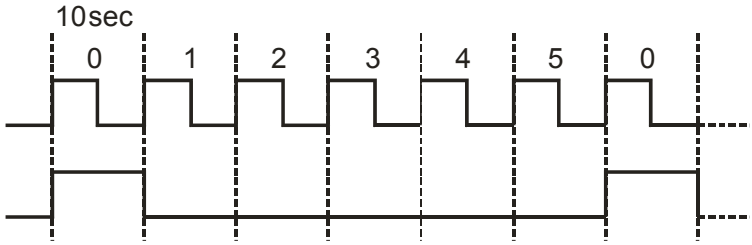
…….verändert, weil

5.1.5 Wie viele Zustandsvariable hat das synchrone Schaltwerk mindestens, das folgende Muster-Folge erzeugen soll? Antwort mit einem begründenden Satz.



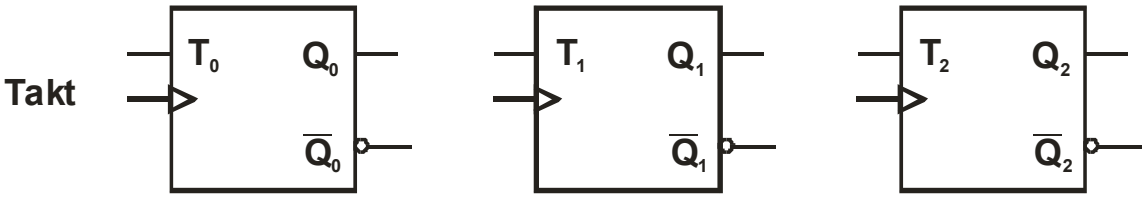
Wie viele Zustandsvariable hat das synchrone Schaltwerk mindestens, das auf Grund einer binären Steuervariablen bewirkt, dass wahlweise die Muster in der gegebenen oder in der umgekehrten Reihenfolge ablaufen? Antwort mit einem begründenden Satz.

5.2 In einer Uhr wird ein Schaltwerk zum Zählen der 10-Sekunden-Takte gebraucht. Das Schaltwerk wird mit einem Takt mit der Periodendauer von 10 sec getaktet und soll alle 60 sec einen Puls von 10 sec abgeben. Das Zeit-Diagramm für die Taktung und die Ausgabe des Schaltwerks ist also wie folgt:



Es ist die Aufgabe, ein **asynchrones** Schaltwerk mit **T**-Flip-Flops zu entwickeln, das den 60 sec-Taktpuls erzeugt.

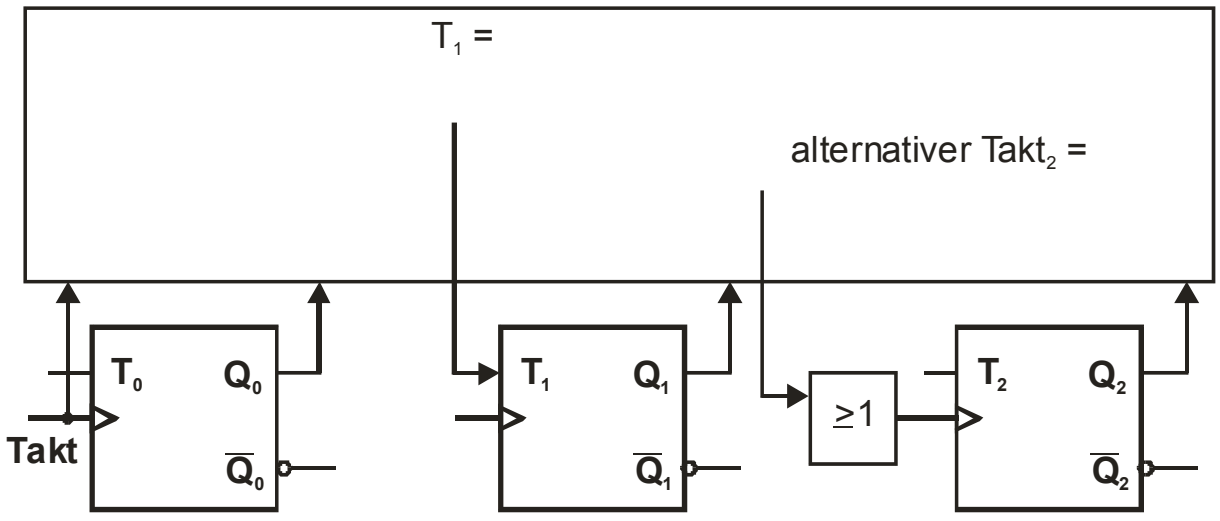
5.2.1 Vervollständigen Sie die Schaltung in der folgenden Skizze, so dass sich ein **Aufwärtszähler** ergibt.



5.2.2 Nun soll ein Schaltnetz entworfen werden, das diesen Zähler im maßgeblichen Zustand zurücksetzt.

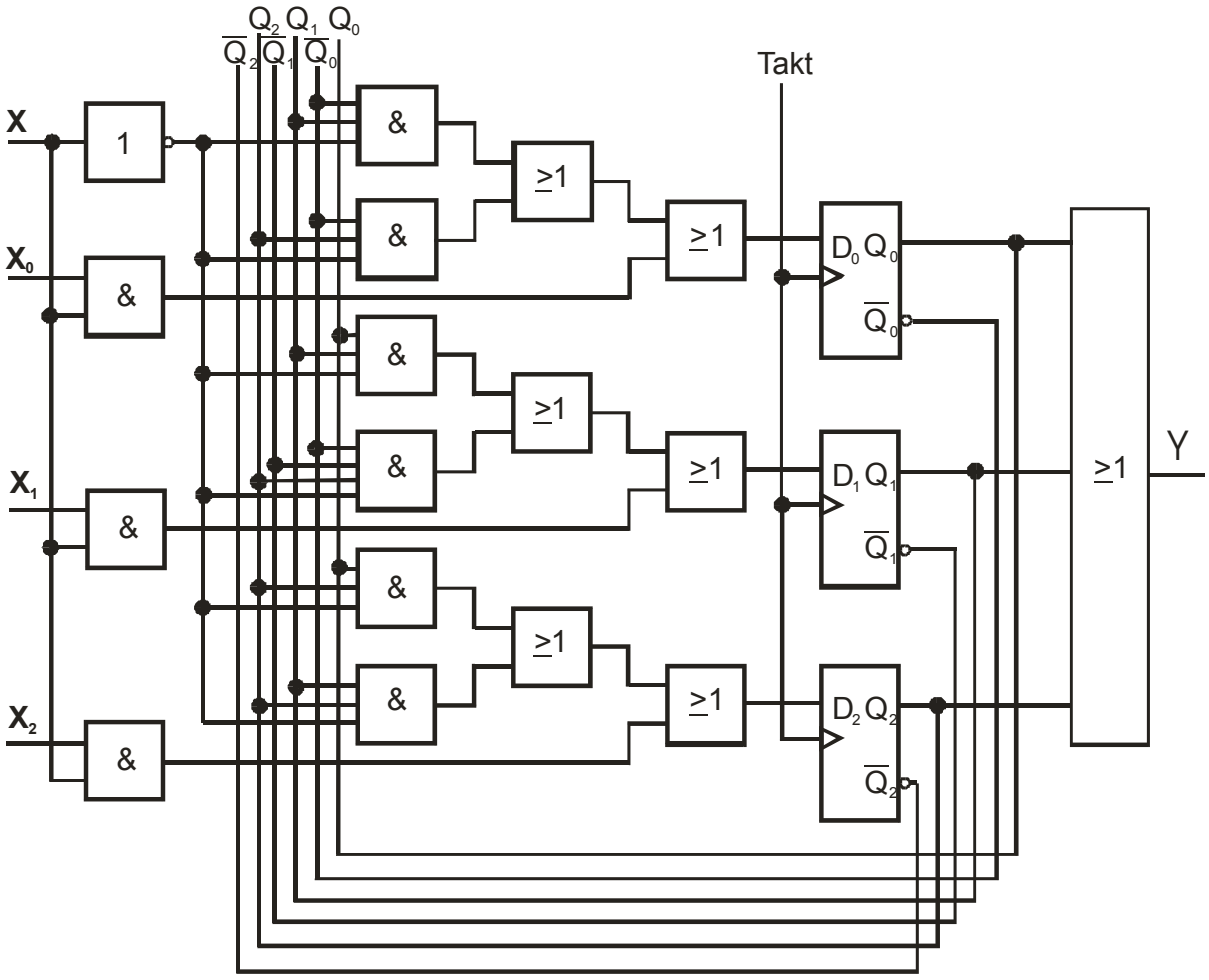
Geben Sie die gesuchten Schaltfunktionen an und ergänzen Sie gemäß Ihrer Antwort unter 4.2.1 den Schaltplan, so dass er eine funktionierende Schaltung darstellt..

Die nächste Seite ist absichtlich leer, damit Sie Ihren Lösungsweg entwickeln können. Der Prüfer braucht Anhaltspunkte, dass Sie Ihren Lösungsweg selbständig entwickelt haben.



6. Schaltwerksanalyse

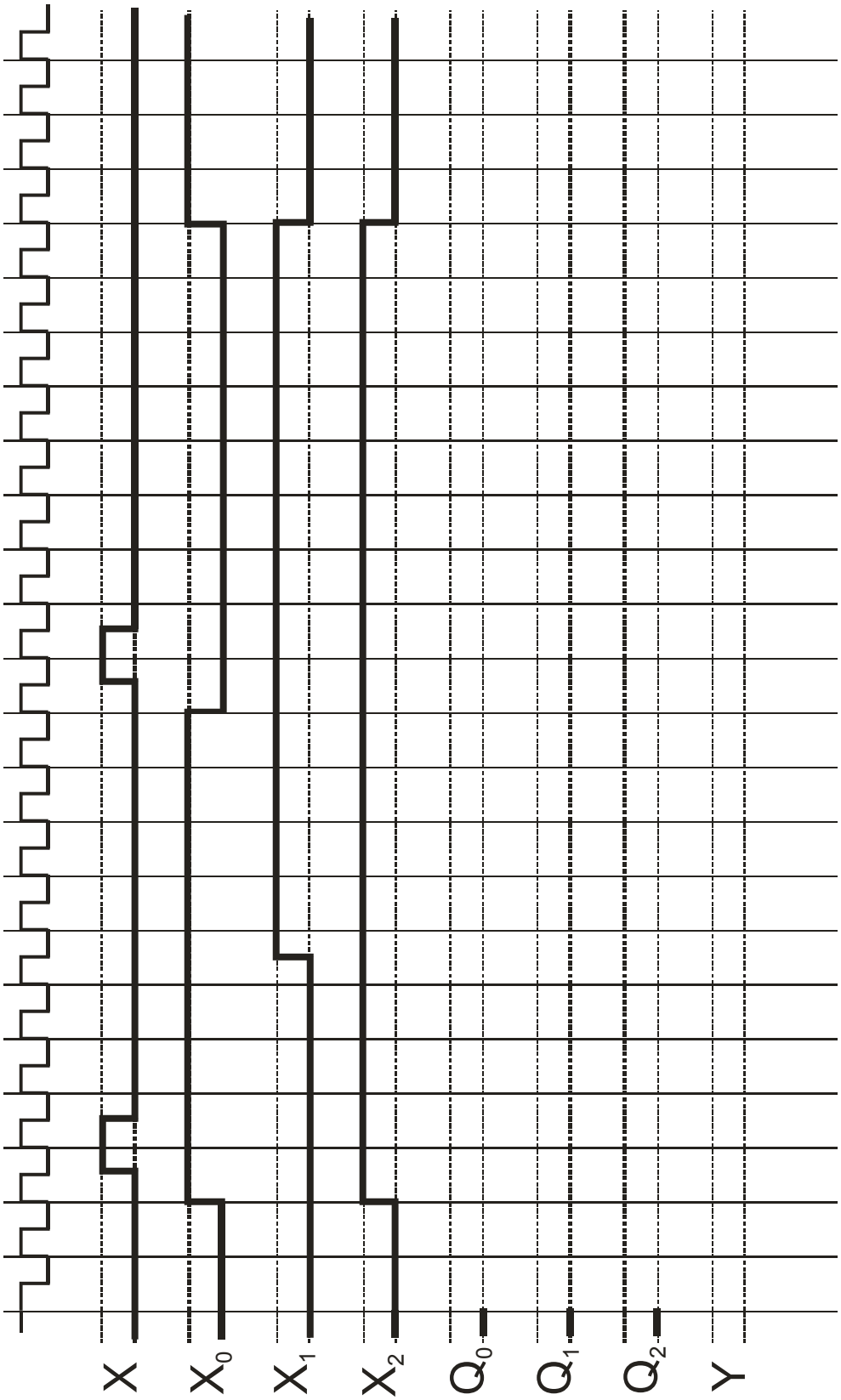
Gegeben ist der folgende Schaltplan eines Schaltwerks.
Vergewissern Sie sich des Typs des Schaltwerks und seiner grundsätzlichen Struktur gemäß dem Automatenmodell.



6.1 Bestimmen Sie die Schaltfunktionen des Übergangsschaltnetzes.

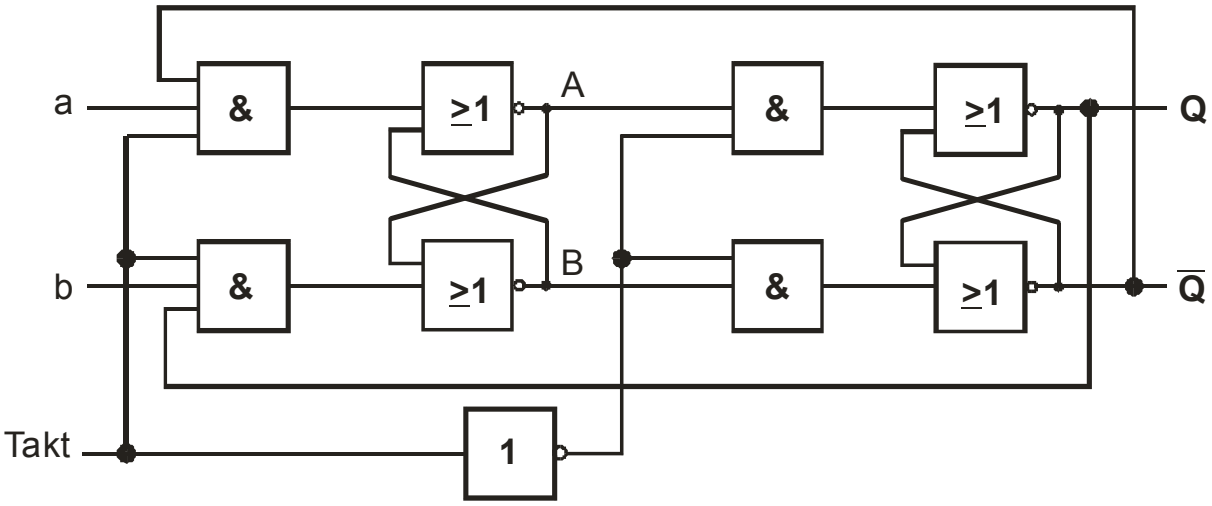
6.2 Es sei **X = 0**. Geben Sie für diesen Fall die Übergangstabelle an.
Bezeichnen Sie zuerst die Bedeutung der Spalten.

6.3 Im Folgenden wird ein Signal-Zeit-Diagramm vorgegeben. Ergänzen Sie den fehlenden Ablauf der angegebenen Signale.

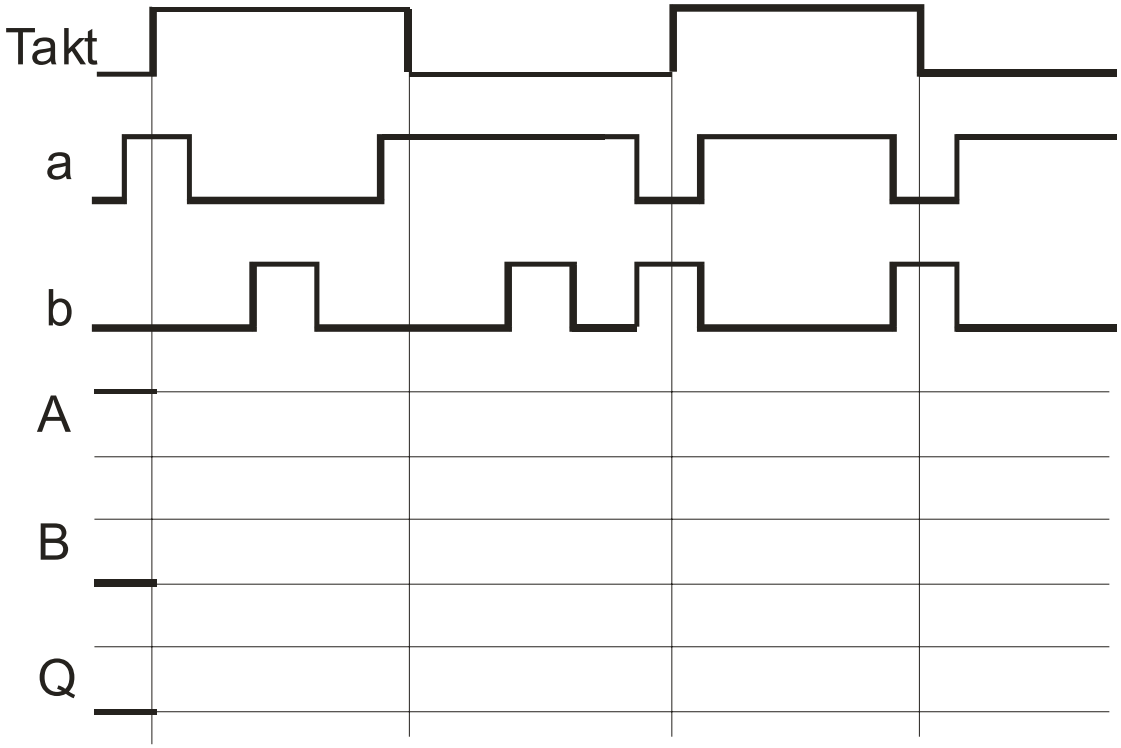


Welche Funktion hat die Schaltung?

6.4 Gegeben ist folgende Schaltung:



Ergänzen Sie den fehlenden zeitlichen Ablauf der Signale in folgendem Signal-Zeit-Diagramm.



Um welchen Flip-Flop-Typ handelt es sich?

Bezeichnen Sie entsprechend die Steuereingänge a und b.

7. Maschinennahes Programmieren

Bei jeder Anmeldung eines Prüfungskandidaten zu einer Prüfung wird ein Datensatz gemäß folgendem Strukturmuster erzeugt. Dieser wird hinter dem Datensatz gespeichert, der bei der vorausgehenden Anmeldung gespeichert wurde.

matnr1_4	: 4 Zeichen, Variablentyp: double word = 32Bit
matnr5_8	: 4 Zeichen, Variablentyp: double word
familienname	: 16 Zeichen, Feld von 4 double words
vorname	: 16 Zeichen, Feld von 4 double words
pruefdatum	: 8 Zeichen, Feld von 2 double words
fachschluessel	: 4 Zeichen, Variablentyp: double word
note	: 4 Zeichen, Variablentyp: double word

Ein Datenträger enthalte nach einer Anmeldeperiode die Datensätze aller Anmeldungen einer Prüfungsperiode. Die Reihenfolge der Speicherung entspricht der zeitlichen Reihenfolge der Anmeldungen.

Ein Assemblerprogramm liest die Datensätze in ein Feld mit dem Namen *anmeldungen* ein und stellt die Anzahl der Datensätze als Wert der Variablen mit dem Namen *anzahl* zur Verfügung.

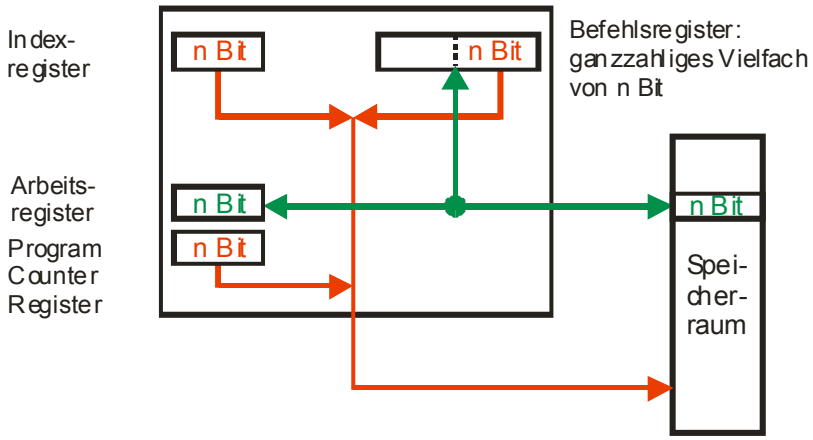
Das sind die Voraussetzungen für folgende Aufgabe.

Aufgabe:

Es ist ein Programm zu entwickeln, das die Datensätze nach einem vorgegebenen Wert in der Variablen *fachschluessel* untersucht und alle Datensätze mit dem gegebenen Fachschlüsselwert in ein Feld *spezielle_anmeldungen* kopiert, wobei eine kleine Anpassung erfolgt.

Die Anpassung besteht darin, dass der Fachschlüssel nicht kopiert wird, d.h. dass das Strukturmuster für die Datensätze im Feld *spezielle_anmeldungen* keine Variable für den Fachschlüssel enthält.

Sie haben in der Vorlesung die Struktur eines Minimal-Prozessors mit seinem Befehlssatz kennen gelernt.



Um das Programmierproblem überschaubarer zu machen, gilt folgendes:

Annahme: n = 32: Das bedeutet u.a., dass nur Speicherwörter der Länge 32 Bit adressiert werden. Damit ist die Adresse einer Variablen hier immer gleich der Adresse des Speicherwortes, in dem sie realisiert ist.

Im Folgenden sind *kursiv* geschriebene Namen als Variablen- bzw. Feldnamen zu behandeln.
Namen in großen BUCHSTABEN sind (ganzzahligen) Konstanten gleich zu setzen.

7.1 Vorbereitung der Konstanten und Variablen, Zeigerarithmetik

7.1.1 Geben Sie der Konstanten SATZLAENGE (entspricht Strukturlänge) den zutreffenden Wert als Vielfaches von Speicherwörtern.
Ordnen Sie dem Namen SATZLAENGE den zutreffenden numerischen Wert mit dem Deklarationsbefehl zu, den Sie in der Vorlesung kennen gelernt haben.

SATZLAENGE ……...

7.1.2 Der Inhalt des Indexregisters wird mit I bezeichnet und ist ein Zeiger.
[I] bzw. [Zeiger] bezeichnen den Inhalt des Speicherwortes, das durch den Zeiger (indirekt) adressiert wird.

<Adresse von> soll ein Operator sein, der den Assembler zur Berechnung der Adresse der Variablen auffordert, die danach angegeben ist.

Was tut folgender Befehl?

MOV I, <**Adresse von**> ***anmeldungen***

7.1.3 Der Inhalt des Indexregisters adressiere den Beginn eines Datensatzes im Feld *anmeldungen*. Sie sollen den Inhalt des Indexregisters durch Befehl so inkrementieren, dass der Beginn des nächsten Datensatzes adressiert wird.
Verwenden Sie im Befehl den Namen der entsprechenden Konstanten.

ADD I, …….…….

7.1.4 Der Inhalt des Indexregisters adressiere den Beginn eines Datensatzes. Sie sollen den Inhalt des Indexregisters durch Befehl inkrementieren, und zwar so, dass das Speicherwort adressiert wird, in dem die Variable *fachschluessel/anmeldungen* im ausgewählten Datensatz steht.
Ordnen Sie dem Namen OFFSET_FACHSCHLUESSEL den zutreffenden numerischen Wert mit dem Deklarationsbefehl zu, den Sie in der Vorlesung kennen gelernt haben und verwenden Sie im Befehl den Namen der entsprechenden Konstanten.

OFFSET_FACHSCHLUESSEL ……..

ADD I, …….…….

Definition der Variablen.

Annahmen:
1. Das Feld *spezielle_anmeldungen* ist durch die folgende Anweisung definiert.

spezielle_anmeldungen **DD** **2600 dup (?)**

Die maximale Anzahl der Datensätze pro Prüfungsfach wird mit 200 angegeben.

Das ergibt 200 · 13 = 2600 Wörter, die reserviert werden müssen.

Ein praxisnahes Beispiel für die Reservierung von 2600 32-Bit-Wörtern mit dem Feldnamen *spezielle_anmeldungen* ist folgende Anweisung gemäß der Notation des IA-32-Assemblers von Intel:

Die Anweisung „dupliziert“ so viele Variable vom Typ „double word“, wie vor der dup-Anweisung angegeben sind, und ordnet das reservierte Feld der Variablen dem Namen zu, mit dem die Anweisung beginnt.
(?) bedeutet, dass der Inhalt gleichgültig ist, d.h. dass die Variablen

DD steht für define double word = 32 Bit-Variable

Die Variable *fachschluessel* ist durch die folgende Anweisung definiert:

fachschluessel **DD** **FACHSCHLUESSEL**

FACHSCHLUESSEL ist ein vorgegebener konstanter Wert, mit dem die Variable *fachschluessel* beim Übersetzen vorbelegt (initialisiert) wird. Dieser wird, je nach Anwendung, später mit einem neuen Wert belegt.

Die übrigen Variablen ergeben sich bei der Entwicklung des Algorithmus. Ergänzen Sie die notwendigen Definitionen hierunter, wenn Sie mit der Entwicklung des Struktogramms fertig sind.

Bereitgestelltes und verwendbares Unterprogramm:

Das **Unterprogramm endlist** hat als Eingangsvariable die Variable ***anzahl.*** Es **multipliziert die Variable *anzahl* mit der Konstanten SATZLÄNGE** und liefert das **Ergebnis in** der Variablen ***listenendzeiger*.**

Können Sie den Dienst dieses Unterprogramms brauchen, rufen Sie es an geeigneter Stelle auf. Der Aufruf ist: **call endlist**.

7.2 Entwicklung des Assembler-Programms

Es wird Ihnen nicht, wie in den vorausgehenden Klausuren, ein Basis-Struktogramm vorgegeben.

Entwickeln Sie selbst das **Struktogramm** für Ihre Lösung.

Die Funktionsbeschreibung in den Blöcken soll auf den Zuständen der Variablen beruhen, die an der Funktion jedes Blockes beteiligt sind.

Wenn Sie mit der Entwicklung des Struktogramms fertig sind, ergänzen Sie bitte die fehlenden Definitionen der Variablen im Aufgabenteil für die Definitionen.

Vermeiden Sie im Struktogramm Beschreibungen der Funktion von Assembler-befehlen. Das Struktogramm ist eine Beschreibung des Algorithmus, und nicht ein Assemblerprogramm in verbaler Form.

Man kann ein Struktogramm grob anlegen, so dass die Funktionsbeschreibungen sehr pauschal sind. Man die Blöcke auch systematisch weiter detaillieren. Dann wird die Funktionsbeschreibung genauer und die Umsetzung in das Assemblerprogramm durchschaubarer.

Achtung!

Bewertet wird nur ein endgültiges, sauber geschriebenes und in seiner Blockstruktur eindeutiges Struktogramm.

Ohne ein solches Struktogramm keine Bewertung irgendwelcher Assembler-Befehle!

Ein stimmiges grobes Struktogramm bringt eine Basispunktzahl. Je mehr systematische Detaillierung hinzugefügt wird, umso mehr Punkte gibt es für das Struktogramm.

Geben Sie den Blöcken eindeutige Nummern. Diese Nummern müssen Sie neben den ersten und den letzten Assemblerbefehl des Assemblerprogrammabschnittes schreiben, das den jeweiligen Block realisiert.

Diese Aufgabe ist nur auf den ersten Blick nicht zeitintensiv. Achten Sie unbedingt auf das Minutenäquivalent.

Setzen Sie die Blöcke in Assembler-Befehle um.

Falls Ihr Code chaotisch wird, nutzen Sie lieber die Möglichkeit, durchzustreichen und in den nachfolgenden Zeilen neu zu beginnen. Die Vorlage ist genau deshalb so lang.

Marken ↓		Blockmerkmal ↓

Befehlssatz

	Mnemotechnische Notation	
Lade reg mit Konstante	mov reg, KONSTANTE	reg = A oder reg = I
Lade A mit I	mov A, I	
Lade I mit A	mov I, A	
Lade reg mit Variable	mov reg, Variablenname	reg = A oder reg = I
Lade Variable mit reg	mov Variablenname, reg	reg = A oder reg = I
Lade A mit Wort in indirekt über I	mov A, [I]	
Lade Wort in indirekt über I aus A	mov [I], A	
Inkrementiere I um 1	inc I	**Z** und **C** verändert
Dekrementiere I um 1	dec I	**Z** und **C** verändert

Addiere reg plus Konstante oder Variable oder Wort in indirekt über I und gib Resultat in reg	add reg, KONSTANTE
	add reg, Variablenname
	add reg, [I]
		reg = A oder reg = I
		Z und **C** verändert

Subtrahiere reg plus Konstante oder Variable oder Wort in indirekt über I und gib Resultat in reg	sub reg, KONSTANTE
	sub reg, Variablenname
	sub reg, [I]
		reg = A oder reg = I
		Z und **C** verändert

Von reg wird der 2. Operand subtrahiert. Das Ergebnis wird nicht in reg geschrieben, sondern es wird in Zustandsbits festgehalten, ob gilt: **Z**ero-Bit: 1 = equal, 0 = not equal **C**arry-Bit: 0 = greater, 1 = below.	cmp reg, KONSTANTE cmp reg, Variablenname	reg = A oder reg = I
jump equal, wenn der compare-Befehl equal festgestellt und festgehalten hat.	je marke	
jump	jmp marke	
Programmablauf anhalten	hlt	