UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TECHNISCHE INFORMATIK

Studienbegleitende Fachprüfung im Rahmen der Bachelorprüfung

Wintersemester 2018/2019

Lehrmodul:
Technische Grundlagen der Informatik 2

Prüfer: Dr.-Ing. Kristian Ehlers

15. März 2019

Name: _____

Matrikelnummer: _____

Punkte:

1	2	3	∑
			/100

Aufgabe 1: Schaltwerksanalyse (33 Punkte)

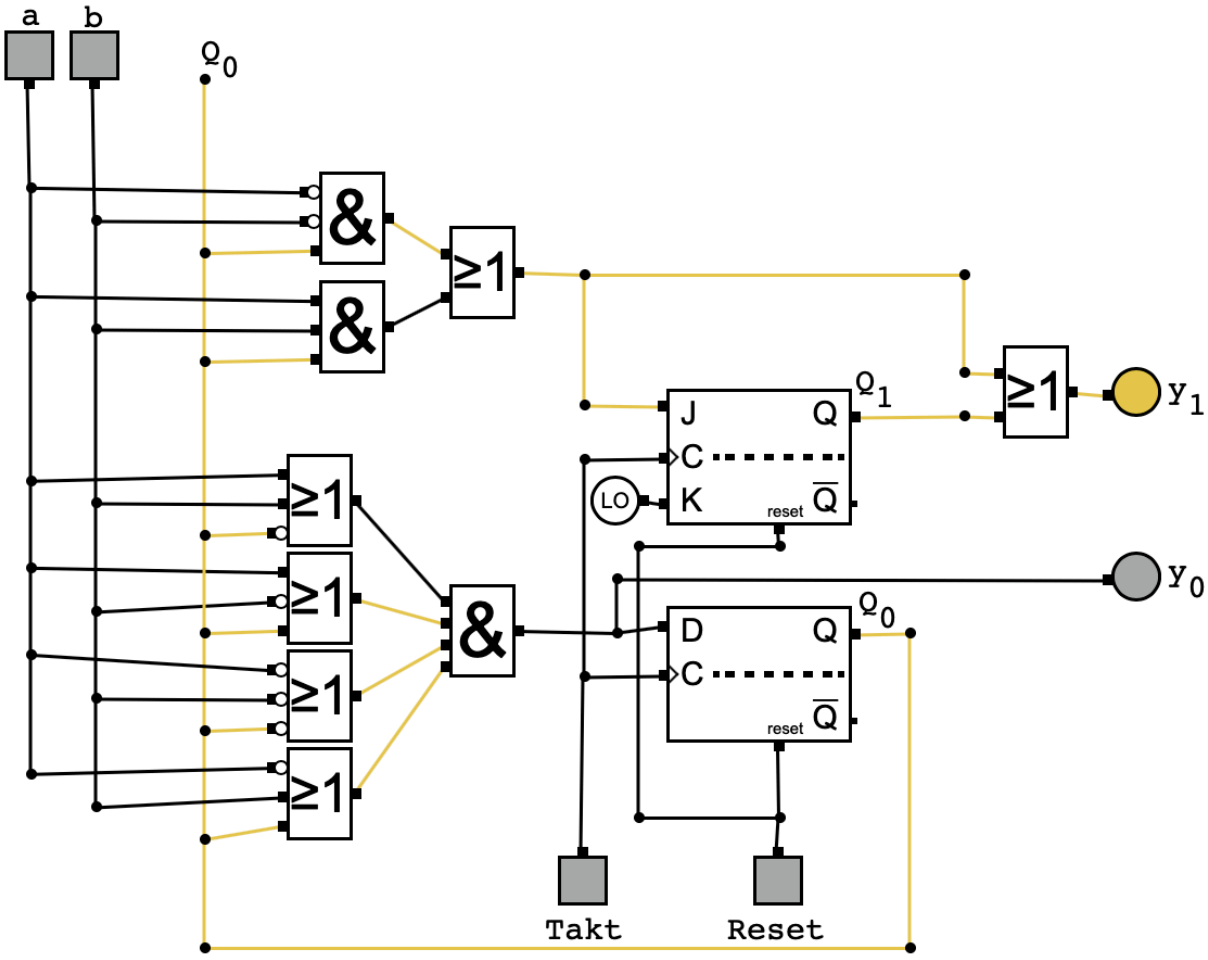


Abbildung 1: Schaltwerk

a) Betrachten Sie für diese Teilaufgabe das in Abbildung 1 dargestellte Schaltwerk, dessen Startzustand der Zustand 0 (Q_1Q_0=00) ist, und beantworten Sie die nachfolgenden Fragen.

Um welchen Automatentyp handelt es sich? Bitte begründen Sie Ihre Antwort kurz.

Der Automat soll nur mit Hilfe von JK-Flipflops realisiert werden. Welche Änderungen sind dafür notwendig? Wie viele Zustände kann der neue Automat annehmen? Begründen Sie Ihre Antwort.

Geben Sie die Ausgabefunktionen y_1 und y_0 des Automaten an.

Geben Sie die Ansteuergleichungen der für die Realisierung des Automaten verwendeten Flipflops (J_1und K_1 sowie D_0) an.

Geben Sie die zu den Ansteuergleichungen korrespondierenden Zustandsübergangsfunktionen (Z_1^{n+1} und Z_0^{n+1}) des Automaten an.

Das gegebene Schaltwerk soll nun in Form des anderen Automaten-Typen realisiert werden (sollte der gegebene Automat ein Moore-Automat sein, so soll er in einen Mealy-Automat transferiert werden oder umgekehrt).
Welche der nachfolgenden Aussagen ist wahr bzw. falsch? Kreuzen Sie die zutreffende Eigenschaft der jeweiligen Aussage an.

wahr	falsch	Aussage
☐	☐	Die Anzahl der Zustände ändert sich in jedem Falle.
☐	☐	Die Anzahl der Eingangsvariablen erhöht sich.
☐	☐	Die Ansteuergleichungen aller Flipflops bleiben gleich.
☐	☐	Die Anzahl der Ausgabebits bleibt gleich.
☐	☐	Die Ausgabe des Automaten bei gleicher Eingabe ändert sich.
☐	☐	Für die Realisierung werden auf jeden Fall weniger Flipflops benötigt als bei der gegebenen Variante.
☐	☐	Das Zurücksetzen des Schaltwerks kann weiterhin unabhängig vom Takt (asynchron) erfolgen, wenn die gleichen Flipflops verwendet werden.

b) Gegeben sei die nachfolgende Zustandsübergangstabelle

Z_1	Z_0	$(Z_1Z_0)^{n+1}$ bei Eingabe ab				$(y_1y_0)^{n+1}$ bei Eingabe ab			
		00	01	10	11	00	01	10	11
0	0	01	00	00	01	01	00	00	01
0	1	10	01	01	10	10	01	01	10
1	0	11	10	10	11	11	10	10	11
1	1	xx	xx	xx	xx	xx	xx	xx	xx

Geben Sie den Zustandsübergangsgraphen des Automaten an.

Welche Ausgabe-Sequenz von y_1y_0 erzeugt der Automat nach der bitweisen Eingabe der beiden Binärstrings a = 010 und b = 001? Als Startzustand sei $Z_1Z_0 = 00$ zu wählen. Es ist egal, ob Sie mit dem MSB oder LSB beginnen.

Welchem Wert zwischen zwei in diesem Falle 3-stelligen Binärzahlen entspricht die durch den Automaten bestimmte letzte Ausgabe?

Geben Sie die **Ausgabefunktionen** für y_0^n **in disjunktiver** und die für y_1^n **in konjunktiver** Minimalform an! Ihnen stehen für die Bestimmung die nachfolgenden KV-Diagramme zur Verfügung. Bitte füllen Sie diese komplett aus und kennzeichnen Sie Ihre Minimierungen!
Geben Sie zudem die Zustandsübergangsfunktionen des Automaten an!

y_1^n

$(Z_1Z_0)^n$ \ $(a,b)^n$	00	01	11	10
00				
01				
11				
10				

y_0^n

$(Z_1Z_0)^n$ \ $(a,b)^n$	00	01	11	10
00				
01				
11				
10				

y_1^n

$(Z_1Z_0)^n$ \ $(a,b)^n$	00	01	11	10
00				
01				
11				
10				

Ersatz

y_0^n

$(Z_1Z_0)^n$ \ $(a,b)^n$	00	01	11	10
00				
01				
11				
10				

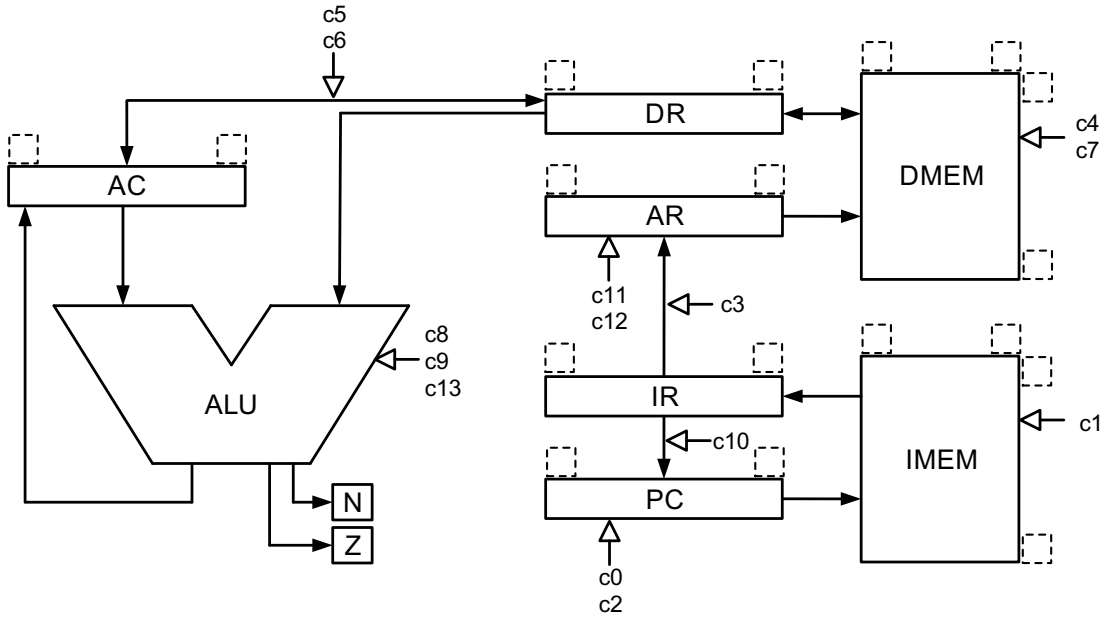
Ersatz

Ausgabefunktionen:

Zustandsübergangsfunktionen:

$\Sigma_{A1} =$ / 33 Punkte

Aufgabe 2: CPU (60 Punkte)



Die Breiten der nachfolgenden Steuersignale müssen hier nicht eingetragen werden!

c0	PC <- 0	c5	AC <- DR	c10	PC <- IR(…)
c1	read IMEM	c6	DR <- AC	c11	AR <- 0
c2	PC <- PC + 1	c7	write DMEM	c12	AR <- AR + 1
c3	AR <- IR(…)	c8	AC <- AC + DR	C13	AC <- AC + 1
c4	read DMEM	c9	N.AC <- AC - DR		

Abbildung 2: Operationswerk. Die Breiten der einzelnen Register und Registerteilbereiche sind bei der Angabe der Steuersignale nicht vorgegeben.

Gegeben sei das in Abbildung 2 dargestellte Operationswerk (OW) mit dem zwölf Bit breiten Datenspeicher DMEM, der bis zu 4096 Werte aufnehmen kann, und dem Instruktionsspeicher IMEM, in dem bis zu 256 Instruktionen abgelegt werden können. In den oberen Bits einer Instruktion ist der Befehl kodiert, wobei die auf Basis dieses OW zu entwerfende CPU bis zu 16 verschiedene Befehle zur Verfügung stellen soll. Gegebenenfalls repräsentieren die unteren Bits einer Instruktion rechtsbündig die Adresse des Befehls, die sich entweder auf den Daten- oder den Instruktionsspeicher beziehen kann.

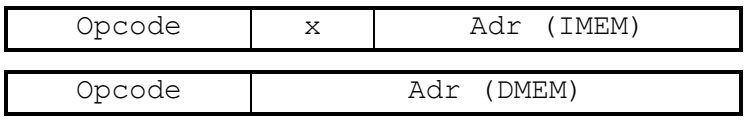


Abbildung 3: Befehlsformat

Das OW besitzt eine ALU, die addieren, subtrahieren sowie den Wert des Akkumulator-Registers AC inkrementieren kann. Dabei setzt sie das Negative-Flag N (N = 1), sollte bei der auszuführenden Operation ein negatives Ergebnis erzeugt werden bzw. das Zero-Flag Z (Z = 1), sollte das Ergebnis der ausgeführten Operation eine Null sein.

a) Ergänzen Sie im OW aus Abbildung 2 die fehlenden Breitenangaben der einzelnen Register in den gestrichelten Kästchen.
b) Untersuchen Sie den nachfolgenen RT-Code. Ergänzen Sie die fehlenden Registerbreiten.

```
declare register    AC(      ), DR(      ), AR(      ), PC(      ),
                    IR(      ), N, Z

declare memory      IMEM(      ), DMEM(      )

INIT:   AR <- 0;

LOOP:   read DMEM, AR <- AR + 1;
        AC <- DR, read DMEM;

MLOOP:  N.AC <- AC - DR |
        if N = 0 then
          goto MLOOP
        fi;
        AC <- AC + DR |
        if AC = 0 then
          goto END
        fi;
        AR <- 0;
        write DMEM, DR <- AC, AR <- AR + 1;
        write DMEM, AR <- 0, goto LOOP;

END:    AC <- DR, AR <- AR + 1;
        write DMEM;

ELOOP:  goto ELOOP;
```

Analysieren Sie nun das durch den RT-Code beschriebene Verhalten, indem Sie ihn taktweise durchgehen und die jeweiligen Änderungen der Register- und Speicherinhalte in der nachfolgende Tabelle angeben.

Welches Verhalten wird durch den RT-Code beschrieben? Kreuzen Sie zutreffendes an, wobei auch Teilberechnungen innerhalb des Algorithmus Berücksichtigung finden sollen.

- ☐ Hammingdistanz
- ☐ Modulo
- ☐ Kleinstes gemeinsames Vielfache (kgV)
- ☐ Multiplikation
- ☐ Wurzelberechnung
- ☐ Größter gemeinsamer Teiler

Takt	AR	DR	AC	N	DMEM		
					Adr 0	Adr 1	Adr 2
0	0	0	0	0	25_{10}	20_{10}	0
1	0						
2	1	25					
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							

c) Das gegebene Operationswerk soll um ein mikroprogrammiertes Steuerwerk zu einer mikroprogrammierten CPU ergänzt werden, die über den nachfolgenden Befehlssatz verfügt.

Opcode	Befehl	Beschreibung
0	LOAD X	Lädt den sich unter Adresse X im Datenspeicher befindenden Wert in das Akkumulator-Register AC
1	STORE X	Speichert den sich im Akkumulator-Register AC befindenden Wert unter der Speicheradresse X im Datenspeicher
2	ADD X	Addiert den sich unter Adresse X im Datenspeicher befindenden Wert auf den des Akkumulators
3	SUB X	Subtrahiert den sich unter Adresse X im Datenspeicher befindenden Wert von dem des Akkumulators
4	JMP X	Setzt den Programmablauf an Adresse X fort
5	JMPP X	Setzt den Programmablauf an Adresse X fort, wenn N = 0 gilt
6	JMPZ X	Setzt den Programmablauf an Adresse X fort, wenn Z = 1 gilt

Ergänzen Sie im nachfolgenden RT-Programm die fehlenden Registerbreiten und Indizes sowie die zu den Registertransferoperationen korrespondierenden Kontrollsignale (rechts neben dem Code. Wo keine Linie ist, soll auch kein Signal angegeben werden).

```
declare register  AC(…), DR(…), AR(…), PC(…),
                  IR(…), N, Z

declare memory IMEM(…), DMEM(…)
```

Es gilt die Deklaration der Register und des Speichers M aus Aufgabe b). Hier nicht nochmal angeben!

```
INIT: PC <- 0;                                                    _____

FETCH:  if AC = 0 then                                  # Setzen des Z-Flags
          Z <- 1
        else
          Z <- 0
        fi,
        read IMEM, PC <- PC + 1 | switch IR(       ) {          _____
        case 0: goto LOAD
        case 1: goto STORE
        case 2: goto ADD
        case 3: goto SUB
        case 4: goto JMP
        case 5: goto JMPP
        case 6: goto JMPZ
        default: goto FETCH };

LOAD:   AR <- IR(      );                                         _____
        read DMEM;                                                _____
        AC <- DR | goto FETCH;                                    _____

STORE:  DR <- AC, AR <- IR(      );                               _____
        write DMEM | goto FETCH;                                  _____

ADD:    AR <- IR(     );                                         _____
        read DMEM;                                                _____
        AC <- AC + DR | goto FETCH;                               _____

SUB:    AR <- IR(      );                                        _____
        read DMEM;                                                _____
        N.AC <- AC - DR | goto FETCH;                             _____

JMP:    PC <- IR(     ) | goto FETCH;                             _____

JMPP:   if N = 0 then
          PC <- IR(     ), goto FETCH                           _____
        else goto FETCH fi;

JMPZ:   if Z = 1 then
          PC <- IR(      ), goto FETCH                          _____
        else goto FETCH fi;
```

d) Implementieren Sie den Befehl `DIV X`, der den Wert von AC ganzzahlig durch den sich unter der Adresse X im Datenspeicher befindenden Wert teilt und das Resultat in AC ablegt. Sie können davon ausgehen, dass hier nur positive Zahlen Verwendung finden. Da hier nur der Akkumulator als eigenständiges Register zur Verfügung steht, müssen Zwischenergebnisse im Datenspeicher abgelegt werden. Es sei hier definiert, dass die CPU die ersten fünf Speicherstellen des DMEM für die Realisierung von Berechnungen exklusiv nutzt, Sie also bei Bedarf Ihre Hilfsvariablen in diesem Bereich ablegen können. Nachfolgend sind einige Hinweise zum Vorgehen bzw. einzelne Schritte kurz benannt.

- Die Division muss auf Basis der Addition und Subtraktion sowie unter ausschließlicher Verwendung der gegebenen Funktionalität des OW realisiert werden
- Legen Sie sich den aktuellen Wert des Akkumulators stets im Speicher ab
- In jedem Berechnungsschritt müssen der aktuelle Wert des Dividenden in das AC-Register und der des Divisors in das DR-Register geladen und der neue Akkumulatorwert im Speicher abgelegt werden

e) Erstellen Sie ein horizontales Mikroprogramm, welches das Verhalten des in der Tabelle gegebenen RT-Codes sowie des JMPP Befehls realisiert. Füllen Sie hierfür nachfolgende Tabelle aus. Leere Felder werden hierbei gegebenenfalls als 0 interpretiert. Sie müssen also nur die 1en eintragen. Ergänzen Sie zudem den für das Mikroprogramm eventuell abgeänderten RT-Code des JMPP Befehls. Es stehen Ihnen lediglich die nachfolgenden Condition Select Signale zur Verfügung:

Condition Select	Funktion
000	Nicht springen
001	Springe zu der dekodierten Opcode-Adresse (Mapping-ROM)
010	Springe, falls Z = 1
011	Springe, falls N = 1
111	Springe unbedingt

f) Erstellen Sie nun ein vertikales Mikroprogramm, indem Sie den entsprechenden Bereich in der Tabelle ergänzen. Eine gesonderte Angabe der Kodierung in einer separaten Tabelle ist nicht erforderlich.

Adresse					cs			Sprungadresse					Horizontale Kodierung Kontrollsignale c[16:0]													Vertikale Kodierung Kontrollsignale c[3:0]				Entsprechender RT-Befehl
4	3	2	1	0	2	1	0	4	3	2	1	0	13	12	11	10	9	8	7	6	5	4	3	2	1	0	3	2	1	0	
0	0	0	0	0																											INIT: PC <- 0;
0	0	0	0	1																											FETCH: read IMEM, PC <- PC + 1 | switch IR(…)
0	0	0	1	0																											LOAD: AR <- IR(…);
0	0	0	1	1																											read DMEM;
0	0	1	0	0																											AC <- DR | goto FETCH;
0	0	1	0	1																											STORE: DR <- AC, AR <- IR(…);
0	0	1	1	0																											write DMEM | goto FETCH;
0	0	1	1	1																											ADD: AR <- IR(…);
0	1	0	0	0																											read DMEM;
0	1	0	0	1																											AC <- AC + DR | goto FETCH;
0	1	0	1	0																											SUB: AR <- IR(…);
0	1	0	1	1																											read DMEM;
0	1	1	0	0																											N.AC <- AC - DR | goto FETCH;
0	1	1	0	1																											JMP: PC <- IR(…) | goto FETCH;
0	1	1	1	0																											JMPP:
0	1	1	1	1																											
1	0	0	0	0																											
1	0	0	0	1																											
1	0	0	1	0																											

g) Geben Sie das Mapping-ROM an.

h) Geben Sie den Speicherbedarf des horizontalen und des reinen vertikalen Mikroprogramms (ohne evtl. Dekodierer usw.) an.

i) Realisieren Sie das als RT-Code in Teilaufgabe b) gegebene Verhalten als Assemblerprogramm. Nutzen Sie hierbei nur die in Teilaufgabe c) gegebenen Assemblerbefehle! Geben Sie erst das kommentierte Assemblerprogramm an und bestimmen Sie anschließend die Speicherinhalte von Daten- und Instruktionsspeicher in der **Hexadezimaldarstellung**. Nutzen Sie als Beispielwerte die aus Teilaufgabe b) (25 und 20).

Assemblerprogramm:

Adresse	Speicherinhalt DMEM
00	
01	
02	
03	
04	
05	
06	
07	
08	
09	
0A	
0B	
0C	
0D	
0E	
0F	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
1A	
1B	
1C	
1D	
1E	
1F	

Adresse	Speicherinhalt IMEM
00	
01	
02	
03	
04	
05	
06	
07	
08	
09	
0A	
0B	
0C	
0D	
0E	
0F	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
1A	
1B	
1C	
1D	
1E	
1F	

$\Sigma_{A2} =$ /60 Punkte

Aufgabe 3: Allgemeine Fragen (7 Punkte)

a) Was versteht man unter einem Koppelterm?

b) Welche Arten von festverdrahteten Steuerwerken gibt es?

c) Was versteht man in Verbindung mit VHDL unter Architecture und Entity?

d) Worin unterscheiden sich das mehrfache und das quasi-horizontale Befehlsformat im Bereich der Mikroprogrammierung und welche Form hat bei gleicher Funktionalität die kürzere Ausführungszeit? Begründen Sie Ihre Antwort kurz.

$\Sigma_{A3} =$ /7 Punkte