UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TECHNISCHE INFORMATIK

Studienbegleitende Fachprüfung im Rahmen der Bachelorprüfung

Klausur aus dem Wintersemester 2020

Lehrmodul:
Technische Grundlagen der Informatik 2

Prüfer: Dr.-Ing. Kristian Ehlers

16. Februar 2021

Name: _____

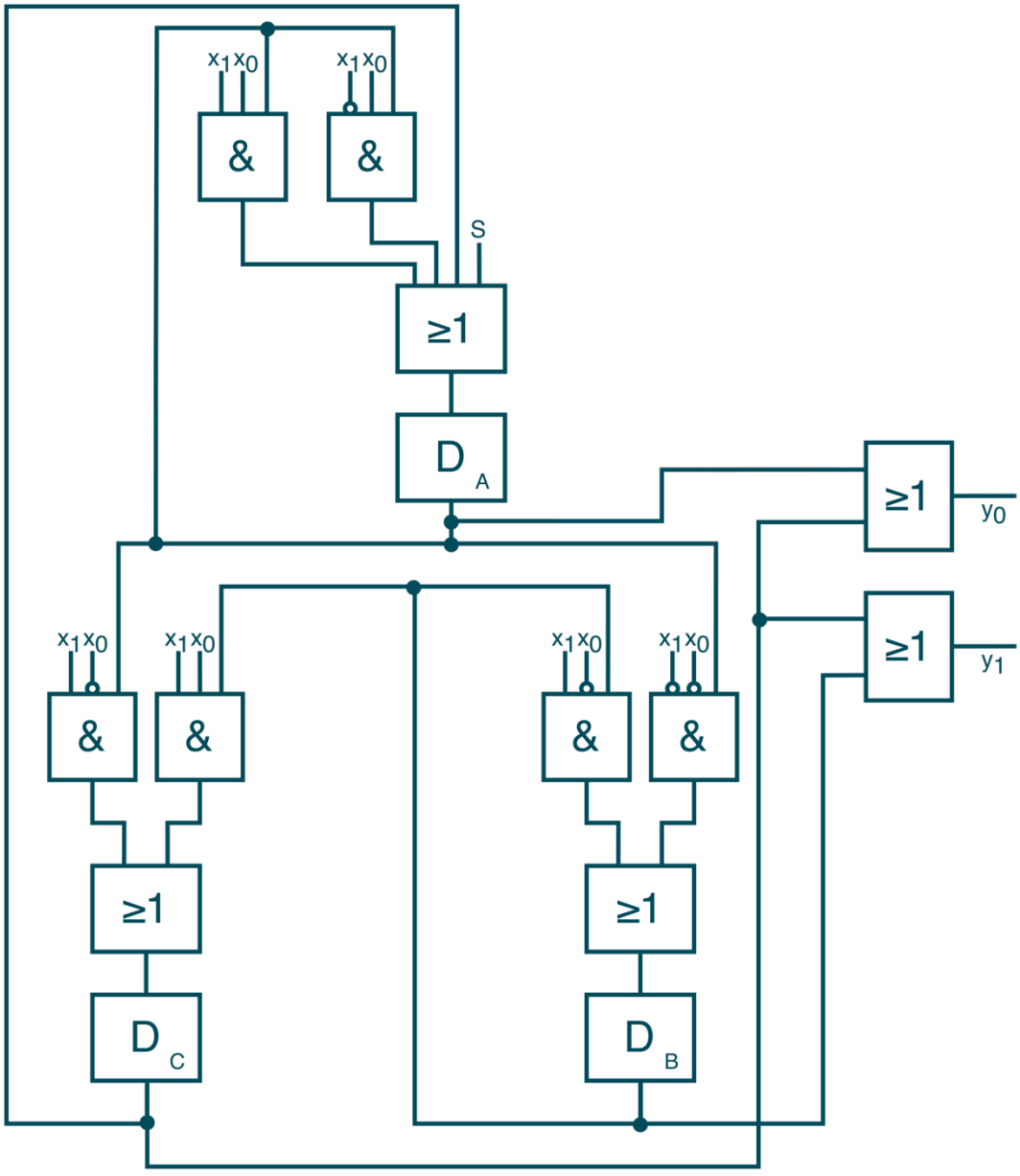
Matrikelnummer: _____

Punkte:

1	2	3	4	∑
				/ 100

Aufgabe 1: Schaltwerksanalyse (40 Punkte)

Betrachten Sie das nachfolgende Schaltwerk.



1) Um welche Form eines festverdrahteten Schaltwerks handelt es sich?

2) Welches Timing weist das Schaltwerk auf (Mealy oder Moore)? Bitte begründen Sie Ihre Antwort kurz.

3) Welche Ausgabe (y_1y_0) erzeugt die taktsequentielle Eingabe (Sx_1x_0) von 100 – 000 – 011 – 011? Falls es hier ein Problem geben sollte, beschreiben Sie es kurz.

4) Welche Ausgabe (y_1y_0) erzeugt die taktsequentielle Eingabe (Sx_1x_0) von 111 – 000 – 000 – 011? Falls es hier ein Problem geben sollte, beschreiben Sie es kurz.

5) Sind beliebige Flipflops als Verzögerungselemente einsetzbar? Was muss dabei evtl. beachtet werden?

6) Kann man das Schaltwerk auch mit weniger Flipflops realisieren? Begründen Sie Ihre Antwort.

7) Zeichnen Sie einen zum Schaltwerk äquivalenten Zustandsübergangsgraphen. Vernachlässigen Sie dabei die Eingabe des Startsignals S und verwenden Sie somit lediglich x_1 und x_0 als Eingaben. Die Zustände sollen die in den Verzögerungselementen gegebenen Bezeichnungen entsprechend (A, B und C) erhalten.
   - Um welchen Automatentypen handelt es sich bei Ihrem gezeichneten Zustandsübergangsgraphen? Begründen Sie Ihre Antwort kurz. Geben Sie an, warum Sie sich für diesen Automatentypen entschieden haben.
   - Ist der von Ihnen erstellte Automat vollständig oder partiell definiert? Begründen Sie Ihre Antwort kurz.

8) Geben Sie die VHDL-Beschreibung eines zum gegebenen Schaltwerk äquivalenten Moore-Automaten an, indem Sie den nachfolgenden Code um die fehlenden Abschnitte ergänzen. Vernachlässigen Sie dabei die Eingabe des Startsignals S und verwenden Sie somit lediglich x_1 und x_0 als Eingaben. Die Zustände sollen die in den Verzögerungselementen gegebenen Bezeichnungen entsprechend (A, B und C) erhalten. Sollten nicht definierte Eingaben auftreten, sorgen Sie dafür, dass der Automat in seinen Startzustand (A) überführt wird.
   - Geben Sie die Entity an.
   - Geben Sie den VHDL-Code an, der die Zustandsübergänge beschreibt.
   - Geben Sie den VHDL-Code an, der die Ausgaben realisiert.

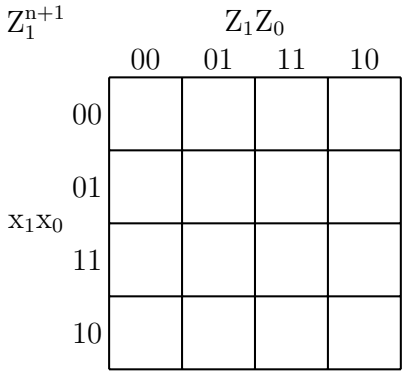
```
-- TODO – Entity angeben

architecture Behavioral of Automat is

signal x : STD_LOGIC_VECTOR(1 downto 0);
signal y : STD_LOGIC_VECTOR(1 downto 0);

type state_type is (A,B,C);
signal state : state_type := A;
signal next_state : state_type := A;

begin

x <= x1 & x0;

process_State_Register : process (reset, clock, next_state)
begin
 if (reset = '1') then
   state <= A;
 elsif rising_edge(clock) then
   state <= next_state;
 end if;
end process process_State_Register;
```

```
process_State_Logic : process (state, x)
begin

-- TODO – Zustandsübergänge definieren

end process process_State_Logic;

process_State_Output : process (state)
begin

-- TODO – Ausgaben definieren

end process process_State_Output;

y1 <= y(1);
y0 <= y(0);

end Behavioral;
```

Betrachten Sie ab hier für den Rest der Aufgabe 1 das nachfolgende in Form der einer Zustandsübergangstabelle gegebene Schaltwerk.

$(Z_1Z_0)^n$	Folgezustand $(Z_1Z_0)^{n+1}$				Ausgabe
	$x_1x_0 = 00$	$x_1x_0 = 01$	$x_1x_0 = 10$	$x_1x_0 = 11$	y_1y_0
00	xx	xx	xx	xx	xx
01	10	01	11	01	01
10	xx	xx	10	11	10
11	01	01	01	01	11

9) Bestimmen Sie sämtliche Formen von $Z_{1,\mathbf{KMF}}^{n+1}$, indem Sie eine Minimierung mit Hilfe eines KV-Diagramms der nachfolgenden Form durchführen und anschließend alle Formen von $Z_{1,\mathbf{KMF}}^{n+1}$ explizit angeben.



10) Verwenden Sie für das Schaltwerk aus der Tabelle die Zustandsübergangsfunktion

$$Z_{0,\mathrm{KMF}}^{n+1} = (Z_0 + x_0) * (Z_1 + x_1 + x_0)$$

und bestimmen Sie die Ansteuergleichung für ein D-Flipflop unter der Bedingung, dass es in einer Realisierung das Zustandsbit Z_0 repräsentieren soll.

Was ist der Vorteil, wenn man $Z_{0,\mathrm{KMF}}^{n+1}$ satt $Z_{0,\mathrm{KKN}}^{n+1}$ als Ausgangspunkt zum Bestimmen der Ansteuergleichung für das D-Flipflop im Rahmen der Realisierung des Schaltwerks nutzt?

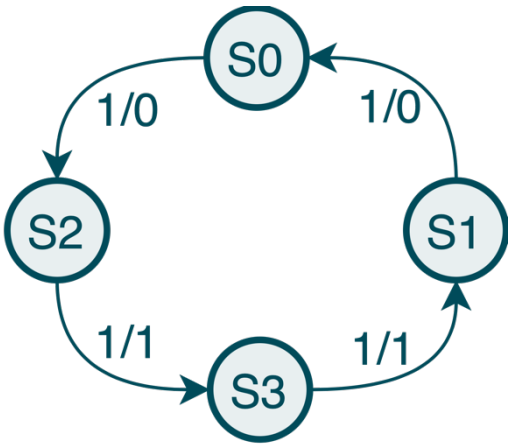
11) Leider stehen Ihnen für die Realisierung des Schaltwerks als Speicherelemente nur ein D-Flipflop und ein RS-Flipflop zur Verfügung. Bestimmen Sie aus diesem Grund direkt aus der Tabelle die Ansteuergleichungen als DMF für das RS-Flipflop (Eine Gleichung für R und eine für S) unter der Annahme, dass dieses das Zustandsbit Z_1 des Automaten repräsentieren soll. Nutzen Sie dafür die nachfolgende Tabelle, die Sie im Vorfeld teilweise vorbereiten oder ausdrucken durften. Sollten Sie das nicht getan haben, müssen Sie diese Tabelle abzeichnen. Sollte sich beim Zustandsübergang das Bit nicht ändern oder don't care sein, so soll das Flipflop durch die Beschaltung mit R und S seinen explizit Wert halten.

$Z_1Z_0x_1x_0$	Z_1^{n+1}	R	S
0000			
0001			
0010			
0011			
0100			
0101			
0110			
0111			
1000			
1001			
1010			
1011			
1100			
1101			
1110			
1111			

12) Geben Sie die Ausgabefunktionen des Schaltwerks an.

13) Realisieren Sie das Schaltwerk in Form einer Schaltung in LogicCircuits-Notation unter der Verwendung der eben bestimmten Ansteuergleichungen und Ausgabefunktionen mithilfe eines taktflankengesteuerten D-Flipflops und eines taktflankengesteuerten RS-Flipflops.

14) Ändern Sie das nachfolgende Schaltwerk soweit ab, dass es als Zustandsübergangstabelle dargestellt (Diese müssen Sie nicht angeben.) keine „don't care"-Terme mehr enthält. Sie dürfen keine Zustände entfernen! Kanten dürfen verändert werden.

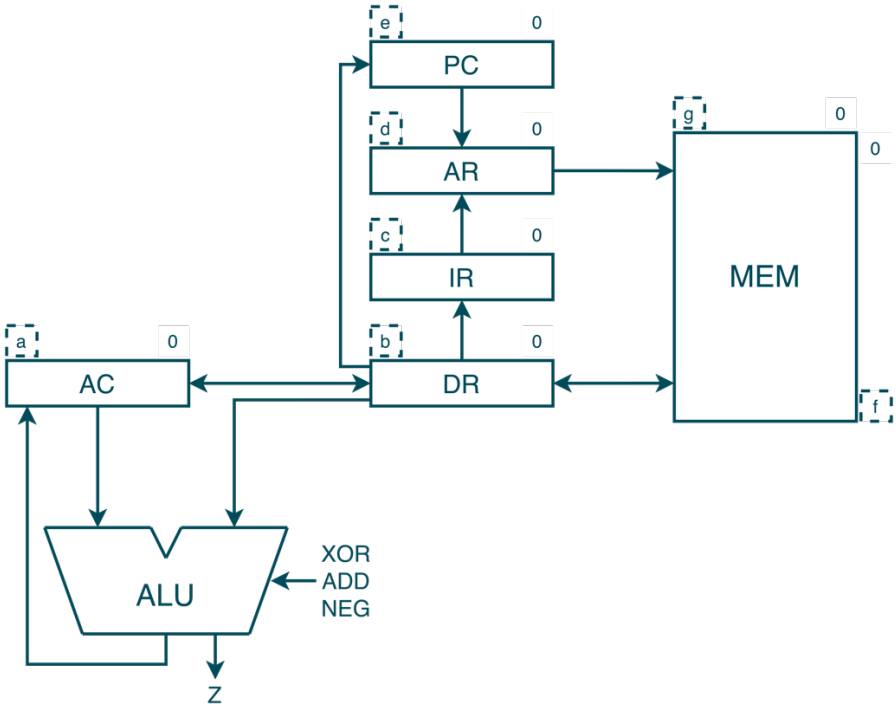


Welche der nachfolgenden Aussagen sind für Ihren abgeänderten Automaten im Vergleich zur vorherigen Version wahr bzw. falsch? Bitte geben Sie die Antwort (wahr/falsch) für jede Frage explizit an. Falsche Antworten führen zu Punktabzügen, allerdings kann es keine negativen Punkte für diese Teilaufgabe geben.

	Aussage
a	Die Anzahl der Zustände ändert sich nicht.
b	Es werden mehr Zustandsbits benötigt als zuvor.
c	Die Ansteuergleichungen aller Flipflops bleiben im Allgemeinen gleich.
d	Eine Realisierung mit JK-MS-Flipflops ist nicht mehr möglich.
e	Die Ausgabe des Automaten bei gleicher Eingabe ändert sich im Allgemeinen nicht.
f	Es gibt jetzt keinen äquivalenten Moore-Automaten mehr

$$\Sigma_{A1} = \_\_\_ \text{ / 40 Punkte}$$

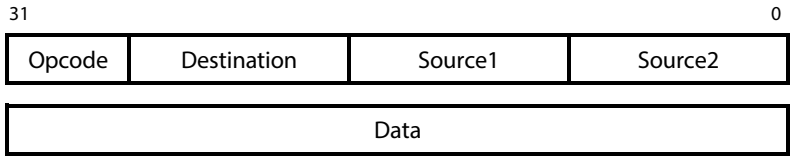
Aufgabe 2: Einfache CPU (34 Punkte)



Steuersignale

```
c0 – PC <– 0
c1 – PC <– PC + 1
c2 – PC <– DR(D)
c3 – PC <– IR(D)
c4 – read MEM
c5 – write MEM
c6 – IR <– DR
c7 – AR <– IR(S1)
c8 – AR <– IR(S2)
c9 – AR <– IR(D)
c10 – AR <– DR(S2)
c11 – AR <– PC
c12 – AC <– DR
c13 – AC <– AC + DR.        ≡ "AND"
c14 – AC <– (not DR) + 1    ≡ "NEG"
c15 – AC <– AC xor DR       ≡ "XOR"
c16 – DR <– AC
```

Hinweis: D, S1, S2 sind Platzhalter für die tatsächlichen Registerindizierungen



Befehls-/Datenformat

Opcode	Befehl	Beschreibung
0	`ADD D, S1, S2`	Addiert die beiden unter den Adressen S1 und S2 im Speicher abgelegten Werte und legt das Ergebnis im Speicher unter Adresse D ab.
1	`JMP D`	Setzt den Programmablauf an Adresse D fort.
2	`BREQ D, S1, S2`	Setzt den Programmablauf an Adresse D fort, sollten die unter den Adresse S1 und S2 im Speicher abgelegten Werte gleich sein.
3	`CLR D`	Setzt den unter Adresse D erreichbaren Speicherinhalt auf 0.
4	`ADI D, S1, S2`	Addiert den unter der in S1 in den unteren 9 Bit abgelegten Adresse erreichbaren Wert des Speichers auf den unter S2 im Speicher abgelegten Wert und legt das Ergebnis im Speicher unter Adresse D ab.

Gegeben sei das oben dargestellte Operationswerk (OW) mit dem Speicher MEM, der bis zu 512 Werte aufnehmen kann. Das Befehls-/Datenformat der CPU ist ebenfalls oben dargestellt. Sollte es sich bei dem Speicherinhalt nicht um einen Befehl handeln, so werden die kompletten 32 Bit als Daten interpretiert und müssen von der ALU verarbeitbar sein. Eventuell durch die Operationen der ALU auftretenden Überläufe sollen verloren gehen. Die CPU stellt das Z-Flag bereit, welches angibt, ob das Ergebnis der letzten Operation eine Null (Z=1) war.

1) Ergänzen Sie im OW die fehlenden Breitenangaben der einzelnen Register in dem Sie den Buchstabe in den gestrichelten Kästchen jeweils die Breite des entsprechenden Registers zuordnen. Jedes Register soll nur so breit sein, wie es notwendig ist. Sollten Sie einige Werte nicht direkt berechnen können genügt die Angabe als Potenz von 2 (beipielsweise 2^2 statt 4). Bei den Angaben von Destination, Source1 und Source2 handelt es sich um Speicheradressen. Das Instruktionsregister muss sowohl den Opcode, als auch die Destination- und Source-Adressen aufnehmen können.

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

Deklarieren Sie die benötigten Register und den Speicher und notieren Sie für das nachfolgenden RT-Programm die Indizierungen (unter Angabe der roten Buchstaben) sowie die zu den Registertransferoperationen korrespondierenden Kontrollsignale unter Angabe der zugehörigen Zeilennummer.

Implementieren Sie zudem die Befehle **CLR** sowie **ADI** unter Einhaltung des **Moore-Timings**.

Hinweis: Das Setzen des Flags ist im nachfolgenden Quellcode nicht implementiert. Dennoch muss es bei Bedarf von Ihnen genutzt werden.

```
 1. -- TODO – Deklaration der Register
 2. -- TODO – Deklaration des Speichers
 3. INIT:   PC <- 0;                                          3. _____
 4.
 5. FETCH:  AR <- PC, PC <- PC + 1;                           5. _____
 6.         read MEM;                                         6. _____
 7.         IR <- DR | switch IR( a ) {                       7. _____
 8.             case 0: goto ADD
 9.             case 1: goto JMP
10.             case 2: goto BREQ
11.             case 3: goto CLR
12.             case 4: goto ADI
13.             default: goto FETCH };
14.
15. ADD:    AR <- IR( b );                                   15. _____
16.         read MEM;                                        16. _____
17.         AC <- DR, AR <- IR( c );                         17. _____
18.         read MEM;                                        18. _____
19.         AC <- AC + DR;                                   19. _____
20.         DR <- AC, AR <- IR( d );                         20. _____
21.         write MEM | goto FETCH;                          21. _____
22.
23. JMP:    PC <- DR( e ) | goto FETCH;                      23. _____
24.
25. BREQ:   AR <- IR( f );                                   25. _____
26.         read MEM;                                        26. _____
27.         AC <- (not DR) + 1, AR <- IR( g );               27. _____
28.         read MEM;                                        28. _____
29.         AC <- AC + DR;                                   29. _____
30.         if (AC = 0) then
31.             PC <- IR( h )                                31. _____
32.         fi | goto FETCH;

-- TODO – Implementierung von
CLR:
-- TODO – Implementierung von
ADI:
```

3) Erstellen Sie ein horizontal mikroprogrammiertes Steuerwerk, welches das durch den **gegebenen** RT-Code aus Aufgabe b) spezifizierte Verhalten realisiert. **CLR** und **ADI** müssen **nicht** realisiert werden! Füllen Sie hierfür die nachfolgende Tabelle aus. Sollten Sie sich diese Tabelle im Vorfeld ausgedruckt oder händisch vorbereitet haben, verwenden Sie diese, anderenfalls müssen Sie die Tabelle abzeichnen. Leere Felder werden hierbei als 0 interpretiert. Sie müssen also nur die 1en eintragen. Ergänzen Sie zudem den für das Mikroprogramm eventuell abgeänderten RT-Code. Es stehen Ihnen **ausschließlich** die nachfolgenden Condition Select Signale zur Verfügung:

Condition Select	Funktion
00	Nicht springen
01	Springe zu der dekodierten Opcode-Adresse (Mapping-ROM)
10	Springe, falls **Z = 1**
11	Springe unbedingt

Adresse					cs		Sprungadresse					Horizontale Kodierung Kontrollsignale c[16:0]																	RT-Code (NUR ÄNDERUNGEN ZUM URSRPÜNGLICHEN CODE EINTRAGEN)
4	3	2	1	0	1	0	4	3	2	1	0	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0																									INIT:
0	0	0	0	1																									FETCH:
0	0	0	1	0																									
0	0	0	1	1																									
0	0	1	0	0																									ADD:
0	0	1	0	1																									
0	0	1	1	0																									
0	0	1	1	1																									
0	1	0	0	0																									
0	1	0	0	1																									
0	1	0	1	0																									
0	1	0	1	1																									JMP:
0	1	1	0	0																									BREQ:
0	1	1	0	1																									
0	1	1	1	0																									
0	1	1	1	1																									
1	0	0	0	0																									
1	0	0	0	1																									
1	0	0	1	0																									
1	0	0	1	1																									
1	0	1	0	0																									
1	0	1	0	1																									

4) Geben Sie das Mapping ROM an.

5) Geben Sie den Speicherbedarf Ihres Mikroprogramms an.

6) Betrachten Sie den gegebenen Automaten, der die Ausführungsphase eines Befehls der CPU dieser Aufgabe beschreibt. Was macht dieser Befehl? Geben Sie die Antwort in einem Satz!

7) Geben Sie eine Folge von Assemblerdirektiven für die hier erstellte CPU an, um in einer Schleife die Werte eines ab Adresse $20 im Speicher abgelegte Arrays der Länge 5 aufzusummieren und das Ergebnis im Speicher unter Adresse $30 abzulegen. Ihnen stehen dabei die folgenden im Speicher abgelegten Hilfsvariablen zur Verfügung. Bis auf die Werte des Arrays dürfen sämtliche Speicherinhalte verändert werden. Sie dürfen alle in dieser Aufgabe realisierten Befehle (inkl. CLR und ADI) verwenden.

Adresse	Wert	Erklärung
…		
$2A	$5	Länge des Arrays
$2B	$20	Anfangsadresse des Arrays
$2C	-1	
$2D	1	
$2E	0	
…		

$$\Sigma_{A2} = \_\_\_ / 34 \text{ Punkte}$$

Aufgabe 3: RISC-V und EduCore-V Tiny (21 Punkte)

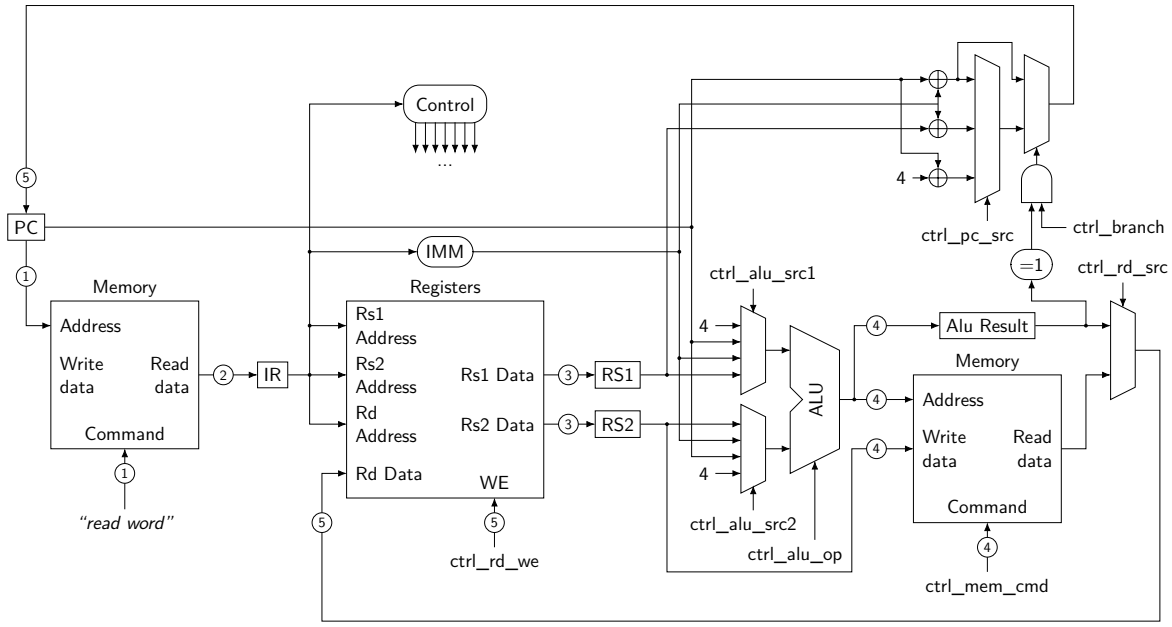


Abbildung 1: Schematische Darstellung des EduCore-V Tiny

31 30 29 28 27 26 25	24 23 22 21 20	19 18 17 16 15	14 13 12	11 10 9 8 7	6 5 4 3 2 1 0
func7	rs2	rs1	func3	rd	opcode
0 0 0 0 0 1 1	- - - - -	- - - - -	0 1 0	- - - - -	0 1 1 0 0 1 1

Abbildung 2: Befehlsformat des `det` Befehls

`ctrl_alu_src1`	`REG, IMM, PC, 4`
`ctrl_alu_src2`	`REG, IMM, PC, 4`
`ctrl_mem_cmd`	`WRITE_B, WRITE_H, WRITE_W, READ_B, READ_H, READ_W, READ_BU, READ_HU, NOP`
`ctrl_alu_op`	`NOP, OP1, OP2, ADD, SUB, OR, AND, XOR, SLL, SRL, SRA, EQ, NEQ, LT, GE, LTU, GEU,` **`DET`**
`ctrl_rd_src`	`ALU_RESULT, MEM_DATA`
`ctrl_rd_we`	`false, true`
`ctrl_pc_src`	`PC_NEXT, PC_IMM, RS1_IMM`
`ctrl_branch`	`false, true`

Abbildung 3: Kontrollsignale ergänzt um `det`

In dieser Aufgabe sollen Sie den RV32I Basisbefehlssatz des in Abbildung 1 schematisch dargestellten EduCore-V Tiny um die Determinante einer Matrix erweitern. Der Befehl `det` soll die Berechnung der Determinante einer Matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathbb{N}^{2\times 2}$ in der Art vornehmen, dass

$$D = det\begin{pmatrix} a & b \\ c & d \end{pmatrix} = a \cdot d - c \cdot b$$

wobei für die einzelnen Elemente der Matrix $0 \leq a, b, c, d < 2^{16}$ gilt.

Für die Implementierung des Befehls wird die Matrix als ihre beiden Zeilenvektoren (a, b) und (c, d) betrachtet. Diese Vektoren werden jeweils auf ein eigenes Register abgebildet, wobei das erste Element des Vektors auf die oberen 16 Bit des Registers und das zweite Element des Vektors auf die unteren 16 Bit des Registers abgebildet werden:

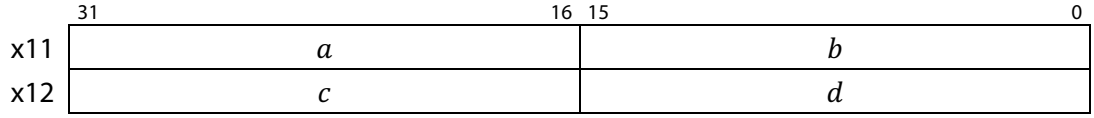


Abbildung 4: Beispiel, wie Register `x11` und `x12` eine Matrix darstellen

Gemäß Abbildung 2 soll der Befehl `det` zu den Register-Register Befehlen gehören. Er bekommt die beiden Spaltenvektoren als Parameter (`rs1`: erster Vektor, `rs2`: zweiter Vektor) und schreibt den Wert der Determinante in das Zielregister `rd`. Der Befehl soll den `opcode` 0110011 erhalten. Ferner wird der `func3` Teil auf 010 und der `func7` teil auf 0000011 gesetzt.

1) Könnten der func7 und func3 auch anders gewählt werden? Begründen Sie Ihre Antwort kurz.

2) Realisieren Sie die Berechnung der Determinante unter der ausschließlichen Verwendung des RV32I Basisbefehlssatzes sowie der Pseudoinstruktionen als RISC-V Assembler Programm. **Zudem darf der in der praktischen Übung realisierte `mul` Befehl verwendet werden**. Laden Sie in die beiden Register `x11` und `x12` die nachfolgende Matrix M gemäß Abbildung 9 und führen Sie die Berechnung der Determinante durch, wobei das Ergebnis nach der Berechnung in Register `x10` abgelegt werden soll:

$$M = \begin{pmatrix} 0\text{x}11 & 0\text{x}22 \\ 0\text{x}33 & 0\text{x}44 \end{pmatrix}$$

3) Geben Sie die vom Kontrollsignalgenerator für die Realisierung des `det` Befehls zu erzeugenden Kontrollsignale an, indem Sie **sämtliche** Signalwerte in der nachfolgenden Tabelle ergänzen. Orientieren Sie sich dafür an Abbildung 1 und an die in Abbildung 3 angegebenen Werte für die Kontrollsignale.

Kontrollsignal	Wert
`ctrl_alu_src1`	
`ctrl_alu_src2`	
`ctrl_mem_cmd`	
`ctrl_alu_op`	
`ctrl_rd_src`	
`ctrl_rd_we`	
`ctrl_pc_src`	
`ctrl_branch`	

4) Ergänzen Sie nun den nachfolgenden Codeausschnitt der ALU um den `det` Befehl. Nutzen Sie für die case-Abfrage die Konstante `CTRL_ALU_OP_DET`. Für die Beschreibung der Operation brauchen Sie keine Konvertierung der Operanden in unsigned angeben. Sie brauchen den gegebene Quellcode dabei nicht abzuschreiben.

```
architecture rtl of m_alu is
begin

    process(reg_op1, reg_op2, operation)
    begin
        case (operation) is
            -- ...
            when CTRL_ALU_OP_ADD => result <= reg_op1 + reg_op2;
            when CTRL_ALU_OP_SUB => result <= reg_op1 - reg_op2;

            when CTRL_ALU_OP_MERGE
                 => result <= reg_op1(31 downto 16) & reg_op2(15 downto 0);

            when CTRL_ALU_OP_OR  => result <= reg_op1 or  reg_op2;
            when CTRL_ALU_OP_AND => result <= reg_op1 and reg_op2;
            when CTRL_ALU_OP_XOR => result <= reg_op1 xor reg_op2;
            -- ...
-- LOESUNG BEGIN -









-- LOESUNG END -
            when others => result <= 32x"0";
        end case;
    end process;

end architecture rtl;
```

5) Realisieren Sie die Berechnung der Determinante unter Verwendung des von Ihnen erweiterten RV32I Basisbefehlssatzes sowie Pseudoinstruktionen als RISC-V Assembler Programm. Laden Sie in die beiden Register `x11` und `x12` die Matrix M aus Aufgabenteil c) und führen Sie die Berechnung der Determinante durch, wobei das Ergebnis nach der Berechnung in Register `x10` abgelegt werden soll. Nutzen Sie dafür explizit den von Ihnen realisierten Befehl `det`. Bedenken Sie, dass der Compiler Ihren Befehl nicht kennt.

$$\Sigma_{A3} = \_\_\_ \text{ / 21 Punkte}$$

Aufgabe 4: Allgemeine Fragen (5 Punkte)

1) Nennen Sie einen Vor- und einen Nachteil eines partiell definierten Automaten.

2) Stimmt es, dass Programme für CISC-Prozessoren meist kürzer sind als Programme gleicher Funktionalität für RISC-Prozessoren? Begründen Sie ihre Antwort!

3) Nennen Sie einen Vor- und einen Nachteil festverdrahteter Steuerwerke im Vergleich zu mikroprogrammierten Steuerwerken.

4) Sind Mikroprogramme mit mehrfachem Befehlsformat immer länger als welche im einfachen horizontalen Befehlsformat? Begründen Sie Ihre Antwort.

5) Kann jeder Moore-Automat in einen Mealy-Automaten überführt werden? Begründen Sie Ihre Antwort.

$\Sigma_{A4} =$ ___ / 5 Punkte