Department of Computer Engineering
School of Computation, Information and Technology
Technische Universität München

Lehrstuhl für
Rechnerarchitektur & Parallele Systeme
Prof. Dr. Martin Schulz
Dominic Prinz
Jakob Schäffeler

Lehrstuhl für
Design Automation
Prof. Dr.-Ing. Robert Wille
Stefan Engels
Benjamin Hien

Einführung in die Rechnerarchitektur

Wintersemester 2025/2026

Übungsblatt 10: Altklausur & Bonusaufgabe 22.12.2025 – 09.01.2026

Als Weihnachtsgeschenk der ERA-Übungsleitung und zur Vorbereitung auf die Klausur im Februar, finden Sie eine Bonusaufgabe (s.u.) auf Artemis sowie die Endterm-Klausur des letzten Jahres auf den folgenden Seiten.

Schauen Sie sich diese bitte schon im Vorhinein an und geben sie ihrem:ihrer Tutor:in Bescheid, welche Aufgaben der Altklausur Sie gerne ausführlich im Tutorium besprechen möchten. Die restlichen Aufgaben sind dann im Selbststudium zu erledigen.

Wichtig! Die hier zur Verfügung gestellte Altklausur dient lediglich als Orientierungshilfe. Sie soll einen Eindruck von möglichen Aufgabenformaten und der Struktur der Klausur vermitteln.

Bitte beachten Sie, dass sich Themenschwerpunkte, Aufgabenstellungen, Aufgabentypen, Bepunktung usw. in zukünftigen Klausuren ändern können. Eine alleinige Vorbereitung mit dieser Klausur ist deshalb nicht ausreichend und ersetzt keinesfalls die intensive Auseinandersetzung mit den aktuellen Vorlesungs- und Zentralübungsmaterialien sowie den Übungsblättern (inkl. Hausaufgaben). Das Auswendiglernen alter Lösungen ersetzt nicht das Verständnis der zugrundeliegenden Konzepte.

Aus der Altklausur können keinerlei Ansprüche hinsichtlich Form oder Inhalt bevorstehender Klausuren abgeleitet werden.

RFC 9402 (Bonusaufgabe 1)

Freiwillige Bearbeitung und Abgabe der Bonusaufgabe 1 auf `https://artemis.tum.de/courses/516` bis **Sonntag, den 25.01.2026, 23:59 Uhr**.

Diese Bonusaufgabe dient der Wiederholung der Konzepte der Assembly-Programmierung. Sie können bis zu 10 Punkte erreichen. Diese Punkte sind Bonuspunkte – sie zählen also nicht zu den „normal“ erreichbaren Punkten. Mit anderen Worten können Sie verpasste Punkte aus anderen Hausaufgaben wieder gut machen bzw. sich einen kleinen Puffer erarbeiten.

Wir hoffen, dass Ihnen die Bonusaufgabe Spaß machen wird! Für Fragen nutzen Sie bitte den eigens eingerichteten Channel auf Zulip. Frohe Weihnachten und einen guten Rutsch ins Jahr 2026!

Lehrstuhl für Rechnerarchitektur & Parallele Systeme, Lehrstuhl für Design Automation
Department of Computer Engineering
Technische Universität München

Eexam
Sticker mit SRID hier einkleben

Hinweise zur Personalisierung:

- Ihre Prüfung wird bei der Anwesenheitskontrolle durch Aufkleben eines Codes personalisiert.
- Dieser enthält lediglich eine fortlaufende Nummer, welche auch auf der Anwesenheitsliste neben dem Unterschriftenfeld vermerkt ist.
- Diese wird als Pseudonym verwendet, um eine eindeutige Zuordnung Ihrer Prüfung zu ermöglichen.

Einführung in die Rechnerarchitektur

Klausur:	Altklausur Endterm 2024/25	**Datum:**	N/A
Prüfer:	Prof. Dr. Martin Schulz	**Uhrzeit:**	N/A
	Prof. Dr. Robert Wille		

Bearbeitungshinweise

- Diese Klausur umfasst **28 Seiten** mit insgesamt **11 Aufgaben**.
  Bitte kontrollieren Sie jetzt, dass Sie eine vollständige Angabe erhalten haben.
- Die Gesamtpunktzahl in dieser Klausur beträgt 120 Punkte.
- Das Heraustrennen von Seiten aus der Prüfung ist untersagt.
- Als Hilfsmittel sind zugelassen:
  - ein **analoges Wörterbuch** Deutsch ↔ Muttersprache **ohne jegliche Anmerkungen**
- Darüber hinaus sind keine Hilfsmittel – insbesondere keine selbst mitgebrachten Schmierblätter – zugelassen.
- Mit * gekennzeichnete Teilaufgaben sind ohne Kenntnis der Ergebnisse vorheriger Teilaufgaben lösbar.
- **Es werden nur solche Ergebnisse gewertet, bei denen der Lösungsweg erkennbar ist.** Auch Textaufgaben sind **grundsätzlich zu begründen**, sofern es in der jeweiligen Teilaufgabe nicht ausdrücklich anders vermerkt ist. Geben Sie unaufgefordert mehrere Antworten, wird die schlechteste zur Bewertung herangezogen.
- Schreiben Sie weder mit roter / grüner Farbe noch mit Bleistift. Es darf kein Tipp-Ex benutzt werden!
- Bei Multiple Choice-Fragen müssen **alle** korrekten Antwortmöglichkeiten angekreuzt werden. Es ist jeweils mindestens eine Antwort korrekt.
- Wenn Sie eine Aufgabe nicht im vorgesehenen Platz lösen können, finden Sie am Ende der Klausur zusätzliche Blätter. Kennzeichnen Sie die Benutzung dieser deutlich bei der entsprechenden Aufgabe.
- Schalten Sie alle mitgeführten elektronischen Geräte (insbesondere Smartwatches) vollständig aus, verstauen Sie diese in Ihrer Tasche und verschließen Sie diese.
- Sofern nicht anders angegeben, sind für Assembly-Aufgaben nur die Instruktionen der `RV32IM` ISA sowie die offiziellen Pseudoinstruktionen zu benutzen. Es gilt die ratifizierte ISA-Spezifikation „RISC-V Unprivileged ISA“ in der Version 20191213. Sie finden eine Befehlsreferenz am Ende der Klausur. **Halten Sie sich außerdem stets an die Calling-Convention!**

Hörsaal verlassen von _____ bis _____ / Vorzeitige Abgabe um _____

Aufgabe 1 **Allgemeine Fragen (11 Punkte)**

Kreuzen Sie richtige Antworten an ☒
Kreuze können durch vollständiges Ausfüllen gestrichen werden ■
Gestrichene Antworten können durch nebenstehende Markierung erneut angekreuzt werden ×■

a)* Wie lautet die binäre Repräsentation der Zahl `0xB4FF`?

- ☐ 1011 0100 1111 1111
- ☐ 1111 1111 0100 1011
- ☐ 0100 1011 1111 1111
- ☐ 1111 1111 1011 0100

b)* Wie lautet die hexadezimale Repräsentation der Binärzahl `0101 1110`?

- ☐ `0xE5`
- ☐ `0x5E`
- ☐ `0xD3`
- ☐ `0x3D`

c)* Welcher Wert ergibt sich, wenn Sie die Zahl 1100 1101 als 8-Bit `unsigned char` interpretieren?

- ☐ 220
- ☐ 179
- ☐ 61
- ☐ 205

d)* Welcher Wert ergibt sich, wenn Sie die Zahl 1100 1101 als 8-Bit `signed char` interpretieren?

- ☐ -36
- ☐ -51
- ☐ 61
- ☐ 77

e)* Der Wertebereich einer Zahl mit n Bits im Zweierkomplement ist ...

- ☐ $\left[-2^{n-1}, 2^{n-1}+1\right]$
- ☐ $\left[-2^{n-1}+1, 2^{n-1}\right]$
- ☐ $\left[-2^{n-1}-1, 2^{n-1}\right]$
- ☐ $\left[-2^{n-1}, 2^{n-1}-1\right]$

f)* 8 KiB sind ...

- ☐ 64000 Bits
- ☐ 8000 Bytes
- ☐ 65536 Bits
- ☐ 8192 Bytes

g)* In den folgenden Abbildungen ist zu sehen, wie das Datum `0xC337DE7D` im Speicher abgelegt werden kann.

Adresse	Daten
n+0	C3
n+1	37
n+2	DE
n+3	7D

Abbildung 1.1

Adresse	Daten
n+0	7D
n+1	DE
n+2	37
n+3	C3

Abbildung 1.2

Welche Aussagen sind korrekt?

- ☐ Abbildung 1.1 zeigt das big-endian Format.
- ☐ Abbildung 1.2 zeigt das big-endian Format.
- ☐ Abbildung 1.2 zeigt das little-endian Format.
- ☐ Abbildung 1.1 zeigt das little-endian Format.

h)* Welche Aussagen sind korrekt?

- ☐ Pascal-Strings sind NULL-terminiert
- ☐ Pascal-Strings haben ein Längenpräfix
- ☐ C-Strings sind NULL-terminiert
- ☐ C-Strings haben ein Längenpräfix

i)* Die ASCII-Kodierung arbeitet mit . . .

- ☐ 8 Bit
- ☐ 7 Bit
- ☐ 2 Byte
- ☐ 1 Byte

j)* Gegeben sei folgendes `struct`:

```
struct quiz {
    long era;
    int ist;
    unsigned short super;
} exam;
```

Nehmen Sie für die folgenden Aufgaben an, dass ein `short` 16-Bit, ein `int` 32-Bit und ein `long` 64-Bit groß ist.
Die Adresse von `exam` sei in `a0` gespeichert. Welche/r Befehl/e ist/sind korrekt, um `ist` in `a1` zu laden?
Hinweis: Es gibt hier kein Padding.

- ☐ `lh a1, 8(a0)`
- ☐ `lb a1, 4(a0)`
- ☐ `lh a1, 4(a0)`
- ☐ `lw a1, 8(a0)`
- ☐ `lw a1, 4(a0)`
- ☐ `lb a1, 8(a0)`

k)* Die Adresse von `exam` sei in `a0` gespeichert. Welche/r Befehl/e ist/sind korrekt, um `super` in `a1` zu laden?

- ☐ `lb a1, 10(a0)`
- ☐ `lh a1, 10(a0)`
- ☐ `lhu a1, 10(a0)`
- ☐ `lbu a1, 10(a0)`
- ☐ `ld a1, 12(a0)`
- ☐ `lw a1, 12(a0)`
- ☐ `lbu a1, 12(a0)`
- ☐ `lw a1, 10(a0)`
- ☐ `lhu a1, 12(a0)`
- ☐ `lb a1, 12(a0)`
- ☐ `lh a1, 12(a0)`
- ☐ `ld a1, 10(a0)`

Aufgabe 2 RISC-V Basics (11 Punkte)

Nehmen Sie für die folgenden Aufgaben an, dass ein `short` 16-Bit und ein `int` 32-Bit groß ist. Außerdem sind alle Pointer 32-Bit groß.
Gegeben sei folgendes RISC-V Unterprogramm `mystery`, das ein Argument entgegennimmt:

```
mystery:
    li   a1, 0
loop:
    andi t0, a0, 1
    add  a1, a1, t0
    srli a0, a0, 1
    bne  a0, zero, loop
    mv   a0, a1
```

0 1 2 3

a)* Sei `a0 := 0b0111 1110 1001`. Geben Sie ohne weitere Begründung den Rückgabewert nach der Ausführung von `mystery` in dezimaler Darstellung an.

0 1 2

b)* Welche Operation setzt `mystery` um?

0 1 2 3

c)* Gegeben sei folgender C-Code:

```
unsigned int strlen(const char *str) {
    unsigned int length = 0;
    while (str[length] != 0)
        length++;

    return length;
}
```

Übersetzen Sie das Unterprogramm `strlen` in RISC-V Assembly.
*Hinweis: Sie finden bei dieser Aufgabe mehrere Vordrucke für Ihre Lösung. Sollten Sie davon mehr als einen verwenden, kennzeichnen Sie **deutlich** (z.B. durch Durchstreichen der anderen Vordrucke), welcher zur Bewertung herangezogen werden soll.*

```
strlen:
    mv   t0, a0
    li   a0, 0
loop:
    ____  t1, ____
    ____  t1, zero, end
    addi a0, a0, 1
    addi t0, t0, 1
    j    loop
end:
    ret
```

```
strlen:
    mv   t0, a0
    li   a0, 0
loop:
    ____  t1, ____
    ____  t1, zero, end
    addi a0, a0, 1
    addi t0, t0, 1
    j    loop
end:
    ret
```

d)* Gegeben sei folgende Signatur einer C-Funktion:

```
int test(uint64_t a, short b, int c);
```

Wo wird `a` übergeben?

☐ `a2`	☐ `a0`	☐ `a6`	☐ `a4`
☐ `a5`	☐ `a3`	☐ `a1`	☐ Stack

e)* Wo wird `b` übergeben?

☐ `a6`	☐ `a5`	☐ `a3`	☐ `a1`
☐ `a0`	☐ `a4`	☐ `a2`	☐ Stack

f)* Was passiert laut Calling Convention mit den restlichen Bits eines Registers, wenn der zu speichernde Datentyp kleiner als die Registergröße ist? 0 1
Beispiel: Ein 8-Bit großer `char` *soll im 32-Bit großen Register* `t0` *gespeichert werden.*

Aufgabe 3 Assoziative Caches (10 Punkte)

Gegeben sei ein System mit 4 Kernen, einem geteilten inklusiven L2-Cache und je einem L1-Cache pro Kern. Im Folgenden wird der gemeinsame L2-Cache betrachtet. Dieser Cache sei insgesamt 2 MiB groß und 16-fach assoziativ, wobei jede Cachezeile 64 B breit sei.
Die Architektur sei eine reine 32-Bit Architektur und alle Zugriffe und Speicheradressen 32-Bit breit.
Erinnerung: Machen Sie Ihren Rechenweg deutlich erkennbar!

a)* Erklären Sie kurz den Unterschied zwischen einer inklusiven und exklusiven Cache-Hierarchie. 0 1

b)* Wie viele Cache-Sets hat der Cache? 0 1 2

0 1 2

c) Wie viele Bits werden für Index, Offset und Tag benötigt? Der Cache soll wie immer Byte-adressierbar sein.

0 1

d) In welchem Cache-Set (ab 0 nummeriert) werden die Daten der Adresse `0x4FF` gecached?

0 1 2

e)* Erläutern Sie kurz, wann ein Conflict-Miss in einem mengenassoziativen Cache auftreten kann.

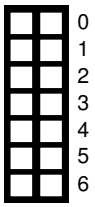
0 1 2

f)* Erläutern Sie kurz einen Vorteil sowie einen Nachteil der FIFO Ersetzungsstrategie.

Aufgabe 4 RISC-V Strings (18 Punkte)

Halten Sie sich für alle folgenden Aufgaben unbedingt an die vorgegebene Struktur und benutzen Sie die vorgegebenen Stellen nach den Kommentaren! Sie dürfen zusätzliche Labels einfügen.
Als standardisiertes Zeichenformat wird das ASCII-Format für Strings verwendet. Im Folgenden werden C-Strings verwendet.

a)* Schreiben Sie ein RISC-V Unterprogramm `starts_with`, welches feststellt, ob ein String mit einem bestimmten Substring beginnt. Dafür bekommt das Unterprogramm als erstes Argument einen Pointer auf den String und als zweites Argument einen Pointer auf den Substring übergeben. Das Programm soll 1 zurückgeben, falls der String mit dem Substring beginnt (insbesondere soll 1 zurückgegeben werden, wenn der zweite String leer ist). In allen anderen Fällen soll 0 zurückgegeben werden.



Beispiel: Angenommen, der erste String lautet „Hallo Welt“ und der zweite String „Hallo“. Dann soll das Unterprogramm 1 zurückgeben.

```
starts_with:
; Platz, falls Sie Register sichern wollen




; Laden der Buchstaben




; Testen, ob am Ende des zweiten Strings angekommen


; Testen, ob geladene Buchstaben gleich sind


; Pointer inkrementieren und Sprung um nächste Buchstaben zu laden




; Rückgabewerte schreiben
equal:




not_equal:




```

0 1 2 3 4 5 6 7 8 9 10

b)* Schreiben Sie ein RISC-V Unterprogramm `strstr`, welches feststellt, ob ein Substring in einem String enthalten ist. Auch dieses Programm bekommt als erstes Argument einen Pointer auf den String und als zweites Argument einen Pointer auf den Substring übergeben. Wenn der Substring enthalten ist, soll das Unterprogramm einen Pointer auf den Beginn des ersten Vorkommens des Substrings im String zurückgeben. Falls der Substring nicht enthalten ist, soll 0 zurückgegeben werden.

Verwenden Sie für diese Aufgabe das in der vorherigen Teilaufgabe geschriebene Unterprogramm `starts_with` und halten Sie sich an die durch die Kommentare gegebene Struktur.

Beispiel: Angenommen, der erste String lautet „Hallo Welt“ und der zweite String „Welt“. Dann soll das Unterprogramm die Adresse des Buchstabens „W“ im ersten String zurückgeben.

```
strstr:
; Sichern von Registern









; Schleife für Durchlaufen des ersten übergebenen Strings
loop:
; Test, ob am Ende









; Aufruf von starts_with









; Rückgabe von starts_with prüfen. Falls nicht fertig, weiter suchen
```

```
; Falls fertig, Adresse zurückgeben.
found:









; Rückgabe von 0, falls Substring nicht gefunden
not_found:
```

c)* Gegeben sei folgender Ausschnitt aus einem Programm, das die Funktion `strstr` mit der zuvor beschriebenen Funktionalität aufruft.

0
1
2

Erinnerung: `.org 0x400` *sorgt dafür, dass das erste Byte von substr an der Adresse* `0x400` *gespeichert wird. Zudem wird der zweite String direkt hinter dem ersten abgespeichert.*

```
1  prog:
2      la a0, str
3      la a1, substr
4      call strstr
5      ret
6
7
8  .data
9  .org 0x400
10
11 substr: .asciz "Welt"
12 str:    .asciz "Hallo Welt"
```

Geben Sie (ohne weitere Begründung) in hexadezimaler Schreibweise die Adresse an, die von `strstr` nach Ausführung von Zeile 4 zurückgegeben wird.

Aufgabe 5 Stack (10 Punkte)

Gegeben sei das folgende Programm in RISC-V Assembly mit den Speicheradressen der Instruktionen auf der linken Seite. Tipp: Machen Sie sich keine Gedanken bzgl. der Calling Convention.

```
        .text

        program:
0x200       la a0, a
0x204       addi sp, sp, -4
0x208       sw ra, 0(sp)

0x20c       jal ra, fun
0x210       lw ra, 0(sp)
0x214       addi sp, sp, 4
0x218       ret

        fun:
0x21c       lw t0, 0(a0)
0x220       beq t0, zero, end
0x224       addi a0, a0, 4
0x228       addi sp, sp, -8
0x22c       sw ra, 0(sp)
0x230       sw t0, 4(sp)
0x234       call fun ⊙
0x238       lw t0, 4(sp)
0x23c       mul a0, a0, t0
0x240       lw ra, 0(sp)
0x244       addi sp, sp, 8
0x248       ret

        end:
0x24c       addi a0, zero, 1
0x250       ret


        .data
        .org 0x400

        a:
0x400       .word  0x7, 0x1, 0x6, 0x0
```


0 1 2 3 4 5 6 7 8 9

a)* In dieser Aufgabe sollen Sie den Stack-Zustand bei Ausführung des Programms `program` nachvollziehen. Die mit ⊙ im Programm gekennzeichnete Zeile wird insgesamt 3 mal ausgeführt. Vervollständigen Sie den Stack-Zustand des Programms zu diesen 3 Zeitpunkten in den entsprechend markierten Tabellen. Schreiben Sie die Daten an der entsprechenden Adresse in **hexadezimaler** Notation. Falls die Daten unbekannt sind, markieren Sie die Zeile explizit mit einem „?“. Schreiben Sie auch die Daten in die Stacks, die sich im Vergleich zum vorherigen Ausschnitt nicht ändern.
Zeichnen Sie außerdem den Stackpointer zu diesen Zeitpunkten als Pfeil auf die entsprechende Zeile der jeweiligen Tabelle ein.

Anmerkung: Der Stackpointer zeigt zu Beginn des Programms auf die Zelle, die mit Bottom markiert ist. Der Stack soll jeweils den Zustand <u>nach</u> der ausgeführten Zeile darstellen, aber vor dem nächsten Befehl.
*Hinweis: Sie finden bei dieser Aufgabe mehrere Vordrucke für Ihre Lösung. Sollten Sie davon mehr als einen verwenden, kennzeichnen Sie **deutlich** (z.B. durch Durchstreichen der anderen Vordrucke), welcher zur Bewertung herangezogen werden soll.*

(1)

Adressen	**Werte**
0x7fffff00	*Bottom*
0x7ffffefc	
0x7ffffef8	
0x7ffffef4	
0x7ffffef0	
0x7ffffeec	
0x7ffffee8	
0x7ffffee4	
0x7ffffee0	

(2)

Adressen	**Werte**
0x7fffff00	*Bottom*
0x7ffffefc	
0x7ffffef8	
0x7ffffef4	
0x7ffffef0	
0x7ffffeec	
0x7ffffee8	
0x7ffffee4	
0x7ffffee0	

(3)

Adressen	**Werte**
0x7fffff00	*Bottom*
0x7ffffefc	
0x7ffffef8	
0x7ffffef4	
0x7ffffef0	
0x7ffffeec	
0x7ffffee8	
0x7ffffee4	
0x7ffffee0	

(1)

Adressen	**Werte**
0x7fffff00	*Bottom*
0x7ffffefc	
0x7ffffef8	
0x7ffffef4	
0x7ffffef0	
0x7ffffeec	
0x7ffffee8	
0x7ffffee4	
0x7ffffee0	

(2)

Adressen	**Werte**
0x7fffff00	*Bottom*
0x7ffffefc	
0x7ffffef8	
0x7ffffef4	
0x7ffffef0	
0x7ffffeec	
0x7ffffee8	
0x7ffffee4	
0x7ffffee0	

(3)

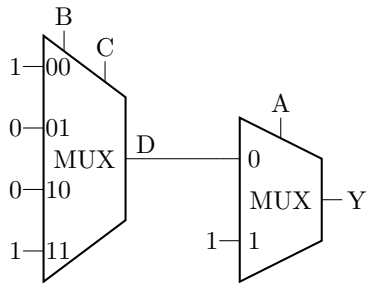
Adressen	**Werte**
0x7fffff00	*Bottom*
0x7ffffefc	
0x7ffffef8	
0x7ffffef4	
0x7ffffef0	
0x7ffffeec	
0x7ffffee8	
0x7ffffee4	
0x7ffffee0	

b)* Welcher Wert wird vom Programm `program` zurückgegeben? (ohne Erklärung)

0
1

Aufgabe 6 Digitale Schaltungen und Logiksynthese (20 Punkte)

Gegeben sei der folgende Schaltkreis mit 1bit-Eingängen *A*, *B*, und *C* sowie 1bit-Ausgang Y.
Hinweis: *Beim linken Multiplexer (mit Ausgang D) steuert B das linke und C das rechte Bit.*



0 1 2 3 4 5

a)* Prüfen oder widerlegen Sie die folgende Aussage mittels einer Wahrheitstabelle:

$$Y \equiv (\neg A \wedge B \wedge \neg C) \,\overline{\vee}\, (\neg A \wedge \neg B \wedge C)$$

wobei *Y* das Ausgabesignal des obigen Schaltkreises sei und $\overline{\vee}$ den NOR-Operator darstellt. Geben Sie dabei auch Ergebnisse für Zwischenschritte an und schlussfolgern Sie, ob die Aussage wahr oder falsch ist.
Hinweis: Sie finden bei dieser Aufgabe mehrere Vordrucke für Ihre Lösung. Sollten Sie davon mehr als einen verwenden, kennzeichnen Sie ***deutlich*** *(z.B. durch Durchstreichen der anderen Vordrucke), welcher zur Bewertung herangezogen werden soll.*

A	B	C	D	Y	$\neg A \wedge B \wedge \neg C$	$\neg A \wedge \neg B \wedge C$	$(\neg A \wedge B \wedge \neg C) \,\overline{\vee}\, (\neg A \wedge \neg B \wedge C)$
0	0	0					
0	0	1					
0	1	0					
0	1	1					
1	0	0					
1	0	1					
1	1	0					
1	1	1					

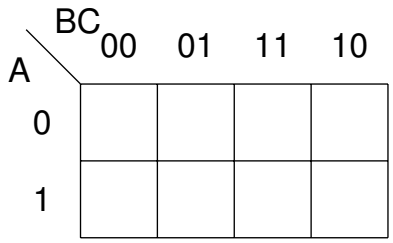
A	B	C	D	Y	$\neg A \wedge B \wedge \neg C$	$\neg A \wedge \neg B \wedge C$	$(\neg A \wedge B \wedge \neg C) \,\overline{\vee}\, (\neg A \wedge \neg B \wedge C)$
0	0	0					
0	0	1					
0	1	0					
0	1	1					
1	0	0					
1	0	1					
1	1	0					
1	1	1					

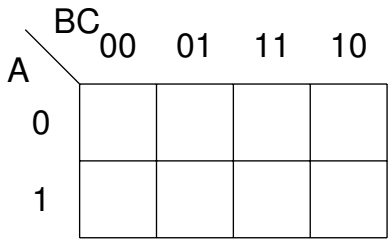
Schlussfolgerung:

b) Bestimmen Sie ein Minimalpolynom für *Y* (Ausgang des Schaltbildes) mithilfe einer K-Map. Zeichnen Sie die zugehörigen Primblöcke in der K-Map ein.

0 1 2 3

Hinweis: Sie finden bei dieser Aufgabe mehrere Vordrucke für Ihre Lösung. Sollten Sie davon mehr als einen verwenden, kennzeichnen Sie ***deutlich*** *(z.B. durch Durchstreichen der anderen Vordrucke), welcher zur Bewertung herangezogen werden soll.*





c) Überführen Sie das Minimalpolynom aus Teilaufgabe b) in eine Schaltung. Ist Ihre Schaltung gegen Logik-Hazards abgesichert? Falls ja, weshalb? Falls nein, sichern Sie das Schaltnetz gegen Logik-Hazards ab.

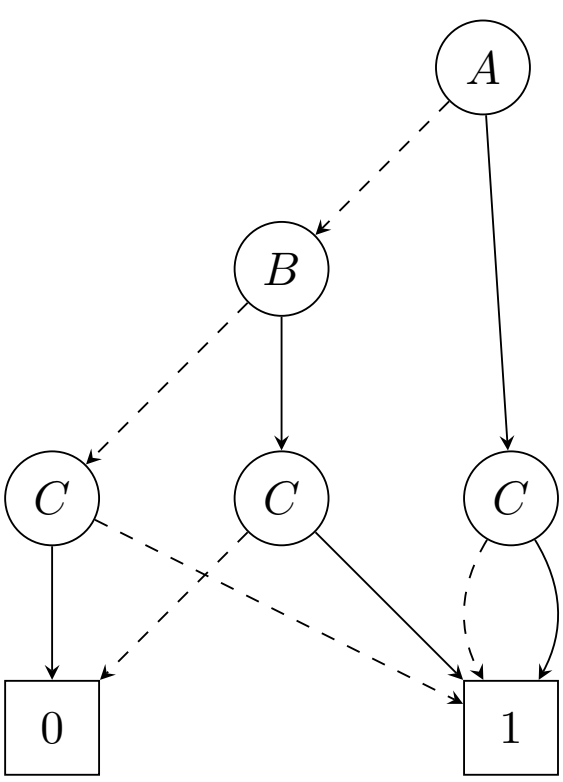
0 1 2 3

0 1 2 3

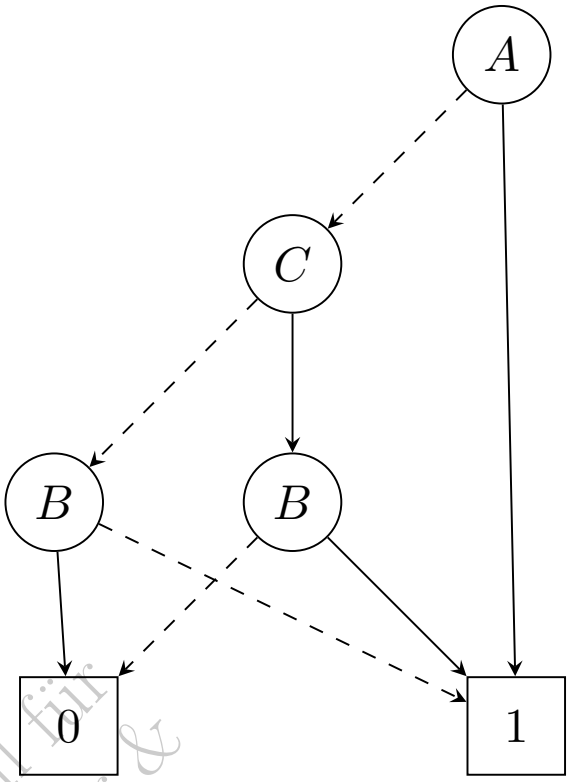
d)* Bestimmen Sie (ohne Begründung) einen Ausdruck für Y (Ausgang des Schaltbildes) ausschließlich unter Verwendung von ITE sowie Konstanten und Literalen. Zeichnen Sie davon ausgehend (ohne weitere Begründung) das ROBDD mit der Variablenordnung $A \prec B \prec C$, welches die Funktionalität von Y realisiert.

0 1 2

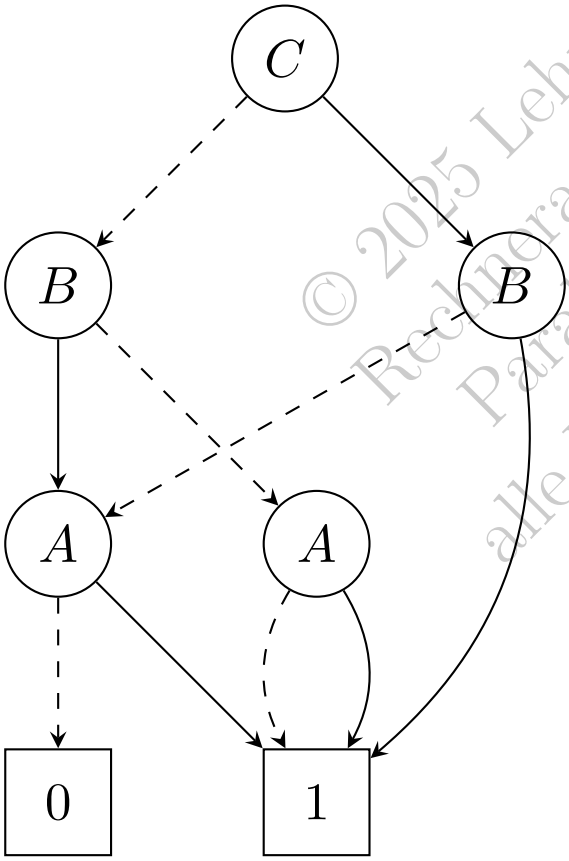
e) In Abbildung 6.1 sind vier BDDs für den Ausdruck $Z := \neg(B \oplus C \oplus (A \wedge B) \oplus (A \wedge C))$ gegeben. Prüfen Sie mit Hilfe dieser Informationen, ob $Y \equiv Z$ gilt. Geben Sie dabei unbedingt explizit an, welche konkreten BDDs Sie in Ihrer Argumentation verwenden.



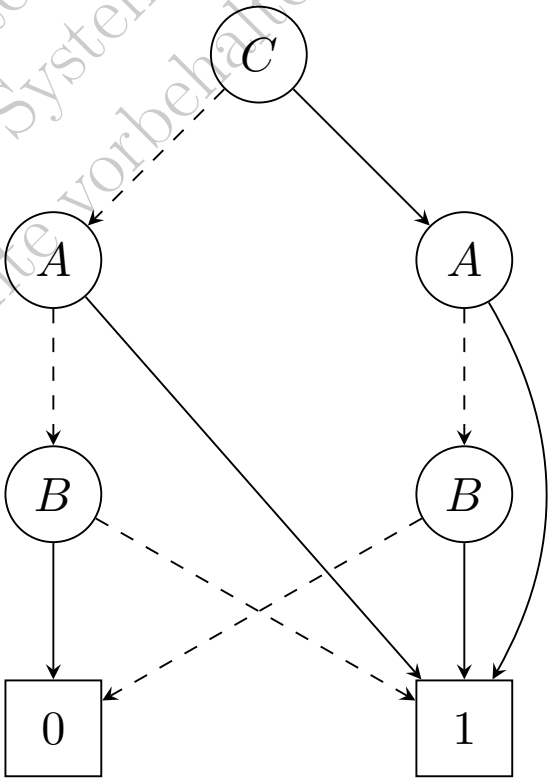
(a) BDD 1 für Z



(b) BDD 2 für Z



(c) BDD 3 für Z



(d) BDD 4 für Z

Abbildung 6.1: Verschiedene BDDs, die jeweils Z realisieren

Kreuzen Sie richtige Antworten an ☒
Kreuze können durch vollständiges Ausfüllen gestrichen werden ■
Gestrichene Antworten können durch nebenstehende Markierung erneut angekreuzt werden ×■

f)* Geben Sie den zu $(x\overline{y} + 0) \cdot z$ dualen Ausdruck an.

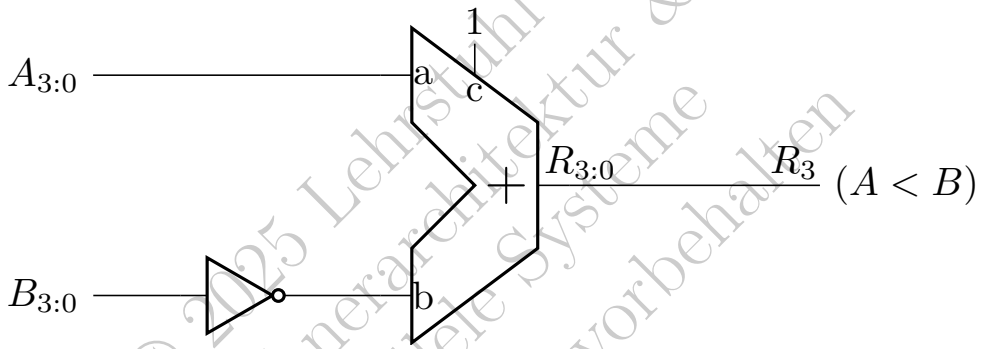
☐ $((x + \overline{y}) \cdot 1) + z$ ☐ $((\overline{x} + y) \cdot 1) + \overline{z}$ ☐ $((x + \overline{y}) \cdot 0) + z$

☐ $(\overline{x}y + 0) \cdot \overline{z}$ ☐ $(\overline{x}y + 1) \cdot \overline{z}$ ☐ $((\overline{x} + y) \cdot 0) + \overline{z}$

g)* Welche der folgenden Operatormengen ist/sind funktional vollständig?
Hinweis: Beachten Sie, dass die Verwendung der Konstanten 0 und 1 in der Definition der funktionalen Vollständigkeit ausgeschlossen wird. Für die funktionale Vollständigkeit müssen daher auch Konstanten mit Hilfe der entsprechend Operatoren (und Literalen) darstellbar sein, z.B., $0 = x \wedge \neg x$ *und* $1 = x \vee \neg x$.

☐ {Implikation} ☐ {NAND} ☐ {XNOR} ☐ {XOR}

h)* *(2 Punkte)* Gegeben seien zwei *vorzeichenbehaftete* (signed) 4bit-Binärzahlen (A und B) im Zweierkomplement. Betrachten Sie nun folgenden Komparator:



bestehend aus zwei 4bit-Eingängen A und B, einem NICHT-Gatter, einem 4bit-Addierer, sowie einem 1bit-Ausgang ($A < B$), der genau dann 1 sein soll, wenn A kleiner als B ist. Für welche der folgenden Belegungen (Angaben im Dezimalsystem) ist das an ($A < B$) anliegende Signal *fehlerhaft*.

☐ $A = -4$, $B = -6$: ($A < B$) fälschlicherweise 1.

☐ $A = -2$, $B = 7$: ($A < B$) fälschlicherweise 0.

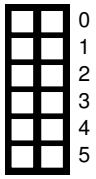
☐ $A = 3$, $B = 7$: ($A < B$) fälschlicherweise 0.

☐ $A = -4$, $B = 4$: ($A < B$) fälschlicherweise 0.

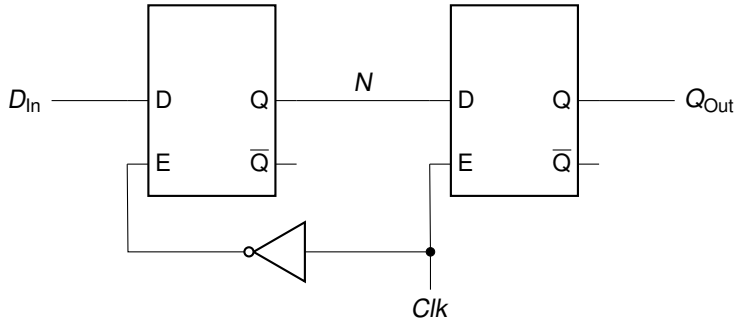
☐ $A = 3$, $B = -5$: ($A < B$) fälschlicherweise 1.

☐ Es gibt keine fehlerhaften Belegungen. ($A < B$) ist für *alle* Eingangsbelegungen korrekt.

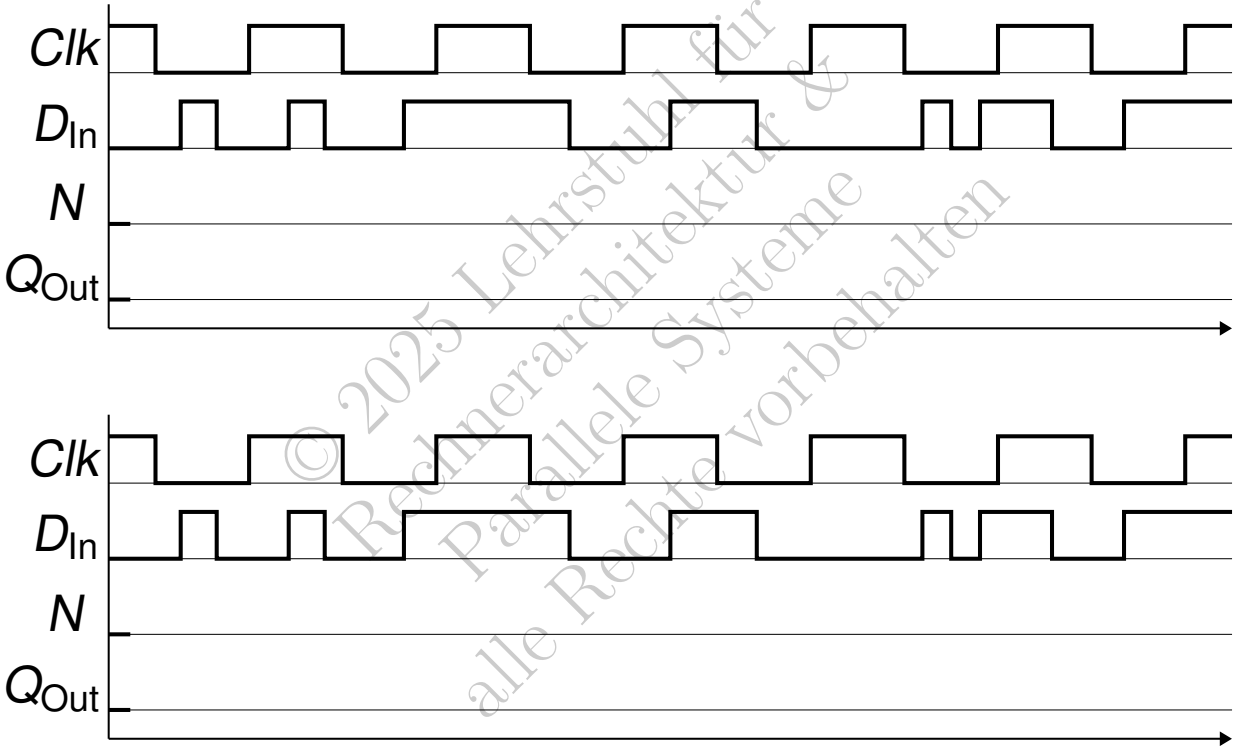
Aufgabe 7 Sequentielle Schaltungen (5 Punkte)



Betrachten Sie den folgenden D-FlipFlop, der aus zwei D-Latches besteht. Die D-Latches sind transparent, wenn das Enable-Signal aktiv ist ($E = 1$).



Vervollständigen Sie die untenstehenden Wellenformen für diese Schaltung.
*Hinweis: Sie finden bei dieser Aufgabe mehrere Vordrucke für Ihre Lösung. Sollten Sie davon mehr als einen verwenden, kennzeichnen Sie **deutlich** (z.B. durch Durchstreichen der anderen Vordrucke), welcher zur Bewertung herangezogen werden soll.*



Aufgabe 8 Single-Cycle RISC-V Prozessor (11 Punkte)

Betrachten Sie das Schaltbild des Single-Cycle RISC-V Prozessors in Abbildung 8.1.
Der Instruktionssatz des Prozessor soll um den Befehl `bltu` (branch if less than unsigned) erweitert werden. Der Befehl `bltu rs1, rs2, imm` vergleicht zwei Register `rs1`, `rs2` und führt einen Sprung aus falls `rs1 < rs2`. Die Sprungadresse berechnet sich aus der Summe des PC und dem sign-extended Immediate-Wert `PC + signext(imm)`.

0 1 2 3 4 5 6 7 8 9

a)* Erweitern Sie den Datenpfad des Prozessors mit möglichst wenig Logik, sodass der Befehl `bltu` unterstützt wird. Die Ein- und Ausgangssignale der Control Unit dürfen *nicht* verändert werden. Tragen Sie auch die Werte aller Kontrollsignale des Steuerwerks an die entsprechenden Signale im Schaltbild ein. Geben Sie *don't cares* explizit mit X an. Nehmen Sie weiterhin an, dass das Extend-Bauteil wie folgt gesteuert wird:

Befehlstyp	`ImmSrc`
I	00
S	01
B	10
J	11

Im Folgenden hat die ALU die Outputs `ALUResult` und `Zero`. Fügen Sie keine weiteren Flags zum Output hinzu. Nehmen Sie an, dass die ALU die folgenden Befehle unterstützt:

`ALUControl`	Instruction
000 (Add)	`ALUResult = SrcA + SrcB`
001 (Sub)	`ALUResult = SrcA - SrcB`
101 (Set Less Than)	`ALUResult = 1 if SrcA < SrcB else ALUResult = 0`
110 (Set Less Than Unsigned)	`ALUResult = 1 if SrcA < SrcB else ALUResult = 0`
011 (Or)	`ALUResult = SrcA` $\vee$ `SrcB` (Bitweises Oder)
010 (And)	`ALUResult = SrcA` $\wedge$ `SrcB` (Bitweises Und)

Hinweis: *Der Prozessor unterstützt bisher die Befehle* `lw`, `sw`, `R-type`, `I-type ALU`, `jal` *und* `beq`. *Stellen Sie sicher, dass diese nach die Erweiterung weiterhin korrekt funktionieren.*
Hinweis: Sie finden bei dieser Aufgabe mehrere Vordrucke für Ihre Lösung. Sollten Sie davon mehr als einen verwenden, kennzeichnen Sie ***deutlich*** *(z.B. durch Durchstreichen der anderen Vordrucke), welcher zur Bewertung herangezogen werden soll.*

b)* *(2 Punkte)* Die ALU wird nun erweitert und gibt neben dem `ALUResult` und dem Zero-Flag zusätzlich die Flags Negative, Overflow und Carry aus.
Um den Vergleich `rs1 < rs2` für den Befehl `bltu` korrekt durchzuführen, führt die ALU die Subtraktion `rs1 - rs2` aus, d.h., `ALUControl` wird mit 001 (Sub) belegt. Welche der folgenden Flags sind erforderlich, um diesen Vergleich eindeutig zu bestimmen?

Kreuzen Sie richtige Antworten an ☒
Kreuze können durch vollständiges Ausfüllen gestrichen werden ■
Gestrichene Antworten können durch nebenstehende Markierung erneut angekreuzt werden ×■

- ☐ Carry (C): Dieses Flag zeigt an, ob bei unsigned-Arithmetik ein Übertrag auftritt.
- ☐ Negative (N): Dieses Flag zeigt an, ob das Ergebnis negativ ist.
- ☐ Overflow (V): Dieses Flag zeigt an, ob ein Überlauf bei Darstellung im Zweierkomplement auftritt.
- ☐ Zero (Z): Dieses Flag zeigt an, ob das Ergebnis der Subtraktion gleich null ist.

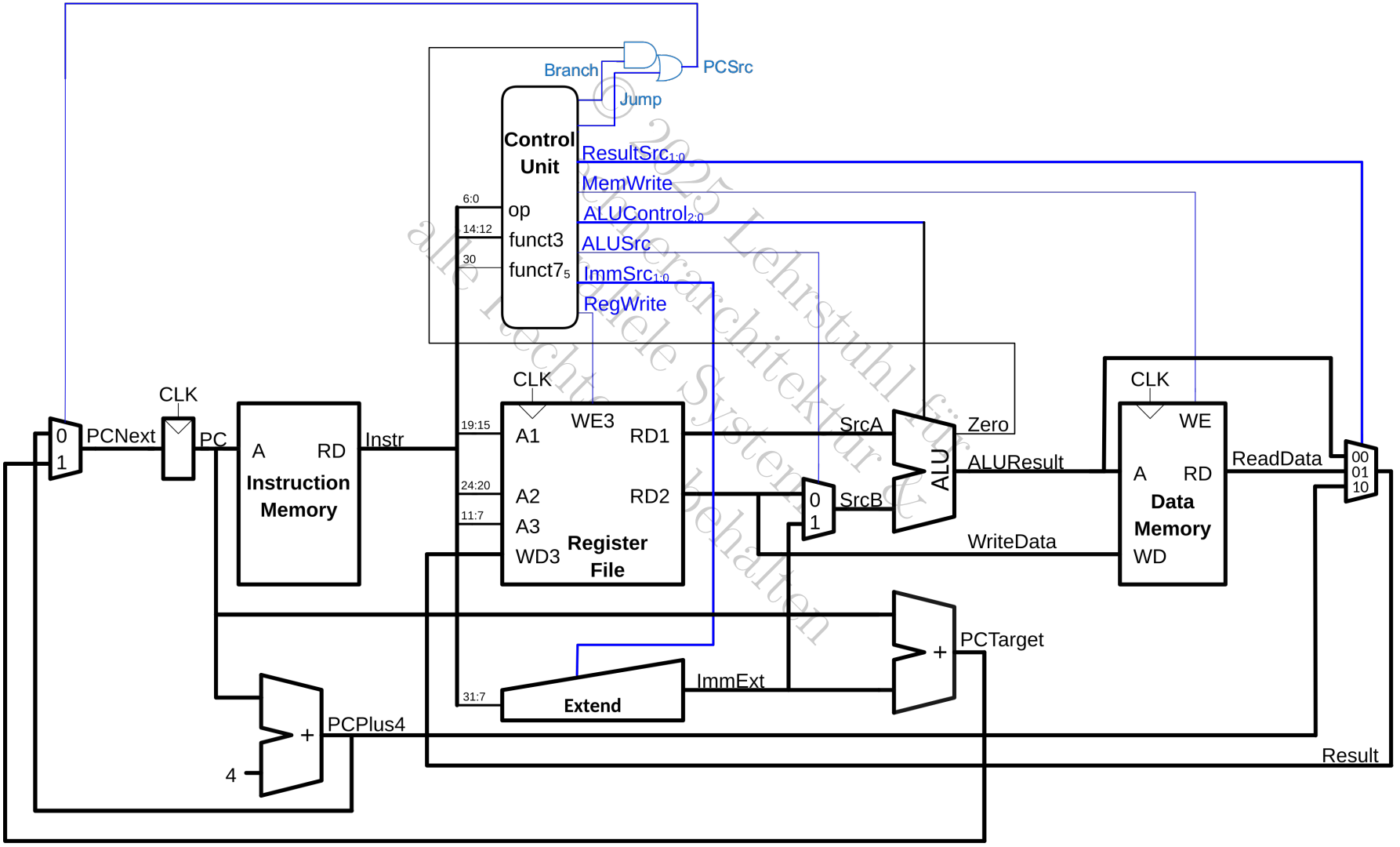


Abbildung 8.1: Schaltbild des Single-Cycle RISC-V Prozessors

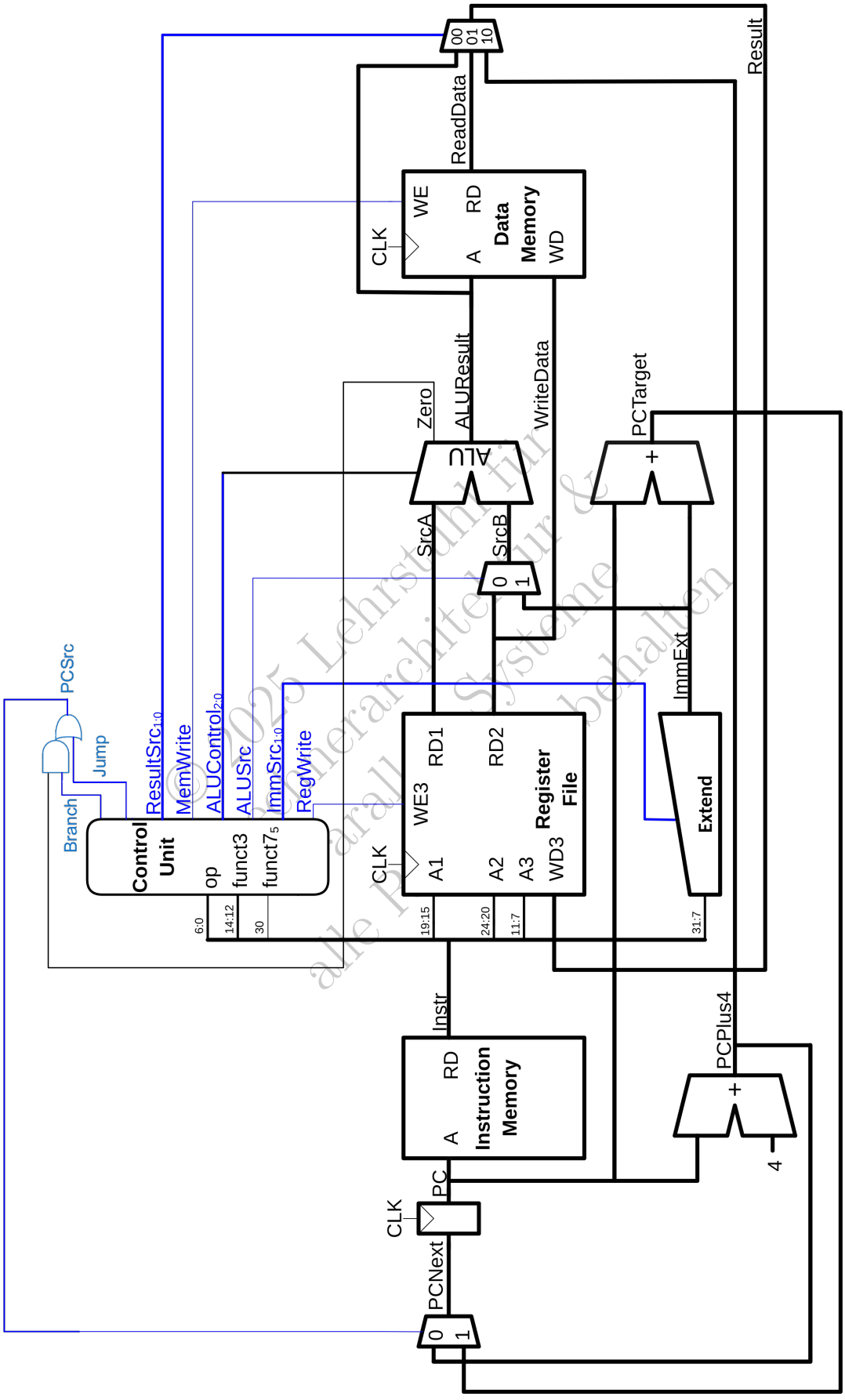
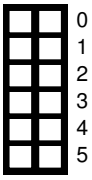
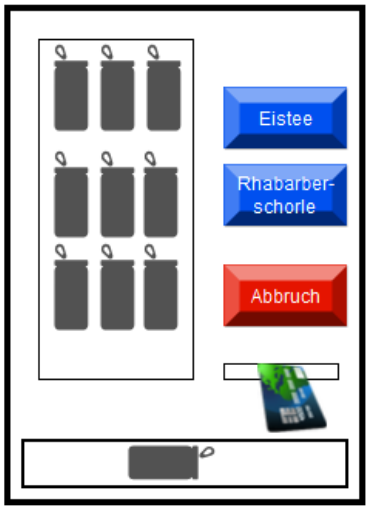


Abbildung 8.2: Kopie des Schaltbilds des Single-Cycle RISC-V Prozessors

Aufgabe 9 Endliche Automaten und Multi-Cycle RISC-V (11 Punkte)



a)* Ein neuer Getränkeautomat wird von der ERA-Übungsleitung im MI aufgestellt. Dieser verkauft zwei Arten von Getränken: Eistee und Rhabarberschorle. Um ein Getränk zu erwerben muss ein(e) KundIn zunächst seine/ihre Student-Card einführen (Kartenleser). Danach kann er/sie die Taste mit der entsprechenden Bezeichnung drücken, um das Getränk zu erhalten. Nach Bereitstellung des Getränks wird ebenfalls die Student-Card ausgegeben. Alternativ kann ein(e) KundIn nach Einführen der Student-Card durch Betätigen der roten Abbruchtaste die Student-Card zurückerhalten, ohne eine Transaktion durchzuführen.



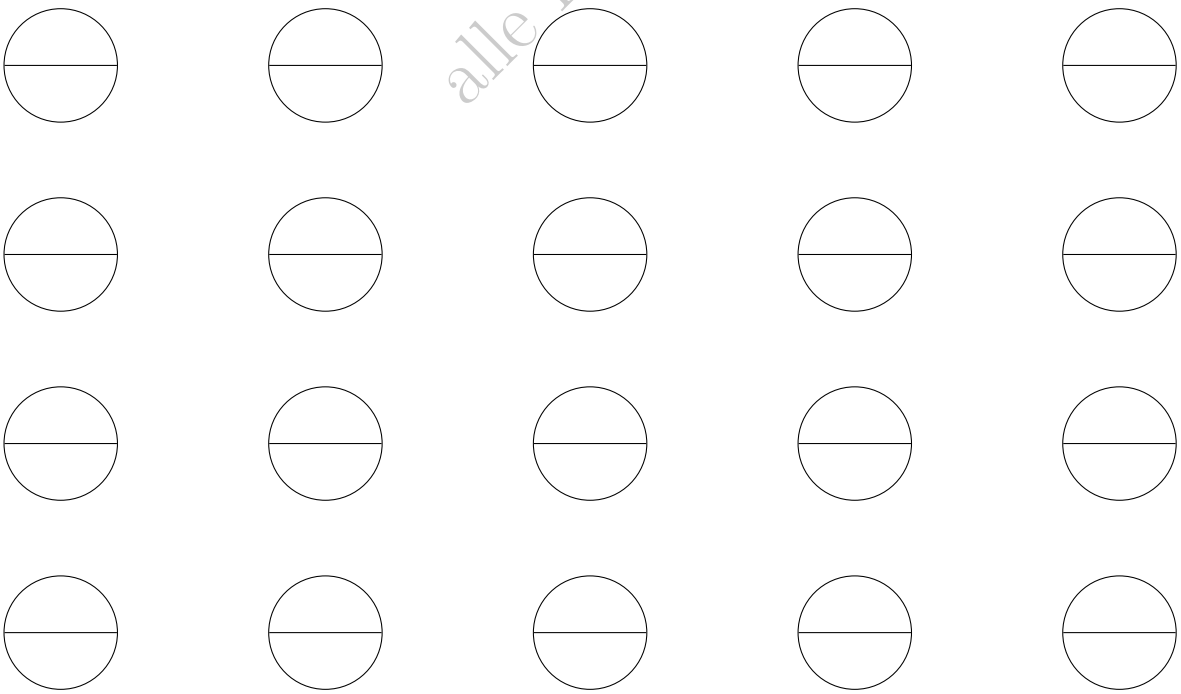
Entwerfen Sie einen Moore-Automaten für die Steuerung des Getränkeautomaten. Folgende Eingangssignale stehen Ihnen zur Verfügung:

- *E* (1bit): 1 wenn Taste „Eistee“ gedrückt wird.
- *R* (1bit): 1 wenn Taste „Rhabarberschorle“ gedrückt wird.
- *A* (1bit): 1 wenn Taste „Abbruch“ gedrückt wird.
- *SC* (1bit): 1 wenn eine Student-Card eingeführt wird.

Es darf davon ausgegangen werden, dass jeweils maximal eines der Signale *E*, *R* und *A* gleichzeitig 1 ist. Der Moore-Automat soll ein 2bit-Ausgabesignal (G) steuern

- *G* = 0: Keine Aktion
- *G* = 1: Ausgabe eines Eistees
- *G* = 2: Ausgabe einer Rhabarberschorle
- *G* = 3: Ausgabe der Student-Card

Es darf davon ausgegangen werden, dass eine weitere Student-Card erst eingeführt werden kann, wenn die Vorherige bereits vollständig ausgegeben wurde. Außerdem kann pro Kaufvorgang maximal ein Getränk erworben werden.
Vervollständigen Sie untenstehendes Zustandsdiagramm eines Moore-Automaten für die Steuerung des oben beschriebenen Getränkeautomatens.
Hinweis: *Die Anzahl der Kreise muss nicht der der Automatenzustände entsprechen. Wenn ein Übergang nicht beschriftet ist, so wird dies als „in allen anderen Fällen“ interpretiert.*



0 1

b) Sie möchten den in Teilaufgabe a) entwickelten Automaten nun in einen Schaltkreis überführen. Wie viele 1bit-FlipFlops müssen Sie verwenden, wenn Sie die

- One-Hot Kodierung verwenden: ☐
- Binärkodierung verwenden: ☐

Schreiben Sie Ihre Lösung (ohne Begründung) direkt in die jeweilige Box.

0 1 2 3

c)* Betrachten Sie das Schaltbild des Multi-Cycle RISC-V Prozessors in Abbildung 9.1 und den endlichen Automaten des sequentiellen Steuerwerks in Abbildung 9.2, sowie folgendes Programm:

```
start:
    addi s0, zero, 0  # i = 0
    lw   s1, 20(zero) # sum = memory[20]
    addi t3, zero, 3  # t3 = 3
loop:
    beq  s0, t3, done # if i == 3, goto done
    add  s1, s1, s0   # sum = sum + i
    addi s0, s0, 1    # i = i + 1
    jal  zero, loop   # goto loop
done:
    sw   s1, 20(zero) # memory[20] = sum
```

Wie viele Zyklen werden benötigt, um das o.g. Programm auf einem Multi-Cycle RISC-V Prozessor auszuführen? ***Erinnerung:*** *Zeigen Sie dabei Ihren Rechenweg.*

0 1 2

d)* Betrachten Sie das Schaltbild des Multi-Cycle RISC-V Prozessors in Abbildung 9.1 und den endlichen Automaten des sequentiellen Steuerwerks in Abbildung 9.2. Angenommen, der Prozessor hat ein fehlerhaftes Kontrollsignal, wodurch ALUSrcA konstant 01 ist.
Begründen Sie, weshalb die Instruktion `jal` dennoch korrekt ausgeführt wird.

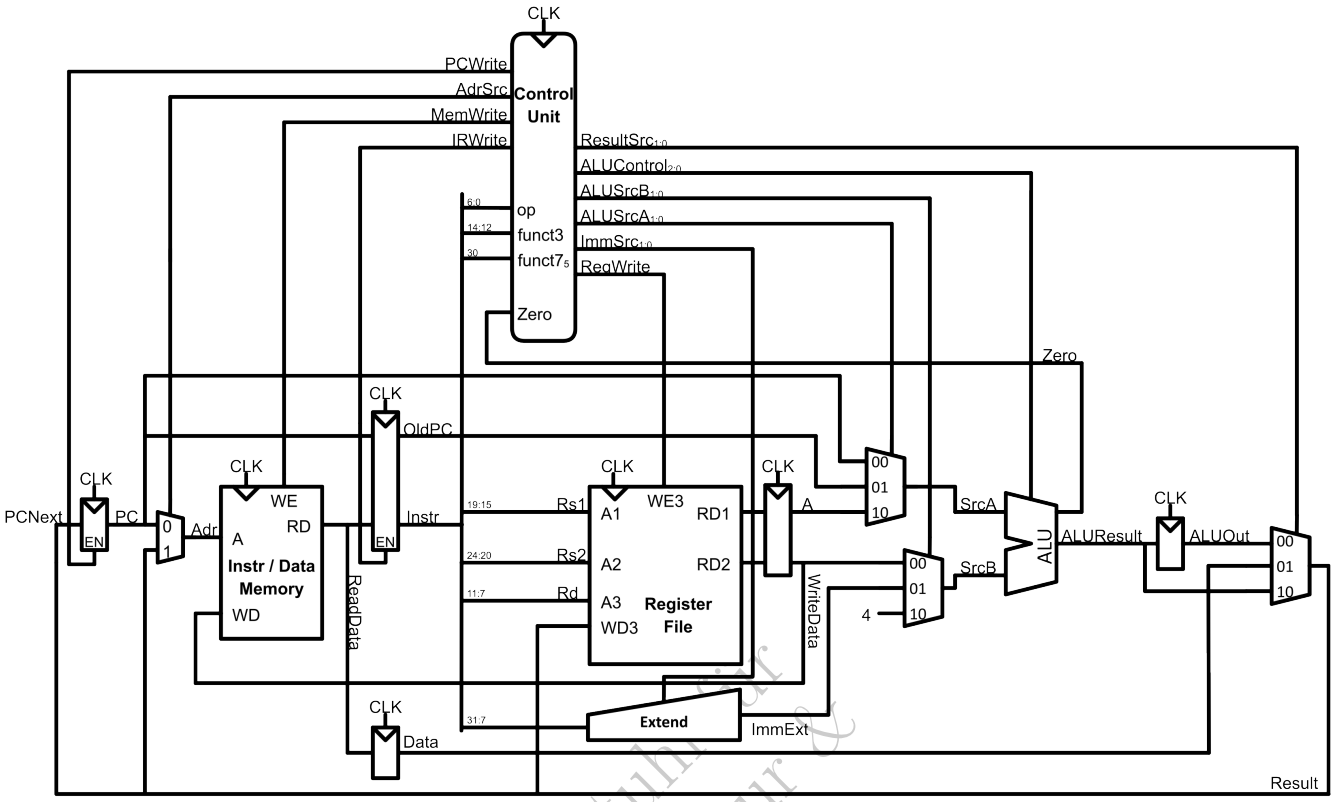
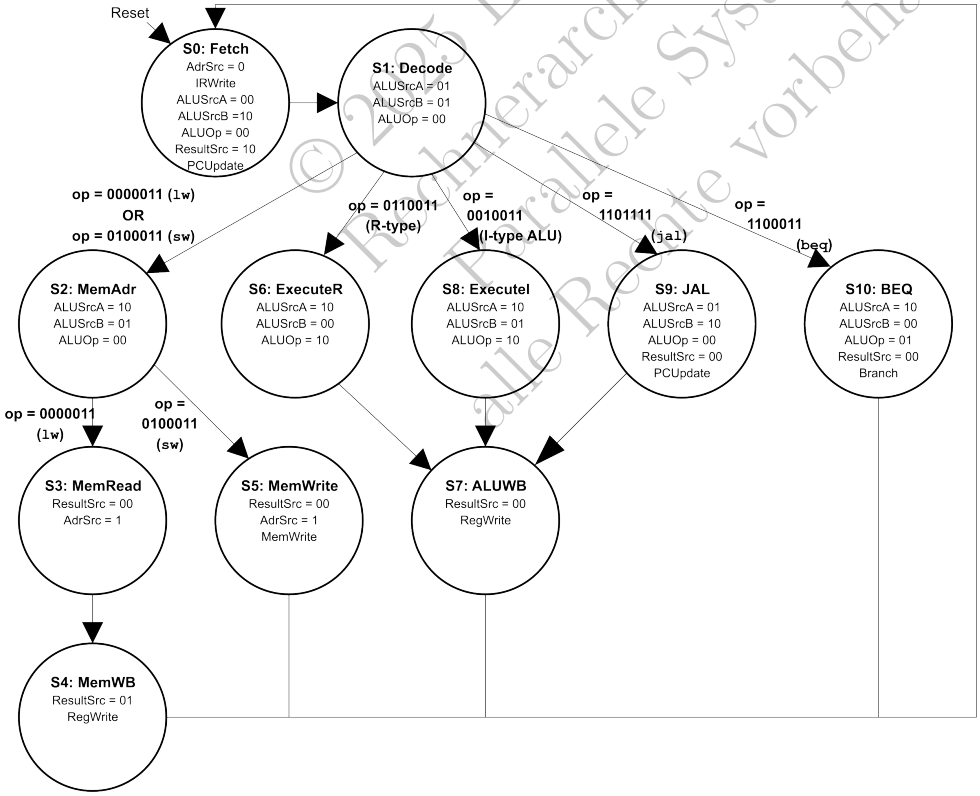
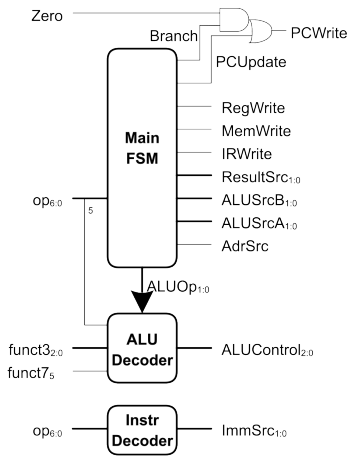


Abbildung 9.1: Schaltbild des Multi-Cycle RISC-V Prozzessors



(a) Endlicher Automat der Main FSM



(b) Low-Level Control Unit

Abbildung 9.2: Steuerwerk des Multi-Cycle RISC-V Prozessors

Aufgabe 10 **Pipelinekonflikte (8 Punkte)**

Gegeben ist der RISC-V Prozessor mit fünfstufiger Pipeline (*fetch, decode, execute, memory, writeback*) ohne Hazard-Unit aus Abbildung 10.1. Die Registerbank wird bei steigender Taktflanke geschrieben.
Auf dem Prozessor wird folgendes Programmfragment ausgeführt:

```
l1:     sub  t3, t2, t4
l2:     ori  t4, t1, 2
l3:     add  t2, t3, t1
l4:     addi t1, t4, 4
l5:     and  t3, t2, t1
l6:     sw   t5, 4(t4)
```

0 1 2 3 4

a)* Geben Sie alle Datenabhängigkeiten im Programm an, die den Befehl **`l5`** betreffen und klassifizieren Sie diese entsprechend. Nutzen Sie dafür die folgende Tabelle.

Art der Abhängigkeit	Betroffene Befehle	Betroffenes Register

0 1 2 3 4

b)* Finden und erläutern Sie *alle* Pipeline-Konflikte des Programms. In Takt 1 wird die Instruktion `l1` geladen.

Beispiel zum Ausfüllen der Tabelle (ohne logischen Zusammenhang zur Aufgabe): Im Takt 8 tritt ein Data Hazard auf, da t4 in `l5` *gelesen wird, aber t4 noch nicht von* `l3` *geschrieben wurde.*

Takt	Betroffene Befehle	Grund des Konflikts
8	`l5, l3`	RAW t4

Tragen Sie Ihre Lösung in die untenstehende Tabelle ein. Nur diese Tabelle wird für die Bewertung der Aufgabe berücksichtigt.

Takt	Betroffene Befehle	Grund des Konflikts

Hinweis: Auf der nächsten Seite finden Sie eine Tabelle zur Hilfestellung bei der Lösung dieser Aufgabe (**diese fließt nicht in die Benotung mit ein**).

Hinweis: Die folgende Tabelle kann bei der Bearbeitung der Aufgabe zu Hilfe gezogen werden (**sie fließt nicht in die Benotung mit ein**).

Takt	F	D	E	M	W
1	11				
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					

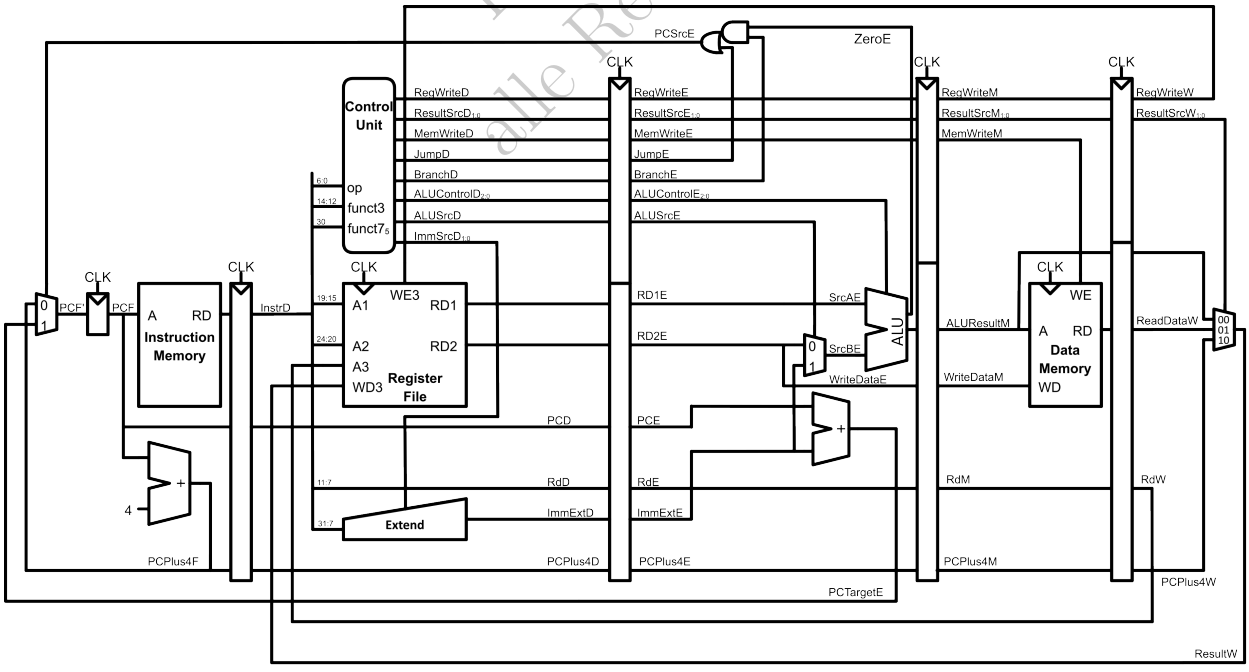
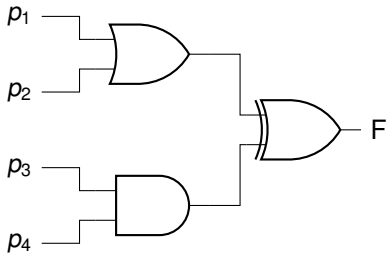


Abbildung 10.1: Schaltbild des pipelined RISCV Prozessors

Aufgabe 11 Verifikation mit SAT (5 Punkte)

Die folgende Schaltung kann durch die Formel $F := (p_1 \vee p_2) \oplus (p_3 \wedge p_4)$ beschrieben werden und dient als Eingabe für einen SAT-Solver:



0 1 2

a)* Führen Sie für die gegebene Formel den ersten Schritt der Tseitin Transformation durch, indem Sie für jedes Gatter eine Hilfsvariable einführen. Zeichnen Sie ihre Benennung in die Schaltung ein. Geben Sie Ihre Lösung als Konjunktion von Termen an. Die einzelnen Terme dürfen beliebige Operatoren beinhalten und sollen noch nicht in KNF überführt werden.

0 1 2 3

b) Wandeln Sie den Term, der das UND-Gatter beschreibt in die konjunktive Normalform (KNF) um und dokumentieren Sie alle Zwischenschritte.

31:25	24:20	19:15	14:12	11:7	6:0	
funct7	rs2	rs1	funct3	rd	op	R-Type
$imm_{11:0}$		rs1	funct3	rd	op	I-Type
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op	S-Type
$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op	B-Type
$imm_{31:12}$				rd	op	U-Type
$imm_{20,10:1,11,19:12}$				rd	op	J-Type
fs3	funct2	fs2	fs1	funct3	fd	op	R4-Type
5 bits	2 bits	5 bits	5 bits	3 bits	5 bits	7 bits	

- `imm`: signed immediate in $\mathbf{imm}_{11:0}$
- `uimm`: 5-bit unsigned immediate in $\mathbf{imm}_{4:0}$
- `upimm`: 20 upper bits of a 32-bit immediate, in $\mathbf{imm}_{31:12}$
- `Address`: memory address: **rs1** + SignExt($\mathbf{imm}_{11:0}$)
- `[Address]`: data at memory location Address
- `BTA`: branch target address: PC + SignExt({$\mathbf{imm}_{12:1}$, 1'b0})
- `JTA`: jump target address: PC + SignExt({$\mathbf{imm}_{20:1}$, 1'b0})
- `label`: text indicating instruction address
- `SignExt`: value sign-extended to 32 bits
- `ZeroExt`: value zero-extended to 32 bits
- `csr`: control and status register

op	funct3	funct7	Type	Instruction	Description	Operation
0000011 (3)	000	–	I	`lb rd, imm(rs1)`	load byte	`rd = SignExt([Address]7:0)`
0000011 (3)	001	–	I	`lh rd, imm(rs1)`	load half	`rd = SignExt([Address]15:0)`
0000011 (3)	010	–	I	`lw rd, imm(rs1)`	load word	`rd = [Address]31:0`
0000011 (3)	100	–	I	`lbu rd, imm(rs1)`	load byte unsigned	`rd = ZeroExt([Address]7:0)`
0000011 (3)	101	–	I	`lhu rd, imm(rs1)`	load half unsigned	`rd = ZeroExt([Address]15:0)`
0010011 (19)	000	–	I	`addi rd, rs1, imm`	add immediate	`rd = rs1 + SignExt(imm)`
0010011 (19)	001	0000000*	I	`slli rd, rs1, uimm`	shift left logical immediate	`rd = rs1 << uimm`
0010011 (19)	010	–	I	`slti rd, rs1, imm`	set less than immediate	`rd = (rs1 < SignExt(imm))`
0010011 (19)	011	–	I	`sltiu rd, rs1, imm`	set less than imm. unsigned	`rd = (rs1 < SignExt(imm))`
0010011 (19)	100	–	I	`xori rd, rs1, imm`	xor immediate	`rd = rs1 ^ SignExt(imm)`
0010011 (19)	101	0000000*	I	`srli rd, rs1, uimm`	shift right logical immediate	`rd = rs1 >> uimm`
0010011 (19)	101	0100000*	I	`srai rd, rs1, uimm`	shift right arithmetic imm.	`rd = rs1 >>> uimm`
0010011 (19)	110	–	I	`ori rd, rs1, imm`	or immediate	`rd = rs1 | SignExt(imm)`
0010011 (19)	111	–	I	`andi rd, rs1, imm`	and immediate	`rd = rs1 & SignExt(imm)`
0010111 (23)	–	–	U	`auipc rd, upimm`	add upper immediate to PC	`rd = {upimm, 12'b0} + PC`
0100011 (35)	000	–	S	`sb rs2, imm(rs1)`	store byte	`[Address]7:0 = rs27:0`
0100011 (35)	001	–	S	`sh rs2, imm(rs1)`	store half	`[Address]15:0 = rs215:0`
0100011 (35)	010	–	S	`sw rs2, imm(rs1)`	store word	`[Address]31:0 = rs2`
0110011 (51)	000	0000000	R	`add rd, rs1, rs2`	add	`rd = rs1 + rs2`
0110011 (51)	000	0100000	R	`sub rd, rs1, rs2`	sub	`rd = rs1 – rs2`
0110011 (51)	001	0000000	R	`sll rd, rs1, rs2`	shift left logical	`rd = rs1 << rs24:0`
0110011 (51)	010	0000000	R	`slt rd, rs1, rs2`	set less than	`rd = (rs1 < rs2)`
0110011 (51)	011	0000000	R	`sltu rd, rs1, rs2`	set less than unsigned	`rd = (rs1 < rs2)`
0110011 (51)	100	0000000	R	`xor rd, rs1, rs2`	xor	`rd = rs1 ^ rs2`
0110011 (51)	101	0000000	R	`srl rd, rs1, rs2`	shift right logical	`rd = rs1 >> rs24:0`
0110011 (51)	101	0100000	R	`sra rd, rs1, rs2`	shift right arithmetic	`rd = rs1 >>> rs24:0`
0110011 (51)	110	0000000	R	`or rd, rs1, rs2`	or	`rd = rs1 | rs2`
0110011 (51)	111	0000000	R	`and rd, rs1, rs2`	and	`rd = rs1 & rs2`
0110111 (55)	–	–	U	`lui rd, upimm`	load upper immediate	`rd = {upimm, 12'b0}`
1100011 (99)	000	–	B	`beq rs1, rs2, label`	branch if =	`if (rs1 == rs2) PC = BTA`
1100011 (99)	001	–	B	`bne rs1, rs2, label`	branch if ≠	`if (rs1 ≠ rs2) PC = BTA`
1100011 (99)	100	–	B	`blt rs1, rs2, label`	branch if <	`if (rs1 < rs2) PC = BTA`
1100011 (99)	101	–	B	`bge rs1, rs2, label`	branch if ≥	`if (rs1 ≥ rs2) PC = BTA`
1100011 (99)	110	–	B	`bltu rs1, rs2, label`	branch if < unsigned	`if (rs1 < rs2) PC = BTA`
1100011 (99)	111	–	B	`bgeu rs1, rs2, label`	branch if ≥ unsigned	`if (rs1 ≥ rs2) PC = BTA`
1100111 (103)	000	–	I	`jalr rd, rs1, imm`	jump and link register	`PC = rs1 + SignExt(imm), rd = PC + 4`
1101111 (111)	–	–	J	`jal rd, label`	jump and link	`PC = JTA, rd = PC + 4`

*Encoded in $instr_{31:25}$, the upper seven bits of the immediate field

Pseudoinstruction	RISC-V Instructions	Description	Operation
`li rd, imm31:0`	`lui rd, imm31:12`* `addi rd, rd, imm11:0`	load 32-bit immediate	`rd = imm31:0`
`mv rd, rs1`	`addi rd, rs1, 0`	move (also called “register copy”)	`rd = rs1`
`j label`	`jal x0, label`	jump	`PC = label`
`jal label`	`jal ra, label`	jump and link	`PC = label, ra = PC + 4`
`jr rs1`	`jalr x0, rs1, 0`	jump register	`PC = rs1`
`jalr rs1`	`jalr ra, rs1, 0`	jump and link register	`PC = rs1, ra = PC + 4`
`ret`	`jalr x0, ra, 0`	return from function	`PC = ra`
`call label`	`jal ra, label`	call nearby function	`PC = label, ra = PC + 4`
`call label`	`auipc ra, offset31:12`* `jalr ra, ra, offset11:0`	call far away function	`PC = PC + offset, ra = PC + 4`
`la rd, symbol`	`auipc rd, symbol31:12`* `addi rd, rd, symbol11:0`	load address of global variable	`rd = PC + symbol`

*If bit 11 of the immediate / offset / symbol is 1, the upper immediate is incremented by 1. `symbol` and `offset` are the 32-bit PC-relative addresses of a label and a global variable, respectively.

Abbildung 11.1: RISC-V Befehlsreferenz

Zusätzlicher Platz für Lösungen. Markieren Sie deutlich die Zuordnung zur jeweiligen Teilaufgabe. Vergessen Sie nicht, ungültige Lösungen zu streichen.