KIT-Fakultät für Informatik
Prof. Dr. Uwe Hanebeck, Prof. Dr.-Ing. Jörg Henkel

Musterlösungen zur Klausur

Digitaltechnik und Entwurfsverfahren (TI-1)

und

Rechnerorganisation (TI-2)

am 07. März 2025, 08:00 – 10:00 Uhr

Name:	Vorname:	Matrikelnummer:
Bond	James	007

Digitaltechnik und Entwurfsverfahren (TI-1)	
Aufgabe 1	9 von 9 Punkten
Aufgabe 2	7 von 7 Punkten
Aufgabe 3	12 von 12 Punkten
Aufgabe 4	7 von 7 Punkten
Aufgabe 5	10 von 10 Punkten

Rechnerorganisation (TI-2)	
Aufgabe 6	10 von 10 Punkten
Aufgabe 7	8 von 8 Punkten
Aufgabe 8	9 von 9 Punkten
Aufgabe 9	10 von 10 Punkten
Aufgabe 10	8 von 8 Punkten

Gesamtpunktzahl:	90 von 90 Punkten

	Note: **1,0**

Aufgabe 1 *Rechnerarithmetik* (9 Punkte)

1. 712_{10} in Basis 7: 2 P.

$$\begin{aligned} 712 : 7 &= 101\,\text{Rest}5 \\ 101 : 7 &= 14\,\text{Rest}3 \\ 14 : 7 &= 2\,\text{Rest}0 \\ 2 : 7 &= 0\,\text{Rest}2 \\ \Rightarrow 712_{10} &= 2035_7 \end{aligned}$$

2. $712,25_{10}$ in IEEE 754-Standard für einfache Genauigkeit: 3 P.
   Binärdarstellung von 712_{10}:

$$\begin{aligned} 712_10 &\rightarrow 1011001000_2 \\ 0,25_{10} &\rightarrow 0,01_2 \\ &\Rightarrow 1011001000,01_2 \end{aligned}$$

   Charakteristik: $1011001000,01_2 = 1,01100100001_2 \cdot 2^9$

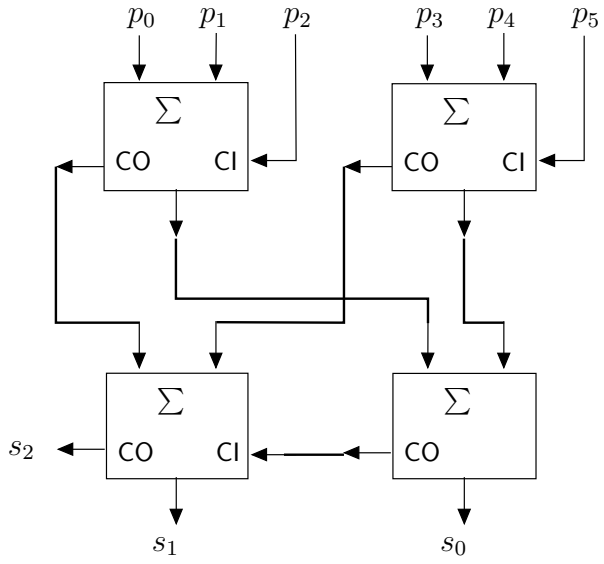
$$9 + 127 = 136_{10} = 10001000_2$$

   IEEE 754-Standard für einfache Genauigkeit:

$$\underbrace{0}_{positiv} \quad \underbrace{100\,0100\,0}_{Charakteristik} \quad \underbrace{011\,0010\,0001\,0000\,0000\,0000}_{Mantisse}$$

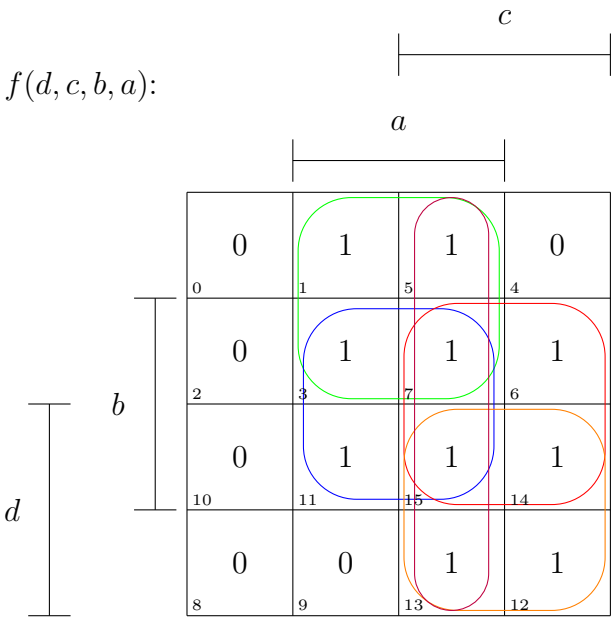
   Hexadezimale Repräsentation: $0x44321000$

3. Schaltung zur Berechnung der Summe $s_2\,s_1\,s_0$: 4 P.



Aufgabe 2 *Schaltfunktion* (7 Punkte)

1. KV-Diagramm: 5 P.



Die Primimplikanten sind:

$\overline{d} \wedge a$	$b \wedge a$	$c \wedge b$
$d \wedge c$	$c \wedge a$	

2. Disjunktive Minimalformen: $(c \wedge a)$ ist entbehrlich. Alle anderen Primplikante sind kern und prim. Es folgt als einzige disjunktive Minimalform: 2 P.

$$f(d,c,b,a) = \overline{d}\,a \vee b\,a \vee c\,b \vee d\,c$$

Aufgabe 3 *Schaltnetze* (12 Punkte)

1. Schaltfunktion z: 1 P.

$$z = \Big((\overline{c} \leftrightarrow b)\,(\overline{b} \vee a) \Big) \vee \Big((\overline{c}\,\overline{a})\,(b \vee (a \nleftrightarrow b)) \Big)$$

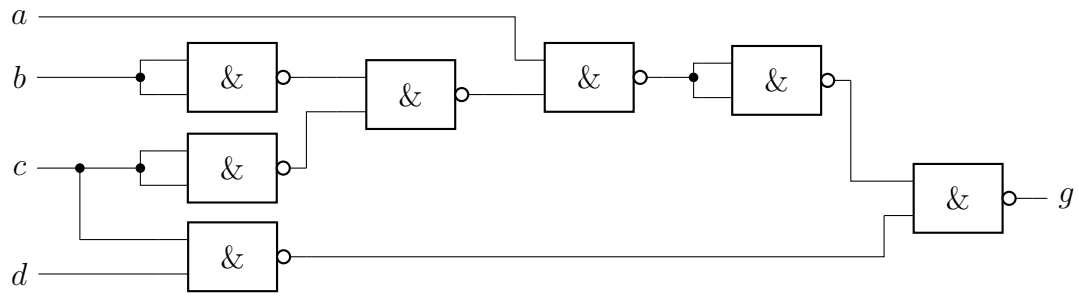
2. Zweistufige disjunktive Form von z: 3 P.

$$\begin{aligned}
z &= \Big((\overline{c} \leftrightarrow b)\,(\overline{b} \vee a) \Big) \vee \Big((\overline{c}\,\overline{a})\,(b \vee (a \nleftrightarrow b)) \Big) \\
&= \Big(\Big(\overline{c}\,b \vee c\,\overline{b} \Big)\,(\overline{b} \vee a) \Big) \vee \Big((\overline{c}\,\overline{a})\,\Big(b \vee \Big(a\,\overline{b} \vee \overline{a}\,b \Big) \Big) \Big) \\
&= \underbrace{\overline{c}\,b\,\overline{b}}_{=0} \vee \overline{c}\,b\,a \vee \underbrace{c\,\overline{b}\,\overline{b}}_{=c\,\overline{b}} \vee c\,\overline{b}\,a \vee \overline{c}\,\overline{a}\Big(\underbrace{b \vee b\,\overline{a}}_{=b} \vee \overline{b}\,a \Big) \\
&= \overline{c}\,b\,a \vee\ c\,\overline{b} \vee c\,\overline{b}\,a \vee\ \overline{c}\,\overline{a}\,b \vee \overline{c}\,\overline{a}\,b\,a \\
&= \overline{c}\,b\,a \vee\ c\,\overline{b} \vee c\,\overline{b}\,a \vee\ \overline{c}\,b\,\overline{a} \\
&= c\,\overline{b} \vee\ \overline{c}\,b\,a \vee\ \overline{c}\,b\,\overline{a} \\
&= c\,\overline{b} \vee\ \overline{c}\,b\ (a \vee \overline{a}) \\
&= c\,\overline{b} \vee\ \overline{c}\,b
\end{aligned}$$

3. g durch NAND-Gatter: 4 P.

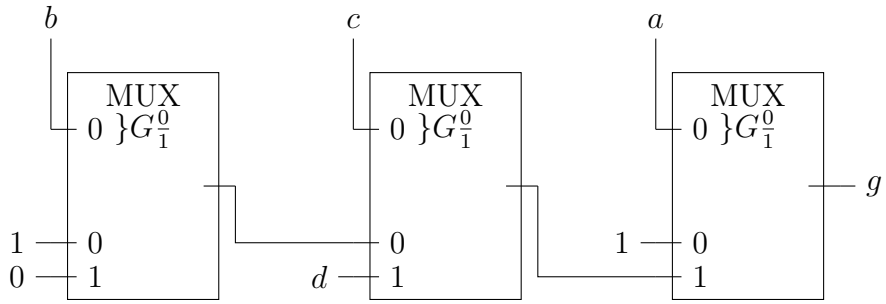
$$\begin{aligned}
g &= \overline{\overline{(\overline{a} \vee\ \overline{c}\,\overline{b}) \vee c\,d}} = \overline{(a \wedge \overline{(\overline{c} \wedge \overline{b})}) \wedge \overline{(c \wedge d)}} \\
&= \overline{\overline{(a \wedge \overline{(\overline{c} \wedge \overline{b})})} \wedge \overline{(c \wedge d)}} \\
&= \overline{(a \mathbin{\overline{\wedge}} (\overline{c} \mathbin{\overline{\wedge}} \overline{b}))} \mathbin{\overline{\wedge}} (c \mathbin{\overline{\wedge}} d)
\end{aligned}$$

mit $\overline{x} = \overline{x \wedge x}$ folgt:

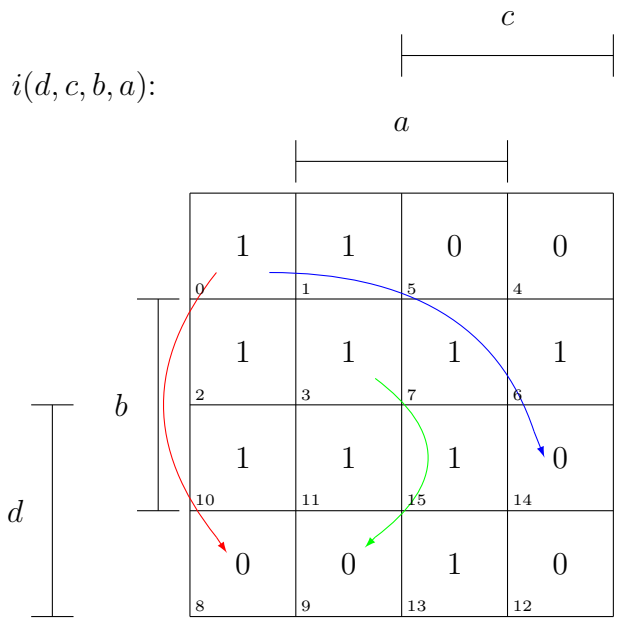


4. Multiplexer-Realisierung von g: 4 P.

$$\begin{aligned} g(d,c,b,a) &= \overline{a} \vee\ \overline{c}\,\overline{b} \vee c\,d \\ &= a\,(\overline{c}\,\overline{b} \vee c\,d) \vee \overline{a}\,(1) \\ &= a\,\Big(\overline{c}\,(\overline{b}) \vee c\,(d)\Big) \vee \overline{a}\,(1) \\ &= a\,\Big(\overline{c}\,\Big(b\,(0) \vee \overline{b}\,(1)\Big) \vee c\,\Big(d(1) \vee \overline{d}(0)\Big)\Big) \vee \overline{a}\,(1) \end{aligned}$$



Aufgabe 4 *Laufzeiteffekte* (7 Punkte)



1.

(a) $B_0 \to B_8$: 2 P.
Übergang $(0,0,0,0) \to (1,0,0,0)$ mit 1-Variable-Wechsel: Der Übergang enthält keinen Funktionshazard, da Übergänge mit einem 1-Variable-Wechsel keine Funktionshazards verursachen können.

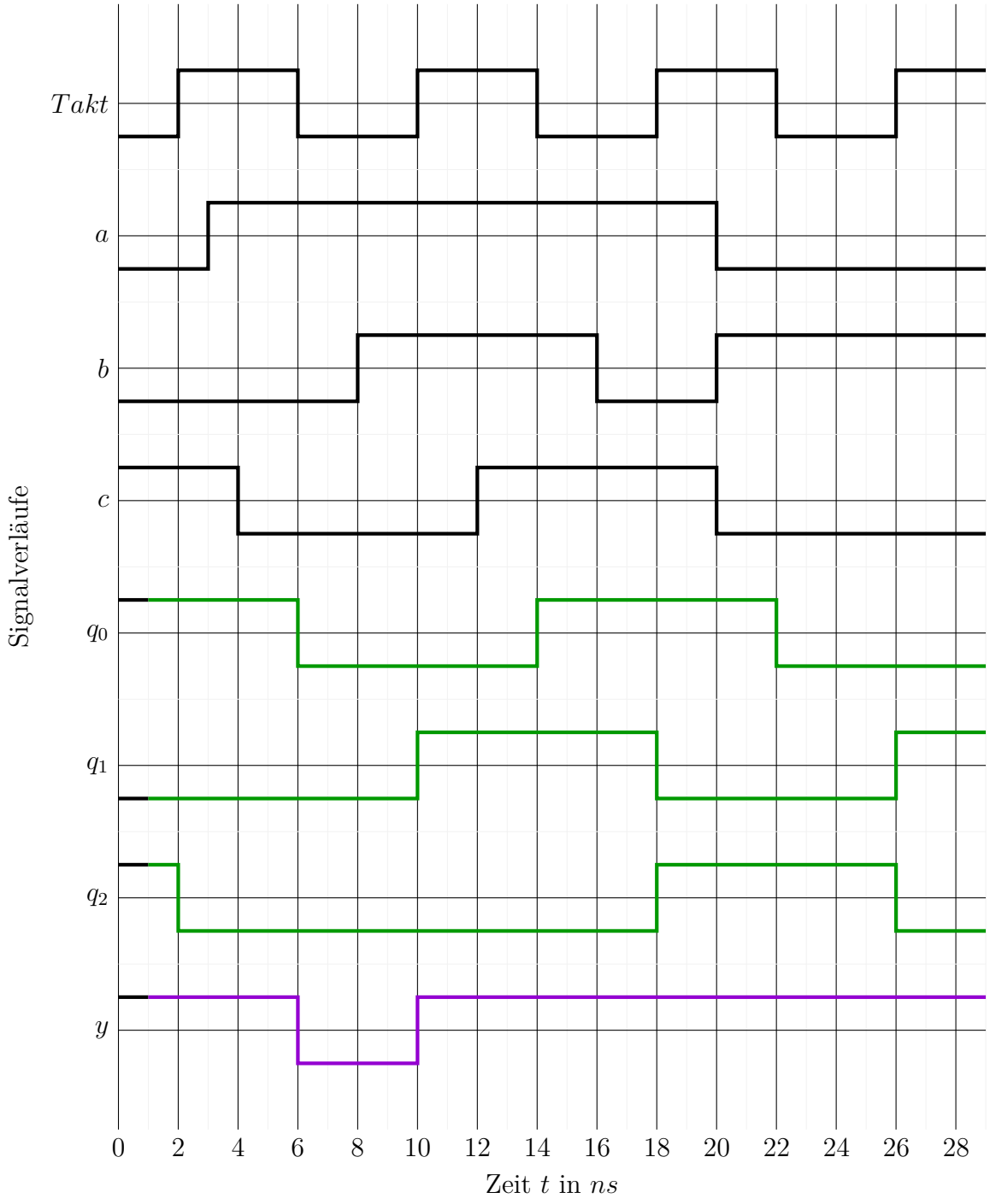
(b) $B_0 \to B_{14}$: 2 P.
Übergang $(0,0,0,0) \to (1,1,1,0)$ mit 3-Variable-Wechsel $\Rightarrow 3! = 6$ Wege. Es existiert mindestens eine nicht-monotone Folge der Funktionswerte (z.B. $B_0 \to B_4 \to B_6 \to B_{14}$) $\Rightarrow$ Übergang ist mit einem dynamischen 1-0 Funktionshazard behaftet.

(c) $B_3 \to B_9$: 2 P.
Übergang $(0,0,1,1) \to (1,0,0,1)$ mit 2-Variable-Wechsel $\Rightarrow 2! = 2$ Wege. Alle zugehörigen Folgen der Funktionswerte sind monoton $\Rightarrow$ Übergang ist frei von Funktionshazards.

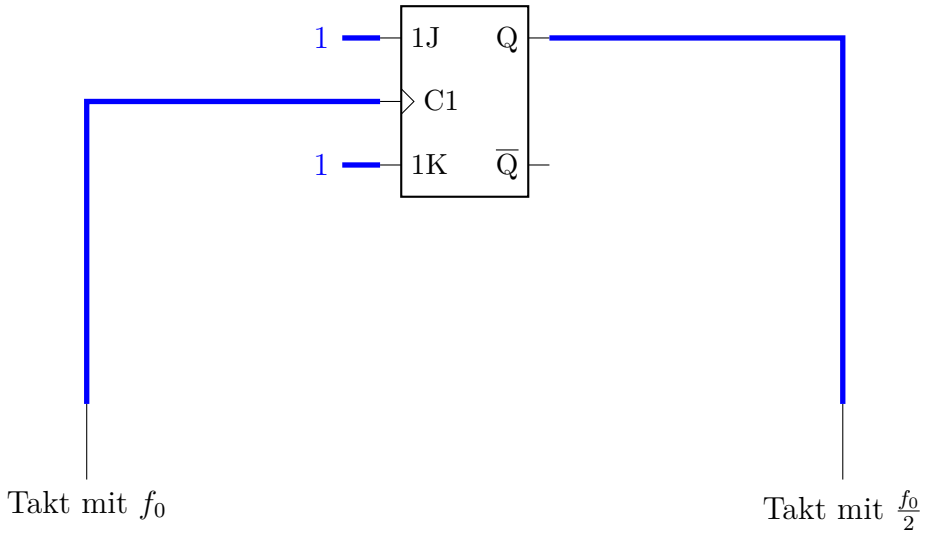
2. Behebungsmöglichkeit: 1 P.
Funktionshazards können nicht behoben werden, da der Hazard durch die Funktion selbst begründet ist und somit jede Behebungsmaßnahme zu einer Änderung der Schaltfunktion führt. Es kann lediglich durch geschickte Wahl der Verzögerungszeiten das Auftreten des Hazardfehlers vermieden werden.

Aufgabe 5 *Schaltwerke* (10 Punkte)

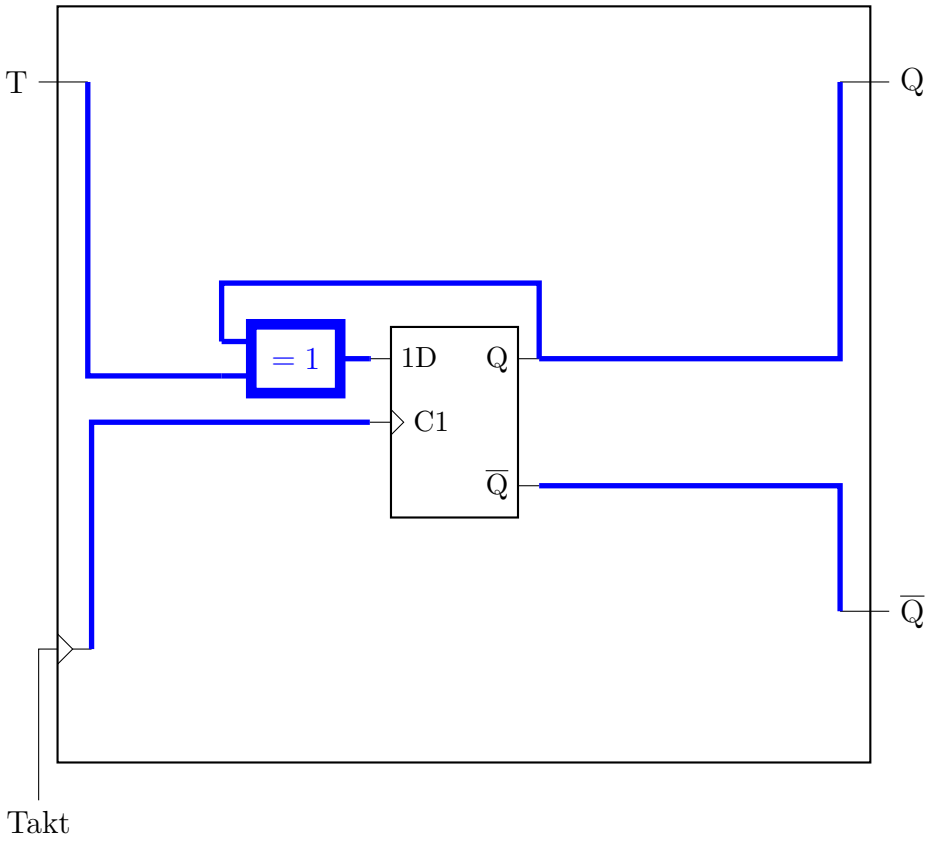
1. Zeitdiagramm: 7 P.



2. Frequenzteiler: 1 P.



3. T-Flipflop aus D-Flipflop: 2 P.



Aufgabe 6 *MIMA-Architektur* (10 Punkte)

1. Mikroprogramm für die Lese-Phase: 5 P.

   1. Takt: IAR $\rightarrow$ SAR; IAR $\rightarrow$ X; R = 1
   2. Takt: Eins $\rightarrow$ Y; R = 1
   3. Takt: ALU auf Addieren; R = 1
   4. Takt: Z $\rightarrow$ IAR
   5. Takt: SDR $\rightarrow$ IR

   } Lese-Phase

2. Ausführungsmodell: 2 P.
   Die MIMA unterliegt der **Akkumulator-Architektur**. Dabei spielt der Akkumulator eine zentrale Rolle. Der Akkumulator dient zeitgleich als Operand und Ziel einer Operation. Dies führt zu einem kompakten Befehlssatz und einer einfachen Implementierung mit Nachteilen in der Parallelisierbarkeit und der Ausführungsgeschwindigkeit, da viele Speicherzugriffe nötig (2. Operand meist aus dem Speicher) sind.

3. Wert im Akkumulator und Meldesignal N: 3 P.

MIMA-Assemblercode	**Inhalt des Akkumulator**
`LDC 0x0ABCD`	`0x00ABCD`
`ADD 0x00001`	`0x00ABDE`
`XOR 0x00002`	`0x00A16E`
`EQL 0x00003`	`0XFFFFFF`

   Als Meldesignal N liegt somit eine 1 an.

Aufgabe 7 *RISC-V Assembler* (8 Punkte)

1. Verhalten des Codes: 2 P.
   `t2` wird in jeder Iteration um i Bits nach links verschoben, wodurch 2er-Potenzen erzeugt werden. Diese werden in `t4` aufsummiert. Am Ende wird `t4` im Speicher an der Adresse `0x10010000` abgelegt.

2. Inhalt des Speichers an der Adresse `0x10010000`: 2 P.
   Die Summe der Potenzen von 2^0 bis 2^9 beträgt $1024 - 1 = 1023$. In Hexadezimal ist dies `0x000003FF`.

3. Nur gerade Werte von i: 2 P.
   Anstatt `addi t0, t0, 1` muss `addi t0, t0, 2` verwendet werden.

4. Zwischenergebnisse in einem Array speichern: 2 P.

```
1       ...
2       sw    t4, 0(t3)      # Speichere das Ergebnis in den Speicher
3       addi  t3, t3, 4      # Erhöhe die Speicheradresse um 4 Byte
4
5       blt   t0, t1, loop   # Wiederhole, solange i < n
6
```

Aufgabe 8 *Pipelining* (9 Punkte)

1. Aufgaben der Pipeline-Stufen: 2 P.

   - **IF** (Instruction Fetch):
     Es wird der aktuelle Befehl anhand des Befehlszählers (Program Counters) aus dem Speicher geladen. Der PC wird anschließend inkrementiert. Der geladene Befehl wird an die nächste Stufe weitergegeben.
   - **ID** (Instruction Decode):
     Der Befehl wird dekodiert, um zu erkennen, dass es sich um einen Lade-Befehl handelt. Der Basisregisterwert und das Offset werden aus dem Befehlswort extrahiert.
   - **EX** (Execution):
     Die effektive Adresse wird berechnet, indem der Basisregisterwert und das Offset addiert werden.
   - **MEM** (Memory Access):
     Die berechnete Speicheradresse wird verwendet um das Datum aus dem Speicher zu lesen. Der gelesene Wert wird an die nächste Stufe weitergegeben.
   - **WB** (Write Back):
     Der aus dem Speicher gelesene Wert wird in das Zielregister geschrieben.

2. Grund: 2 P.
   Ausgabeabhängigkeiten (*Output dependence*) und Gegenabhängigkeiten (*Anti-dependence*) können nicht zu Konflikten führen, da

   - das Lesen aus Registern immer in der Stufe 2 (ID) und
   - das Schreiben in Register immer in der Stufe 5 (WB) erfolgt.

3. Bedingte Sprünge: 3 P.
   Erst in der Ausführungsphase wird entschieden, ob der Sprung ausgeführt wird. Somit werden Befehle in die Pipeline geladen, die möglicherweise nicht ausgeführt werden sollen.
   Behandlungsmöglichkeiten:

   - Verzögerte Sprungtechnik (*delayed branch technique*): z.B. drei Verzögerungszeitschlitze (*delay slots*) mit Leerbefehlen (`NOP`) nach jedem Sprungbefehl.
   - Befehlsumordnung: Befehle, die in der logischen Programmreihenfolge vor dem Sprungbefehl liegen, in die Verzögerungszeitschlitze verschieben.
   - Pipeline-Leerlauf (ineffizient): Die Hardware erkennt die Verzweigungsbefehle in der ID-Phase und lädt keine weiteren Befehle in die Pipeline, bis die Zieladresse berechnet und im Fall bedingter Sprungbefehle die Sprungentscheidung getroffen ist.
   - Sprungvorhersage: Spekulation über Sprungentscheidung und/oder Sprungziel, um möglichst wenig Pipeline-Leerlauf/Delay Slots zu haben.

4. *Result-Forwarding und Load-Forwarding*: 2 P.
   Result-Forwarding ist eine Technik, bei der das Ergebnis einer arithmetisch-logischen Operation (ALU-Operation) direkt an nachfolgende Befehle weitergeleitet wird, ohne dass es zuerst in einem Register abgelegt werden muss. Ähnliches gilt für Load-Forwarding, nur dass hier die Speicherdaten direkt an nachfolgende Befehle weitergeleitet werden, ohne dass es zuerst in einem Register abgelegt wird.

Aufgabe 9 *Cache-Speicher* (10 Punkte)

1 P.

1. Zugriffsdauer t_m:

$$t_m = h_{\text{Cache}} \cdot t_{\text{Cache}} + m_{\text{Cache}} \cdot t_{\text{Hauptspeicher}} = 0,9 \cdot 10 + 0,1 \cdot 200 = 29 \text{ [ns]}$$

1 P.

2. Anzahl der Sätze:
   Über die Anzahl der Zeilen und die Organisationsform des Caches kann die Anzahl der Sätze berechnet werden.
   Anzahl der Zeilen:

$$\#\text{Zeilen} = \frac{\text{Kapazität}}{\text{Blocksize}} = \frac{64 \text{ KiByte}}{64 \text{ Byte}} = \frac{2^{16}}{2^6} = 2^{10}$$

   Es folgt:

$$\#\text{Sätze} = \frac{\#\text{Zeilen}}{\text{Assoziativität}} = \frac{2^{10}}{4} = 2^8$$

   Es sind somit $2^8 = 256$ Sätze vorhanden.

1 P.

3. Länge des Tags:
   Jedes Byte im Hauptspeicher muss adressierbar sein. Der Hauptspeicher verfügt über eine Kapazität von 1024 KiByte = 2^{20} Byte. Somit ist die Adresse 20 Bit lang. Es sind 8 Bit für den Satzindex und 6 Bit für den Blockoffset nötig. Es folgt für den Tag: $20 - 8 - 6 = 6$.

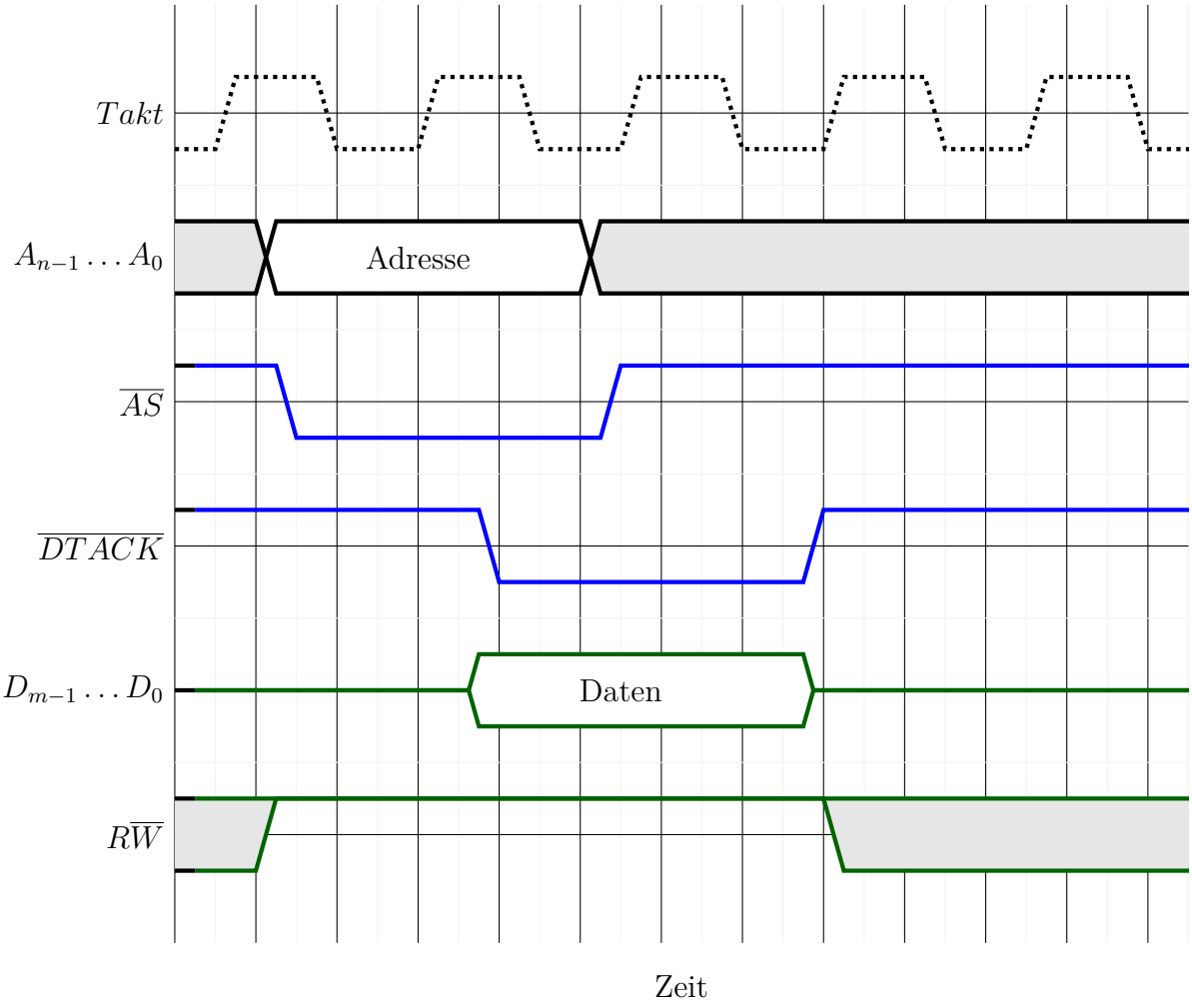
5+2 P.

4.

hex. Adresse	**0x00 000**	**0x00 A32**	**0xFA 711**	**0x36 421**	**0xFA 717**	**0x36 729**	**0x5F 321**	**0x00 038**	**0x76 F29**	**0x76 F0A**
read/write	r	r	w	r	w	w	r	w	w	r
hex. Index	0x00	0x28	0x9C	0x90	0x9C	0x9C	0xCC	0x00	0xBC	0xBC
hex. Tag	0x00	0x00	0x3E	0x0D	0x3E	0x0D	0x17	0x00	0x1D	0x1D
Hit/Miss	M	M	M	M	(H)	M	M	H	M	(H)

Aufgabe 10 *Verbindungsstrukturen* (8 Punkte)

1. Zeitdiagramm für das Lesen: 4 P.



2. Mehrere Prozessorchips: 3 P.
   Eine Bus-Struktur ist nicht für diese Anwendung geeignet, da die Prozessorchips sich gegenseitig blockieren und ausbremsen würden. Besser geeignet sind für diesen Zweck Punkt-zu-Punkt-Verbindungen, die eine direkte Kommunikation zwischen den Prozessorchips und dem Speicher ermöglichen und so keine Blockade entsteht. Außerdem sind höhere Datenrate, eine niedrigere Latenz und eine bessere Skalierbarkeit gegeben. Ein Beispiel für eine solche Verbindungsstruktur ist das QuickPath Interconnect (QPI) von Intel.

3. *USB* ausgeschrieben: 1 P.
   USB steht für *Universal Serial Bus*.