KIT-Fakultät für Informatik
Prof. Dr. Mehdi Tahoori, Prof. Dr.-Ing. Jörg Henkel

Musterlösungen zur Klausur

Digitaltechnik und Entwurfsverfahren (TI-1)

und

Rechnerorganisation (TI-2)

am 15. August 2025, 11:00 – 13:00 Uhr

Name:	Vorname:	Matrikelnummer:
Bond	James	007

Digitaltechnik und Entwurfsverfahren (TI-1)	
Aufgabe 1	6 von 6 Punkten
Aufgabe 2	12 von 12 Punkten
Aufgabe 3	7 von 7 Punkten
Aufgabe 4	6 von 6 Punkten
Aufgabe 5	14 von 14 Punkten

Rechnerorganisation (TI-2)	
Aufgabe 6	10 von 10 Punkten
Aufgabe 7	6 von 6 Punkten
Aufgabe 8	9 von 9 Punkten
Aufgabe 9	12 von 12 Punkten
Aufgabe 10	8 von 8 Punkten

Gesamtpunktzahl:	90 von 90 Punkten

	Note: **1,0**

Aufgabe 1 Rechnerarithmetik (6 Punkte)

1. Darstellung im Zweierkomplement: 2 P.

   - -1_{10}:
     Für $n = 1$ Bits gilt der der Wertebereich $[-1, 0]$. Es folgt $-1_{10} = 1_{ZK}$.

   - -13_{10}:
     Für $n = 5$ Bits gilt der der Wertebereich $[-16, 15]$. Es folgt $-13_{10} \rightarrow 13_{10} = 0\ 1101_2 = 1\ 0010_{EK} = 1\ 0011_{ZK}$.

   - 256_{10}:
     Für $n = 10$ Bits gilt der der Wertebereich $[-512, 511]$. Es folgt $256_{10} = 01\ 0000\ 0000_{ZK}$.

   - -17_{10}:
     Für $n = 6$ Bits gilt der der Wertebereich $[-32, 31]$. Es folgt $-17_{10} \rightarrow 17_{10} = 01\ 0001_2 = 10\ 1110_{EK} = 10\ 1111_{ZK}$.

2. Dezimale Darstellung: 1 P.

   - 1111_{ZK}:
     Umrechnung: $1111_{ZK} = 1110_{EK} = 0001_2 \rightarrow -1_{10}$.

   - 10101_{ZK}:
     Umrechnung: $1\ 0101_{ZK} = 1\ 0100_{EK} = 0\ 1011_2 \rightarrow -11_{10}$.

3. Ungleichung: 1 P.

$$2^k \geq m + k + 1$$

4. Länge des Codewortes: 1 P.
   In obige Gleichung eingesetzt mit $m = 11$ folgt für $k = 4$. Die Länge des Codewortes beträgt dann $m + k = 15$.

5. Erläuterung der Ungleichung: 1 P.
   2^k gibt die Anzahl der unterschiedlichen Bitmuster (abzudeckenden Fälle) an, die sich mit k Prüfbits darstellen lassen. $m + k$ entspricht der Anzahl möglicher Fehlerpositionen, da ein Fehler sowohl an einem der m Datenbits als auch an einem der k Prüfbits auftreten kann. $+1$ berücksichtigt den Fall, dass kein Fehler aufgetreten ist. Auch dieser muss eindeutig erkannt und korrekt behandelt werden.

Aufgabe 2 Minimierungsverfahren (12 Punkte)

1. + 2. 1. Quinesche Tabelle: 1+6 P.

#	Nr.	0. Ordnung	
0	0	0 0 0 0	✓
1	1	0 0 0 1	✓
2	3	0 0 1 1	✓
	12	1 1 0 0	✓
3	7	0 1 1 1	✓
	13	1 1 0 1	✓
	14	1 1 1 0	✓
4	15	1 1 1 1	✓

#	Nr.	1. Ordnung	
0	0,1	0 0 0 -	no
1	1,3	0 0 - 1	A
2	3,7	0 - 1 1	B
	12,13	1 1 0 -	✓
	12,14	1 1 - 0	✓
3	7,15	- 1 1 1	C
	13,15	1 1 - 1	✓
	14,15	1 1 1 -	✓

#	Nr.	2. Ordnung	
2	12,13,14,15	1 1 - -	D
	~~12,14,13,15~~	~~1 1 - -~~	

Die Primimplikanten sind:

$$A : \overline{x_3}\,\overline{x_2}\,x_0 \quad B : \overline{x_3}\,x_1\,x_0 \quad C : x_2\,x_1\,x_0 \quad D : x_3\,x_2$$

(Das „Primimplikant“ über die Minterme 0,1 besteht nur aus Don’t-Care-Stellen und ist somit kein Primimplikant der Funktion f.)

3. 2. Quinesche Tabelle : 2 P.

	Minterme												
Primimplikant	3	7	12	13	14	15							
A	✓					✓							
B	✓	✓											
C		✓				✓							
D			✓	✓	✓	✓							

Die Kernprimimplikanten sind:

$$D : x_3\, x_2$$

4. 2 P.

Reduzierte Tabelle:

	Minterme												
Primimplikant	3	7											
A	✓												
B	✓	✓											
C		✓											

Konjunktive Form der Überdeckungsfunktion:
Aus der Tabelle folgt die konjunktive Überdeckungsfunktion $\ddot{u}_f = (w_a \vee w_b)(w_b \vee w_c)$.

5. Disjunktive Minimalform (DMF): 1 P.
Aus der Überdeckungsfunktion und den Kermprimimplikanten folgen die disjunktiven Minimalformen:

$$f(x_3, x_2, x_1, x_0) = x_3\, x_2 \vee \begin{cases} \overline{x_3}\, x_1\, x_0 \\ \overline{x_3}\, \overline{x_2}\, x_0 \vee x_2\, x_1\, x_0 \end{cases}$$

Aufgabe 3 Boolesche Algebra & Schaltnetze (7 Punkte)

1. Vollständiges Operatorensystem: 3 P.
   Zu zeigen ist, dass die Operatorenmenge $\{\oplus, \wedge\}$ (XOR, UND) zusammen mit der Konstanten 1 ein vollständiges Operatorensystem bildet. Dazu muss gezeigt werden, dass die Konjunktion, Disjunkion und die Negation mit Hilfe der Operatorenmenge $\{\oplus, \wedge\}$ und der Konstanten 1 gebildet werden können.

   - Konjunktion $\wedge$: Trivialerweise, da $\wedge$ bereits in der Operatorenmenge enthalten ist.
   - Negation $\overline{a}$: Es gilt $\overline{a} = a \oplus 1$. Begründung:

a	1	$a \oplus 1$	$\overline{a}$
0	1	1	1
1	1	0	0

   - Disjunktion $\vee$:
     Es gilt $a \vee b = \overline{\overline{a} \wedge \overline{b}}$ (De-Morgan-Regel). Es folgt:

$$\begin{aligned} a \vee b = \overline{\overline{a} \wedge \overline{b}} &= \overline{(a \oplus 1) \wedge (b \oplus 1)} \\ &= ((a \oplus 1) \wedge (b \oplus 1)) \oplus 1 \end{aligned}$$

2. Schaltfunktion $g(c, b, a)$: 1 P.

$$g(c, b, a) = \Big((c \leftrightarrows b) \wedge (\overline{c} \vee \overline{b})\Big) \vee \Big(a \vee (\overline{c} \vee \overline{b})\Big) \vee \overline{(c\, b\, \overline{a})}$$

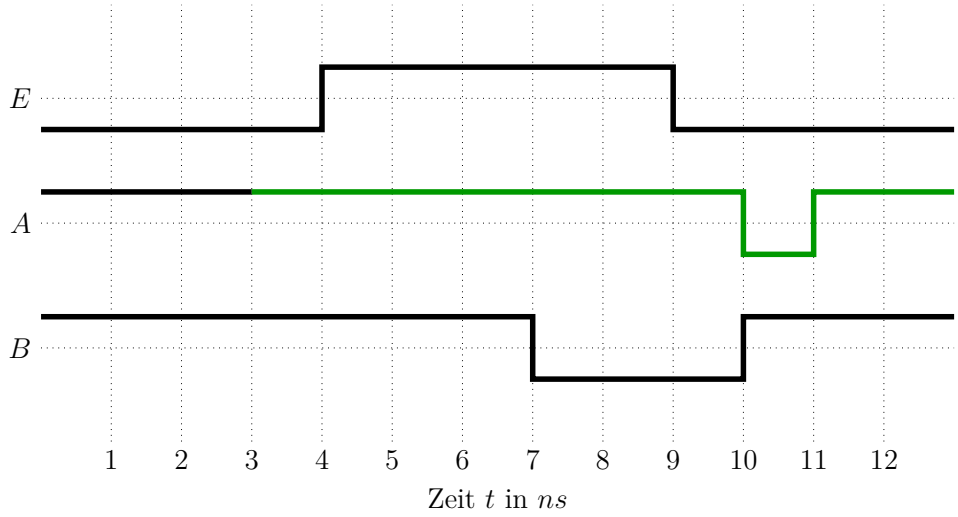
3. Wahrheitstabelle von $g(c, b, a)$: 2 P.

	Hilfsspalten (nicht gefordert)					
$c\ b\ a$	$c \leftrightarrows b$	$\overline{c} \vee \overline{b}$	$(c \leftrightarrows b) \wedge (\overline{c} \vee \overline{b})$	$a \vee (\overline{c} \vee \overline{b})$	$\overline{(c\, b\, \overline{a})}$	$g(c, b, a)$
0 0 0	0	1	0	1	1	1
0 0 1	0	1	0	1	1	1
0 1 0	1	1	1	1	1	1
0 1 1	1	1	1	1	1	1
1 0 0	1	1	1	1	1	1
1 0 1	1	1	1	1	1	1
1 1 0	0	0	0	0	0	0
1 1 1	0	0	0	1	1	1

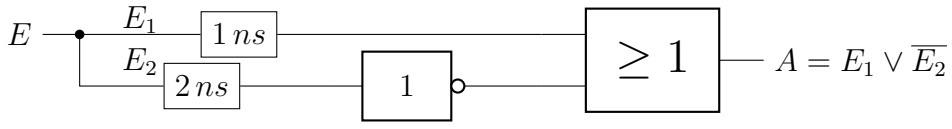
4. Konjunktive Minimalform (KMF): 1 P.
   Aus der Wahrheitstabelle folgt $g(c, b, a) = \overline{(c\, b\, \overline{a})} = \overline{c} \vee \overline{b} \vee a$.

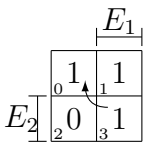
Aufgabe 4 Laufzeiteffekte (6 Punkte)

1. Verlauf des Signals A: 2 P.



2. Analyse auf Hazardfehler: 2 P.
Es handelt sich um einen statischen 1-Strukturhazardfehler.

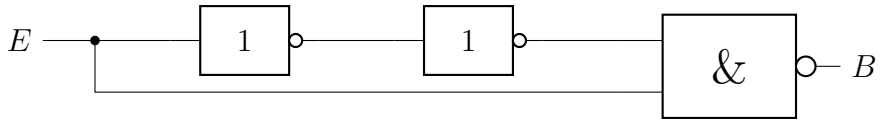




Der Hazardfehler tritt auf, wenn E von 1 zu 0 wechselt, d.h. beim Übergang (E_2, E_1) : $(1,1) \to (0,0)$. Da $\tau_{E_1} < \tau_{E_2}$ wechselt E_1 vor E_2 und der Weg $(1,1) \to (1,0) \to (0,0)$ wird angenommen. Die zugehörige Folge des Strukturausdrucks $1 \to 0 \to 1$ ist nicht monoton.

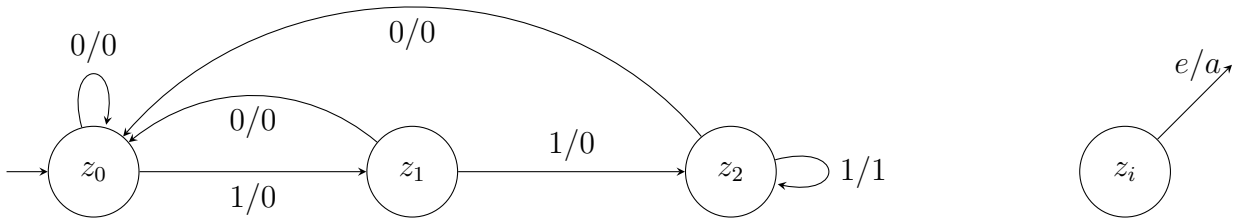
Alternativ: Die Funktion hat nur eine Eingangsvariable. Ein Übergang, bei dem nur eine Eingangsvariable wechselt, ist immer frei von Funktionshazards. Da aber offenbar dennoch ein Hazardfehler auftritt, muss es sich zwingend um einen statischen 1-Strukturhazardfehler handeln (Funktionswert bleibt gleich). Die Ursache ist die unterschiedliche Verzögerung der Signale E_1 und E_2. E_1 schaltet vor E_2. Es kommt zu einem Signalverlauf $1 \to 0 \to 1$.

3. Schaltnetz für Signal B: 2 P.



Aufgabe 5 Schaltwerke (14 Punkte)

1. Automatengraph: Anzahl der Zustände: 3 — 4 P.



2. Kodierte Ablauftabelle: — 2 P.

Zustand q_1^t	Zustand q_0^t	Eingabe e^t	Folgezustand q_1^{t+1}	Folgezustand q_0^{t+1}	FF-Ansteuersignale D_1^t	FF-Ansteuersignale D_0^t	Ausgabe a^t
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	0	0	0	0	0
0	1	1	1	0	1	0	0
1	0	0	0	0	0	0	0
1	0	1	1	0	1	0	1
1	1	0	-	-	-	-	-
1	1	1	-	-	-	-	-

3. (a) DMF der Zustandsübergangsgleichungen: — 2 P.

q_1^{t+1}:

q_0^t \ q_1^t e^t	00	01	11	10
0	0 (0)	0 (1)	1 (5)	0 (4)
1	0 (2)	1 (3)	- (7)	- (6)

$$q_1^{t+1} = e^t\, q_0^t \vee e^t\, q_1^t$$

q_0^{t+1}:

q_0^t \ q_1^t e^t	00	01	11	10
0	0 (0)	1 (1)	0 (5)	0 (4)
1	0 (2)	0 (3)	- (7)	- (6)

$$q_0^{t+1} = e^t\, \overline{q_1}^t\, \overline{q_0}^t$$

(b) DMF der Ansteuerfunktionen der Flipflops: 2 P.

D_1^t:

q_0^t \ e^t	0	1
0	0 (0)	0 (1)
1	1 (2)	- (3)
	0 (4)	- (5)
	1 (6)	- (7)

$$D_1^t = q_1^{t+1} = e^t\, q_0^t \vee e^t\, q_1^t$$

D_0^t:

q_0^t \ e^t	0	1
0	0 (0)	1 (1)
1	0 (2)	0 (3)
	0 (4)	- (5)
	0 (6)	- (7)

$$D_0^t = q_0^{t+1} = e^t\, \overline{q_1}^t\, \overline{q_0}^t$$

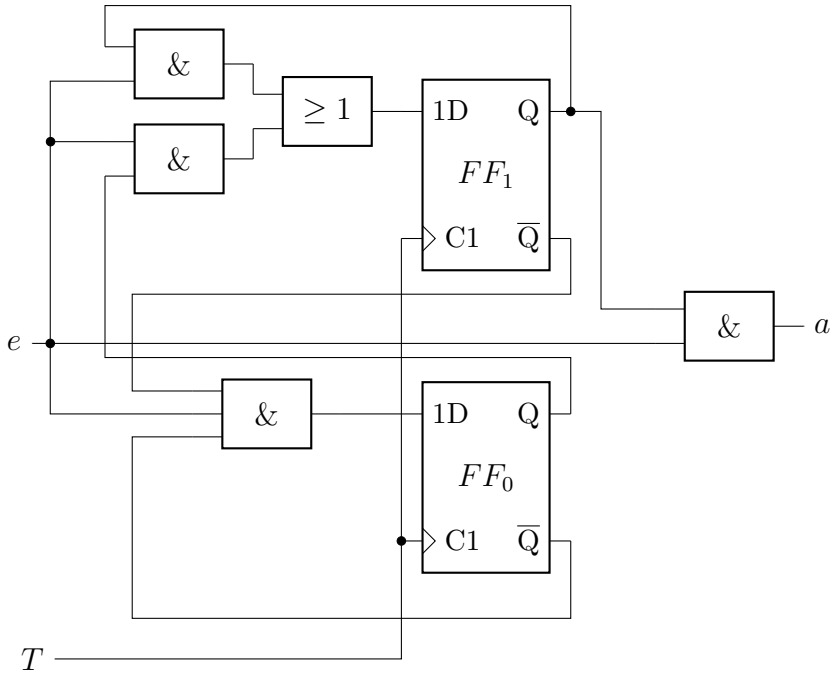
(c) DMF der Ausgabegleichung: 1 P.

a^t:

q_0^t \ e^t	0	1
0	0 (0)	0 (1)
1	0 (2)	- (3)
	1 (4)	- (5)
	0 (6)	- (7)

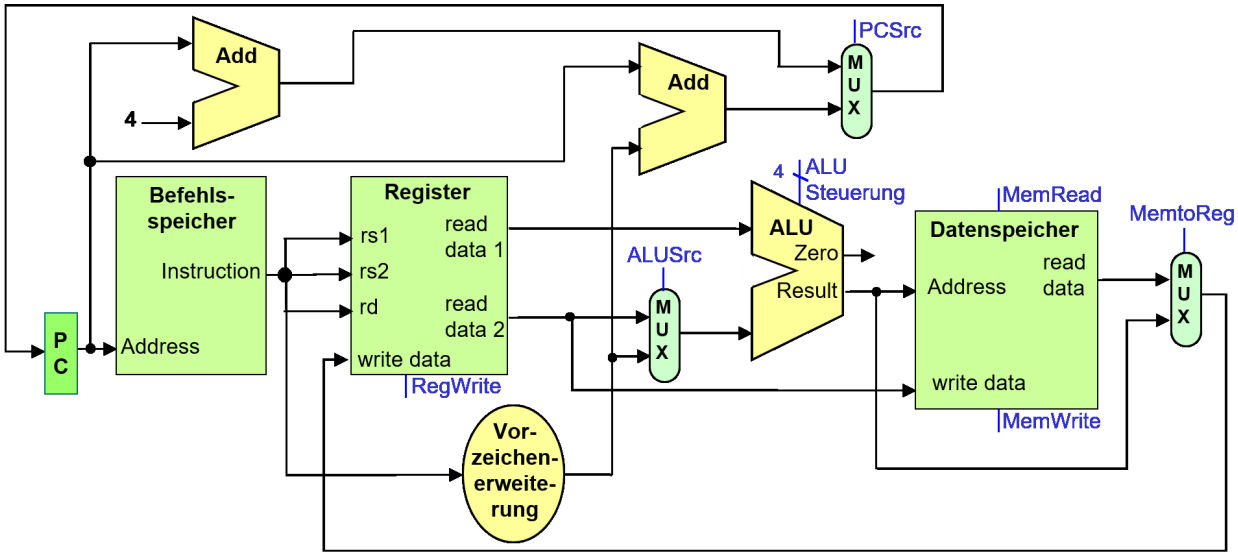
$$a^t = q_1^t\, e^t$$

4. Schaltung des Schaltwerkes: 3 P.



Aufgabe 6 RISC-V-Architektur (10 Punkte)

1. Vereinfachter Datenpfad: 5 P.



2. Rolle der Steuerleitungen: 3 P.

   - RegWrite: Aktiviert das Schreiben in ein Register. Es ist relevant bei add und lw, da hier Registerwerte aktualisiert werden.
   - PCSrc: Wählt zwischen dem nächsten Programmzählerwert (PC+4) und dem Zieladresse des Sprungs (im Fall von beq). Es ist somit bei allen Befehlen relevant.
   - ALUSrc: Wählt zwischen dem zweiten Operanden der ALU (Registerwert) und dem Immediate-Wert (bei lw und sw). Es ist somit bei allen Befehlen relevant.

3. Ablauf von *lw*: 2 P.
   Der PC gibt die Adresse an den Befehlsspeicher und somit wir der Befehl *lw* geladen. Der Registerspeicher liefert den Inhalt von `rs1` und somit die Basisadresse. Der `offset` wird vorzeichenerweitert und auf die Basisadresse via ALU addiert. Dafür muss der Multiplexer mit dem ALUSrc-Signal auf den Immediate-Wert geschaltet werden. Das Ergebnis der Addition ist die Speicheradresse, von der der Wert geladen wird. Der Datenwert wird aus dem Datenspeicher geladen und soll in das Register `rd` geschrieben werden. RegWrite wird dabei aktiviert. Gleichzeitig wird der Multiplexer mit dem Steuersignal `MemtoReg` so geschaltet, dass der geladene Wert in das Register `rd` geschrieben wird. Der PC wird um 4 erhöht, um auf den nächsten Befehl zu springen.

Aufgabe 7 RISC-V Assembler (6 Punkte)

1. Verhalten des Programms: 2 P.
   Das Programm summiert alle nicht-negativen Zahlen im Array `array` = $[3, 7, -2, 5, -1, 9]$.

2. Register $a0$: 1 P.

$$a0 = 3 + 7 + 5 + 9 = 24$$

3. Kontrollfluss: 2 P.
   Die Schleife wird ingesamt 6-mal durchlaufen (`t2` von 0 bis 5). Bei jedem Durchlauf wird zunächst geprüft, ob der Wert in `t2` noch kleiner als 6 ist, andernfalls Sprung zur Marke `end`. Anschließend wird ein Element des Arrays `array` geladen und auf ≥ 0 geprüft. Falls ja, dann wird zur Marke `addit` gesprungen, andernfalls zur Marke `skip`. Nach *addit* oder *skip* wird wieder an den Anfang der Schleife gesprungen.

4. Anzahl Speicherzugriffe: 1 P.
   In jedem Schleifendurchlauf wird einmal auf den Speicher zugegriffen. Es sind insgesamt 6 Schleifendurchläufe, also 6 Speicherzugriffe.

Aufgabe 8 Pipelining (9 Punkte)

1. Durchsatz: 1 P.
   Der Durchsatz ist die Anzahl der abgeschlossenen Instruktionen pro Takt. Im idealen Pipelining kann jede Taktperiode eine neue Instruktion abgeschlossen werden → Durchsatz = 1 Instruktion/Takt. Ohne Pipelining: 1 Instruktion alle 5 Takte → Durchsatz = 0,2 Instruktionen/Takt.

2. Taktzyklen: 1 P.
   Der erste Befehl benötigt 5 Taktzyklen zum „Füllen“ der Pipeline. Anschließend wird pro Takt ein weiterer Befehl abgeschlossen.

$$\text{Taktzyklen} = 5 + (100 - 1) = 104 \text{ Taktzyklen}$$

3. CPI: 1 P.

$$\text{CPI} = \frac{\text{Taktzyklen}}{\text{Anzahl der Instruktionen}} = \frac{104}{100} = 1,04$$

4. Ohne Pipeline: 1 P.
   Jede Instruktion benötigt 5 Taktzyklen, um ausgeführt zu werden. Bei 100 Instruktionen ergibt sich:

$$\text{Taktzyklen} = 5 \cdot 100 = 500 \text{ Taktzyklen}$$

5. Speedup: 1 P.

$$\text{Speedup} = \frac{\text{Taktzyklen ohne Pipeline}}{\text{Taktzyklen mit Pipeline}} = \frac{500}{104} \approx 4,81$$

6. Echte Datenabhängigkeit (Erklärung): 1 P.
   Es besteht eine echte Datenabhängigkeit von $Inst_i$ zu $Inst_j$, wenn $Inst_i$ einen Wert erzeugt, der von $Inst_j$ als Eingabe gelesen wird. ($Inst_i$ wird vor $Inst_j$ ausgeführt.) Diese Art von Abhängigkeit ist kritisch für Pipelines, da sie zu Stalls führen kann, wenn $Inst_j$ auf das Ergebnis von $Inst_i$ warten muss.

7. Echte Datenabhängigkeiten: 3 P.

S1 → S2 (x5) S1 → S4 (x5)
S2 → S3 (x6)
S3 → S4 (x7)
S4 → S5 (x8)
S5 → S6 (x5)

Aufgabe 9 Cache-Speicher (12 Punkte)

1. Direkt-abgebildeter Cache mit 32 Speicherblöcken: 6 P.

Adresse	Hilfsspalte (Binär)	Tag	Index	Offset	Hit/Miss
`0x03`	`0b0000 0011`	0	01	1	Miss
`0xb4`	`0b1011 0100`	2	1A	0	Miss
`0x2b`	`0b0010 1011`	0	15	1	Miss
`0x02`	`0b0000 0010`	0	01	0	Hit
`0xbf`	`0b1011 1111`	2	1F	1	Miss
`0x58`	`0b0101 1000`	1	0C	0	Miss
`0xbe`	`0b1011 1110`	2	1F	0	Hit
`0x0e`	`0b0000 1110`	0	07	0	Miss
`0xf5`	`0b1111 0101`	3	1A	1	Miss
`0x2c`	`0b0010 1100`	0	16	0	Miss
`0xba`	`0b1011 1010`	2	1D	0	Miss
`0xfd`	`0b1111 1101`	3	1E	1	Miss

2. Direkt-abgebildeter Cache mit 16 Speicherblöcken: 6 P.

Adresse	Hilfsspalte (Binär)	Tag	Index	Offset	Hit/Miss
`0x03`	`0b0000 0011`	0	0	3	Miss
`0xb4`	`0b1011 0100`	2	D	0	Miss
`0x2b`	`0b0010 1011`	0	A	3	Miss
`0x02`	`0b0000 0010`	0	0	2	Hit
`0xbf`	`0b1011 1111`	2	F	3	Miss
`0x58`	`0b0101 1000`	1	6	0	Miss
`0xbe`	`0b1011 1110`	2	F	2	Hit
`0x0e`	`0b0000 1110`	0	3	2	Miss
`0xf5`	`0b1111 0101`	3	D	1	Miss
`0x2c`	`0b0010 1100`	0	B	0	Miss
`0xba`	`0b1011 1010`	2	E	2	Miss
`0xfd`	`0b1111 1101`	3	F	1	Miss

Aufgabe 10 Speicherverwaltung (8 Punkte)

1. Unterteilung der virtuellen Adresse: 1 P.

2. Physikalische Adressen: 4 P.

Virtuelle		Physikalische	
Adresse	Seitennummer	Seitennummer	Adresse
1023	1	7	$7 \cdot 512 + 511 = 4095$
1024	2	3	$3 \cdot 512 + 0 = 1536$
1998	3	-	*page-fault*
2049	4	9	$9 \cdot 512 + 1 = 4609$

3. Bedingungen für Beschleunigung: 1 P.
   Eine Beschleunigung der Adressumsetzung durch den TLB wird erst beim zweiten Zugriff auf eine Seite und solange die entsprechenden Einträge aus dem Seitentabellen-Verzeichnis und der Seitentabelle aus dem TLB nicht verdrängt wurden erreicht.

4. Breite des Tags: 2 P.

   Seitengröße ist 4 KiByte $\Rightarrow$ Byte-Offset ist 12 Bit breit.

   Der Tag ist dann $(n - 12)$ Bits breit