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

Musterlösungen zur Klausur

Digitaltechnik und Entwurfsverfahren (TI-1)

und

Rechnerorganisation (TI-2)

am 17. August 2020, 9:00 – 11:00 Uhr

Name:	Vorname:	Matrikelnummer:
Bond	James	007

Digitaltechnik und Entwurfsverfahren (TI-1)	
Aufgabe 1	7 von 7 Punkten
Aufgabe 2	10 von 10 Punkten
Aufgabe 3	6 von 6 Punkten
Aufgabe 4	11 von 11 Punkten
Aufgabe 5	11 von 11 Punkten

Rechnerorganisation (TI-2)	
Aufgabe 6	5 von 5 Punkten
Aufgabe 7	9 von 9 Punkten
Aufgabe 8	10 von 10 Punkten
Aufgabe 9	12 von 12 Punkten
Aufgabe 10	9 von 9 Punkten

Gesamtpunktzahl:	90 von 90 Punkten

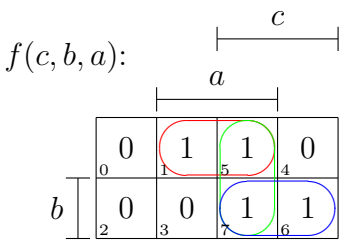
	Note: **1,0**

Aufgabe 1 *Schaltfunktionen* (7 Punkte)

1. DNF von $f(c,b,a)$: 1 P.

$$\begin{aligned} f(c,b,a) &= \text{MAXt}(0,2,3,4) \\ &= \text{MINt}(1,5,6,7) \\ &= (\overline{c}\ \overline{b}\ a) \vee (c\ \overline{b}\ a) \vee (c\ b\ \overline{a}) \vee (c\ b\ a) \end{aligned}$$

2. KV-Diagramm: 3 P.



Primimplikante:

- Kernprimimplikante: $\overline{b} \wedge a$, $c \wedge b$
- Entbehrliche Primimplikante: $c \wedge a$

DMF von $f(c,b,a)$: $(\overline{b} \wedge a) \vee (c \wedge b)$

3. Schaltnetz: 1 P.

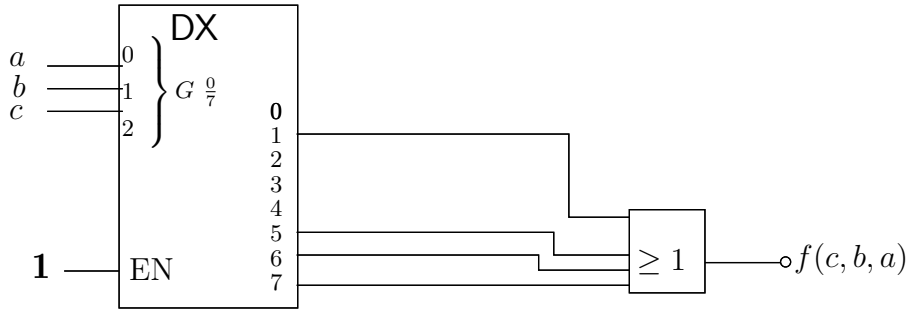
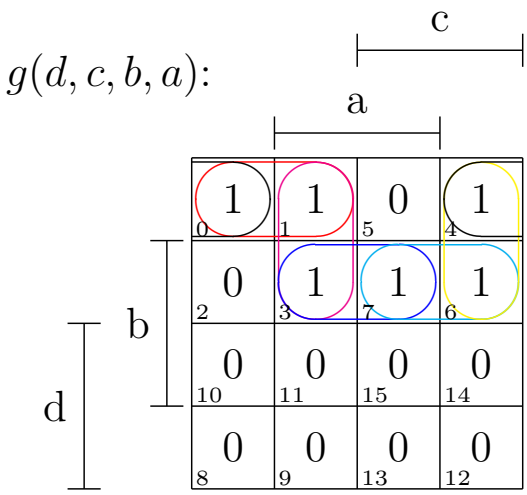


Abbildung 1: Schaltnetz

4. Existenz von g 2 P.
   Ja:

$g(d, c, b, a)$:



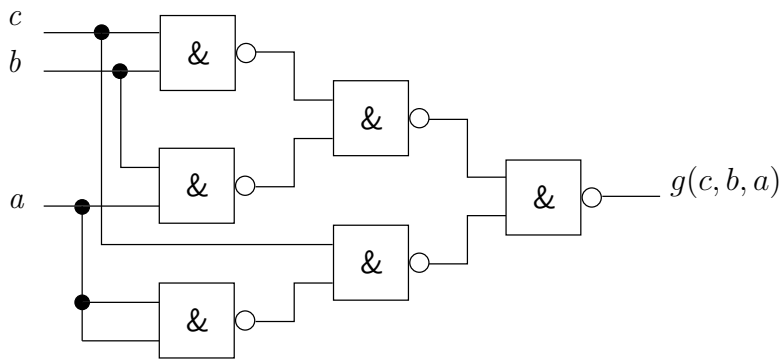
Alle eingezeichneten Blöcke sind Primimplikanten. Jedoch ist keines davon ein Kernprimimplikant. g ist zudem offensichtlich nicht die Nullfunktion.

Aufgabe 2 *Schaltfunktionen, CMOS-Technologie*(10 Punkte)

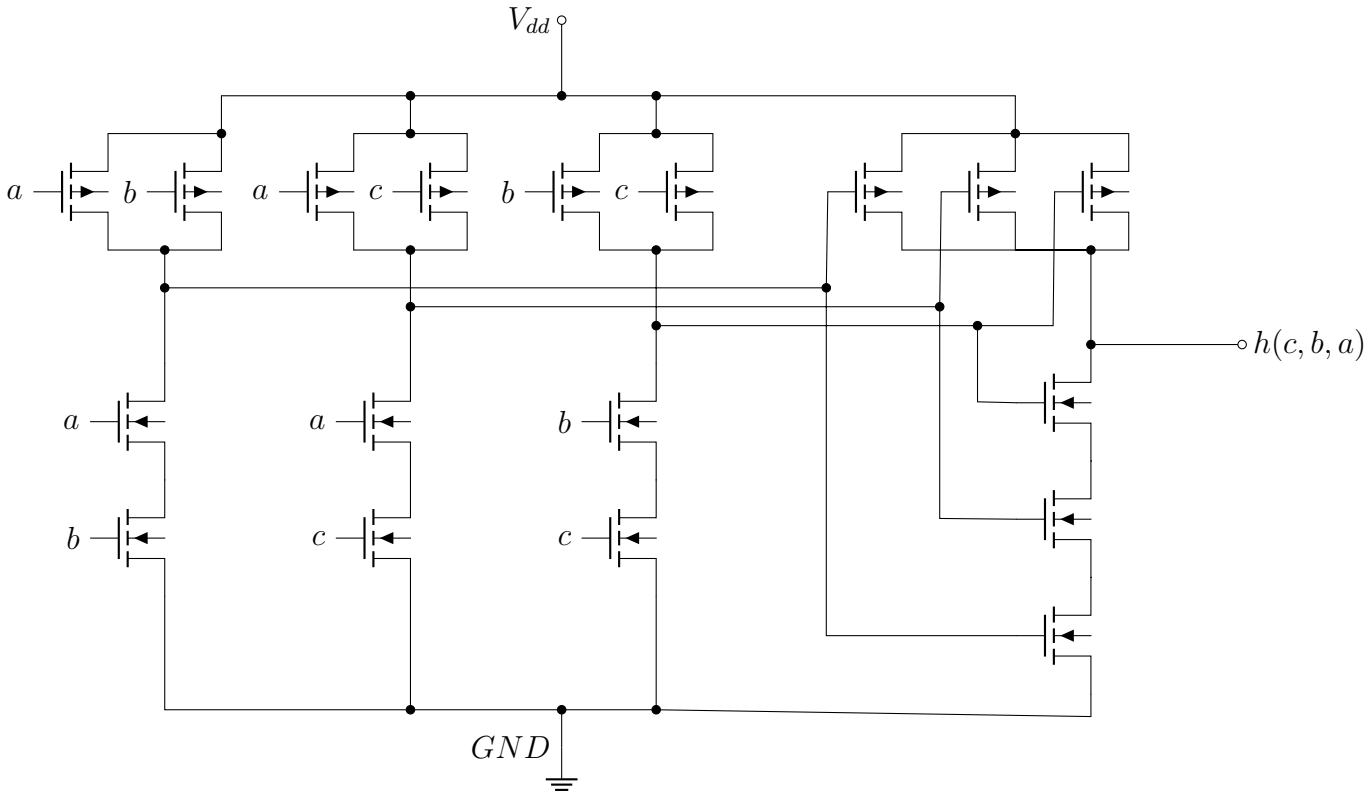
1. Realisierung von $g(c,b,a)$ mit NAND-Gattern: 5 P.

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

Schaltbild:



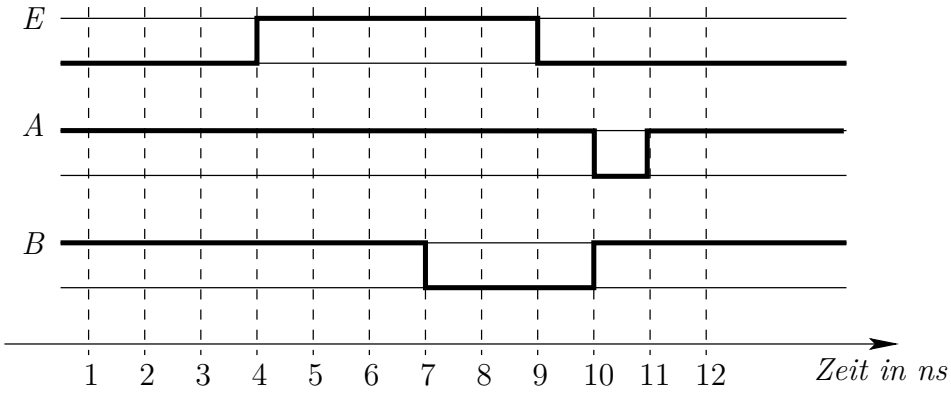
2. CMOS-Transistorschaltung von $h(c, b, a)$: 3 P.



3. CMOS-Transistorschaltung ist nicht geeignet, weil die Belegung $b\ a\ = 11$ zu einem Kurzschluss führt. 2 P.

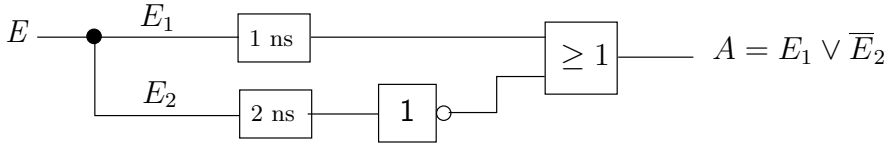
Aufgabe 3 *Laufzeiteffekte* (6 Punkte)

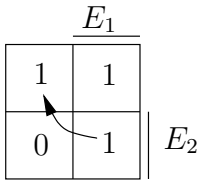
1. Verlauf von A — 2 P.



2. Hasardfehler: Statischer 1–Strukturhasard — 2 P.

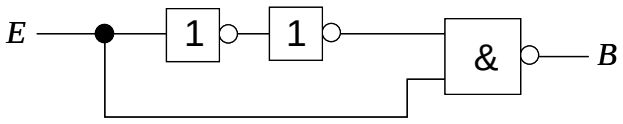
   Begründung:





   Der Hasardfehler 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.

3. Schaltnetz für das Signal B: — 2 P.

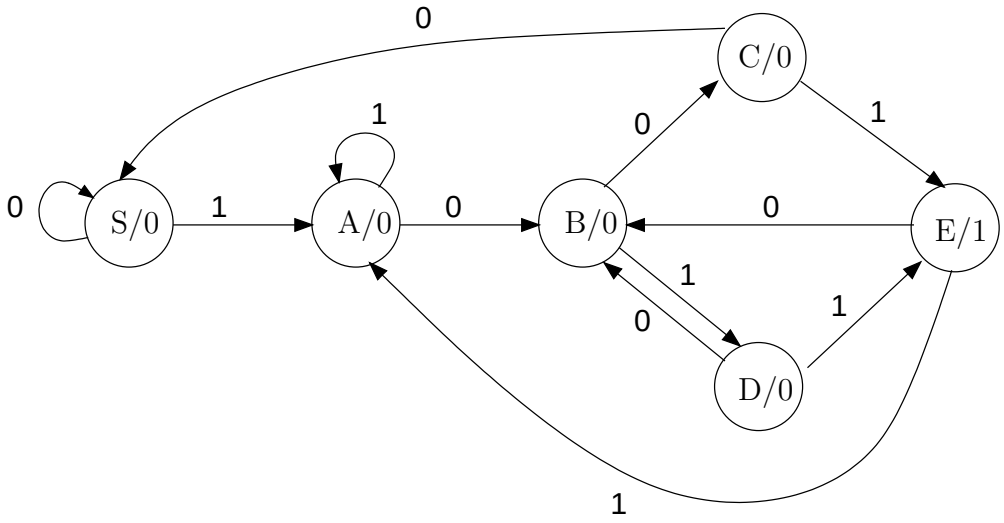


Aufgabe 4 *Schaltwerke* (11 Punkte)

1. Unterschied zwischen einem Mealy- und einem Moore-Automaten: 1 P.

   Beim Mealy-Automaten hängt die Ausgabe vom aktuellen Zustand und von der Eingabe ab. Beim Moore-Automaten hängt die Ausgabe nur vom aktuellen Zustand ab.

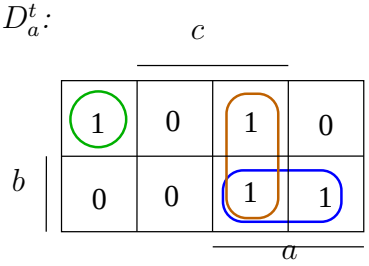
2. Moore-Automatengraph: 6 P.



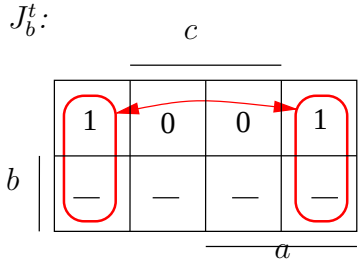
3. DMF der Ansteuerfunktionen der Flipflops: 4 P.

Zustand			Folgezustand			Ansteuerfunktionen der Flipflops			
a^t	b^t	c^t	a^{t+1}	b^{t+1}	c^{t+1}	D_a^t	J_b^t	K_b^t	T_c^t
0	0	0	1	1	1	1	1	–	1
0	0	1	0	0	0	0	0	–	1
0	1	0	0	0	1	0	–	1	1
0	1	1	0	1	0	0	–	0	1
1	0	0	0	1	1	0	1	–	1
1	0	1	1	0	0	1	0	–	1
1	1	0	1	0	1	1	–	1	1
1	1	1	1	1	0	1	–	0	1

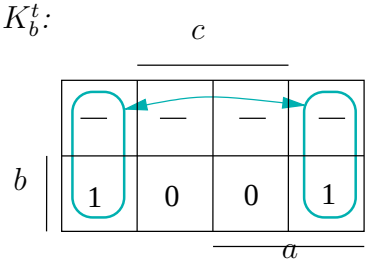
D_a^t:



J_b^t:



K_b^t:



Man erhält:

$$
\begin{aligned}
D_a^t &= b\,a \vee c\,a \vee \overline{c}\,\overline{b}\,\overline{a} \\
J_b^t &= \overline{c} \\
K_b^t &= \overline{c} \\
T_c^t &= 1
\end{aligned}
$$

Aufgabe 5 *Rechnerarithmetik & Codes* (11 Punkte)

1. Dezimalwert der Belegung 1001 1000: 1 P.

$$Z \;=\; (-1)^1 \cdot 2^{1-3} \cdot (1,1000)_2 \;=\; -0,011_2 \;=\; -(0,25+0,125)_{10} \;=\; -0,375$$

2. Größte Dezimalzahl: 1 P.

$$Z \;=\; (-1)^0 \cdot 2^{7-3} \cdot (1,1111)_2 \;=\; 11111_2 = 31_{10}$$

3. Kleinste positive Dezimalzahl: 1 P.

$$Z \;=\; (-1)^0 \cdot 2^{0-3} \cdot (1,0000)_2 \;= 0,001_2 \;=\; 0,125_{10}$$

4. Nichtdarstellbare Zahl: Null 1 P.

5. $N_4 = (123,02)_4$ 2 P.

$$\begin{aligned} N_{10} &= 1 \cdot 4^2 + 2 \cdot 4^1 + 3 \cdot 4^0 + 0 \cdot 4^{-1} + 2 \cdot 4^{-2} \\ &= 16 + 8 + 3 + 0 + 0,125 \\ &= 27,125_{10} \end{aligned}$$

$$\begin{aligned} N_{16} &= 1 \cdot 4^2 + 11 \cdot 4^0 + 2 \cdot 4^{-2} \\ &= 1 \cdot 16^1 + 11 \cdot 16^0 + 2 \cdot 16^{-1} \\ &= 1B,2_{16} \end{aligned}$$

oder

$$N_2 \;=\; (01\ 10\ 11,00\ 10)_2 \;=\; (0001\ 1011,0010)_2 \;=\; 1B,2_{16}$$

6. Gray-Code: einschrittig, zyklisch 1 P.
Ausführung arithmetischer Operationen im Gray-Code ist schwierig, weil die Stellen des Codes keine feste Stellenwertigkeit besitzen.

7. BCD-Arithmetik und Dual-Arithmetik: 1 P.
Die BCD-Arithmetik ist genauer (Darstellung von periodischen Brüchen). Sie ist aber langsamer und benötigt mehr Speicherplatz, da sie mit Tetraden arbeitet.

8. $x = 31$ $r = 2$ Dezimalwert $= 16_{10}$ 3 P.

Aus $(100)_{r+2} = (24)_{r+4}$ folgt: $1 \cdot (r+2)^2 = 2 \cdot (r+4)^1 + 4(r+4)^0 \;\Rightarrow\; r = 2$
Durch Einsetzen von r in die Gl. $x = (r+2)^2$ erhält man: $x = 16_{10} = (31)_5$

Aufgabe 6 *MIMA-Architektur* (5 Punkte)

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

Aufgabe 7 *MIPS-Assembler* (9 Punkte)

1. MIPS-Assembler: 3 P.

   (a)
   ```
   bne $s4, $s3, label
   add $s5, $s4, $s3
   label:   ...
   ```

   (b)
   ```
   beq $s4, $s3, label1
   add $s5, $s3, $s4
   j label2
   label1:  sub $s5, $s3, $s4
   label2:  ....
   ```

   (c)
   ```
   slt $s5, $s3, $s4
   ```

2. Register- und Speicherinhalte nach der Ausführung : 4 P.

Registersatz

Register	Inhalt
$t0	0x16
$t1	**0x40**
$t2	0x24
$t3	**0x28**
$t4	**0x56**

Hauptspeicher

Adresse	Inhalt
$0x20	0x10
$0x24	0x30
$0x28	**0x56**
$0x2C	0x50
$0x30	0x60

3. Der Arithmetik-Prozessor hat 32 Bit breite Fließkomma-Register. Bei Arithmetik-Operationen mit doppelter Genauigkeit (64-Bit) werden zwei Register für einen Operanden verwendet. Die Nummerierung startet bei 0 und somit gilt zur Vermeidung von Überschneidungen eine gerade Registernummer. 2 P.

Aufgabe 8 *Pipelining* (10 Punkte)

1. Aufgaben der einzelnen Pipeline-Stufen der DLX-Pipeline für bedingte Sprünge mit PC-relativer Adressierung:
   In der Instruction Fetch-Stufe wird der Befehl aus dem Speicher geladen. Die Instruction-Decode-Stufe erzeugt die prozessorinternen Steuersignale und leitet den vorzeichenerweiterten Operanten und den PC sowie den Vergleichsregisterinhalt weiter. In der Execute-Stufe wird einerseits entschieden, ob der Sprung genommen wird und andererseits aus dem PC und dem vorzeichenerweiterten Operanten der neue PC berechnet. Falls der Sprung genommen wird, wird in der MEM-Stufe der alte PC vom neuen PC ersetzt. Im nächsten Takt kann in der IF-Stufe der Befehl von der Sprungzieladresse geladen werden. Die Write-Back-Stufe ist bei bedingten Sprüngen nicht weiter von Bedeutung. 2 P.

2. (a) Datenabhängigkeiten: 5 P.

   - Echte Abhängigkeiten *(True Dependence)*

     $S_1 \rightarrow S_3$ ($\$t1$)
     $S_2 \rightarrow S_3$ ($\$t2$) $S_2 \rightarrow S_4$ ($\$t2$) $S_2 \rightarrow S_6$ ($\$t2$) $S_2 \rightarrow S_8$ ($\$t2$)
     $S_3 \rightarrow S_6$ ($\$t3$) $S_3 \rightarrow S_9$ ($\$t3$)
     $S_4 \rightarrow S_7$ ($\$t4$)

   - Gegenabhängigkeiten *(Anti-Dependence)*: $S_1 \rightarrow S7$ ($1000(\$zero)$), $S2 \rightarrow S8$ ($1000(\$zero)$)
   - Ausgabe-Abhängigkeiten *(Output Dependence)*: $S_5 \rightarrow S_6$ ($\$t5$)

   (b) Behebung der Konflikte: 3 P.

```
S1:    lw    $t1, 1000($zero)
S2:    lw    $t2, 1004($zero)
NOP
NOP
S3:    add   $t3, $t2, $t1
S4:    addi  $t4, $t2, 8
S5:    subi  $t5, $zero, 2
S6:    and   $t5, $t3, $t2
S7:    sw    $t4, 1000($zero)
S8:    sw    $t2, 1004($zero)
S9:    sw    $t3, 1008($zero)
```

Aufgabe 9 *Cache-Speicher* (12 Punkte)

1. (a) Anzahl der Cache-Einträge: 1 P.

$$\frac{\text{Kapazität}}{\text{Blockgröße}} = \frac{256 \text{ KiByte}}{2^4 \text{ Byte}} = \frac{2^8 \text{ KiByte}}{2^4 \text{ Byte}} = 16 \text{ Ki Einträge}$$

(b) Cache-Organisation:
12 Bit Index-Feld ⇒ Es lassen sich $2^{12} = 4$ Ki Sätze im Cache adressieren 2 P.

$$\text{Assoziativität} = \frac{16 \text{ Ki}}{4 \text{ Ki}} = 4$$

Der Cache ist als 4-fach assoziativer Speicher (*4-way set associative*) organisiert

2. Speicherbedarf: 3 P.
Für jede Zeile sind (Tag + 2 Statusbit + Daten pro Zeile) Bits erforderlich.

- Daten pro Zeile 16 Byte = 128 Bit
- Tag = 32 - 6 - 4 = 22 Bit (6 Bit als Satzindex und 4 Bit als Byte-Offset)

Speicherbedarf für eine Zeile: $22 + 2 + 128$ Bit = 152 Bits = 19 Bytes

Speicherbedarf für den gesamten Cache: $152 \cdot 64 \cdot 9$ = 87552 Bits = 10944 Bytes

3. 6 P.

Adresse **Adresse**	**64** 1_00_0000	**32** 0_10_0000	**64** 1_00_0000	**0** 0_00_0000	**112** 1_11_0000	**64** 1_00_0000	**128** 10_00_0000	**48** 0_11_0000	**240** 11_11_0000	**0** 0_00_0000
read/write	r	r	r	r	w	w	r	r	r	w
Index	0	2	0	0	3	0	0	3	3	0
Tag	1	0	1	0	1	1	2	0	3	0
Hit/Miss	Miss	Miss	Hit	Miss	Miss	Hit	Miss	Miss	Miss	Miss
write back?	nein	nein	nein	nein	nein	nein	nein	nein	ja	ja

Aufgabe 10 *Speicher & Speicherverwaltung* (9 Punkte)

1. Anordnung von Speicher verschiedener Technologie und damit auch verschiedener Eigenschaften nach absteigenden Zugriffsgeschwindigkeiten und aufsteigenden Speicherkapazitäten. In der Hierarchie nehmen die Zugriffszeiten und die Kapazität in den unteren Ebenen der Speicherhierarchie zu, während der Preis pro Bit sinkt. [2 P.]

2. 512×8-Organisation : 512 Zellen mit 8-Bit Wörter ⇒ 512 Zellen müssen adressiert werden. Dazu sind 9 Adressleitungen erforderlich. [1 P.]

3. Ein RAM-Baustein der Organisation 8k×1 kann 8k Wörter mit einer Wortbreite von 1 Bit realisieren. Somit sind es 16 RAM-Bausteine der Organisation 8k×1 notwendig, um einen Speicher mit einer Kapazität von 8k Wörter und einer Wortbreite von 16 Bit zu realisieren. [1 P.]

4. ROM-Baustein der Speicherkapazität von 2048 Bits und 7 Adressleitungen ⇒ Es können 128 Zellen adressiert werden ⇒ 128×16-Organisation [1 P.]

5. Physikalische Adresse von 3112: [4 P.]

   $3112/512 = 6 +$ Rest $40 \Rightarrow$ virtuelle Seitennummer 6

   Aus der Tabelle ⇒ physikalische Seitennummer 8

   ⇒ physikalische Adresse ist: $8 * 512 + 40 = 4136$ oder $3112 + 2 * 512 = 4136$

   Physikalische Adresse von 1417:

   $1417/512 = 2 +$ Rest $393 \Rightarrow$ virtuelle Seitennummer 2

   Aus der Tabelle ⇒ physikalische Seitennummer 7

   ⇒ physikalische Adresse ist: $7 * 512 + 393 = 3977$ oder $1417 + 5 * 512 = 3977$