KIT-Fakultät für Informatik
Prof. Dr. Mehdi Tahoori, Prof. Dr. Wolfgang Karl

# Musterlösungen zur Klausur

Digitaltechnik und Entwurfsverfahren (TI-1)

und

Rechnerorganisation (TI-2)

am 25. Februar 2019, 13:30 – 15:30 Uhr

| Name: | Vorname: | Matrikelnummer: |
|---|---|---|
| Bond | James | 007 |

| **Digitaltechnik und Entwurfsverfahren (TI-1)** | |
|---|---|
| Aufgabe 1 | 11 von 11 Punkten |
| Aufgabe 2 | 12 von 12 Punkten |
| Aufgabe 3 | 11 von 11 Punkten |
| Aufgabe 4 | 6 von 6 Punkten |
| Aufgabe 5 | 5 von 5 Punkten |

| **Rechnerorganisation (TI-2)** | |
|---|---|
| Aufgabe 6 | 6 von 6 Punkten |
| Aufgabe 7 | 11 von 11 Punkten |
| Aufgabe 8 | 10 von 10 Punkten |
| Aufgabe 9 | 14 von 14 Punkten |
| Aufgabe 10 | 4 von 4 Punkten |

| **Gesamtpunktzahl:** | 90 von 90 Punkten |
|---|---|

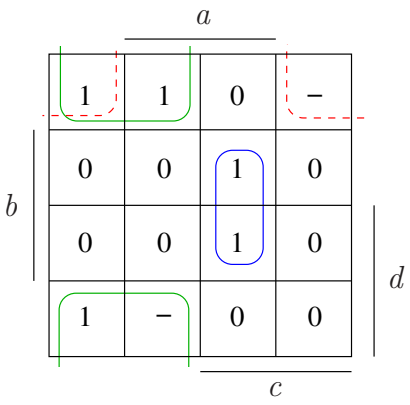
| | **Note:** **1,0** |
|---|---|

# **Aufgabe 1** *Schaltfunktionen* (11 Punkte)

1. DMF:

$$y_{DMF} = c\,b\,a \;\vee\; \overline{c}\,\overline{b}$$

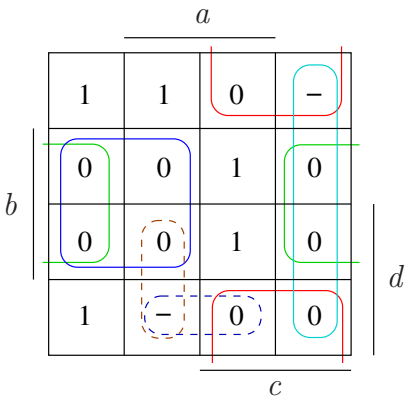
2 P.




2. KMF:

3 P.

$$\begin{aligned} y_{KMF} &= (c \vee \overline{b}) \cdot (\overline{c} \vee b) \cdot (\overline{b} \vee a) \qquad \text{oder} \\ y_{KMF} &= (c \vee \overline{b}) \cdot (\overline{c} \vee b) \cdot (\overline{c} \vee a) \end{aligned}$$

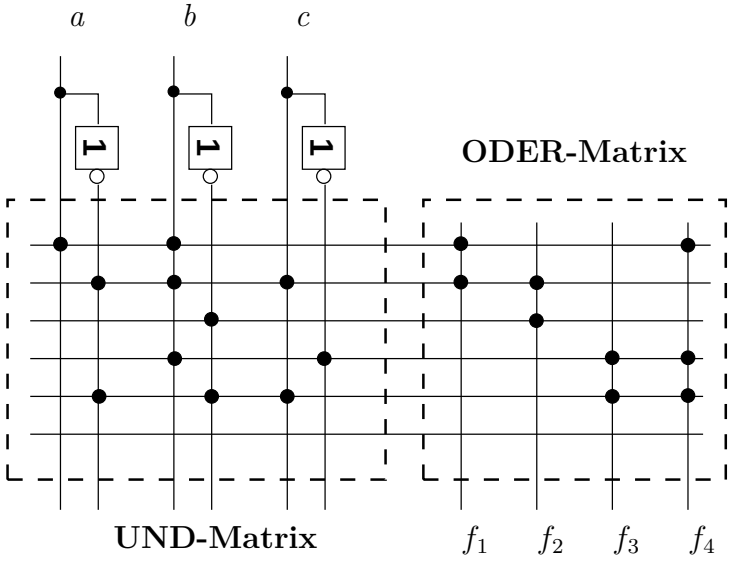


3. 2 P.

| Produktterm | **X** | Erklärung |
|---|---|---|
| $\overline{d}\,\overline{c}\,b$ | | überdeckt werder $m_0$ noch $m_{10}$ |
| $c\,\overline{b}$ | **X** | $\overline{d}\,\overline{c}\,\overline{a}$ wird kein Kernpimimplikant mehr |
| $d\,\overline{b}\,a$ | | Nicht angrenzend an den beiden Kernpimimplikanten. |
| $c\,b\,\overline{a}$ | **X** | $\overline{c}\,b\,\overline{a}$ wird kein Kernpimimplikant mehr. |

4. PLA: Bündelminimierung der Funktionen: 4 P.

$$f_1 = b\,a \vee c\,b\,\overline{a} \qquad f_2 = \overline{b} \vee c\,b\,\overline{a}$$
$$f_3 = \overline{c}\,b \vee c\,\overline{b}\,\overline{a} \qquad f_4 = \overline{c}\,b \vee b\,a \vee c\,\overline{b}\,\overline{a} = f_3 \vee b\,a$$



# **Aufgabe 2** *Minimierungsverfahren* (12 Punkte)

1. KMF von $f(c,b,a)$: 3 P.

$$\begin{aligned} f(c,b,a) &= \overline{c}\,(\overline{b}\,a \;\vee\; b\,\overline{a}\,\overline{c} \;\vee b\,a\,c) \;\vee\; c\,(\overline{a} \;\vee\; c\,a) \\ &= \overline{c}\,\overline{b}\,a \;\vee \overline{c}\,b\,\overline{a} \;\vee\; c\,a \;\vee\; c\,\overline{a} \\ &= \overline{c}\,\overline{b}\,a \;\vee \overline{c}\,b\,\overline{a} \;\vee\; c \\ &= \overline{b}\,a \;\vee\; b\,\overline{a} \;\vee\; c \\ &= (b\,\overline{b} \;\vee a\,\overline{a} \;\vee\; b\,\overline{a} \;\vee\; \overline{b}\,a) \;\vee\; c \\ &= (\overline{b} \;\vee\; \overline{a})\,(b \;\vee a) \;\vee c \\ &= (c \;\vee\; \overline{b} \;\vee\; \overline{a})\,(c \;\vee b \;\vee a) \end{aligned}$$

4 P.

2.

| Nr. | gebildet aus | Würfel c | b | a | gestrichen wegen |
|---|---|---|---|---|---|
| 1 | | 0 | 1 | 0 | $\subset 6$ |
| 2 | | 1 | 1 | 0 | $\subset 6$ |
| 3 | | 1 | 0 | 1 | $\subset 9$ |
| 4 | | 0 | 1 | 1 | $\subset 7$ |
| 5 | | 1 | 1 | 1 | $\subset 7$ |
| 6 | 2,1 | – | 1 | 0 | $\subset 8$ |
| 7 | 5,4 | – | 1 | 1 | $\subset 8$ |
| 8 | 7,6 | – | 1 | – | **Primimplikant** |
| 9 | 8,3 | 1 | – | 1 | **Primimplikant** |

Die Primimplikanten sind: $b$ und $c\,a$ 2 P.

3. (a) $h(d,c,b,a)$ ist unvollständig definiert.

Mögliche Begründungen:

- Der Primimplikant F überdeckt drei Minterme, jedoch überdecken Primimplikanten bei vollständig definierten Funktionen $2^n$ Minterme.
- $A$ (Minterm 4) wäre kein Primimplikant, weil er in F enthalten ist und somit umschließt $A$ auch don't care Stellen.
- $B$ (Minterm 8) wäre kein Primimplikant, wenn die Funktion vollständig definiert ist, da $B$ dann in C (Mintermen 8 und 9) enthalten wäre und somit $B$ don't care Stellen umschließt.

(b) DMF von $h(d, c, b, a)$ 3 P.

| | 4 | 5 | 6 | 8 | 9 | 10 | 13 |
|---|---|---|---|---|---|---|---|
| A | × | | | | | | |
| B | | | | × | | | |
| C | | | | × | × | | |
| D | | | | × | | × | |
| E | | | | | × | | × |
| F | × | × | × | | | | |
| G | | × | | | | | × |

- Kern-Primimplikanten: F und D → Spalten mit den Mintermen 4, 5, 6, 8 und 10 werden gestrichen → reduzierte Tabelle:

| | 9 | 13 |
|---|---|---|
| A | | |
| B | | |
| C | × | |
| E | × | × |
| G | | × |

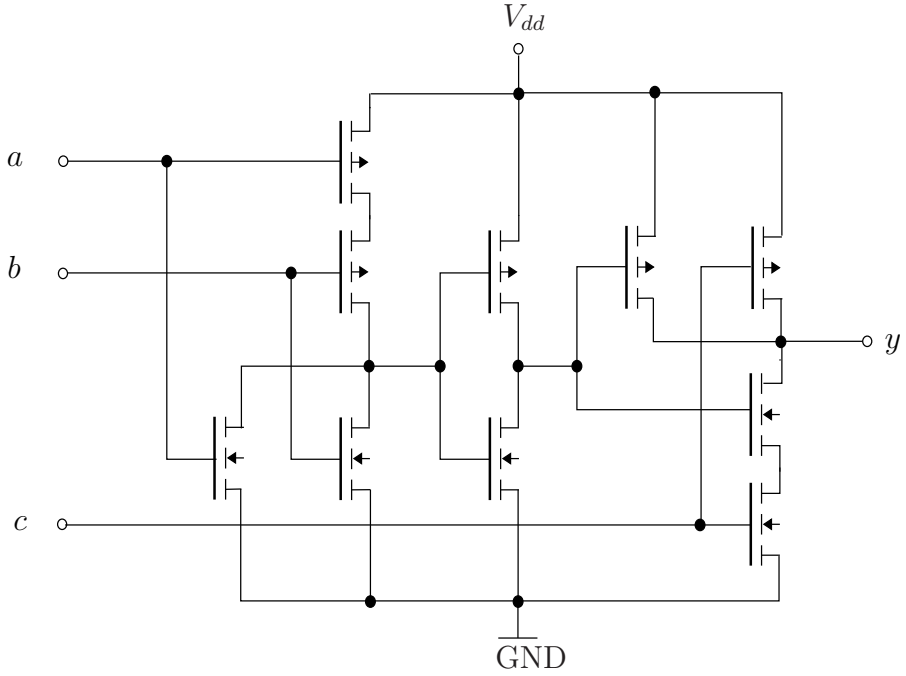
- Primimplikant E dominiert C und G (überdeckt sowohl 9 als auch 13)
- A und B sind entbehrlich.
  DMF:

$$h(d, c, b, a) \;=\; F \;\vee D \;\vee E \;=\; \overline{d}\, c \;\vee d\, \overline{c}\, \overline{a} \;\vee\; d\, \overline{b}\, a$$

# **Aufgabe 3** *Spezielle Bausteine* (11 Punkte)

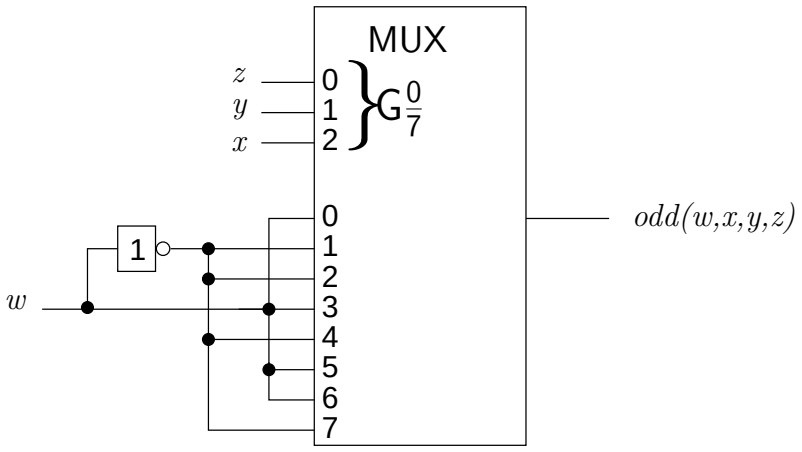
1. CMOS-Transistor-Schaltbild von $f(c,b,a)$: 4 P.

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



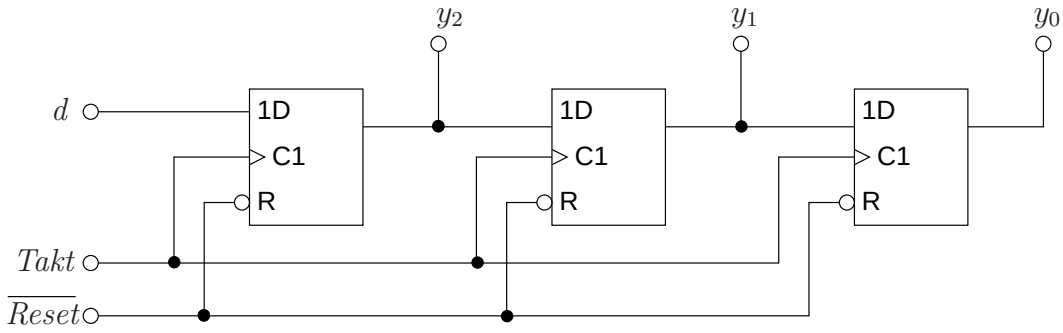

2. Schaltnetz von $p = \mathrm{odd}(w,x,y,z)$: 4 P.

$$p = \mathrm{odd}(w,x,y,z) = \mathrm{MINt}(1,2,4,7,8,11,13,14)$$



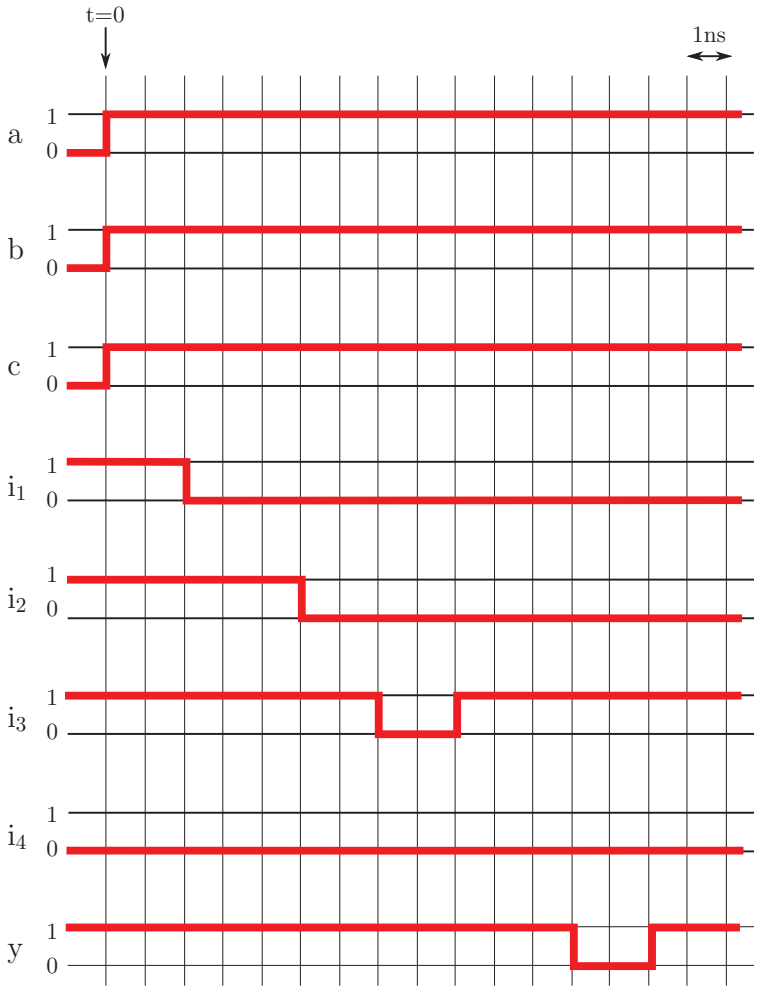
Hinweis: high-aktives Reset ist ebenfalls eine gültige Lösung.

3. 3-Bit Schieberegister: 3 P.




## Aufgabe 4 *Laufzeiteffekte* (6 Punkte)

1. Zeitdiagramm: 3 P.



2. Hasardfehler (falls ja, Analyse): 3 P.

   Ja, es tritt ein Hasardfehler auf, denn $y$ ist zu Beginn und Ende des Übergangs 1, wechselt während des Übergangs jedoch kurzzeitig auf 0.

   Betrachtet man nun die möglichen Folgen von Funktionswerten beim Übergang $(0,0,0) \rightarrow (1,1,1)$, stellt man fest, dass die Folge von Funktionswerten bei bestimmten Eingabewechseln (z.B. $a$, $b$, $c$) nicht monoton ist (siehe KV-Diagramm auf nächster Seite). Somit handelt es sich um einen Funktionshasard.

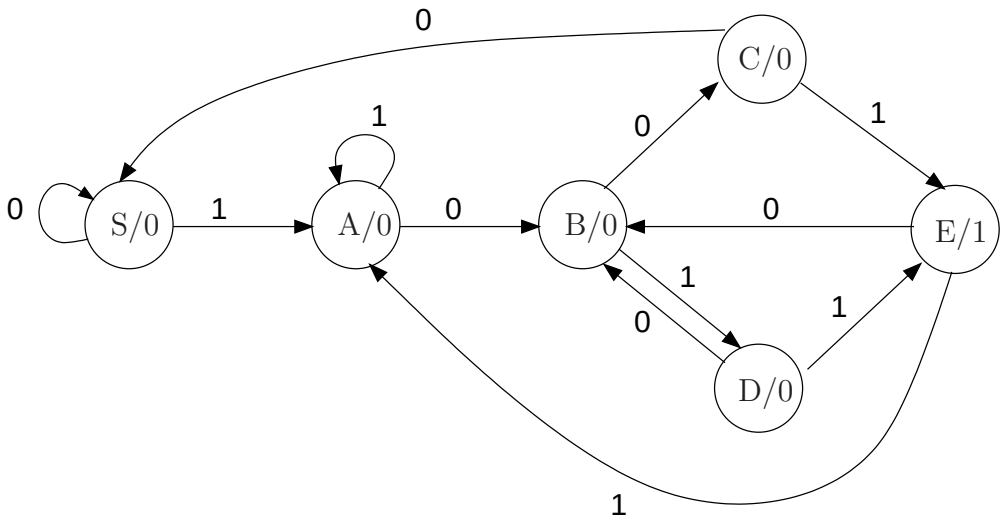
   Insgesamt ist der Hasard also als 1-statischer Funktionshasard zu klassifizieren.

$y(c,b,a)$:

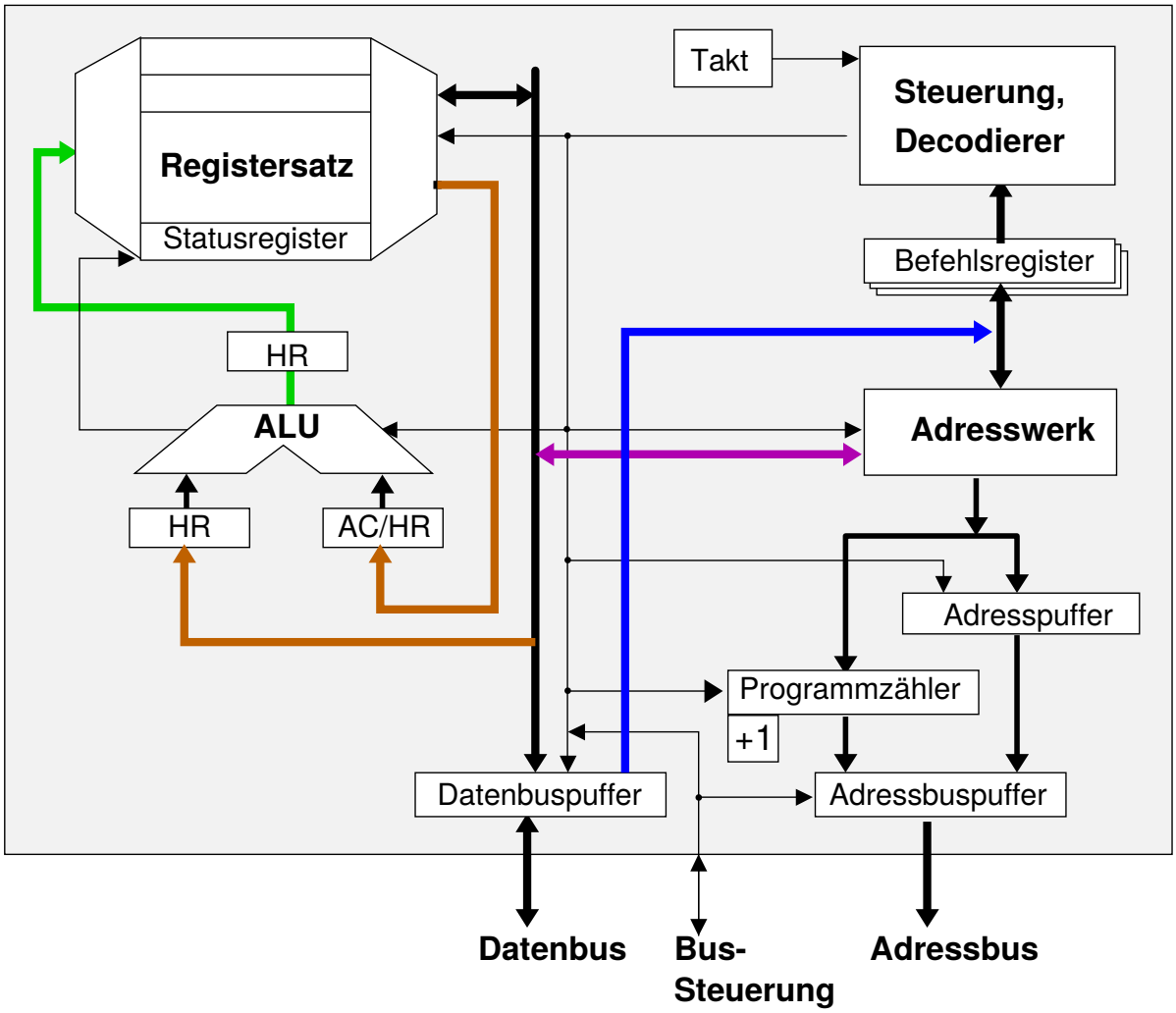
| $b$ \ $c$ | | $a$ | | |
|---|---|---|---|---|
| | 1 | 1 | 1 | 1 |
| | 0 | 1 | 3 | 2 |
| | 4 | 5 | 7 | 6 |
| | 1 | 1 | 0 | 1 |

## Aufgabe 5 *Schaltwerke* (5 Punkte)

1. Automatengraph: 5 P.



## Aufgabe 6 *Mikroprozessor* (6 Punkte)




## Aufgabe 7 *C, MIPS–Assembler & MIMA* (11 Punkte)

1. MIPS-Programmstücke in C-Sprache 6 P.

(a)
```
a = b = c = 0;
if (i < 5) {
   a = 1;
   b = 2;
   c = 3;
}
d = 5;
```

(b)
```
a = b = c = 0;
```

```
if (i < 5) {
   a = 1;
   b = 2;
   c = 3;
}
else {
   a = 4;
   b = 5;
   c = 6;
}
d = 5;
```

(c)
```
sum = 0;
for (i = 0; i < 100; i++)
  sum = sum + a[i];
```

Hinweis: Deklarationen und Definitionen von Variablen und Arrays sind nicht nötig.

2. Mikroprogamm für ADD a bei der MIMA in Register-Transfer-Schreibweise: 5 P.

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

} Lese-Phase

6. Takt: Dekodiere Befehl } Dekodier-Phase

7. Takt: $\mathsf{IR} \rightarrow \mathsf{SAR}; \mathsf{R} = 1$
8. Takt: $\mathsf{Akku} \rightarrow \mathsf{X}; \mathsf{R} = 1$
9. Takt: $\mathsf{R} = 1$
10. Takt: $\mathsf{SDR} \rightarrow \mathsf{Y}$;
11. Takt: $\mathsf{ALU}$ auf $\mathsf{ADD}$ $(C_2C_1C_0 = 001)$
12. Takt: $\mathsf{Z} \rightarrow \mathsf{Akku}$

} Ausführungsphase

# Aufgabe 8 *Pipelining* (10 Punkte)

1. Datenabhängigkeiten: 3 P.

   - Echte Abhängigkeiten (*True Dependence* $\delta^t$):

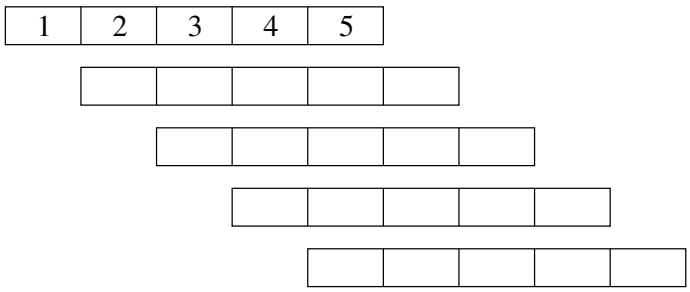
     `S1-S3 (R5)` `S1-S4 (R5)` `S2-S4 (R1)` `S3-S4 (R5)`

   - Ausgabeabhängigkeiten (*Output Dependence* $\delta^o$):

     `S1-S3 (R5)` `S2-S4 (R1)`

   - Gegenabhängigkeit (*Anti Dependence* $\delta^a$): Keine

2. Pipelinekonflikte: 3 P.

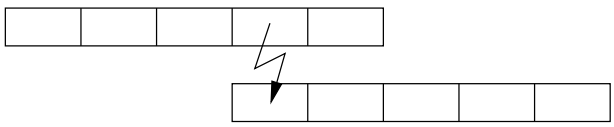



   Bei echten Abhänigigkeiten müssen mindestens 3 Befehle dazwischen liegen. ⇒ Alle echten Datenabhänigigkeiten führen zu Konflikten.

3. Beseitigung der Konflikte: 2 P.

```
S1:   MULT    R5, R2, R2
S2:   MULT    R1, R4, R2
      NOP
      NOP
S3:   ADD     R5, R5, R3
      NOP
      NOP
      NOP
S4:   ADD     R1, R1, R5
```

4. Problem: 2 P.



   Es entsteht ein Konflikt zwischen der Befehlshol-Phase und der Speicherzugriffs-Phase, da evtl. beide aus dem Cache lesen wollen.

# Aufgabe 9 *Cache-Speicher* (14 Punkte)

1. (a) Größe eines Cache-Blocks in Byte: 1 P.

   5 Bits Byte-Offset → Blockgröße: $2^5 = 32$ Byte

   (b) Kapazität des Cache-Speichers: 1 P.

   11 Bits Index → $2^{11}$ Sätze

   A2-Cache $2 \times 2^{11} = 2^{12}$ Cache-Blöcke mit je 32 Bytes

   Cache-Kapzität: $2^{12} \times 32 = 2^{12} \times 2^5$ Byte $= 2^{17}$ Byte $= 128$ KByte

   (c) Der insgesamt erforderliche Speicherbedarf: 2 P.

   Kapazität + (Tag-Länge + 2 Statusbits) × (Anzahl der Cache-Blöcke)
   128 KByte + $(16 + 2) \cdot 2^{12}$ Bit = 128 KByte + $2 \cdot 2^{12}$ Byte + $2 \cdot 2^{12}$ Bit =
   128 KByte + 8 KByte + 1 KByte = 137 KByte

   (d) Zugriff auf die Adresse `0x00EF1A34`: 2 P.

   A2-Cache → Es wird ein Vergleich mit 2 Zeilen im durchgeführt.
   Satz-Index = 0001 1010 $001_2 = 209_{10}$
   → Der Vergleich wird mit den Zeilen 418 (0x1A2) und 419 (0x1A3) durchgeführt.

   4 P.

2.

| **Adresse** | **0x44** | **0xA0** | **0xC3** | **0x9E** | **0x66** | **0x2D** | **0x6B** | **0x49** |
|---|---|---|---|---|---|---|---|---|
| **Index** | 4 | 2 | 4 | 1 | 6 | 2 | 6 | 4 |
| **Tag** | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| **read/write** | w | r | w | r | r | w | r | w |
| **Hit/Miss** | × | — | — | × | × | — | × | — |
| **write back?** | nein | nein | ja | nein | nein | nein | nein | ja |

3. Widerlegung durch Beispiel bei dem Aussage falsch ist: 4 P.
   Angenommen wir betrachten einen 2-fach und 4-fach satzassoziativen Cache, welcher jeweils aus 4 Cachezeilen besteht, sowie die Speicherzugriffsfolge: $ABCD(ABEFCD)^n$. Für den 2-fach satzassoziativen Cache werden die Speicherzugriffe A, B, E, F auf den ersten Satz und C und D auf den zweiten Satz abgebildet. Die Zustände der Caches verändern sich durch diese Folge wie folgt:

   2-fach assoziativ

| | | | | | |
|---|---|---|---|---|---|
| A | **A** | E | A | E | |
| B | **B** | F | B | F | |

| | | |
|---|---|---|
| C | **C** | **C** |
| D | **D** | **D** |

4-fach assoziativ

| | | | |
|---|---|---|---|
| A | **A** | C | E |
| B | **B** | D | F |
| C | E | A | C |
| D | F | B | D |

Die fett makierten Buchstaben geben einen Cache-Hit an. Der 2-fach satzassoziative Cache ermöglicht für die gewählte Sequenz 6 Hits, während der 4-fach satzassoziative Cache nur 2 Hits ermöglicht. Somit ist die Behauptung widerlegt!

## **Aufgabe 10** *Allgemeines* (4 Punkte)

1. Arithmetisches Pipelining:
   Lange Berechnung in Teilschritte zerlegen (in mehrere Pipelinestufen), wichtig bei komplizierteren arithmetischen Operationen, wie Multiplikation, Division, Fließkommaoperationen, ...

2. Zwei Eigenschaften einer superskalaren Pipeline:

   - Mehrere voneinander unabhängige Ausführungseinheiten
   - Zur Laufzeit werden pro Takt mehrere Befehle aus einem sequentiellen Befehlsstrom den Verarbeitungseinheiten zugeordnet und ausgeführt
   - Dynamische Erkennung und Auflösung von Konflikten

3. Mooresches Gesetz: Anzahl der Transistoren (Schaltkreiskomponenten), die auf einem IC integriert werden können, verdoppelt sich alle (ein bis) zwei Jahre