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

Musterlösungen zur Klausur

Digitaltechnik und Entwurfsverfahren & Rechnerorganisation

und

Technische Informatik I/II

am 25. Juli 2016, 14:00 – 16:00 Uhr

Name:	Vorname:	Matrikelnummer:
Bond	James	007

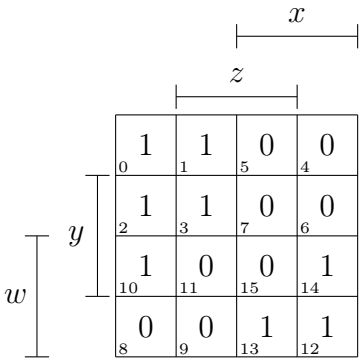
Digitaltechnik und Entwurfsverfahren/TI-1	
Aufgabe 1	10 von 10 Punkten
Aufgabe 2	11 von 11 Punkten
Aufgabe 3	9 von 9 Punkten
Aufgabe 4	7 von 7 Punkten
Aufgabe 5	8 von 8 Punkten
Rechnerorganisation/TI-2	
Aufgabe 6	8 von 8 Punkten
Aufgabe 7	13 von 13 Punkten
Aufgabe 8	12 von 12 Punkten
Aufgabe 9	9 von 9 Punkten
Aufgabe 10	3 von 3 Punkten

Gesamtpunktzahl:	90 von 90 Punkten

	Note: **1,0**

Aufgabe 1

1. KV-Diagramm:



2. Primimplikanten und Kernprimimplikanten:

   $\overline{w} \wedge \overline{x}$ (K) $\quad w \wedge x \wedge \overline{y}$ (K) $\quad w \wedge y \wedge \overline{z} \quad w \wedge x \wedge \overline{z} \quad \overline{x} \wedge y \wedge \overline{z}$

   (K: Kernprimimplikant)

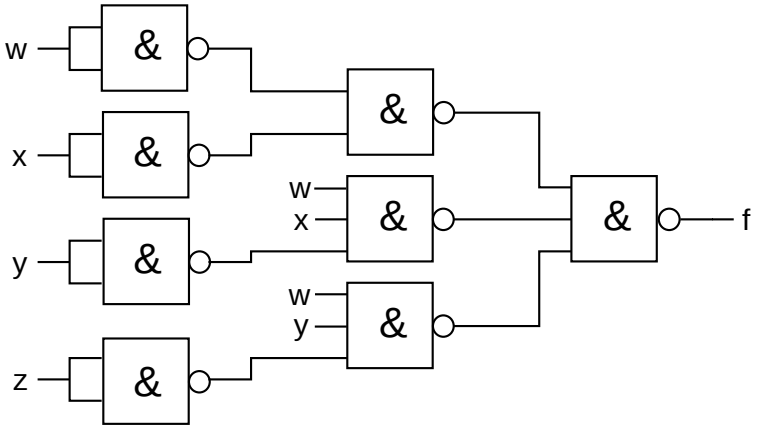
3. Disjunktive Minimalform:

   $(\overline{w} \wedge \overline{x}) \vee (w \wedge x \wedge \overline{y}) \vee (w \wedge y \wedge \overline{z})$

4. Schaltnetz mit NAND-Gattern:

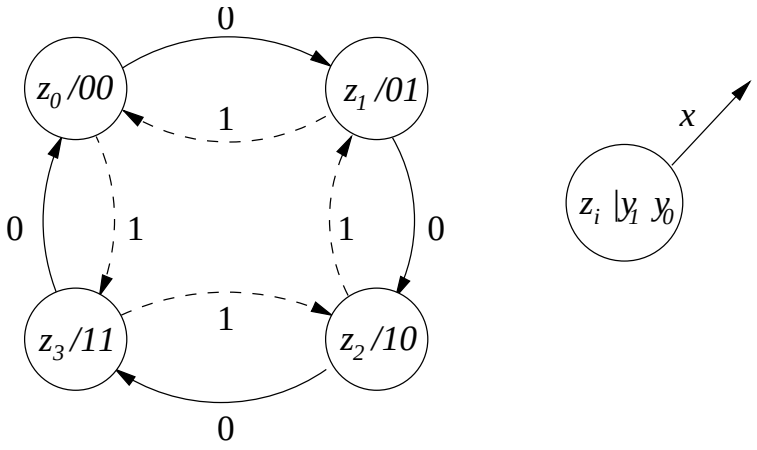
   Umwandlung von disjunktiver Form in NAND-Form:

$$\begin{aligned} f &= \overline{\overline{(\overline{w} \wedge \overline{x}) \vee (w \wedge x \wedge \overline{y}) \vee (w \wedge y \wedge \overline{z})}} \\ &= \overline{\overline{\overline{w} \wedge \overline{x}} \wedge \overline{w \wedge x \wedge \overline{y}} \wedge \overline{w \wedge y \wedge \overline{z}}} \\ &= \mathrm{NAND}_3(\mathrm{NAND}_2(\overline{w}, \overline{x}), \mathrm{NAND}_3(w, x, \overline{y}), \mathrm{NAND}_3(w, y, \overline{z})) \end{aligned}$$



Aufgabe 2

1. Automatengraph:



   Anzahl der erforderlichen Flipflops: 2

2. Kodierte Ablauftabelle:

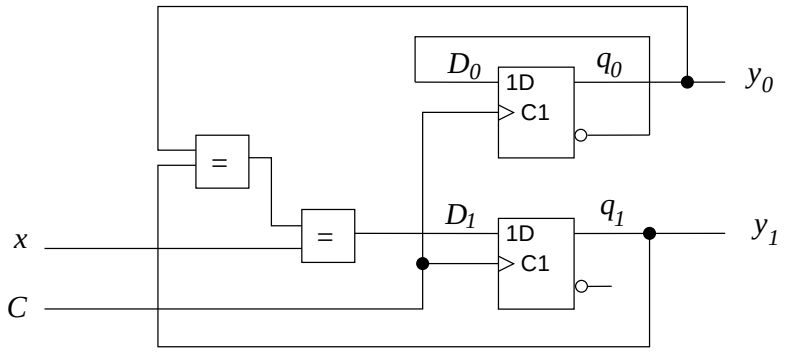
Eingabe	Zustand		Folgezustand		Ausgang		FF-Ansteuersignale	
x^t	q_1^t	q_0^t	q_1^{t+1}	q_0^{t+1}	y_1^t	y_0^t	D_1^t	D_0^t
0	0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	1	0
0	1	0	1	1	1	0	1	1
0	1	1	0	0	1	1	0	0
1	0	0	1	1	0	0	1	1
1	0	1	0	0	0	1	0	0
1	1	0	0	1	1	0	0	1
1	1	1	1	0	1	1	1	0

3. Ansteuerfunktionen der Flipflops:

$$
\begin{aligned}
D_1 &= \overline{x}\,\overline{q}_1 q_0 \vee \overline{x}\, q_1 \overline{q}_0 \vee x\,\overline{q}_1\overline{q}_0 \vee x\, q_1 q_0 \\
&= \overline{x}(\overline{q}_1 q_0 \vee q_1\overline{q}_0) \vee x(\overline{q}_1\overline{q}_0 \vee q_1 q_0) \\
&= \overline{x}(q_1 \not\leftrightarrow q_0) \vee x(q_1 \leftrightarrow q_0) \\
&= x \leftrightarrow (q_1 \leftrightarrow q_0)
\end{aligned}
$$

$$
\begin{aligned}
D_0 &= \overline{x}\,\overline{q}_1\overline{q}_0 \vee \overline{x}\, q_1\overline{q}_0 \vee x\,\overline{q}_1\overline{q}_0 \vee x\, q_1\overline{q}_0 \\
&= \overline{q}_1\overline{q}_0(\overline{x} \vee x) \vee q_1\overline{q}_0(\overline{x} \vee x) \;=\; (q_1 \vee \overline{q}_1)\overline{q}_0 = \overline{q}_0
\end{aligned}
$$

4. Schaltung des Schaltwerks:

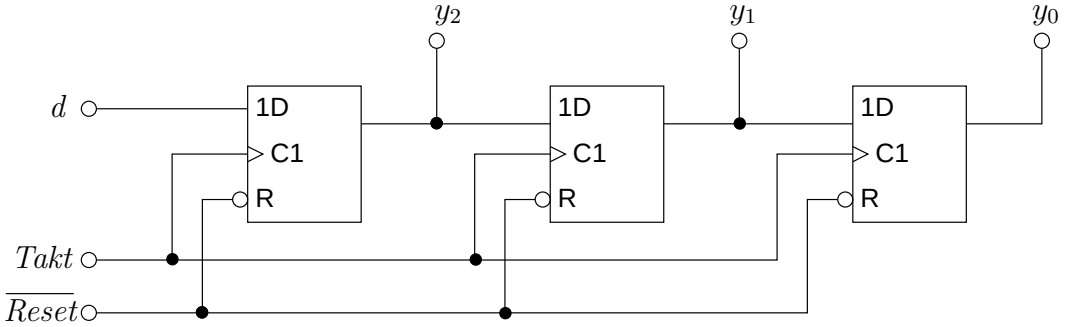


Aufgabe 3

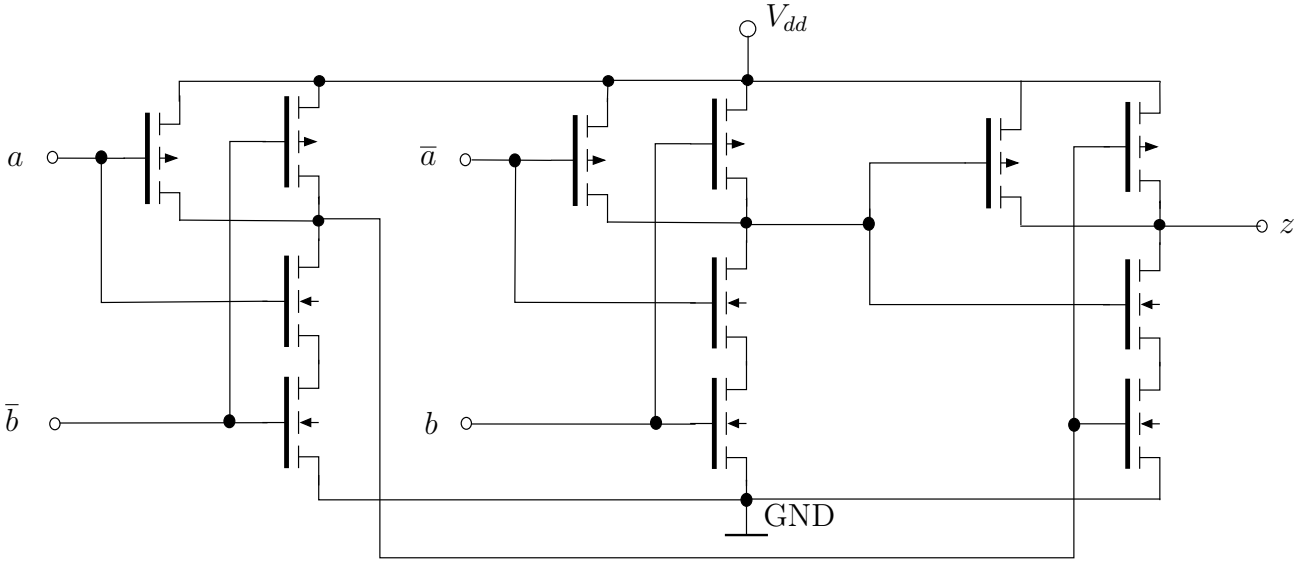
1. Schaltfunktion:

$$y = \overline{[\overline{a}\ b\ \vee\ a\ d]}\ \overline{d}\ (\overline{c}\ d\ \vee\ c)\ \vee \overline{[\overline{a}\ b\ \vee\ a\ d]}\ d\ (\overline{a}\ c\ \vee\ a\ c)\ \vee$$
$$[\overline{a}\ b\ \vee\ a\ d]\ \overline{d}\ (c)\ \vee\ [\overline{a}\ b\ \vee\ a\ d]\ d\ (b)$$

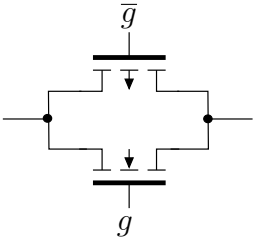
2. 3-Bit Schieberegister:



3. CMOS-Schaltnetz: $y = \overline{\overline{b}\ a\ \vee\ b\ \overline{a}} = (\overline{b}\ \overline{\wedge}\ a)\ \overline{\wedge}\ (b\ \overline{\wedge}\ \overline{a})$, {*alternativ:* $(b\ \overline{\wedge}\ a)\ \overline{\wedge}\ (\overline{b}\ \overline{\wedge}\ \overline{a})$}



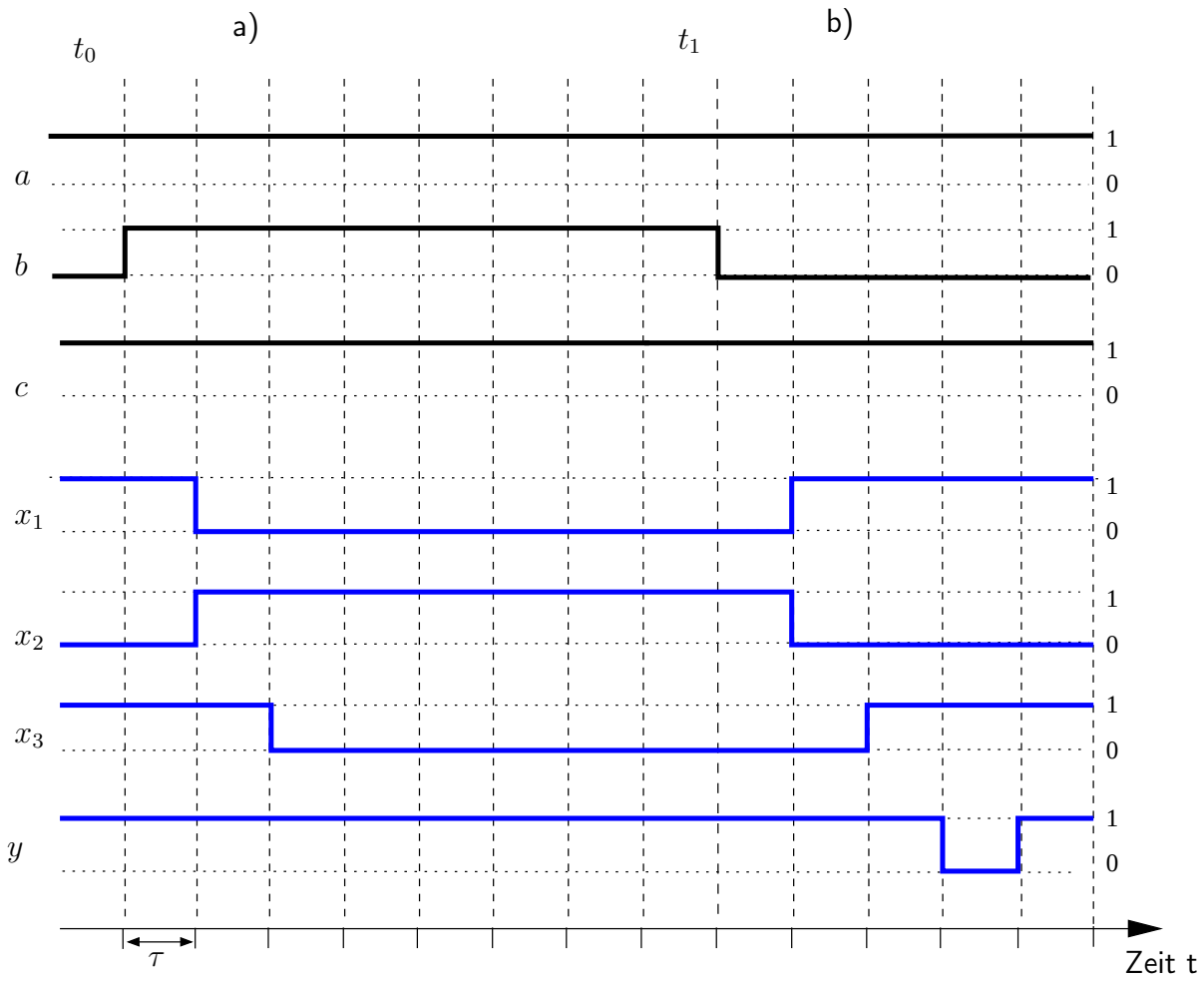
4. Transmission-Gate:



Die Transistoren müssen komplementär angesteuert werden (in Abbildung g und $\overline{g}$).

Aufgabe 4

1.



2. Typ des Fehlers und Behebungsmöglichkeit:

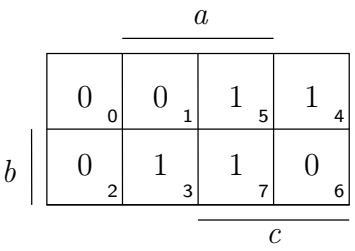
   Es tritt ein Hasardfehler beim Übergang $B_7 \rightarrow B_5$ zum Zeitpunkt t_1 auf.

   Es handelt sich hierbei um einen Übergang, bei dem nur eine Variablen b ihren Wert wechselt $\Rightarrow$ Der Übergang ist frei von Funktionshasards; der Hasardfehler tritt nicht aufgrund eines Funktionshasards auf und kann nur durch einen Strukturhasard bedingt sein $\Rightarrow$ **1-statischer Strukturhasard**.

   Behebung:

   - Satz von Eichelberger: Realisierung der Schaltfunktion als die Disjunktion aller Primimplikanten (Fehlender Primiplikant $c\,a$ in die Realisierung aufnehmen, d. h. $y = b\,a \vee c\,\bar{b} \vee c\,a$
   - Die beim Übergang konstant bleibenden Eingangsvariablen (a und c) über ein zusätzliches UND-Gatter verknüpfen und das Ergebnis mit dem Ausgang des Schaltnetzes ODER-verknüpfen.

3. Übergang mit einem statischen 1-Funktionshasard:



Beispiele für Übergänge mit statischem 1-Funktionshasard:
$B_3 \leftrightarrow B_4$, $B_3 \leftrightarrow B_5$, $B_4 \leftrightarrow B_7$.

Begründung: Jeder Übegang, bei dem die zugehörige Folge der Funktionswerte nicht monoton ist, ist mit einem Funktionshasard behaftet.

Aufgabe 5

1. Anzahl der Prüfbits:

   Aufwand: $2^k \geq m + k + 1$. Hier: m $= 48 \Rightarrow k = 6$

2. Bereiche:

   - Vorzeichen
   - Exponent
   - Mantisse

   Begründung:

   - -4 ist negativ $\Rightarrow$ Das Vorzeichen ändert sich.
   - Die Multiplikation mit einer 2er Potenz ändert den Exponenten.
   - Die Mantisse ändert sich am Randbereich. Wenn der Exponent überläuft, wird die Mantisse auf Null gesetzt.

3. Mindest-Bitanzahl für die Darstellung von 63 als Zweierkomplementzahl: 7 Bit
   Begründung: 63 als 6-Bit-Dualzahl: $63 = 11\ 1111_{2,6}$.
   $\Rightarrow$ Interpretiert als eine 6-Bit-Zweierkomplement-Zahl wäre dies eine negative Zahl.
   $\Rightarrow$ Es sind mindestens 7 Bit zur Darstellung von 63 als Zweierkomplementzahl notwendig.

   - 63 mit minimaler Bitanzahl: $+63 = 011\ 1111_{2K,7}$
   - 63 als 16-Bit Zweierkomplementzahl: $0000\ 0000\ 0011\ 1111_{2K,16}$

4. Pseudotetraden:
   Bei Pseudotetraden handelt es sich um diejenigen Zustände einer binär kodierten Dezimalzahl, welche keine gültige Dezimalziffer darstellen.

5. 1000 1001 1000 0000 0000 0000 0000 0011:

   (a) BCD: 89800003

   (b) Vorzeichenlose Dualzahl: $2^{31} + 2^{27} + 2^{24} + 2^{23} + 2^1 + 2^0$

   (c) Gleitkomma-Zahl im IEEE-754-Standard in einfacher Genauigkeit:

$$
\begin{aligned}
VZ &= 1 \\
Char &= 00010011 = 19 \\
Exp &= Char - 127 = -108 \\
M &= 00000000000000000000011 \Rightarrow \\
\\
Z &= (-1)^1 \cdot (1,00000000000000000000011) \cdot 2^{-108} \\
&= -(1 + 2^{-22} + 2^{-23}) \cdot 2^{-108}
\end{aligned}
$$

Aufgabe 6

1. Kodierung des Mikroprogramms für die Lese-Phase:

Takt	Adresse	Befehl in hexadezimaler Schreibweise
1. Takt	0x00	2 1 0 8 8 0 1 (X = P_w = S = 1; R = 1)
2. Takt	0x01	1 4 0 0 8 0 2 (Y = E = 1; R = 1)
3. Takt	0x02	0 0 0 1 8 0 3 (C_2-C_0 = 001; R = 1)
4. Takt	0x03	0 A 0 0 0 0 4 (Z = P_r = 1)
5. Takt	0x04	0 0 9 0 0 0 5 (I_r = D_w = 1)

2. Mikroprogramme:

LDV	STV
7. Takt: IR → SAR; R = 1 8. Takt: R = 1 9. Takt: R = 1 10. Takt: SDR → Akku	7. Takt: Akku → SDR 8. Takt: IR → SAR; W = 1 9. Takt: W = 1 10. Takt: W = 1
EQL	**JMP**
7. Takt: IR → SAR; R = 1 8. Takt: Akku → X; R = 1 9. Takt: R = 1 10. Takt: SDR → Y 11. Takt: ALU auf Vergleich 12. Takt: Z → Akku	7. Takt: IR → IAR

Aufgabe 7

1. Pseudobefehl / `li`:

   Bei einem Pseudobefehl handelt es sich um ein Mnemonik, das vom Assemblierer nicht auf einen Maschinenbefehl abgebildet wird. Es findet eine Ersetzung durch einen oder mehrere andere Befehle statt. Der Befehlsumfang kann mit dieser Technik für den Programmierer einfach erweitert werden, ohne dass die Komplexität der CPU erhöht wird.

   Pseudobefehle sind üblicherweise direkte Spezialfälle allgemeinerer Befehle (z.B. `b`, `clear` und `neg`) und Befehle wie `li`, die aus technischen Gründen nicht als ein Befehl codiert werden können. Das Problem des `li` Befehls liegt in der festen Breite eines Befehlswortes (32 Bit). Diese feste Länge verhindert das Speichern des Opcodes und der 32 Bit langen Konstanten in einem Befehlswort.

2. C-Kontrollstruktur in MIPS-Assembler:

```
       add     $a0, $zero, $zero   # i = 0
loop:  addi    $v0, $zero, 100     # Temporäre Variable für Vergleich
       bgt     $a0, $v0, end       # Wenn i > 100, dann gehe zu end
       add     $a1, $a1, $a0       # j += i
       addi    $a0, $a0, 10        # i += 10
       b       loop                # Gehe zu loop
end:
```

3. Fehlerfreie Version:

```
       add     $v0, $zero, $zero
loop:  lw      $t0, 0($a0)
       sne     $t1, $t0, $zero     # Fehler: seq statt sne
       add     $v0, $v0, $t1
       addi    $a0, $a0, 4         # Fehler: Adresse nur um Eins inkrementiert
       ble     $a0, $a1, loop      # Fehler: blt statt ble
```

4. Inhalte der Zielregister:

Befehl	Zielregister = (z. B. `$s6 = 0x0000 F00A`)
`subi $s1, $zero, 0x2`	`$s1 = 0xFFFF FFFE`
`sra  $s2, $s1, 4`	`$s2 = 0xFFFF FFFF`
`slti $s3, $s2, 100`	`$s3 = 0x0000 0001`
`lui  $s4, 0x40`	`$s4 = 0x0040 0000`
`xor  $s5, $s1, $s4`	`$s5 = 0xFFBF FFFE`

5. Unterschiede zwischen Registern `$t0-$t7` und die Register `$s0-$s7`:

   In den `$tX`-Registern werden vorwiegend temporäre Variablen abgelegt, während die `$sX`-Register zur längerfristigen Speicherung von Werten verwendet werden.

   Entsprechend besagt die Aufrufkonvention, dass der Wert der `$tX`-Register von Funktionen beliebig überschrieben werden darf.

   Vor dem Verändern eines `$sX`-Register muss hingegen der alte Wert gesichert und vor dem Rücksprung wieder hergestellt werden. Das Speichern erfolgt typischerweise auf dem Stack.

   Es handelt sich hierbei um eine Konvention der Programmierer, die keine Auswirkungen auf das Hardware-Design hat. Auf Hardwareebene sind die Register identisch behandelte General Purpose Register.

Aufgabe 8

1. Datenabhängigkeiten:

   - Echte Abhängigkeiten *(True Dependence)*

     $S_1 \rightarrow S_2$ $(R1)$ $S_2 \rightarrow S_3$ $(R2)$ $S_3 \rightarrow S_4$ $(R3)$
     $S_1 \rightarrow S_3$ $(R1)$ $S_2 \rightarrow S_4$ $(R2)$

   - Gegenabhängigkeiten *(Anti-Dependence)*:

     $S_1 \rightarrow S_2$ $(R2)$ $S_2 \rightarrow S_3$ $(R3)$ $S_3 \rightarrow S_4$ $(R1)$
     $S_1 \rightarrow S_3$ $(R3)$ $S_2 \rightarrow S_4$ $(R1)$

   - Ausgabe-Abhängigkeiten *(Output Dependence)*:

     $S_1 \rightarrow S_4$ $(R1)$

2. Belegung der Register nach Ablauf des Programms und Zustand der Pipeline:

Takt	IF	DE	OF	EX	WB	R1	R2	R3
1	S1	—	—	—	—	2	8	4
2	S2	S1	—	—	—	2	8	4
3	S3	S2	S1	—	—	2	8	4
4	S4	S3	S2	S1	—	2	8	4
5	—	S4	S3	S2	S1	4	8	4
6	—	—	S4	S3	S2	4	6	4
7	—	—	—	S4	S3	4	6	6
8	—	—	—	—	S4	32	6	6

   Anzahl der Takte: 8 Takte

3. Belegung der Register bei sequentieller Bearbeitung des Programms:

R1	R2	R3
32	8	4

4. Behebung der Pipelinekonflikte durch Einfügen von NOP-Befehlen:

```
S1:  SUB R1,R2,R3
     NOP
     NOP
S2:  ADD R2,R1,R3
     NOP
     NOP
S3:  SUB R3,R2,R1
     NOP
     NOP
S4:  MUL R1,R2,R3
```

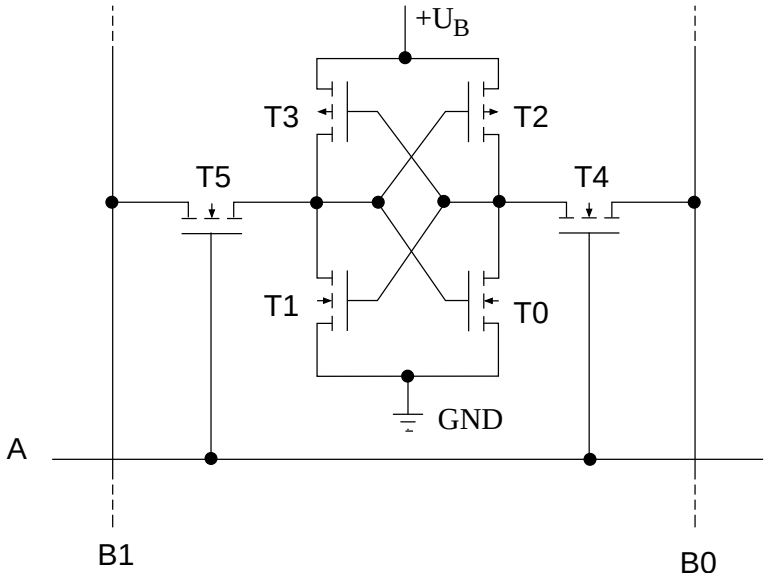
Anzahl der Takte: 14 Takte

Aufgabe 9

1. Das Mooresche Gesetz der Mikroelektronik:

   Für den gleichen Preis erreicht die Mikroelektronik die doppelte Leistung in jeweils weniger als zwei Jahren (oder die 100-fache Leistung in jeweils 10 Jahren).

2. Aufbau einer statischen MOS-Speicherzelle:



3. Zugriffszeit *(access time)*:

   Die maximale Zeitdauer, die vom Anlegen einer Adresse an den Speicher bis zur Ausgabe der gewünschten Daten vergeht.

   Zykluszeit *(cycle time)*:

   Die minimale Zeitdauer, die zwischen zwei aufeinanderfolgenden Aufschaltungen von Adressen auf den Speicher vergehen muss.

4. Vor- und Nachteile magnetischer Speichermedien:

   + Speicherung ohne Energiezufuhr
   + große Speicherkapazität/-dichte
   + Speicherung dauerhaft (Archivierung möglich)
   + ausgereifte Technologie
   + günstig im Preis pro Bit

   – große Zugriffszeiten (10 bis 20 ms bei Platten, bis zu Sekunden bei Bandspeichern)
   – Empfindlich (magnetische Störfelder, Temperatur, Verschmutzung usw.)
   – Hochpräzise Mechanik nötig

   Magnetische Speicher in einer Speicherhierarchie:

   In der unteren Ebene (Speichermedien mit großer Kapazität)

Aufgabe 10

1. *Dirty*-Bit:

   Kennzeichnet diejenigen Daten im Cache, die beim Verdrängen aus dem Cache in den Hauptspeicher zurückgeschrieben werden müssen, da sie gegenüber ihrem Original im Hauptspeicher verändert worden sind (Cache-Speicher mit *write back*-Schreibverfahren).

2. Unmittelbare und direkte Adressierung:

   Bei der unmittelbaren Adressierung enthält der Befehl nicht die Adresse des Operanden oder einen Zeiger darauf, sondern den Operanden selbst.

   Bei der direkten Adressierung enthält der Befehl die logische Adresse des Operanden.

3. Zweistufige und einstufigen Speicher-Adressierung:

   Bei der einstufigen Speicher-Adressierung ist nur eine Adressberechnung zur Ermittlung der effektiven Adresse nötig, d.h. es sind keine mehrfachen Speicherzugriffe zur Adressermittlung erforderlich.

   Bei der zweistufigen Speicher-Adressierung sind zur Adressermittlung mehrere sequenzielle Adressberechnungen und Speicherzugriffe erforderlich. Das Ergebnis der ersten Berechnung liefert eine Speicherzelle, deren Inhalt wieder eine Adresse oder ein Offset zur weiteren Berechnung ist.