



UNIVERSITÄT KARLSRUHE (TH)

Fakultät für Informatik

Institut für Prozessrechentechnik, Automation und Robotik (IPR)

Prof. Dr. U. Brinkschulte, Dr. T. Asfour

## Musterlösungen

zur Klausur „Technische Informatik I/II“

am 12. September 2005, 14.00 - 16.00 Uhr

Name:	Vorname:	Matrikelnummer:
<b>Bond</b>	<b>James</b>	<b>007</b>

<b>Technische Informatik I</b>	
Aufgabe 1	11 von 11 Punkten
Aufgabe 2	5 von 5 Punkten
Aufgabe 3	9 von 9 Punkten
Aufgabe 4	11 von 11 Punkten
Aufgabe 5	9 von 9 Punkten

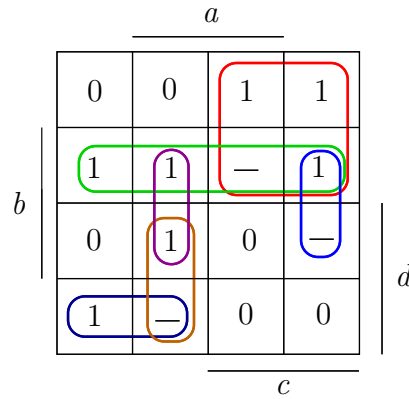
<b>Technische Informatik II</b>	
Aufgabe 6	4 von 4 Punkten
Aufgabe 7	12 von 12 Punkten
Aufgabe 8	12 von 12 Punkten
Aufgabe 9	10 von 10 Punkten
Aufgabe 10	7 von 7 Punkten

<b>Gesamtpunktzahl:</b>	90 von 90 Punkten
-------------------------	-------------------

<b>Note:</b>	<b>1,0</b>
--------------	------------

# Aufgabe 1

1. (a) Prim-Einsblöcke und Primimplikanten:



Primimplikanten:

$$\bar{d}c \quad \bar{d}b \quad d\bar{c}\bar{b} \quad d\bar{c}a \quad \bar{c}ba \quad cb\bar{a}$$

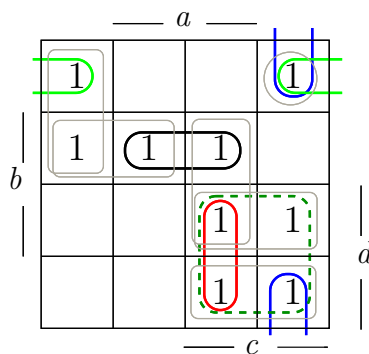
(b) DMFs:

$$y = \bar{d}c \vee \bar{d}b \vee d\bar{c}\bar{b} \vee \begin{cases} \bar{c}ba \\ d\bar{c}a \end{cases}$$

(c) Kürzeste Gleichung für das Nelson-Verfahren: KMF, aber ohne Berücksichtigung der *don't care*-Stellen

$$y = (d \vee c \vee b)(\bar{d} \vee \bar{c} \vee \bar{a})(\bar{d} \vee \bar{c} \vee b)(\bar{d} \vee c \vee \bar{b} \vee a)$$

2. Consensus-Blöcke:



3. (a)

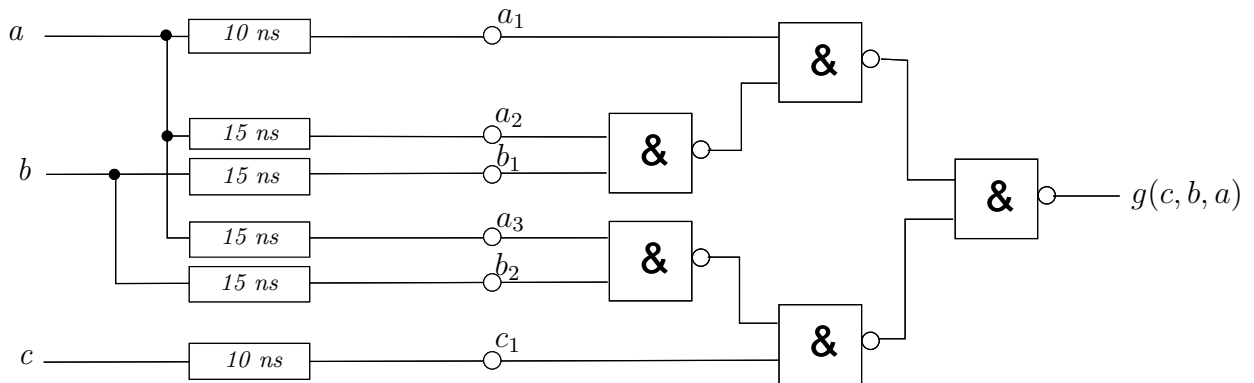
$$\begin{aligned} y \vee \bar{x}z \vee x\bar{y} &= (y \vee x)(y \vee \bar{y}) \vee \bar{x}z \\ &= y \vee x \vee \bar{x}z \\ &= y \vee (x \vee \bar{x})(x \vee z) \\ &= y \vee x \vee z \end{aligned}$$

(b)

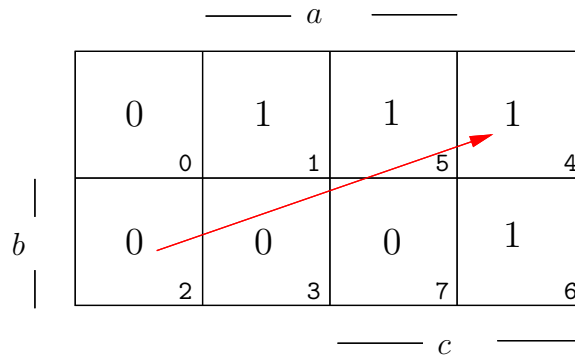
$$\begin{aligned}
 a &= a \vee 0 = a \vee (a \vee \bar{a}) = (a \vee a) \cdot (a \vee \bar{a}) \\
 &= (a \vee a) \cdot 1 = a \vee a
 \end{aligned}$$

## Aufgabe 2

1. Totzeitmodell:



2. Übergang  $(c, b, a) : (0, 1, 0) \rightarrow (1, 0, 0)$ :

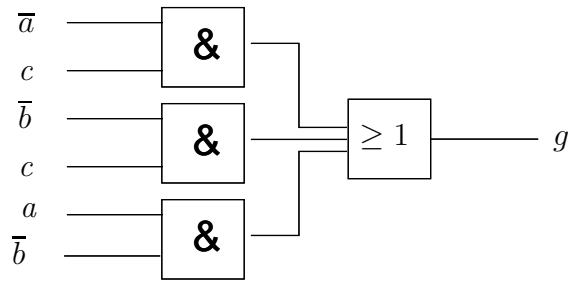


Zu beiden Wegen ( $B_2 \rightarrow B_0 \rightarrow B_4$  und  $B_2 \rightarrow B_6 \rightarrow B_4$ ) von der Anfangs- zur Endbelegung gehören monotonem Folgen der Funktionswerte, deshalb ist der Übergang frei von Funktionshasards.

3. Von allen statischen Strukturhasards freies Schaltnetz: Zweistufiges Schaltnetz aus der Disjunktion aller Primimplikanten oder aus der Konjunktion aller Primimplikate (Satz von Eichelberger)

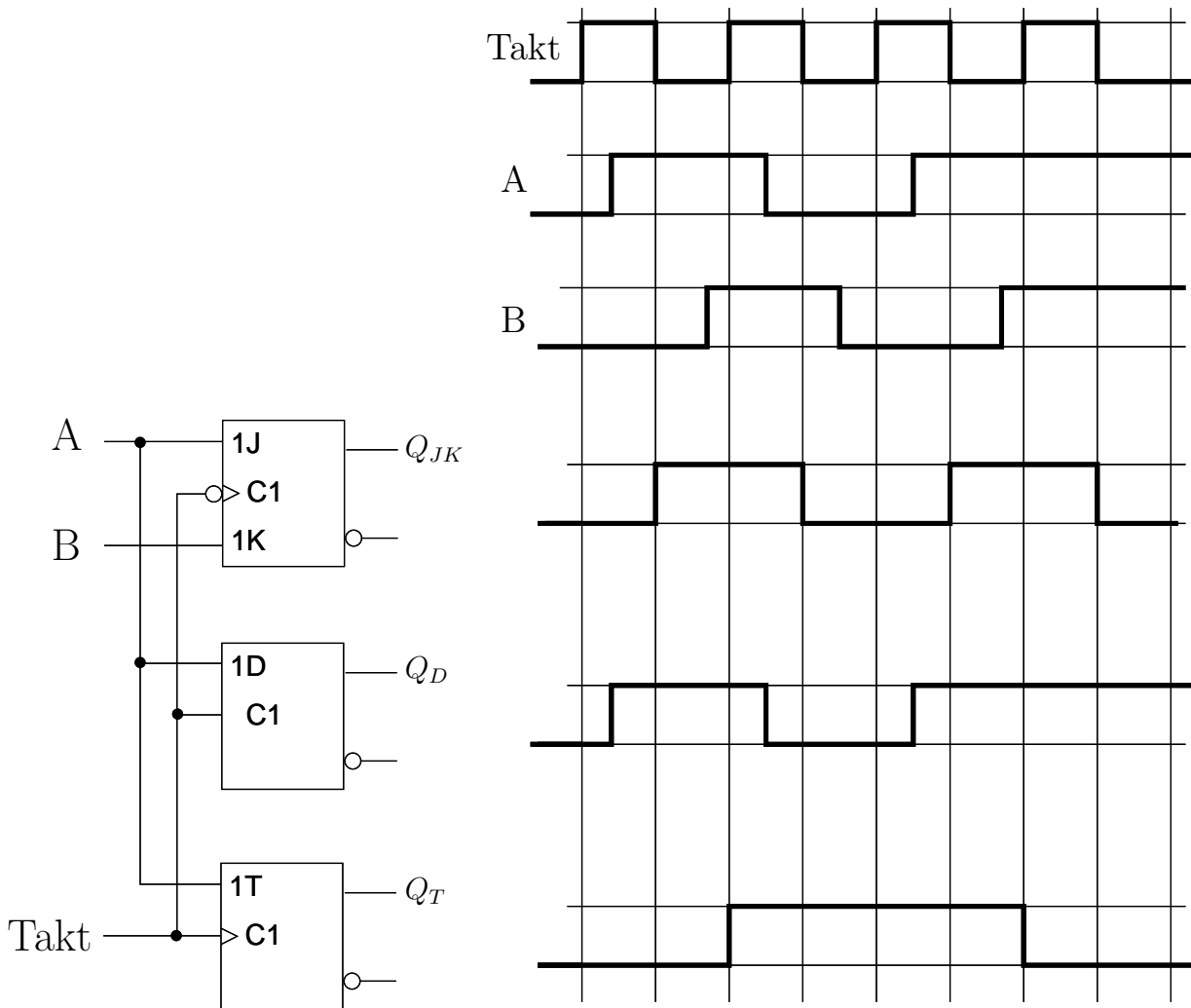
Schaltnetz:

$$g(c, b, a) = c\bar{a} \vee c\bar{b} \vee \bar{b}a$$



### Aufgabe 3

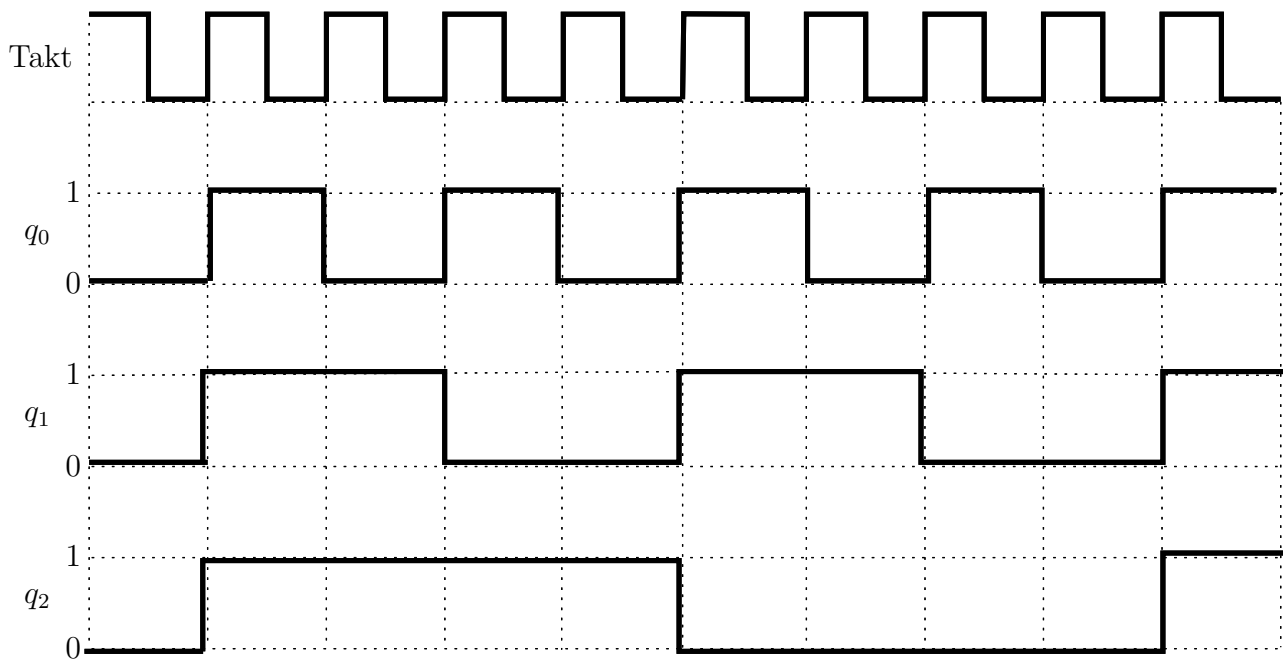
1. Verläufe der Signale  $Q_{JK}$ ,  $Q_D$  und  $Q_T$ :



2. (a) Anzahl der Zustände:  $2^3 = 8$

(b) Automatentyp: Es ist nicht möglich, den Automatentypen anzugeben, da nicht bekannt ist, wie die Ausgabe gebildet wird.

(c) Verläufe der Signale  $q_2$ ,  $q_1$  und  $q_0$ :



(d) Funktion des Schaltwerks: Modulo 8-Rückwärtszähler



## Aufgabe 5

### 1. Register-Transfer-Operationen

Takt	$S$	$F$	$R$	$D$	$RA$	$RQ$	$RM$
1	1	0	0	0	1	3	1
2	0	1	0	0	1	3	0
3	0	0	1	0	1	3	0
4	0	1	0	0	0	3	3
5	0	0	0	1	0	3	3

### 2. ALU-Design:

$p_3$	$p_2$	$p_1$	$p_0$	Operation	Bedeutung
0	1	1	0	$F_i \leftarrow A_i \leftrightarrow B_i$	Bitweise Antivalenz
1	0	0	0	$F_i \leftarrow A_i \wedge B_i$	Bitweise Antivalenz
1	1	1	0	$F_i \leftarrow A_i \vee B_i$	Bitweise Antivalenz
0	0	1	1	$F_i \leftarrow \overline{A_i}$	Einerkomplement
1	1	0	0	$F_i \leftarrow A_i$	Select $A$

## Aufgabe 6

1.

<i>Speicher-Bausteine</i>	<i>richtig</i>	<i>falsch</i>
Dynamische RAM-Bausteine speichern die Information in Kondensatoren.	×	
Statische RAM-Bausteine lassen sich besser als dynamische integrieren und besitzen eine kürzere Zugriffszeit.		×
Die Zugriffszeit eines DRAM-Speicher-Bausteins ist kleiner als die Zykluszeit.	×	
Bei DDRAM-Bausteinen wird der Datendurchsatz durch die Verdoppelung ihrer Taktfrequenz verdoppelt.		×

2.

<i>Assembler</i>	<i>richtig</i>	<i>falsch</i>
Assemblerdirektiven erzeugen immer Maschinencode.		×
Pseudobefehle sind prinzipiell nicht notwendig, da sie durch „echte“ Maschinenbefehle ersetzt werden können.	×	
In eingebetteten Systemen mit beschränkten Speicherressourcen wird bevorzugt in Assembler programmiert.	×	
Assemblersprachen sind maschinenspezifisch.	×	

3.

<i>DMA/Interrupts</i>	<i>richtig</i>	<i>falsch</i>
DMA-Controller führen die Datenübertragung hardwaremäßig, d. h. ohne ein im Hauptspeicher liegendes Programm durch.	×	
Das <i>fly-by</i> -Transferverfahren kann auch für Speicher-Speicher-Transfer benutzt werden.		×
Interrupt-Controller ermitteln, bei mehreren vorliegenden maskierbaren und nicht-maskierbaren Interrupts, die Interruptquelle mit der höchsten Priorität.		×
Unterbrechungen sind recht selten in einem Rechner und deuten auf Programmierfehler hin.		×

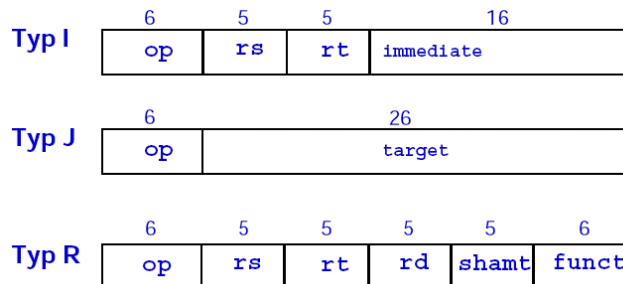
4.

<i>Virtuelle Speicherverwaltung</i>	<i>richtig</i>	<i>falsch</i>
Das Betriebssystem ist allein zuständig für die Umsetzung virtueller in physikalische Adressen.		×
Die Umsetzung virtueller in physikalische Adressen wird durch spezielle Hardware beschleunigt.	×	
Die Seitengrößen können je nach Anforderung dynamisch wachsen.		×
Externe Fragmentierung ist ein typisches Problem von Segmentierung.	×	

## Aufgabe 7

1. MIPS-Befehlsformate:

- R-Typ (*register*)
- I-Typ (*immediate*)
- J-Typ (*jump*)



2. Kommentieren der Zeilen:

Zeile	Kommentar
li \$t2, 0x40	Lade 0x40 ins Register \$t2
ori \$t3, \$zero, 125	Bitweise ODER ( $0 \vee 125_{10}$ )
div \$t3, \$t2	\$t3 div \$t2
mfhi \$t3	\$t3 = \$t3 mod \$t2
andi \$t3, \$t3, 0x0E	Bitweise UND ( $\$t3 \wedge 0E_{16}$ )

Register-Inhalte:

Zeile	Register mit einem neuen Wert
li \$t2, 0x40	\$t2 = x40
ori \$t3, \$zero, 125	\$t3 = 125 (0x7D)
div \$t3, \$t2	
mfhi \$t3	\$t3 = 61 (0x3D)
andi \$t3, \$t3, 0x0E	\$t3 = 0x0C

3. while-Schleife in MIPS:

```
Loop:   add $t1, $s3, $s3
        add $t1, $t1, $t1
        add $t1, $t1, $s6
        lw $t0, 0($t1)
        bne $t0, $s5, Exit
        add $s3, $s3, $s4
        j Loop
Exit:
```

4. Werte von x und y:

x = 17	y = 17
--------	--------

5. ptr = &i; in MIPS-Assembler:

```
la $t0, i    # Adresse bestimmen
sw $t0, ptr
```

6. i = \*ptr; in MIPS-Assembler:

```
lw $t0, ptr
lw $t0, 0($t0)
sw $t0, i
```

## Aufgabe 8

1. Datenabhängigkeiten:

- True Dependence ( $\delta^t$ ):

$$S_1 \longrightarrow S_2 \qquad S_3 \longrightarrow S_4 \qquad S_1 \longrightarrow S_5$$

- Anti Dependence ( $\delta^a$ ):

$$S_2 \longrightarrow S_3 \qquad S_2 \longrightarrow S_4 \qquad S_1 \longrightarrow S_5$$

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

$$S_3 \longrightarrow S_4$$

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

Takt	IF	ID/RF	EX	MEM	WB	\$t1	\$t2	\$t3	\$t4	\$t5
1	S1	—	—	—	—	3	5	7	9	2
2	S2	S1	—	—	—	3	5	7	9	2
3	S3	S2	S1	—	—	3	5	7	9	2
4	S4	S3	S2	S1	—	3	5	7	9	2
5	S5	S4	S3	S2	S1	7	5	7	9	2
6	—	S5	S4	S3	S2	7	5	7	4	2
7	—	—	S5	S4	S3	7	5	10	4	2
8	—	—	—	S5	S4	7	5	10	4	2
9	—	—	—	—	S5	7	9	10	4	2

Anzahl der Takte: 9

3. Belegung der Register bei sequentieller Bearbeitung des Programms:

\$t1	\$t2	\$t3	\$t4	\$t5
7	9	13	0	2

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

```

S1:  addi  $t1, $t2, 2
      noop
      noop
S2:  sub   $t4, $t3, $t1
S3:  muli  $t3, $t5, 5
    
```

```
      noop
      noop
S4:  addi  $t3, $t3, 3
S5:  addi  $t2, $t1, $t5
```

Anzahl der Takte: 13

5. Bedingte Sprünge (Problem und zwei Lösungsmöglichkeiten):

Erst am Ende der 4. Stufe wird entschieden, ob gesprungen wird oder nicht. Die folgenden drei Befehle sind schon in der Pipeline und müssen bei einem Sprung gelöscht werden.

Lösungsmöglichkeiten:

- Verzögerten Sprungtechnik (*delayed branch technique*): Drei Verzögerungsschlitze (*delay slots*) mit Leerbefehlen (`noop`) nach jedem Sprungbefehl.
- Befehlsumordnung: Befehle, die in der logischen Programmreihenfolge vor dem Sprungbefehl liegen, in die Verzögerungsschlitze 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.
- *Sprungvorgehrsage*: Spekulation auf nicht ausgeführte bedingte Sprünge. Man nimmt an, dass ein Sprung nicht ausgeführt wird, und lädt die nachfolgenden Befehle in die Pipeline.

6. Befehlssatz mit Befehlen variabler Länge in der DLX-Pipeline:

Nein. Es ist nicht möglich, Befehle variabler Länge in der DLX-Pipeline auszuführen, da in diesem Fall die Adresse des nächsten Befehls frühestens nach der Befehlsdekodierung (Ende der 2. Stufe) bestimmt werden kann.



## Aufgabe 10

1. Nachteil von Befehlssätzen mit *OpCodes* variabler Länge:

Dekodierung schwieriger und Dekodierschaltungen komplizierter.

2. *two cycle transfer*-Verfahren bei DMA:

Der DMA-Controller führt zunächst einen Lesezugriff auf den Speicher durch und lädt das Datum in ein internes Pufferregister. Danach spricht er den Peripherie-Baustein an und überträgt das Datum (Schreibzugriff). Es sind also zwei Speicherzugriffe für den Datentransfer nötig.

3. *Speicherbezogene* und *isolierte* Adressierung von Peripherie-Bausteinen:

- Speicherbezogene Adressierung: Kein Unterschied zwischen Speicheradresse und Adresse eines Registers eines Peripherie-Bausteins. Häufig wird ein zusammenhängender Speicherbereich für Peripherie-Bausteine verwendet (I/O-Page).
- Isolierte Adressierung: Getrennte Adressräume für Speicher und Peripherie-Bausteine. Auswahl durch  $M/\overline{IO}$ -Signal

4. Aufbau eines Systemschnittstellen-Bausteins:

