Karlsruher Institut für Technologie

Fakultät für Informatik
Prof. Dr. Tamim Asfour, Prof. Dr. Wolfgang Karl

Musterlösungen zur Klausur

Digitaltechnik und Entwurfsverfahren/Rechnerorganisation

Technische Informatik I/II

am 28. Februar 2014, 14:00 – 16:00 Uhr

Name:	Vorname:	Matrikelnummer:
Bond	James	007

Digitaltechnik und Entwurfsverfahren/TI-1	
Aufgabe 1	10 von 10 Punkten
Aufgabe 2	8 von 8 Punkten
Aufgabe 3	7 von 7 Punkten
Aufgabe 4	10 von 10 Punkten
Aufgabe 5	10 von 10 Punkten

Rechnerorganisation/TI-2	
Aufgabe 6	6 von 6 Punkten
Aufgabe 7	12 von 12 Punkten
Aufgabe 8	12 von 12 Punkten
Aufgabe 9	11 von 11 Punkten
Aufgabe 10	4 von 4 Punkten

Gesamtpunktzahl:	90 von 90 Punkten

	Note: **1,0**

Aufgabe 1

1. KNF von $f(w,x,y,z)$: (KV-Diagramm oder algebraisch)

$$\begin{aligned} f(w,x,y,z) &= \text{MAXt}(0,4,6,7,14,15) \\ &= (w \vee x \vee y \vee z) \cdot (w \vee x \vee \overline{y} \vee z) \cdot (w \vee \overline{x} \vee \overline{y} \vee z) \cdot \\ &\quad (w \vee \overline{x} \vee \overline{y} \vee \overline{z}) \cdot (\overline{w} \vee \overline{x} \vee \overline{y} \vee z) \cdot (\overline{w} \vee \overline{x} \vee \overline{y} \vee \overline{z}) \end{aligned}$$

2. DMF von $g(c,b,a)$:

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

3. Kern-Primimplikate: C

   Reduzierte Tabelle: (Gestrichene Spalten: a, d)

	b	c	e
A	×	×	
B			×
D	×		×
E	×	×	×

4. Dominierte Maxterme: c

   Reduzierte Tabelle: (Gestrichene Spalte: b)

	c	e	Kosten
A	×		$\frac{1}{2}$
B		×	$\frac{1}{2}$
D		×	$\frac{1}{3}$
E	×	×	$\frac{1}{3}$

5. Dominierende Primimplikate: E

   Reduzierte Tabelle: (Gestrichene Zeilen: A, B und D)

	c	e
E	×	×

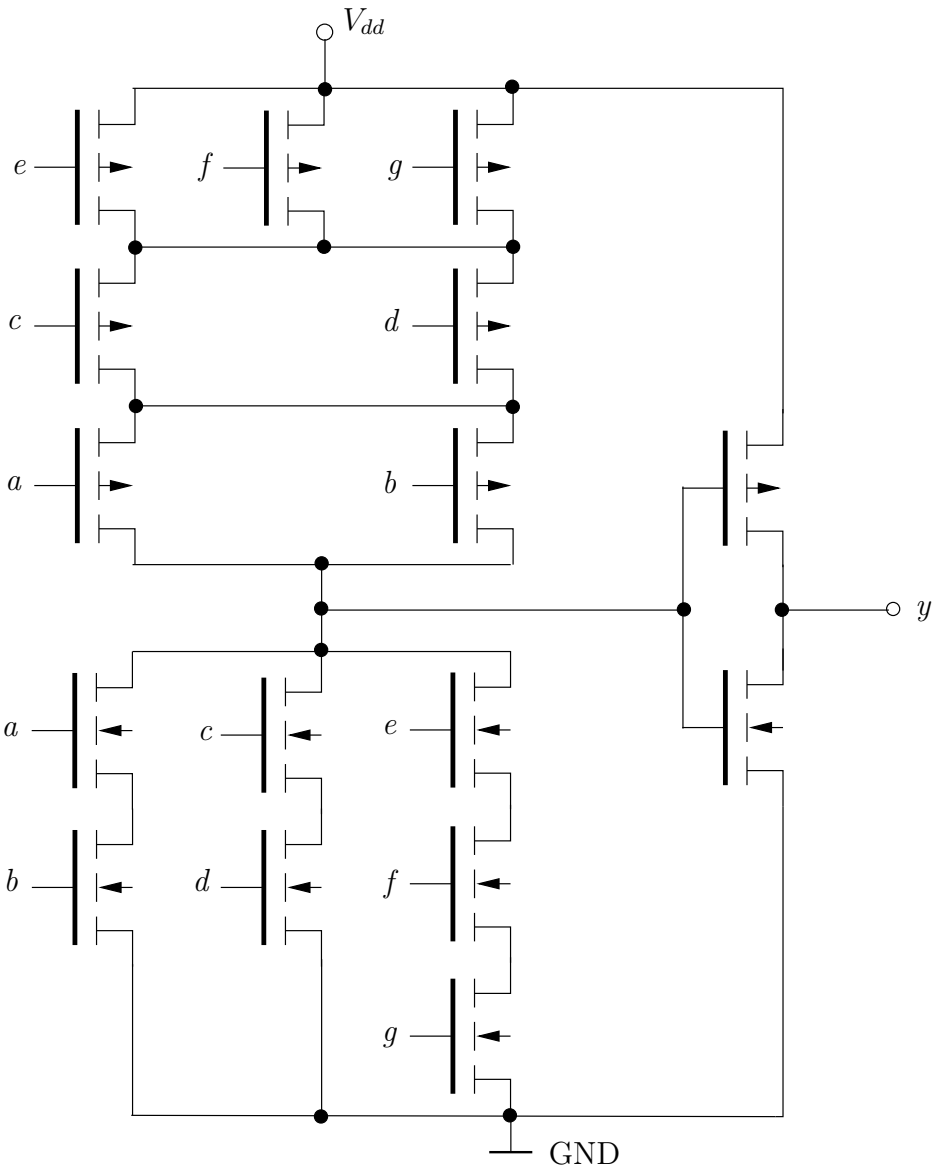
6. Minimalform der Funktion z: $z = C \cdot E$

Aufgabe 2

Schaltwerk	**1**	**2**	**3**	**4**
zählt vorwärts			×	×
zählt rückwärts		×		
ist synchron	×	×	×	
kann bei *jedem* Zählerstand mit Hilfe von x angehalten werden.		×		×

Aufgabe 3

1. CMOS-Schaltnetz:



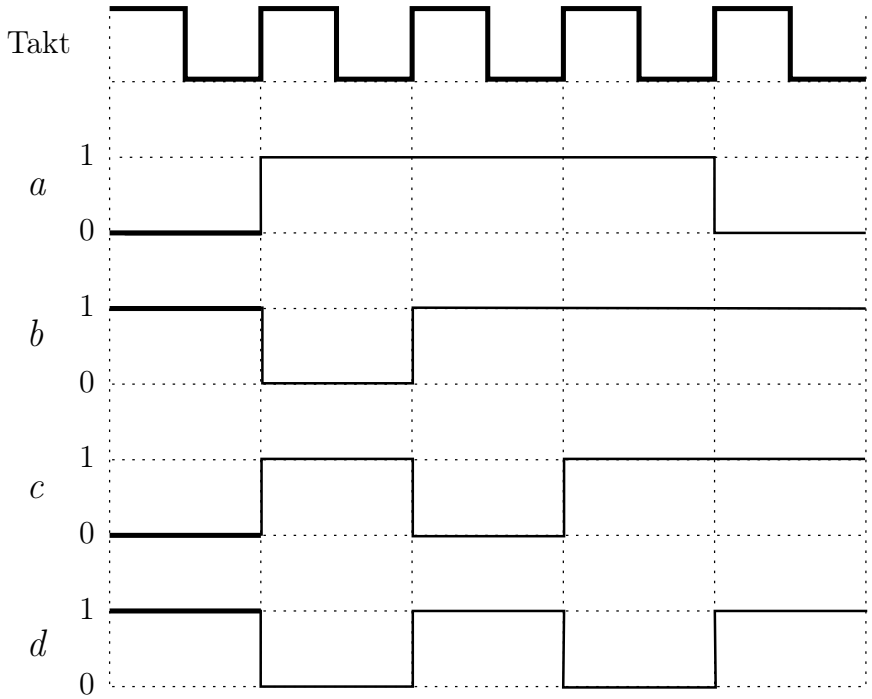
2. Schaltfunktion: $z = \overline{ad \vee bd \vee aef \vee bef \vee ced \vee cf}$

Aufgabe 4

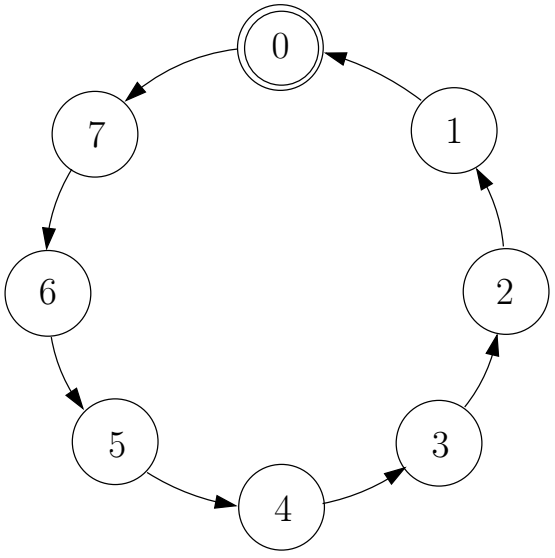
1. (a) Das Schaltwerk ist *synchron*

   (b) Maximale Anzahl der Zustände ist: $2^4 = 16$ Zustände

   (c) Verläufe der Signale a, b, c und d:



2. (a) Automatengraph:



(b) Kodierte Ablauftabelle:

q_2^t	q_1^t	q_0^t	q_2^{t+1}	q_1^{t+1}	q_0^{t+1}	e_2^t	e_1^t	e_0^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	1	1	1
1	0	1	1	0	0	0	0	1
1	1	0	1	0	1	0	1	1
1	1	1	1	1	0	0	0	1

(c) Minimalformen der Ansteuerfunktionen der Flipflops: Aus der Ablauftabelle ablesbar.

$$
\begin{aligned}
e_2^t &= \overline{q}_2^t\ \overline{q}_1^t\ \overline{q}_0^t\ \vee\ q_2^t\ \overline{q}_1^t\ \overline{q}_0^t\ =\ \overline{q}_1^t\ \overline{q}_0^t \\
e_1^t &= \overline{q}_0^t \\
e_0^t &= 1
\end{aligned}
$$

Aufgabe 5

1. Gleitkommadarstellung von

   (a) $29,375_{10} = 11101,011_2 = 1,1101011_2 \cdot 2^4$

   $$Exp = 4 \rightarrow Char = 127 + 4 = 131_{10} = 1000\ 0011_2$$
   $$Mantisse = 1101011 \qquad VZ = 0$$

31	30 23	22 0
0	1000 0011	1101 0110 0000 ... 000

   (b) $-0,75_{10} = -0,11_2 = -1,1 \cdot 2^{-1}$

   $$Exp = -1 \rightarrow Char = 127 - 1 = 126_{10} = 0111\ 1110_2$$
   $$Mantisse = 1 \qquad VZ = 1$$

31	30 23	22 0
1	0111 1110	1000 0000 ... 000

2. (a) `0011 1111 1000 0000 0000 0000 0000 0000` als
   `0011 1111 1100 1000 0000 0000 0000 0000` als

   - Zweierkomplement:

   $$\begin{aligned} X &= 00111111100000000000000000000000_2 \quad \rightarrow \\ &= (2^{29} + 2^{28} + 2^{27} + 2^{26} + 2^{25} + 2^{24} + 2^{23}) \end{aligned}$$

   $$\begin{aligned} X &= 00111111110010000000000000000000_2 \quad \rightarrow \\ &= (2^{29} + 2^{28} + 2^{27} + 2^{26} + 2^{25} + 2^{24} + 2^{23} + 2^{22} + 2^{19}) \end{aligned}$$

   - Gleitkommazahl (IEEE-754):

   $$\begin{aligned} VZ &= 0 \\ Char &= 01111111_2 = 127_{10} \rightarrow Exp = Char - 127 = 127 - 127 = 0 \\ Mantisse &= 00000000000000000000000 \quad \rightarrow \\ Zahl &= (-1)^{VZ} \cdot (1, Mantisse) \cdot 2^{Exp} \\ &= (-1)^0 \cdot (1, 00000000000000000000000) \cdot 2^0 \\ &= 1 \end{aligned}$$

$$
\begin{aligned}
VZ &= 0 \\
Char &= 01111111_2 = 127_{10} \rightarrow Exp = Char - 127 = 127 - 127 = 0 \\
Mantisse &= 10010000000000000000000 \quad \rightarrow \\
Zahl &= (-1)^{VZ} \cdot (1, Mantisse) \cdot 2^{Exp} \\
&= (-1)^0 \cdot (1, 10010000000000000000000) \cdot 2^0 \\
&= 1,1001
\end{aligned}
$$

Alternative gültige Lösungsvariante für 32-Bit-Zahl aus der Aufgabenstellung

(b) `1001 0000 0110 1000 0000 0000 0000 0000` als

- Zweierkomplement:

$$
\begin{aligned}
X &= 10010000011010000000000000000000 \\
-X &= 01101111100110000000000000000000 \\
&= (2^{30} + 2^{29} + 2^{27} + 2^{26} + 2^{25} + 2^{24} + 2^{23} + 2^{20} + 2^{19}) \quad \rightarrow \\
X &= -(2^{30} + 2^{29} + 2^{27} + 2^{26} + 2^{25} + 2^{24} + 2^{23} + 2^{20} + 2^{19})
\end{aligned}
$$

- Gleitkommazahl (IEEE-754):

$$
\begin{aligned}
VZ &= 1 \\
Char &= 00100000_2 = 31_{10} \rightarrow Exp = Char - 127 = 32 - 127 = -95 \\
Mantisse &= 11010000000000000000000 \quad \rightarrow \\
Zahl &= (-1)^{VZ} \cdot (1, Mantisse) \cdot 2^{Exp} \\
&= (-1)^1 \cdot (1, 11010000000000000000000) \cdot 2^{-95} \\
&= -(1 + 2^{-1} + 2^{-2} + 2^{-4}) \cdot 2^{-95}
\end{aligned}
$$

3. Ausnahmeregel für die Null im IEEE-754-Standard: Liegt an der Darstellung einer normalisierten Zahl, bei der die führende 1 nur implizit dargestellt wird. Auch wenn die Mantisse 0 ist, steht eine 1 vor dem Dezimalpunkt.

4. Datenwort:

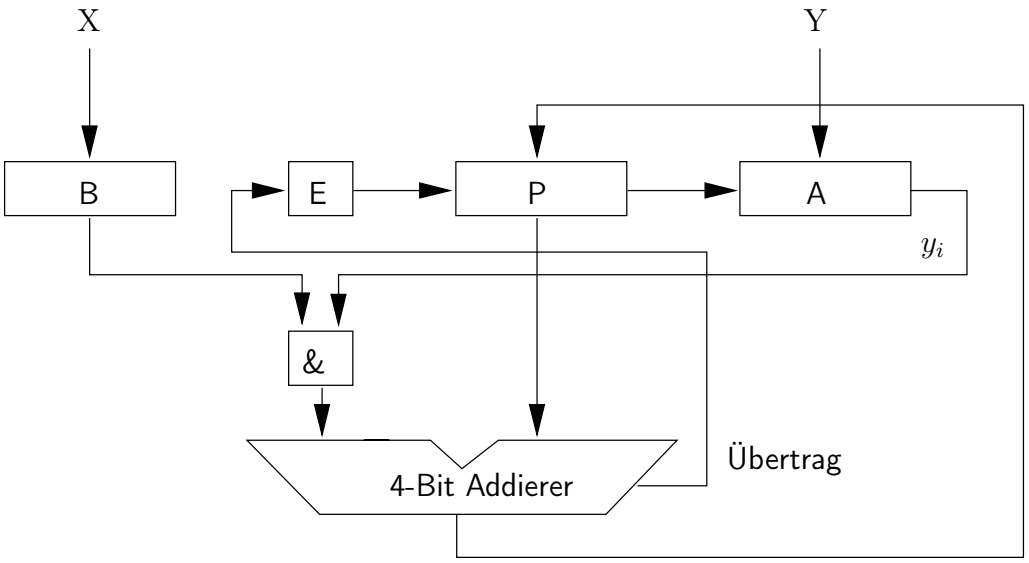
Position	12	11	10	9	8	7	6	5	4	3	2	1
	m_8	m_7	m_6	m_5	k_4	m_4	m_3	m_2	k_3	m_1	k_2	k_1
Codewort	1	1	0	1	0	1	1	1	0	1	1	1

Die Prüfbits lassen sich nach den folgenden Regeln berechnen:

$$
\begin{aligned}
k_1 &= k_1 \oplus m_1 \oplus m_2 \oplus m_4 \oplus m_5 \oplus m_7 \\
k_2 &= k_2 \oplus m_1 \oplus m_3 \oplus m_4 \oplus m_6 \oplus m_7 \\
k_3 &= k_3 \oplus m_2 \oplus m_3 \oplus m_4 \oplus m_8 \\
k_4 &= k_4 \oplus m_5 \oplus m_6 \oplus m_7 \oplus m_8
\end{aligned}
$$

Codewort: **1 1 0 1 0 1 1 1 0 1 1 1** $\Rightarrow$ $k_4\ k_3\ k_2\ k_1 = 1\ 0\ 1\ 0$ $\Rightarrow$
Bitfehler bei Bit 10 (m_6) $\Rightarrow$ Datenwort: **1 1 1 1 1 1 1 1**

5. Serieller Multiplizierer nach der PPS-Methode:



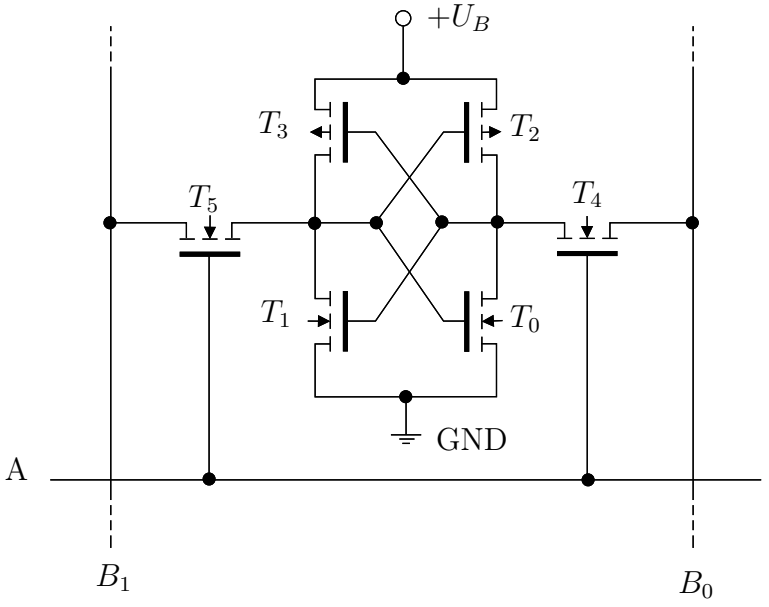
Aufgabe 6

1. *von-Neumann-Flaschenhals:* Daten und Maschinenbefehle werden über die gleiche Verbindungseinrichtung zwischen dem Prozessor und dem Hauptspeicher transportiert. Befehle und Operanden können nur nacheinander aus dem Speicher geladen werden.

2. Bits im Statusregister:

   - Übertragsbit *(Carry Flag CF)*: Übertrag bei Addition oder Subtraktion (Borrow).
   - Hilfsübertragsbit *(Auxiliary Carry AF)*: Übertrag von Bit 3 in Bit 4 des Ergebnisses bei BCD-Arithmetik.
   - Nullbit *(Zero Flag ZF)*: Zeigt an, ob das Ergebnis der letzten Operation gleich 0 war (bedingte Programmverzweigungen, Zählschleifen).
   - Vorzeichenbit *(Sign Flag SF)*: Zeigt an, dass das Ergebnis negativ ist.
   - Überlaufbit *(Overflow Flag OF)*: Bereichsüberschreitung im Zweierkomplement.
   - Even Flag *(EF)*: zeigt an, ob das Ergebnis eine gerade Zahl ist.
   - Paritätsbit *(Parity Flag PF)*: Signalisiert ungerade Parität des Ergebnisses

3. Statische CMOS-Speicherzelle:



Aufgabe 7

1. Inhalte der Zielregister:

Befehl	Zielregister = (z. B. `$s6 = 0x0000 F00A`)
`andi $s1, $zero, 40`	`$s1 = 0x0000 0000`
`sra $s2, $s1, 2`	`$s2 = 0x0000 0000`
`slt $s3, $zero, $s2`	`$s3 = 0x0000 0000`
`sub $s4, $s3, $s2`	`$s4 = 0x0000 0000`
`xori $s5, $s4, -1`	`$s5 = 0xFFFF FFFF`

2. (a) Registerinhalte:

Register	Inhalt
`$t1`	`$t1 = 0x0000 0014`
`$t2`	`$t2 = 0x0000 0010`
`$t3`	`$t3 = 0x0000 0000`
`$t4`	`$t4 = 0xFFFF FFFC`

(b) MIPS-Code zur Speicherung der Adresse von `vec` im Register `$s0`:

```
la $s0, vec
```

(c) Programmschleife zur Ausgabe von Partialsummen aus `vec`:

```
        .data
vec:    .word 20, 16, 12, 0, -3, 0, -4, 16, 20, 2014
leer:   .asciiz " "

        .text

        la   $s0, vec          # Adresse von vec in $a0
        addi $t0, $0, 0        # Zähler = 0
        addi $t1, $0, 10       # Zum Beenden der Schleife
        add  $t2, $0, $s0      # Zeiger auf aktuelles Element
        addi $t3, $0, 0        # Summenvariable
loop:   add  $a0, $0, $t3      # Auszugebende Zahl nach $a0 kopieren
        li   $v0, 1            # print_int
        syscall
        la   $a0, leer
        li   $v0, 4            # print-str (Leerzeichen)
        syscall
        lw   $t4, 0($t2)       # Element aus vec in $t4 laden
        add  $t3, $t3, $t4     # Auf Summenvariable addieren
        addi $t0, $t0, 1       # Zähler inkrementieren
        addi $t2, $t2, 4       # Zeiger auf vec inkrementieren
        bne  $t0, $t1, loop
```

3. Pseudobefehl:

   Pseudobefehle stellen eine Erweiterung der Assemblerbefehle dar. Sie werden vom Assembler in eine Folge von Maschinenbefehlen umgesetzt. Für den Assemblerprogrammierer unterscheiden sie sich nicht von den echten Maschinenbefehlen.

   Assemberdirektive:

   Assemblerdirektiven sind Anweisungen an den Assembler, die z. B. der Steuerung des Assembliervorgangs, der Platzierung vom Programmcode und Daten, der statischen Allokierung von Speicherplatz, der Wertzuweisung an Operanden und der Erzeugung von Konatanten dienen.

4. Trennung von Programmen und Daten bei der MIPS-R2000-Architektur:

   Der Hauptspeicher wird in mehreren Segementen unterteilt. Es existiert ein Datensegment für die Daten und ein Textsegment für Programme.

Aufgabe 8

1. Datenabhängigkeiten:

   - True Dependence (δ^t):

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

   - Anti Dependence (δ^a):

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

   - Output Dependence (δ^o):

   $$S_1 \longrightarrow S_4$$

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	—	—	—	—	4	8	0	-2	15
2	S2	S1	—	—	—	4	8	0	-2	15
3	S3	S2	S1	—	—	4	8	0	-2	15
4	S4	S3	S2	S1	—	4	8	0	-2	15
5	S5	S4	S3	S2	S1	8	8	0	-2	15
6	—	S5	S4	S3	S2	8	8	2	-2	15
7	—	—	S5	S4	S3	8	8	2	0	15
8	—	—	—	S5	S4	-8	8	2	0	15
9	—	—	—	—	S5	-8	0	2	0	15

   Anzahl der Takte: 9

3. Belegung der Register bei sequentieller Bearbeitung des Programms:

$t1	$t2	$t3	$t4	$t5
-8	18	6	12	15

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

```
S1:  and   $t1, $t2, $t5
     nop
     nop
S2:  add   $t3, $t1, $t4
     nop
     nop
S3:  sll   $t4, $t3, 1
S4:  sub   $t1, $0,  $t1
     nop
S5:  add   $t2, $t3, $t4
```

   Der `NOP`-Befehl zwischen S4 und S5 kann alternativ auch zwischen S3 und S4 eingefügt werden.

   Anzahl der Takte: 14

5. Bedingte Sprünge (Problem und zwei Lösungsmöglichkeiten):

   Erst am Ende der 4. Stufe wird entscheiden, 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 Verzögerungszeitschlitze *(delay slots)* mit Leerbefehlen (`noop`) nach jedem Sprungbefehl.
   - Befehlsumordnung: Befehle, die in der logischen Programmreihenfolge vor dem Sprungbefehl liegen, in die Verzögerungszeitschlitze 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 9

1. Unterteilung der Hauptspeicheradresse:

2. Speicherbedarf:
   Es sind für jede Zeile (Tag + 2 Statusbits + Daten pro Zeile) Bits erforderlich
   Für den gesamten Cache:

$$(26 + 2 + 128) \cdot 8 \;=\; 156 \cdot 8 \text{ Bit} \;=\; 156 \text{ Byte}$$

3. Cache-Miss (**M**) oder Cache-Hit(**H**):

Adresse	Hit/Miss
0x743	M
0x2A1	M
0x183	M
0x2A9	H
0x06F	M
0x69C	M
0x2A3	H
0x26A	M
0x18C	H
0x3C7	M
0x2A8	H
0x475	M

4. *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)

Aufgabe 10

1.

Speicher-Bausteine	*richtig*	*falsch*
Der Hauptspeicher heutiger Rechner wird aus statischen RAM-Bausteinen realisiert.		×
Statische RAM-Bausteine lassen sich schlechter als dynamische RAM-Bausteine integrieren; sie besitzen jedoch eine kürzere Zugriffszeit.	×	
Statische RAM-Bausteine müssen ständig „refresht“ werden.		×
Statische RAM-Bausteine speichern die Information in Kondensatoren.		×

2.

Programmiersprache C	*richtig*	*falsch*
Globale Variablen werden als Variablen der Speicherklasse *static* definiert.	×	
Ein *Pointer* ist ein Zeiger auf einen Speicherbereich.	×	
In C sind rekursive Funktionsaufrufe nicht zulässig.		×
Eine Variable, auf die schnell zugegriffen werden muss, wird als eine Variable der Speicherklasse *register* definiert.	×	

3.

Adressierung	richtig	falsch
Bei einem von-Neumann-Rechner kann anhand der Adresse eines Datums erkannt werden, ob es sich bei diesem Datum um einen Operanden oder um einen Befehl handelt.		×
Beim Little-Endian-Datenformat ist das niedrigstwertige Byte eines Wortes an seiner niedrigsten Adresse.	×	
Bei einer speicherbezogenen Adressierung *(memory mapping)* von Peripherie-Bausteinen existieren zwei getrennte Adreßräume für Speicher und Peripherie.		×
Durch das Multiplexen bestimmter Signalgruppen kann die Anzahl der Anschlüsse am Prozessorgehäuse verringert werden.	×	

4.

Virtuelle Speicherverwaltung	richtig	falsch
Externe Fragmentierung ist ein typisches Problem der Segmentierung.	×	
Das Betriebsystem ist allein zuständig für die Umsetzung virtueller in physikalische Adressen.		×
Seitengrößen können je nach Anforderung dynamisch wachsen.		×
Die Umsetzung virtueller in physikalische Adressen wird durch die MMU beschleunigt.	×	