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

Aufgabenblätter zur Prüfung

Digitaltechnik und Entwurfsverfahren (TI-1)

und

Rechnerorganisation (TI-2)

am 28. März 2022, 10:30 – 12:30 Uhr

- Beschriften Sie bitte gleich zu Beginn jedes Lösungsblatt deutlich lesbar mit Ihrem Namen und Ihrer Matrikelnummer.
- Diese Aufgabenblätter werden nicht abgegeben. Tragen Sie Ihre Lösung deshalb ausschließlich in die für jede Aufgabe vorgesehenen Bereiche der Lösungsblätter ein. Lösungen auf separat abgegebenen Blättern werden nicht gewertet.
- Außer Schreibmaterial sind während der Klausur keine Hilfsmittel zugelassen. Täuschungsversuche durch Verwendung unzulässiger Hilfsmittel führen unmittelbar zum Ausschluss von der Klausur und zur Note „nicht bestanden“.
- Soweit in der Aufgabenstellung nichts anderes angegeben ist, tragen Sie in die Lösungsblätter bitte nur Endergebnisse und Rechenweg ein. Die Rückseiten der Aufgabenblätter können Sie als Konzeptpapier verwenden. Weiteres Konzeptpapier können Sie auf Anfrage während der Klausur erhalten.
- Halten Sie Begründungen oder Erklärungen so kurz und präzise wie möglich. Der auf den Lösungsblättern für eine Aufgabe vorgesehene Platz lässt nicht auf den Umfang einer korrekten Lösung schließen.
- Die Gesamtpunktzahl beträgt 90 Punkte. Zum Bestehen der Klausur sind mindestens 40 Punkte zu erreichen.

Viel Erfolg und viel Glück!

Aufgabe 1 *Schaltfunktionen* (10 Punkte)

Gegeben sei die vollständig definierte Schaltfunktion $f(w, x, y, z)$:

$$f(w, x, y, z) = (\overline{w} \vee \overline{y})\ (x \vee y \vee w)\ (w \vee \overline{x} \vee \overline{z})\ (\overline{x} \vee \overline{y} \vee \overline{z})$$

1. Tragen Sie die Nullstellen der Schaltfunktion $f(w, x, y, z)$ in das im Lösungsblatt vorbereitete KV-Diagramm ein. 3 P.

2. Geben Sie alle Primimplikanten von f an. Kennzeichnen Sie, bei welchen Primimplikanten es sich um Kernprimimplikanten handelt. 2 P.

3. Geben Sie eine disjunktive Minimalform (DMF) von $f(w, x, y, z)$ an. 1 P.

4. Überführen Sie die DMF in ein Schaltnetz, in dem alle Eingangsvariablen nur bejaht zur Verfügung stehen und nur NOR-Gatter verwendet werden. 4 P.

Aufgabe 2 *Laufzeiteffekte* (8 Punkte)

Gegeben ist das in Abbildung 1 dargestellte Schaltnetz. NAND- und NOR-Gatter haben eine Totzeit von $5ns$, das OR-Gatter eine Totzeit von $3ns$ und die Inverter eine Totzeit von $2ns$. Die Eingangsvariablen a, b und c wechseln zum Zeitpunkt $t = 0$ *gleichzeitig* von 0 auf 1.

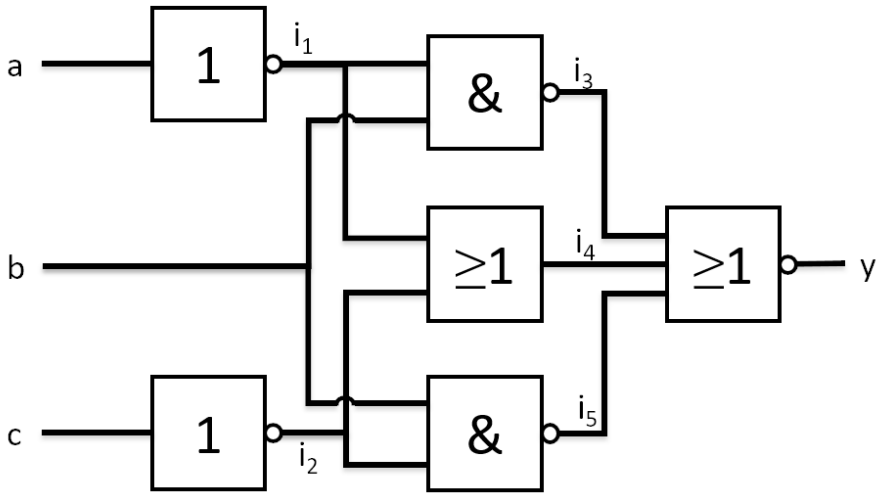


Abbildung 1: Schaltnetz

1. Geben Sie die Verläufe der Signale i_1, i_2, i_3, i_4, i_5 und y an, indem Sie das im Lösungsblatt angegebene Zeitdiagramm vervollständigen. 4 P.

2. Tritt im Zeitverlauf ein Hazardfehler auf? Falls ja, um welchen Typ handelt es sich? Begründen Sie Ihre Antwort. 4 P.

Aufgabe 3 *Boolesche Algebra* (11 Punkte)

1. Beweisen Sie durch schaltalgebraische Umformungen die Gültigkeit des Assoziativgesetzes für den zweistelligen Äquivalenzoperator: 3 P.

$$(a \leftrightarrow b) \leftrightarrow c \;=\; a \leftrightarrow (b \leftrightarrow c)$$

2. Zeigen Sie mit Hilfe der Regeln der booleschen Algebra, dass für drei boolesche Variablen a, b und c gilt: 3 P.

$$a \leftrightarrow b \leftrightarrow c \;=\; a \nleftrightarrow b \nleftrightarrow c$$

   Äquivalenz und Antivalenz sind assoziative Verknüpfungen, deswegen wurden die Klammern weggelassen.

3. Zeigen Sie, dass das Operatorensystem $\{\rightarrow, 0\}$ vollständig ist. 5 P.

Aufgabe 4 *Schaltwerke* (6 Punkte)

In Tabelle 1 ist die kodierte Ablauftabelle eines synchronen Schaltwerks dargestellt. Das Schaltwerk hat eine Eingangsvariable x^t und vier Zustände, die durch die Zustandsvariablen q_1^t, q_0^t dual kodiert sind. Zur Speicherung der Zustandsvariablen q_0 soll ein JK-Flipflop verwendet werden, während q_1 mit einem D-Flipflop realisiert werden soll.

q_0^t	q_1^t	x^t	q_0^{t+1}	q_1^{t+1}
0	0	0	1	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

Tabelle 1: Kodierte Ablauftabelle des Automaten

1. Tragen Sie die Ansteuerfunktionen der Flipflops in die Tabelle ein und geben Sie die dazugehörigen disjunktiven Minimalformen(DMF) an. 3 P.

2. Zeichnen Sie das Schaltwerk. 3 P.

Aufgabe 5 *Rechnerarithmetik* (10 Punkte)

1. Geben Sie $-\frac{1}{4}$ unter Verwendung des 32-Bit IEEE-754 Gleitkomma-Formats an. Ist die Darstellung exakt? 2 P.

2. Gegeben seien die Zahlen $s = 0xBFD3$, $t = 0xAB12$ und u als 16-Bit unsigned Hexadezimalzahlen. Berechnen Sie die Differenz $u = s - t$. 1 P.

3. $s = 0xBFD3$, $t = 0xAB12$ und u werden nun als 16-Bit signed Hexadezimalzahlen als Zweierkomplement interpretiert. Berechnen Sie die Differenz $u = s - t$. 1 P.

4. Ein BCD-Addierer für eine Tetrade soll aus zwei 4-Bit-Volladdierern und einem Schaltnetz zur Pseudotetraden- und Übertragskorrektur realisiert werden (siehe Abbildung 2). Ergänzen Sie das im Lösungsblatt wiederholt angegebene Schaltbild. 3 P.

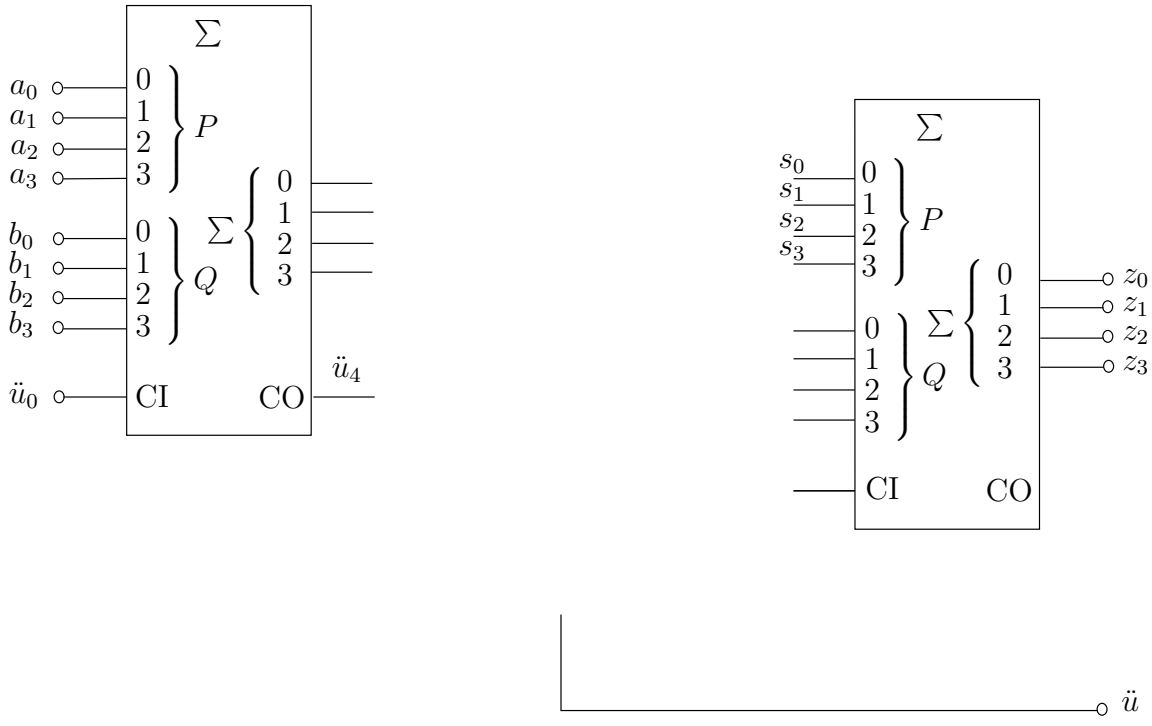


Abbildung 2: BCD-Addierer für eine Tetrade

Dabei stellt $(s_3 s_2 s_1 s_0)$ das Dual-Ergebnis der Addition der Tetraden $(a_3 a_2 a_1 a_0)$ und $(b_3 b_2 b_1 b_0)$ dar; $\ddot{u}_4$ ist der sich dabei ergebende Übertrag. $\ddot{u}$ ist der Übertrag der BCD-Addition der beiden Tetraden.

5. Der Hauptspeicher eines Rechners mit 8-Bit Datenwortbreite unterstützt eine Einzelfehler-Korrektur (Hamming-Code). Aus dem Speicher erhält man die beiden Codewörter: 3 P.

   - Codewort 1: **0 1 1 0 0 0 0 1 1 0 0 0**
   - Codewort 2: **1 0 0 0 0 0 0 1 1 0 0 1**

   Prüfen Sie beide Codewörter auf Fehler, die beim Übertragen oder Speichern entstanden sein könnten und korrigieren Sie diese (falls vorhanden). Geben Sie die zugehörigen Datenwörter an.

Aufgabe 6 *MIMA-Architektur* (8 Punkte)

Die MIMA ist die Ihnen aus der Vorlesung bekannte mikroprogrammierte Minimalmaschine (**siehe Beiblatt: Architektur der MIMA**). In der Lese-Phase wird ein über IAR adressierter Befehl aus dem Speicher gelesen und im IR abgelegt. Die Lese-Phase dauert 5 Taktzyklen. Im 6. Taktzyklus wird der Befehl dekodiert (Dekodier-Phase). Die Ausführungsphase beginnt im 7. Taktzyklus. Nach der Ausführung des Befehls folgt ein Zugriff auf den nächsten Befehl.

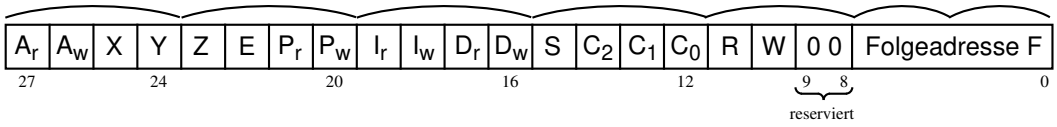
Nehmen Sie an, dass ein Hauptspeicherzugriff (Lesen und Schreiben) drei Takte dauert und währenddessen R = 1 bzw. W = 1 sein muss. Eine ALU-Operation sei nach einem Takt abgeschlossen.

Das Mikroprogramm für die Lese-Phase besteht aus fünf Mikrobefehlen:

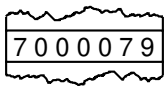
1. Takt: IAR → SAR; IAR → X; R = 1
2. Takt: Eins → Y; R = 1
3. Takt: ALU auf Addieren; R = 1
4. Takt: Z → IAR
5. Takt: SDR → IR

} Lese-Phase

1. Kodieren Sie das oben angegebene Mikroprogramm für die Lese-Phase. Das Mikroprogramm soll bei der 8-Bit-Adresse 0x00 beginnen. Verwenden Sie das folgende 28-Bit-Mikrobefehlformat: 4 P.



Beispiel:



A_w = X = Y = 1 (Akku -> X; Akku -> Y)
Adresse des nächsten Befehls ist 0x79

2. Geben Sie die Mikroprogramme für die Ausführungsphasen der folgenden Maschinenbefehle an (jeweils ab dem 7. Takt, also nach der Lese-Phase und der Dekodier-Phase): 4 P.

LDV, STV, EQL, JMP

Beispiel:

AND:

7. Takt: IR → SAR; R = 1
8. Takt: Akku → X; R = 1
9. Takt: ...
...

Aufgabe 7 *C und RISC-V Assembler* (10 Punkte)

1. Schreiben Sie das folgende C-Statement in RISC-V Assemblersprache. Gehen Sie davon aus, dass die Variablen f, g und h sich bereits in den Registern $t0$, $t1$ bzw. $t2$ befinden. Verwenden Sie die minimale Anzahl an Befehlen. 1 P.

```
f = g + (h - 5)
```

2. Schreiben Sie ein einziges C-Statement, welches die folgenden RISC-V Anweisungen umsetzt. Gehen Sie davon aus, dass die Register $t0$, $t1$, $t2$ und $t3$ mit den Variablen f, g, h bzw. i korrespondieren. 1 P.

```
add t0, t1, t2
add t0, t3, t0
```

3. Übersetzen Sie das folgende C-Statement in RISC-V Assemblersprache. Gehen Sie davon aus, dass die Variablen i und j den Registern $t0$ und $t1$ zugewiesen sind. Gehen Sie davon aus das die Basisadresse der Arrays A und B in den Registern $a0$ und $a1$ gespeichert sind. Gehen Sie zudem davon aus, dass die Elemente, welche in den Arrays gespeichert werden, Wörter der Größe 4 Byte sind. 4 P.

```
B[8] = A[i] + A[j]
```

4. Geben Sie für das folgende RISC-V Programmstück den Inhalt des Zielregisters in hexadezimaler Schreibweise nach der Ausführung des jeweiligen Befehls an. 4 P.

```
addi s1, zero, 0x28
srai s2, s1, 1
slti s3, s2, 10
lui s4, 0x21
xor s5, s4, s1
andi s6, s5, -1
```

Aufgabe 8 *Pipelining* (10 Punkte)

Gegeben seien die aus der Vorlesung bekannten Pipelinestufen mit den folgenden Latenzen:

IF	ID	EX	MEM	WB
250ps	350ps	250ps	400ps	200ps

1. Was ist die Zykluszeit bei einem Prozessor ohne Pipelining? Was ist sie bei einem Prozessor mit Pipelining? [1 P.]

2. Was ist die gesamte Latenz eines *lw* Befehls auf einem Prozessor ohne Pipelining und was auf einem Prozessor mit Pipelining? [1 P.]

3. Angenommen man könnte eine Pipelinestufe in zwei Stufen mit jeweils halber Latenz der Ursprungsstufe unterteilen. Welche Stufe würde sich hierfür am ehesten eignen und wie groß wäre dann die Zykluszeit des neuen Prozessors? [2 P.]

Betrachten Sie nun das folgende Programmstück:

```
S0:    addi t1, t0, 5
S1:    add  t2, t1, t0
S2:    addi t3, t1, 15
S3:    add  t4, t3, t1
```

Nehmen Sie an, dass das Programm auf einem Prozessor läuft, welcher Konflikte aufgrund von Datenabhängigkeiten nicht erkennt oder auflöst. Der Programmierer muss durch Einfügen von *NOPs* dafür sorgen, dass das Programm das gewünschte Ergebnis liefert. $t0$ sei zu Beginn mit 7 initialisiert; alle anderen Register mit 0.

4. Es seien noch keine *NOPs* eingefügt. Was ist der Inhalt der Register $t1$ bis $t4$ in dezimaler Schreibweise? Bedenken Sie, dass Register zu Beginn eines Zyklus beschrieben und am Ende eines Zyklus ausgelesen werden. Somit können Werte im gleichen Zyklus, wie sie zurückgeschrieben worden sind, dekodiert werden. [2 P.]

5. Modifizieren Sie das Programmstück mit *NOPs*, sodass die Sequenz das korrekte Ergebnis liefert. Nutzen Sie so wenige *NOPs* wie möglich. Sie dürfen die Reihenfolge der Befehle nicht ändern. Sie können die Marker S0-S3 als Abkürzung der Befehlszeile nutzen, um Schreibarbeit zu sparen. Was ist der Inhalt der Register $t1$ bis $t4$ in dezimaler Schreibweise? [4 P.]

Aufgabe 9 *Cache-Speicher* (9 Punkte)

Gegeben sind ein direkt-abgebildeter Cache *(direct mapped,* Abkürzung: DM), ein 8-fach satzassoziativer Cache *(8-way-set-associativ,* Abkürzung: A8) und ein vollassoziativer Cache *(fully-associativ,* Abkürzung: AV). Die drei Cache-Speicher haben jeweils eine Speicherkapazität von 128 Byte und werden in Blöcken von je 4 Byte geladen. Die Hauptspeicheradresse ist 32 Bit breit. Falls notwendig wird die »Least Recently Used«-Ersetzungsstrategie verwendet.

Betrachten Sie die Folge der Lesezugriffe auf die folgenden, in hexadezimaler Schreibweise angegebenen Hauptspeicheradressen:

0x0B, 0x2B, 0x07, 0x0C, 0x1E, 0x0A, 0x1A, 0x05, 0x04, 0x29

1. Skizzieren Sie die Unterteilung der Hauptspeicheradresse für die drei Cache-Architekturen. 3 P.

2. Geben Sie die Anzahl der erforderlichen Vergleicher für jede der drei Cache-Architekturen an. 1 P.

3. Nehmen Sie an, die Caches seien zu Beginn leer. Kennzeichnen Sie in der vorbereiteten Tabelle im Lösungsblatt für jeden Cache-Speicher, ob es sich beim Lesezugriff auf die jeweiligen Adressen um einen Treffer (Cache-Hit) oder um keinen Treffer (Cache-Miss) handelt. Verwenden Sie dabei »×« für Cache-Hit und »-« für Cache-Miss. 5 P.

Aufgabe 10 *Virtuelle Speicherverwaltung* (8 Punkte)

Die Speicherverwaltungseinheit in einem Rechnersystem bildet den virtuellen Adressraum der Größe von 16 GiByte auf den physikalischen von 128 MiByte ab. Die Seitengröße beträgt 4 KiByte.

1. Wie viele Bits umfasst die virtuelle Adresse? 1 P.
2. Wie viele Bits umfasst die physikalische Adresse? 1 P.
3. Wie viele Bits umfasst die virtuelle Seitennummer? 1 P.
4. Wie viele Bits umfasst die physikalische Seitennummer? 1 P.

Der Arbeitsspeicher eines Rechners sei bis auf drei Lücken der Größen 1600 Bytes, 1400 Bytes und 500 Bytes vollständig belegt (siehe Abbildung 3).

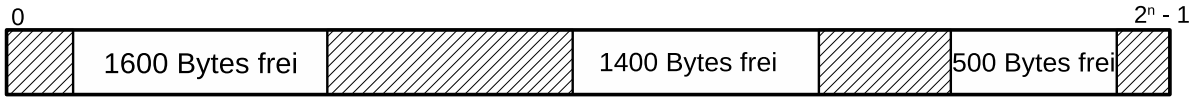
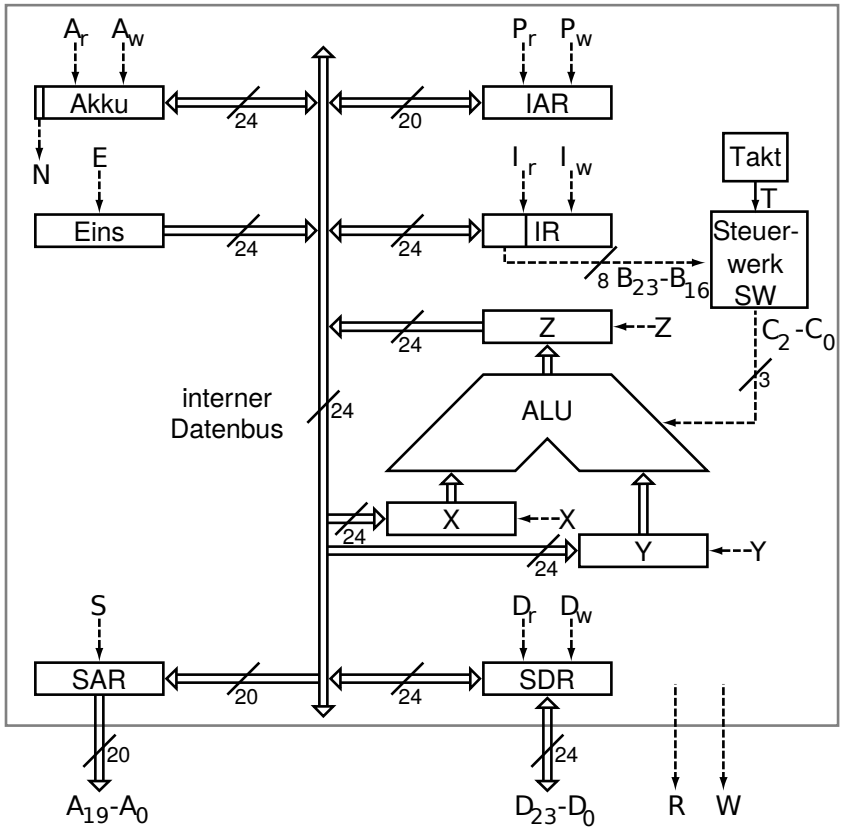


Abbildung 3: Arbeitsspeicher

Die nachfolgenden Speicheranforderungen benötigen Segmente der Größen 1100 Bytes, 1300 Bytes, 350 Bytes und 500 Bytes.

5. Wenden Sie die *first-fit*-Zuweisungsstrategie (beginnend bei niedrigen Adressen) und die *best-fit*-Zuweisungsstrategie auf diese Folge von Speicheranforderungen an. Zeichnen Sie hierzu die Speicherbelegung in das Schaubild auf dem Lösungsblatt nach den vier Speicheranforderungen ein. Welche der beiden Strategien ist günstiger? 2 P.
6. Vergleichen Sie beide Strategien bezüglich ihres Suchaufwands und der Fragmentierung. 2 P.

Architektur der MIMA



Register

Akku: Akkumulator
X: 1. ALU Operand
Y: 2. ALU Operand
Z: ALU Ergebnis
Eins: Konstante 1
IAR: Instruktionsadressregister
IR: Instruktionsregister
SAR: Speicheradressregister
SDR: Speicherdatenregister

Steuersignale vom SW

– für den internen Datenbus

A_r: Akku liest
A_w: Akku schreibt
X: X-Register liest
Y: Y-Register liest
Z: Z-Register schreibt
E: Eins-Register schreibt
P_r: IAR liest
P_w: IAR schreibt
I_r: IR liest
I_w: IR schreibt
D_r: SDR liest
D_w: SDR schreibt
S: SAR liest

– für die ALU

C_2-C_0: Operation auswählen

– für den Speicher

R: Leseanforderung
W: Schreibanforderung

Meldesignale zum SW

T: Takteingang
N: Vorzeichen des Akku
B_{23}-B_{16}: OpCode-Feld im IR

$C_2C_1C_0$	*ALU Operation*
0 0 0	tue nichts (d.h. Z → Z)
0 0 1	X + Y → Z
0 1 0	rotiere X nach rechts → Z
0 1 1	X AND Y → Z
1 0 0	X OR Y → Z
1 0 1	X XOR Y → Z
1 1 0	Eins-Komplement von X → Z
1 1 1	falls X = Y, -1 → Z, sonst 0 → Z

OpCode	*Mnemonik*	*Beschreibung*
0	LDC c	C → Akku
1	LDV a	<a> → Akku
2	STV a	Akku → <a>
3	ADD a	Akku + <a> → Akku
4	AND a	Akku AND <a> → Akku
5	OR a	Akku OR <a> → Akku
6	XOR a	Akku XOR <a> → Akku
7	EQL a	falls Akku = <a>:-1 → Akku sonst : 0 → Akku
8	JMP a	a → IAR
9	JMN a	falls Akku < 0 : a → IAR
F0	HALT	stoppt die MIMA
F1	NOT	bilde Eins-Komplement von Akku → Akku
F2	RAR	rotiere Akku eins nach rechts → Akku

Befehlsformate