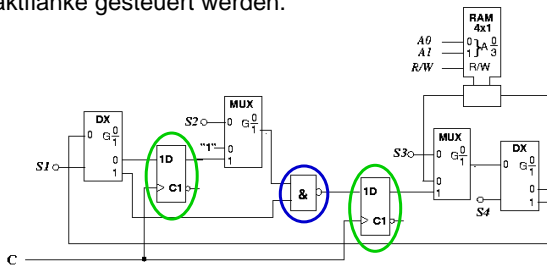


1. Übung

- Mikroprogrammierung (2 Aufgaben)
- MIMA-Architektur

Aufgabe 1

Mit einem einfachen Rechenwerk soll die Funktion $x_1 \leftrightarrow x_2$ implementiert werden. Der Datenfluss enthält lediglich ein NAND-Gatter zur logischen Verknüpfung zweier Operanden. Die Daten stehen in D-Flipflops, die mit ansteigender Taktflanke gesteuert werden.



Aufgabe 1.1

- Geben Sie einen schaltalgebraischen Ausdruck für die Antivalenz, der nur NAND-Verknüpfungen enthält.

$$x_1 \leftrightarrow x_2 = \overline{\overline{x_1} x_2 \vee x_1 \overline{x_2}}$$

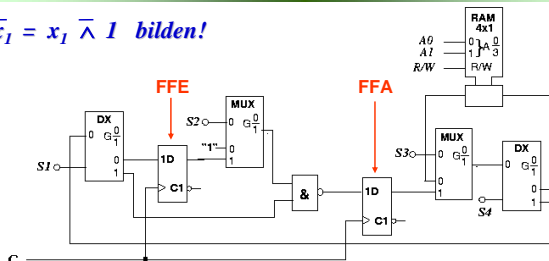
$$= (\overline{x_1} \wedge \overline{x_2}) \wedge (x_1 \wedge \overline{x_2})$$

$$\overline{x_1} = x_1 \wedge 1$$

$$\overline{x_2} = x_2 \wedge 1$$

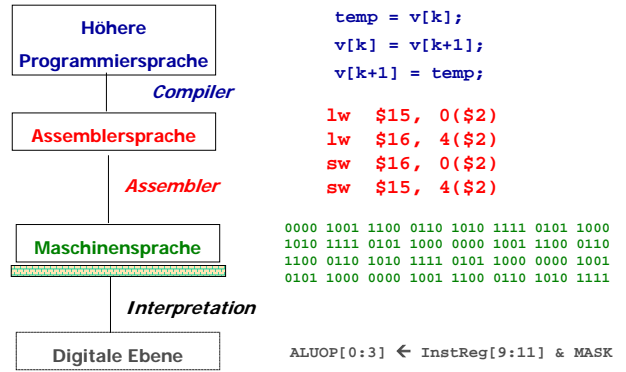
Aufgabe 1.2

$\overline{x_1} = x_1 \wedge 1$ bilden!



Takt	S1	S2	S3	S4	A1	A0	R/W	Kommentar
1.	1	0	0	1	0	0	1	$\overline{x_1}$ bilden und in Flipflop FFA speichern

Hierarchie



```

temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;

lw $15, 0($2)
lw $16, 4($2)
sw $16, 0($2)
sw $15, 4($2)
  
```

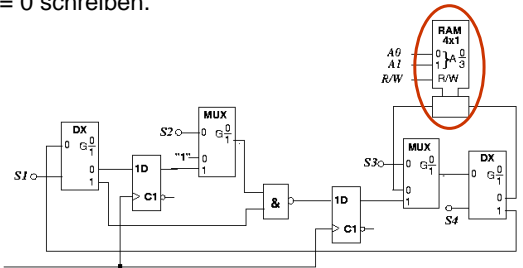
```

0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
  
```

```
ALUOP[0:3] ← InstReg[9:11] & MASK
```

Aufgabe 1

Die Variablen x_1 und x_2 stehen im RAM unter der Adresse 00 und 01, das Ergebnis soll in 11 abgelegt werden. Die Speicherzelle mit der Adresse 10 ist frei verfügbar z. B. für Zwischenergebnisse. Dabei bedeutet R/W = 1 lesen und R/W = 0 schreiben.



Aufgabe 1.2

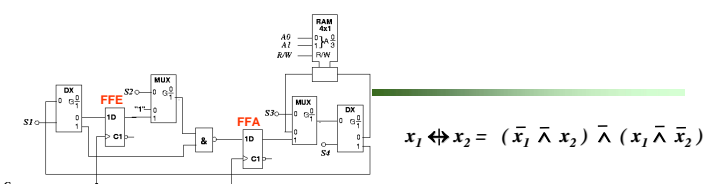
- Geben Sie die zur Steuerung des Datenflusses notwendigen Bitkombinationen (Belegungen der Steuervariablen) für die 2:1-MUX/DMUX an, die zur Berechnung der Antivalenz nötig sind.

Tragen Sie diese Bitkombinationen zeilenweise in einer Tabelle ein, wobei jede Zeile einer Periode des Taktes entspricht.

Wird mit einer Zeile (=Bitkombination) der RAM-Baustein angesprochen, so ist die Adresse in der Tabelle einzutragen.

Takt	S1	S2	S3	S4	A1	A0	R/W	Kommentar
1.	1	0	0	1	0	0	1	$\overline{x_1}$ → FFA

Aufgabe 1.3



$$x_1 \leftrightarrow x_2 = (\bar{x}_1 \bar{\wedge} x_2) \bar{\wedge} (x_1 \bar{\wedge} \bar{x}_2)$$

Takt	S1	S2	S3	S4	A1	A0	R/W	Kommentar
1.	1	0	0	1	0	0	1	$\bar{x}_1 \rightarrow$ FFA
2.	0	-	1	1	-	-	1	FFA \rightarrow FFE
3.	1	1	0	1	0	1	1	$\bar{x}_1 \bar{\wedge} x_2 \rightarrow$ FFA
4.	-	-	1	0	1	0	0	FFA \rightarrow RAM (adr. 10)
5.	1	0	0	1	0	1	1	$\bar{x}_2 \rightarrow$ FFA
6.	0	-	1	1	-	-	1	FFA \rightarrow FFE
7.	1	1	0	1	0	0	1	$x_1 \bar{\wedge} \bar{x}_2 \rightarrow$ FFA
8.	0	-	1	1	-	-	1	FFA \rightarrow FFE
9.	1	1	0	1	1	0	1	(<Adr. 10> $\bar{\wedge}$ FFE) \rightarrow FFA
10.	-	-	1	0	1	1	0	FFA \rightarrow RAM (adr. 11)

3. Entwerfen Sie eine „Programmiersprache“, d. h. Befehle in lesbarem Quellcode, wobei ein Befehl einer Zeile der Tabelle (= Objektcode) entspricht. Jeder Befehl soll die auszuführende Operation und ggf. die RAM-Adresse (Hauptspeicheradresse) enthalten.

Befehlsformat: **Operation [RAM-Adresse]**

Aufgabe 1.3

Es werden vier verschiedene Befehle benötigt, z. B.

➤ **NOT [adresse]**

Das Bit an der Adresse **adresse** holen und invertieren.
Das Ergebnis steht in FFA

➤ **NAND[adresse]**

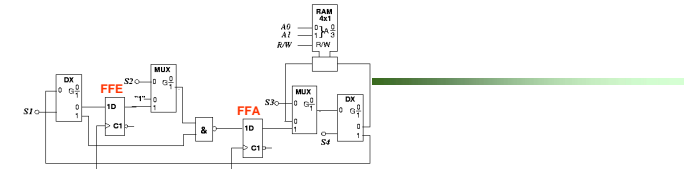
Das Bit an der Adresse **adresse** holen und mit dem Bit in FFE durch die NAND-Funktion verknüpfen.
Das Ergebnis steht in FFA

➤ **TAE**

Transferiert den Inhalt von FFA nach FFE

➤ **STA[adresse]**

Speichert den Inhalt von FFA an der Adresse **adresse**



Takt	S1	S2	S3	S4	A1	A0	R/W	Kommentar
1.	1	0	0	1	0	0	1	$\bar{x}_1 \rightarrow$ FFA
2.	0	-	1	1	-	-	1	FFA \rightarrow FFE
3.	1	1	0	1	0	1	1	$\bar{x}_1 \bar{\wedge} x_2 \rightarrow$ FFA
4.	-	-	1	0	1	0	0	FFA \rightarrow RAM (adr. 10)
5.	1	0	0	1	0	1	1	$\bar{x}_2 \rightarrow$ FFA
6.	0	-	1	1	-	-	1	FFA \rightarrow FFE
7.	1	1	0	1	0	0	1	$x_1 \bar{\wedge} \bar{x}_2 \rightarrow$ FFA
8.	0	-	1	1	-	-	1	FFA \rightarrow FFE
9.	1	1	0	1	1	0	1	(<Adr. 10> $\bar{\wedge}$ FFE) \rightarrow FFA
10.	-	-	1	0	1	1	0	FFA \rightarrow RAM (adr. 11)

Aufgabe 1.3

Quellcode	Kommentar
NOT[00]	$\bar{x}_1 \rightarrow$ FFA
TAE	FFA \rightarrow FFE
NAND[01]	$\bar{x}_1 \bar{\wedge} x_2 \rightarrow$ FFA
STA [10]	FFA \rightarrow RAM (adresse 10)
NOT[01]	$\bar{x}_2 \rightarrow$ FFA
TAE	FFA \rightarrow FFE
NAND[00]	$x_1 \bar{\wedge} \bar{x}_2 \rightarrow$ FFA
TAE	FFA \rightarrow FFE
NAND[10]	(<adr. 10> $\bar{\wedge}$ FFE) \rightarrow FFA
STA[11]	FFA \rightarrow RAM (adr. 11)

Aufgabe 1.4

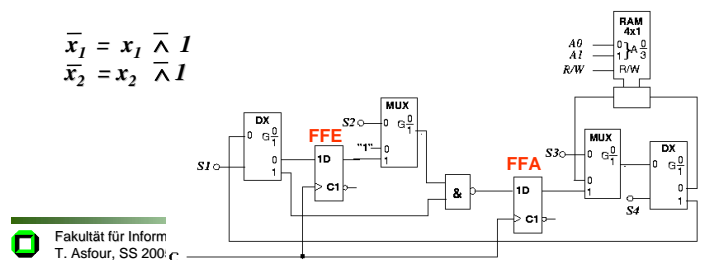
4. Geben Sie ein „Programm“ zur Berechnung der Äquivalenzfunktion $x_1 \leftrightarrow x_2$ an.

$$x_1 \leftrightarrow x_2 = \overline{\bar{x}_1 \bar{x}_2 \vee x_1 x_2}$$

$$= (\bar{x}_1 \bar{\wedge} \bar{x}_2) \bar{\wedge} (x_1 \bar{\wedge} x_2)$$

$$\bar{x}_1 = x_1 \bar{\wedge} 1$$

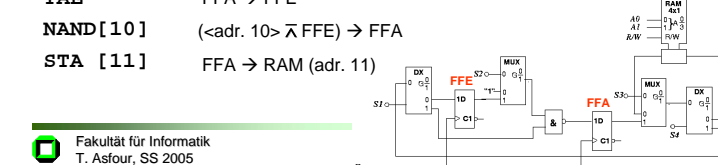
$$\bar{x}_2 = x_2 \bar{\wedge} 1$$



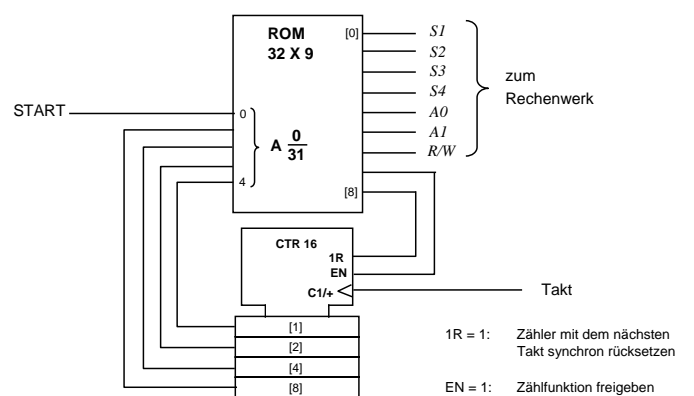
Aufgabe 1.4

Quellcode	Kommentar
NOT[00]	$\bar{x}_1 \rightarrow$ FFA
STA [10]	FFA \rightarrow adresse 10
NOT[01]	$\bar{x}_2 \rightarrow$ FFA
TAE	FFA \rightarrow FFE
NAND[10]	$\bar{x}_1 \bar{\wedge} \bar{x}_2 \rightarrow$ FFA
STA [10]	FFA \rightarrow adresse 10
LOAD[00]	$x_1 \rightarrow$ FFE
NAND [01]	$x_1 \bar{\wedge} x_2 \rightarrow$ FFA
TAE	FFA \rightarrow FFE
NAND[10]	(<adr. 10> $\bar{\wedge}$ FFE) \rightarrow FFA
STA [11]	FFA \rightarrow RAM (adr. 11)

➤ **LOAD[adresse]**
Holt den Inhalt der Adresse **adresse** ins FFE



Steuerwerk



zum Rechenwerk

Takt

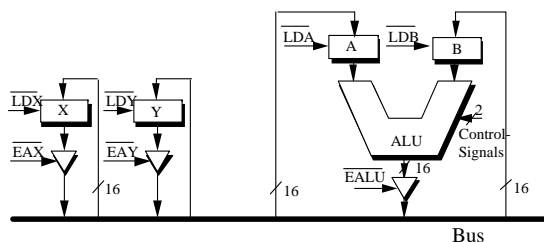
1R = 1: Zähler mit dem nächsten Takt synchron rücksetzen

EN = 1: Zählfunktion freigeben

Aufgabe 2

Gegeben:

Eine mikroprogrammierbare Schaltung besteht aus einem Bus, 4 Registern, einer ALU und einigen Tristate-Treibern

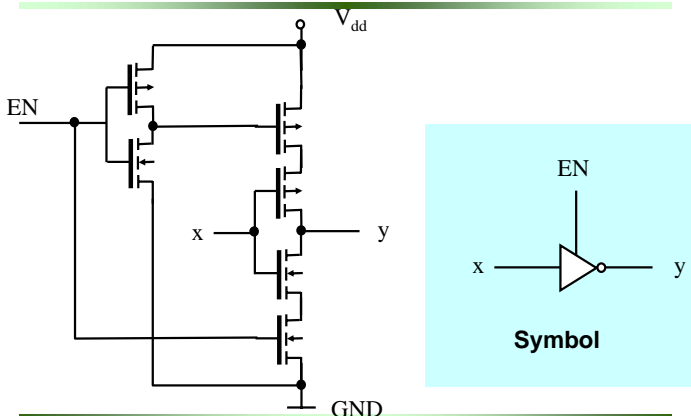


Tri-State-Treiber

Gatter, die neben den Pegelzuständen **H** (high) und **L** (low) einen **dritten hochohmigen Zustand** besitzen.

- In diesem Zustand ist der Ausgang hochohmig gegen Betriebsspannungen beider Polaritäten.
- Diese Gatter ermöglichen es, mehrere Gatterausgänge auf eine gemeinsame Leitung zusammenzuschalten (Bussystem)
- Sie dienen auch durch Ausgangstreiber zur elektrischen Anpassung der Prozessorsignale an die Signalspezifikationen, die von anderen Systemkomponenten verlangt werden.

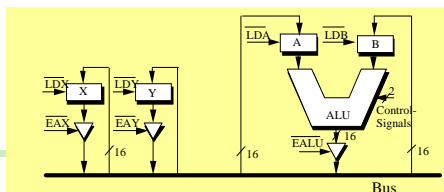
CMOS Tri-state-Inverter



Lösung 2

Mikroprogramme für die Operationen 1-4:

- X auf dem Bus legen. Warten bis die Daten stabil anliegen.
- X ins Register A laden.
- Y auf dem Bus legen. Warten bis die Daten stabil anliegen.
- Y ins Register B laden.
- Ergebnis auf den Bus. Warten bis es stabil anliegt.
- Ergebnis ins Register X laden.



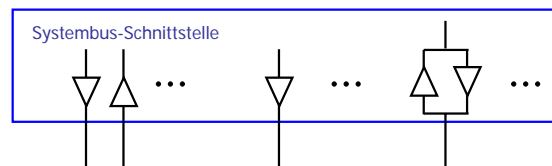
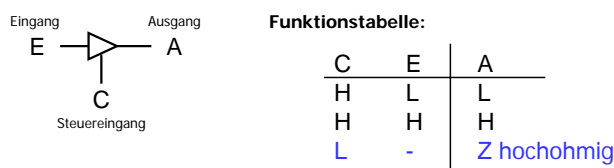
Aufgabe 2

Gesucht:

Mikroprogramme für die folgenden Operationen 1-5 durch Programmierung eines Datenpfades:

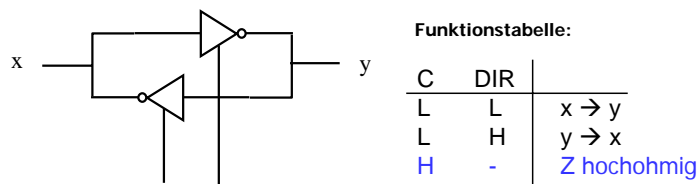
		Controlsignale	Operation
1.	$x = x + y$	C_1 C_0 0 0	A + B
2.	$x = x - y$	0 1	A - B
3.	$x = x \text{ and } y$	1 0	A und B
4.	$x = x \text{ or } y$	1 1	A oder B
5.	$y = x$ (move x to y)		

Tri-State-Treiber



Steuerleitungen Adressleitungen Datenleitungen (Bidirektional)

Bidirektionale Tri-state-Gatter



Lösung

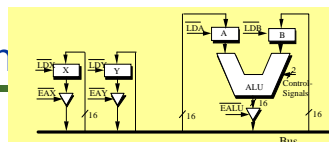
$\overline{\text{EAX}}$	0	0	1	1	1	1	
$\overline{\text{LDX}}$	1	1	1	1	1	0	
$\overline{\text{EAY}}$	1	1	0	0	1	1	
$\overline{\text{LDY}}$	1	1	1	1	1	1	
$\overline{\text{LDA}}$	1	0	1	1	1	1	
$\overline{\text{LDB}}$	1	1	1	0	1	1	
$\overline{\text{EALU}}$	1	1	1	1	0	0	
C_1	-	-	-	0	0	0	0 1 1
C_0	-	-	-	0	0	0	1 0 1

$x = x + y$

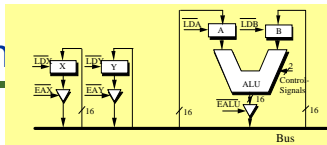
$x = x - y$

$x = x \text{ and } y$

$x = x \text{ or } y$



Lösung



Mikroprogramm für die Operation 5:

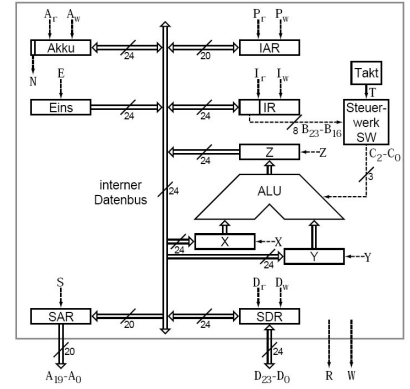
➤ X auf dem Bus legen. Warten bis die Daten stabil anliegen.

➤ Y laden.

EAX	0	0
LDX	1	1
EAY	1	1
LDY	1	0
LDA	1	1
LDB	1	1
EALU	1	1

Mima-Architektur (Siehe Übungsblatt 2)

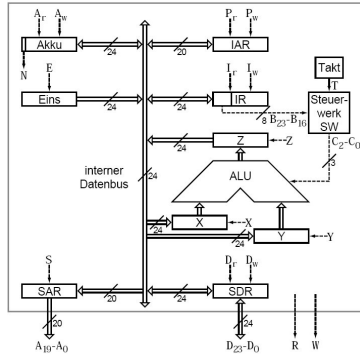
- Mikroprogrammierte Minimalmaschine (von-Neumann-Prinzip)
- SW mit 10 Meldesignale, 18 Steuersignale und Mikroprogrammspeicher für maximal 256 Mikrobefehle
- Befehlsabarbeitung:
 - Lese-Phase
 - Dekodierphase
 - Ausführungsphase
- 3 Taktzyklen für Lese- und Schreibzugriffe



Mima-Architektur

➤ Mikroprogramm für die Lesephase besteht aus 5 Mikrobefehlen:

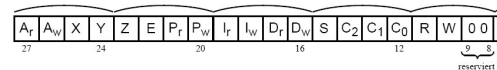
1. Takt: IAR → SAR; IAR → X; R=1
2. Takt: Eins → Y; R=1
3. Takt: ALU auf Addieren; R=1
4. Takt: Z → IAR
5. Takt: SDR → IR



Mima-Architektur

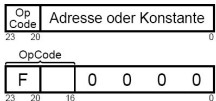
1. Takt: IAR → SAR; IAR → X; R = 1
2. Takt: Eins → Y; R = 1
3. Takt: ALU auf Addieren; R = 1
4. Takt: Z → IAR
5. Takt: SDR → IR

➤ Mikrobefehlsformat



Beispiel: 0x77: **700079** | $A_w = X = Y = 1$ (Akku → X; Akku → Y)
0x78: **700079** | Adresse des nächsten Befehls ist 0x79

Befehlsformate, ALU-Operationen, ...



C_2, C_1, C_0	ALU Operation
0 0 0	tue nichts (d.h. Z → Z)
0 0 1	$X + Y \rightarrow Z$
0 1 0	rotiere X nach rechts → Z
0 1 1	$X \text{ AND } Y \rightarrow Z$
1 0 0	$X \text{ OR } Y \rightarrow Z$
1 0 1	$X \text{ XOR } Y \rightarrow Z$
1 1 0	Eins-Komplement von X → Z
1 1 1	falls $X = Y, -1 \rightarrow Z$, sonst $0 \rightarrow Z$

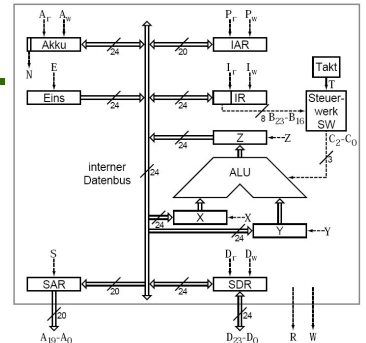
OpCode	Mnemonic	Beschreibung
0	LDC c	c → Akku
1	LDV a	<a> → Akku
2	STV a	Akku → <a>
3	ADD a	Akku + <a> → Akku
4	AND a	Akku AND <a> → Akku
5	OR a	Akku OR <a> → Akku
6	XOR a	Akku XOR <a> → Akku
7	EQL a	falls Akku = <a>: -1 → Akku sonst: 0 → Akku
8	JMP a	a → IAR
9	JMN a	falls Akku < 0: a → IAR
F0	HALT	stoppt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

Beispiel

Mikroprogramm für:

OR a Akku OR <a> → Akku

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



Mima-Architektur

➤ JAVA-Simulation der MIMA, Beispielprogramme und ein c-Interpreter auf der TI-Homepage:

<http://i61www.ira.uka.de/users/asfour/TI/Mima/>