
1. Übung

- ❑ Organisation
- ❑ Mikroprogrammierung
- ❑ Einführung in die Assemblerprogrammierung



Organisation

- Einführung in TI-1 für die Informationswirte:
 - Ort: Informatik-Hauptgebäude (HS -101)
 - Zeit: nächste Woche
 - Mi. 14.00 Uhr



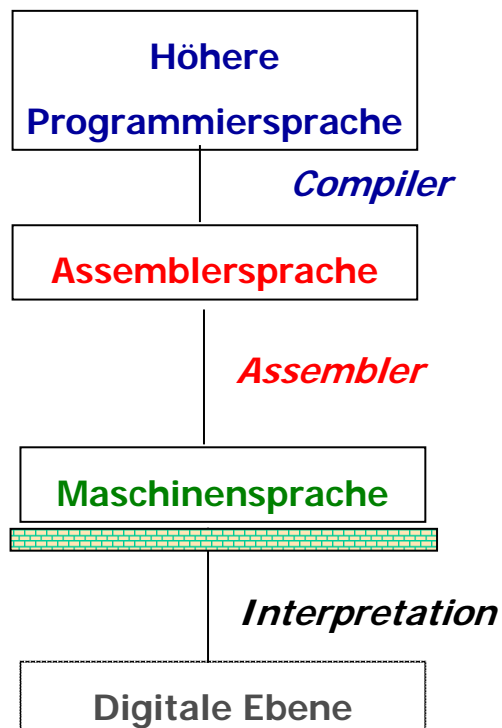
Organisation

- Keine Tests in der größten Übung

➔ Kein Bonussystem



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)
```

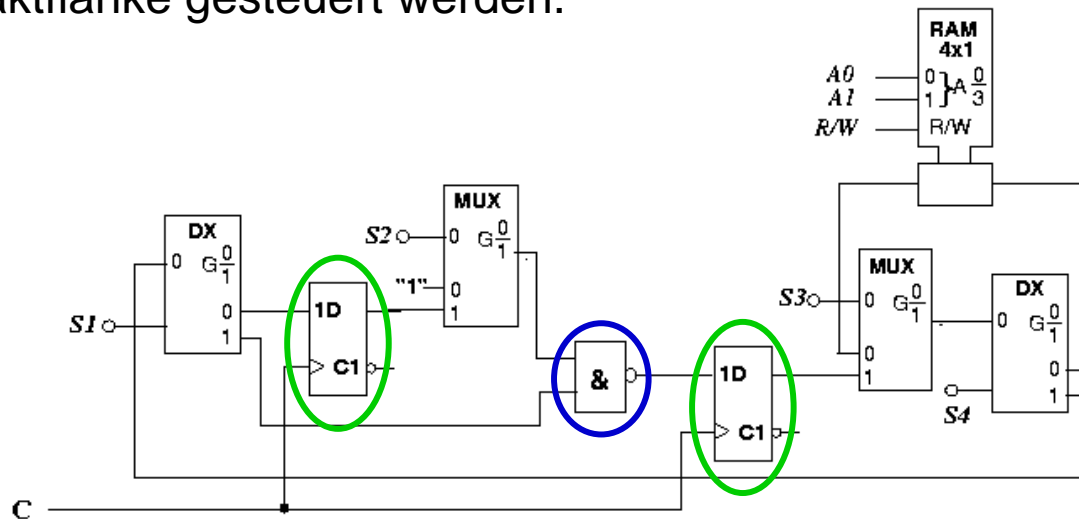
```
0000 1001 1100 0110 1010 1111 0101 1000  
1010 1111 0101 1000 0000 1001 1100 0110  
1100 0110 1010 1111 0101 1000 0000 1001  
0101 1000 0000 1001 1100 0110 1010 1111
```

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



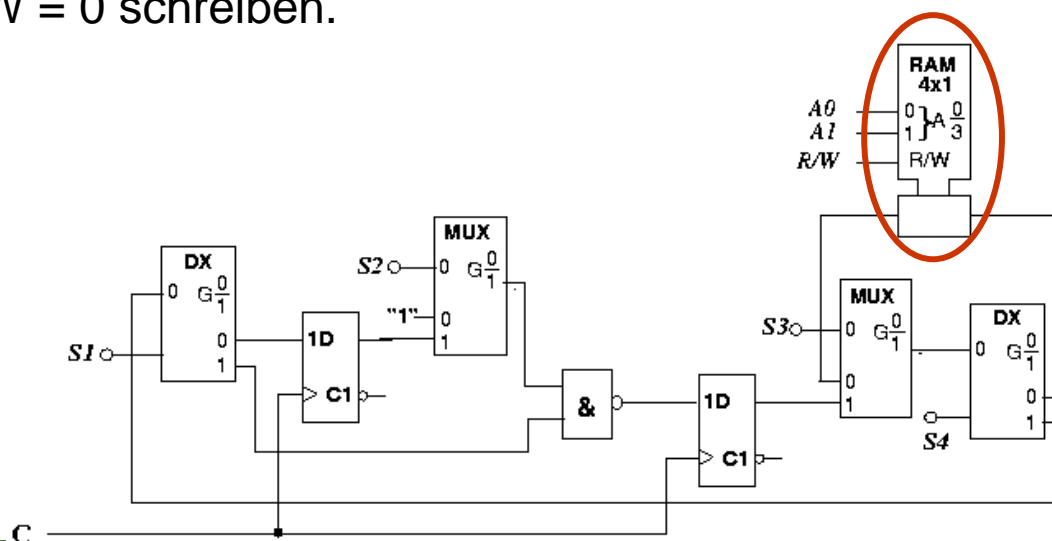
Aufgabe 1

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

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 1.1

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

$$\begin{aligned}x_1 \leftrightarrow x_2 &= \overline{\overline{x_1} x_2 \vee x_1 \overline{x_2}} \\&= (\overline{x_1} \wedge x_2) \wedge (x_1 \wedge \overline{x_2})\end{aligned}$$

$$\begin{aligned}\overline{x_1} &= x_1 \wedge \overline{1} \\ \overline{x_2} &= x_2 \wedge \overline{1}\end{aligned}$$



Aufgabe 1.2

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.

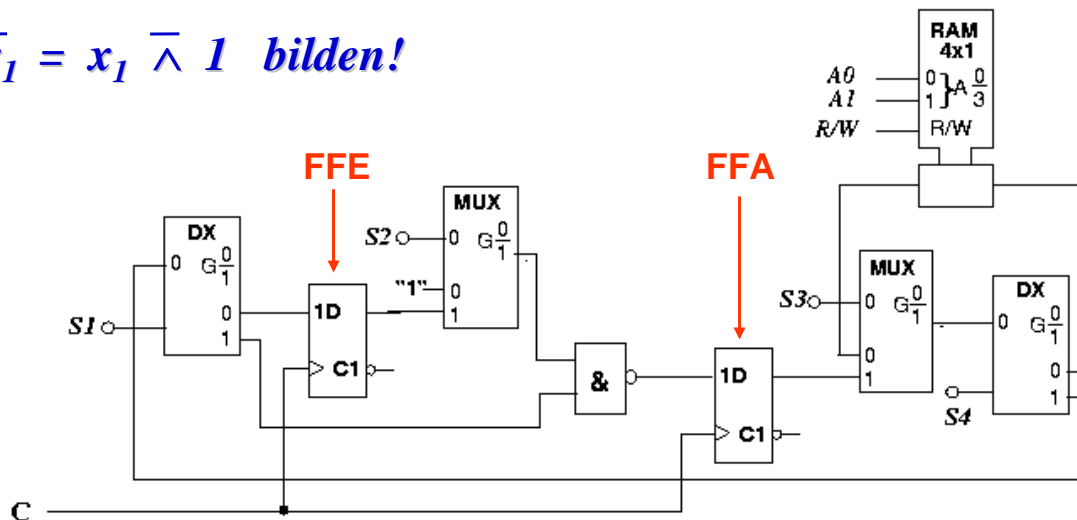
Tragen Sie diese Bitkombinationen zeilenweise in einer Tabelle ein, wobei jede Zeile einer Periode des Taktes entspricht.

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



Aufgabe 1.2

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

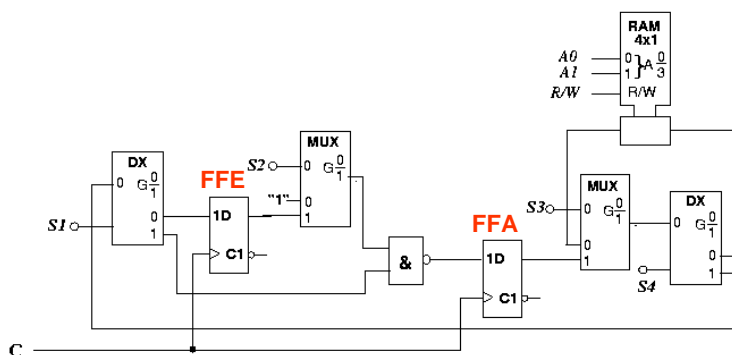


Takt	S1	S2	S3	S4	A1	A0	R/W	Kommentar
	1	0	0	1	0	0	1	\bar{x}_1 bilden und in Flipflop FFA speichern



Fakultät für Informatik
T. Asfour, SS04

Ü1-9

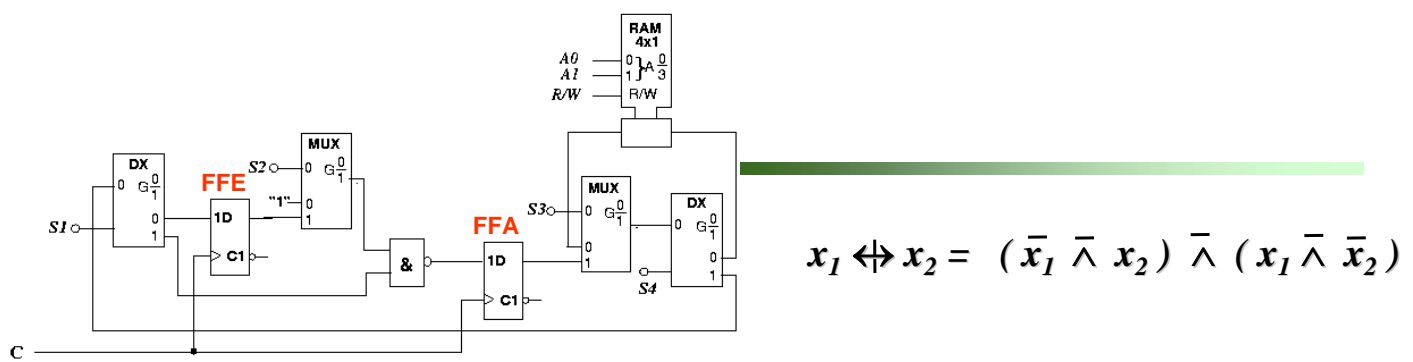


Takt	S1	S2	S3	S4	A1	A0	R/W	Kommentar
1.	1	0	0	1	0	0	1	$\bar{x}_1 \rightarrow$ FFA



Fakultät für Informatik
T. Asfour, SS04

Ü1-10



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

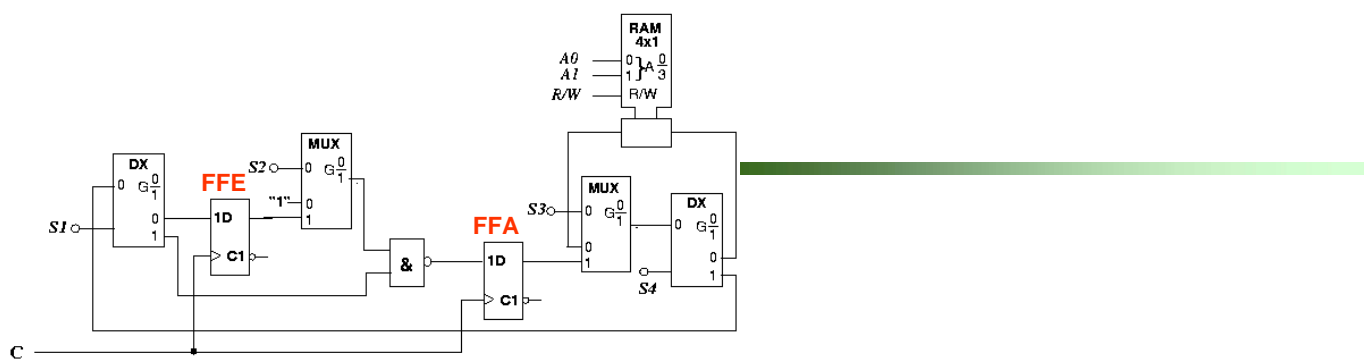


Aufgabe 1.3

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]**





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



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**



 Fakultät für Informatik
T. Asfour, SS04

-  Fakultät für Inform
T. Asfour, SS04

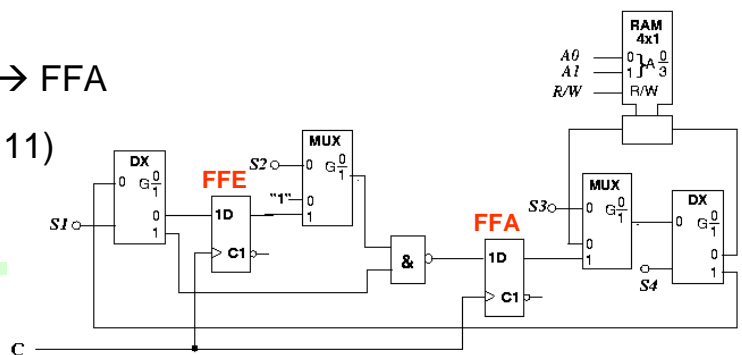
Aufgabe 1.4

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

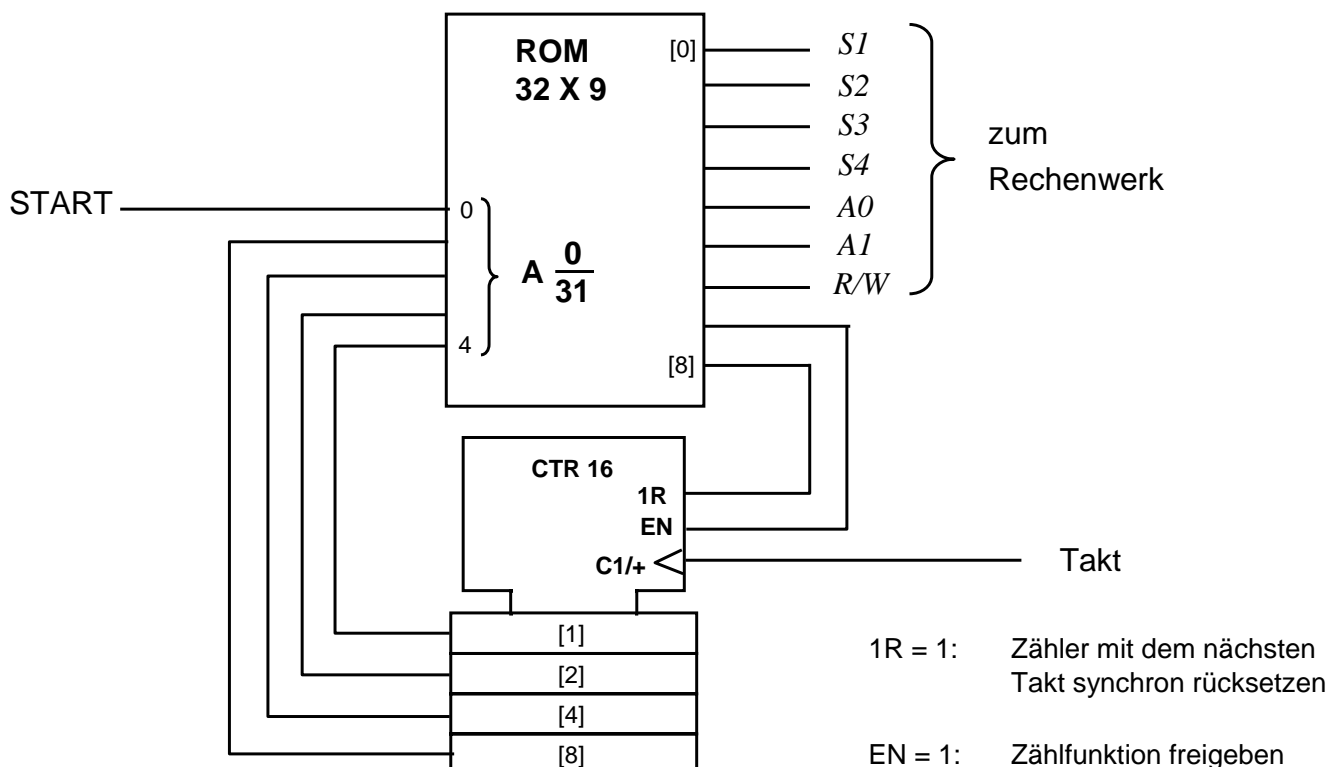
➤ **LOAD[adresse]**
Holt den Inhalt der Adresse **adresse** ins FFE



Fakultät für Informatik
T. Asfour, SS04



Steuerwerk

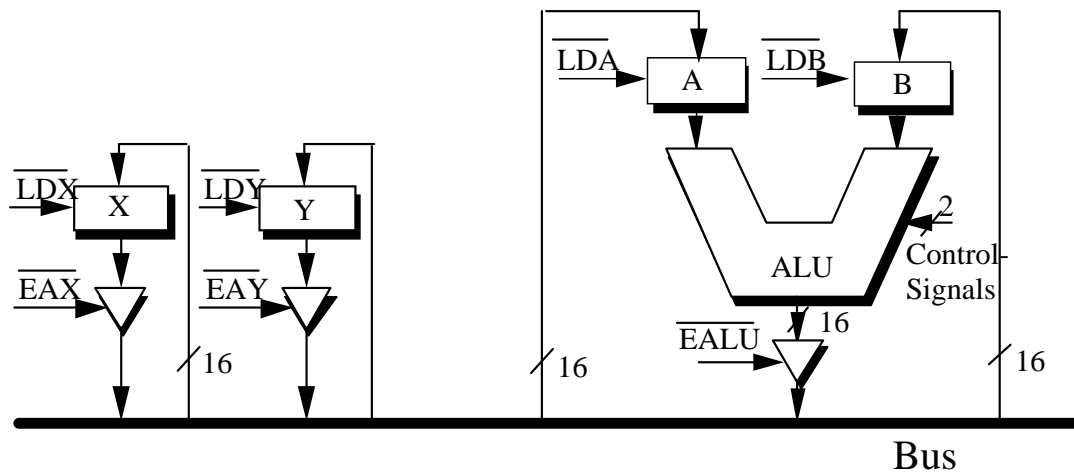


Fakultät für Informatik
T. Asfour, SS04

Aufgabe 2

Gegeben:

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



Aufgabe

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)		



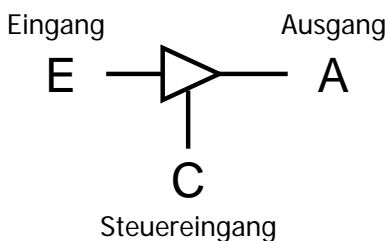
Tri-State-Treiber

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.

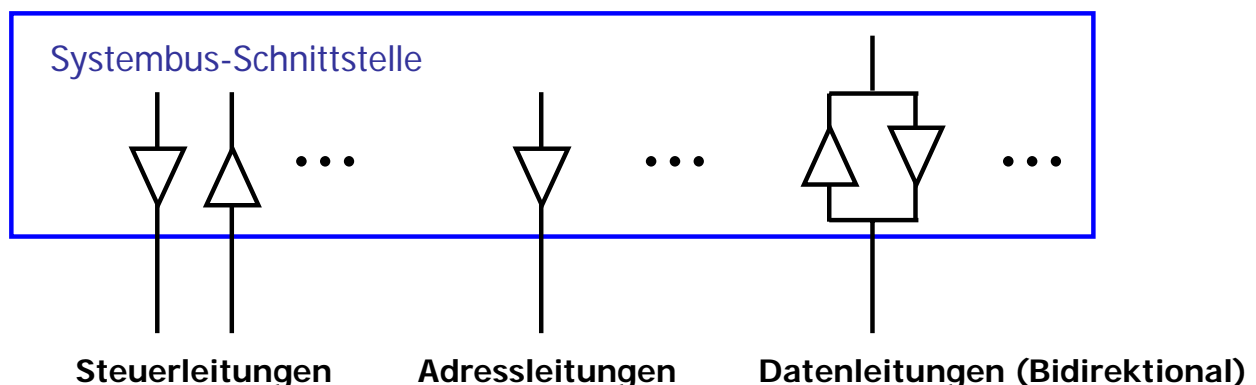


Tri-State-Treiber

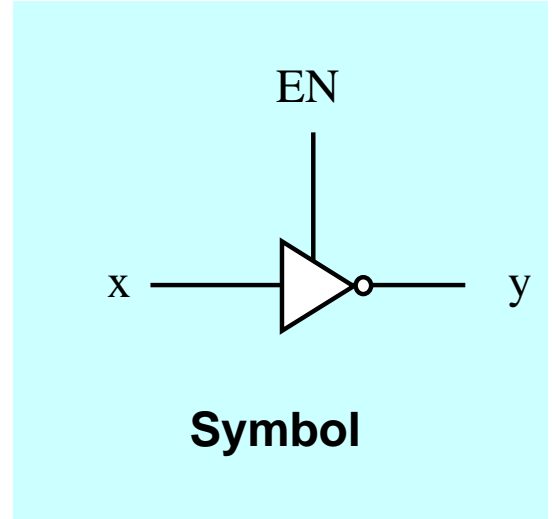
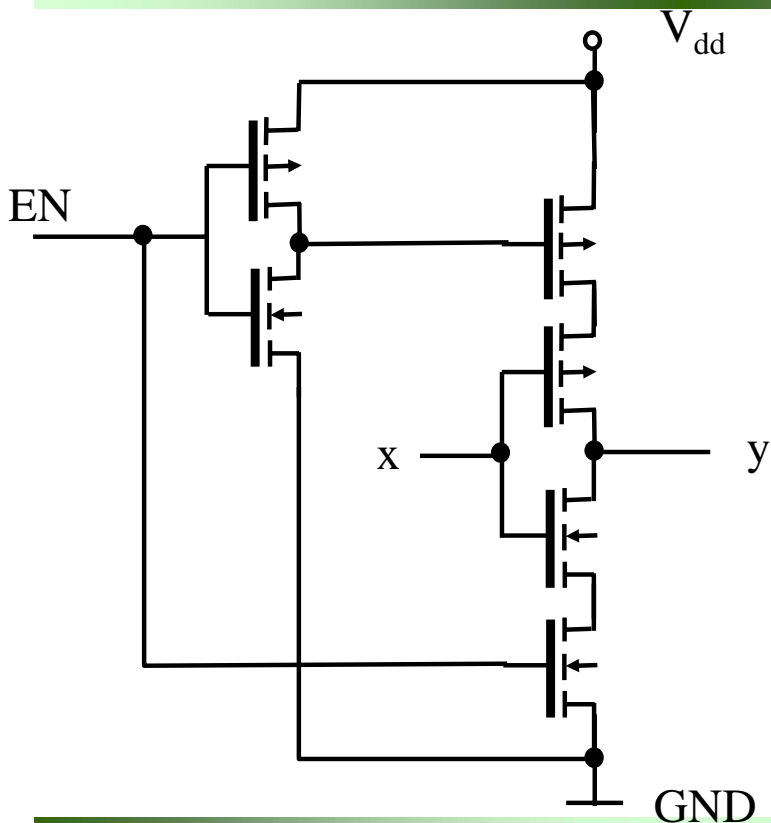


Funktionstabelle:

C	E	A
H	L	L
H	H	H
L	-	Z hochohmig



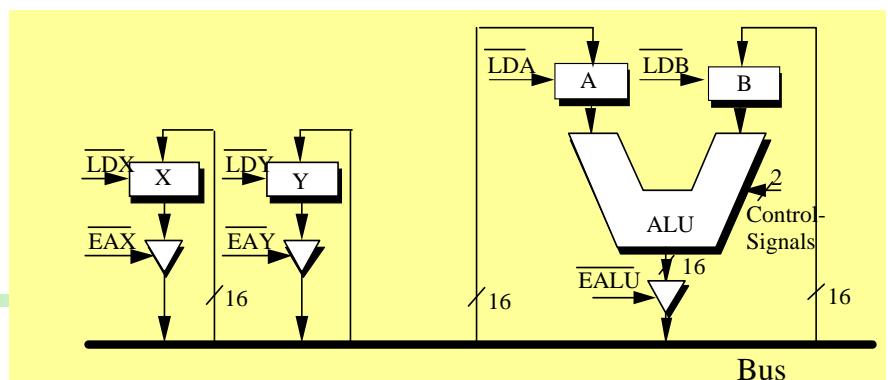
CMOS Tri-state-Inverter



Lösung

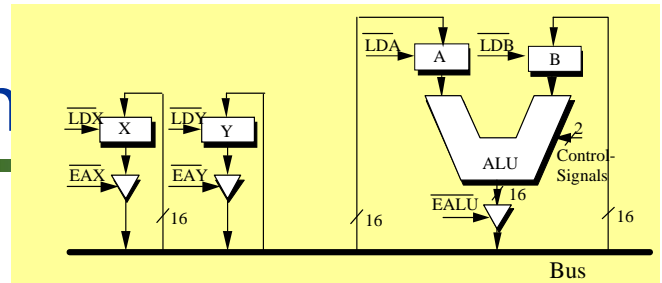
Mikroprogramme für die Operationen 1-4:

- 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.



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
C_0	-	-	-	0	0	0



$x = x + y$

0 1 1

$x = x - y$

$x = x$ and y

1 0 1

$x = x$ or y

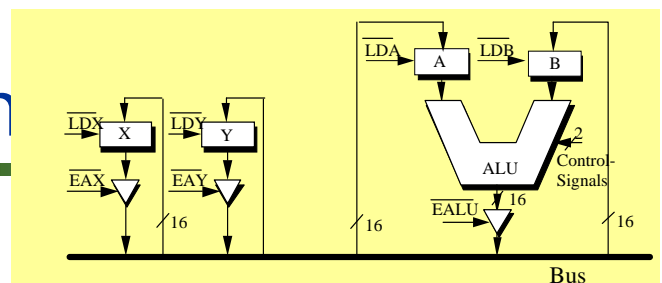


Lösung

Mikroprogramm für die
Operation 5:

- X auf dem Bus legen. Warten bis die Daten stabil anliegen.
- Y laden.

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



□ Programm-Darstellung

- Symbolische Darstellung
- Maschinencode-Darstellung

□ Programm-Übersetzung

- Assemblersprache
- Assembleranweisungen
- Assemblierung



Begriffe

Maschinensprache: Repräsentation von Anweisungen, die für einen Mikroprozessor unmittelbar verständlich sind, z. B.

00000000110000100011000000100001

Assemblersprache: Symbolische Repräsentation der Maschinensprache, die für den Menschen verständlich und anschaulich ist, z. B.

add R1, R1, R2 # R1 := R1 + R2

Symbolischer Befehl ≡ Maschinen-Befehl



Programmieraufgabe

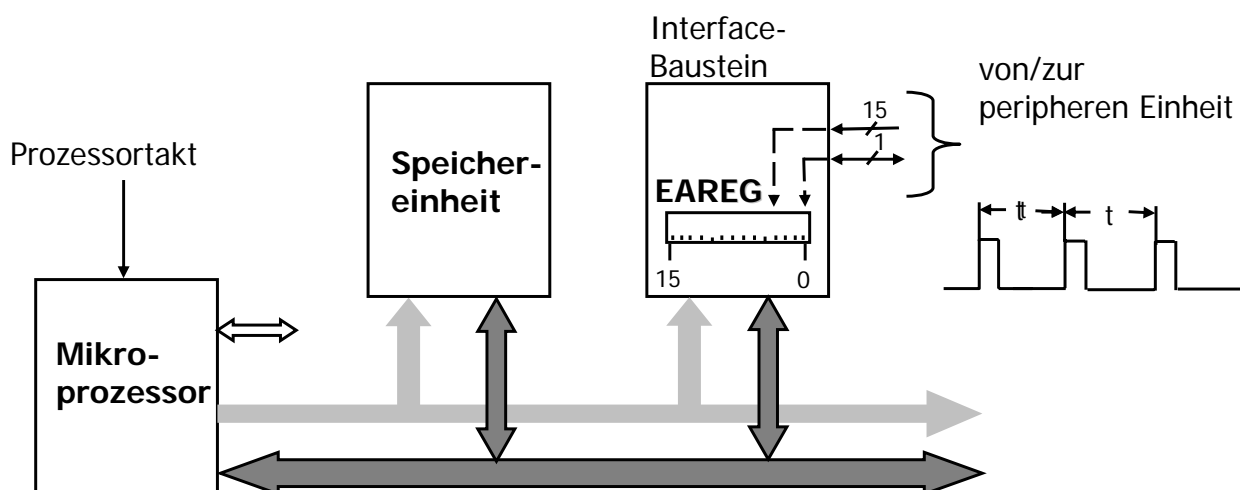
Aufgabenstellung

Es soll ein Impulsgeber mit Hilfe eines μ P-Systems aufgebaut werden, der in konstanten Zeitabständen Impulse an eine Ein-/Ausgabeeinheit abgibt

- Festlegen eines Satzes von Maschinenbefehlen zur Lösung dieser Aufgabe
- Programm in der Assemblersprache und in der Maschinensprache



Programmieraufgabe



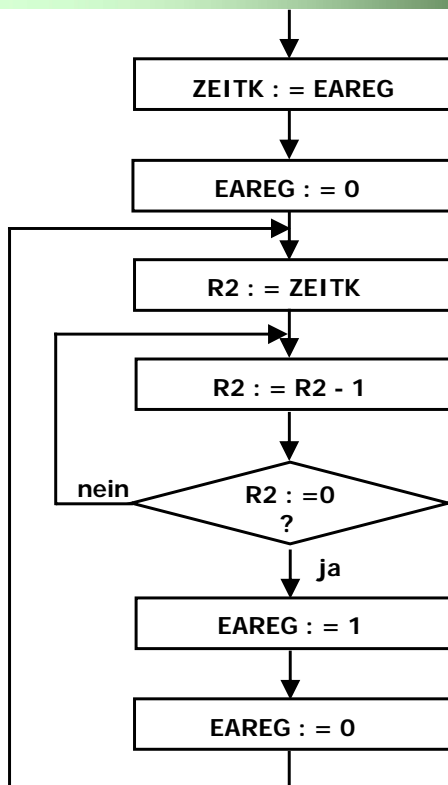
EA-Einheit mit einem 16 Bit Register **EAREG**

Absolute Adresse von EAREG: **\$8000 (32768)**

Periodendauer der Impulsfolge: **t**



Programmablauf in Form eines Flussdiagramms



- Periodendauer aus EAREG in die Hauptspeicherzelle **ZEITK**
- **EAREG** mit **NULL** initialisieren
(**NULL** und **EINS** sind Speicherzellen mit konstanten Operanden)
- Periodendauer ist bestimmt durch die Anzahl der Durchläufe der innere Schleife und die Verarbeitungszeiten der einzelnen Befehlen



Notwendige Befehle zur Lösung

MOVE QADR, ZADR

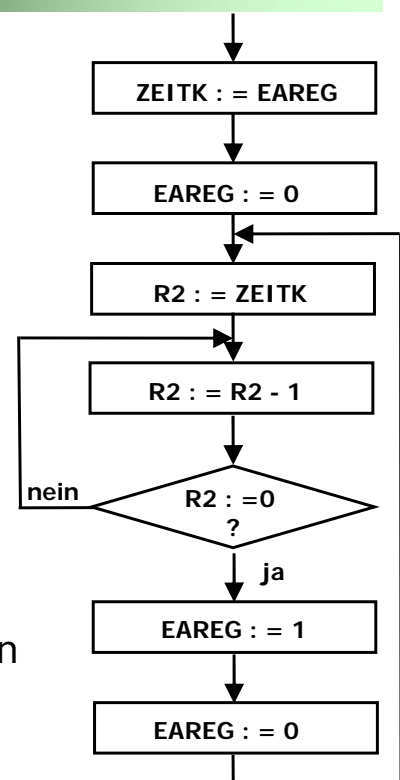
Inhalt von QADR (Quelladresse)
nach ZADR (Zieladresse)

SUB QADR, ZADR

Subtrahiere den Inhalt von QADR
vom Inhalt von ZADR und schreibe
das Ergebnis in ZADR

CMP ADR1, ADR2

Vergleiche die mit ADR1 und ADR2 adressierten
Operanden. Ergebnis in den CC-Bits des
Prozessor-Statusregisters (Dual- und
2-Komplement-Zahlen)



Notwendige Befehle zur Lösung

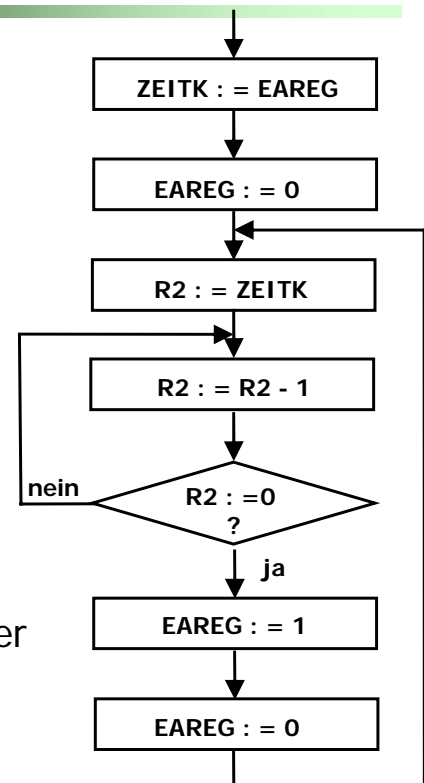
BNE SPRADR

Bedingter Sprung:

lade den Befehlszähler mit der Sprungadresse SPRADR, sofern das Prozessor-Statusregister den Zustand "ungleich" zeigt, sonst wird der nächste Befehl im Programm ausgeführt

JMP SPRADR

Unbedingter Sprung: lade den Befehlszähler mit der Sprungadresse SPRADR



Programm in symbolischer Darstellung

	MOVE	EAREG, ZEITK	6
	MOVE	NULL, EAREG	6
	MOVE	NULL, R0	4
MARKE1	MOVE	ZEITK, R2	4
MARKE2	SUB	EINS, R2	5
	CMP	R0, R2	4
	BNE	MARKE2	3
	MOVE	EINS, EAREG	6
	MOVE	NULL, EAREG	6
	JMP	MARKE1	3
ZEITK			
NULL	0		
EINS	1		

↑
Takte/Befehl

