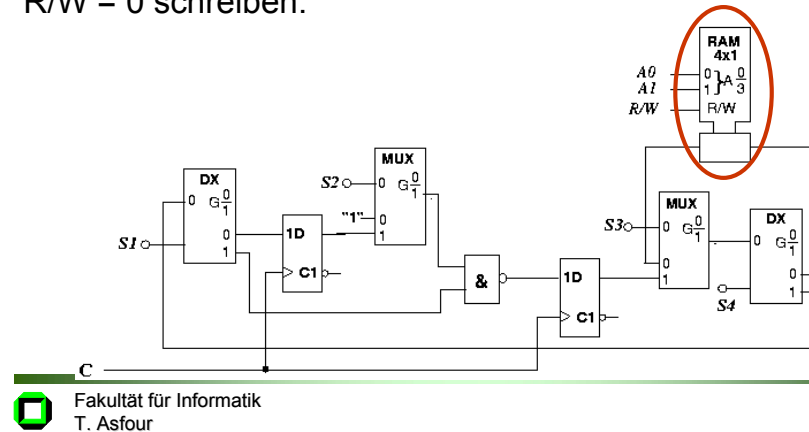


1. Übung

- ❑ **Mikroprogrammierung (2 Aufgaben)**
- ❑ **Einführung in die Assemblerprogrammierung**

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.

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

$$\begin{aligned}\bar{x}_1 &= x_1 \wedge 1 \\ \bar{x}_2 &= x_2 \wedge 1\end{aligned}$$

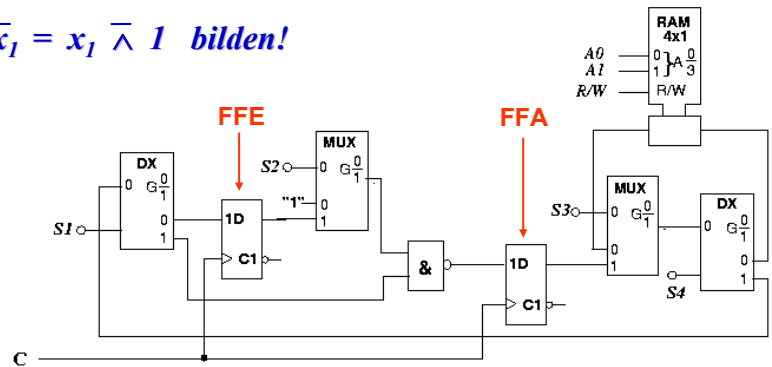
Aufgabe 1.2

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.

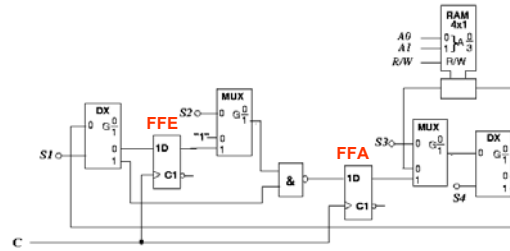
Tragen Sie diese Bitkombinationen zeilenweise in einer Tabelle ein, wobei jede Zeile einer Periode des Taktes entspricht. Wird mit einer Zeile (=Bitkombination) der RAM-Baustein angesprochen, so ist die Adresse in der Tabelle einzutragen.

Aufgabe 1.2

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



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



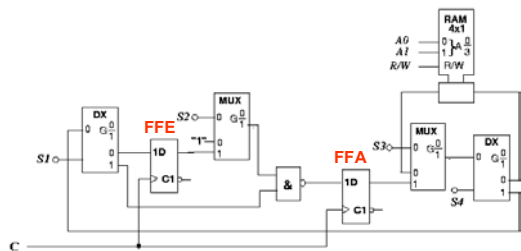
Takt	S1	S2	S3	S4	A1	A0	R/W	Kommentar
1.	1	0	0	1	0	0	1	$\bar{x}_1 \rightarrow$ FFA



Aufgabe 1.3

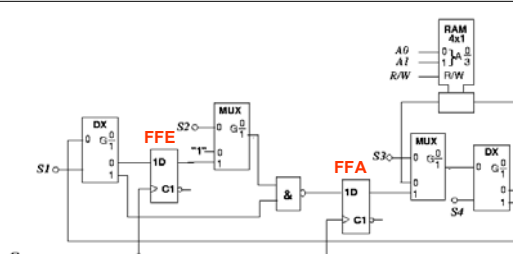
- Entwerfen Sie eine „Programmiersprache“, d. h. Befehle in lesbarem Quellcode, wobei ein Befehl einer Zeile der Tabelle (= Objektcode) entspricht. Jeder Befehl soll die auszuführende Operation und ggf. die RAM-Adresse (Hauptspeicheradresse) enthalten.

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



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





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	$(\text{<Adr. 10>} \bar{\wedge} \text{FFE}) \rightarrow \text{FFA}$
10.	-	-	1	0	1	1	0	$\text{FFA} \rightarrow \text{RAM (adr. 11)}$

Fakultät für Informatik
T. Asfour

Ü1-10

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**

Aufgabe 1.3

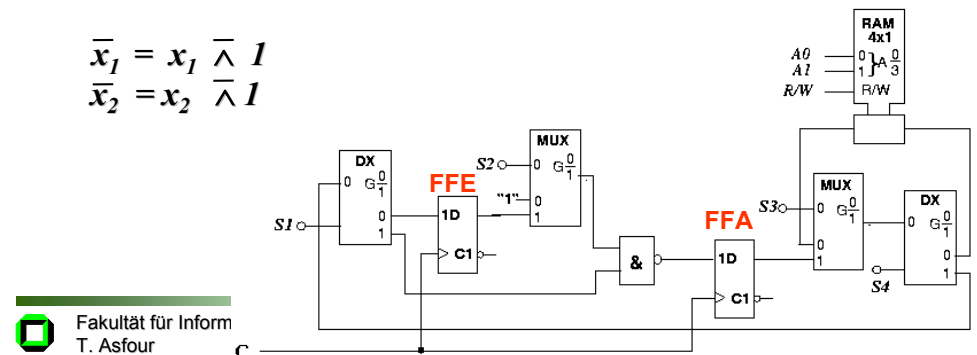
Quellcode	Kommentar
NOT [00]	$\bar{x}_1 \rightarrow \text{FFA}$
TAE	$\text{FFA} \rightarrow \text{FFE}$
NAND [01]	$\bar{x}_1 \bar{\wedge} x_2 \rightarrow \text{FFA}$
STA [10]	$\text{FFA} \rightarrow \text{RAM (adresse 10)}$
NOT [01]	$\bar{x}_2 \rightarrow \text{FFA}$
TAE	$\text{FFA} \rightarrow \text{FFE}$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

Aufgabe 1.4

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

$$\begin{aligned}
 x_1 \leftrightarrow x_2 &= \overline{\bar{x}_1 \bar{x}_2 \vee x_1 x_2} \\
 &= (\bar{x}_1 \bar{\wedge} \bar{x}_2) \bar{\wedge} (x_1 \bar{\wedge} x_2)
 \end{aligned}$$

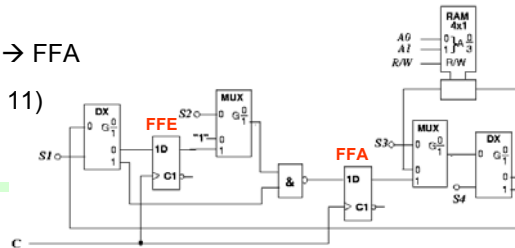
$$\begin{aligned}
 \bar{x}_1 &= x_1 \bar{\wedge} 1 \\
 \bar{x}_2 &= x_2 \bar{\wedge} 1
 \end{aligned}$$



Aufgabe 1.4

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

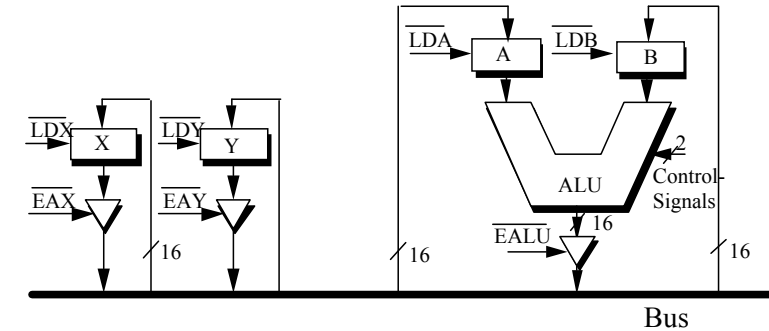
- **LOAD[adresse]**
Holt den Inhalt der Adresse **adresse** ins FFE



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:

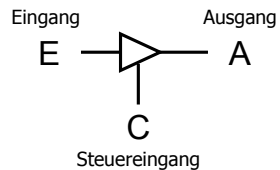
		Controlsignale	Operation
1.	$x = x + y$	$\begin{matrix} C_1 & C_0 \\ 0 & 0 \end{matrix}$	A + B
2.	$x = x - y$	$\begin{matrix} 0 & 1 \end{matrix}$	A - B
3.	$x = x \text{ and } y$	$\begin{matrix} 1 & 0 \end{matrix}$	A und B
4.	$x = x \text{ or } y$	$\begin{matrix} 1 & 1 \end{matrix}$	A oder B
5.	$y = x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

Tri-State-Treiber

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

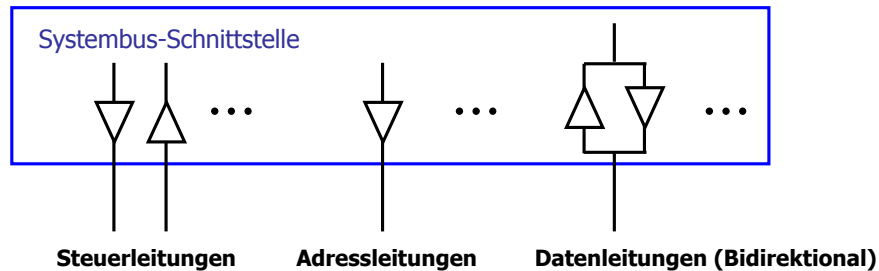
- In diesem Zustand ist der Ausgang hochohmig gegen Betriebsspannungen beider Polaritäten.
- Diese Gatter ermöglichen es, mehrere Gatterausgänge auf eine gemeinsame Leitung zusammenzuschalten (Bussystem)
- Sie dienen auch durch Ausgangstreiber zur elektrischen Anpassung der Prozessorsignale an die Signalspezifikationen,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

Tri-State-Treiber



Funktionstabelle:

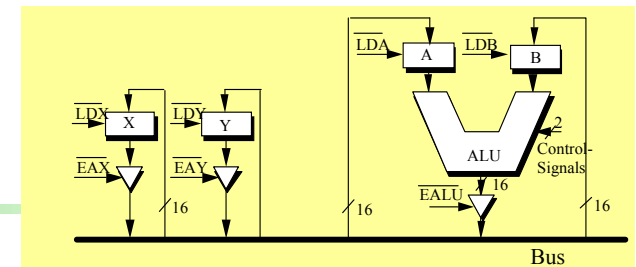
C	E	A
H	L	L
H	H	H
L	-	Z hochohmig



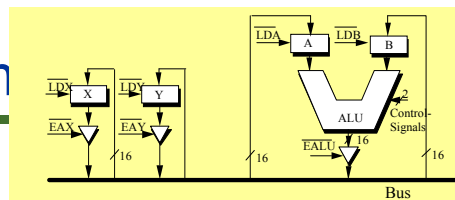
Lösung

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.



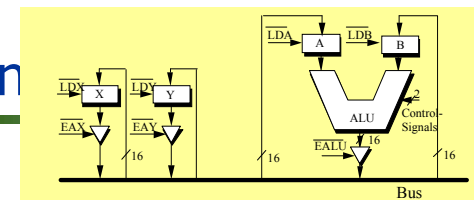
Lösun



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

$x = x + y$
 $x = x - y$
 $x = x \text{ and } y$
 $x = x \text{ or } y$

Lösun



Mikroprogramme 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

□ Programm-Darstellung

- Symbolische Darstellung
- Maschinencode-Darstellung

□ Programm-Übersetzung

- Assemblersprache
- Assembleranweisungen
- Assemblierung



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

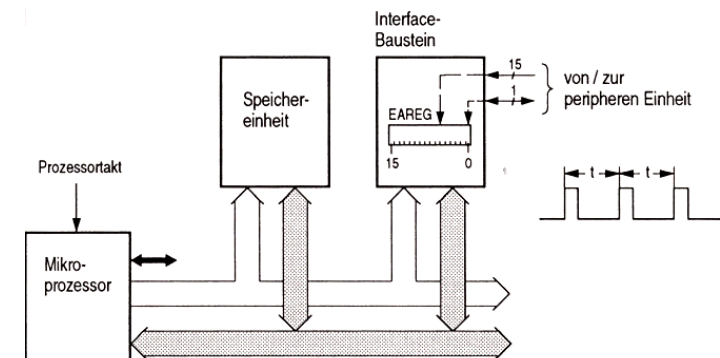
Aufgabenstellung

Es soll ein Impulsgeber mit Hilfe eines mP-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



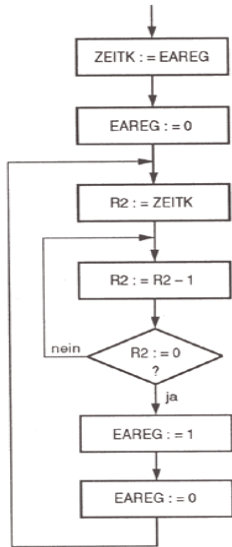
EA-Einheit mit einem 16 Bit Register **EAREG**

Absolute Adresse von EAREG: **0x8000 (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 Befehle



Notwendige Befehle zur Lösung

MOVE QADR, ZADR

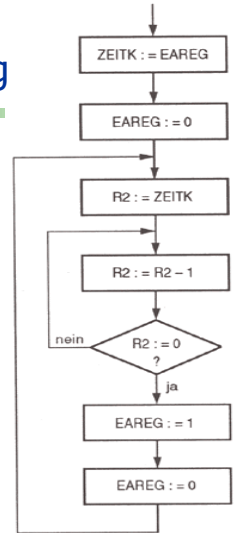
Inhalt von QADR (Quelladresse)
nach ZADR (Zieladresse)

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

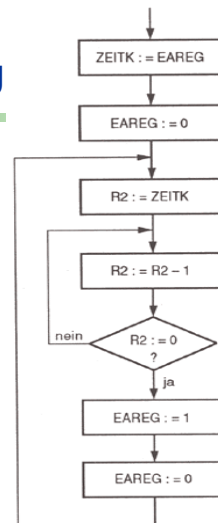
BNE SPRADR

Bedingter Sprung:

lade den Programmzähler 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 Programmzähler
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



Maschinencode-Darstellung

Symbolische Angaben durch ihre äquivalente binäre Darstellungen ersetzen:

- Zuordnung der symbolischen Operationscodes zu binären Operationscodes liegt durch eine **Zuordnungstabelle** fest:

Symbol	Binärcode
MOVE	0000 0001
SUB	0000 0010
CMP	0010 0001
BNE	0000 1100
JMP	0000 0101

Maschinencode-Darstellung

- Die Ersetzung symbolischer Adressen durch numerische Adressen ergibt sich aus der Lage des Programms im Hauptspeicher
- Numerische Adressen von Registern liegen fest.
- Adresse von EAREG ist durch die Adressdecodierung der EA-Einheit vorgegeben (0x8000)

Voraussetzung:

Programm belegt den Hauptspeicher ab der Zelle 0

➔ Adresszuordnung als Symboltabelle

Maschinencode-Darstellung

Symboltabelle für unser Programm:

```

MOVE EAREG, ZEITK
MOVE NULL, EAREG
MOVE NULL, R0
MOVE ZEITK, R2
SUB EINS, R2
CMP R0, R2
BNE MARKE2
MOVE EINS, EAREG
MOVE NULL, EAREG
JMP MARKE1

ZEITK 0
NULL 1
EINS 1
    
```

Symbol	Adresse			Dezimal	Hex.
	Dual				
MARKE1	0000 0000 0000 1000			8	0008
MARKE2	0000 0000 0001 1010			10	000A
ZEITK	0000 0000 0001 0111			23	0017
NULL	0000 0000 0001 1000			24	0018
EINS	0000 0000 0001 1001			25	0019
EAREG	1000 0000 0000 0000			32768	8000

Maschinencode-Darstellung

Adresse		Maschinencode		Symbol	
Dez.	Dual	Dual	Hex.		
0	00000000	0000000100000000	0100	MOVE	
1	00000001	1000000000000000	8000	EAREG	
2	00000010	0000000000010111	0017	ZEITK	
3	00000011	0000000100000000	0100	MOVE	
4	00000100	0000000000011000	0018	NULL	
5	00000101	1000000000000000	8000	EAREG	
6	00000110	0000000100001000	0108	MOVE	
7	00000111	0000000000011000	0018	NULL	
8	00001000	0000000100001010	010A	MOVE	
9	00001001	0000000000010111	0017	ZEITK	
10	00001010	0000000100001010	020A	SUB	
11	00001011	0000000000011001	0019	EINS	
12	00001100	0010000110001010	218A	CMP	
13	00001101	0000110000000000	0C00	BNE	
14	00001110	0000000000001010	000A	M2	
15	00001111	0000000100000000	0100	MOVE	
16	00010000	0000000000011001	0019	EINS	
17	00010001	1000000000000000	8000	EAREG	
18	00010010	0000000100000000	0100	MOVE	
19	00010011	0000000000011000	0018	NULL	
20	00010100	1000000000000000	8000	EAREG	
21	00010101	0000010100000000	0500	JMP	
22	00010110	0000000000001000	0008	M1	
23	00010111	xxxxxxx00000000			
24	00011000	0000000000000000	0000		
25	00011001	0000000000000001	0001		

Programm

Daten

Symbol	Binärcode
MOVE	0000 0001
SUB	0000 0010
CMP	0010 0001
BNE	0000 1100
JMP	0000 0101

Programm-Übersetzung (Assemblierung)

Übersetzung kann „per Hand“ erfolgen. Dazu benötigt man:

- die Zuordnungstabelle für die OpCodes und
- die Adresse des ersten Befehls im Speicher, um die Symboltabelle mit den Adresszuordnungen aufstellen zu können.

Nachteil: sehr zeitaufwendig und fehleranfällig

Übersetzungsvorgang läuft jedoch nach festen Regeln ab:

➔ **Ausführung durch den Prozessor selbst**



Programm-Übersetzung (Assemblierung)

Assembler:

Programm, das einen Quellcode in Assemblersprache in eindeutiger Weise in Maschinencode übersetzt.

Die Regeln zur symbolischen Programmierung ergeben sich aus der Definition einer Assemblersprache

Format einer Programmzeile:

Namensfeld (label)	Operationsfeld (OpCode)	Adreßfeld (Operanden)	Kommentarfeld
<i>symbolische Adressierung einer Programmzeile</i>	<i>symbolischer OpCode</i>	<i>symbolische Adreßangaben</i>	



Assemblerdirektiven (Pseudobefehle)

Pseudobefehle sind Befehle für den Assembler

- Erzeugung von Konstanten
- Wertzuweisung an Operanden
- Reservierung von Speicherplatz
- Steuerung des Assemblierungsvorgangs
- erzeugen bei der Übersetzung nicht immer Binärkode (im Gegensatz zu Maschinenbefehlen)
- sie unterliegen dem Format einer Programmzeile



Assemblerdirektiven (Pseudobefehle)

[symbol]	ORG	c	origin of program or data
[symbol]	DS	c	define storage
[symbol]	DC	c	define constant
symbol	EQU	c	equate
	END		end of program

Im Beispiel:

```
                ORG    0
EAREG    EQU    32678
NULL     DC     0
EINS     DC     1
ZEITK    DS     1
```



Impulsgeber-Programm

Name	Opcode	Adreßangaben	Kommentar
* * Impulsgeberprogramm			
EAREG	ORG	0	Speicherbelegung ab Adresse 0
	EQU	32768	Ein-/Ausgabeadresse festlegen
	MOVE	EAREG,ZEITK	Zeitkonstante einlesen
	MOVE	NULL,EAREG	
M1	MOVE	NULL,R0	
	MOVE	ZEITK,R2	Zeitschleife initialisieren
	SUB	EINS,R2	
	CMP	R0,R2	Zeitbedingung abfragen
M2	BNE	M2	
	MOVE	EINS,EAREG	positive Flanke
	MOVE	NULL,EAREG	negative Flanke
	JMP	M1	
* Beginn des Datenbereichs			
ZEITK	DS	1	
NULL	DC	0	
EINS	DC	1	
	END		



Assemblierung

Zwei Phasen:

1. Phase:

- Adresszuordnung herstellen
- Fehlerliste anfertigen

2. Phase:

- Maschinencode erzeugen
- Symbolische und binäre Auflistung

Beim Erreichen der END-Anweisung müssen alle Symbole definiert sein!



Impulsgeber- Programmliste

Nr.	Adresse	Inhalt	Name	Opcode	Adreßangaben	Kommentar
1			*			
2			* Impulsgeberprogramm			
3						
4				ORG	0	speichern ab Adr. 0
5			EAREG	EQU	32768	E/A-Adr. festlegen
6	0000	0100		MOVE	EAREG,ZEITK	Zeitkonst. einlesen
		8000				
		0017				
7	0003	0100		MOVE	NULL,EAREG	
		0018				
		8000				
8	0006	0108		MOVE	NULL,R0	
		0018				
9	0008	010A	M1	MOVE	ZEITK,R2	Zeitschleife init.
		0017				
10	000A	020A	M2	SUB	EINS,R2	
		0019				
11	000C	218A		CMP	R0,R2	Zeitbed. abfragen
12	000D	0C00		BNE	M2	
		000A				
13	000F	0100		MOVE	EINS,EAREG	positive Flanke
		0019				
		8000				
14	0012	0100		MOVE	NULL,EAREG	negative Flanke
		0018				
		8000				
15	0015	0500		JMP	M1	
		0008				
16			*			
17	0017		ZEITK	DS	1	Datenbereichsanfang
18	0018	0000	NULL	DC	0	
19	0019	0001	EINS	DC	1	
20				END		

