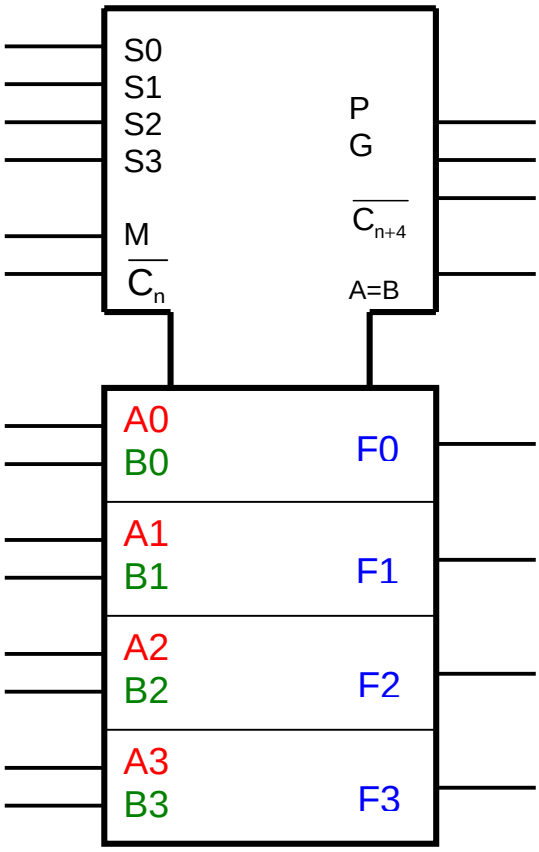
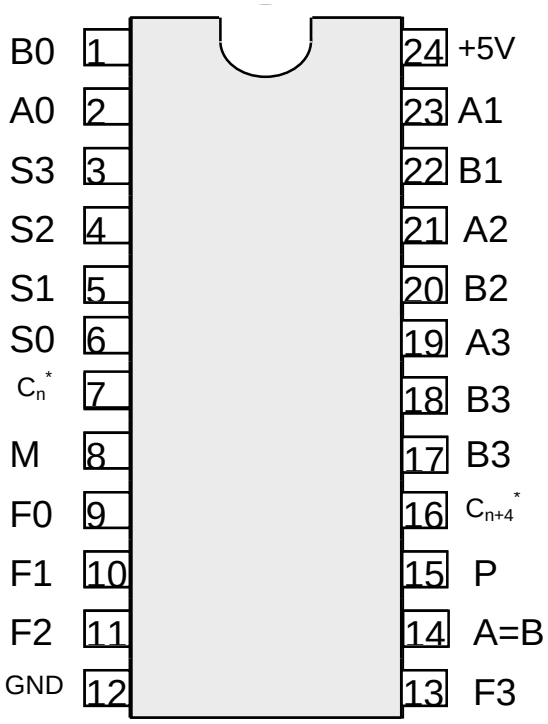
Die Arithmetisch-Logische Einheit ALU 74181

Blockschaltbild:



IC 74181 Pin-Ansicht



Die ALU hat

- 6 Steuerleitungen

  S0, S1, S2, S3, M, $\overline{C_n}$

Die ALU ist eine 4-Bit-ALU d.h. sie hat

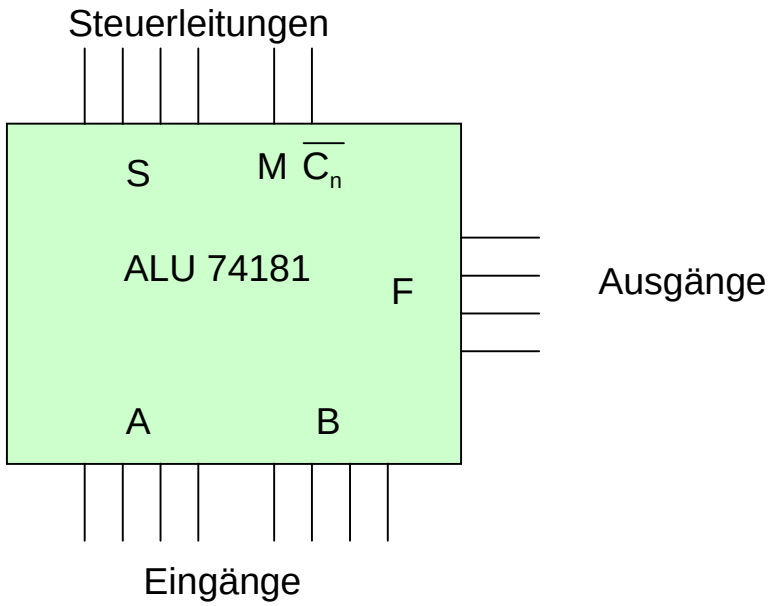
- 4 Eingangsleitungen $A_0;A_1;A_2;A_3$ für den A-Operanden
- 4 Eingangsleitungen $B_0;B_1;B_2;B_3$ für den B-Operanden

- 4 Ausgangsleitungen für das Ergebnis der Operation
- 1 Ausgangsleitung $\overline{C_{n+4}}$ für einen eventuellen Übertrag
- 1 Vergleichsausgang A=B (nur in Verbindung mit der Operation A-B !)
- Zwei Ausgänge P und G für die Übertragsauswertung

Die Ausgangsleitung A=B zeigt eine 1 , wenn die Steuerleitungen so eingestellt sind, dass die Operation „SUBTRAKTION“ eingestellt ist und die beiden Operanden identisch sind.

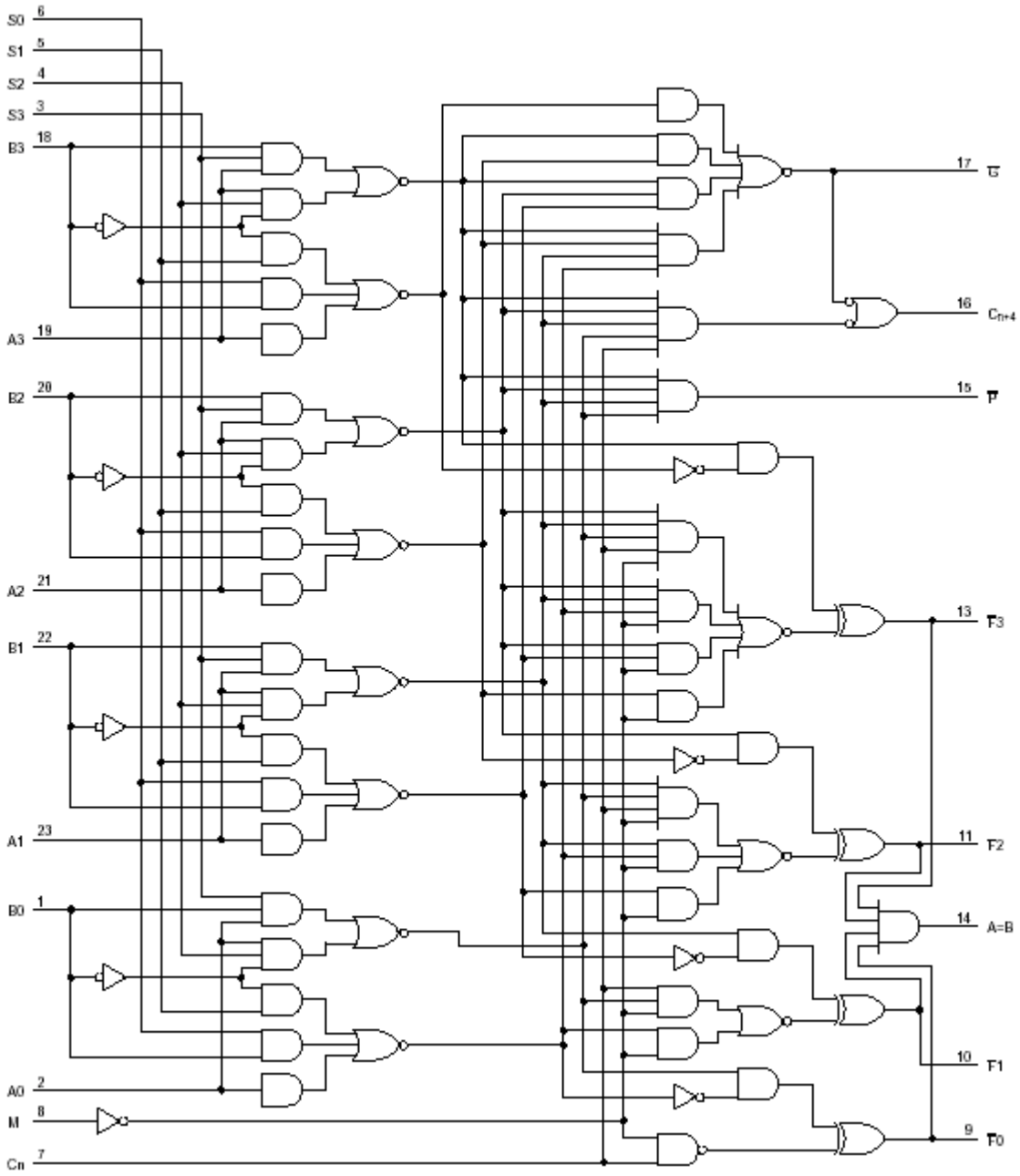
Die beiden Ausgangsleitungen P und G sind für einen speziellen Übertragsauswertungsbaustein IC 74182 bestimmt, der den korrekten Übertrag ermittelt, wenn man 4 solcher 4-Bit-ALUs zu einer 16-Bit-ALU zusammenfügt.

Eine einfachere Darstellung des Blockschaltbilds:



Die Innenschaltung der ALU 74181

(Quelle : Philips Semiconductors 74181 Produkt Specification – Datenblatt)



Alte Darstellung der Digitalbausteine:						
						
AND	NAND	OR	NOR	XOR	Inverter	

Funktionen der ALU 74181

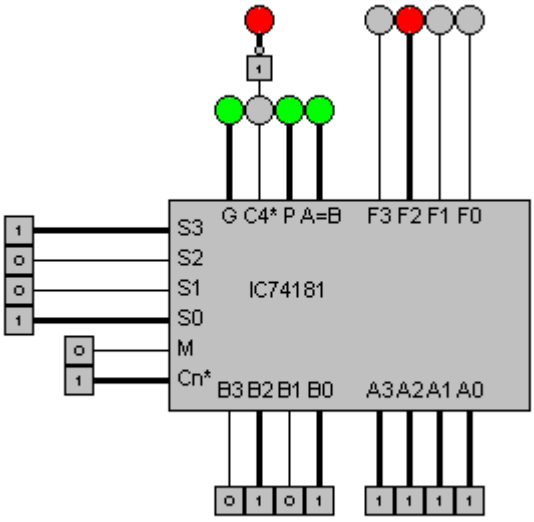
Steuerleitungen				M=1	M=0	
				logische	arithmetische Operationen	
S0	S1	S2	S3	Operationen	$\overline{C_n}=0$	$\overline{C_n}=1$
0	0	0	0	$F=\overline{A}$	F=A+1	F=A
0	0	0	1	$F=\overline{A \vee B}$	$F=(A \vee B)+1$	$F=(A \vee B)$
0	0	1	0	$F=\overline{A} \wedge B$	$F=(A \vee \overline{B})+1$	$F=(A \vee \overline{B})$
0	0	1	1	F=0	F=0	$F=-1$
0	1	0	0	$F=\overline{A \wedge B}$	$F=A+(A \wedge \overline{B})+1$	$F=A+(A \wedge \overline{B})$
0	1	0	1	$F=\overline{B}$	$F=(A \vee B)+(A \wedge \overline{B})+1$	$F=(A \vee B)+(A \wedge \overline{B})$
0	1	1	0	$F=A \dot{\vee} B$	$F=A-B$	$F=A-B-1$
0	1	1	1	$F=A \wedge \overline{B}$	$F=A \wedge \overline{B}$	$F=(A \wedge \overline{B})-1$
1	0	0	0	$F=\overline{A} \vee B$	$F=A+(A \wedge B)+1$	$F=A+(A \wedge B)$
1	0	0	1	$F=\overline{A \dot{\vee} B}$	$F=A+B+1$	$F=A+B$
1	0	1	0	F=B	$F=(A \vee \overline{B})+(A \wedge B)+1$	$F=(A \vee \overline{B})+(A \wedge B)$
1	0	1	1	$F=A \wedge B$	$F=A \wedge B$	$F=A \wedge B-1$
1	1	0	0	F=1	$F=A+A+1$	$F=A+A$
1	1	0	1	$F=A \vee \overline{B}$	$F=(A \vee B)+A+1$	$F=(A \vee B)+A$
1	1	1	0	$F=A \vee B$	$F=(A \vee \overline{B})+A+1$	$F=(A \vee \overline{B})+A$
1	1	1	1	F=A	$F=A$	$F=A-1$

Beachte, dass A,B und F 4-Bit-Worte sind!
Beispiel 1: A=1010
B=1100
Dann ist z.B.:
AvB= 1110 $F=(A \vee B)+A$ =1000 A+A=0100

Beispiel 2: A=1111
B=0110

Dann ist z.B.:

$F = A + B = 0100$ $\quad \overline{C_{n+4}} = 0$ d.h. $C_{n+4} = 1$



Testschaltung mit LOCAD : IC74181_test.lws

Aufgabe 1 : Die Steuerleitungen werden so eingestellt, dass
S=1001 , M=0 und $\overline{C_n}$ =1 ist. D.h.: Die Operation ist F=A+B
Für den A-Operand (1. Summand) wählen wir A=1011 und für den B-Operand (2. Summand) B=0110

Färbe alle auf Datenleitungen die ein 1-Signal haben auf der Kopie des Datenblatts rot ein und alle Datenleitungen die eine 0 tragen grün ein.
Zeige damit, dass die ALU das Ergebnis korrekt ausgibt.

	1	0	1	1		11
+	0	1	1	0	+	6
	1	1				
1	0	0	0	1		17

	1	0	1	1		-5
+	0	1	1	0	+	6
	1	1				
1	0	0	0	1		1

Aufgabe 2 : Die Steuerleitungen werden so eingestellt, dass
S=1000 , M=1 und $\overline{C_n}$ =0 ist. D.h.: Die Operation ist $F = \overline{A} \vee B$
Für den A-Operand wählen wir A=1010 und für den B-Operand B=0110

Färbe alle Datenleitungen die ein 1-Signal haben auf der Kopie des Datenblatts rot ein und alle Datenleitungen die eine 0 tragen grün ein.
Zeige damit, dass die ALU das Ergebnis korrekt ausgibt.

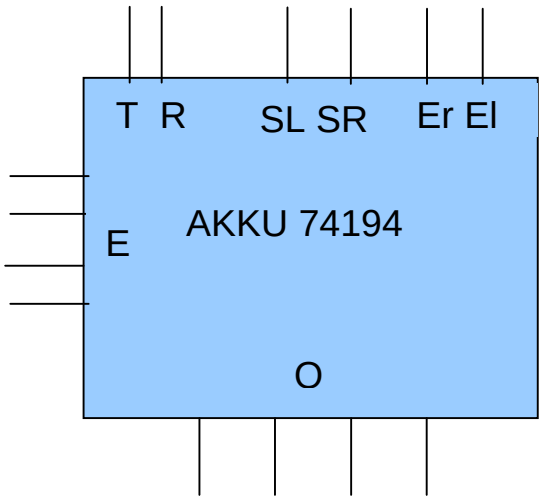
A	1	0	1	0
$\overline{A}$	0	1	0	1
B	0	1	1	0
$\overline{A} \vee B$	0	1	1	1

Aufgabe 3 : Fülle folgende Tabelle aus:
und teste mit LOCAD, ob das Ergebnis korrekt ist.

A		1	0	1	0
$\overline{A}$		0	1	0	1
B		0	1	1	0
$\overline{B}$		1	0	0	1
$F = \overline{A} \wedge B$					
$F = \overline{A \wedge B}$					
$F = \overline{A \dot{\vee} B}$					
$F = A + A$					
$F = A - B - 1$					
$F = A + (A \wedge \overline{B})$					
$F = (A \vee \overline{B}) + (A \wedge B)$					

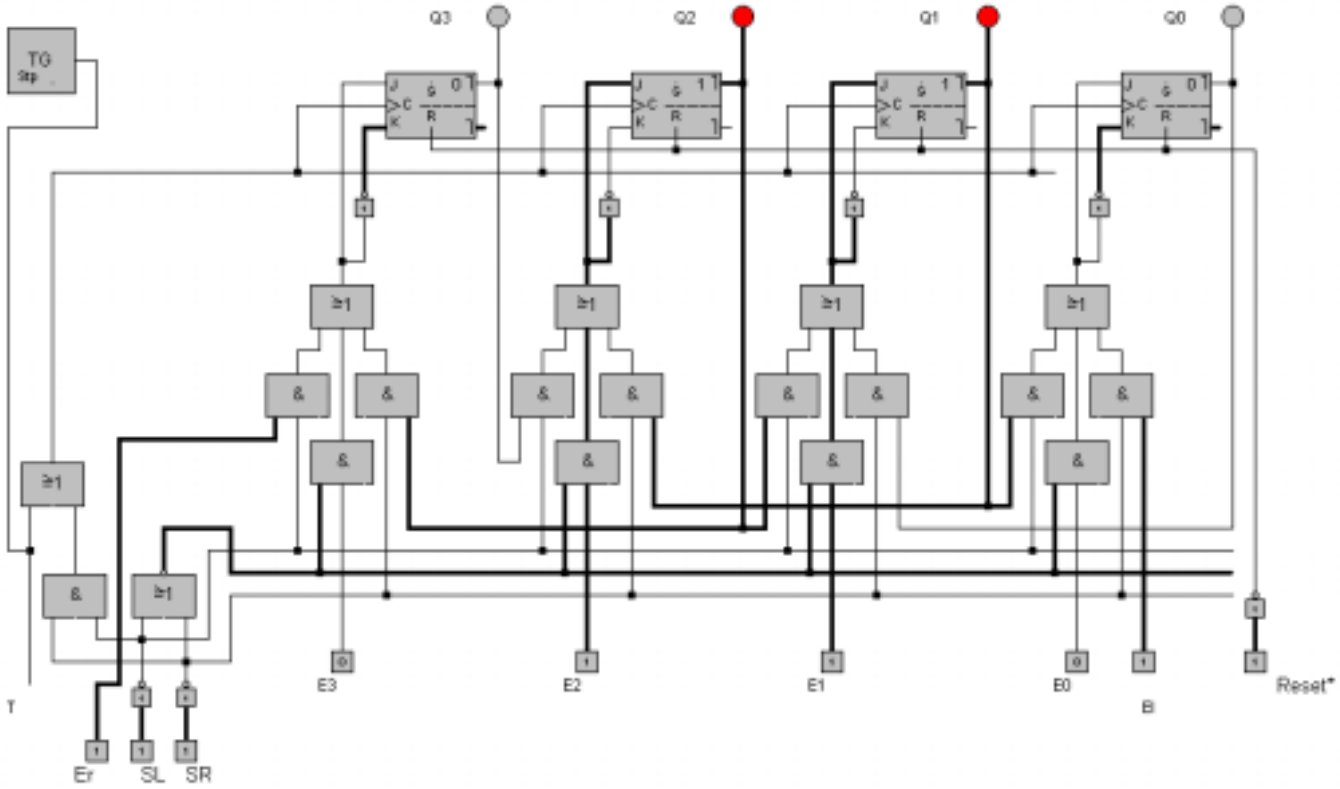
Um die Daten, die die ALU ausgibt speichern und verarbeiten zu können, verwendet man häufig ein

4-Bit- Schieberegister mit paralleler Übernahme und Reset



Bei einem Taktimpuls wird das 4-Bit-Eingangswort E in den Akku übernommen und am Ausgang Q angelegt, falls SL=1 und SR=1 ist. Für SL=1 und SR=0 arbeitet das Register als Linksschieberegister und für SL=0 und SR=1 als Rechtsschieberegister. Für SL=0 und SR=0 ergibt sich keine Änderung.

Das IC 74194



Testschaltung mit LOCAD : IC74194.LWS

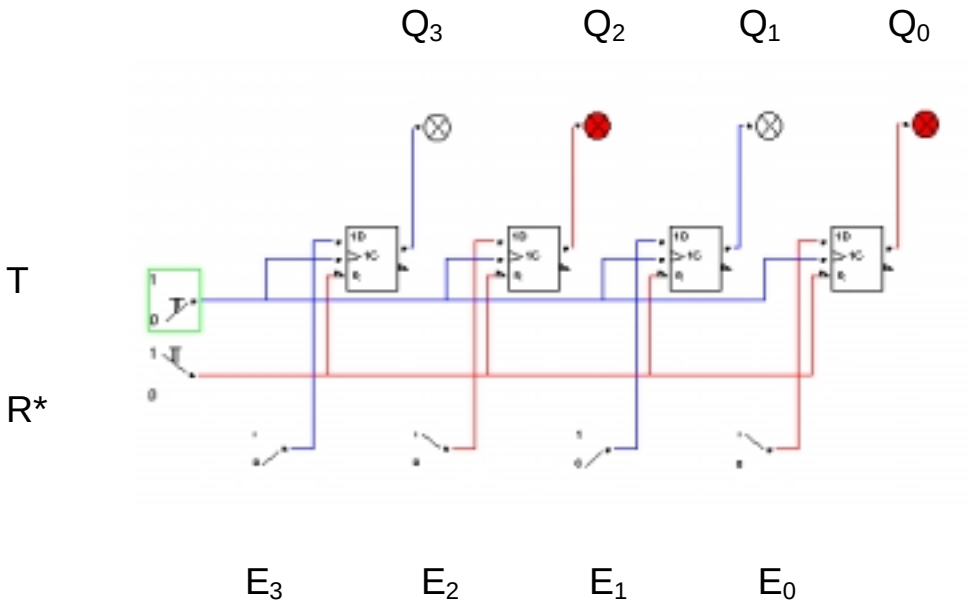
Die Eingänge Er und El legen fest wie das Register beim Rechts- bzw. Linksschieben aufgefüllt wird.

Das Eingangswort E=0110 ist hier bei einer negativen Flanke des Taktimpulses in das Register übernommen worden und steht am Ausgang Q=0110 an.

Der Reset-Eingang ist invertiert, d.h. ein Reset wird bei Reset*=0 ausgeführt.

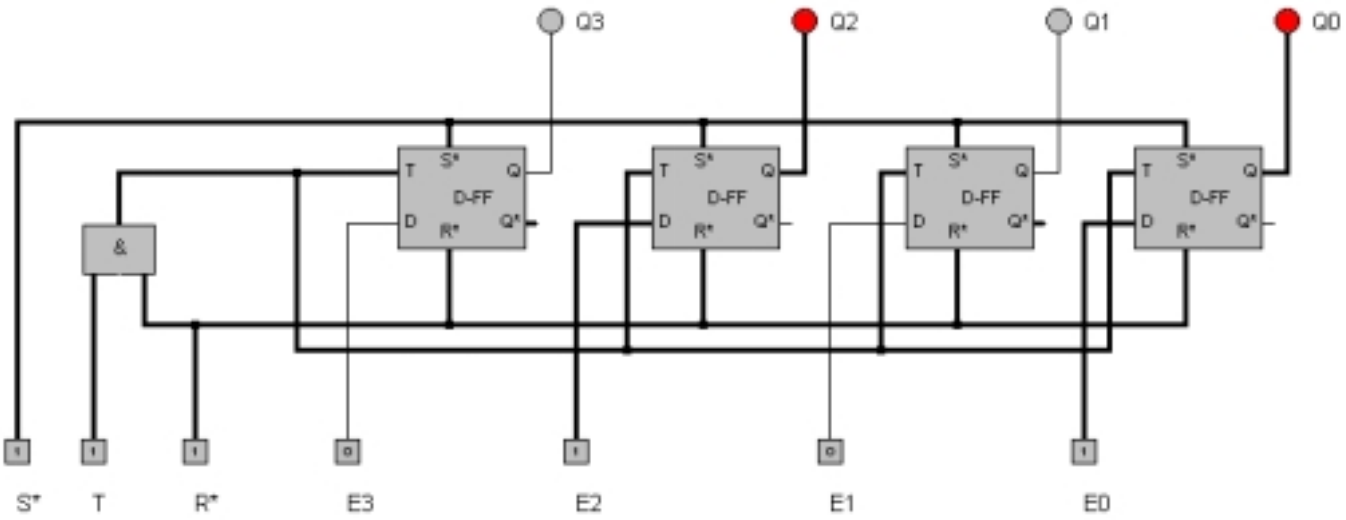
Ein einfacheres 4-Bit-Register (Akku) lässt sich mit vier D-Flip-Flops realisieren:

Das IC 74175 Quad D-type-FF with reset positive-edge trigger



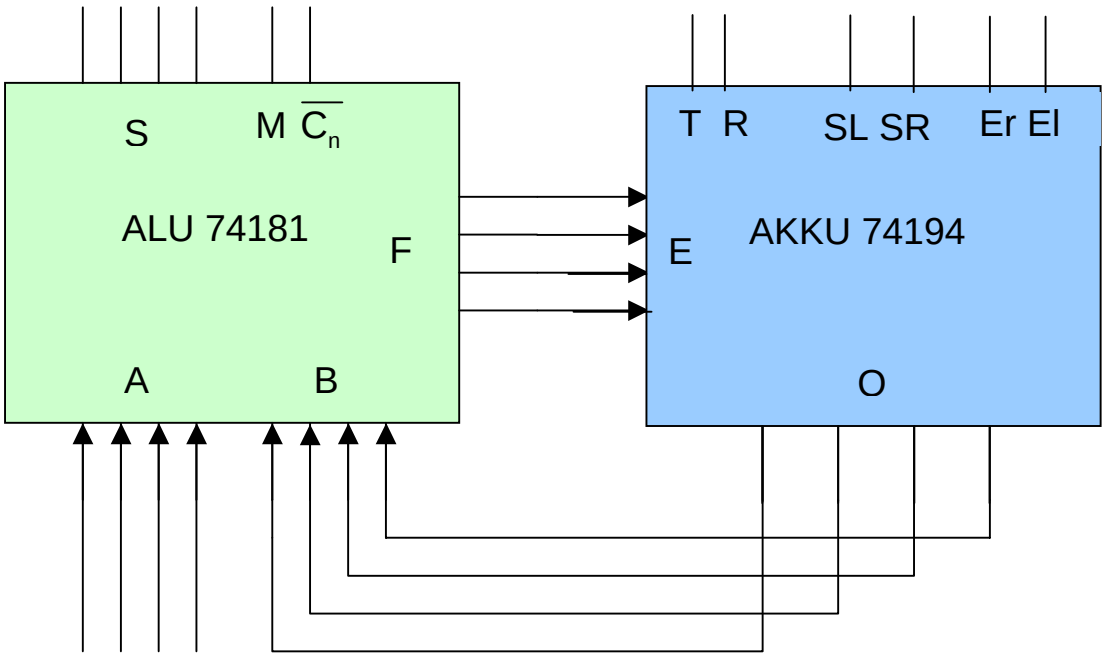
Testschaltung mit DigiSim : IC74175.dsim

Bei einer positiven Flanke am Takteingang T wird das 4-Bit-Datenwort in das Register übernommen und an den Q-Ausgängen zur Verfügung gestellt.



Testschaltung mit LOCAD : IC74175.lws

Mit diesen Bausteinen kann man nun eine programmierbare Recheneinheit aufbauen:
Diese Schaltung ist als Modell fest aufgebaut :



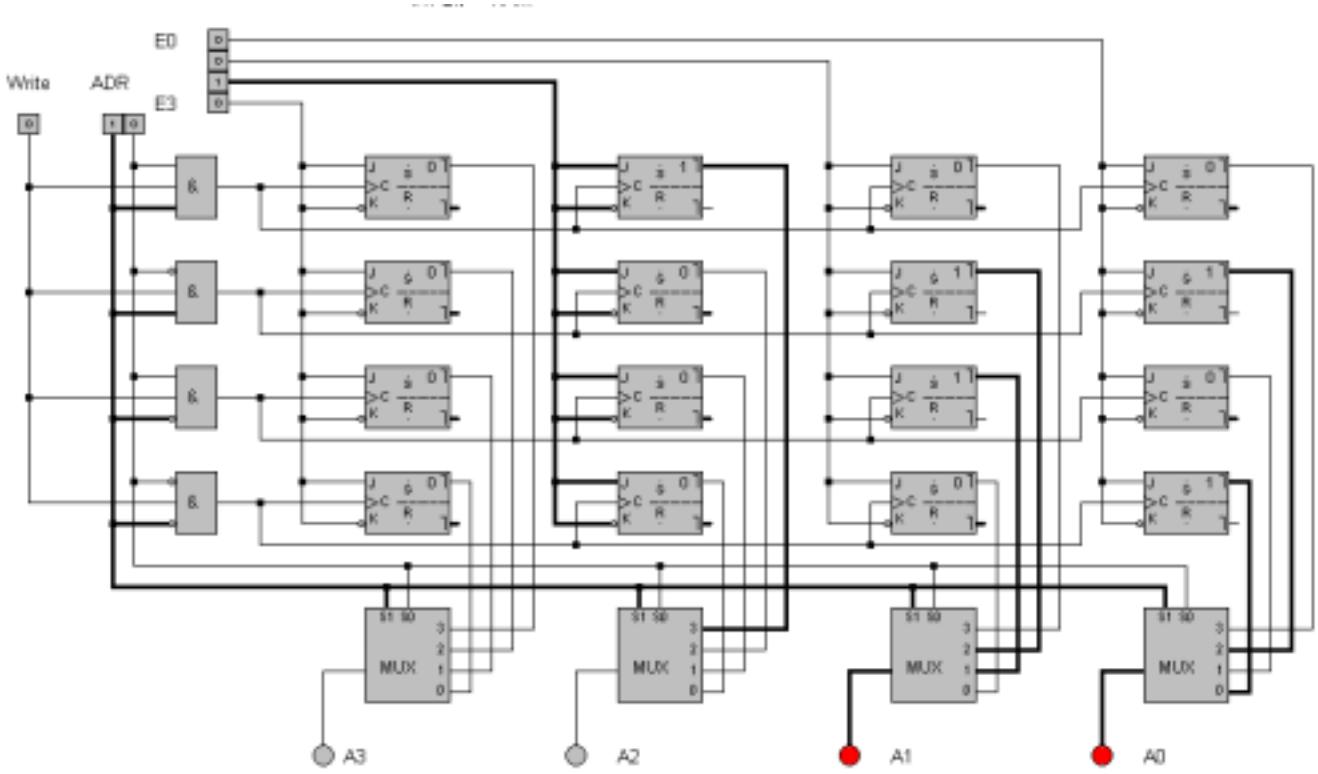
Programm zur Berechnung von (8+5) DIV 2										
	S3	S2	S1	S0	M	$\overline{C_n}$	SL	SR	Er	Wirkung
A=1000	0	0	0	0	0	1	1	1	0	F=A Übernahme 1000
Taktimpuls										
A=0101	1	0	0	1	0	1	1	1	0	F=A+B Übernahme 1101
Taktimpuls										
A=xxxx	x	x	x	x	x	x	0	1	0	Shift Right 0110
Taktimpuls										
A=0000	1	0	0	1	0	1	1	1	0	F=A+B Übernahme 0110
Taktimpuls										
										(8+5) DIV 2=6

Ein Programm besteht dabei immer aus den gleichen Sequenzen :

1.) Steuerleitungen einstellen, A-Operand anlegen, Taktimpuls geben
2.) Steuerleitungen einstellen, A-Operand anlegen, Taktimpuls geben
3.) Steuerleitungen einstellen, A-Operand anlegen, Taktimpuls geben
.....
u.s.w.

Wenn dieser Vorgang automatisiert ablaufen soll, dann müssen wir sowohl das Programm als auch die Daten abspeichern können. Dazu benötigen wir Speicherbausteine!

RAM 4 X 4-BIT aufgebaut mit FlipFlops

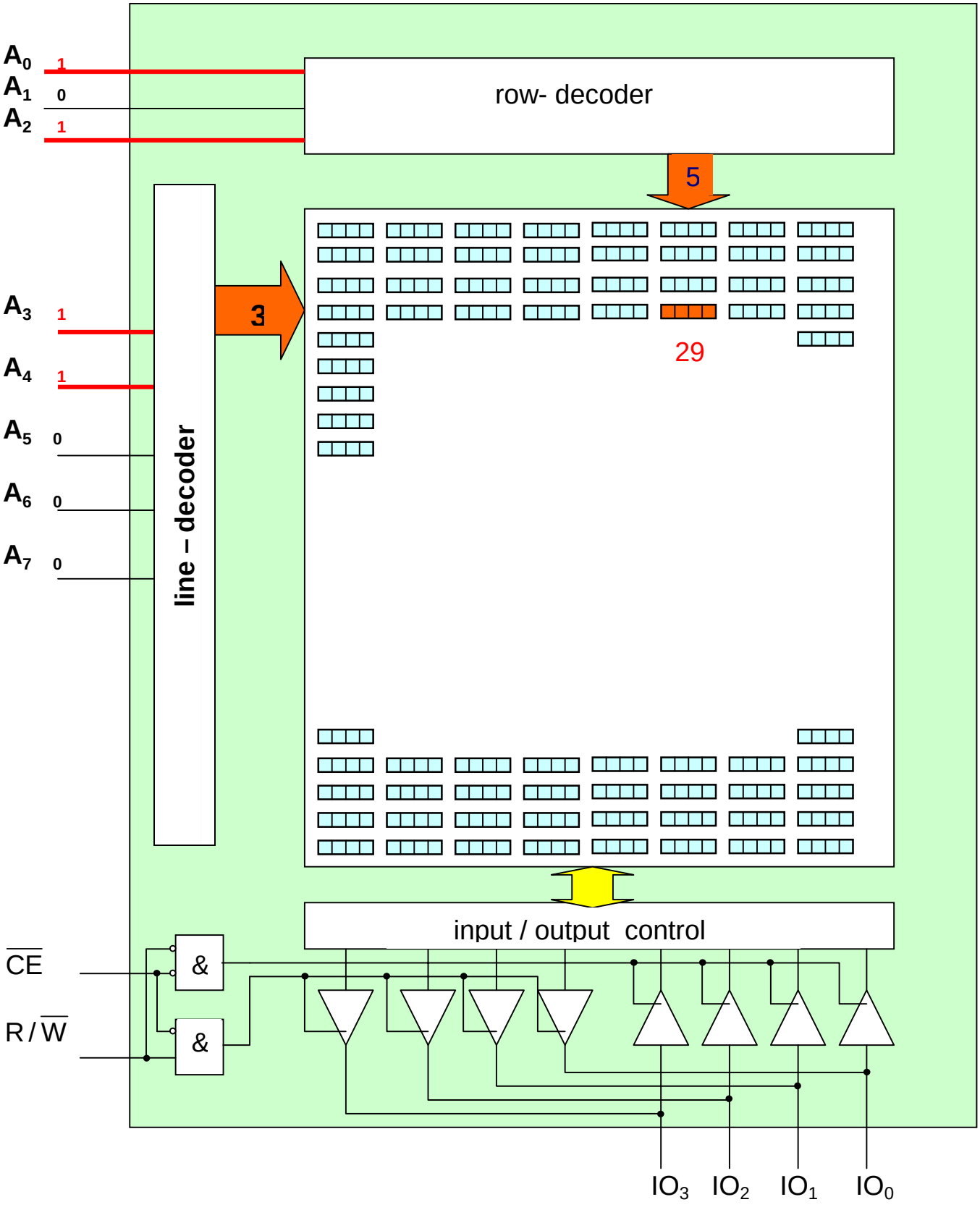


Testschaltung mit LOCAD : RAM4X4.LWS

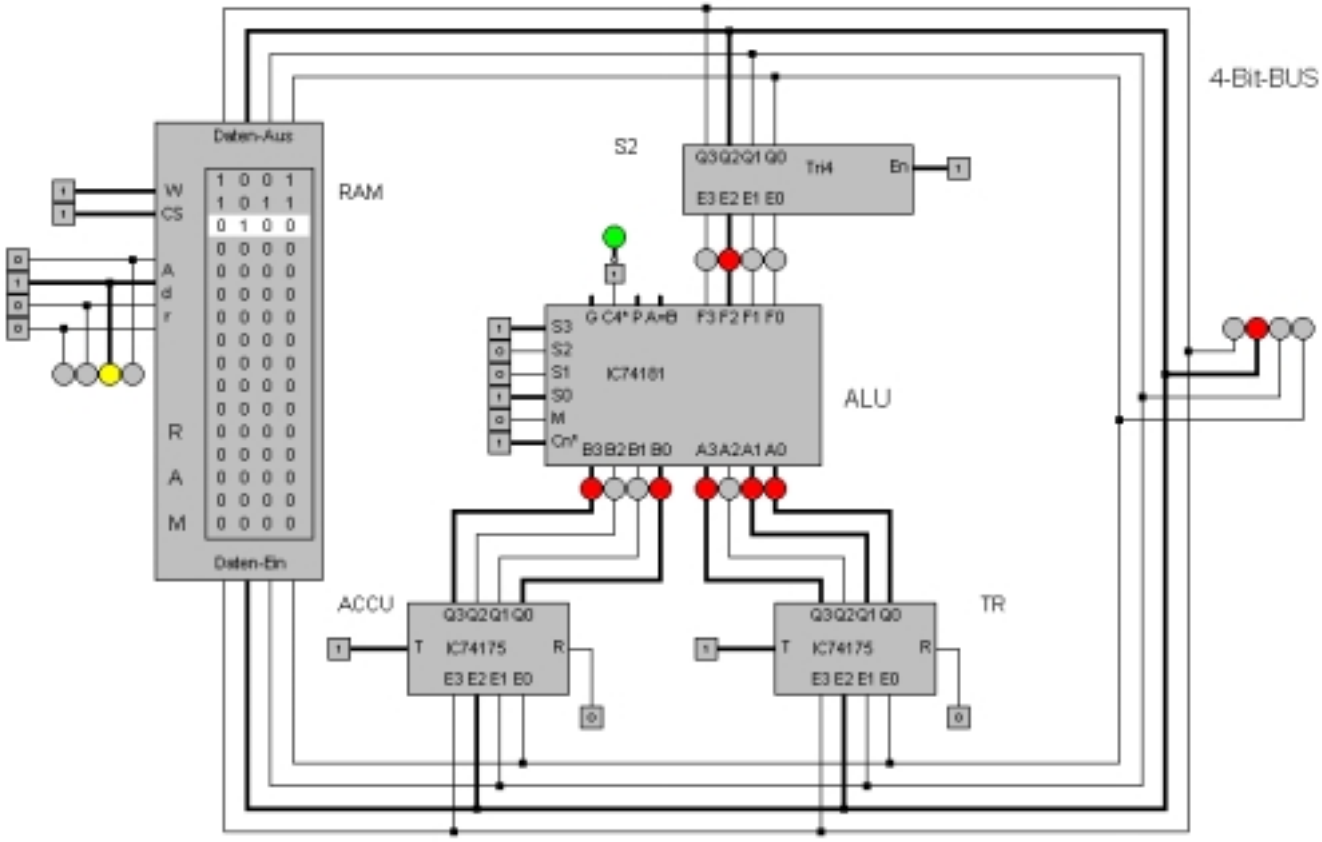
An den Adressleitungen werden die Adressen der Zeilen (0...4 von unten nach oben) eingegeben.
In der obigen Abbildung ist die Zeile 2 (Adresse 10) adressiert und der Inhalt dieser Zeile liegt am Ausgang A an.
Soll in eine Zeile geschrieben werden, so wird das einzuschreibende Datenwort an den Eingängen E0...E3 angelegt und danach am Write-Eingang eine negative Flanke erzeugt. Dann wird das Datenwort in die adressierte Zeile geschrieben und steht sofort am Ausgang zur Verfügung.

Aufgabe:
Entwickle die Innenschaltung eines 4 auf 1 Multiplexers(4-fach-Weiche), der diejenige Eingangsleitung auf den Ausgang durchschaltet, der durch die beiden Adressleitungen adressiert ist.

IC 2101 RAM 1024 Bit (32x8x4)



Aufbau einer CPU (1. Stufe)



ACCU TR

Die Daten, die verarbeitet werden sollen, werden in einem RAM 16x4 Bit gespeichert.

W : Steuerleitung write
Ein auf dem Bus liegendes Datenwort wird in die Adressierte Speicherzelle übernommen, wenn an W eine negative Flanke erscheint und CS=1 ist.

CS : cable select Die Datenausgänge sind über einem Tri-state-Baustein mit dem BUS verbunden: CS=0 „entkoppelt“ CS=1 „gekoppelt“

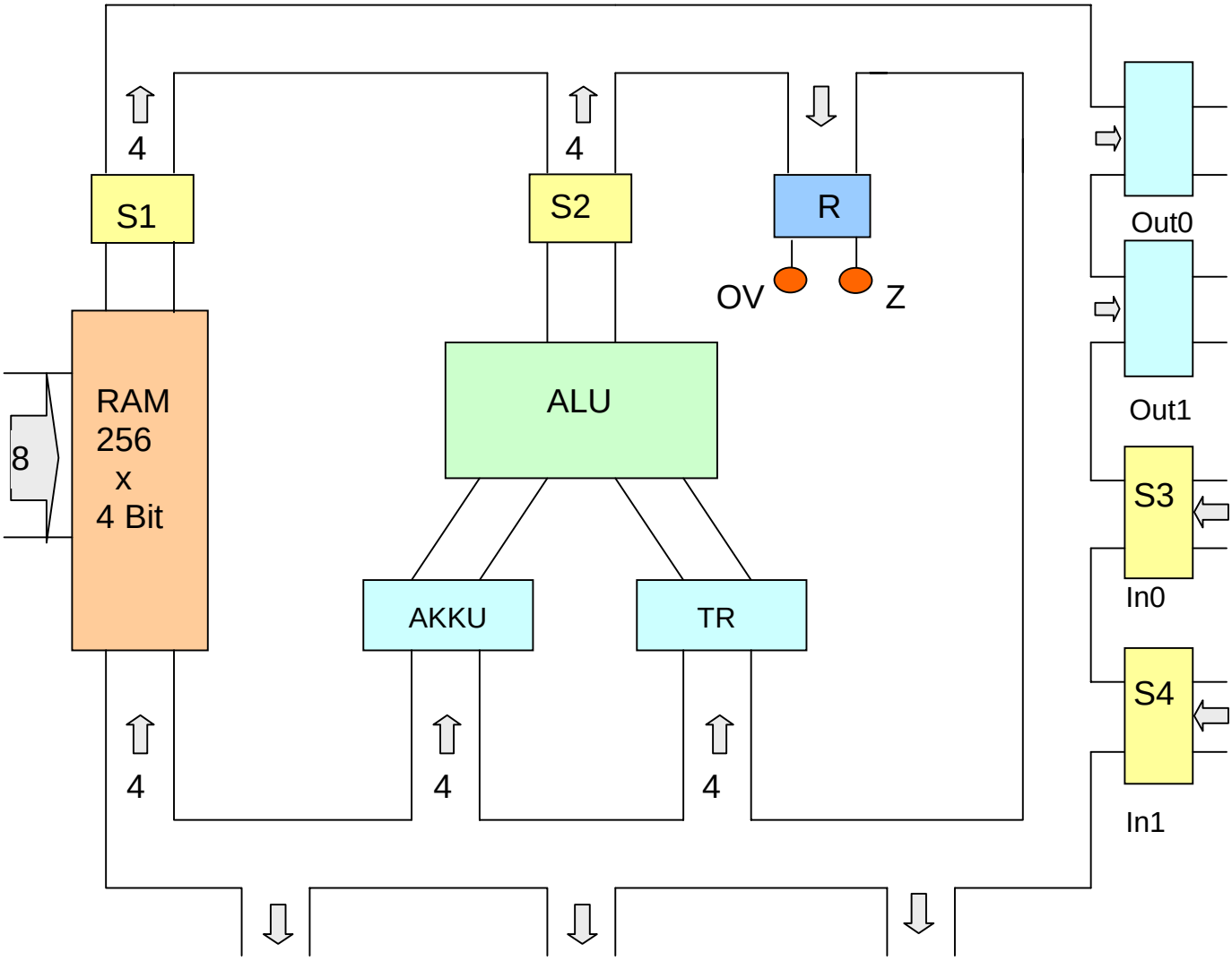
Über den 4-Bit-Adressbus können die 16 Speicherzellen mit den Adressen 0000.....1111 adressiert werden.

Mit dem Schalter S2 kann der F-Ausgang auf den BUS gelegt werden (En=1) bzw. vom BUS entkoppelt werden (En=0)

Die Steuerleitungen müssen in diesem Entwicklungsstadium noch von Hand mit 0 bzw. 1 belegt werden, um die gewünschte Operation zu erhalten.

Aufgabe: Ein Programm soll den Inhalt der Speicherzelle 11 und den Inhalt der Speicherzelle 12 addieren und das Ergebnis in die Speicherzelle 13 einschreiben. Beschreibe den Ablauf dieses Programms detailliert !

Blockschaltbild ohne Steuerleitungen:



AKKU, TR, Out0, Out1 :	IC74175	4-Bit-D-Register mit Reset
S1, S2, S3, S4 :	IC74125	4-Bit-Tri-State-Leistungstreiber(Schalter)
RAM :	IC 2101	256x4-Bit (8-Bit-Adressbus)
ALU:	IC74181	4-Bit-ALU
R :	IC7474	2-Bit-D-Register mit Reset

Bisher haben wir in unserem Modellcomputer nur Daten im Hautspeicher abgelegt und mussten die Einstellung der Steuerleitungen und den logischen Ablauf selbst regeln.

Wenn wir diese Aufgabe ebenfalls dem Computer übertragen wollen, dann müssen neben den Daten auch die sogenannten Maschinenbefehle im Hauptspeicher abgelegt werden können. Da der Hauptspeicher aus 4-Bit-Speicherzellen besteht, können wir in einer Speicherzelle 16 Operationen für unseren Modellcomputer verschlüsseln:

OP-Code				Mnemonic	Bedeutung)
0	0	0	0	JMP	Jump
0	0	0	1	JNZ	Jump if not zero
0	0	1	0	CALL	Call
0	0	1	1	RET	Return
0	1	0	0	LDA	Load accumulator
0	1	0	1	STA	Store accumulator
0	1	1	0	ADDA	Add accumulator
0	1	1	1	DECM	Decrement memory
1	0	0	0	OUT0	Output 0
1	0	0	1	OUT1	Output 1
1	0	1	0	INA	Input accumulator
1	0	1	1	JBY	Jump if busy
1	1	0	0	ENTA	Enter accumulator
1	1	0	1	INCA	Increment accumulator
1	1	1	0	NANDA	Nand accumulator
1	1	1	1	WAIT	Wait

Mnemonic : Gedächtnisstütze (Griech.: μνημονικο mnimoniko)

Diese 16 Maschinenbefehle teilen sich in 3 Gruppen auf :

Die Befehle JMP,JNZ,CALL,LDA,STA,ADDA,DECM und JBY sind sogenannte 3-Wort-Befehle ,die als Bezug immer die Adresse einer Speicherzelle haben, mit deren Inhalt die entsprechende Operation ausgeführt wird:

Beispiel LDA 135

0	1	0	0	LDA	opcode
0	1	1	1	low adress	adresscode
1	0	0	0	high adress	

Syntax: Lade den Akku mit dem Inhalt der Speicherzelle 135

Die Befehle OUT0,ENTA,INCA und NANDA sind sogenannte 2-Wort-Befehle die als Bezug immer ein 4-Bit-Datenwort haben, mit dem die entsprechende Operation ausgeführt wird:

Beispiel INCA 5

0	1	0	0	INCA	opcode
0	1	1	1	5	wordcode

Syntax: Incrementiere den Akkuinhalt um 5 <acc>:=<acc>+n

Die Befehle RET,OUT1,INA und WAIT sind sogenannte 1-Wort-Befehle ,die nur eine Operation ohne Bezug ausführen:
Beispiel OUT1

0	1	0	0	OUT1	opcode

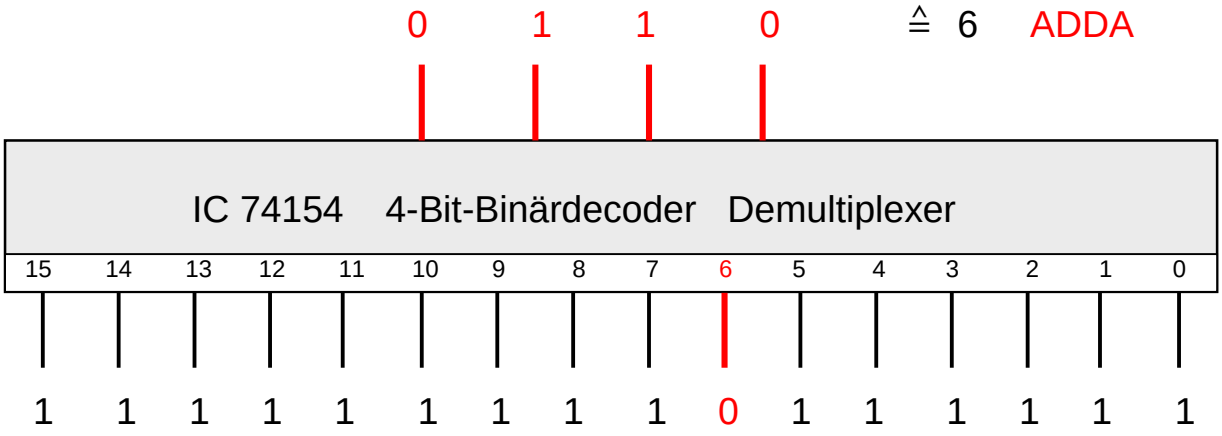
Syntax: Gib den Akkuinhalt am Ausgang OUT1 aus <OUT1>:=<acc>

Damit ergibt sich die vollständige Befehlstabelle:

OP-Code				Mnemonic	Bedeutung
0	0	0	0	JMP adr	Jump: Unbedinter Sprung auf die Adresse adr
0	0	0	1	JNZ adr	Jump if not zero Sprung auf die Adresse adr, falls der Akkuinhalt <>0 ist
0	0	1	0	CALL adr	Call Unterprogrammsprung auf die Adresse adr
0	0	1	1	RET	Return Unterprogrammrücksprung
0	1	0	0	LDA adr	Load accumulator <acc>:=<adr>
0	1	0	1	STA adr	Store accumulator <adr>:=<acc>
0	1	1	0	ADDA adr	Add accumulator <acc>:=<acc>+<adr>
0	1	1	1	DECM adr	Decrement memory <adr>:=<adr>-1
1	0	0	0	OUT0 n	Output 0 <OUT0>:=n
1	0	0	1	OUT1	Output 1 <OUT1>:=<acc>
1	0	1	0	INA	Input accumulator <acc>:=<IN0/IN1>
1	0	1	1	JBY adr	Jump if busy Sprung auf Adresse adr, falls Signal an BY-Buchse = 0
1	1	0	0	ENTA n	Enter accumulator <acc>:=n
1	1	0	1	INCA n	Increment accumulator <acc>:=<acc>+n
1	1	1	0	NANDA n	Nand accumulator $< \text{acc} >:= \overline{< \text{acc} > \wedge n}$
1	1	1	1	WAIT	Wait Stop des Recners für eine einstellbare Zeitdauer

Wie wird der Operationscode unserer Maschinenbefehle vom Computer entschlüsselt ?

Ein neuer Befehlscode , etwa 0110 für ADDA gelangt zuerst in das sogenannte Instruktionsregister IR, das den gleichen Aufbau wie der Akku und das TR besitzt. Von dort gelangt dieser Code zu einem 4-Bit-Binärdekoder (Demultiplexer) . Das ist ein Baustein, der die Aufgabe hat genau eine von 16 Ausgangsleitungen auf 0 zu legen, nämlich gerade die, die am Eingang adressiert ist.



Dieser Binärdecoder dient nun dazu mit Hilfe einer Diodenmatrix die Steuerleitungen unseres Rechners so zu stellen, dass der Befehl korrekt ausgeführt werden kann.

Aufgabe:

Entwickle die Innenschaltung eines 2-Bit-Binärdecoders (Demultiplexers).

IC 74154

E_3	E_2	E_1	E_0		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	
0	0	0	1		1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	
0	0	1	0		1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	
0	0	1	1		1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	
0	1	0	0		1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	
0	1	0	1		1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	
0	1	1	0		1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	
0	1	1	1		1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	
1	0	0	0		1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	
1	0	0	1		1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	
1	0	1	0		1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	
1	0	1	1		1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	
1	1	0	0		1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	
1	1	0	1		1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	1	1	0		1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	1	1	1		0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	

Konjunktive Normalform:

0 = $E_3 \vee E_2 \vee E_1 \vee E_0 = \overline{\overline{E_3} \wedge \overline{E_2} \wedge \overline{E_1} \wedge \overline{E_0}}$

1 = $E_3 \vee E_2 \vee E_1 \vee \overline{E_0} = \overline{\overline{E_3} \wedge \overline{E_2} \wedge \overline{E_1} \wedge E_0}$

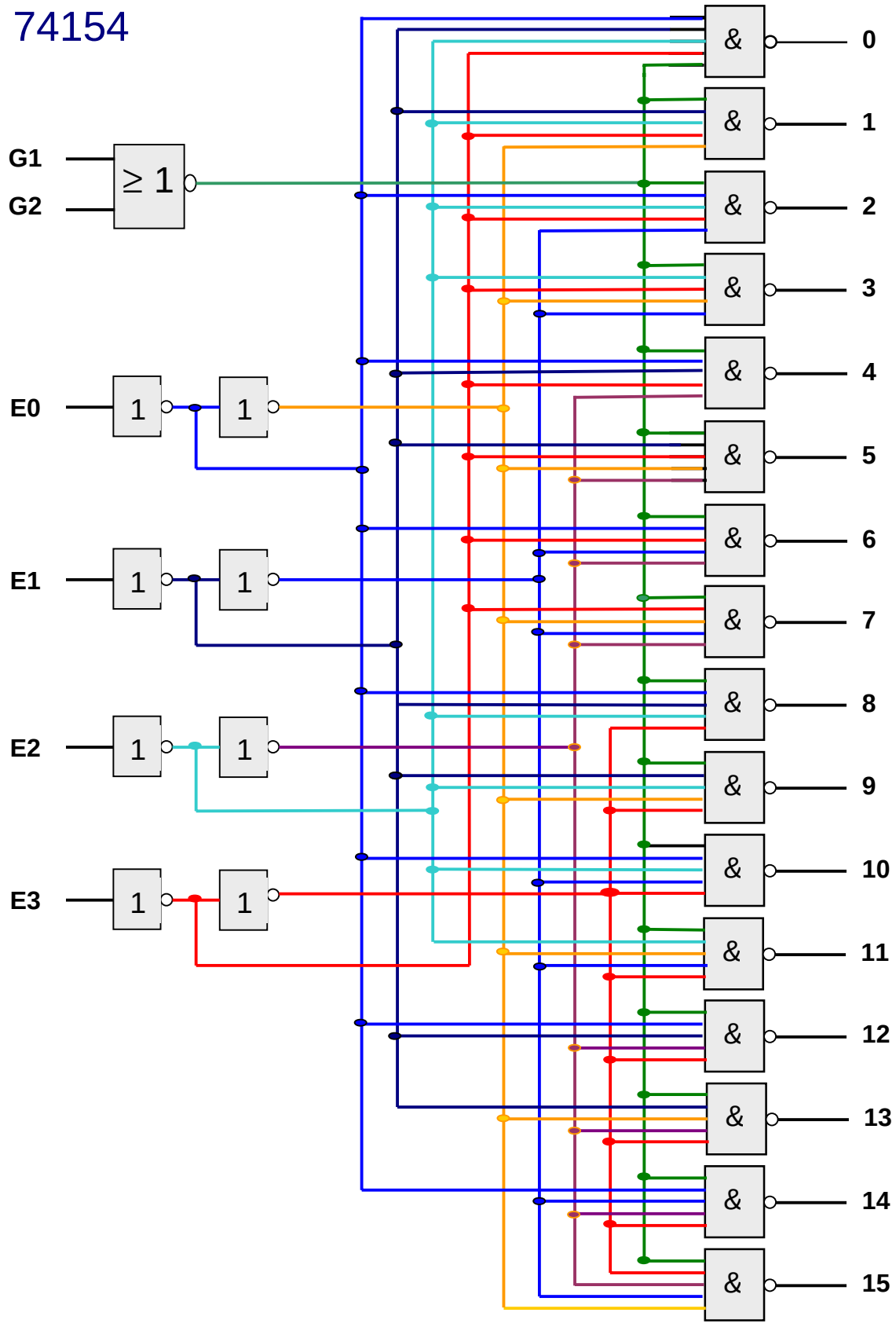
.

.

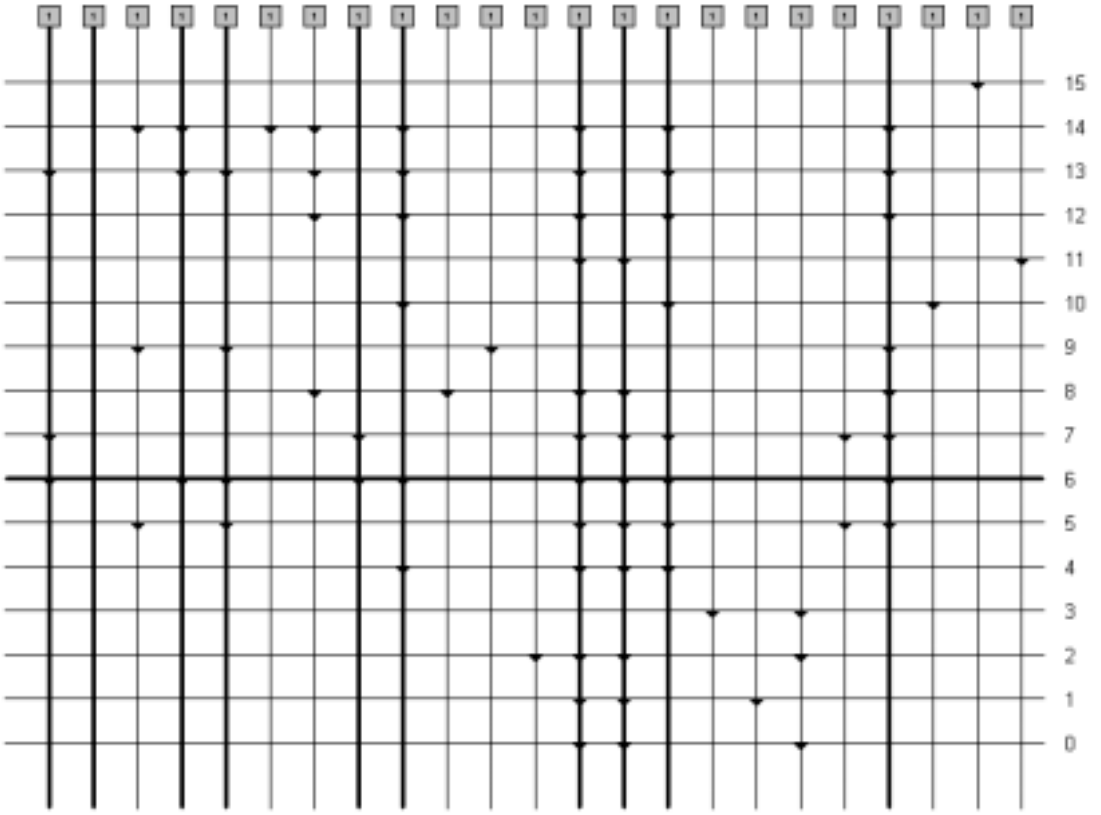
.

15 = $\overline{E_3} \vee \overline{E_2} \vee \overline{E_1} \vee \overline{E_0} = \overline{E_3 \wedge E_2 \wedge E_1 \wedge E_0}$

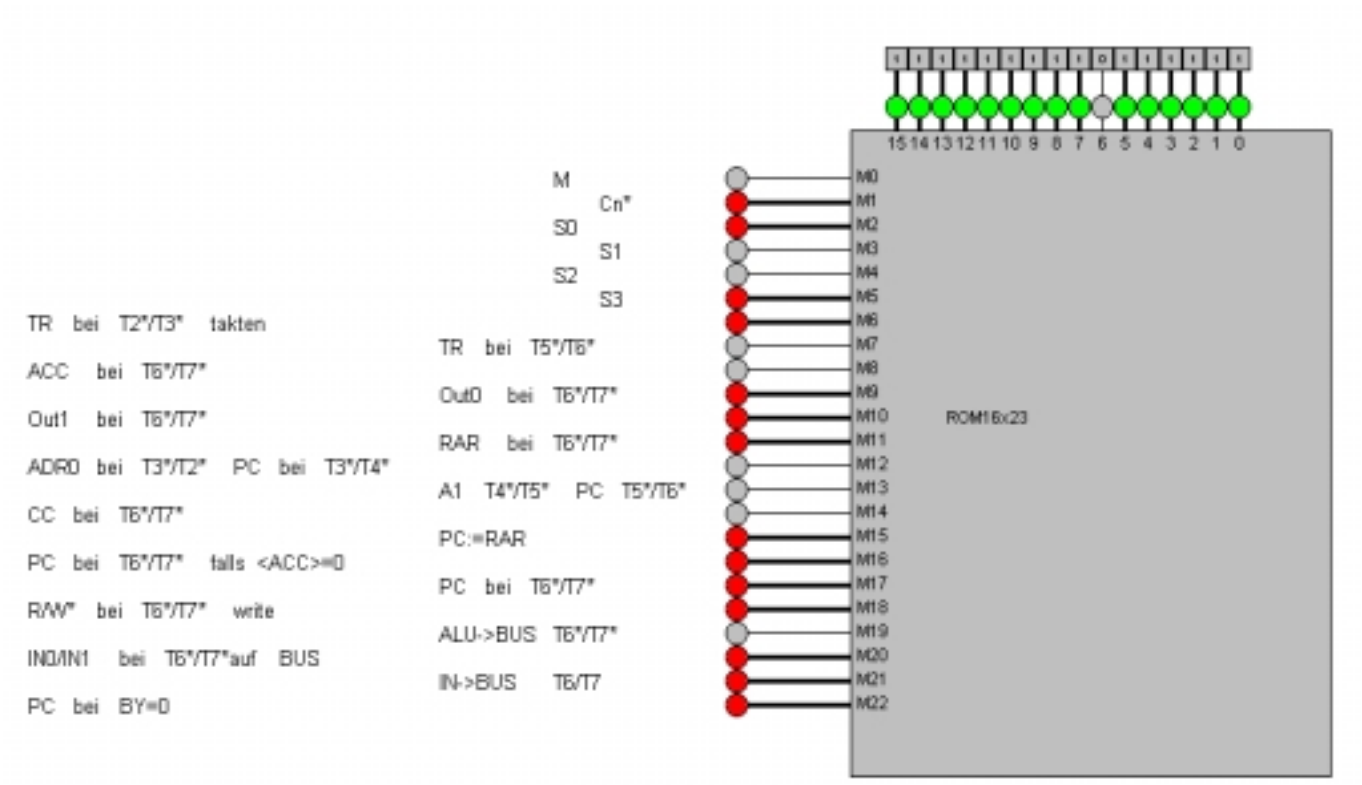
IC 74154



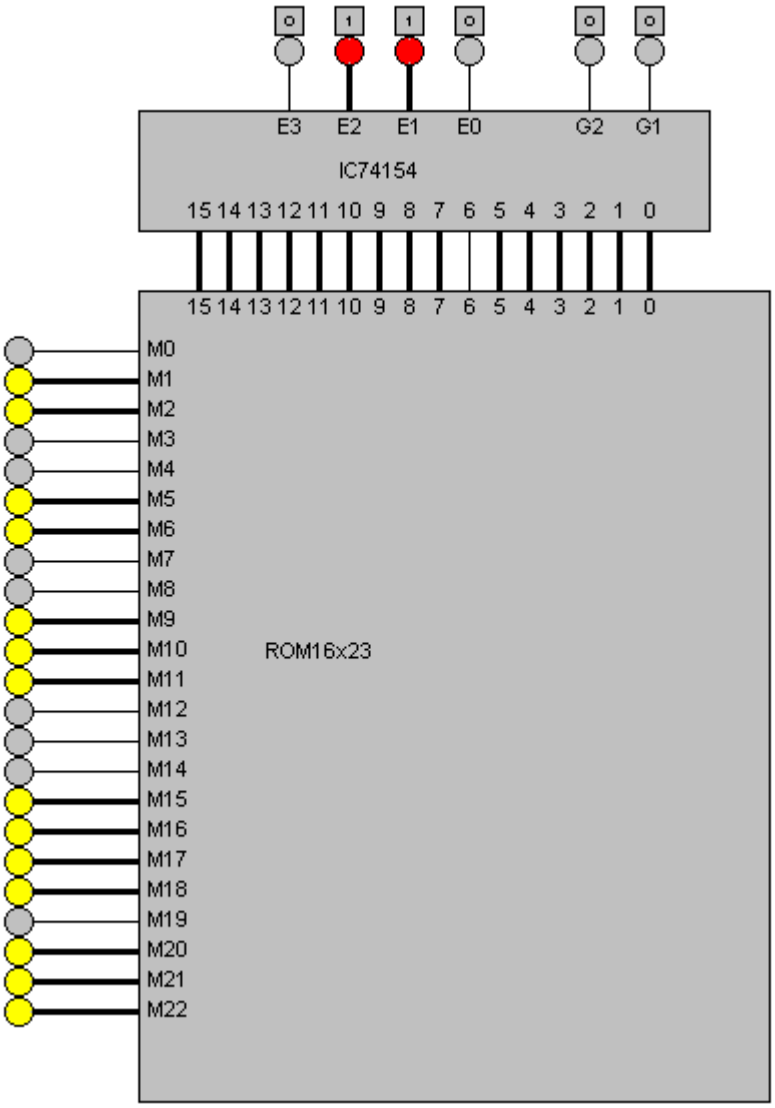
ROM - Diodenmatrix (hier realisiert mit LOCAD)



Testschaltung mit LOCAD : ROM.LWS

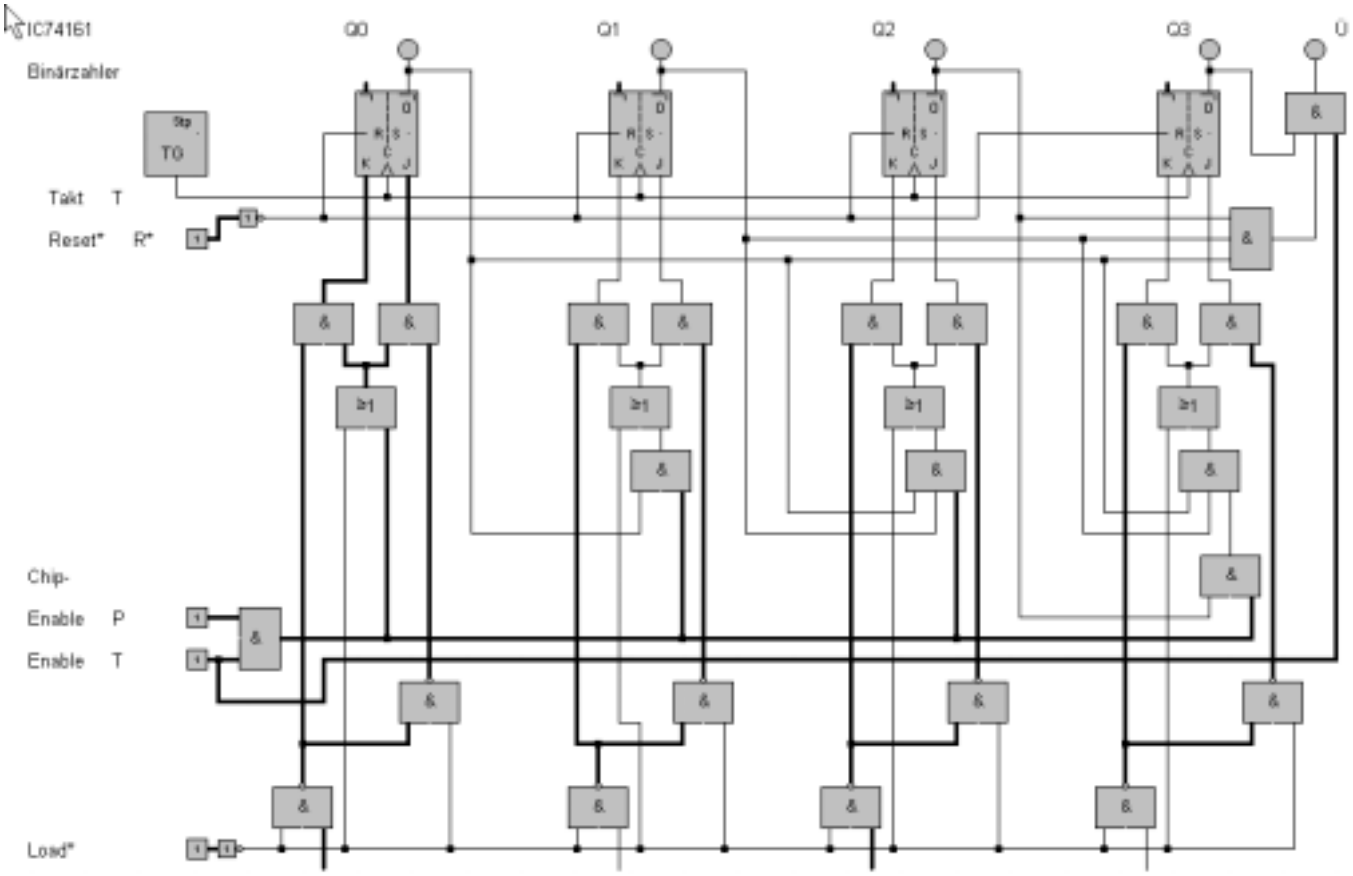


Testschaltung mit LOCAD : ROM16x23_TEST.LWS



Testschaltung mit LOCAD : IC74154_ROM.LWS

IC 74161 4-Bit-Binärzähler



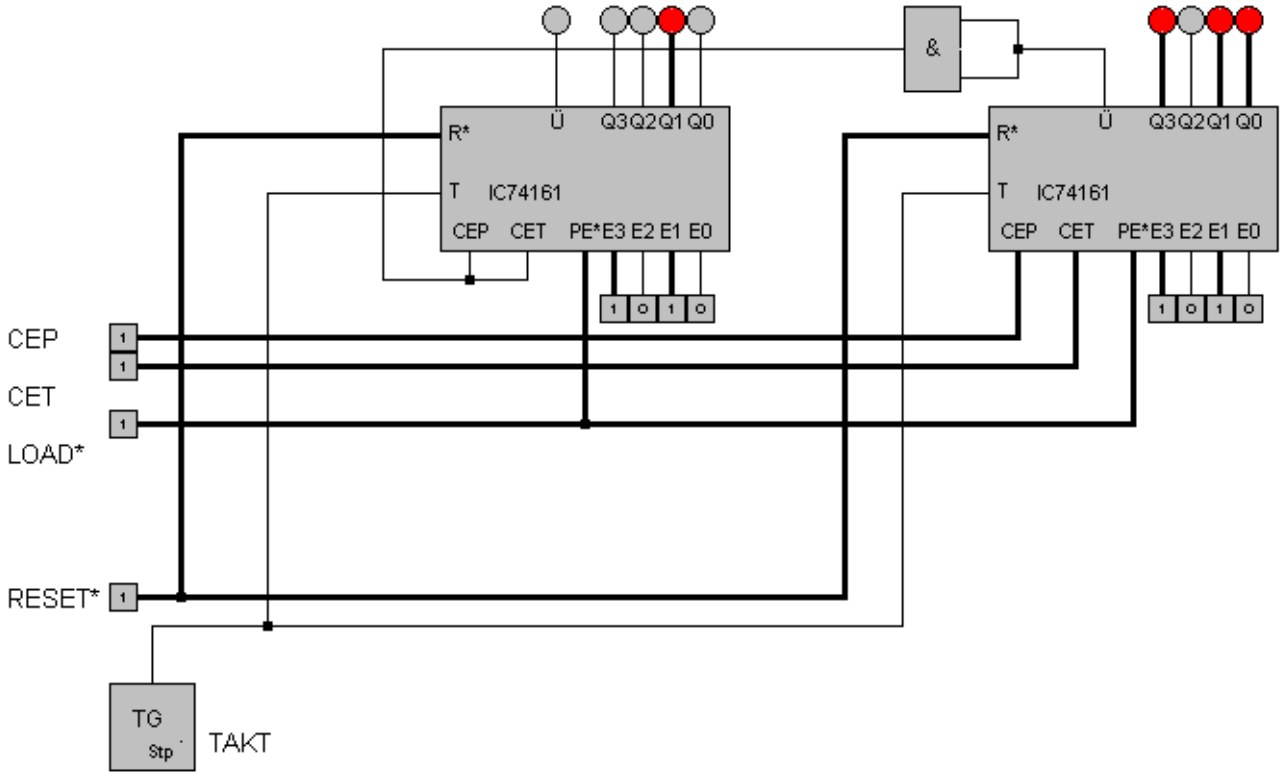
Testschaltung mit LOCAD : IC74161_2.LWS

Funktion :

Steuerleitungen					Ausgang	Operation
Reset*	Load*	CET	CEP	Takt	Q_3,Q_2,Q_1,Q_0	
0	X	X	X	X	0,0,0,0	Reset
1	0	X	X	↓	E_3,E_2,E_1,E_0	Preset
1	1	0	X	↓	Q = Q	Stop
1	1	X	0	↓	Q = Q	Stop
1	1	1	1	↓	Q = Q+1	Count

Der Program-Counter PC

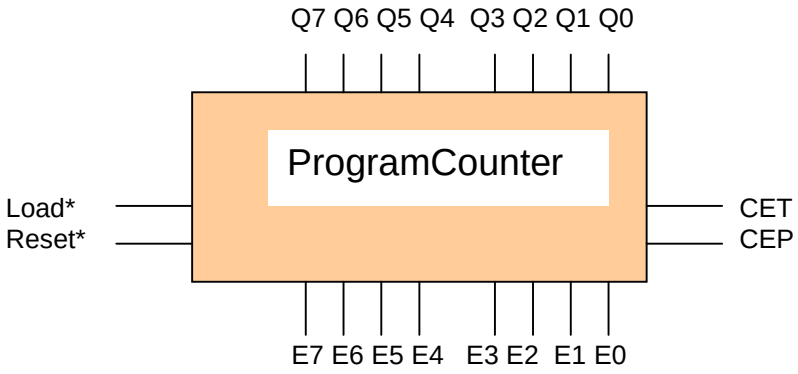
aufgebaut mit zwei IC 74161



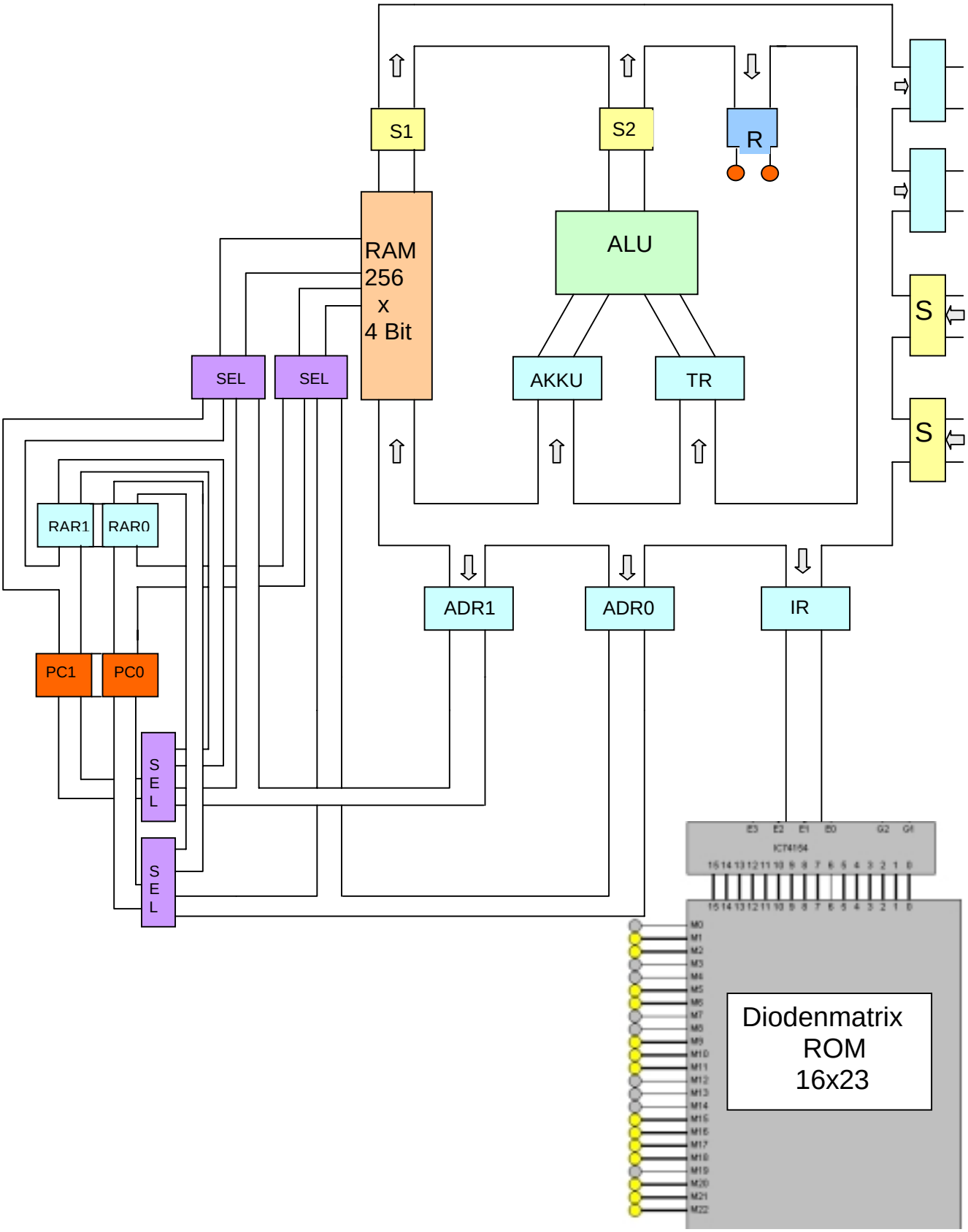
Testschaltung mit LOCAD : ProgramCounter.LWS

Der Program-Counter enthält immer die Adresse der Speicherzelle, die gerade angesprochen wird.

Blockschaltbild:



Die Erweiterung des Modellcomputers:



Programm : Addition zweier 4-Bit-Dualzahlen
Addiere den Inhalt der Speicherzelle 30 und den Inhalt der Speicherzelle 31 und speichere den Wert der Summe in der Speicherzelle 32 ab:

$$\langle 30 \rangle + \langle 31 \rangle \rightarrow \langle 32 \rangle$$

Struktogramm

Lade den Inhalt der Speicherzelle 30 in den Akku
Addiere zum Akkuinhalt den Inhalt der Speichezelle 31.
Speichere den Akkuinhalt in Speichezelle 32 ab
Wiederhole endlos %

Maschinenprogramm

Adr.	Mnemonic	Code				Bemerkung
0	LDA 30	0	1	0	0	LDA
1		1	1	0	0	30 ≙ 00011100
2		0	0	0	1	
3	ADDA31	0	1	1	0	ADDA
4		1	1	1	1	31 ≙ 00011111
5		0	0	0	1	
6	STA 32	0	1	0	1	STA
7		0	0	0	0	32 ≙ 00100000
8		0	0	1	0	
9	JMP 9	0	0	0	0	JMP
10		1	0	0	1	9 ≙ 00001001
11		0	0	0	0	(Endlosschleife !)
....						
....						
30		0	1	0	1	1. Summand
31		0	1	0	0	2. Summand
32		0	0	0	0	Wert der Summe
....						

Wie läuft nun der Befehl 0 LDA 30 detailliert ab ?

Fetch-Phase	$0 \rightarrow PC$	Damit ist die Speicherzelle 0 adressiert und der Inhalt dieser Zelle liegt auf dem BUS $\langle BUS \rangle = 1100$ Das ist der Code für LDA
	$\langle BUS \rangle \rightarrow IR$	Der Code für den Befehl LDA wird in das Instruktionsregister übernommen. Der Demultiplexer sorgt zusammen mit der Diodenmatrix dafür, dass die Steuerleitungen so gesetzt werden, dass der Befehl LDA ausgeführt werden kann.
	$\langle PC \rangle + 1 \rightarrow PC$	Der Inhalt des PC wird um 1 erhöht. Damit ist die Speicherzelle 1 adressiert und der Inhalt dieser Zelle liegt auf dem BUS $\langle BUS \rangle = 1110$ Das ist der niederwertige Adressteil LOW-Adress
	$\langle BUS \rangle \rightarrow ADR0$	Diese niederwertige Adresse wird ins nieder-wertige Adressregister ADR0 übernommen.
	$\langle PC \rangle + 1 \rightarrow PC$	Der Inhalt des PC wird um 1 erhöht. Damit ist die Speicherzelle 2 adressiert und der Inhalt dieser Zelle liegt auf dem BUS $\langle BUS \rangle = 0001$ Das ist der höherwertige Adressteil HIGH-Adress
Jetzt ist die Hol-Phase abgeschlossen, der Rechner weiß, was er zu tun hat.		
Execute-Phase	$\langle ADR1/ADR0 \rangle \rightarrow RAM$	Die Adresse (hier $30 \triangleq 00011110$) wird an den AdressBUS des RAM übernommen. Der Inhalt der Speicherzelle (hier 0101) liegt auf dem BUS
	$\langle BUS \rangle \rightarrow TR$	Die Steuerleitungen der ALU sind so belegt, dass F=A ist, d.h. der
	$\langle ALU \rangle \rightarrow BUS$	ALU-Ausgang F liegt auf dem BUS
	$\langle BUS \rangle \rightarrow Akku$	Der BUS-Inhalt wird in den Akku übernommen
	$\langle PC \rangle + 1 \rightarrow PC$	Die nächste Speicherzelle wir adressiert, d.h. der nächste Befehl wird vorbereitet.

Analog wird der Befehl 3 ADDA 31 abgearbeitet :

Fetch-Phase	$3 \rightarrow PC$	Damit ist die Speicherzelle 3 adressiert und der Inhalt dieser Zelle liegt auf dem BUS $\langle BUS \rangle = 0110$ Das ist der Code für ADDA
	$\langle BUS \rangle \rightarrow IR$	Der Code für den Befehl ADDA wird in das Instruktionsregister übernommen. Der Demultiplexer sorgt zusammen mit der Diodenmatrix dafür, dass die Steuerleitungen so gesetzt werden, dass der Befehl ADDA ausgeführt werden kann.
	$\langle PC \rangle + 1 \rightarrow PC$	Der Inhalt des PC wird um 1 erhöht. Damit ist die Speicherzelle 4 adressiert und der Inhalt dieser Zelle liegt auf dem BUS $\langle BUS \rangle = 1111$ Das ist der niederwertige Adressteil LOW-Adress
	$\langle BUS \rangle \rightarrow ADR0$	Diese niederwertige Adresse wird ins niederwertige Adressregister ADR0 übernommen.
	$\langle PC \rangle + 1 \rightarrow PC$	Der Inhalt des PC wird um 1 erhöht. Damit ist die Speicherzelle 5 adressiert und der Inhalt dieser Zelle liegt auf dem BUS $\langle BUS \rangle = 0001$ Das ist der höherwertige Adressteil HIGH-Adress
Jetzt ist die Hol-Phase abgeschlossen, der Rechner weiß, was er zu tun hat.		
Execute-Phase	$\langle ADR1/ADR0 \rangle \rightarrow RAM$	Die Adresse (hier $31 \triangleq 00011111$) wird an den AdressBUS des RAM übernommen. Der Inhalt der Speicherzelle (hier 0100) liegt auf dem BUS
	$\langle BUS \rangle \rightarrow TR$	Die Steuerleitungen der ALU sind so belegt, dass F=A+B ist, d.h. $F = \langle TR \rangle + \langle AKKU \rangle$
	$\langle ALU \rangle \rightarrow BUS$	ALU-Ausgang F liegt auf dem BUS
	$\langle BUS \rangle \rightarrow AKKU$	Der BUS-Inhalt wird in den Akku übernommen
	$\langle PC \rangle + 1 \rightarrow PC$	Die nächste Speicherzelle wir adressiert, d.h. der nächste Befehl wird vorbereitet.

Analog wird der Befehl 6 STA 32 abgearbeitet :

Fetch-Phase	$6 \rightarrow PC$	Damit ist die Speicherzelle 6 adressiert und der Inhalt dieser Zelle liegt auf dem BUS $\langle BUS \rangle = 0101$ Das ist der Code für STA
	$\langle BUS \rangle \rightarrow IR$	Der Code für den Befehl STA wird in das Instruktionsregister übernommen. Der Demultiplexer sorgt zusammen mit der Diodenmatrix dafür, dass die Steuerleitungen so gesetzt werden, dass der Befehl STA ausgeführt werden kann.
	$\langle PC \rangle + 1 \rightarrow PC$	Der Inhalt des PC wird um 1 erhöht. Damit ist die Speicherzelle 7 adressiert und der Inhalt dieser Zelle liegt auf dem BUS $\langle BUS \rangle = 0000$ Das ist der niederwertige Adressteil LOW-Adress
	$\langle BUS \rangle \rightarrow ADR0$	Diese niederwertige Adresse wird ins nieder-wertige Adressregister ADR0 übernommen.
	$\langle PC \rangle + 1 \rightarrow PC$	Der Inhalt des PC wird um 1 erhöht. Damit ist die Speicherzelle 8 adressiert und der Inhalt dieser Zelle liegt auf dem BUS $\langle BUS \rangle = 0010$ Das ist der höherwertige Adressteil HIGH-Adress
Jetzt ist die Hol-Phase abgeschlossen, der Rechner weiß, was er zu tun hat.		
Execute-Phase	$\langle ADR1/ADR0 \rangle \rightarrow RAM$	Die Adresse (hier $32 \triangleq 00100000$) wird an den AdressBUS des RAM übernommen. Steuerleitung R/W auf WRITE
	$\langle ALU \rangle \rightarrow BUS$	ALU-Ausgang F liegt auf dem BUS Steuerleitungen so, dass F=B, als $F = \langle AKKU \rangle$
	$\langle BUS \rangle \rightarrow RAM$	Der BUS-Inhalt wird in die adressierte Zelle des RAM-Bausteins übernommen
	$\langle PC \rangle + 1 \rightarrow PC$	Die nächste Speicherzelle wir adressiert, d.h. der nächste Befehl wird vorbereitet.

Programm : Subtraktion zweier 4-Bit-Dualzahlen
Subtrahiere den Inhalt der Speicherzelle 31 von dem Inhalt der Speicherzelle 30 und speichere den Wert der Differenz in der Speicherzelle 32 ab:

$$\langle 30 \rangle - \langle 31 \rangle \rightarrow \langle 32 \rangle$$

Idee: a-b = a+(-b)

Struktogramm

Lade den Inhalt der Speicherzelle 31 in den Akku
Bilde das Zweierkomplement des Akkuinhalts
Addiere zum Akkuinhalt den Inhalt der Speichezelle 30.
Speichere den Akkuinhalt in Speichezelle 32 ab
Wiederhole endlos | %

Maschinenprogramm

Adr.	Mnemonic	Code				Bemerkung
0	LDA 31	0	1	0	0	LDA
1		1	1	1	1	31 ≙ 00011111
2		0	0	0	1	
3	NANDA15	1	1	1	0	NANDA erzeugt Einerkomplement
4		1	1	1	1	15
5	INCA 1	1	1	0	1	INCA
6		0	0	0	1	1
7	ADDA 30	0	1	1	0	ADDA
8		1	1	0	0	30 ≙ 00011100
9		0	0	0	1	
10	STA 32	0	1	0	1	STA
11		0	0	0	0	32 ≙ 00100000
12		0	0	1	0	
13	JMP 13	0	0	0	0	JMP
14		1	1	0	1	13 ≙ 00001101
15		0	0	0	0	
....						
30		0	1	0	1	1. Summand
31		0	1	0	0	2. Summand
32		0	0	0	0	Wert der Summe

Mit dem Trick NANDA 15 erzeugt man das Einerkomplement und durch Addition einer 1 das Zweierkomplement :

A	0100
15	1111
$\overline{A \wedge 15}$	1011
$\overline{A \wedge 15}+1$	1100

$$5-4 = 5+(-4) \triangleq 0101+1100 = 1\boxed{0001} \triangleq +1$$
$$4-5 = 4+(-5) \triangleq 0100+1011 = \boxed{1111} \triangleq -1$$
$$-3-4 = (-3)+(-4) \triangleq 1101+1100 = 1\boxed{1001} \triangleq -7$$

Programm : Multiplikation zweier positiver 4-Bit-Dualzahlen
Multipliziere den Inhalt der Speicherzelle 30 mit dem Inhalt der Speicherzelle 31 und speichere den Wert des Produkts in der Speicherzelle 32 ab:

$$\langle 30 \rangle \bullet \langle 31 \rangle \rightarrow \langle 32 \rangle$$

Struktogramm

Wiederhole	$\langle 32 \rangle \rightarrow$ AKKU
	$\langle$AKKU$\rangle + \langle 32 \rangle \rightarrow$ AKKU
	$\langle 30 \rangle - 1 \rightarrow 30$
	$\langle$AKKU$\rangle \rightarrow 32$
	$\langle 30 \rangle \rightarrow$ AKKU
Solange bis	$\langle 30 \rangle = 0$
$\langle 32 \rangle \rightarrow$ AKKU	
$\langle$AKKU$\rangle \rightarrow$ Ausgabe	

Maschinenprogramm:

Adr.	Mnemonic	Code	Bemerkung
0	LDA 32	0 1 0 0	LDA
1		0 0 0 0	32 ≙ 00100000
2		0 0 1 0	
3	ADDA 31	0 1 1 0	ADDA
4		1 1 1 1	31 ≙ 00011111
5		0 0 0 1	
6	DECM 30	0 1 1 1	DECM
7		1 1 1 0	30 ≙ 00011110
8		0 0 0 1	
9	STA 32	0 1 0 1	STA
10		0 0 0 0	32 ≙ 00100000
11		0 0 1 0	
12	LDA 30	0 1 0 0	LDA
13		1 1 1 0	30 ≙ 00011110
14		0 0 0 1	
15	JNZ 0	0 0 0 1	JMP
16		0 0 0 0	0 ≙ 00000000
17		0 0 0 0	
18	LDA 32	0 1 0 0	LDA
19		0 0 0 0	32 ≙ 00100000
20		0 0 1 0	
21	OUT1	1 0 0 1	OUT1
22	JMP 22	0 0 0 0	JMP
23		0 1 1 0	22 ≙ 00010110
24		0 0 0 1	
….			
30		0 1 0 0	1. Faktor
31		0 0 1 1	2. Faktor
32		0 0 0 0	Wert des Produkts
....			

Aufgabe: Wandle das Programm so ab, dass es auch richtig arbeitet wenn
a) der erste Faktor negativ und der zweite Faktor positiv ist
b) beide Faktoren negativ sind