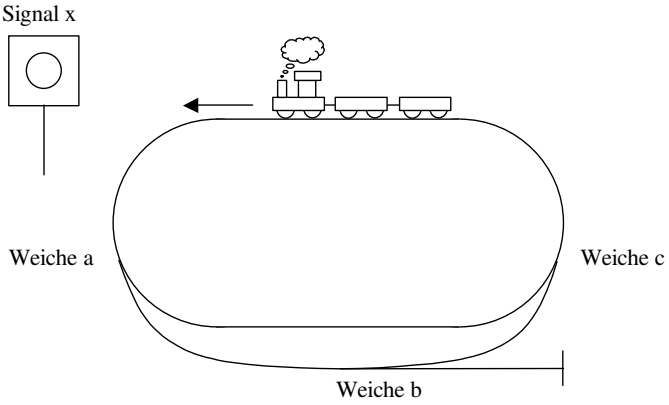
Skript

Schaltnetze / Schaltwerke

Die didaktischen Ideen dieses Skripts sind im wesentlichen dem Buch von Karl-Heinz Loch: "Technische Informatik mit LOCAD" aus dem Pädagogik & Hochschulverlag entnommen.

1 Einführendes Beispiel

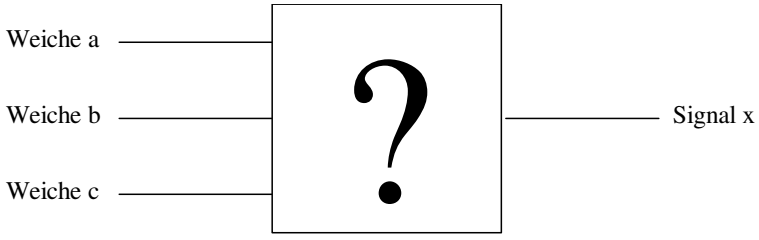
Schauen wir uns als erstes ein Beispiel an. Wir wollen die folgende Eisenbahnanlage davor schützen, dass der darauf fahrende Zug durch falsch eingestellte Weichen von den Schienen springt. Dazu soll ein Signal aufleuchten, wenn die gegebene Weichenstellung zu einem Entgleisen des Zuges führen würde.



Überlegen wir uns eine geeignete Kodierung für die Stellung der Weichen. Eine Weiche dieser Eisenbahnanlage kann entweder nach innen oder nach außen gestellt sein. Bezeichnen wir also eine nach **innen gestellte Weiche mit 0** und eine nach **außen gestellte Weiche mit 1**. Dann lassen sich die Möglichkeiten der Weichenstellungen und deren zugehöriges Signalverhalten übersichtlich in einer Tabelle darstellen:

Weiche a (0: innen / 1: außen)	**Weiche b**	**Weiche c**	**Signal x** (0: aus / 1: an)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Eine solche Tabelle nennt man auch **Schalttabelle**, da sie die Grundlage für eine **Schaltung** bildet, d. h., man versucht eine Schaltung zu entwerfen, die genauso arbeitet wie die Tabelle. Wir wollen auch solche Schaltungen entwerfen und nutzen dafür das Konstruktionsprogramm LOCAD, welches uns eine Vielzahl von Bauteilen zur Verfügung stellt. Machen wir uns also ans Werk und simulieren die Schaltung in LOCAD:
Wir haben drei Eingangsleitungen (a, b und c) für die Weichen und eine Ausgangsleitung für das Signal (x). Was dazwischen passiert, können wir nur beschreiben, da wir noch nicht die passenden Schaltelemente kennen.

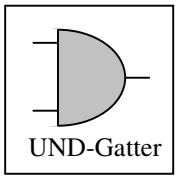


Nehmen wir unsere Tabelle für die Beschreibung, so können wir folgende Aussage treffen:

Signal x leuchtet, wenn
- a = 0 und b = 0 und c = 1 oder
- a = 0 und b = 1 und c = 1 oder
- a = 1 und b = 0 und c = 0

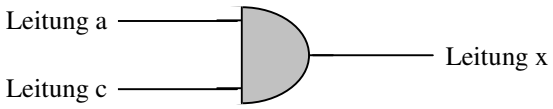
Wahrscheinlich hast du schon festgestellt, dass es bei den ersten beiden Möglichkeiten egal ist, ob die Weiche b nach innen oder außen gestellt wird. Man kann also die Aussage über das Signal x etwas vereinfacht darstellen:

Signal x leuchtet, wenn
- a = 0 und c = 1 oder
- a = 1 und b = 0 und c = 0


UND-Gatter

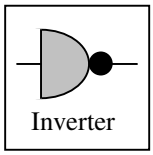
Kümmern wir uns als erstes um die erste Bedingung. In dieser Bedingung werden zwei Leitungen mit UND verknüpft. Das passende Bauteil – das **UND-Gatter** – sieht in LOCAD wie rechts abgebildet aus. Das Prinzip dieser Schaltung ist wie folgt: Liegt an allen Eingangsleitungen Strom an, so liegt auch an der Ausgangsleitung Strom an. Ist irgendeine der Eingangsleitungen aus, so ist auch die Ausgangsleitung aus.

Aufgabe 1: Erstelle mit LOCAD die folgende Minischaltung und teste alle Eingangskombinationen. Bei welchen Eingangskombinationen fließt auch Strom in der Ausgangsleitung? Stelle die Ergebnisse in einer Schalttabelle dar.

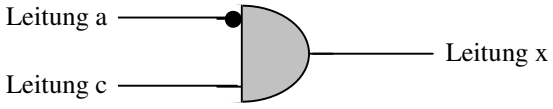


Leitung a (0: aus / 1: an)	**Leitung c**	**Leitung x**
0	0	
0	1	
1	0	
1	1	

Du hast festgestellt, dass die Ausgangsleitung x nur im 4. Fall an ist. Schauen wir uns noch einmal unser Beispiel an. Wir wollten eigentlich erreicht haben, dass Leitung x Strom führt, wenn a = 0 und c = 1 ist (und nicht wenn a = 1 und c = 1 ist). Wenn wir nun ein Bauteil hätten, welches genau dann Strom durchlässt, wenn das Eingangssignal aus ist und genau dann keinen Strom durchlässt, wenn das Eingangssignal an ist, dann könnten wir dieses Bauelement vor dem UND-Gatter in Leitung a einbauen.
Ein solches Bauteil heißt **Inverter** und dreht den Strom einer Leitung genau um. Das Bauelement in LOCAD sieht wie rechts abgebildet aus.
Häufig wird statt des Inverters nur ein Negationspunkt gesetzt. Die folgende Schaltung liefert damit den gewünschten Effekt für die erste Bedingung unseres Weichen-Beispiels:


Inverter

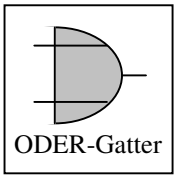
Aufgabe 2: Prüfe die folgende Schaltung mit LOCAD, und fülle zur Sicherheit die Schalttabelle aus.



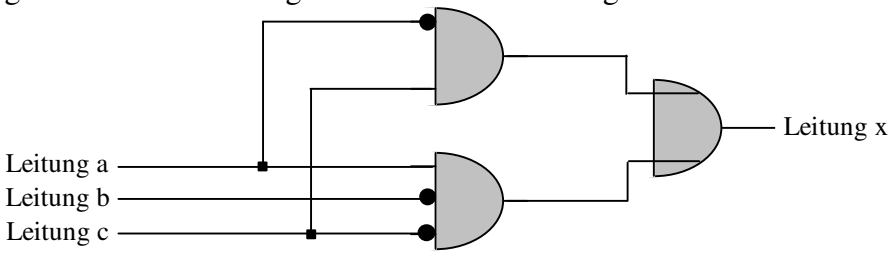
Leitung a (0: aus / 1: an)	**Leitung c**	**Leitung x**
0	0	
0	1	
1	0	
1	1	

Aufgabe 3: Erstelle nun eine Schaltung für die zweite Bedingung (UND-Gatter mit drei Eingangsleitungen und einer Ausgangsleitung) die lautet: a = 1 und b = 0 und c = 0. Teste sie mit LOCAD.

Kommen wir nun zur Verknüpfung der beiden Bedingungen des Beispiels. Das Signal sollte leuchten, wenn eine der beiden Bedingungen erfüllt war, d. h. das Signal x ist gesetzt, wenn Bedingung 1 erfüllt ist ODER Bedingung 2 erfüllt ist. Das passende Bauteil – das **ODER-Gatter** – sieht in LOCAD wie rechts abgebildet aus. Das Prinzip dieser Schaltung ist wie folgt: Liegt an keiner der Eingangsleitungen Strom an, so liegt auch an der Ausgangsleitung kein Strom an. Ist irgendeine der Eingangsleitungen an, so ist auch die Ausgangsleitung an.



Insgesamt ergibt sich damit die folgende LOCAD-Schaltung für unser Weichen-Beispiel:



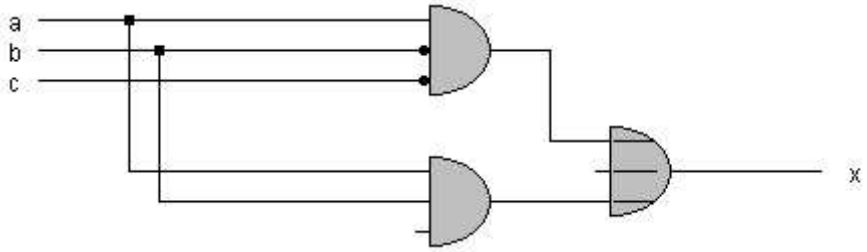
Die eckigen Verbindungspunkte für die Leitungen a und c sind deshalb erforderlich, da man sonst nicht weiß, welche Leitungen verbunden sind und welche nicht. Würde man diese Verbindungspunkte nicht machen, so wären keine Überkreuzungen von Leitungen möglich.

Aufgabe 4: Realisiere die Schaltung in LOCAD und überprüfe damit die Schalttabelle unseres Beispiels.

Aufgabe 5: Erstelle mit LOCAD zur folgenden Schalttabelle eine Schaltung. Überlege dir zuerst wie im Weichen-Beispiel, ob man mehrere Bedingungen eventuell zusammenfassen kann. (Man nennt dies „eine Schaltung optimieren“.)

Leitung a	Leitung b	Leitung c	Ausgang x
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Aufgabe 6: **a)** Welche Schalttabelle gehört zur folgenden Schaltung? Teste dein Ergebnis mit LOCAD.



b) Kann man diese Schaltung einfacher gestalten? Versuche den gleichen Effekt mit weniger Bauelementen (Gattern) zu erhalten. Schaue dir dazu die Schalttabelle genauer an.

Informatiker sind von Natur aus faule Menschen. Sie suchen stets nach Möglichkeiten, sich das Leben einfacher zu gestalten. Da das Aufschreiben einer Schalttabelle sehr aufwendig ist, benutzt er Abkürzungen:
ODER wird abgekürzt durch $\vee$
UND wird abgekürzt durch $\wedge$
a = 1 wird abgekürzt durch **a**
a = 0 wird abgekürzt durch $\bar{\mathbf{a}}$ (gesprochen : „nicht a“)

Damit kann man die Schalttabelle aus dem Einführungsbeispiel mit der Eisenbahnanlage wie folgt schreiben: Signal x leuchtet, wenn

a = 0 und b = 0 und c = 1 a = 0 und b = 1 und c = 1 a = 1 und b = 0 und c = 0	oder oder	$s = \bar{a} \wedge \bar{b} \wedge c \vee \bar{a} \wedge b \wedge c \vee a \wedge \bar{b} \wedge \bar{c}$

bzw. in der bereits vereinfachten Form:

a = 0 und c = 1 a = 1 und b = 0 und c = 0	oder	$s = \bar{a} \wedge c \vee a \wedge \bar{b} \wedge \bar{c}$

Aufgabe 7: Gib zu den Schalttabellen aus Aufgabe 5 und 6 jeweils die kurze Schreibweise an.

2 Schaltnetze

Die Schaltung, die wie im ersten Kapitel kennen gelernt haben, nennt man ein Schaltnetz. In diesem Kapitel wollen wir weitere Schaltnetze entwickeln und optimieren.
Erinnerst Du dich noch an die Addition von Binärzahlen? Genau diese Addition wollen wir als Schaltnetz realisieren. Fangen wir erst mal mit dem einfachsten an: der Addition von zwei Bits. Bei dieser Addition kann je nach zu addierenden Zahlen die Summe 1 oder 0 betragen. Werden sogar zwei Einsen addiert, so erhält man einen Übertrag und die Summe wird 0. Genau wie bei der Eisenbahnanlage kann man dieses Verhalten für die Summe und den Übertrag mit Hilfe einer Schaltung nachbauen. Wir benötigen dafür lediglich zwei Eingangsleitungen a und b, welche die beiden zu addierenden Bits darstellen, und zwei Ausgangsleitungen – eine zur Anzeige der Summe (s) und eine zur Anzeige des Übertrags (ü). In der Kurzschreibweise für Schalttabellen ergibt sich folgende Belegung der Ausgangsleitungen:

$$s = \bar{a} \wedge b \vee a \wedge \bar{b} \qquad \text{und} \qquad ü = a \wedge b$$

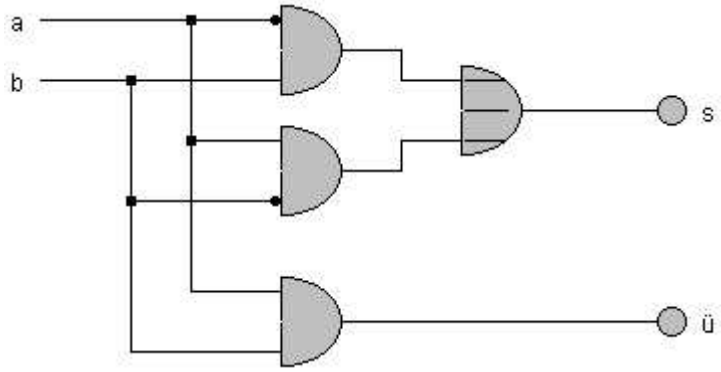
Aufgabe 8: Erstelle zu der Kurzschreibweise die zugehörige Schalttabelle sowohl für den Summen-Ausgang s als auch für den Übertrag-Ausgang ü. Überzeuge dich von der Korrektheit der beiden Formeln.

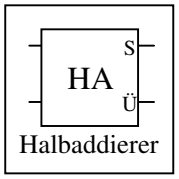
Aufgabe 9: **a)** Entwerfe mit LOCAD eine Schaltung mit den Eingängen a und b und den Ausgang s (Summe).
b) Entwerfe eine Schaltung für den Ausgang ü (Übertrag)
c) Kombiniere beide Schaltungen in einer Schaltung mit den Eingängen a und b und den Ausgängen s und ü.

So, ich hoffe, du konntest die beiden Aufgaben lösen. Die Schalttabelle sieht wie folgt aus:

1. Bit (a)	2. Bit (b)	Summe (s)	Übertrag (ü)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Die zugehörige Schaltung:





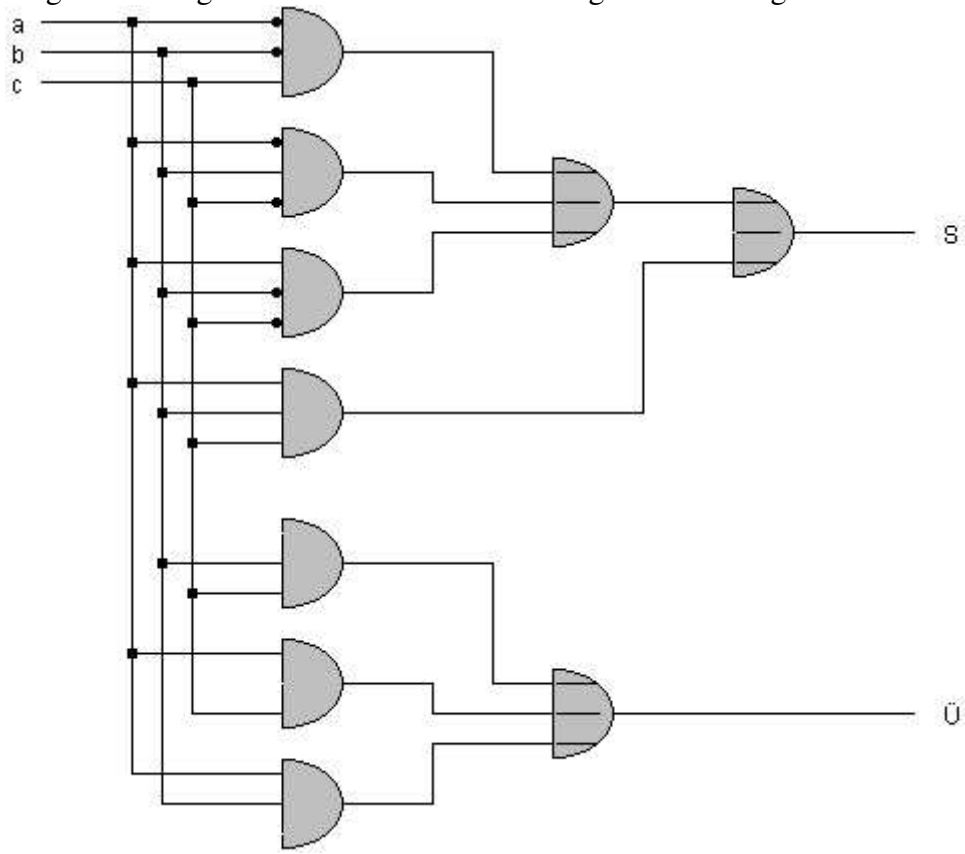
Da man diese Schaltung sehr häufig benötigt, stellt LOCAD dir dafür ein eigenes Bauteil (siehe rechter Bildschirmrand) zur Verfügung. Man nennt dieses Bauteil auch Halbaddierer. Ja, warum denn nur Halbaddierer? Was ist dann wohl ein Volladdierer? Bei der Addition von zwei Binärzahlen fängt man bei der letzten Stelle an zu addieren. Bei dieser letzten Stelle kommen wir auch mit einem solchen Halbaddierer klar. Was ist allerdings, wenn ein Übertrag zur zweitletzten Stelle auftritt? dann müssen drei Bits addiert werden – die zwei Bits plus den evtl. Übertrag. Eine Schaltung die dieses leistet benötigt also drei Eingänge (a und b: die beiden zu addierenden Bits und c den evtl. Übertrag) und zwei Ausgänge (die Summe s und den Übertrag ü). Wie müsste die Schalttabelle für diese Schaltung aussehen? Fangen wir mit dem Summen-Ausgang an: dieser ist 0, wenn entweder kein Eingang belegt ist oder wenn zwei Eingänge belegt sind. Er ist 1 wenn entweder nur ein Eingang belegt ist oder wenn alle drei Eingänge belegt sind. Bei dem Übertrag-Ausgang sieht es so aus, dass er 1 ist, wenn mehr als eine Eingang belegt ist. Als Schalttabelle geschrieben:

1. Bit (a)	2. Bit (b)	evtl. Übertrag (c)	Summe (s)	Übertrag (ü)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

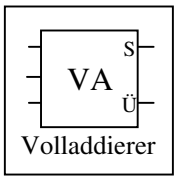
Aufgabe 10: **a)** Erstelle mit LOCAD die zugehörige Schaltung und Überprüfe ihre Korrektheit.
b) Optimiere die Schaltung, indem du überprüfst, ob einige Zeilen zusammengefasst werden können. Erstelle auch zur Optimierung eine Schaltung.

Und, hast du die Lösung gefunden?

Die zugehörige Schaltung inklusive ihrer Vereinfachung sieht wie folgt aus:



Mögliche Vereinfachungen ergeben sich nur bei dem Übertrag. Dieser kann mit nur drei UND-Gattern und einem ODER-Gatter realisiert werden. Da auch diese Schaltung relativ Häufig benötigt wird, stellt LOCAD auch dazu ein Bauteil zur Verfügung: den Volladdierer.



Aufgabe 11: Prüfe die beiden Bauteile Halbaddierer und Volladdierer mit LOCAD. Vergleiche die Ausgänge s und ü mit den Schalttabellen.

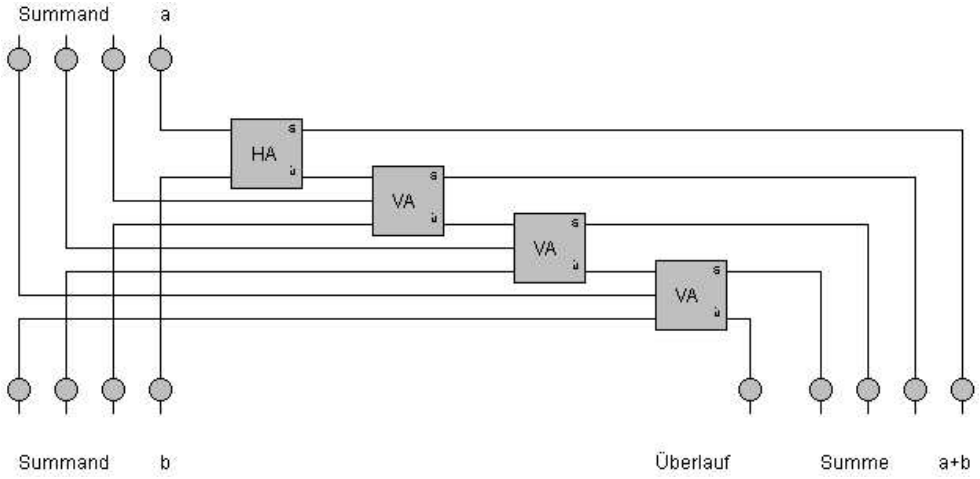
Versuchen wir nun, die Halbaddierer und Volladdierer zu verwenden, um eine Schaltung aufzubauen, welche zwei 4-Bit Binärzahlen addiert. Analysiere dazu an der rechts abgebildeten Rechenaufgabe das Vorgehen bei der Addition zweier Binärzahlen:

```
  0101
+ 1111
= (1) 0100
```

1. Schritt: Addiere die beiden letzten Bits der Summanden mit Hilfe eines Halbaddierers. Die Summe ist das letzte Bit des Ergebnisses.
2. Schritt: Addiere die beiden vorletzten Bits der Summanden und den Übertrag aus dem 1. Schritt mit Hilfe eines Volladdierers. Die Summe ist das vorletzte Bit des Ergebnisses.
3. Schritt: Addiere die beiden drittletzten Bits der Summanden und den Übertrag aus dem 2. Schritt mit Hilfe eines Volladdierers. Die Summe ist das drittletzte Bit des Ergebnisses.
4. Schritt: Addiere die beiden ersten Bits der Summanden und den Übertrag aus dem 3. Schritt mit Hilfe eines Volladdierers. Die Summe ist das erste Bit des Ergebnisses.
5. Schritt: Der Übertrag aus dem 4. Schritt zeigt an, ob ein Überlauf vorlag.

Aufgabe 12: Erstelle eine Schaltung, welche zwei 4-Bit Binärzahlen addiert.

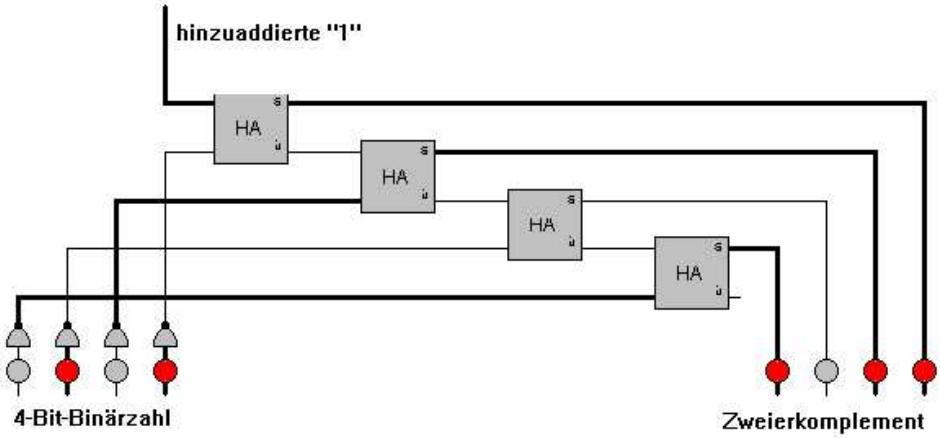
Und? Hat es geklappt? Du solltest ungefähr das folgende Schaltbild herausbekommen haben:



Man nennt diese Schaltung auch einen **Paralleladdierer**. „Parallel“ deshalb, da alle Bits der Summanden zeitgleich verarbeitet werden.

Kümmern wir uns nun um die Subtraktion von Binärzahlen. Du weißt hoffentlich noch, dass die Subtraktion realisiert werden kann, indem man zum Subtrahenden das Zweierkomplement bildet und schließlich Minuend und Zweierkomplement addiert. Deshalb müssen wir zuerst eine Schaltung entwerfen, welche zu einer Binärzahl das Zweierkomplement bildet.
Erinnerung: Das Zweierkomplement wird gebildet, indem man zum Einerkomplement der Zahl eine „1“ hinzuaddiert.

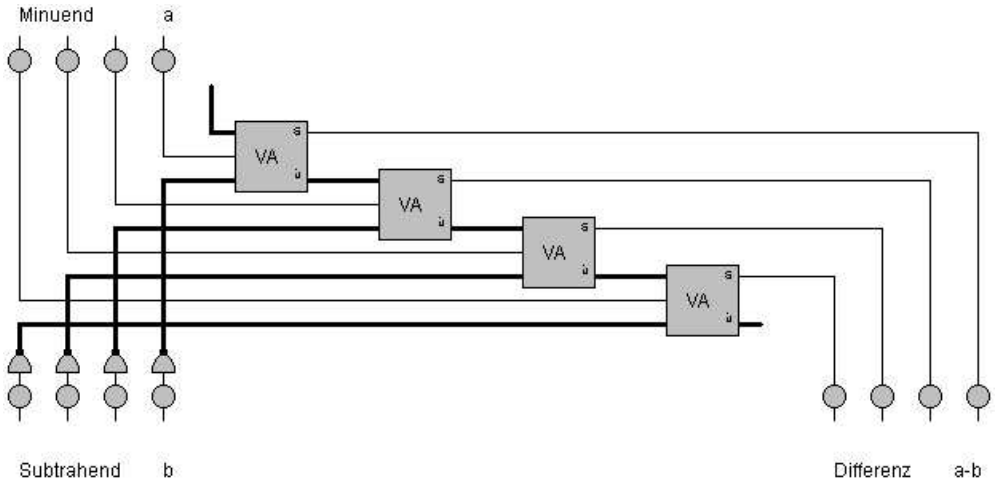
Aufgabe 13: Erstelle und analysiere folgende Schaltung mit LOCAD. Überprüfe seine Funktionsweise und teste an den Zahlen 5_{10} (schon eingestellt) , 12_{10} und -7_{10}, ob auch tatsächlich die Gegenzahl berechnet wird.



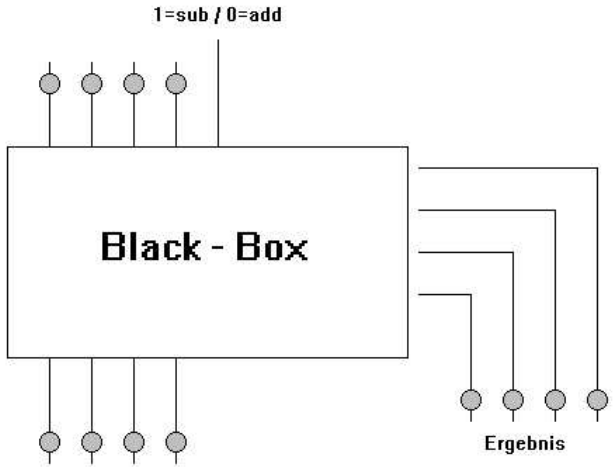
So, das wäre geschafft. Aber wie können wir nun einen **Parallelsubtrahierer** bauen? Schaue Dir dazu noch einmal die Schaltungen für den Paralleladdierer und das Zweierkomplement an. Wenn Du die Schaltungen geeignet kombinierst, dann klappt es bestimmt.

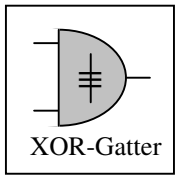
Aufgabe 14: Erstelle eine Schaltung zum Parallelsubtrahierer (Subtraktion von zwei 4-Bit-Binärzahlen) unter Verwendung der oben genannten Schaltungen.

Ich hoffe, du hast es herausbekommen. Die folgende Schaltung subtrahiert wie gewünscht zwei 4-Bit-Binärzahlen. Solltest Du die Schaltung nicht herausbekommen haben, dann teste sie wenigstens mit LOCAD.



Wir wollen noch einen Schritt wagen. Es wäre schön, wenn man statt zwei autonomen Schaltnetzen für Addition und Subtraktion ein einziges Schaltnetz hätte, welches beide Rechenarten beherrscht. Als Black-Box könnte man die Schaltung wie folgt zeichnen:



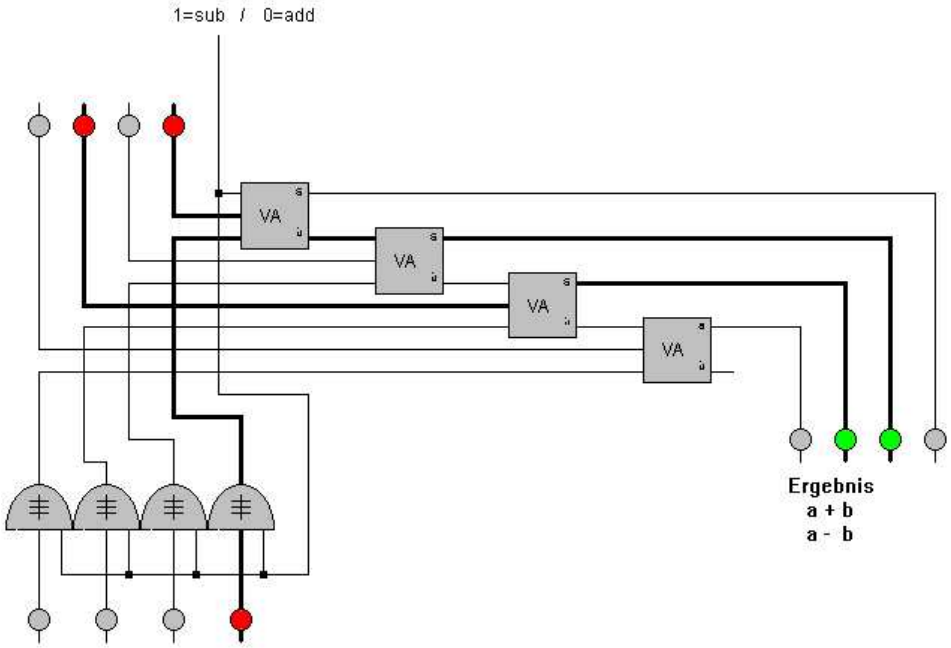


Dabei kann man das Schaltnetz über die obere Eingangsleitung steuern, ob nun die Addition oder Subtraktion durchgeführt werden soll. Die Schwierigkeit wird sein, je nach Steuerleitungszustand (Sub/Add) vom zweiten Operator das Zweierkomplement zu bilden oder den Operator so zu lassen, wie er ist. Wenn die Steuerleitung 1 ist, dann sollen beim zweiten Operator die Bits gekippt werden, wenn die Steuerleitung 0 ist, dann sollen die Originale durchgelassen werden. Eine Schaltung, welche dies leistet ist das **XOR-Gatter**. „XOR“ bedeutet soviel wie „Entweder Oder“. Und genauso verhält sich das Bauelement auch: sind mehr als ein Eingang oder gar kein Eingang belegt, so liefert der Ausgang eine 0. Ansonsten liefert der Ausgang eine 1.

Leitung a	Leitung b	Leitung x
0	0	0
0	1	1
1	0	1
1	1	0

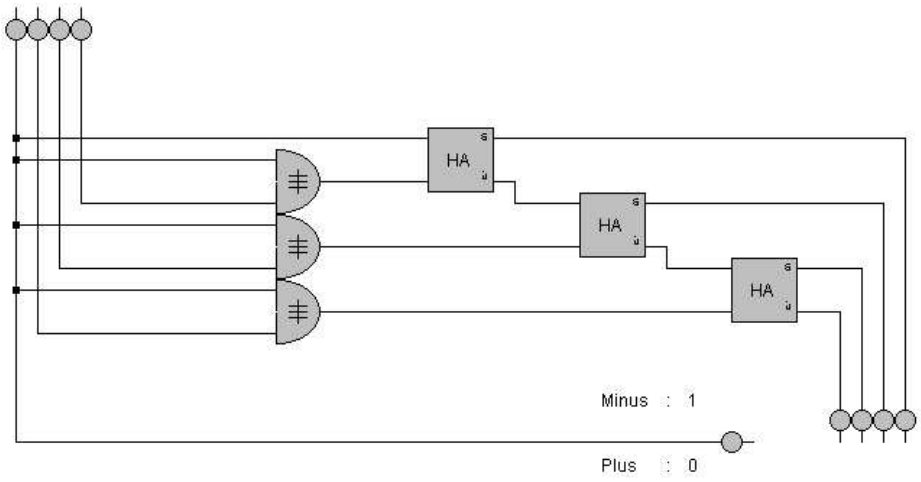
Aufgabe 15: Nutze dieses Bauelement, um einen umschaltbaren Paralleladdierer/–subtrahierer zu entwerfen. Probier es zuerst alleine und schaue dann erst die Lösung an.

Ich gebe zu, die Schaltung war nicht gerade einfach – aber doch machbar. Du siehst hier, wie mit Hilfe der XOR-Gatter der gewünschte Effekt beim zweiten Operanden erzielt wird. Wenn du die Aufgabe nicht gelöst hast, dann baue diese Schaltung wenigstens mit LOCAD nach und teste sie ausführlich.

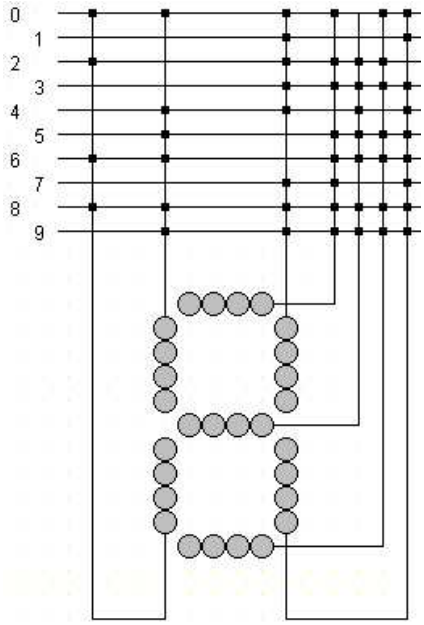


Jetzt wäre es noch schön, wenn am Ergebnis klarer zu erkennen wäre, welche Zahl berechnet wurde. Dazu sollst Du nun eine Schaltung entwickeln, welche zu einer vorzeichenbehafteten 4-Bit-Binärzahl deren Betrag mit vier Leuchtdioden und deren Vorzeichen mit einer Leuchtdiode anzeigt. Mit anderen Worten heißt das, dass zur 4-Bit-Binärzahl im Falle eines gesetzten Vorzeichenbits das Zweierkomplement auf den Betragsausgang gelegt werden muss. Ansonsten können die vier Eingangsleitungen direkt durchgeschaltet werden.

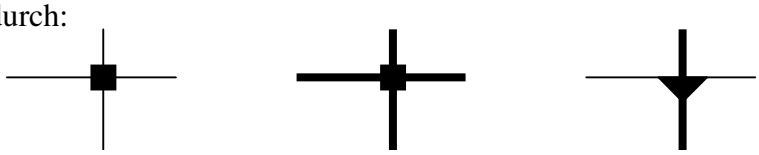
Aufgabe 16: Entwickle eine Schaltung, welche eine 4-Bit-Binärzahl mit fünf LEDs in folgender Form darstellt: 1. Bit: Vorzeichen 2. – 5. Bit: Betrag der 4-Bit-Binärzahl.



Bisher waren unsere Überlegung eher mathematischer Natur. Dies soll sich jetzt ändern. Wir wollen als nächstes eine Schaltung aufbauen, welche eine Sieben-Segment-Anzeige (du kennst sie vielleicht von deinem Radiowecker) steuert. Dazu soll die darzustellende Zahl durch 10 Eingangsleitungen (Zahlen 0 bis 9) eingegeben werden. Die Ausgabe erfolgt über sieben Ausgangsleitungen, welche zu den einzelnen Segmenten der Anzeige führen. Prinzipiell ist die Sache einfach: Wir müssen lediglich von der Eingangsleitung zur Zahl 1 den Strom für die rechten beiden Segmente abgreifen. Dies können wir, indem wir die Leitungen an der entsprechenden Stelle verbinden:



Bevor du diese Schaltung mit LOCAD nachbaust überlege dir zuvor, was passieren würde, wenn man auf die Leitung 1 Strom legen würde. Laufe die Leitung 1 ab und teile überall dort den Strom, wo ein Verbindungspunkt sitzt. Du siehst, dass der Strom durch alle vorhanden Leitungen fließt, da LOCAD keinen Unterschied zwischen Eingans- und Ausgangsleitung macht. Der Strom läuft von Eingangsleitung 1 über die erste Ausgangsleitung zur Eingangsleitung 2 und damit auch zu falschen Ausgangsleitungen.
Eine Möglichkeit, dies zu umgehen sind sogenannte Koppeldioden. Diese lassen Strom nur in eine Richtung durch:



Verbindung ohne Strom | Verbindung mit Strom | Koppelung: Strom überspringt Leitung

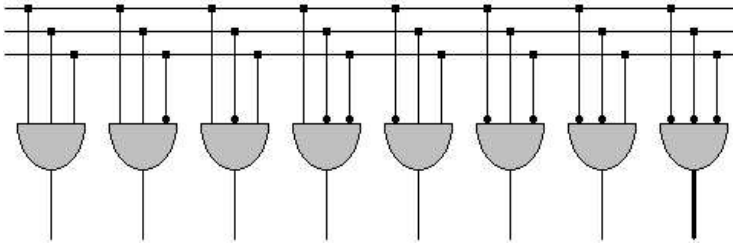
Ersetzt man die Verbindungen durch diese Koppeldioden, so haben wir den gewünschten Effekt: Die Ausgangsleitungen beeinflussen nicht die Eingangsleitungen.

Aufgabe 17: Teste die Schaltung mit Koppeldioden mit Hilfe von LOCAD. Du findest die Koppeldioden unter dem Menüpunkt *Leitungen|Koppeln*.

Aufgabe 18: Teste die Funktionsweise eines Decoders ohne Takteingang. Du findest das Bauteil unter *Bauteile|Decoder|Ohne Takteingang*. Integriere das Bauteil passend in die Schaltung der Sieben-Segment-Anzeige.

Aufgabe 19: Wie müsste die Schaltung zum Decoder aussehen? Entwerfe eine Decoderschaltung mit LOCAD.

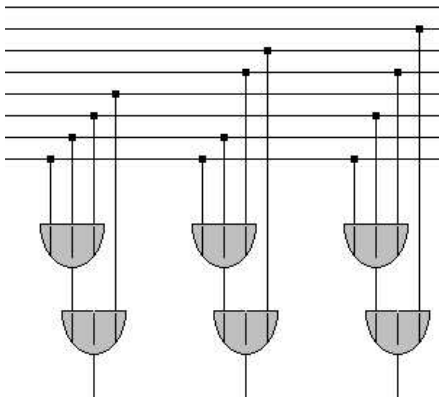
Ich hoffe, du bist nicht an der letzten Aufgabe verzweifelt. Die richtige Lösung sieht wie folgt aus: Der sogenannte 3 zu 8 Decoder – drei Eingangsleitungen und 8 Ausgangsleitungen.



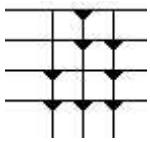
Genauso gut kann man die Frage umdrehen und eine Schaltung suchen, welche eine 8 zu 3 Decodierung vornimmt.

Aufgabe 20: Versuche in LOCAD eine Schaltung aufzubauen, welche aus acht Eingangsleitungen (eine für jede Zahl zwischen 0 und 7) eine 3-Bit-Binärzahl macht, d. h. auf drei Ausgangsleitungen passend verteilt. Man nennt diese Schaltung den 8 zu 3 Decoder.

Versuche es erst einmal alleine und schaue dir dann erst die folgende Lösung an. Fällt Dir vielleicht auch eine Lösung ein, die nur mit Koppeldioden arbeitet?



Nun aber zurück zu den Koppeldioden. Als wir über den Aufbau eines Computers gesprochen haben, fielen Begriffe wie **RAM** (Random Access Memory), Hauptspeicher und **ROM** (Read Only Memory). Der Hauptspeicher ist ein RAM, du kennst es sicherlich aus dem Computerprospekt: „512 MB RAM Arbeitsspeicher“ steht dort des öfteren. Ein ROM ist ein fest definierter, unveränderbarer Speicher. Wir wollen einen solchen Speicher, der die ersten 4 Primzahlen enthält, nachbauen – und zwar mit Koppeldioden. Mit vier Eingangsleitungen soll der Speicher adressiert werden. Das Ergebnis soll als 3-Bit-Binärzahl ausgegeben werden.

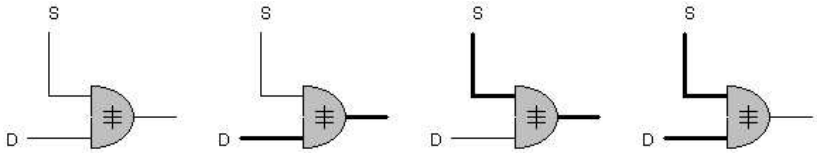


Diese Schaltung gibt je nach belegter Eingangsleitung 1 bis 4 die 1. Primzahl (2), 2. Primzahl (3), 3. Primzahl (5) oder 4. Primzahl (7) auf die Ausgangsleitungen.

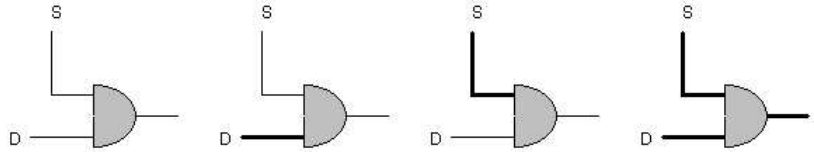
Aufgabe 21: Erweitere deine Sieben-Segment-Anzeige um dieses ROM, so dass die 1. bis 4. Primzahl im Display erscheint.

So, das war nun fast alles zu den Schaltnetzen. Bevor wir jedoch zu den Schaltwerken kommen, müssen wir uns noch einmal etwas intensiver mit der Methode von Steuerleitungen beschäftigen. Du kennst die Methode bereits aus dem umschaltbaren Paralleladdierer/-

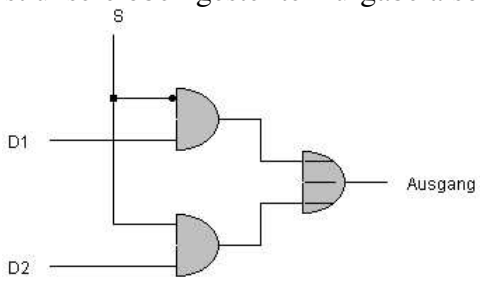
subtrahierer sowie aus der Aufgabe, in der du eine Schaltung entwickeln solltest, die zu einer 4-Bit-Binärzahl mit Vorzeichen den zugehörigen Betrag darstellt. Dort war in beiden Fällen die Schwierigkeit, einen Dateneingang (z. B. das letzte Bit der 4-Bit-Binärzahl) je nach Steuerleitung (1 oder 0) zu invertieren oder einfach nur durchzuschalten. Die Lösung bestand in der Verwendung eines XOR-Gatters. Damals haben wir gesehen, dass man das XOR-Gatter als „steuerbaren Inverter“ verwenden kann. Schauen wir uns das noch mal an den konkreten Möglichkeiten an: (S ist die Steuerleitung, D ist die Datenleitung, welche je nach Steuerleitung invertiert werden soll)



Ist die Steuerleitung aus (1. und 2. Fall) so wird die Datenleitung durchgeschaltet. Ist die Steuerleitung an (3. und 4. Fall) so wird die Datenleitung invertiert. Der gewünschte Effekt ist also erreicht.
Die Frage ist nun, ob man auch eine Schaltung aufbauen kann, welche mit Hilfe einer Steuerleitung eine Datenleitung „unterbricht“. Konkret ist folgendes Problem: Je nach Zustand einer Steuerleitung (S = 0 oder 1) soll entweder die Datenleitung 1 (D1) oder die Datenleitung 2 (D2) durchgeschaltet werden. Die jeweils andere soll für den Ausgang keine Rolle spielen. Man soll also praktisch mit Hilfe der Steuerleitung auswählen können, welche Leitung auf den Ausgang gelegt wird. Um dieses zu erreichen sollten wir uns noch mal die Wirkung der UND-Gatter vor Augen führen:

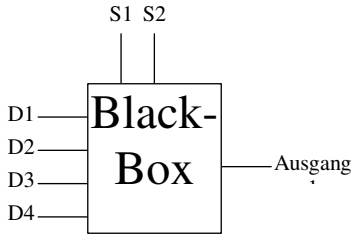


Ist die Steuerleitung aus (1. und 2. Fall) so wird die Datenleitung einfach ignoriert (Ausgang = 0). Ist die Steuerleitung an (3. und 4. Fall) so wird die Datenleitung durchgeschaltet (D = 0 $\Rightarrow$ Ausgang = 0, D = 1 $\Rightarrow$ Ausgang = 1). Der gewünschte Effekt ist also erreicht.
Mit Hilfe des UND-Gatters ist unsere oben gestellte Aufgabe also lösbar:

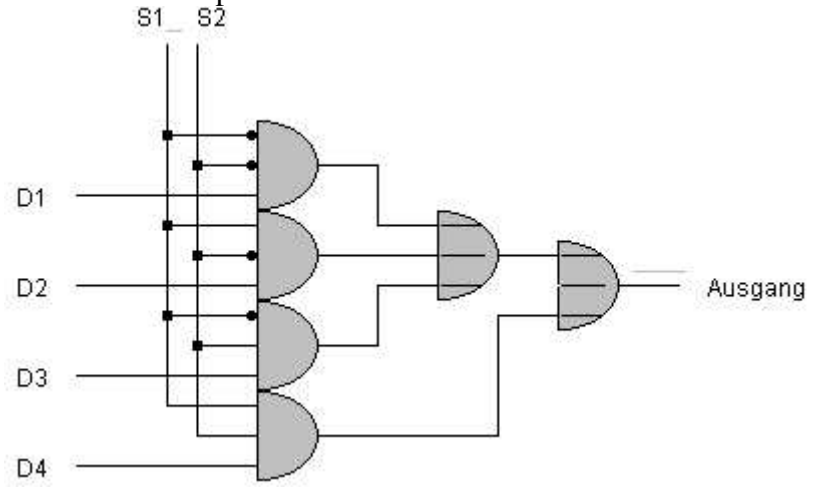


Mit Hilfe des Negationspunktes am oberen UND-Gatter wird bei S=0 die Datenleitung D1 durchgeschaltet. Ist dagegen S=1, so wird nur die Datenleitung D2 durchgeschaltet. Perfekt. Man nennt diese Schaltung einen 1-aus-2-Multiplexer (MUX).

Aufgabe 22: Baue eine Schaltung zum 1-aus-4-Multiplexer. Dieses ist eine Schaltung, welche es ermöglicht, mit zwei Steuerleitungen (S1 und S2) eine von vier Datenleitungen (D1 bis D4) auszuwählen. Schau dir zum Verständnis das Bild rechts an.



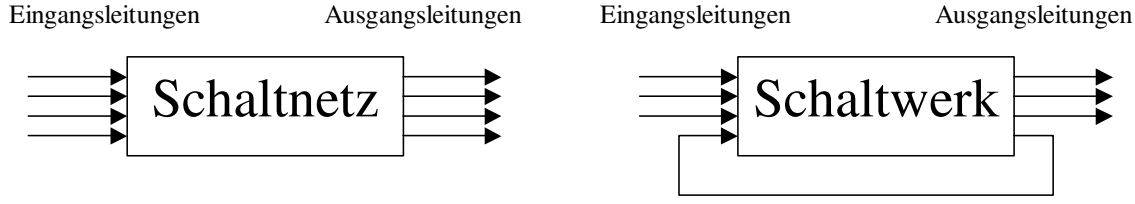
So, ich hoffe, du hast die Lösung herausbekommen. Wenn nicht, dann versuche wenigstens, die Schaltung zum 1-zu-4-Multiplexer nachzuvollziehen.



So, fassen wir noch mal kurz zusammen, was wir bisher gelernt haben. Du weißt jetzt,

- wie die Grundschaltelemente (UND-Gatter, ODER-Gatter, NOT-Gatter und XOR-Gatter) arbeiten und kennst deren Schaltsymbole,
- was eine Schalttabelle ist, wofür sie eingesetzt wird und wie man eine solche Schalttabelle optimiert,
- wie die Schaltungen von Halb- und Volladdierer aussehen und wie man mit diesen Schaltelementen einen Paralleladdierer, eine Zweierkomplementschaltung und einen Parallelsubtrahierer erzeugen kann,
- wie man mit Hilfe des XOR-Gatters (mit Hilfe einer Steuerleitung) umschaltbare Rechenwerke erzeugen kann,
- was Koppeldioden sind und wie man mit Hilfe dieser Koppeldioden komplexere Schaltnetze aufbauen kann (8-zu-3-Decoder, Siebensegmentanzeige, ROM, etc.)
- wie man schaltbare Schaltnetze mit Hilfe von UND- und XOR-Gattern realisieren kann.

Wie du siehst, die **Schaltnetze** können schon eine ganze menge. Allerdings haben die Schaltnetze einen großen Nachteil. Um diesen besser zu verstehen, wollen wir uns zuerst noch einmal in die Programmiersprache VisualBasic zurückversetzen. Dort gab es eine WENN-DANN-SONST-Anweisung, welche bewirkte, dass der Computer abhängig von einem zuvor berechneten Wert unterschiedliche Befehle auszuführen hatte. Dabei sind die Berechnung eines Wertes und die unterschiedlichen Befehle durch Schaltung realisierbar. D. h. im Klartext, dass der Computer eine Schaltung ist, welche sich selbst beeinflusst. Bisher haben wir allerdings nur Schaltnetze entwickelt, welche zwar steuerbar (Paralleladdierer/-subtrahierer) waren, welche sich jedoch nicht selbst beeinflussten. Selbst beeinflussende Schaltungen nennt man **Schaltwerke**. Diesen Unterschied kann man grafisch wie folgt darstellen:



Genau mit diesen Schaltwerken wollen wir uns jetzt beschäftigen.

3 Schaltwerke

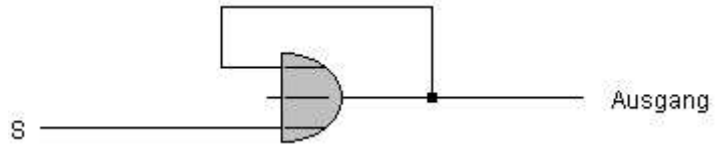
Kommen wir noch einmal kurz auf die WENN-DANN-SONST-Anweisung von Visual-Basic zurück. Um überhaupt eine Verzweigung im Programm vornehmen zu können, ist es nötig, eine Bedingung dafür aufzustellen. Solche Bedingungen lauteten z. B. `a <> 0` oder `a > b`. D. h., die Verzweigung findet mit Hilfe eines zuvor (z. B. in die Variable a) gespeicherten Wertes statt. Genau diese Speicherung eines Wertes wollen wir uns als erstes vornehmen, denn diese ist grundlegend für alle Arten von Schaltwerken.

3.1 Das Flipflop

Wir wollen eine Schaltung bauen, welche sich eine einmal eingegebene Zahl „merkt“, d. h. speichert. Im konkreten Fall soll dies eine Schaltung sein, welche sich eine einmal eingegebene „eins“ merkt. Jetzt könnte man ja sagen, dass eine einfache Leitung sich auch die eingegebene „Zahl“ merkt:

Leitung mit „null“ Leitung mit „eins“

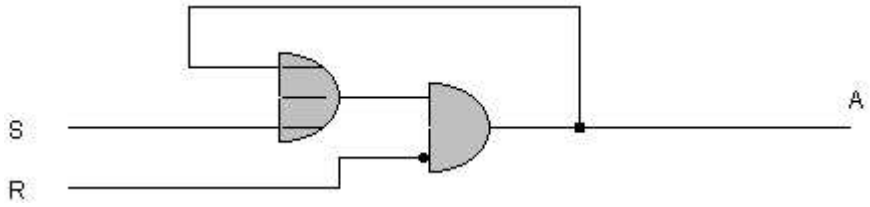
Das Problem dabei ist allerdings, dass die „eins“ sofort verloren geht, wenn man die Leitung wieder auf „null“ setzt. Aber genau das soll verhindert werden. Eine einmal eingestellt „eins“ soll bestehen bleiben, egal ob die Eingangsleitung wieder zurückgesetzt wird oder nicht. Was wir brauchen ist also eine Art „Rückkopplung“, welche uns mitteilt, ob schon einmal eine „eins“ vorlag oder nicht:



Die Rückkopplung wird „eins“, sobald die Eingangsleitung S einmal gesetzt wurde. Danach liegt am ODER-Gatter immer Strom an, so dass der Ausgang auch immer auf „eins“ bleibt. Die Schaltung „merkt“ sich also eine einmal gesetzte „eins“ an der Eingangsleitung.
Der Nachteil dieser Schaltung ist jedoch, dass die Ausgangsleitung nicht wieder verändert werden kann. Es wäre jedoch sinnvoll, eine Möglichkeit zu schaffen, die Ausgangsleitung wieder zurückzusetzen. Wir brauchen also eine zweite Eingangsleitung (R), welche den zuvor „gemerkten“ Wert wieder löscht. Konkret heißt das, das Ausgangssignal muss mit Hilfe dieser Eingangsleitung „unterbrochen“ werden. Im letzten Kapitel hast du bei der Schaltung zum Multiplexer eine solche Möglichkeit kennen gelernt. Die Unterbrechung war mit Hilfe eines UND-Gatters möglich. Ist die Steuerleitung am UND-Gatter gesetzt, so wird das Eingangssignal durchgeschaltet, ansonsten wird das Eingangssignal unterbrochen.

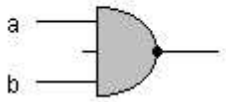
Aufgabe 23: Vervollständige die unten abgebildete Schaltung insofern, dass mit Hilfe des S(etz-Eingangs) die A(usgangsleitung) angeschaltet und mit Hilfe des R(ücksetz-Eingangs) wieder ausgeschaltet werden kann.

Na? Hast du es herausbekommen? Ein mögliche Lösung ist die folgende:



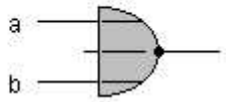
In der Schaltungslehre ist es üblich, möglichst alle Schaltungen mit Hilfe von NAND- und NOR-Gattern aufzubauen. Diese Schaltelemente arbeiten fast genauso wie die AND- und OR-Gatter, allerdings mit dem feinen Unterschied, dass sie genau das umgekehrte Ausgangssignal liefern:

NAND-Gatter:



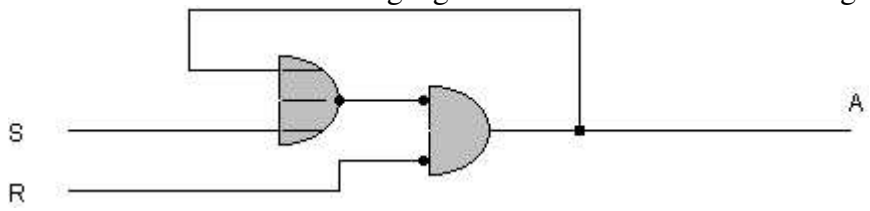
a	b	Ausgang
0	0	1
0	1	1
1	0	1
1	1	0

NOR-Gatter:

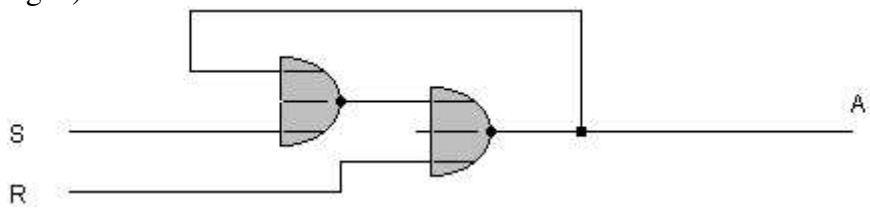


a	b	Ausgang
0	0	1
0	1	0
1	0	0
1	1	0

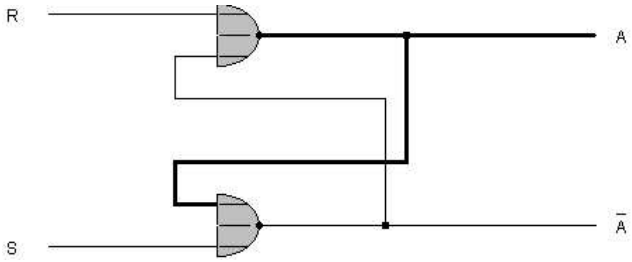
Unser Ziel ist es nun, die obige Schaltung nur durch NOR-Gatter aufzubauen. Das erste obere OR-Gatter macht dabei wenig Schwierigkeiten. Hängen wir hinten einfach einen Negationspunkt an und machen ihn am Eingang des AND-Gatters wieder rückgängig:



Jetzt aber das AND-Gatter. Wie könnte ein AND-Gatter durch ein NOR-Gatter ersetzt werden? Schaue dir dazu noch mal die obige Tabelle des NOR-Gatters an. Wenn beide Eingänge „null“ sind, so ist der Eingang „eins“. In allen anderen Fällen ist er „null“. Aber genau das ist ja unser Ziel. Am Ausgang des AND-Gatter liegt genau dann Strom an, wenn beide Eingangsleitungen „null“ sind (beachte die Negationspunkte bei den Eingangsleitungen). Also könne wir das AND-Gatter (mit seinen Negationspunkten an den Eingangsleitungen) durch ein NOR Gatter ersetzen:



Wenn wir jetzt noch die Bauteile etwas schöner anordnen, so erhalten wir genau die Schaltung eines RS-Flipflops (siehe nächste Seite). Flipflop heißt soviel wie: „umschaltbar von 0 auf 1 und zurücksetzbar von 1 auf 0“ wobei das setzen und rücksetzen mit den Eingangsleitungen R und S geschieht.

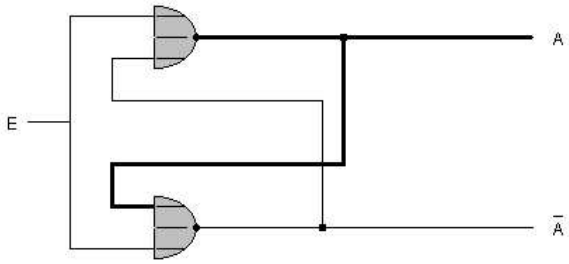


Schaltung eines RS-Flipflops aus NOR-Gattern

Die zweite Ausgangsleitung liefert immer genau das entgegengesetzte Signal der Ausgangsleitung. Deshalb heißt diese Ausgangssignal auch „Invertierter Ausgang“ (not(A) oder auch kurz $\overline{A}$). Dieses Ausgangssignal spielt zwar im Moment noch keine Rolle, jedoch werden wir später Anwendungsmöglichkeiten dieses Ausgangssignals kennen lernen.

Aufgabe 24:
a) Erstelle in LOCAD die Schaltung des RS-Flipflops und überprüfe seine Wirkungsweise.
b) Lege an beiden Eingangsleitungen Strom an (R=1 und S=1). Was passiert?
c) Schalte nun beide Eingangsleitungen aus (R=0 und S=0). Was passiert?
d) Schalte nun wieder beide Eingangsleitungen an. Schalte danach wieder beide Eingangsleitungen aus, allerdings nun in der anderen Reihenfolge (S=0 und R=0). Wie unterscheidet sich das Ausgangssignal von dem Ergebnis aus Teilaufgabe c)?

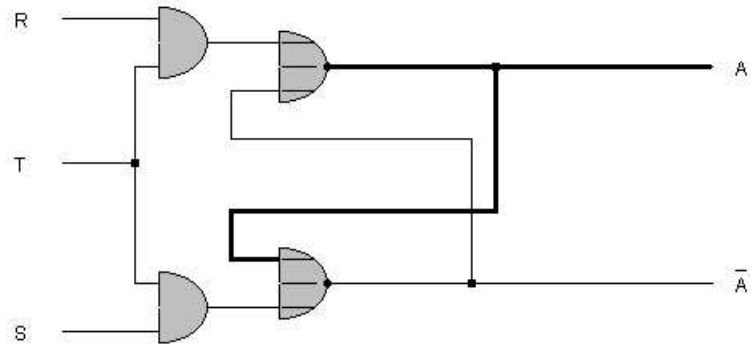
Aufgabe 25: Ändere die Schaltung des RS-Flipflops wie unten gezeigt ab. Mit dieser Schaltung kann man das gleichzeitige An- bzw. Abschalten beider Eingangsleitungen simulieren. Beschreibe was passiert!



Ich denke, du hast die Problematik erkannt. Sind beide Eingänge auf „eins“ geschaltet, so ist das eigentlich noch unproblematisch. Allerdings ist der Zustand der Ausgangsleitung nicht vorhersehbar, wenn beide Eingangsleitungen wieder ausgeschaltet werden. In Aufgabe 24 hast du gesehen, dass das Ausgangssignal davon abhängig ist, in welcher Reihenfolge (erst R=0 und dann S=0 bzw. erst S=0 und dann R=0) die Eingangsleitungen abgeschaltet werden. In Aufgabe 25 hast du gesehen, dass ein gleichzeitiges Abschalten zu einem ständigen Wechsel der Ausgangsleitungen führt. Dies zeigt besonders deutlich die Unberechenbarkeit der Ausgangsleitungen.
Allerdings ist das alles nur passiert, weil wir einmal beide Eingangsleitungen angeschaltet haben (R=1 und S=1). Und genau das wollen wir aus den oben genannten Gründen in Zukunft verbieten. Insgesamt ergibt sich dann folgende Zustandstabelle:

R	S	A_n	$\overline{A}_{nachher}$	
0	0	A_v	$\overline{A}_{vorher}$	speichern
0	1	1	0	setzen
1	0	0	1	rücksetzen
1	1	––	––	verboten!!!

Dieses soeben entwickelte Flipflop zeigt das grundlegende Prinzip für alle komplexeren Flipflop-Typen und wird uns bei der Entwicklung verbesserter Flipflops begleiten. Das Prinzip, dass ein zeitlich begrenztes Signal an einem Eingang den Zustand der Ausgänge dauerhaft verändert, ermöglicht es uns, ein Bit zu speichern. Eine 8-Bit-Binärzahl benötigt also zur Speicherung genau 8 solcher Flipflops. Versuche dir einmal vorzustellen, welche Schaltung wohl hinter einem 128 Megabyte-RAM-Baustein steckt. Über 134 Millionen Flipflops werden dafür benötigt. Das ist schon eine ganze Menge.
Nun aber zu Verbesserungen des Grundflipflops. Oftmals müssen beim Speichern von Informationen mehrere Flipflops gleichzeitig gesetzt werden. So muss z. B. das Speichern einer 8-Bit-Binärzahl in acht Flipflops gleichzeitig geschehen. Dazu ist es allerdings notwendig, die Flipflops zu synchronisieren, d. h. die am Eingang vorliegende Information muss von allen Flipflops gleichzeitig übernommen werden. Das Signal dafür wird mit Hilfe einer Taktleitung gegeben. Wird die Taktleitung auf „eins“ gesetzt, so werden die anliegenden Informationen von allen Flipflops gleichzeitig übernommen. Steht die Taktleitung auf „null“, so werden die Eingangssignale (Setzen und Rücksetzen) mit Hilfe von Torschaltungen (AND-Gatter) unterbrochen:



Schaltung eines Getakteten RS-Flipflops (RS-Auffang-Flipflop)

Die Zustände der Ausgänge hängen damit von den jetzt drei Eingangsleitungen R, S und T ab:

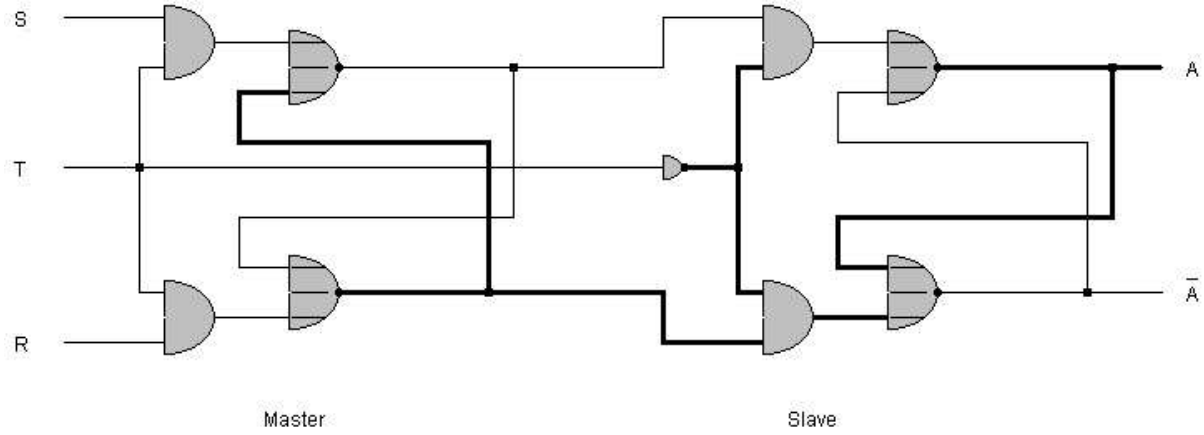
R	S	T	$A_{nachher}$	
b	b	0	A_{vorher}	b = beliebig
0	0	1	A_{vorher}	speichern
0	1	1	1	setzen
1	0	1	0	rücksetzen
1	1	1	––	verboten!!!

Aufgabe 26: Baue die Schaltung des RS-Auffang-Flipflops in LOCAD nach. Teste seine Funktionsweise und bestätige damit die Zustandstabelle.

Oftmals müssen zwei Speicherelemente so hintereinandergeschaltet werden, dass das nachfolgende die gespeicherte Information des vorhergehenden übernimmt, während dieses eine neue Eingangsinformation aufnehmen muss. Ein denkbare Möglichkeit wäre, zwei getaktete RS-Flipflops direkt hintereinander zu schalten.

Aufgab 27: Erstelle eine Schaltung mit zwei direkt hintereinandergeschalteten RS-Auffangflipflops, wobei die Eingänge des zweiten Flipflops die Ausgänge des ersten Flipflops sind. Beobachte das Verhalten.

Du wirst den Mangel erkannt haben! Weil beide Flipflops den gleichen Takt haben, wird die anliegende Information direkt bis zum Ausgang des zweiten Flipflops durchgeschaltet. Das wollten wir aber gerade verhindern. Besser wäre es, wenn beide Flipflops mit unterschiedlichem Takt oder besser mit genau entgegengesetztem Takt arbeiten würden. Wenn der Takt aus ist, so sollen die Informationen des ersten Flipflops in das zweite Flipflop übernommen werden, die Aufnahme in das erste Flipflop ist dabei gesperrt. Wenn der Takt an ist, so soll das zweite Flipflop gesperrt sein und das erste Flipflop soll die Informationen der Eingänge (R, S) übernehmen. Das sogenannte Master-Slave-Flipflop enthält genau diese zwei gegeneinander getakteten Flipflops. Liegt der Takt auf 0, so ist das Master-Flipflop gesperrt. Das Slave-Flipflop hat dann den Takt 1 und übernimmt die Werte des Master-Flipflops. Beim Wechsel auf Takt 1 werden die Tore zum Master-Flipflop geöffnet und die anliegenden Eingangsleitungen werden übernommen. Das zugehörige Schaltbild sieht wie folgt aus:

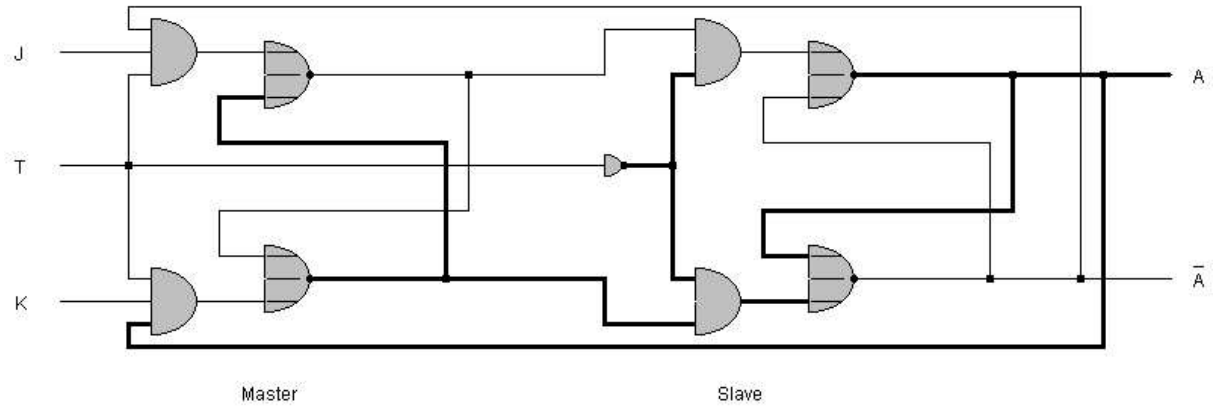


Schaltung des RS-Master-Slave-Flipflops

Durch diese Anordnung zweier Flipflops ist es möglich geworden, während der Ausgabe von Informationen durch das Slave-Flipflop im Master-Flipflop bereits neue Informationen aufzunehmen, ohne dass sich dadurch die Information am Ausgang ändert. Diese Eigenschaft ist für den Aufbau von Schieberegistern, welche Informationen von Flipflop zu Flipflop schieben, von besonderer Bedeutung.

Aufgabe 28: Baue die Schaltung des RS-Master-Slave-Flipflops mit LOCAD nach. Überprüfe anhand der Schaltung, dass an den Eingangsleitungen anliegende Informationen erst bei einem Taktwechsel $0 \rightarrow 1 \rightarrow 0$ am Ausgang anliegen.

Leider ist auch dieses Flipflop immer noch anfällig für die Eingangsbelegung R=1 und S=1. Teste dies an deiner in Aufgabe 28 aufgebauten Schaltung. Wie du siehst entsteht das Problem des ständigen Wechsels allerdings erst bei negativer Taktflanke, d. h. bei Takt 0. Mit der folgenden Änderung kann dieser Nachteil auch vermieden werden:
Bisher wurden die Eingangsleitungen mit Hilfe eines Tores (UND-Gatter) zurückgehalten. Dieses Tor wurde mit Hilfe der Taktleitung geöffnet bzw. geschlossen. Wenn wir jetzt zusätzlich noch den gegenüberliegenden Ausgang in die Tore einführen, so bewirkt die Eingabe von R=1 und S=1 lediglich einmal einen Zustandswechsel. Danach ist nämlich entweder die vom Ausgang A oder die vom Ausgang $\overline{A}$ zurückgeführte Leitung „null“, so dass das entsprechende Tor geschlossen wird. Die folgende Schaltung veranschaulicht diese Idee:



Schaltung des JK-Master-Slave-Flipflops

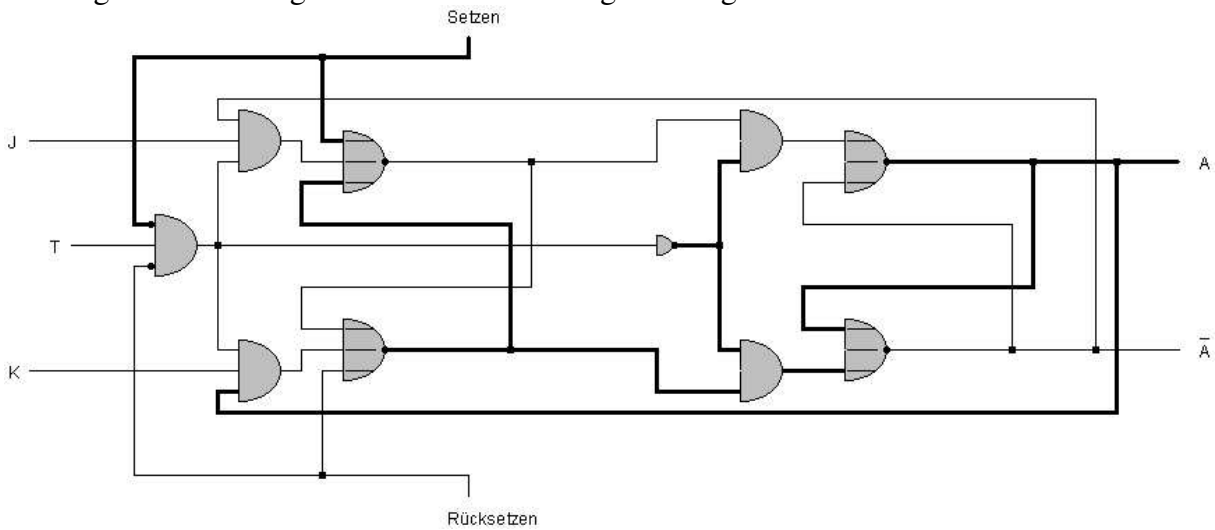
Die zugehörige Zustandstabelle zeigt, dass jetzt alle Eingangskombinationen gültig sind:

J	K	$A_{nachher}$	
0	0	A_{vorher}	speichern
0	1	0	rücksetzen
1	0	1	setzen
1	1	$\overline{A}_{vorher}$	**wechseln!**

dabei passieren die Aktionen „setzen“, „rücksetzen“ und „wechseln“ erst bei negativer Taktflanke (Takt 0).

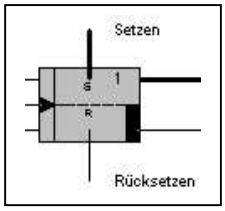
Aufgabe 29: Baue mit LOCAD das JK-Master-Slave-Flipflop auf und mache dir die Funktionsweise klar. Versuche zu verstehen, warum die Belegung R=1 und S=1 nun nicht mehr zu einem ständigen Wechsel der Ausgangsleitungen führt.

Zu guter letzt wäre es noch sinnvoll, wenn unabhängig von der Taktsteuerung die Flipflops initialisiert werden könnten. Dazu baut man sich zwei weitere Leitungen (statisches Setzen und Rücksetzen) zu den Eingängen des ersten RS-Flipflops und schaltet gleichzeitig den Takt aus. Somit hat das statische Setzen und Rücksetzen Vorrang vor der getakteten Initialisierung. Die folgende Schaltung veranschaulicht die zugrundeliegende Idee:



Schaltung des JK-Master-Slave-Flipflops mit statischen Setz- und Rücksetzeingang

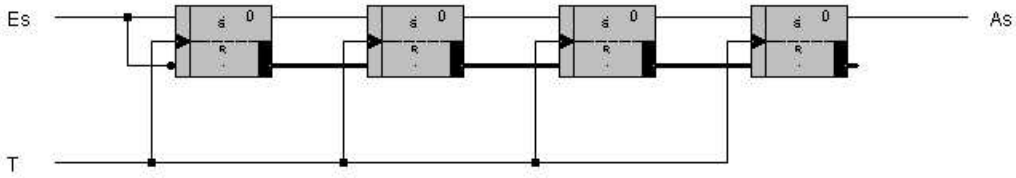
Da dieses Bauteil universell einsetzbar ist, wurde in LOCAD dafür ein eigenes Schaltsymbol erstellt, welches du rechts abgebildet siehst.



3.2 Das Schieberegister

Kommen wir nun zu dem Schieberegister, welches bereits beim RS-Master-Slave-Flipflop angesprochen wurde. Die Speicherung von Zahlen geschieht mittels mehrerer (in der Regel 8, 16 oder 32) Flipflops, welche in einer Reihe hintereinandergeschaltet sind. Man nennt eine solche Reihe von Flipflops **Register**. Sind die Flipflops so miteinander verbunden, dass mit jedem Taktimpuls eine Verschiebung der Informationen um eine Stelle nach links bzw. rechts erfolgt, so liegt ein sogenanntes **Schieberegister** vor.
Die besondere Eigenschaft der Flipflops ist, dass die Ausgänge erst nach einem Taktwechsel $0 \rightarrow 1 \rightarrow 0$ belegt werden. Somit ist es möglich, Informationen von einem Flipflop zu einem anderen Flipflop zu schieben. Die Ausgänge des ersten Flipflops bilden dabei die Eingänge des zweiten Flipflops. Dessen Ausgänge können nun wieder als Eingänge für ein drittes Flipflop dienen, usw..

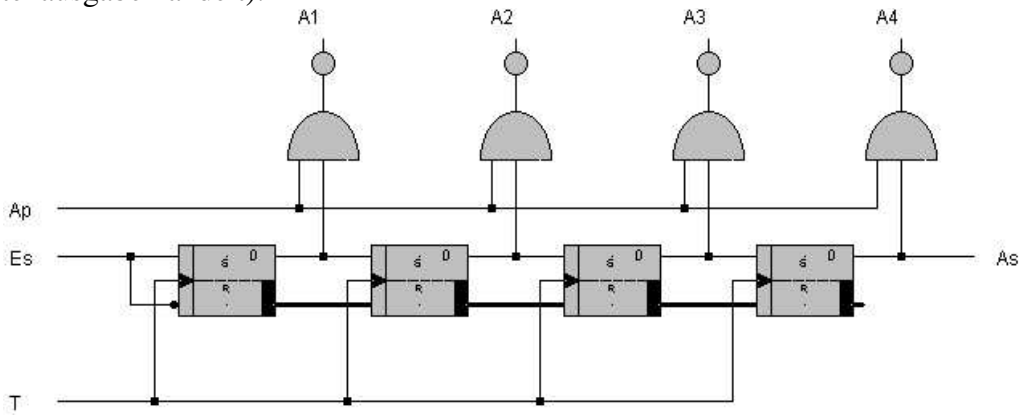
Aufgabe 30: Baue die folgende Schaltung in LOCAD nach und erläutere die Bedeutung der Leitungen $\mathbf{E}_S$ und $\mathbf{A}_S$. Wie kann man z. B. die Binärzahl 1011_2 mit 4 Taktimpulsen in das Schieberegister einschreiben?



Prinzipiell gibt es zwei Arten, die Ausgabe der im Schieberegister gespeicherten Informationen auszugeben:

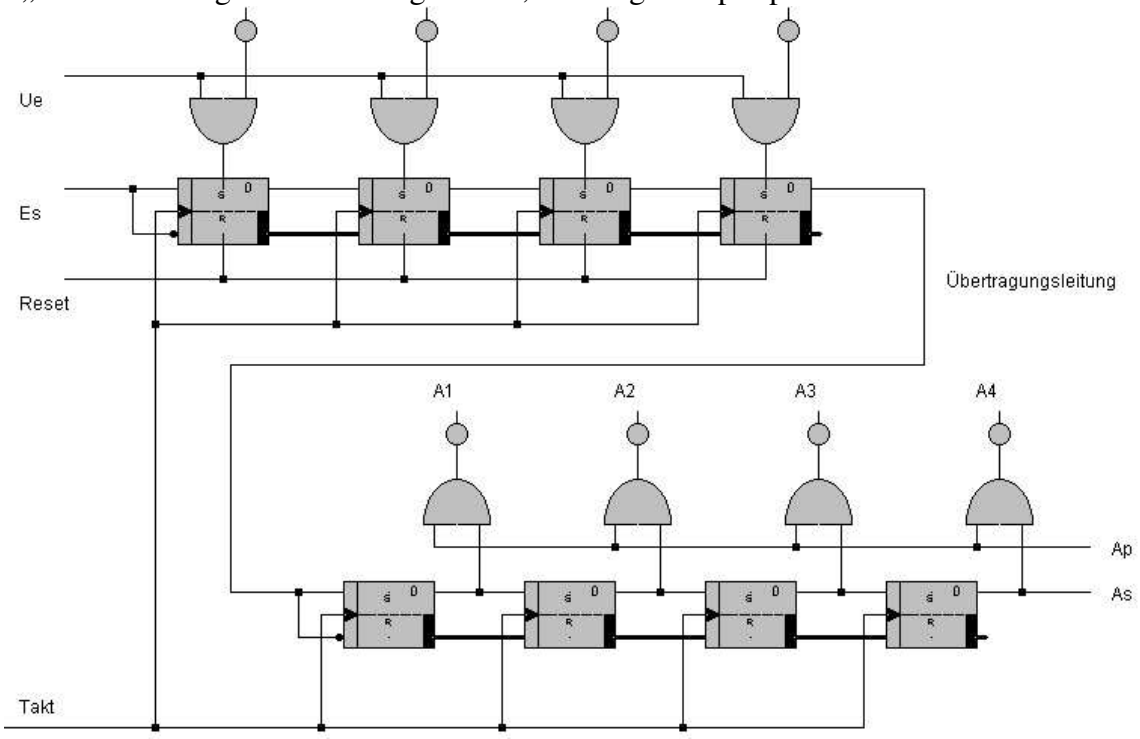
(1) serielle Ausgabe: Die Ausgabe der Flipflop-Inhalte erfolgte in diesem Beispiel seriell (der Index $_S$ an den Leitungen A_S und E_S zeigt dies an), d. h. mit jedem Taktimpuls wurde eine Ziffer der Binärzahl an der Ausgangsleitung ausgegeben. Diese Ausgangsleitung kann nun abgegriffen werden und weiter verarbeitet werden. Z. B. wäre es möglich, mit Hilfe der einen Ausgangsleitung den Inhalt dieses Schieberegisters in ein weiteres Schieberegister zu kopieren.

(2) parallele Ausgabe: Eine andere Möglichkeit besteht darin, die Informationen der einzelnen Flipflops gleichzeitig, also parallel auszugeben. Dazu bräuchte man eine Steuerleitung, welche die gleichzeitige Ausgabe ermöglicht. Dazu werden die Ausgänge aller Flipflops über Tore (welche mit der eben genannten Steuerleitung geöffnet werden) auf die Ausgabeleitungen durchgeschaltet. Die folgende Schaltung zeigt dieses Verfahren (hier bedeutet der Index $_p$ bei A_p, dass es sich um eine Steuerleitung zur parallelen Datenausgabe handelt):

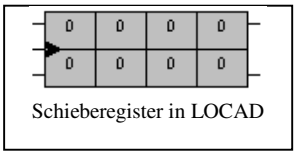


Aufgabe 31: **a)** Baue die Schaltung in LOCAD nach und teste deren Funktion.
b) Entwickle eine Schaltung für die parallele Daten**eingabe**. Verwende dazu die statischen „Setz“-Eingänge der Flipflops.

Wie bereits erwähnt ist eine Verarbeitung der Daten an beiden Ausgängen (A_P und A_S) möglich. Zur reinen Ausgabe mit Hilfe von Lämpchen oder einer LED-Anzeige bietet sich der parallele Ausgang an. Schließlich weiß man ja, dass genau vier Bits für eine 4-Bit-Zahl auszugeben sind. Anders sieht es aus, wenn Daten kopiert werden müssen. Wenn du z. B. an eine Programmiersprache denkst, so ist es mit einer Zuweisung möglich, dass entweder nur eine einzige Zahl (i. A. 16 Bits) oder eine ganze Zeichenkette (Länge der Zeichenkette mal 8 Bits) zugewiesen werden muss.
Eine Zuweisung zu tätigen bedeutet auf Maschinenebene, eine bestimmte Anzahl von Bytes von einem Register in ein anderes Register zu schieben. Jetzt ist es kaum möglich, den Verschiebevorgang parallel durchzuführen, denn sonst benötigte man alleine für die Zuweisung des Textes „Hallo du da!“ 12 mal 8 = 96 Datenleitungen. Deshalb werden viele Kopier- und Verschiebevorgänge mit Hilfe des seriellen Ausgangs (A_S) durchgeführt.
Die folgende Schaltung simuliert einen solchen Verschiebevorgang. Der Benutzer kann an den parallelen Dateneingängen E1 bis E4 eine 4-Bit-Binärzahl angeben, welche mit der Datenübernahmeleitung (Ue) in die vier Flipflops übernommen wird – dies ist übrigens die Lösung zu Aufgabe 30.
Mit vier Taktimpulsen kann nun der Inhalt der ersten vier Flipflops in die vier „Ausgabe“-Flipflops geschoben werden. Hieran erkennt man deutlich das Prinzip der seriellen Datenübertragung, d. h. der Datenübertragung über nur eine einzige Leitung.
Mit Hilfe der parallelen Ausgabeleitung (A_P) kann nun der Inhalt der unteren vier Flipflops parallel auf die Ausgabe-Leuchtdioden (A1 bis A4) durchgeschaltet werden.
Die „Reset“-Leitung bietet die Möglichkeit, die Eingabeflipflops auf 0 zurückzusetzen.



Aufgabe 32: Baue die Schaltung mit LOCAD nach und Überzeuge dich von deren Korrektheit.

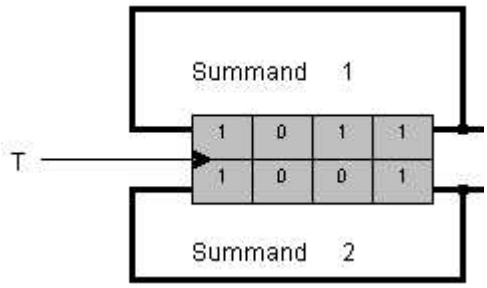


Schieberegister in LOCAD

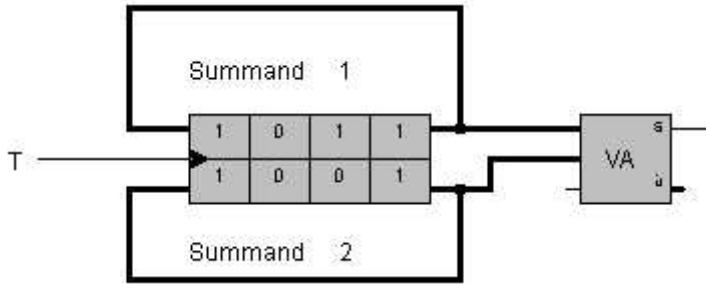
LOCAD bietet eine vorgefertigte Schaltung für dieses Schieberegister. Du findest es unter *Bauteile|Schieberegister*, welches in LOCAD wie rechts dargestellt aussieht. Häufig wird dieses Bauteil zur Speicherung zweier 4-Bit-Binärzahlen benötigt – es kann allerdings mit einigen Hilfsleitungen auch für die obige Schaltung verwendet werden. Probiere das einmal aus!

3.3 Serielle Rechenwerke

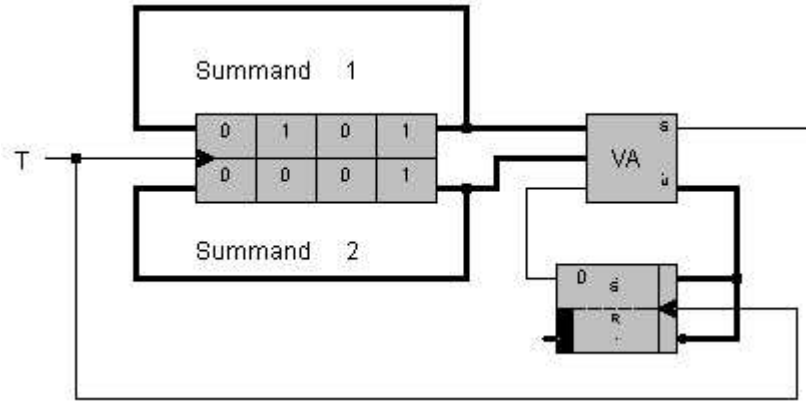
Kommen wir nun zu wirklich interessanten Anwendungen des Schieberegisters. Bisher haben wir schon eine Schaltung kennen gelernt, welche zwei 4-Bit-Binärzahlen addierte, den sogenannten Paralleladdierer. Allerdings waren dafür ein Halb- und 3 Volladdierer notwendig. Mit Hilfe eines Schieberegisters ist es nun möglich, ein Schaltwerk für die Addition zweier 4-Bit-Binärzahlen aufzubauen, welches mit nur einem Volladdierer auskommt. Man nennt dieses Schaltwerk das Serienaddierwerks. Wie der Name schon sagt, wird hierbei die Datenübertragung seriell statt parallel vorgenommen.
Wollen wir erst einmal sammeln, was wir für die Schaltung brauchen. Als erstes wäre da ein Schieberegister, welches die beiden 4-Bit-Binärzahlen aufnimmt. diese sollen durch vier Taktimpulse Bit für Bit addiert werden. Damit die beiden Zahlen jedoch nicht verloren gehen, wollen wir ein nach rechts rausgeschobenes Bit wieder links einfügen, so dass nach vier Taktimpulsen die Zahl einmal um sich selbst rotiert wurde.



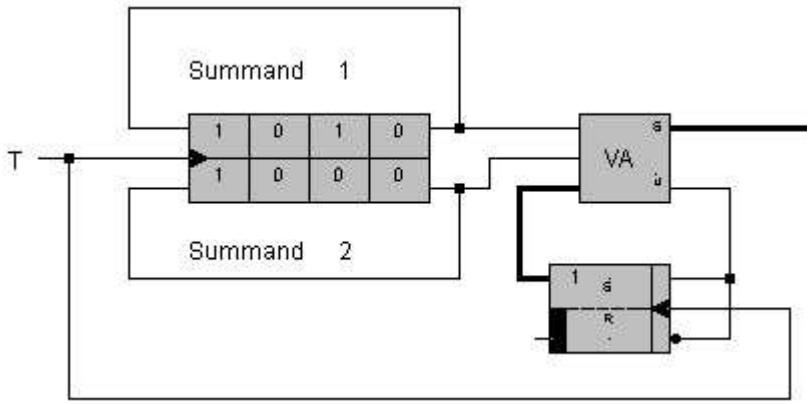
Die Frage ist jetzt aber, was machen wir mit den beiden Ausgängen, an denen nach jedem Taktimpuls die nächsten beiden zu addierenden Bits anliegen.
Erinnern wir uns noch mal an den Paralleladdierer. Dort wurde die Addition zweier Bits durch einen Volladdierer erreicht. Zusätzlich wurde der eventuelle Übertrag aus der vorangehenden Addition eingeleitet. Genau das wollen wir hier auch machen:



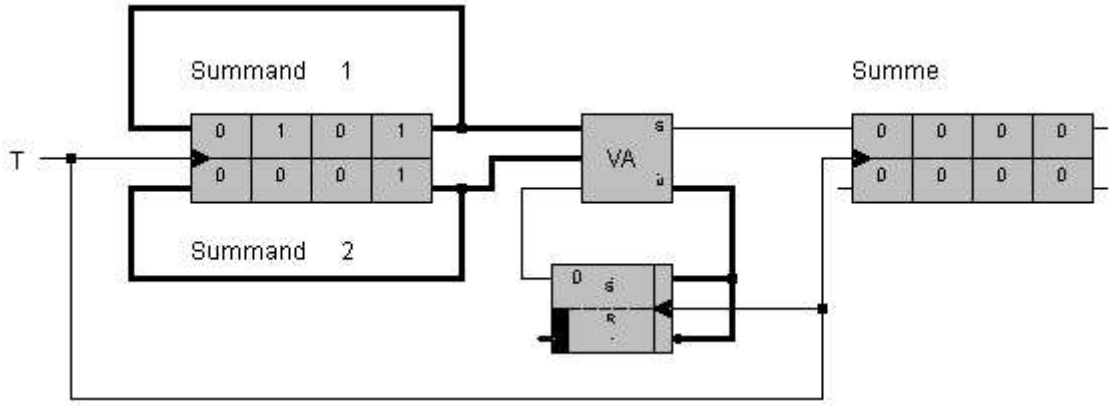
Die Frage ist nur, woher nehmen wir den eventuellen Übertrag der letzten beiden Bits. In der obigen Schaltung müsste der Übertrag (Ü des Volladdierers ist 1) zwischengespeichert werden, bevor er bei der Addition der nächsten beiden Bits zur Geltung kommt. Die Speicherung eines Bits können wir jedoch mit Hilfe eines Flipflops erreichen:



Wie du an der Schaltung erkennen kannst, wird der Übertrag in das Flipflop eingeleitet. Jedoch wird erst beim nächsten Taktsignal der Wert übernommen. Einen Takt später sähe die Schaltung wie folgt aus:



Das Flipflop hat den Übertrag aufgenommen und gibt diesen nun bei der jetzt folgenden Addition der zweiten beiden Bits zusätzlich in den Volladdierer.
Die Frage die bleibt ist, wie zeigen wir das Ergebnis an? In der vorherigen Schaltung hast du gesehen, dass am Summenausgang (S) des Volladdierers eine 0 anlag. Dieses ist das letzte Bit des Ergebnisses. In der letzten Schaltung liegt nun eine 1 am Summenausgang an. Dies ist das zweitletzte Bit des Ergebnisses, usw.. Was wir jetzt nur noch zu tun haben, ist alle Ergebnisbits der Reihe nach in ein Schieberegister zu schieben:



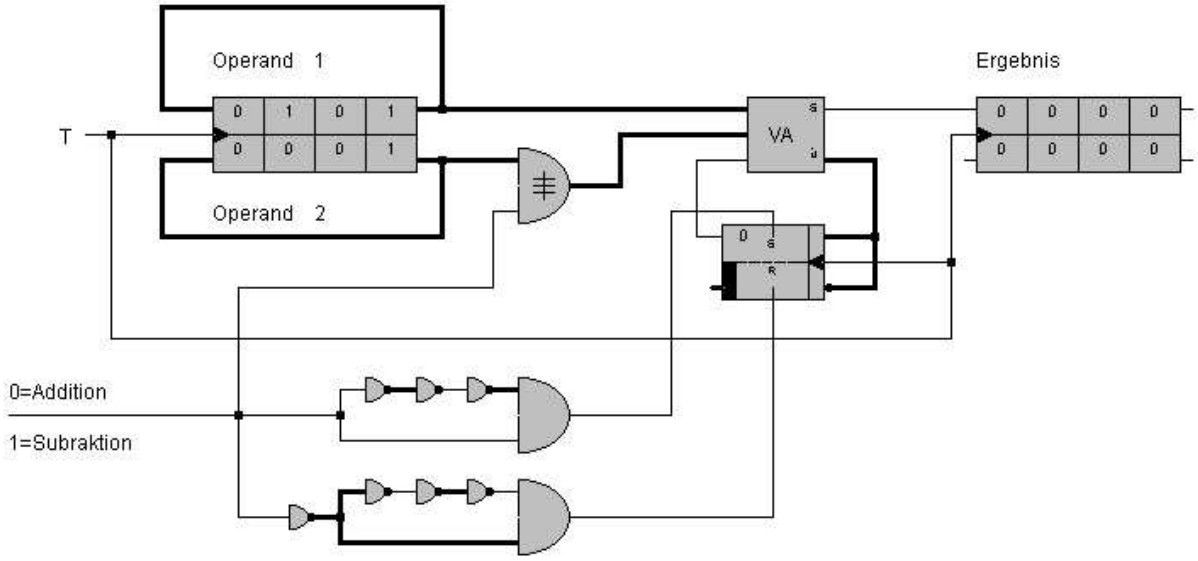
Schaltung eines Serienaddierers

Die Schaltung des **Serienaddierers** befindet sich im Ursprungszustand, d. h. vor dem ersten Taktsignal. Dementsprechend ist die Summe auch noch 0000. Das zweite Schieberegister wird bei dieser Schaltung nicht benötigt.

Aufgabe 33: **a)** Baue die Schaltung mit LOCAD nach. Teste sie, indem du vier Taktimpulse auf die Taktleitung legst. Kommt wirklich das richtige Ergebnis raus?
b) Wie müsste man die Schaltung verändern, damit zwei 4-Bit-Binärzahlen seriell **subtrahiert** werden. Ändere die Schaltung zu einem Seriensubtrahierer ab.

Ich hoffe, du hast daran gedacht, dass von der zweiten 4-Bit-Binärzahl das Zweierkomplement gebildet wird und die Zahl 1 aufaddiert wird. Konkret heißt das für die Schaltung, dass das Flipflop mit 1 vorinitialisiert werden muss und am Ausgang des Schieberegisters für den Subtrahenden ein Inverter zwischengeschaltet werden muss.
Ich möchte nicht direkt auf diese Schaltung weiter eingehen, sondern direkt eine Lösung für ein umschaltbares serielles Rechenwerk ansteuern.

Soll das Rechenwerk steuerbar sein (Addition/Subtraktion), so brauchen wir eine Steuerleitung, welche uns dies signalisiert. Beim umschaltbaren Paralleladdierer/-subtrahierer haben wir die Steuerleitung dazu genutzt, die für das Zweierkomplement hinzuzuaddierende 1 in den ersten Volladdierer mit einzuleiten. Hier müssen wir nun mit Hilfe der Steuerleitung das Flipflop vorinitialisieren.
Des weiteren müssen wir statt des Inverters hinter der zweiten Eingabezahl ein XOR-Gatter verwenden, wie wir es auch schon beim umschaltbaren Paralleladdierer/-subtrahierer gemacht haben.
Insgesamt ergibt sich folgende Schaltung:



Schaltung eines umschaltbaren seriellen Rechenwerks

Im ersten Moment wird dir die Schaltung für die Steuerleitung etwas unsinnig vorkommen. Es sieht so aus, als ob am Ausgang beider AND-Gatter stets 0 herauskommt. Durch die Inverter wird jedoch der Stromfluss soweit „verlangsamt“, dass für eine ganz kurze Zeit entweder am ersten oder am zweiten AND-Gatter Strom herauskommt. Schaltet man z. B. in der obigen Schaltung die Steuerleitung auf 1, so liegen im oberen AND-Gatter für kurze Zeit zwei „einsen“ an den Eingängen. Die Folge ist, dass durch diesen kurzen Stromimpuls der statische Setzeingang des Flipflops betätigt wird. Das Flipflop bekommt also seine 1 für das Zweierkomplement.

Aufgabe 34: Baue die Schaltung mit LOCAD auf. Überzeuge dich von deren Korrektheit. Schaue dir insbesondere die Impulsschaltung (oder auch Verzögerungsschaltung genannt) der Steuerleitung (Add/Sub) an.

Aufgabe 35: Benutze für die Takterzeugung einen Taktgeber. Du findest ihn unter *Bauteile\Taktgeber*. Wenn du nun die Schaltung einschaltest, so kannst du mit Hilfe der oben angezeigten Buttons entweder einen Einzeltakt, einen Dauertakt oder eine bestimmte Taktzahl auslösen. Wähle eine Taktfolge von 4 Taktimpulsen. Anschließend sollten die zwei Zahlen verrechnet sein.

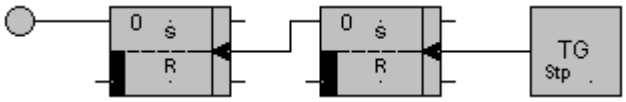
3.4 Zählschaltungen

Mit der soeben erstellten Schaltung haben wir im Prinzip das Herzstück eines jeden Rechners – wenn auch stark vereinfacht – gebaut: Man nennt dieses auch die ALU, die arithmethisch-logische-Unit (dt.: Einheit). Weitere wichtige Bausteine des Rechners müssen nun folgen, denn woher soll unser „Rechner“ die Daten zum addieren und subtrahieren bekommen? D. h., wir werden einen Speicher (den RAM: random access memory) bauen müssen, in dem die

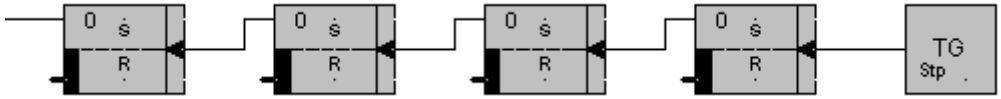
Daten abgelegt werden können. Des weiteren benötigen wir eine Schaltung, welche zählen kann, damit man immer weiß, welcher Befehl gerade ausgeführt werden soll. Und um genau diese wollen wir uns als erstes kümmern.

Wir haben bereits gesehen, dass ein JK-Master-Slave-Flipflop bei der Eingangsbelegung J=1 und K=1 den Zustand wechselt, wenn ein Taktsignal gegeben wird. Diesen Umstand machen wir uns jetzt zu Nutze.

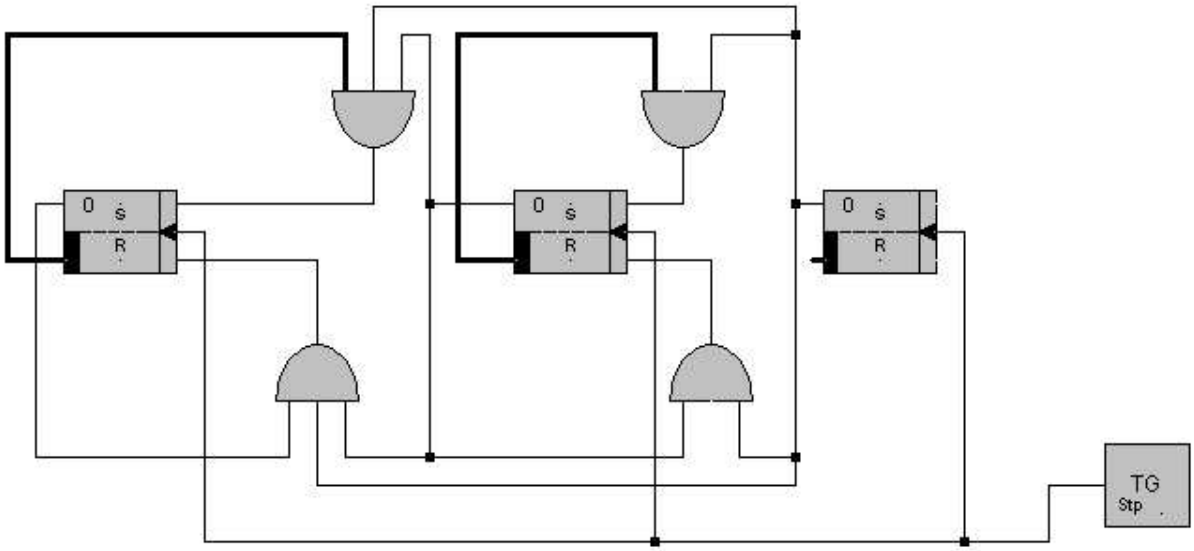
Aufgabe 36: Teste die folgende Schaltung. Wie viele Takte benötigt man, um die Lampe an zu schalten?



Aufgabe 37: Teste nun die folgende Schaltung. Wie weit kann man mit dieser Schaltung zählen?



Man nennt diese Schaltung ein Zählschaltung, genauer gesagt: eine asynchrone Zählschaltung. Asynchron bedeutet hier, dass die Flipflops nicht gleichzeitig getaktet sind. Will man erreichen, dass die Flipflops synchronisiert sind, d. h., dass alle Flipflops gleichzeitig einen Taktimpuls bekommen, so ist ein Vielfaches an Arbeit aufzuwenden. Als Beispiel soll hier ein Synchronzähler reichen, der von 0 bis 7 zählen kann:

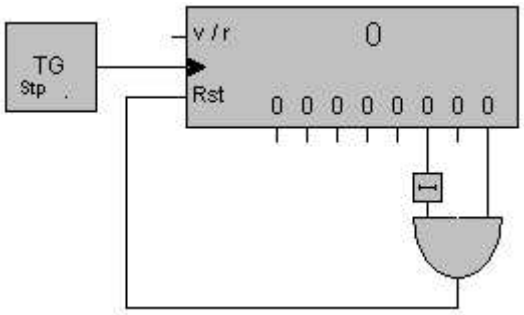


Aufgabe 38: Teste den synchronen Vorwärtszähler in Locad. Entwickle einen synchronen Vorwärtszähler, der von 0 bis 15 zählen kann.

Aufgabe 39: Erstelle einen **asynchronen** (**!** ein synchroner wäre zu kompliziert!) Rückwärtszähler, d. h., dass der Zähler von 15 bis 0 runterzählen soll. Schaffst du es anschließend auch noch, die Schaltung in einen umschaltbaren Vorwärts-Rückwärtszähler umzumodeln?

LOCAD hat bereits einen solchen Binärzähler, der wahlweise vorwärts von 0 bis 255 aber auch rückwärts zählen kann, in seinen vordefinierten Bauteilen integriert. Wenn man die

Ausgangsleitungen geschickt nutzt, so kann man diesen Zähler dafür verwenden, eine bestimmte Anzahl von Takten aus einem Dauertaktsignal auszukoppeln. Wie dies gemeint ist, zeigt die folgende Schaltung:



In dem UND-Gatter wird Strom durchgelassen, wenn die Zählschaltung auf 5 (2^2- und 2^0-Leitungen sind an) umspringt. Das Bauteil auf der 2^2-Leitung bewirkt lediglich, dass der Strom dieser Leitung etwas verzögert in das UND-Gatter eingeleitet wird. (Warum könnte dies nötig sein?). Der Rst-Eingang des Zählers bewirkt nun, dass der Zähler zurückgesetzt wird, sobald die beiden Leitungen für 2^2 und 2^0 an sind. Damit ist klar, dass der Zähler bei jedem 5. Taktimpuls wieder auf Null gesetzt wird und somit 5 Takte aus einem Dauertaktsignal auskoppeln kann.

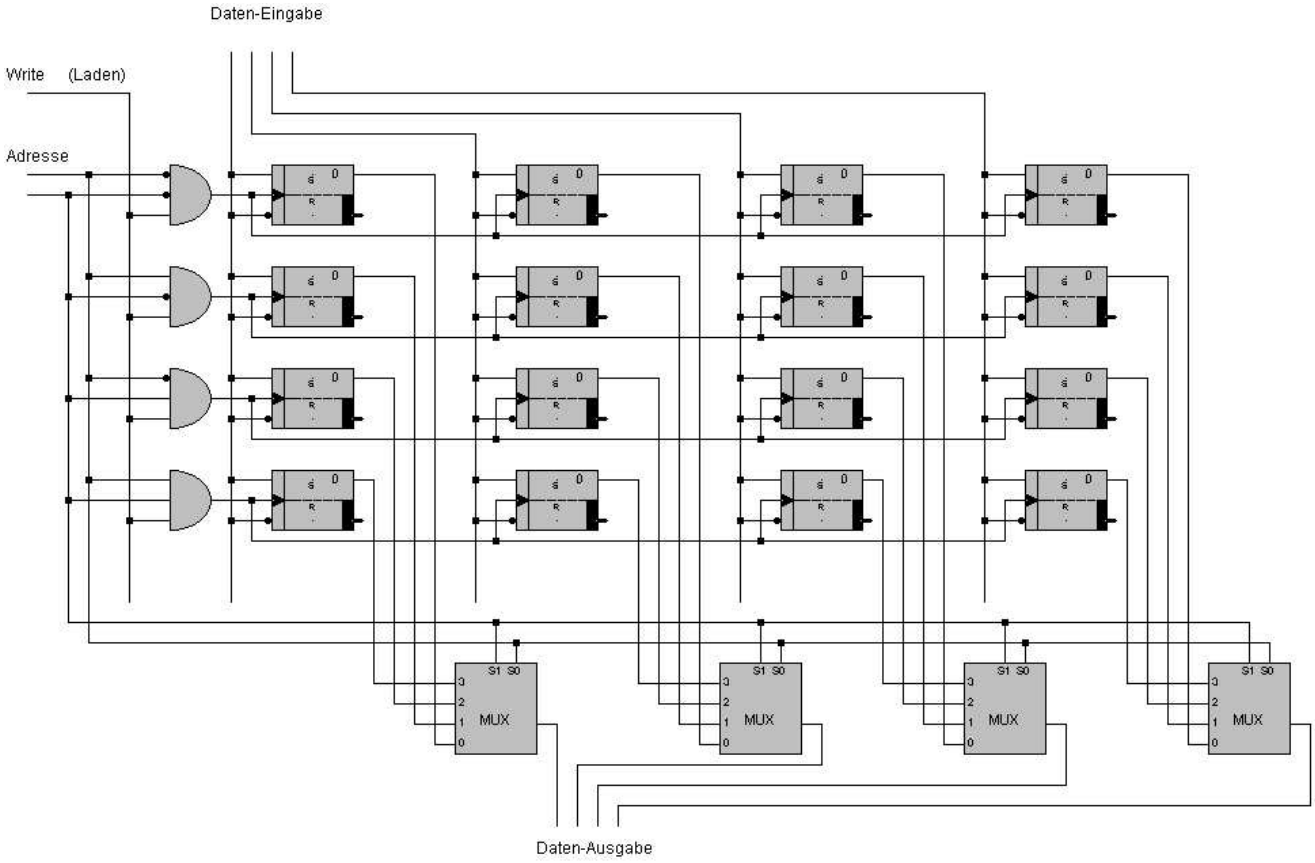
Aufgabe 40: Erstelle eine Schaltung, welche aus einem Dauertaktsignal 9 (15, 17) Takte auskoppelt. Nutze Verzögerungsglieder, um das gewünschte Verhalten zu erzielen.

3.5 Ein Modellrechner

Mit diesem Zähler kommen wir zu einem weiteren wichtigen Element eines Computers: dem Steuerwerk. Wenn wir z. B. ein Programm in Visual Basic schreiben, so muss der Computer stets wissen, welcher Befehl gerade ausgeführt werden soll. Er muss sozusagen die Befehle durchzählen (das macht unser Zähler) und je nach Befehl etwas anderes tun (das macht unser Steuerwerk). Wo allerdings findet das Steuerwerk die ganzen Befehle, die auszuführen sind? Hier kommt nun der Hauptspeicher (das sogenannte RAM) ins Spiel. Der Name kommt von der englischen Bezeichnung Random-Access-Memory. Die wichtigsten Eigenschaften eines Hauptspeichers sind, Informationen an gezielte Stellen zu speichern und von bestimmten Stellen die gespeicherten Informationen auch wieder preiszugeben. Das bedeutet, man kann die unterschiedlichen Stellen (Speicherzellen) adressieren kann.
Zur Speicherung verwenden wir wieder unsere Flipflops, wobei wir wiederum 4 Flipflops zu einer Speicherzelle zusammenfassen wollen.
Auf der nächsten Seite findest du die Schaltung eines Hauptspeichers mit 4 Speicherzellen, in die jeweils eine Zahl zwischen 0 und 15 gespeichert werden kann. Zur Erläuterung:

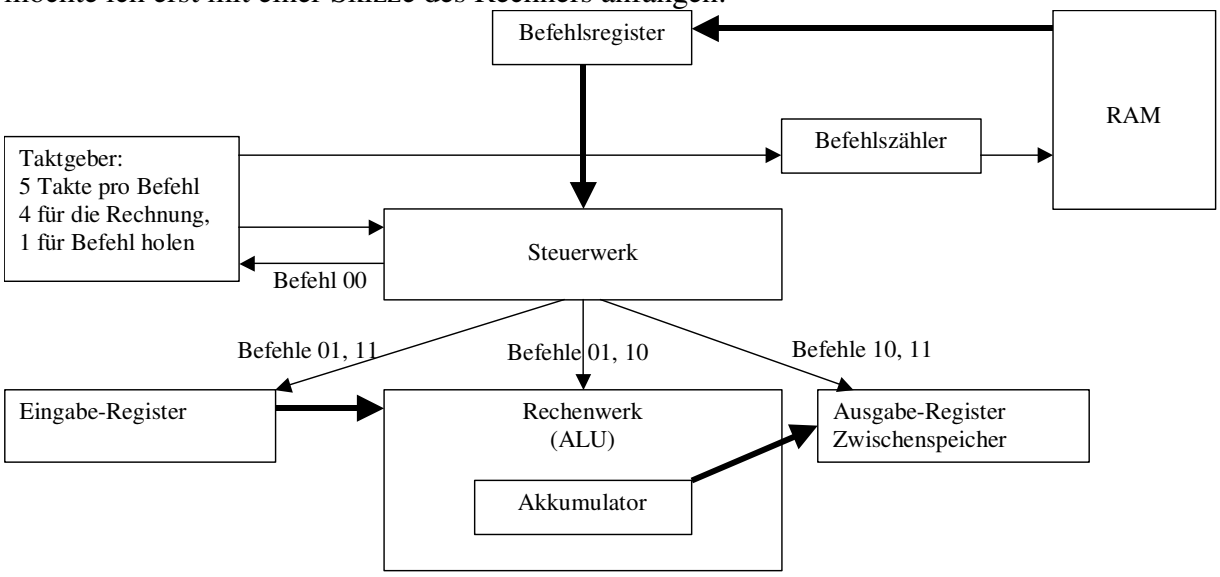
- Mit den *Adresse*-Leitungen adressiert man die Speicherzelle, mit der man gerade arbeiten will.
- Mit den *Daten-Eingabe*-Leitungen stellt man ein, welche Zahl in das RAM übernommen werden soll.
- Mit der *Write (Laden)*-Leitung übernimmt man die an der *Daten-Eingabe* anliegenden Informationen in die zuvor adressierte Speicherzelle.
- An der *Daten-Ausgabe* liegt immer die Information der aktuelle adressierten Speicherzelle an. Die Multiplexer sorgen dafür, dass die richtige Speicherzelle durchgeschaltet wird.

Im Prinzip sollte klar sein, dass dieser Hauptspeicher beliebig erweiterbar ist, sowohl was die Anzahl der Speicherplätze als auch was die Größe der zu speichernden Informationen angeht.



Aufgabe 41: Baue die Schaltung des RAMs im LOCAD nach. Teste seine Funktionsweise.

Zum Glück hat LOCAD auch hierfür ein entsprechendes Bauteil parat. Probiere doch mal aus, wie der von LOCAD zur Verfügung gestellte Speicherbaustein funktioniert. Du findest ihn auch unter dem Menüpunkt Bauteile.
Wir werden diesen RAM-Baustein später für die Speicherung unsere Programme benutzen. Aber was heißt eigentlich später? Im Prinzip haben wir alle wichtigen Komponenten eine Computers in LOCAD nachgebaut und getestet. Jetzt gilt es nur noch, diese Teile zu einem Sinnvollen Ganzen zusammenzusetzen. Um die Funktionsweise etwas besser zu verstehen, möchte ich erst mit einer Skizze des Rechners anfangen:

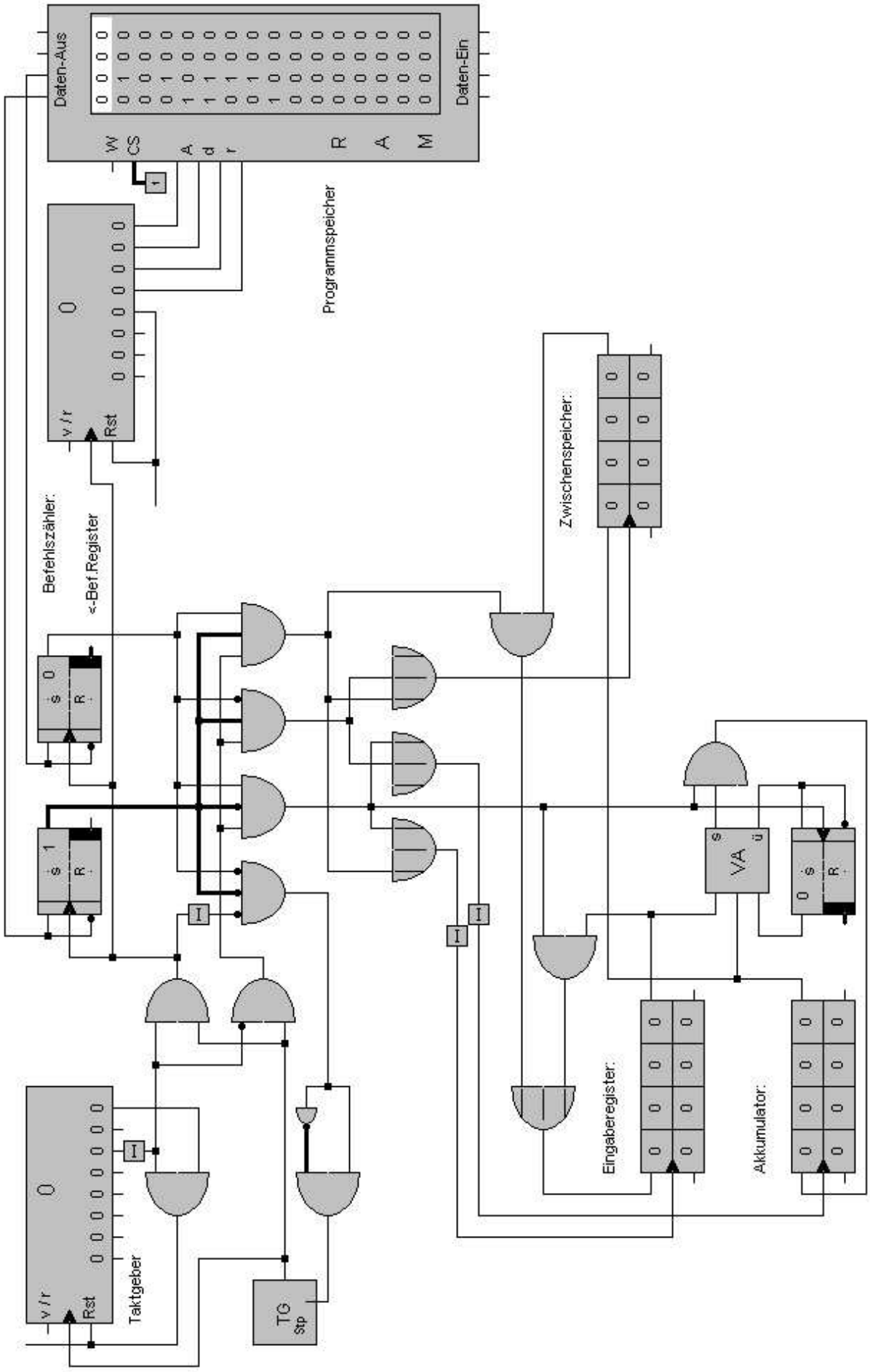


In dem Diagramm bedeuten dünne Pfeile, dass Steuersignale zwischen den Einheiten gesendet werden. Die dicken Pfeile bedeuten, dass Daten transportiert werden. Im Einzelnen passiert folgendes:
Der Taktgeber koppelt aus einem Dauertaktsignal jeweils 5 Taktimpulse aus.
Die ersten vier Takte sind dazu da, die Zahlen im Rechenwerk zu verrechnen (4 Bit-Binärzahlen). Der jeweils 5. Takt ist dafür da, den Befehlszähler um eins zu erhöhen und den aktuellen Befehl in das Befehlsregister zu übertragen. Dabei wird der aktuelle Befehl aus dem RAM genau an der Stelle ausgelesen, welche durch den Befehlszähler adressiert wird.
Das Steuerwerk gibt nun je nach Befehl (steht ja im Befehlsregister) unterschiedliche Signale an das Rechenwerk. Z. B. bedeutet der Befehl „01“, dass der Wert des Eingaberegisters auf den Akkumulator aufaddiert wird. Der Befehl „10“ bedeutet, dass der Wert des Akkumulators ausgegeben wird, d. h. in den Zwischenspeicher geschoben wird. Der Befehl „11“ bedeutet schließlich, dass der Wert des Zwischenspeichers in das Eingaberegister geschoben wird.
Um dem Benutzer trotz eines Dauertaktsignals auch die Möglichkeit der Eingabe zu geben, gibt es den Befehlscode „00“, welcher dafür sorgt, dass der Taktgeber gestoppt wird. Somit ist es möglich, komplexere Berechnungen durchzuführen. Am folgenden Befehlscode wird erläutert, wie sich die einzelnen Befehle auswirken:

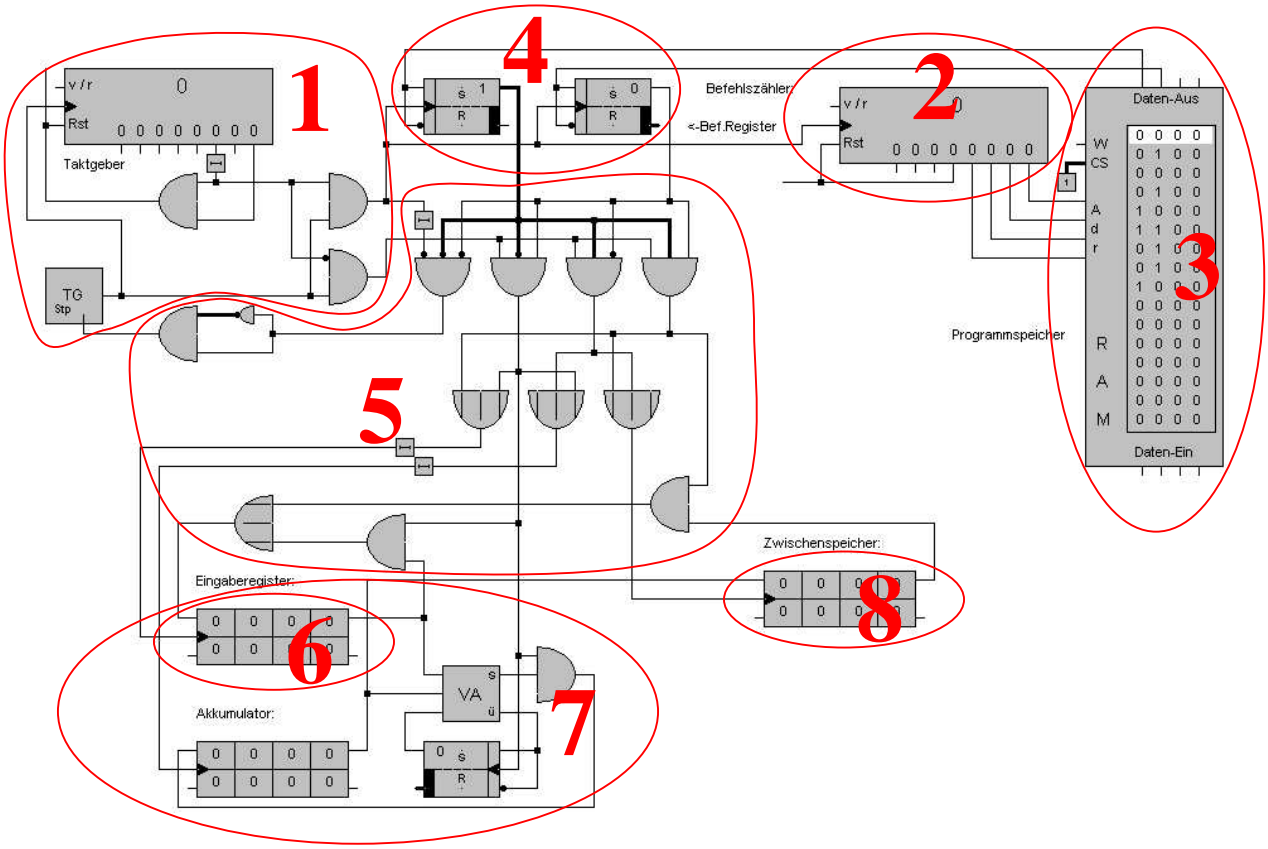
Befehlscode	**Beschreibung**	**Kurz**
00	Stop! Der Anwender bekommt Gelegenheit, im Eingaberegister die Zahl **a** einzugeben.	EinReg ← a
01	Addiere! Der eingegebene Wert (a) wird auf den Akku addiert. Da dieser zu Anfang 0 ist, bekommt der Akku den Wert des Eingaberegisters.	Akku ← Akku + EinReg
00	Stop! Der Anwender bekommt Gelegenheit, eine zweite Zahl einzugeben.	EinReg ← b
01	Addiere! Der eingegebene Wert (b) wird auf den Akku addiert.	Akku ← Akku + EinReg
10	Ausgabe! Der Akku wird im Zwischenspeicher ausgegeben	ZWS ← Akku
11	Eingabe! Das Eingaberegister bekommt den Wert des Zwischenspeichers	EinReg ← ZWS
01	Addiere! Der Wert des Eingaberegisters wird auf den Akku addiert. Dieser ist 0, deshalb bekommt der Akku den Wert des Eingaberegisters.	Akku ← Akku + EinReg
01	Addiere! Der Wert des Eingaberegisters wird noch mal auf den Akku addiert.	Akku ← Akku + EinReg.
10	Ausgabe! Der Wert des Akkus wird im Zwischenwertspeicher ausgegeben.	ZWS ← Akku

Rechnerisch passiert also folgendes:
Akku = a ; Akku = a+b ; ZWS = Akku, also a+b
Eingaberegister = ZWS, also (a+b) ; Akku = (a+b)
Akku = (a+b)+(a+b) ; ZWS = Akku, also 2·(a+b).
Die Ausgabe des Programms lautet also 2·(a + b).

Dies war nun allerdings sehr theoretisch und soll deshalb von dir selbst ausprobiert werden. Dazu ist es notwendig, den eben nur beschriebenen Modellrechner nachzubauen. Dieser sieht wie folgt aus:


Daten-Aus
Daten-Ein
W
CS
A
d
r
R
A
M
Programmspeicher
Befehlszähler:
<-Bef.Register
Zwischenspeicher:
Eingaberegister:
Akkumulator:
Taktgeber
TG
Stp
Rst
V/A

Im Einzelnen nochmals die Bestandteile:



(1) Der Taktgeber gibt setzt sich nach 5 Takten auf Null zurück. Vier Takte dienen der Rechnung (unteres UND-Gatter), ein Takt dient dem Befehl holen (oberes UND-Gatter). Welcher Befehl dabei geholt wird, das entscheidet...

(2) der Befehlszähler. Dies ist ein Binärzähler, der den aktuellen Befehl adressiert. Dabei befinden sich alle Befehle hintereinander im sogenannten...

(3) RAM, oder auch Hauptspeicher. An die Datenausgänge werden die Informationen der adressierten Speicherzelle weitergegeben, welche dann an das...

(4) Befehlsregister weitergegeben werden. Damit steht immer der aktuell durchzuführende Befehl in diesen beiden FlipFlops. Je nach Belegung des Befehlsregisters muss nun das...

(5) Steuerwerk unterschiedliche Signale an die Bauteile abgeben. Ist z. B. der Befehlscode 00, so wird ein kurzer Impuls an den Taktgeber zurückgesendet (linkes UND-Gatter). Ist der Befehlscode 01 (2. UND-Gatter), so erhält z. B. das...

(6) Eingaberegister ein Taktsignal, wodurch die Zahl des Eingaberegisters nach rechts geschoben wird. Gleichzeitig bekommt aber auch der Akkumulator ein Taktsignal, so dass beide Zahlen (Eingaberegister und Akkumulator) im...

(7) Serienaddierwerk (hier unsere ALU) verrechnet werden. Die Rückführung des Ergebnisses in den Akkumulator sorgt dafür, dass das Ergebnis erhalten bleibt.
Soll das Ergebnis nun auch angezeigt werden, so muss das Steuerwerk den Befehlscode 01 (3. UND-Gatter) geben. Dies bewirkt, dass sowohl der Akkumulator, als auch das...

(8) Ausgaberegister einen Taktimpuls bekommen. Der Wert des Akkumulators wird also in den Zwischenspeicher (Ausgabe) geschoben.

Aufgabe 42: Baue den Modellrechner in LOCAD nach. Programmiere anschließend Programme zur Berechnung von

a) Akku = a + 2·b **b)** Akku = 3·(a + b) + c **c)** Akku = 3·x + 2·y