Mathematik, Berechnung und Basteleien

Author:
Oliver Skoček

Inhaltsverzeichnis

Kapitel 1

Einführung

Das Ziel dieses Buches ist es dem Leser die Grundkonzepte von Rechenmaschinen zu vermitteln. Viel mehr noch soll die Fähigkeit vermittelt werden einen Computer selbst zu erdenken und zu bauen.

Dies geschieht auf mehreren Ebenen. Es soll dabei sowohl auf theoretische Konzepte der Programmierung als auch praktischer Konzepte aus der Elektronik eingegangen werden.

Im ersten Kapitel werden Konzepte aus der theoretischen Informatik vermittelt. Hierzu gehört der konzeptionelle Aufbau einer Rechenmaschine aber auch Prinzipien aus der Programmierung, wie die Church-Turing Hypothese oder die Kleensche Normalform.

Im zweiten Kapitel geht es um Zahlensysteme, wobei bereits Konzepte aus dem ersten Kapitel verwendet werden um dem Leser zu zeigen wie diese angewendet werden können.

Im dritten Kapitel werden Konzepte aus der Aussagenlogik erarbeitet. Das Konzept der Wahrheitstafel wird eingefürt, und eine Verbindung zu algebraischen Konzepten wird gezogen.

Im vierten Kapitel werden grundlegende Konzepte der Elektronik eingeführt. Dem Leser wird vermittelt was ein elektrisches Netzwerk ist, was Strom und Spannung sind und wie man die beiden Grössen an belibigen Punkten in einem solchen Netzwerk berechnen kann. Dies geschieht zuerst im zeitunabhängigen Fall und dann im zeitabhängigen Fall. Hierbei werden schrittweise elektronische Komponenten vorgestellt und es werden auch auf die Grenzen der vorgestellten Modelle zur Berechnung eingegangen.

Im fünften und letzten Kapitel wird eine Rechenmaschine entwickelt. Hier-

bei wird auf die letzten Kapitel zurückgegriffen um eine möglichst einfache Maschine zu konstruieren, die dennoch im Prinzip ein vollständiger, funktionierender Computer ist

Kapitel 2

Aufbau einer Rechenmaschine

Was muss eine Rechenmaschine können, damit sie im Prinzip "alles berechnen" kann? Diese Frage wurde Mitte des zwanzigsten Jahrhunderts beantwortet. Mehrere Personen entwickelten verschiedene einleuchtende Definitionen des Berechenbaren. Im Laufe der Zeit wurde dann gezeigt, dass all diese Konzepte gleichwertig sind.
Man sagt zwei Rechenmaschinen A und B sind gleichwertig, falls jede Berechnung die auf Rechenmaschine A durchgeführt werden kann, auch auf Rechenmaschine B durchgeführt werden kann, und umgekehrt. Anders ausgedrückt sind sie gleichwertig wenn jeweils eine Maschine die andere simulieren kann. Dies gilt unabhängig davon ob es sich um eine analoge Rechenmaschine, eine digitale Rechenmaschine, eine Quantenrechenmaschine oder einen Menschen mit Papier und Stift handelt.

Wir wollen bei der Analyse der Berechenbarkeit die Größe des Speichers ausklammern. Keine physikalische Rechenmaschine wird je universelle Berechenbarkeit erreichen, weil es ein endliches Gerät mit endlich viel Speicher ist. Man kann einfach sagen, dass der Speicher ja immer vergrössert werden kann bei Bedarf, und das so rechtfertigen, aber das ist Geschmacksache.

Was ist eine Rechenmaschine?
Eine Rechenmaschine ist ein Apparat, der Listen von Befehlen abarbeitet und dabei drei fundamentale Bestandteile aufweist:

Register: Eine Komponente, die ein Wort, daher eine Kette von Symbolen, speichern kann. Das Wort muss dementsprechend nachdem es im Register platziert wurde zu einem späteren Zeitpunkt abgerufen werden können. Bei Bedarf muss es möglich sein das Wort im Register durch ein Anderes zu ersetzen.

Ein Schmierzettel zusammen mit Bleistift und Radiergummi sind ein Beispiel aus dem Alltag.

Grundoperationen: Eine Liste von fundamentalen Operationen, welche die Maschine auf ein Wort oder mehrere Worte anwenden kann. Eine Operation ist dabei ein Vorgang der ein Wort oder Liste von Worten, die Operanden, nimmt und dann ein Wort als Resultat ausgibt.

Ein Alltagsbeispiel sind die Grundoperationen wie Addition, Subtraktion, Multiplikation und Division die man in der Schule gelernt hat.

Bedingte Verzweigung: Ein Mechanismus, der es erlaubt, dass der Ausgang einer Operation die Reihenfolge der abzuarbeitenden Befehle abändert. Diese Konstrukte haben für gewöhnlich die Form: "Wiederhole bestimmte Teilliste von Befehlen, bis eine Bedingung erreicht ist" oder "Falls Bedingung erfüllt ist springe zum so und so vielten Befehl und mach von dort weiter.

Die feste Wortlänge ist notwendig, da eine Realisierung einer Rechenmaschine, wenn es sich nicht gerade um einen Menschen mit Papier und Bleistift handelt, eine fixe Anzahl an Zahnrädern, Schaltkreisen, oder Ähnliches haben wird.

Bedingte Verzweigungen machen es notwendig, dass die Rechenmaschine einen Zustand hat, der es erlaubt Eigenschaften des Ausgangs der letzten abgelaufenen Grundoperation auszulesen.
Dazu dienen so genannte **Flags**, das sind ähnlich wie Register, die zwei Werte haben können, daher "WAHR" oder "FALSCH" und für eine Eigenschaft des Ausgangs einer Operation stehen. Ein Flag kann zum Beispiel dafür stehen, dass zuletzt Null das Resultat war, oder eine Addition einen Überlauf, also "eins weiter", bei der höchsten Stelle hatte. Im Gegensatz zu Registern, kann man den Wert des Flags nur indirekt über den Ausgang einer Grundoperation steuern.

In den nächsten Abschnitten werden wir uns näher mit den Befehlen, die eine Rechenmaschine abarbeitet, auseindandersetzen.
Zuvor soll hier aber noch das Konzept der Rechenmaschine mit dem der Programmiersprache in Beziehung gesetzt werden.

Was ist eine Programmiersprache?
Eine Programmiersprache ist ein Regelwerk mit Rechtschreibung und Grammatik, nach dem ein Text verfasst werden kann. Dieser Text muss als

Liste von Befehlen interpretierbar sein, die dann von einer Rechenmaschine abgearbeitet werden können.

Einen solchen Text nennt man auch ein **Programm**.

In der Regel hat jede Rechenmaschine eine eigene interne Programmiersprache, ihre **Machinensprache** und sie kann nur Programme verstehen und die darin beschriebene Berechnung durchführen, wenn sie in dieser Maschinensprache geschrieben ist.

Ein Regelwerk einer Programmiersprache hat die folgenden Komponenten:

1. **Konstanten:** Eine Liste von Symbolen für Zahlen. Als Beispiel nehmen wir die natürlichen Zahlen in dekadischer Darstelung $\{0, 1, 2, 3, \dots\}$.

2. **Variablen:** Eine Liste von Symbolen für Variablen, also Symbole die mit Zahlen belegt werden können. Ein Beispiel wäre $\{x, y, z\}$.

3. **Operationen:** Eine Liste von Symbolen die für Operation auf Zahlen, Zahlenpaaren oder Listen von Zahlen ausgeführt werden können. Ein Beispiel wäre $\{+, -, *, :, \}$.

4. **Vergleichssymbole**: Eine Liste von Vergleichen zwischen Zahlen. Ein Beispiel wäre $\{<, >, \leq, \geq, ==\}$.

5. **Schreiboperationen:** Eine Liste von Variablenoperationen, daher Operationen die den Wert einer Variablen verändern, oder das Verhalten von Variablen verändern. Ein Beispiel wäre $=$ oder $\leftarrow$. Hier soll einer Variable links vom Symbol der Wert rechts vom Symbol zugeordnet werden.
$$x \leftarrow 3$$
Das Verhalten dieser Operationen kann unterschiedlich sein abhängig von der Programmiersprache. Es muss dabei nicht notwendigerweise eine Kopie des zugeordneten Wertes entstehen.

6. **Steuersymbole:** Eine Liste von Symbolen, die für bedingte Verzweigungen stehen. Einige werden wir in Kürze kennen lernen. Vorweggenommen sei das Beispiel $\{WHILE, IF, END\}$.

Im Regelwerk der Programmiersprache muss stehen wie man aus Operationen, Konstanten und Variablen Terme formen kann. Eine Definition kann folgendermaßen aussehen:

1. Eine Konstante oder Variable ist ein Term.

2. Falls f eine Operation ist, x_1 x_2, ... x_k Terme sind und f auf k Zahlen anwendbar ist, dann ist $f(x_1, x_2, \ldots, x_k)$ auch ein Term. Im Fall von $+$, $-$, und $*$ würde man schreiben: $x_1 + x_2$, $x_1 - x_2$ und $x_1 * x_2$ respektive.

Eine solche Defintion nennt man rekursiv und wir werden im Laufe dieses Buches diesen Begriff noch öfter hören. Terme stehen wenn allen Variablen Werte zugeordnet sind für den entsprechenden Zahlenwert den man durch ausführen der Operationen in der geeigneten Reihenfolge erhÏlt.

Übrige Regeln müssen noch festlegen wie mit Hilfe der Vergleichssymbole Aussagen gebaut werden können und ganz allgemein wie Steuersymbole, Schreiboperationen, Aussagen und Terme angeordnet werden dürfen um als Befehlsliste interpretierbare Programme zu bilden.

Programmieren ist das entwerfen von Programmen. Jeder der noch nichts damit zu tun hatte, kann sich das komplett analog zu einem Koch vorstellen, der ein Rezept für einen Laien schreibt. Das Rezept ist eine Abfolge an Operationen, die der Laie in der Küche ausführen muss, um eine Speise herzustellen. Dem Laien muss man die Anweisungen ganz genau aufschreiben, also präzise formuliert, nach gewissen Regeln, damit die Anweisung immer eindeutig ist und keine Interpretationsfreiheit zulässt. Genauso verhält es sich mit der Programmierung einer Rechenmaschine.

2.1 GOTO-Programme

Eine der einfachsten und am leichtesten verständlichen Arten von Programmiersprachen ist die Klasse der GOTO-Programmiersprachen. In der Praxis sind GOTO-Programmiersprachen nur von geringer Bedeutung. Grund hierfür ist, dass Programme, die in solchen Sprachen geschrieben wurden, schwer lesbar und daher auch nur schwer wartbar sind. Allerdings sind die internen Maschinensprachen von allen (mir) bekannten Rechenmaschinen GOTO-Programmiersprachen und vor allem für die Theoriebildung interessant.

Wir wollen nun eine sehr einfache GOTO-Programmiersprache definieren, die bereits alle Zutaten für eine universelle Programmiersprache enthält. Fortan soll sie uns als Ausgangspunkt für Diskussionen des Berechenbaren dienen.

Variablen sind fast synonym zu verstehen mit Registern, es sind Dinge die ein Wort speichern können, das gelesen, und durch ein anderes Wort ersetzt werden kann. Ein Wort soll in der folgenden Diskussion, um möglichst einfach zu bleiben, eine natürliche Zahl sein.

Definition 2.1. *Ein GOTO-Programm ist durch folgendes Schema definiert:*

1. *Das Kopieren des Wertes einer Variablen X auf eine Variable Y, kurz $Y = X$, ist ein GOTO-Programm.*

2. *Einer Variable X, einen bestimmten Wert A geben, kurz $X = A$[1], ist ein GOTO-Programm.*

3. *Einer Variablen Z die Summe zweier Variable X und Y als Wert zuordnen, kurz $Z = X + Y$, ist ein GOTO-Programm.*

4. *Einer Variablen Z die Differenz zweier Variable X und Y als Wert zuordnen, kurz $Z = X - Y$, ist ein GOTO-Programm.*[2]

5. *Das Stoppen des Programmes, kurz $HALT$ ist ein GOTO-Programm.*

6. *Seien W und V GOTO-Programme, dann ist*

```
W
V
```

   ein GOTO-Programm. Daher zwei GOTO-Programme untereinandergeschrieben bilden ein GOTO-Programm. Praktisch bedeutet dies, dass zuerst das obere Programm und dann das untere Programm ausgeführt wird. Beispiel:

```
X = Y
Z = X + Y
```

   Hier wird der Variable X der Wert der Variablen Y zugeordnet und anschließend wird der Variablen Z die Summe der Variablen X und Y zugeordnet.

7. *Das Springen zu der n-ten Zeile des GOTO-Programms, falls die Variable X einen Wert ungleich Null aufweist, und nichts tun falls es einen Wert gleich null hat, kurz IF X GOTO n.*
   Beispiel (Arithmetische Multiplikation):

```
X = 0
X = X + Y
Z = Z - 1
IF Z GOTO 2
```

[1] A ist zum Beispiel 12, daher kurz $X = 12$.

[2] Da wir mit natürlichen Zahlen und keinen ganzen Zahlen arbeiten, setzen wir eine Differenz, die in einer negativen Zahl resultiren würde auf den Wert Null.

Das Beispielprogramm unter Punkt sieben beschreibt die Multiplikation zweier natürlicher Zahlen Z und Y, daher wenn das Programm zu Ende gelaufen ist, steht in der Variable X das Produkt der Werte, die zum Start des Programmes in der Variablen Z und Y gespeichert waren. Es ist zu beachten, dass wir eins basiert nummerieren. Also mit Zeilennummerierung als Orientierung sieht unser Programm so aus:

```
1: X = 0
2: X = X + Y
3: Z = Z - 1
4: IF Z GOTO 2
```

Wie nicht unschwer zu erkennen ist, steckt der Grund warum wir dies eine GOTO-Programmiersprache nennen im siebten Punkt der Definition. Dieser Punkt legt fest wie bedingte Verzweigung in der Programmiersprache funktioniert. Im Laufe der nächsten Kapitel werden wir weitere Möglichkeiten kennen lernen wie man dies bewerkstelligen kann.

Übungsbeispiel 1: Schreibe ein GOTO-Programm, dass die Fakultät einer natürlichen Zahl n berechnet, kurz $n!$. Die Fakultät ist rekursiv definiert durch:

$$0! = 1$$

$$(n+1)! = (n+1) * n!$$

In einer einzigen Formel lässt sie sich aber auch, weniger präzise, folgendermaßen definieren:

$$n! = n * (n-1) * \cdots * 2 * 1$$

Beispiel: $5! = 5 * 4 * 3 * 2 * 1$

2.1.1 Grundoperationen

Die Grundoperationen unserer GOTO-Programmiersprache sind die Summe und die Differenz zweier natürlicher Zahlen. Alternativ gibt es aber auch noch andere mögliche Operationen, die in Kombination unsere Grundoperationen ausdrücken können und damit genauso Kandidaten für Grundoperationen sind. Hier wollen wir kurz ein paar Beispiele bringen.

Nachfolger und Vorgänger Alternativ zur Addition und Subtraktion kann man auch die einstellige Operation des Nachfolgers ($S(n) = n + 1$) und seiner Umkehroperation Vorgängers ($T(n) = n - 1$)[3] einer natürlichen Zahl verwenden. Offensichtlich lässt sich die Addition in diesen Operationen ausdrücken:

[3] Es soll gelten $T(0) = 0$.

```
1: X = S(X)
2: Y = T(Y)
3: IF Y GOTO 1
```

Übungsbeispiel 2: Zeige wie sich die Subtraktion durch S und T in einem GOTO-Programm ausdrücken lässt.

Nachfolger und Gleichheit Die Vorgängeroperation lässt sich auch durch eine Gleichheitsrelation austauschen. Relationen sind Operationen, die als Resultat entweder Null für falsch oder Eins für wahr ausgeben. In unserem Fall nimmt die Operation zwei natürliche Zahlen X und Y und schreibt eine Eins, falls die Werte gleich sind, und sonst eine Null, in die Variable Z.
Als Operation schreiben wir dies als: $Z = (X == Y)$.

Übungsbeispiel 3: Zeige wie sich die Vorgängeroperation durch die Nachfolgeroperation und die Gleichheitsrelation in einem GOTO-Programm ausdrücken lässt.

Übungsbeispiel 4: Zeige wie sich die Gleichheitsrelation in in der ursprünglichen Definition eine GOTO-Programms ausdrücken lässt.

2.2 WHILE-Programme

Eine, wegen ihrer leichteren Lesbarkeit, beliebtere Klasse von Programmiersprachen sind die WHILE-Programmiersprachen. Die meisten modernen Programmiersprachen sind unter anderem auch WHILE-Programmiersprachen.

Es folgt nun wie bei den GOTO-Programmen eine Beschreibung einer einfachen Form einer WHILE-Programmiersprache. Der einzige Untschied zu GOTO-Programmen liegt im siebenten Punkt der Definition, nämlich der Implementierung bedingter Verzweigungen.

Definition 2.2. *Ein WHILE-Programm ist durch folgendes Schema definiert (übernehme Punkt 1-6 von der Definition eines GOTO-Programmes, lies einfach wo auch immer GOTO-Programm geschrieben steht, WHILE-Programm):*

7. *Wenn X ein WHILE-Programm ist und Z eine Variable, dann ist:*

```
WHILE Z:
    X
```

   auch ein WHILE-Programm. Es bedeutet, dass das WHILE-Programm X solange ausgeführt wird bis die Variable Z Null ist.
   Beispiel:(Arithmetische Multiplikation)

```
X = 0
WHILE Z:
    X = X + Y
    Z = Z - 1
```

Wir sehen hier, wie bereits für GOTO-Programme gemacht ein WHILE-Programm, das das Produkt zweier natürlicher Zahlen X und Z berechnet.

An diesem einfachen Beispiel lässt sich wenn wir es mit dem zugehörigen GOTO-Programm vergleichen bereits erkennen, dass und inwiefern WHILE-Berechenbarkeit und GOTO-Berechenbarkeit equivalent sind. Zuerst definieren wir was es bedeutet für zwei Programme equivalent zu sein.

Definition 2.3. *Seien P und Q Programme, dann sagen wir P und Q sind equivalent, wenn beide Programme bei gleicher Eingabe den selben Wert produzieren*[4]*.*

Zwei Programmiersprachen nennen wir equivalent, wenn es zu jedem Programm in der einen Programmiersprache ein equivalentes Programm in der anderen Programmiersprache gibt und umgekehrt. Was uns sogleich zum nächsten Abschnitt führt.

Übungsbeispiel 5: Mach Übungsbeispiel 1, 2 und 3 mit WHILE-Programmen, anstelle von GOTO-Programmen.

2.3 Die Church-Turing Hypothese

Die Church-Turing Hypothese (CT-Hypothese) ist eine Vermutung über das was überhaupt berechenbar ist. Inspiriert wurde sie durch die Beobachtung, dass alle Berechenbarkeitsmodelle, nur Berechnungen ausdrücken die sich auch als GOTO-Programm schreiben lassen. Die CT-Hypothese besagt also, dass jede Berechnung die im Prinzip möglich ist von einer Maschine ausgeführt werden kann, die GOTO-Programme verarbeiten kann. Es ist unmittelbar klar, dass die Aussage eher philosophisch und nicht beweisbar ist, da "das Berechenbare" nicht leicht fassbar/definierbar ist und wir niemals wissen können was morgen noch für Maschinen sein werden und welchen Gesetzen sie gehorchen werden. Es sei weiters noch angemerkt, dass wir aus ersichtlichem Grund die GOTO-Programmiersprache, sowie jede Programmiersprache, die jede Berechnung ausführen kann, die von einem GOTO-Programm ausgeführt werden kann, eine universelle Programmiersprache genannt werden soll. Eine solche universelle Programmiersprache ist sozusagen maximal in ihrer Fähigkeit Berechnungen auszuführen. Zur

[4]Eine beliebe Eingabe schließt auch die leere Eingabe, also keine Eingabe mit ein.

Motivation der Hypothese werden wir als nächstes beweisen, dass zu jedem GOTO-Programm ein WHILE-Programm existiert, dass daselbe berechnet und umgekehrt

IF THEN END Doch bevor wir dies zeigen soll eine bedingte Verzweigung eingeführt werden, die zwar nicht zwingend notwendig ist, daher wir können sie mit WHILE-Programmen sowie GOTO-Programmen bereits ausdrücken, aber sie werden, die in unseren Beweisen vorkommenden Programme kompakter und leichter lesbar machen, wenn wir sie verwenden.

Definition 2.4. *Sei Z eine Variable, P und Q sind GOTO/WHILE-Programme, dann nennen wir das Konstrukt*

```
IF Z THEN
        P
Q
```

eine IF THEN END Anweisung und sie bedeutet, dass falls Z ungleich Null ist, dann wird P ausgeführt und anschließend Q, andererseits falls Z gleich Null ist, dann überspringen wir P und führen gleich Q aus.

Übungsbeispiel 6: Schreibe ein GOTO-Programm, dass dieselbe Berechnung ausführt wie das oben beschriebene IF THEN END Konstrukt.

Übungsbeispiel 7: Schreibe ein WHILE-Programm, dass dieselbe Berechnung ausführt wie das oben beschriebene IF THEN END Konstrukt.

Übungsbeispiel 8: Schreibe ein WHILE-Programm, dass dieselbe Berechnung ausführt wie das oben beschriebene IF THEN END Konstrukt.

Übungsbeispiel 9: Zeige wie sich die Gleichheitsrelation als WHILE-Programm ausdrücken lässt.

Beweis. **WHILE → GOTO:** Gegeben sei ein WHILE-Programm, wir können nun jedes Vorkommnis einer WHILE-Struktur

```
WHILE Z:
        P
```

nach folgendem Schema schrittweise durch eine GOTO-Struktur ersetzen:

```
n - 2: X = 1 - Z
n - 1: IF X GOTO m + 1
n    : P
m    : IF Z GOTO n
```

wobei n die Nummer des ersten Befehls im Programm P ist und die Rollen von X und Z so zu verstehen sind, dass zu jeder Variablen Z eine eigene Variable X erzeugt werden soll, die sonst nirgendwo verwendet wird.
Wenn dieser Prozess abgeschlossen ist, haben wir ein zu unserem ursprünglichen WHILE-Programm, equivalents GOTO-Programm erzeugt.

GOTO → WHILE: Gegeben sein ein GOTO-Programm P. Wir starten mit der Konstruktion eines equivalenten WHILE-Programmes Q indem wir die ersten beiden Zeilen schreiben:

```
Y = 1
WHILE Y
        X = 0
```

Wir rücken nun ein und der Rest des Programmes, dass wir konstruieren wird nun innerhalb dieser initialen WHILE-Anweisung laufen. Falls die Variable Y bereits in P vorkommt, wähle anstelle von Y eine Variable, die nicht in P vorkommt.

Anschließend fügen wir für jede Zeile im Programm P, der Reihe nach einen Segment folgender Form am unteren Ende von Q, mit der Einrückung der ersten Zeile nach dem ersten WHILE, hinzu.[5]

```
Z = (X == K)
IF Z THEN
        'Befehl von Programm P in der Zeile K'
        X = X + 1
```

Falls der Befehl in der K-ten Zeile von P eine Sprunganweisung ist, ersetzen wir den dort stehenden Befehl:[6]

```
IF R GOTO n
```

durch

```
IF R THEN
        X = n
```

ansonsten schreiben wir den Befehl so wie er in P steht an die entsprechende Position. Das so erzeugte WHILE-Programm ist equivalent zum ursprünglichen GOTO-Programm. □

Abschließend sei noch erwähnt, dass die CT-Hypothese nichts über die Anzahl der Rechenschritte oder die Zeit, die eine Machine zur Berechnung benötigen wird aussagt. Ein Quantencomputer wird eine Faktorisierung

[5] Da unsere WHILE-Programmiersprache keine IF THEN END Anweisungen enthalten, stehen die IF THEN END Anweisungen hier für ein funktionales equivalent in unserer WHILE-Programmiersprache.

[6] R steht hier für die im Befehl vorkommende Variable.

einer großen natürlichen Zahl in kurzer Zeit vollbringen, während eine klassischer Computer Jahrhunderte braucht. Aber diesen Aspekt von unterschiedlichen Programmiersprachen und Maschinen haben wir bereits gesehen, als wir untschiedliche Grundoperationen für GOTO-Programme in betracht gezogen haben.

2.4 Kleenesche Normalform

Im vorangegangenen Abschnitt haben wir gesehen, wie zu jedem GOTO-Programm ein equivalentes WHILE-Programm erzeugt werden kann und umgekehrt.
Das Ziel dieses Abschnitts ist es zu zeigen, dass sich durch eine leichte Modifikation der Sprache ein beliebiges WHILE-Programm in eine Form umwandeln lässt in der nur eine einzige WHILE Anweisung enthalten ist.
Das Geheimnis hinter dieser Umwandlung liegt darin der Sprache um IF-THEN-END Anweisungen zu erweitern. Diese sollen aber nicht wie im letzten Abschnitt beschrieben aus einer WHILE Anweisungen aufgebaut sein, sondern sollen selbst einen elementarer Baustein der Sprache darstellen.

Die resultierende Sprachen nennen wir WHILE/IF-Programmiersprache. Sie ist equivalent zu unsere WHILE-Programmiersprache, da sich IF THEN END durch WHILE Anweisungen ausdrücken lässt, aber erlaubt nun die Konstruktion einer so genannten Kleenschen Normalform zu einem gegebenen WHILE-Programm.

Theorem 2.5. *Sei P ein beliebiges WHILE Programm, dann existiert ein equivalentes WHILE/IF Programm P' mit nur einer einzigen WHILE Anweisung.*

Beweis. Wir folgen einfach dem Beweis aus dem letzten Abschnitt.
Gegeben ist ein WHILE-Programm P, wir wandeln es in ein equivalentes GOTO-Programm Q um, und konstruieren anschließend wie im Beweis ein equivalentes WHILE/IF Programm P', dass nur mehr eine einzelne WHILE-Anweisung besitzt. Beachte, dass im Gegensatz zum Beweis vorangegangenen Abschnitt IF-THEN-END Anweisungen hier keine Konstruktion aus WHILE-Anweisungen sind, sondern elementare Sprachbausteine. □

Abschließend sei hier noch erwähnt, dass sich beliebig komplexe IF-THEN-END Verschachtelungen immer in eine einfache lineare Form bringen lassen.

Theorem 2.6. *Sei P ein beliebiges Programm, das IF-THEN-END Verzweigungen enthält, dann gibt es ein dazu equivalents Programm Q, welches keine IF-THEN-END Verzweigung innerhalb einer anderen IF-THEN-END Verzweigung enthält.*

Beweis. Sei P ein Programm mit IF-THEN-END Verzweigungen und R und S Variablen, dann hat P die allgemeine Form:

```
IF R THEN
        q_1
        IF S THEN
                q_2
        q_3
```

Wobei q_1 ein Programmabschnitt sein soll, der keine Verzweigungen enthält im Gegensatz zu q_2 und q_3, die IF-THEN-END Verzweigungen enthalten können.
Wir nennen IF-THEN-END Verzweigungen innerhalb der ersten IF-THEN-END Anweisung innere Verzweigungen.
Unser Ziel ist es alle inneren Verzweigungen zu eliminieren. Daher die hervorgehobene innere Verzweigung und jene innerhalb der Programmabschnitte q_2 und q_3.

Es ist ein Leichtes ein zu P equivalentes Programm Q zu schreiben, in dem die hervorgehobene innere IF-THEN-END Anweisung eliminiert wurde. Sei X hierbei eine Variable, die nicht in P verwendet wird.

```
X = 0
IF R THEN
        X = R
        q_1
IF X ∧ S THEN
        q_2
IF X THEN
        q_3
```

Die einzigen verbleibenden inneren Anweisungen sind nun innerhalb von q_2 und q_3. Wir können diese Prozedur fortsetzen, indem wir entweder q_2 oder q_3 auswählen und die erste IF-THEN-END Verzweigung darin hervorheben. Wir erreichen dadurch wieder die allgemeine Form wie zu Beginn des Beweises und eliminieren wie gehabt.
Da jedes Programm nur endlich viele Verzweigungen enthält, wird irgendwann keine innere Anweisung mehr da sein womit der Beweis fertig ist. □

Offensichtlich ist die Tiefe einer IF-THEN-END Verschachtelung limitiert durch die Anzahl der zu Verfügung stehenden Variablen.

Insgesamt haben wir gezeigt, dass es keine Einschränkung ist, wenn wir nur eine einzige WHILE Schleife zur Verfügung haben und keine Verschachtelten IF-THEN-END Verzweigungen verwenden dürfen.

Beispiel: Gegeben sei folgendes WHILE-Programm, welches überprüft ob eine gegebene Zahl eine Primzahl ist.

Übungsbeispiel 10:

Kapitel 3

Zahlensysteme

Bislang haben wir Programmiersprachen als etwas angesehen, dass Operationen auf natürlichen Zahlen ausführt. Bislang haben wir darauf verzichtet genauer darauf einzugehen, was genau eine natürliche Zahl ist und wie sie in einer physischen Rechenmaschine dargestellt werden können. Eine Rechenmaschine wird nie beliebig große natürliche Zahlen darstellen können, da sie immer limitiert sein wird in der Anzahl der wohlunterscheidbaren Zustände die sie annehmen kann.

Bisher sind wir nicht näher auf das Konzept der natürlichen Zahl eingegangen, sondern haben eine gewisses Verständnis desser vorausgesetzt. Hier wollen wir damit brechen und unsere Vorstellungen etwas konkretisieren. Fangen wir bei Null an, oder eigentlich bei Eins. Was sind die natürlichen Zahlen? Natürliche Zahlen werden verwendet zum Zählen. Schauen wir uns den Prozess des Zählens an.

3.1 Der Zählvorgang

Wir haben zwei Haufen wohlunterscheidbarer Objekte. Einen Haufen Birnen und einen Haufen äpfel. Wenn wir wissen wollen ob genauso viele Birnen auf dem Birnenhaufen wie äpfel auf dem äpfelhaufen sind, können wir folgendermaßen vorgehen. Wir entfernen eine Birne vom Birnenhaufen und einen Apfel vom Apfelhaufen und tun dies so lange bis einer der Haufen verschwunden ist. Wenn beide gleichzeitig verschwinden, dann sind es gleich viele Birnen wie äpfel, ansonsten gibt es mehr Birnen respektive äpfel wenn der äpfelhaufen beziehungsweise der Birnenhaufen zuerst
verschwunden ist. Durch diesen Prozess ist man in der Lage Anzahlen von allen möglichen Gegenständen durch Haufen von äpfeln darzustellen. Haufen von äpfeln sind eine mögliche Darstellung von natürlichen Zahlen. Ein beliebiger Apfelhaufen kann gebildet werden indem man mit einem Apfel startet und schrittweise weitere äpfel hinzufügt. Wir wünschen uns jetzt

aber eine weniger verderbliche und kompaktere Darstellung von natürlichen Zahlen. Hierzu abstrahieren wir unsere äpfelhaufen.

Definition 3.1. *Jede Konstruktion mit den hier beschriebenen vier Eigenschaften nennen wir eine Darstellung der natürlichen Zahlen.*

1. *Es gibt eine erste natürliche Zahl, wir nennen sie Eins.*
2. *Es gibt zu jeder natürlichen Zahl n, eine eindeutige nächste natürliche Zahl, den Nachfolger $S(n)$.*
3. *Alle natürlichen Zahlen werden durch mehrmaliges Nachfolger bilden aus der Eins konstruiert.*
4. *Der Nachfolger $S(n)$ einer natürlichen Zahl n unterscheidet sich von n und jeder natürlichen Zahl, die im Bildungsprozess von n, also startend bei 1, über alle Nachfolgerbildungen bis hin zu n, auftaucht.*

Satz 3.1. *Beliebige zwei Darstellungen natürlicher Zahlen sind equivalent*[1]*.*

Beweis. Gegeben sind zwei Darstellungen natürlicher Zahlen A und B. Wir assozieren nun die Eins von A , kurz 1_A, mit der Eins von B, kurz 1_B. Falls ein Element x von A mit einem Element y von B assoziert wird dann wird auch der Nachfolger von x bezüglich A mit dem Nachfolger von y bezüglich B assoziert. Nach Eigenschaft vier aus der Definition der Darstellung natürlicher Zahlen werden unterschiedlichen Elementen von A mit unterschiedliche Elemente von B assoziiert. Zusammen mit Eigenschaft drei folgt damit sofort, dass diese Assoziation ϕ eine umkehrbar eindeutige Abbildung ist, welche die Nachfolgerabbildung erhält. In Formeln können wir den Sachverhalt ausdrücken als:

$$\phi(S_A(x)) = S_B(\phi(x))$$

für beliebiges x in B, wobei ϕ die oben definierte Assoziation ist und S_A beziehungsweise S_B die Nachfolgeroperationen von A respektive B sind. □

3.2 Das dekadische Zahlensystem

In der Schule haben wir eine besonders effiziente Darstellung natürlicher Zahlen kennen gelernt, das dekadische Zahlensystem. Zeigen wir nun, dass diese tatsächlich natürliche Zahlen im oben beschriebenen Sinn sind.

[1] Equivalent bedeutet hier, dass es eine eindeutig umkehrbare Abbildung gibt, die mit den Nachfolgeroperationen der beiden Darstellungen verträglich ist.

Wir starten mit zehn Symbolen $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ und der Einfachheit halber starten wir nicht bei Eins, sondern bei Null. [2]

Definition 3.2. *Das dekadische Zahlensystem ist durch folgende Regeln definiert:*

1. *Die zehn Symbole haben eine feste Reihenfolge* $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$. *Diese Reihenfolge definiert eine Operation* u*, die ein Symbol nimmt und das nächste Symbol in der Reihenfolge ausgibt. Wenn es das Symbol* 9 *bekommt gibt es* 0 *aus.*
   Daher $u(0) = 1$*,* $u(1) = 2$*,* $u(2) = 3$*,* $u(3) = 4$*,* $u(4) = 5$*,* $u(5) = 6$*,* $u(6) = 7$*,* $u(7) = 8$*,* $u(8) = 9$*,* $u(9) = 0$*.*

2. *Eine dekadische Zahl ist eine endliche Abfolge dieser zehn Symbole. Hierbei schreiben/lesen wir von rechts nach links. Das erste Symbol oder wir sagen die erste Stelle ist das rechteste Symbol.*
   Beispiele:

   ```
   28420
     455
     390
      89
   ```

3. *Die Eins des dekadischen Zahlensystems ist:*

   ```
   1
   ```

4. *Wir definieren nun die Nachfolgeroperation des dekadischen Zahlensystems als ein Pseudo-GOTO-Programm.*

```
1: 'Starte bei der ersten Stelle.'
2: X = 'Symbol an der derzeitigen Stelle.'
3: X = u(X)
4: 'Setze den Wert der derzeitigen Stelle auf X.'
5: Z = X==0
6: if Z THEN
7:       'Gehe zur n\ächsten Stelle,
         falls keine Stelle mehr \übrig ist
         f\üge eine Stelle mit dem Wert 0 hinzu
         und gehe zu dieser Stelle.'
8:       if Z GOTO 2
9: HALT
```

[2] Der Startpunkt ist irrelevant, solange es ein erstes Element gibt. Falls notwendig kann man einfach das erste Element nachträglich entfernen und man hat eine Struktur die bezüglich der ursprünglichen Struktur mit dem zweiten startet, aber die equivalent zur ursprünglichen Struktur ist.

Übungsbeispiel 11: Spiele den Algorithmus für einige Beispiele von Zahlen durch um dich mit der Nachfolgeroperation vertraut zu machen.

Folgerung 3.3. *Die Stellen des dekadischen Zahlensystems sind selbst ein Zahlensystem.*

Übungsbeispiel 12: Beweise Folgerung 3.3.

3.3 Das binäre Zahlensystem

Nachdem wir uns jetzt das dekadische Zahlensystem angeschaut haben, kommen wir nun zum eigentlichen Thema, dem dualen Zahlensystem oder binären Zahlen. Im binären Zahlensystem gibt es im Gegensatz zu den zehn Symbolen des dekadischen Zahlensystem nur zwei Symbole nämlich $0, 1$. Wie bei den dekadischen Zahlen legen wir eine Reihenfolge der Symbole fest, nämlich $0, 1$ und definieren das duale Zahlensystem analog zum dekadischen.
Zur Wiederholung:

Definition 3.4. *Das dual/binäre Zahlensystem ist durch folgende Regeln definiert:*

1. *Die zwei Symbole haben eine feste Reihenfolge* $0, 1$.
2. *Eine binär Zahl ist eine endliche Abfolge dieser zwei Symbole. Hierbei schreiben/lesen wir von rechts nach links. Das erste Symbol oder wir sagen die erste Stelle ist das rechteste Symbol.*
   Beispiele:

   ```
   101010
      101
      111
       10
   ```

3. *Die Eins des dualen Zahlensystems ist:*

   ```
   1
   ```

4. *Wir definieren nun die Nachfolgeroperation des dualen Zahlensystems als eine Art GOTO-Programm:*

   ```
   1: 'Starte bei der ersten Stelle.'
   2: X = 'Symbol an der derzeitigen Stelle.'
   3: X = u(X)
   4: 'Setze den Wert der derzeitigen Stelle auf X.'
   5: Z = X==0
   ```

```
6: if Z THEN
7:       'Gehe zur n\ächsten Stelle,
         falls keine Stelle mehr \übrig ist
         f\üge eine Stelle mit dem Wert 0 hinzu
         und gehe zu dieser Stelle.'
8:       if Z GOTO 2
9: HALT
```

Beispiel: Alle vier stelligen binären Zahlen

0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Übungsbeispiel 13: Spiele den Algorithmus für einige Beispiele von Zahlen durch um dich mit der Nachfolgeroperation vertraut zu machen.

Übungsbeispiel 14: Berechne die binäre Darstellung der dekadischen Zahlen 23, 12 und 1000. Finde eine schnelle Methode ohne alle Zahlen von 1 bis zu der gewissen Zahl durchzugehen.

Im Allgemeinen nennen wir Zahlensysteme wie das dekadische oder das duale Zahlensystem Stellenwertsysteme. Im Prinzip müssen wir nur ein paar wohlunterscheidbare Symbole aussuchen, eine Reihenfolge festlegen und wir können beliebige derartige Systeme festlegen. In der Informatik ist ein beliebtes Zahlensystem das Hexadezimalsystem. Hier haben wir die Symbole $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F$.

Übungsbeispiel 15: Berechne die hexadezimale Darstellung der dekadischen Zahlen 15, 17 und 432.

In der Praxis haben sich Digitalrechner, also solche die das binäre Zahlensystem verwenden, bewährt. Dies ist weil unsere Rechenmaschinen elektronisch sind und es praktisch einfacher ist einen Schaltkreis zu bauen, der zwei Zustände kennt, nämlich viel Spannung oder Stromstärke und sehr wenig Spannung oder Stromstärke und es genauso einfacher ist diese zwei Zustände für einen oder mehrere weiteren Schaltkreise zu unterscheiden.

Kapitel 4

Aussagenlogik

Es ist nun an der Zeit, dass wir ein wenig Logik ins Spiel bringen. Logik ist die Lehre vom exakten Schließen; in einfacheren Worten beschäftigt sich die Logik damit wie man aus wahren Sätzen, wiederum wahre Sätze erzeugt. In diesem Kapitel werden wir uns mit Aussagenlogik oder Boolscher Logik befassen, doch bevor wir in die Tiefen der Logik starten, müssen wir zuerst ein paar Grundbegriffe definieren.

Eine **Aussage** ist ein sprachliches Konstrukt, dass entweder wahr oder falsch ist. Es muss hierbei prinzipiell möglich sein zu überprüfen ob die Aussage zutrifft, also wahr ist oder nicht. Nehmen wir die Aussage:

Die vierte Nachkommastelle von π ist 5.

Um zu überprüfen ob diese Aussage wahr ist, müssen wir die vierte Nachkommastelle der Kreiszahl π berechnen und überprüfen ob der resultierende Wert gleich 5 ist. Was prinzipiell geht und praktisch möglich oder sinnvoll ist, ist oft verschieden, doch ist eine Diskussion dieses Themas hier fehl am Platz.

Man drückt den Sachverhalt, dass eine Aussage ϕ wahr ist beziehungsweise falsch ist durch den **Wahrheitswert** von ϕ (kurz $w(\phi)$) aus. Wir schreiben den Wahrheitswert 1, falls die Aussage wahr ist oder 0, falls die Aussage falsch ist. Präzise formulier heißt dies für den Wahrheitswert einer Aussage ϕ schreiben wir:

$$w(\phi) = \begin{cases} 1 & \text{falls } \phi \text{ wahr ist.} \\ 0 & \text{falls } \phi \text{ falsch ist.} \end{cases}$$

Logische Verknüpfung sind Operationen die eine bestimmte Anzahl von Aussagen nehmen und daraus eine neue Aussage produzieren, deren Wahrheitswert allein von den Wahrheitswerten der Aussagen aus denen sie produziert wurde abhängt.

Ein Beispiel für eine jedem bekannte logische Verknüpfung, die nur eine

einzelne Aussage nimmt und daraus eine neue Aussage produziert, ist die **Negation** oder Verneinung (kurz $\neg$) einer Aussage.

$$\neg(\text{Der Himmel ist blau}) = \text{Der Himmel ist nicht blau.}$$

Hierbei gilt, dass die Verneinung die Wahrheitswerte umdreht, daher aus wahr mach falsch und aus falsch mach wahr.

$$w(\neg\phi) = 1 - w(\phi) = \begin{cases} 1 & \text{falls } w(\phi) = 0 \\ 0 & \text{falls } w(\phi) = 1 \end{cases} \tag{4.1}$$

Das logische **Und** ist eine logische Verknüpfung, die zwei Aussagen verbindet zu einer Aussage. Seien nun ϕ und ψ Aussagen, dann schreiben wir $\phi \wedge \psi$ für die durch das logische Und erzeugte Verbindung der Aussagen.

$$\phi = \text{Der Himmel ist blau.}$$

$$\psi = \text{Fische leben im Wasser.}$$

$$\phi \wedge \psi = \text{Der Himmel ist blau und Fische leben im Wasser.}$$

Diese Verknüpfung verhält sich genauso wie im gewohnten Sprachgebrauch $\phi \wedge \psi$ wahr, falls ϕ und ψ wahr sind und sonst falsch.

$$w(\phi \wedge \psi) = w(\phi) * w(\psi) = \begin{cases} 1 & \text{falls } w(\phi) = 1 \text{ und } w(\psi) = 1 \\ 0 & \text{sonst} \end{cases} \tag{4.2}$$

Abschließend kommen wir zum logische **Oder**, das wie das logische Und zwei Aussagen verbindet zu einer Aussage. Seien wieder ϕ und ψ Aussagen, dann schreiben wir $\phi \vee \psi$ für die durch das logische Oder erzeugte Verbindung der Aussagen.

$$\phi = \text{Der Himmel ist grün.}$$

$$\psi = \text{Fische leben im Wasser.}$$

$$\phi \vee \psi = \text{Der Himmel ist grün oder Fische leben im Wasser.}$$

Im Gegensatz zum Oder im gewöhnlichen Sprachgebrauch verhält sich das logische Oder jedoch anders. Das logische Oder ist wahr, sobald einer der beiden verbundenen Aussagen wahr ist.

$$w(\phi \vee \psi) = \begin{cases} 1 & \text{falls } w(\phi) = 1 \text{ oder } w(\psi) = 1 \\ 0 & \text{sonst} \end{cases} \tag{4.3}$$

Damit haben wir die wichtigsten logischen Verknüpfungen kennen gelernt und können damit bereits alles ausdrücken, was man in der Aussagenlogik ausdrücken kann. Wir werden später in diesem Kapitel noch weitere logische Verknüpfungen besprechen, die eine besondere Erwähnung verdienen.

Als nächstes wollen wir konkretisieren was ein aussagelogisches System ist. Man startet mit elementaren Aussagen, so genannten **logischen Atomen**. Dies sind die Grundsymbole unseres Systems aus welchen wir zusammen mit den logischen Verknüpfungen alle möglichen Kombinationen bilden können. Wir können nun jeder dieser Kombinationen einen Wahrheitswert geben, indem wir einfach für jedes logische Atom einen Wahrheitswert fixieren. Erst durch diese Zuordnung werden unsere Symbolketten von logischen Verknüpfungen und Atomen eigentlich Aussagen mit definierten Wahrheitswerten. Solange man aber nur die Symbolketten betrachtet und die logischen Atome noch nicht mit Wahrheitswerten belegt hat, nennt man diese Konstrukte aussagenlogische Formeln.

Definition 4.1. *Aussagenlogische Formeln sind durch folgendes Schema definiert:*

1. *Die logischen Atome $\{\phi_1, \phi_2, \dots\}$, daher die Elemente einer Liste von Grundsymbolen sind aussagenlogische Formeln.*

2. *Wenn ϕ und ψ aussagenlogische Formeln sind, dann auch $\neg(\phi)$, $(\phi) \wedge (\psi)$ und $(\phi) \vee (\psi)$. Falls ϕ oder ψ logische Atome sind, darf man an der Stelle wo dies zutrifft die Klammern hier weglassen.*

3. *Jede aussagenlogische Formel wird aus den logischen Atomen und mehrmalige Kombination dieser durch logische Verknüpfungen erzeugt.*

Jede aussagenlogische Formel wird zusammen mit einer Belegung der logischen Atome mit Wahrheitswerten, eine Aussage, durch folgendes Prinzip.

Definition 4.2. *Sei β eine **Belegung** der logischen Atome $\{\phi_1, \phi_2, \dots\}$, daher eine Zuordnung von 0 oder 1 zu jedem logischen Atom, dann lassen sich durch β Wahrheitswerte w für beliebige aussagenlogische Formeln durch folgendes Schema berechnen:*

Sei ϕ eine aussagenlogische Formel, dann gilt

1. *falls ϕ ein logisches Atom ist setzen wir den Wahrheitswert*

$$w(\phi) = \beta(\phi) \tag{4.4}$$

2. *falls ϕ kein logisches Atom ist, muss es nach Konstruktion entweder die Negation $\neg$ angewendet auf eine aussagenlogische Formel p sein, oder eine Verknüpfung durch das logische Und $\wedge$ oder das Oder $\vee$ von zwei aussagenlogischen Formeln p und q sein.*
   Im ersten Fall der Negation setzen wir den Wahrheitswert:

$$w(\phi) = 1 - w(p) \tag{4.5}$$

Im zweiten Fall des logischen Unds, setzen wir den Wahrheitswert:

$$w(\phi) = w(p) * w(q) \tag{4.6}$$

Im dritten Fall des logischen Oders, setzen wir den Wahrheitswert:

$$w(\phi) = \begin{cases} 1 & \textit{falls } w(p) = 1 \textit{ oder } w(q) = 1 \\ 0 & \textit{sonst} \end{cases} \tag{4.7}$$

Falls p beziehungsweise p und q logische Atome sind, wenden wir Punkt 1. an und sind fertig. Falls nicht, wenden wir wiederholt Punkt 2. an, bis wir ausschließlich logische Atome erreicht haben und wenden dann Punkt 1. an.

Wir zeigen nun anhand von einem Beispiel einer aussagenlogischen Formeln wie dies funktioniert.

Beispiel: Wir haben drei logische Atome ϕ_1, ϕ_2 und ϕ_3, mit Wahrheitswerten $w(\phi_1) = 0$, $w(\phi_2) = 1$ und $w(\phi_3) = 1$.

$\phi = (\neg(\phi_1)) \wedge (((\phi_2) \vee \phi_3) \wedge (\neg(\phi_2)))$

Berechne den Wahrheitswert $w(\phi)$:

Wir sehen, dass ϕ eine Und Verknüpfung von $p = \neg(\phi_1)$ und $q = (\phi_2 \vee \phi_3) \wedge (\neg(\phi_2))$ ist. Daher berechnet sich der gesuchte Wert durch

$$w(\phi) = w(p) * w(q) \tag{4.8}$$

Berechnen wir nun den Wahrheitswert für p

$$w(p) = w(\neg(\phi_1)) = 1 - w(\phi_1) = 1 - 0 = 1$$

und um den Wahrheitswert von q zu berechnen, bemerken wir, dass q eine Und Verknüpfung von $s = \phi_2 \vee \phi_3$ und $t = \neg(\phi_1)$ ist.

$$w(q) = w(s) * w(t) \tag{4.9}$$

Es ist ein leichtes die Wahrheitswertes für s und t zu berechnen:

$$w(s) = w(\phi_2 \vee \phi_3) = \begin{cases} 1 & \text{falls } w(\phi_2) = 1 \text{ oder } w(\phi_3) = 1 \\ 0 & \text{sonst} \end{cases} = 1$$

$$w(t) = w(\neg(\phi_2)) = 1 - w(\phi_2) = 1 - 1 = 0$$

Als nächstes setzen wir $w(s)$ und $w(t)$ in (1.9) ein und erhalten:

$$w(q) = w(s) * w(t) = 1 * 0 = 0$$

Abschließend setzen wir noch $w(p)$ und $w(q)$ in (1.8) ein und erhalten:

$$w(\phi) = w(p) * w(q) = 1 * 0 = 0$$

Übungsbeispiel 16: Berechne den Wahrheitswert der logischen Formeln

$$(\neg(\phi_1)) \vee ((\neg(\phi_2)) \wedge (\phi_3))$$

$$(\neg(\neg(\phi_3))) \vee \phi_1$$

$$\phi_3 \wedge (\phi_1 \vee (\neg\phi_1))$$

4.1 Wahrheitstafeln

Wir kommen nun zu einer sehr nützlichen Werkzeug zur Behandlung von aussagenlogischen Formeln, den Wahrheitstafeln. Hier wird eine Tabelle mit allen möglichen Wahrheitswerten für die Atome der aussagenlogischen Formel gebildet und für jede dieser Kombinationen schreibt man in der letzten Spalte den Wahrheitswert den die Formel für diese Kombination ergeben würde.

Wahrheitstafeln sind somit ein allgemeines Format zur Darstellung beliebiger aussagenlogischer Verknüpfunen und eignen sich hervorragend zum Finden von Lösungen aussagenlogischer Erfüllbarkeitsprobleme, die wir gleich anschließend behandeln werden. Starten wir mit den Wahrheitstafeln der drei Grundverknüpfungen Verneinung, Und und Oder.

ϕ	$\neg\phi$
0	1
1	0

ϕ	ψ	$\phi \wedge \psi$
0	0	0
0	1	0
1	0	0
1	1	1

ϕ	ψ	$\phi \vee \psi$
0	0	0
0	1	1
1	0	1
1	1	1

Eine gute Methode um alle möglichen Kombinationen von Wahrheitswerten von K logischen Atomen zu erzeugen und keine zu vergessen, ist es einfach die K stelligen binären Zahlen von Null, also der binären Zahl, die aus K Nullen besteht, bis zur größten binären Zahl mit K-Stellen, nämlich jener, die nur aus Einsern besteht, aufzuschreiben.

Beispiel: Wir haben drei logische Atome ϕ_1, ϕ_2 und ϕ_3. Die Wahrheitstafel für die aussagenlogische Formel $(\neg\phi_1) \vee ((\neg\phi_3) \wedge \phi_2)$ ist:

ϕ_1	ϕ_2	ϕ_3	$(\neg\phi_1) \vee ((\neg\phi_3) \wedge \phi_2)$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Übungsbeispiel 17: Stelle die Wahrheitstafeln zu beiden aussagenlogischen Formeln in Übungsbeispiel 16 auf.

Abschließend sollen in diesem Kapitel noch ein paar logische Verknüpfungen über ihre Wahrheitstabelle vorgestellt werden.

Die logische **Implikation** ist eine Verknüpfung zweier Aussagen ϕ und ψ, die ausdrücken soll, dass wenn ϕ wahr ist, auch ψ wahr sein muss. Wir schreiben hier $\phi \implies \psi$.

ϕ	ψ	$\phi \implies \psi$
0	0	1
0	1	1
1	0	0
1	1	1

Falls ϕ falsch ist, kann ψ falsch oder wahr sein und $\phi \implies \psi$ ist trotzdem wahr.

Übungsbeispiel 18: Zeige mit Hilfe von Wahrheitstafeln, dass $(\neg\psi) \implies (\neg\phi)$ dieselben Wahrheitswerte hat wie $\phi \implies \psi$.

Die logische **äquivalenz** zweier Aussagen ϕ und ψ drückt aus, dass die Wahrheitswerte der Aussagen gleich sind, kurz $\phi \equiv \psi$.

ϕ	ψ	$\phi \equiv \psi$
0	0	1
0	1	0
1	0	0
1	1	1

Übungsbeispiel 19: Die Verneinung der logischen äquivalenz ist das ausschließende Oder, oder auch **XOR** genannt. Schreibe die Wahrheitstafel der XOR Verknüpfung auf.

4.2 Erfüllbarkeitsprobleme und Gleichungen

Aussagenlogische Formeln lassen sich wie die Terme und Gleichungen der Algebra, die wir aus der Schule kennen interpretieren. Logische Atome sind, wenn keine Wahrheitswerte definiert sind, nichts anderes als Variablen, daher Symbole mit unbestimmten Werten. Die Werte sind im Fall von aussagenogischen Formeln entweder Null oder Eins.

Als Gleichheit verwendet man in der Logik die äquivalenz; Diese verhält sich wie die Gleichheit in der Schulalgebra.

Definition 4.3. *Seien daher ϕ, ψ und γ beliebige aussagenlogische Formeln dann gilt:*

1. *Eine Formel ist mit sich selbst äquivalent*

   (4.10) $$\phi \equiv \phi$$

2. *Wenn für zwei Formeln gilt*

   (4.11) $$\phi \equiv \psi \text{ dann gilt auch } \psi \equiv \phi$$

3. *Wenn für drei Formeln gilt*

   (4.12) $$\phi \equiv \psi \text{ und } \psi \equiv \gamma \text{ dann gilt auch } \phi \equiv \gamma$$

Eigentlich ist es notwendig Klammern um die beiden Formeln zu setzen, die links und rechts vom $\equiv$ Symbol sind, aber solange die äquivalenz als Gleichheit betrachten und nicht als Verknüpfung, werden wir die Klammern weglassen.

Eine erste Folgerung aus der Verwendung der äquivalenz als Gleichheit zusammen mit dem Sachverhalt, dass zwei Formeln genau dann äqivalent sind, wenn sie dieselben Wahrheitswerte haben und der Wahrheitswert, den eine logische Verknüpfung ergibt nur von den Wahrheitswerten der verknüpften Aussagen abhängt.

Folgerung 4.4. *Seien ϕ_1, ψ_1, ϕ_2 und ψ_2 aussagenlogische Formeln dann gilt.*

1. *Aus $\phi_1 \equiv \phi_2$ folgt $\neg(\phi_1) \equiv \neg(\phi_2)$.*
2. *Aus $\phi_1 \equiv \phi_2$ und $\psi_1 \equiv \psi_2$ folgt $\phi_1 \wedge \psi_1 \equiv \phi_2 \wedge \psi_2$.*
3. *Aus $\phi_1 \equiv \phi_2$ und $\psi_1 \equiv \psi_2$ folgt $\phi_1 \vee \psi_1 \equiv \phi_2 \vee \psi_2$.*

Da die äquivalenz selbst auch eine logische Verknüpfung darstellt, führt dies natürlich dazu, dass eine aussagenlogische Gleichung, zugleich als Gleichung aber auch als Formel interpretierbar ist.

Wir nehmen daher Abstand vom Begriff Gleichungen und betrachten das so genannte Erfüllbarkeitsproblem Aussagenlogischer Formeln.
In einfachen Worten ist das Erfüllbarkeitsproblem einer Formel ψ, die Suche nach Wahrheitswerten, also Null oder Eins, die wenn sie in ψ für die logischen Atome eingesetzt werden Eins als Wahrheitswert ergeben.

Definition 4.5. *Sei ψ eine aussagenlogische Formel und β eine Belegung der logischen Atome in ψ, sodass der zu β gehörende Wahrheitswert für ψ Eins ist, dann nennen wir die Belegung β eine Lösung des Erfüllbarketisproblems für ψ. Die Suche nach einer solchen Belegung, nennen wir folglich das Erfüllbarkeitsproblem von ψ.*

Genauso wie sich in der Schulalgebra Formeln umformen lassen in andere gültige Formeln, so lassen sich auch aussagenlogische Formeln umformen in äquivalente Formeln. Wir nennen die Algebra, die mit ausagenlogischen Formeln verbunden ist auch Boolsche Algebra nach dem Mathematiker George Bool, der nach unseren Aufzeichnungen, der erste war, der sich mit diesem Thema befasste.

Aussagenlogische Formeln lasssen sich bezüglich der logischen Und und der logischen Oder Verknüpfung genauso umformen wie die Multiplikation und Addition in der Schulalgebra. Die Besonderheit der Boolschen Algebra ist, dass das diese beiden Verknüpfungen in Bezug auf die Rolle, die sie bei den Umformungen einnehmen, austauschbar sind.

Es gilt in der Boolschen Algebra die **Assoziativität**, sowohl für das Und als auch für das Oder. Assoziativität bedeutet, dass solange wir nur Und oder nur Oder Verknüpfungen in einer Verkettung haben, ist die Reihenfolge der Verknüpfungen egal beziehungsweise lassen sich die Klammern beliebig setzen.

Seien ψ, ϕ und γ aussagenlogische Formeln, dann gilt.

$$(\phi \vee \psi) \vee \gamma \equiv \phi \vee (\psi \vee \gamma) \tag{4.13}$$

$$(\phi \wedge \psi) \wedge \gamma \equiv \phi \wedge (\psi \wedge \gamma) \tag{4.14}$$

Übungsbeispiel 20: Beweise die Formeln (1.10) und (1.11) indem du die Wahrheitstafel der Formel links der äquivalenz und rechts der äquivalenz aufschreibst und dich vergewisserst, dass sie gleich sind (Behandle dabei ψ, ϕ und γ wie logische Atome).

Übungsbeispiel 21: Wir haben in (1.10) und (1.11) gesehen, dass die Reihenfolge der Klammerung für drei Formeln verknüpft durch das logische Und oder das logische Oder egal ist. Zeige, dass dies für beliebige Viele gilt.

Eine direkte Konsequenz der Definition des Wahrheitswertes für Und und Oder Verknüpfungen ist die **Kommutativität** der beiden Verknüpfungen. Daher der Sachverhalt, dass es egal ist in welcher Reihenfolge zwei Formeln durch ein Und oder Oder verknüpft werden.

Seien ψ und ϕ aussagenlogische Formeln, dann gilt.

$$\phi \vee \psi \equiv \psi \vee \phi \tag{4.15}$$

$$\phi \wedge \psi \equiv \psi \wedge \phi \tag{4.16}$$

Übungsbeispiel 22: Beweise die Formeln (1.12) und (1.13) anhand der Definition des Wahrheitswertes (1.9), oder mit Wahrheitstafeln.

Bislang haben wir Umformungen von Formeln betrachtet, die nur das logische Und oder nur das logische Oder betrachten. Wir kommen nun zu Umformungen von Kombinationen von Und und Oder. Analog zur Schulalgebra gilt in der Boolschen Algebra das Gesetz der **Distributivität**, also das "Herausheben von Ausdrücken". Zur Erinnerung in der Schulalgebra gilt $a * (b + c) = (a * b) + (a * c)$. Im Unterschied zur Schulalgebra sind aber in der Boolschen Algebra die beiden Verknüpfungen, also das logische Und und das logische Oder gleichberechtigt.

Seien ψ, ϕ und γ aussagenlogische Formeln, dann gilt.

$$\phi \vee (\psi \wedge \gamma) \equiv (\phi \vee \psi) \wedge (\phi \vee \gamma) \tag{4.17}$$

$$\phi \wedge (\psi \vee \gamma) \equiv (\phi \wedge \psi) \vee (\phi \wedge \gamma) \tag{4.18}$$

Übungsbeispiel 23: überzeuge dich von (1.14) und (1.15), indem du die zugehörigen Wahrheitstafeln aufstellst.

Übungsbeispiel 24: Seien a, b, c und d logische Atome und sei

$$\phi = (a \vee b) \wedge (c \vee d)$$

$$\psi = (b \wedge d) \vee (a \wedge c) \vee (b \wedge c) \vee (a \wedge d)$$

Zeige, dass wenn du mit ϕ startest, durch schrittweise Umformung bei der Formel ψ ankommen kannst.

Bevor wir auf Umformungen eingehen, die zusätzlich zum logischen Und und Oder auch die Negation berücksichtigen, soll noch das Konzept der **Idempotenz** vorgestellt werden. Eine Formel durch Und oder Oder verknüpft mit sich selbst ist gleich sich selbst. Sei ψ eine aussagenlogische Formel, dann gilt.

$$\psi \wedge \psi \equiv \psi \tag{4.19}$$

$$\psi \vee \psi \equiv \psi \tag{4.20}$$

Diese Eigenschaft ist eine große Hilfe, wenn man lange komplexe Formeln vor sich hat und auf eine einfachere Form bringen will. Wir werden im folgenden Kapitel mehr nützliche Vereinfachungen dieser Art kennen lernen.

4.3 Umformungen der Verneinung

Bisher haben wir ausschließlich Umformungen bezüglich der logischen Verknüpfungen Und und Oder kennen gelernt. Die wichtigsten Umformungen an der die Negation beteiligt ist und eine Verbindung mit dem logischen Und und Oder herstellt sind die DeMorganschen Gesetze.

Die **DeMorganschen Gesetze** besagen vereinfacht gesagt, dass man die Negation aus einer Formel herausheben kann, wobei sich ein logisches Und in ein Oder umwandelt und ein logisches Oder in ein Und.

Seien nun ϕ und ψ aussagenlogische Formeln, dann gilt das erste DeMorgansche Gesetz.

$$\neg(\phi \vee \psi) \equiv (\neg(\phi)) \wedge (\neg(\psi)) \tag{4.21}$$

und das zweite DeMorgansche Gesetz.

$$\neg(\phi \wedge \psi) \equiv (\neg(\phi)) \vee (\neg(\psi)) \tag{4.22}$$

Übungsbeispiel 25: Vergewissere dich von der Gültigkeit von (1.21) und (1.22) indem du die Wahrheitstafeln der Formeln links und rechts des $\equiv$ aufstellst.

Um diesen Abschnitt abzuschließen seien noch zwei wichtige Methoden zur Vereinfachung von logischen Formeln erwähnt, nämlich die **Elimination**

der Doppelten Negation, die besagt, dass wenn in einer Formel zwei Negationen hintereinander vorkommen, beide gestrichen werden können,

$$\neg(\neg(\phi)) \equiv \phi \tag{4.23}$$

und das Konzept der Tautologie. Eine Tautologie ist eine Formel, die unter beliebiger Belegung der logischen Atome den Wahrheitswert Eins ergibt. Im Gegensatz dazu ist eine Kontradiktion eine Formel, die unter jeder Belegung Null als Wahrheitswert ergibt. Tautologien sind ein nützliches Konstrukt aus dem sich die gesamte Aussagenlogik konstruieren lässt wenn man so will. Als Methode zur Vereinfachung von aussagenlogischen Formeln, kann man sich folgende Sachverhalte zu Nutzen machen.

Lemma 4.6. *Sei ϕ eine aussagenlogische Formel und ψ eine Tautologie dann gilt.*

1. *$\neg(\psi)$ is eine Kontradiktion.(Die Umkehrung gilt auch)*
2. *$\phi \wedge \psi \equiv \phi$*
3. *$\phi \vee \psi$ ist eine Tautologie.*

Übungsbeispiel 26: Beweise den Satz der Elimination der Doppelten Negation (1.23) durch Aufstellen der Wahrheitstafeln.

Übungsbeispiel 27: Zeige durch Anwendung der DeMorganschen Gesetze auf Punkt 2 in Lemma 1.13, dass $\phi \wedge \psi$ eine Kontradiktion ist, wenn ψ eine Kontradiktion ist.

4.4 Die disjunktive Normalform

Wir haben bislang gelernt, was aussagenlogische Formeln sind, wie wir sie umformen und wie wir Formeln mit Wahrheitstafeln darstellen können. In diesem Abschnitt lernen wir, wie man zu beliebigen Wahrheitstafeln, eine aussagenlogische Formel konstruieren kann, welche die Wahrheitstafel erfüllt. Da es aber immer mehr als eine Formel gibt, welche dieselbe Wahrheitstafel hat, muss die zu einer Wahrheitstafel konstruierte Formel eine spezielle Form haben, eine spezielle Darstellung, eine so genannte Normalform.

Eine Wahrheitstafel der Länge n ist, wie wir im Abschnitt Wahrheitstafeln gesehen haben, eine tabellarische Darstellung einer eindeutigen Zuordnung von Null oder Eins zu beliebigen Folgen von Null und Eins fester Länge n. In anderen Worten eine Tabelle in der links alle möglichen Kombinationen von Null und Eins der Länge n einmal stehen und rechts der eindeutige zugeordnete Wert, Null oder Eins.

Hier ein Beispiel einer Wahrheitstafel der Länge 3.

0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Welcher aussagenlogischen Formel von drei logischen Atomen, nennen wir die Atome ϕ_1, ϕ_2 und ϕ_3, entspricht diese Wahrheitstafel?

Starten wir damit die Frage zu beantworten, indem wir für jeden Einser auf der rechten Seite eine aussagenlogische Formel konstruieren, die genau dann wahr ist, wenn die korrespondierende Kombination aus Nullen und Einsen links erfüllt ist. Wir haben vier Einser links und wir nennen die unbekannten Formeln, die noch zu konstruieren sind ψ_1, ψ_2, ψ_3 und ψ_4.

Es sollen also folgende Wahrheitstafeln für ψ_1 bis ψ_4 gelten.

ϕ_1	ϕ_2	ϕ_3	ψ_1	ϕ_1	ϕ_2	ϕ_3	ψ_2	ϕ_1	ϕ_2	ϕ_3	ψ_3	ϕ_1	ϕ_2	ϕ_3	ψ_4
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	1	0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
0	1	1	0	0	1	1	0	0	1	1	1	0	1	1	0
1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	1
1	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0

Wenn wir nun

$$\psi = \psi_1 \vee \psi_2 \vee \psi_3 \vee \psi_4 \tag{4.24}$$

betrachten, erschließt sich sofort, dass die zu ψ gehörende Wahrheitstafel eben die Wahrheitstafel erfüllt, die oben gegeben ist. Wegen diesem Verhalten nennt man das logische Oder auch "logische Addition".

Es fehlt also bloß Formeln zu finden für die ψ_1 bis ψ_4 stehen, also Formeln, welche die zugehörigen Eigenschaften erfüllen.

Fangen wir mit ψ_1 an. ψ_1 soll nur dann war sein, wenn die Atome ϕ_1, ϕ_2 und ϕ_3 alle den Wahrheitswert Null tragen. Andersherum betrachtet bedeutet dies, dass die Verneinungen von ϕ_1 bis ϕ_3 alle Eins beziehungsweise

wahr sein müssen damit die gesuchte Formel wahr ist.
Der einfachste Weg dies zu erreichen ist[1]

$$\psi_1 = (\neg(\phi_1)) \wedge (\neg(\phi_2)) \wedge (\neg(\phi_3)) \tag{4.25}$$

Fahren wir fort mit ψ_2; Wir verlangen von ψ_2, dass es genau dann wahr ist wenn ϕ_1 sowie ϕ_2 den Wahrheitswert Null tragen, also falsch sind, und ϕ_3 den Wahrheitswert Eins hat. Anders ausgedrückt ist ψ_2 genau dann Eins wenn die $\neg(\phi_1)$, $\neg(\phi_2)$ und ϕ_3 wahr sind.
Der einfachste Weg ist wieder die Verknüpfung dieser drei Formeln durch Und Operationen.

$$\psi_2 = (\neg(\phi_1)) \wedge (\neg(\phi_2)) \wedge (\phi_3) \tag{4.26}$$

Übungsbeispiel 28: Stelle die Wahrheitstafel der rechten Seiten der Gleichungen (3.25) und (3.26) auf um dich zu vergewissern, dass wir tatsächlich ein gültiges ψ_1 beziehungsweise ψ_2 gefunden haben.

Nach diesen beiden Beispielen erkennt man bereits das allgemeine Muster. Wenn eine Formel nur für eine bestimmte Belegung seiner logischen Atome mit Nullen und Einsen wahr sein soll, dann schreibe für jedes Atom, dass von dieser Belegung den Wert Null bekommt die Verneinung dieses Atom und für jedes andere Atom, schreiben wir das Atom selbst auf. Am Ende verknüpfen wir alles durch logische Unds.

Zeigen wir dies noch einmal an der gesuchten Formel ψ_3. Sie soll nur unter der Belegung B

$$B(\phi_1) = 0,\ B(\phi_2) = 1,\ B(\phi_3) = 1 \tag{4.27}$$

wahr sein, beziehungsweise für die Kombination in der vieren Zeile auf der linken Seite der Wahrheitstafel oben in diesem Abschnitt.

Wir schreiben somit $\neg(\phi_1)$, weil $B(\phi_1) = 0$; wegen $B(\phi_2) = 1$ und $B(\phi_3) = 1$ schreiben wir auch ϕ_2 und ϕ_2. Am Ende verknüpfen wir alles 'geschriebene':

$$\psi_3 = (\neg(\phi_1)) \wedge (\phi_2) \wedge (\phi_3) \tag{4.28}$$

Übungsbeispiel 29: Berechne ψ_3 nach dem oben beschriebenen Schema und schreibe die vollständige Formel für ψ auf.

[1]Man beachte, dass auf Klammern verzichtet wurde, da wir ja nun wissen dass die Klammerung wenn wir nur Und oder nur Oder haben egal ist, und deshalb eindeutige Lesbarkeit nicht notwendig ist.

4.4.1 NAND und NOR

Wir schließen dieses Kapitel mit einer kurzen Diskussion zweier wichtiger logischer Operationen ab. Die Besonderheit dieser Operationen ist, dass für beide gilt, dass sich jede mögliche logische Operation durch Komposition darstellen lässt.

Die erste dieser Operationen ist die Verneinung des Unds, auch **NAND**, aus dem Englischen 'not and'. Als logische Formel können wir die Operation schreiben als $\neg(\phi \wedge \psi)$ oder über die DeMorgansche Gesetze umgeformt $(\neg(\phi)) \vee (\neg(\psi))$. Die zugehörige Wahrheitstafel ist

ϕ	ψ	ψ_4
0	0	1
0	1	1
1	0	1
1	1	0

Übungsbeispiel 30: Konstruiere die disjunktive Normalform zu dieser Wahrheitstafel.

Übungsbeispiel 31: Zeige, dass die beiden Formeln äquivalent sind, indem du zeigst das beide die obige Wahrheitstafel als zugehörige Wahrheitstafel haben.

Theorem 4.7. *Jede logische Formel und somit jede mögliche Wahrheitstafel, lässt sich durch Komposition logischer Atome durch NAND Operationen darstellen.*

Beweis. Nachdem wir bereits über die Disjunktive Normalform gesehen haben, wie sich beliebige Wahrheitstafeln durch Kompositionen des logischen Unds, Oders und der Verneinung, bleibt zu zeigen, dass sich das Und, das Oder und die Verneinung als Komposition von NAND Operationen darstellen lässt. Seien ϕ und ψ logische Atome dann zeigen wir zuerst, dass sich die Verneinung darstellen lässt:

$$\neg(\phi) \equiv \neg(\phi \wedge \phi) \equiv (\phi)\text{NAND}(\phi) \tag{4.29}$$

Nun kommen wir zum Und. Durch Anwendung der doppleten Verneinung Umformung ist schnell gezeigt, dass

$$\phi \wedge \psi \equiv \neg(\neg(\phi \wedge \psi)) \equiv \neg((\phi)\text{NAND}(\psi)) \tag{4.30}$$

Da wir bereits wissen wie sich die Verneinung darstellen lässt, haben wir schon jetzt gezeigt, dass sich das Und darstellen lässt. Es fehlt noch das

Oder. Wir fangen wieder an mit doppelter Verneinung, dann DeMorgan und kommen auf

$$(4.31) \quad \phi \vee \psi \equiv \neg(\neg(\phi \vee \psi)) \equiv \neg((\neg(\phi)) \wedge (\neg(\psi))) \equiv (\neg(\phi))\text{NAND}(\neg(\psi))$$

□

Es ist nun eine leichte übung zu zeigen, dass das **NOR**, also die Verneinung des Oders, in Formel $\neg(\phi \vee \psi)$ dieselbe Eigenschaften hat.

Übungsbeispiel 32: Zeige Dies.

Kapitel 5

Elektronik

Streng genommen kann man eine Rechenmaschine bauen, die anstelle von elektrischen Schaltkreisen aus Wasserschläuchen und Ventilen oder Zahnrädern besteh. Da moderne Rechenmaschinen elektronisch sind und dies auf absehbare Zeit so bleiben wird, ist es notwendig einige wichtige Konzepte der Elektronik vorzustellen.

5.1 Elektrische Netzwerke

Wer genauere Details über die zugrundeliegende Physik wissen möchte, soll im Anhang nachlesen oder sich ein passendes Fachbuch suchen.

Das Ziel dieses Abschnittes ist den Leser mit den Grundideen elektronischer Vorgänge vertraut zu machen. Hierzu werden wir lernen, was wir in einem elektronischen System messen können, was die Bausteine eines elektronischen Systems sind und wie wir ein physikalisch realisiertes elektronisches System, wie zum Beispiel die Verkabelung in unserem Haus oder das Innenleben eines Radios als Schaltplan mit idealisierten Komponenten abstrahieren können.

In der Schule haben wir gelernt, dass ein elektrischer Stromkreis aus einer Spannungsquelle, Leiter und einem Verbraucher besteht. In einem komplexen elektrischen Netzwerk kann man mehrere Spannungsquellen, beliebig verzweigte Leiterbahnen und Verbraucher mit verschiedensten Eigenschaften haben.

Wir starten mit einem jedem bekannten Beispiel eines elektrischen Stromkreises, dem Kartoffelstromkreis.

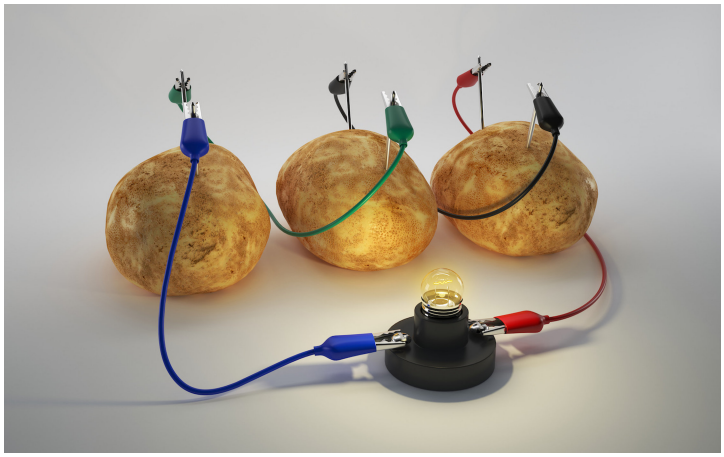


Figure 5.1: Typischer Stromkreis.

In der Abbildung sehen wir drei Kartoffelbatterien, die mit Drähten miteinander und mit einer Glühbirne verbunden sind. Dieser Stromkreis kann als Schaltplan idealisierter Komponenten abstrahiert werden. Hierbei werden irrelevante Eigenschaften der Bausteine des Stromkreises ignoriert und durch idealisierte Bauteile dargestellt. Unser Kartoffelstromkreis wird in folgendem Schaltplan abstrahiert.

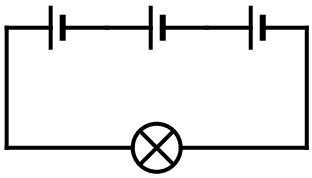


Figure 5.2: Schaltplan.

Wir ignorieren die Farben der Kabeln, die Form der Alligatorklemmen, die Abmessungen der Glühbirne die Länge und Krümmung sowie die Dicke der Kabel. Interessant ist für uns, was für Bauteile da sind, also drei Spannungsquellen (Kartoffelbatterien) und ein Verbraucher (Glühbirne) und wie diese durch Leiter (Kabel) untereinander verbunden sind.

Ein **Schaltplan** ist ein Netzwerk bestehend aus **Knoten** und **elektrischen Bauteilen**, die durch Kanten, welche elektrische Leiter darstellen, miteinander verbunden sind.
Jedes Bauteil hat eine feste Anzahl an Anschlüssen und ein Leiter endet immer an eineer Seite an einem Anschluss eines Bauteils und an der anderen Seite an einem Knoten oder einem weiteren Anschluss eines Bauteiles.

Betrachten wir ein weiteres Beispiel um dies zu illustrieren.

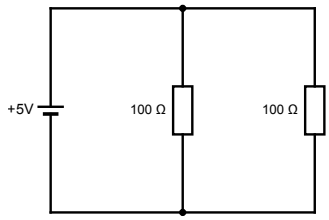


Figure 5.3: Stromkreis.

Dieser Schaltplan besteht aus drei Bauteilen, und zwar einer 5 Volt Spannungsquelle (links im Bild), und zwei 100 Ω Widerstände. Die geraden Linien sind die Kanten und diese symbolisieren Verbindungen durch elektrische Leiter. Die dicken Punkte sind die Knoten. Knoten sind Ausgangpunkte für Verzweigungen von Leitern und jede Verzweigung muss in einem Schaltplan über Knoten dargestellt werden.

Innerhalb eines elektrischen Netzwerkes können zu jedem Zeitpunkt zwei fundamentale Größen gemessen werden. Um diese beiden Größen und ihre Berechnung wird sich das Groß diese Kapitel drehen.

An jedem Punkt auf jedem der Leiter in einem elektrischen Netzwerk kann eine **elektrische Stromsträrke** gemessen werden. Oft werden wir kurz elektrischer Strom oder nur Strom schreiben. Eine gute Anschauung für diese Größe ist eine Flüssigkeit, die **elektrische Ladung**, die durch ein Rohr, den Leiter, fließt. Gemessen wird die Menge an Flüssigkeit, die in einer gewissen Zeit, den Rohrquerschnitt beziehungsweise den Punkt im Schaltplan durchfließt.

Ein Fluss hat auch immer eine Richtung. Da ein Leiter in unserem Schaltplan eine Linie ist, daher nur in eine Richtung Ausdehnung hat, kann dieser Fluß nur zwei Richtungen haben. Die Richtung wird über das Vorzeichen der Stromstärke angegeben. Daher ein elektrischer Strom kann immer nur in Bezug zu einer Messrichtung gemessen werden.

Um das positive Vorzeichen auch in Beziehung mit einer Richtung des Leiters (Kante/Linie) setzen zu können, muss dem Leiter vor der Messung[1] eine Richtung zugesprochen werden. Die Messung muss dann unter Berücksichtigung dieser Richtung vollzogen werden.

[1] Am Besten beim Design des Schaltplanes.

Das Formelsymbol für den elektrische Strom ist das I und die Maßeinheit ist das Ampere, abgekürzt A.

Die **elektrische Spannung** ist im Kontrast eine Größe, die von einem beliebigen Punkt zu einem beliebigen anderen Punkt gemessen wird. Wir sehen also, dass auch diese Größe eine Richtung hat. Die elektrische Spannung stellt im Prinzip den Antrieb des elektrischen Stromes zwischen den beiden Punkten dar, was auch erklärt warum es eine gerichtete Größe ist. Innerhalb unserer Anschauung der Flüssigkeit, die durch Rohre fließt, wäre diese Größe so etwas wie der Druckunterschied zwischen zwei Punkten. Oder noch einfacher der Höhenunterschied zwischen den Rohrabschnitten. Dies legt den Sachverhalt nahe, dass sich nur das Vorzeichen der elektrischen Spannung umdreht wenn man die Reihenfolge der Punkte an denen man misst umdreht.

Das Formelsymbol für die elektrische Spannung ist das U und die Maßeinheit ist das Volt, abgekürzt V.

Jedem zweipoligen elektrischen Bauteil, daher Bauteil mit zwei Anschlüssen, kann eine **elektrische Leistung** kurz P über die angelgte elektrische Spannung U und die das Bauteil durchfließende elektrische Stromstärke I zugeordnet werden.

$$P = U * I \tag{5.1}$$

Die elektrische Leistung hat keine besondere Bedeutung für unsere Diskussion, sollte aber der Vollständigkeit halber erwähnt werden.

5.2 Die drei Grundgesetze elektrischer Netzwerke

In diesem Abschnitt zeigen wir wie wir einen gegebenen Schaltplan in ein Gleichungsystem umwandeln können, dass uns erlaubt an jedem Punkt des elektrischen Netzwerkes den Strom und an jedem Paar von Punkten die Spannung zu berechnen.

Bevor wir uns den drei Gesetzen widmen, die Strom und Spannung in einem Netzwerk bestimmen, gehen wir kurz auf zwei Vereinfachungen ein, die beim Lesen eines Schaltplanes angenommen werden.

Wir nehmen an, dass die Kanten, die Bauteile und Knoten verbinden, den Strom und die Spannung nicht beeinfluß. Ihre Interpretation soll einzig und allein das darstellen von Verbindungen zwischen Komponenten sein. Strom und Spannungs Abhänigkeiten werden allein in die Bauteile gepresst. Wenn

man genau sein will hätten wir den Schaltplan unseres Kartoffelstromkreises so zeichnen müssen:

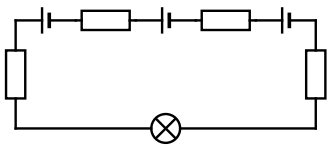


Figure 5.4: Der wahre Kartoffelschaltplan.

Wir sehen, jedem einzelnen Kabel wurde jetzt ein Bauteil zugeordnet, symbolisiert durch Rechtecke. In diesen Schaltsymbolen ist nun der Einfluß des Kabelmaterials auf Strom und Spannung versteckt.
In der Praxis lässt man dies aber oft weg, weil der Einfluß auf das Strom Spannungsverhalten von guten Leitern wie Metall nur sehr klein ist.

Praktisch bedeutet dies für die Interpretation des Schalplanes, dass wenn wir zwischen zwei Punkten die Spannung bestimmen, der Wert nicht von dem Punkten abhängt die man gewält hat, sondern von den Kanten auf denen die Punkte liegen. Daher ist die Spannung zwischen zwei Punkten derselben Spalte im Prinzip Null und wir vergleichen beim Spannungsmessen nicht Punkte sondern ganze Kanten.

Ähnliches gilt für die Interpretation von Knoten, die einzig und allein als Symbold für Verzweigungen im Netzwerk dienen. Es gilt daher, dass die Spannung zwischen zwei Kanten, die am selben Knoten anliegen gleich Null ist.

Kommen wir zurück zu unserer Anschauung des elektrischen Netzwerkes als Rohrwerk durch das eine Flüßigkeit fließt. Hieraus wollen nun das erste unserer Grundgesetze ableiten.

Betrachten wir hierzu eine Verzweigung in einem Rohrwerk. Drei Rohre, Rohr A, Rohr B und Rohr C treffen sich in einem Punkt. Es fließt nun eine bestimmte Flußigkeitsmenge X in einer gegebenen Zeit durch das Rohr A in die Verzweigung. Nachdem das Rohr vollständig mit Flüßigkeit gefüllt ist[2] und sich eine Flüssigkeit nicht verdichten lässt muss in der selben Zeit in Summe die Menge X aus Rohr B und Rohr C hinausfließen.

Das gerade beschriebene Konzept nennt sich **Kontinuitätsgesetz** und ist überall dort anzutreffen wo sich Dinge flüßigkeitsähnlich verhalten.

[2]Damit die Anschauung korrekt ist, müssen wir annehmen, dass keine Luft im Rohrwerk ist.

Zu solchen Dingen zählen neben Wasser, Benzin und Milch, auch Wärme und elektrische Ladung in einem Leiter. In der Elektronik nenn man dieses Konzept auch **Knotenregel**.
In unserer Abstraktion eines elektrischen Netzwerkes als Schaltplan bedeutet dies, zum Einen dass die Summe der in einen Knoten oder in ein elektrisches Bauteil fließenden Ströme gleich der Summe der abfließenden Ströme ist, und zum Anderen, dass auf einer gegebenen Kante jeder Punkt den selben Strom hat.

Beispiel: Nehmen wir einen Knoten mit zwei zufließenden Kanten und einer wegfließenden Kante.

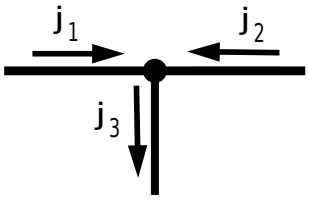


Figure 5.5: Ströme.

Der Fluß bezieht sich dabei auf die vor der Messung festgelegte Referenz Flußrichtung jeder Kante. Ein positiver Strom bedeutet also Fluß entlang der Richtung, während ein negativer Strom der Richtung entgegengesetzt ist. In diesem Fall besagt das Kontinuitätssatz:

$$i_1 + i_2 = i_3 \tag{5.2}$$

Übungsbeispiel: Formuliere das Kontinuitätsgesetz für die folgende Anordnung:

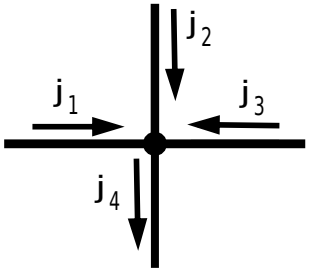


Figure 5.6: Aufgabe.

Als nächstes widmen wir uns dem zweiten Grundgesetz. So wie die Knotenregel die Struktur des Netzwerkes in Verbindung mit den elektrischen Strömen

setzt, so verbindet dieses die Netzstruktur mit den Spannungen.

Zur Motivation des neuen Gesetzes gehen wir wieder zurück zur Anschaung des flüssigkeitsdurchflossenen Röhrensystems. Wir haben ja bereits in diesem Bilde die Spannung als Höhenunterschied der Rohrabschnitte beschrieben.

Damit diese Anschauung der Spannung aber mit unserer Erfahrung zu Höhenunterschieden übereinstimmt, muss gelten, dass sie unabhängig vom Weg ist, der zurückgelegt wurde um Teilabschnitte zu vermessen. Zur Erklärung: Wir machen uns zusammen auf, einen Berg zu besteigen. Dabei starten wir beide am selben Punkt, sagen wir dem Parkplatz am Fuß des Berges. Anschließend wandern wir auf unterschiedlichen Pfaden den Berg hinauf und machen zwischendurch Halt an jeweils verschiedenen Berghütten. Am Ende treffen wir uns beide am Gipfel. Messen wir beide bei der Pause in der Hütte den Höhenunterschied zum Parkplatz und dann am Gipfel den Höhenunterschied zu der Hütte, an der wir Pause gemacht haben, dann muss die Summe dieser Werte für uns beide gleich sein. Als Konsequenz können wir offensichtlich eine absolute Höhe definieren als den Höhenunterschied zum Parkplatz (oder irgendeinem anderen festen Punkt). Jeder Höhenunterschied zwischen einem Punkt A und einem Punkt B ist dann genau gleich der Differenz der absoluten Höhe am Punkt B minus der am Punkt A.

Dieses Prinzip, dass eine Eigenschaft eines Weges, die nur abhängig vom Anfangs und Endpunkt des Weges ist, eine absolute Eigenschaft für jeden Punkt definiert, ist ein wichtiges Konzept aus der Potentialtheorie, einem Teilgebiet der Mathematik und hat in der Elektronik eine analoge Bedeutung.

In einem Schaltplan gilt dieses Konzept ganz analog für die elektrische Spannung. Man wähle zwei Kanten im Schaltplan und einen Pfad zwischen den beiden Kanten. Wenn man an der ersten Kante startet und den Pfad durchwandert und dabei die Spannungen zwischen aufeinanderfolgenden Kanten misst, dann ergibt die Summe der Spannungen sobald man bei der letzten Kante ankommt dieselbe Spannung die man zwischen der ersten und letzten Kante misst. Insbesondere bedeutet dies, dass für jeden geschlossenen Pfad die Summe dieser Spannungen Null sein muss. Diese Regel nennt man die **Maschenregel**, wobei Maschen sich auf jene geschlossenen Pfade im Schaltplan bezieht.

Theorem 5.1. *Gegeben sei ein Schaltplan, dann gilt die Maschenregel genau dann wenn es eine Funktion ψ gibt, die jeder Kante einen Zahlenwert zuordnet, sodass die gemessene Spannung zwischen der Kante x zur Kante y genau die Differenz $\psi(y) - \psi(x)$ ist.*

Eine solche Funktion ψ der Kanten nennen wir eine **Potentialfunktion** und wir werden anstelle der Maschenregel immer von der Existenz einer Potentialfunktion ausgehen, da dies eine einfachere Analyse elektrischer Netzwerke erlaubt.

Folgerung 5.2. *Eine Potentialfunktion eines Schaltplans X ist bis auf eine additive Konstante eindeutig festgelegt, daher falls ψ eine Potentialfunktion von X ist und c ein reele Zahl, dann ist $\psi + c$ auch eine Potentialfunktion von X.*

Bisher haben wir den Einfluß der Struktur des Schalplans, also des Netzwerkes an Verbindungen, behandelt. Der Einfluß der elektrischen Bauteile selbst auf Spannungen und Stromstärken wurde bislang ausgelassen. Dies stellt das dritte Gesetz elektrischer Netzwerke dar.
Ein elektrisches Bauteil hat eine bestimmte feste Anzahl von Anschlüssen oder Pole an welche elektrische Leiter/Kanten befestigt/geklemmt werden können. Das Verhalten eines elektrischen Bauteiles wird durch die **Strom-Spannungs-Kennlinie** oder allgemeiner **Charakteristik** kurz Φ bestimmt. Sie legt fest welche Ströme zwischen den Anschlüssen bei gegebenen Spannungen an den Anschlüssen erlaubt sind. Die allgemeinste abstrakte Definition des Konzeptes ist:

Definition 5.3. *Eine Charakteristik Φ eines elektrischen Bauteiles mit N Anschlüssen ist eine Teilmenge des $2 * (N-1)$-dimensionalen Raumes. Wobei die ersten $N-1$ Komponenten Potentialwerte an den Anschlüssen repräsentieren und die restlichen Komponenten die entsprechenden Stromstärken.*[3]

Die allgemeine Form einer Charakteristik eines N-poligen Bauteiles ist eine Gleichung oder ein Gleichungssystem, deren Variablen die $N-1$ Potentialwerte und $N-1$ Stromsträrken sind und damit eine Teilmenge festlegt.
Wir werden bis auf wenige Ausnahmen aber davon ausgehen, dass die Charakteristik, hier im Fall eines zweipoligen Bauteiles, eine Funktion der Form

$$I = \Phi(U)$$

ist. Also eine Zuordnung die den $N-1$ Potentialwerten eindeutig die $N-1$ Ströme zuordnet.

Die Charakteristik eines Bauteiles kann direkt gemessen werden (zb.: mit einem Oszilloskop; siehe Anhang) oder typischerweise im vom Hersteller zur Verfügung gestellten Datenblatt des elektrischen Bauteiles.

[3]Die Stromstärken sind bereits wegen der Knotenregel durch $N-1$ Werte festgelegt. Da Potentiale genauso wegen ihrer Eindeutigkeit bis auf eine additive Konstante.

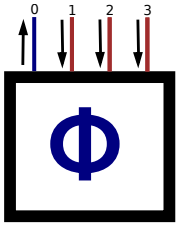


Figure 5.7: Charakteristik eines mehrpoligen Bauteiles.

Die Abbildung (4.4) zeigt eine Möglichkeit die Charakteristik eines vierpoligen elektrisches Bauteiles darzustellen. Man beachte, dass eine der Anschlüsse, Nummer 0, als Ausgang deklariert ist und diesem das Referenzpotential Null zugeordnet wird, sowie eine abfließende Stromstärke, die gleich der Summme der an den Eingängen 1, 2 und 3 zufließenden Strömen ist. Die Charakteristik Φ hat drei Komponenten für jeden Ausgang.

$$\begin{aligned} \Phi &= (\Phi^1, \Phi^2, \Phi^3) \\ i_1 &= \Phi^1(\phi_1, \phi_2, \phi_3) \\ i_2 &= \Phi^2(\phi_1, \phi_2, \phi_3) \\ i_3 &= \Phi^3(\phi_1, \phi_2, \phi_3) \end{aligned} \tag{5.3}$$

Eine **ideale elektrische Spannungsquelle** ist ein zwei poliges elektrisches Bauteil von dem unabhängig von der abgenommenen Stromstärke eine konstante Spanung abgenommen werden kann.

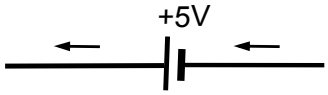


Figure 5.8: Schaltsymbol ideale Spannungsquelle.

Beachte die Pfeile in der obigen Abbildung, die nicht Teil des eigentlichen Schaltsymbols sind, sondern eine Konvention der Flußrichtung anzeigen. Die Konvention besagt, dass die längere der beiden paralellen Linien als der Ausgang angesehen wird.

Die Strom- Spannungs- Kennlinie einer 5V Spannungsquelle sieht folgendermaßen aus.

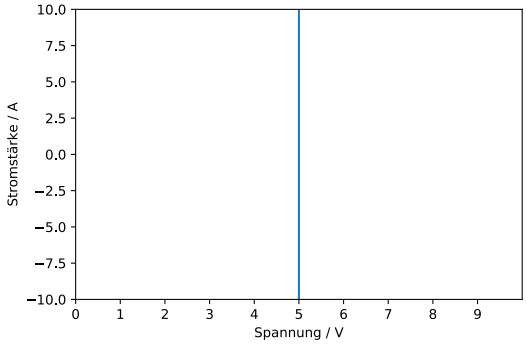


Figure 5.9: Charakteristik Spannungsquelle.

Dies ist ein Beispiel bei dem die Charakteristik keine Funktion ist sondern man sich zur Beschreibung die allgemeinere Form, eine Gleichung, bedienen muss: $U = 5$.

In unserer Flüßigkeitsanalogie des elektrischen Stromes wäre dies eine Pumpe mit konstantem Druck.
Reale Spannungsquellen, wie Zink/Kohle Batterien oder Lithium- Ionen-Akkumulatoren haben nur in einem sehr kleinen Stromstärke Bereich eine derartige vertikale Charakteristik.

Als nächstes Bauteil lernen wir die Glühbirne kennen. Eine Glübirne ist ein zweipoliges Bauteil, das Licht erzeugt wenn es mit Strom durchflossen wird.

Figure 5.10: Schaltsymbol Glühbirne.

Eine elektrische Glühbirne hat eine parabolische Charakteristik.

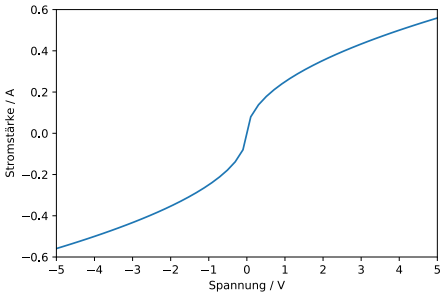


Figure 5.11: Charakteristik Glühbirne.

Man bemerke hier, dass im Gegensatz zur Spannungsquelle, hier die Charakteristik durch den Ursprung $(0, 0)$ geht. Es ist leicht zu sehen, dass Quellen diese Eigenschaft haben müssen um ihre Funktion zu erfüllen. Außerdem ist die Charakteristik symmetrische im Sinne von

$$\Phi(-U) = -j.$$

Daher sie sieht gleich aus wenn man an der $x-$Achse und anschließend and der $y-$Achse spiegelt. Bei Charakteristiken mit dieser Symmetrie ist es egal welchen der Pole man als Eingang und welchen als Ausgang zählt. Ein derartiges Bauteil nennt man auch ungerichtet. [4]

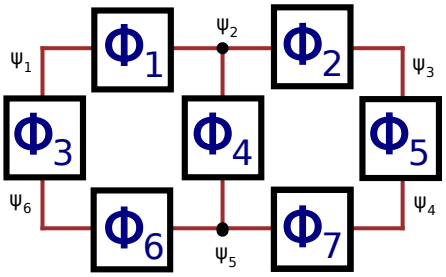
Übungsbeispiel: Überzeuge dich davon, dass es tatsächlich keinen Unterschied macht welcher Pol als Eingang gesehen wird.

5.2.1 Zusammenfassung

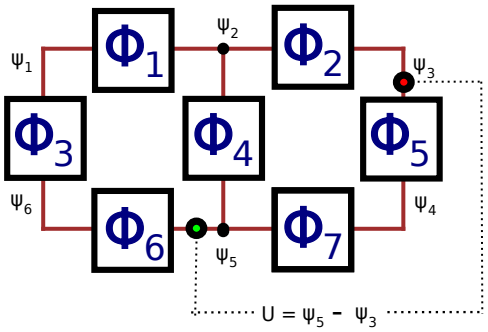
Gegeben sei ein elektrischer Stromkreis bestehend aus Knoten und elektrischen Bauteilen verbunden durch Kanten, welche elektrische Verbindungen beziehungsweise Leiter darstellen.

Hier ein Beispiel eines abstrakten elektrischen Stromkreises.

[4] In der Regel erkennt man, dass ein Bauteil ungerichtet ist wenn das Schaltsymbol symmetrisch ist.

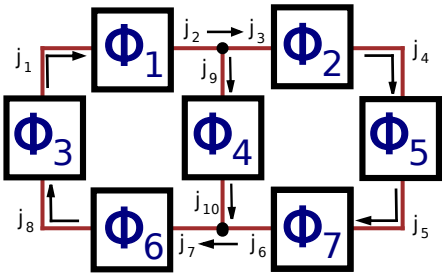


Die Kästchen symbolisieren die elektrischen Bauteile, mitsamt ihrer Charakteristik Φ. Die Kanten wurden zur besseren Lesbarkeit rot angefärbt. In der Mitte gibt es oben und unten zwei Knoten, wieder symbolisiert durch einen dicken schwarzen Punkt.
Gesucht ist nun eine Potentialfunktion ψ, die jeder Kante einen Potentialwert zuordnet, und somit jedem elektrischen Bauteil eindeutig die elektrische Spannung an jedem der Paare von Eingägen.
Kanten, welche direkt über Knoten verbunden sind, haben den selben Potentialwert, da wir ja festgelegt haben, dass zwischen darauf liegenden Punkten die Spannung Null ist.
Somit haben wir sechs Bereiche an denen die Potentialfunktion konstanten Wert hat; im Bild angezeigt durch $\{\psi_1, \psi_2, \psi_3, \psi_4, \psi_5, \psi_6\}$.



Hier wird gezeigt wie die Spannung U von einem Punkt bei ψ_3 zu einem Punkt an ψ_5 gemessen und mit Hilfe der Potentialfunktion berechnet werden kann

Als nächstes betrachten wir die elektrischen Stromstärken die innerhalb des Stromkreises auftreten und die Flußrichtungen:



Die Charakteristiken der elektrischen Bauteile wiederum legen für jede Kante eindeutig die elektrische Stromstärke fest. Bedenke dabei, dass immer der zufließende Strom in unserer Definition einer Charakteristik betrachtet wird und immer in Bezug gesetzt zur Differenz des Potential des Einganges der betrachtet wird minus dem Potentials am Ausgang. Zustzlich vergess nicht, dass die Flussrichtungen im Netzwerk nicht mit den vorher fr die Bauteil Anschlüsse festgelegten Richtungen übereinstimmen muss.

$$
\begin{aligned}
j_1 &= \Phi_1(\psi_1 - \psi_2) \\
j_3 &= \Phi_2(\psi_2 - \psi_3) \\
j_8 &= \Phi_3(\psi_6 - \psi_1) \\
j_9 &= \Phi_4(\psi_2 - \psi_5) \\
j_4 &= \Phi_5(\psi_3 - \psi_4) \\
j_7 &= \Phi_6(\psi_5 - \psi_6) \\
j_5 &= \Phi_7(\psi_4 - \psi_5)
\end{aligned}
\tag{5.4}
$$

Wir sehen, dass viele der Stromstärken bereits direkt über die Potentialfunktion und die Charakteristiken gegeben sind.
Die Knotenregel erlaubt es uns den Rest der Ströme zu bestimmen. Wir können diese direkt aus dem Bild ablesen. Beachte die jeweilige Flußrichtung.

$$
\begin{aligned}
j_1 &= j_2 \\
j_2 &= j_3 + j_9 \\
j_3 &= j_4 \\
j_4 &= j_5 \\
j_5 &= j_6 \\
j_9 &= j_{10} \\
j_7 &= j_8 \\
j_8 &= j_1 \\
j_{10} + j_6 &= j_7
\end{aligned}
\tag{5.5}
$$

Wir sehen also, dass wenn eine Potentialfunktion gegeben ist, bereits alle meßbaren Größen im Stromkreis, daher Spannungen und Stromstärken, eindeutig bestimmt sind.[5]

Kombinieren wir, die aus der Knotenregel abgeleitet Gleichungen (4.4), und die Gleichungen (4.3), die aus den Charakteristiken bestimmt wurden, so erhalten wir ein Gleichungssystem dessen Variablen die mglichen Werte der Potentialfunktion des Stromkreises darstellen. Dies ermöglicht es uns die zu einem Stromkreis gehörende unbekannte Potentialfunktion zu bestimmen. Wir werden das im Detail im Kapitel über lineare Stromkreise zeigen.

Abschließend ist zu bedenken, dass wir eine Reihe von Idealisierungen in unserer Beschreibung getroffen haben. Ein wesentlicher Aspekt dabei ist, dass wir von einem statischen elektrischen Stromkreis ausgehen. Weder die Stromstärke noch die Spannung sind abhängig von der Zeit. In einem realen elektrischen Stromkreis, braucht die Spannung so wie der Strom eine gewisse Zeit bis er bei Anschluß der Spannungsquelle von der einen Seite des Stromkreises zur anderen gewandert ist. Zusätzlich gibt es elektrische Bauteile, die bisher noch nicht erwähnt wurden, die eine interne Dynamik haben, daher zu einem Stromkreis mit zeitabhängigen Strom und Spannungswerten führen. Ein weiterer nicht unwesentlicher Aspekt ist der Einfluß der Umwelt auf den elektrischen Stromkreis. Hierzu gehören die Temperatur, elektromagnetische Felder, Feuchtigkeit etc. Auf der anderen Seite werden einige von diesen Umwelteigenschaften, wie Temperatur und Elektromagnetismus von unserem Stromkreis beeinflußt. Normalerweise wird versucht solche Einflüße so klein wie möglich zu halten, aber oft lässt es sich nicht vermeiden und darf nicht vergessen werden.

5.3 Lineare elektrische Netzwerke

Eine wichtige Erkenntnis in der Elektronik ist das Ohm'sche Gesetz, welches besagt, dass bei kleinen Stromstärken und Spannungen die Charakteristiken von vielen Materialen annähernd linear sind.

[5] Eindeutig nur falls die Charakteristiken eindeutige Zuordnungen sind.

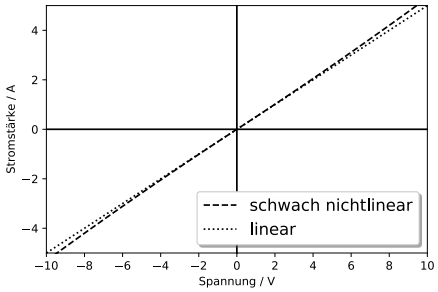


Figure 5.12: Charakteristik Widerstand.

Ein elektrisches Bauteil mit linearer Charakteristik gehorcht dem Ohm'schen Gesetz; Wenn wir nun die Charakteristik Φ des Bauteils als lineare Funktion schreiben, erhalten wir eine Form des Ohm'schen Gesetzes.

$$I = \Phi(U) = G * U \tag{5.6}$$

Wobei G, also die Steigung der Strom/Spannungskurve, **elektrische Leitfähigkeit** genannt wird.
Die gewohnte Form des Ohm'schen Gesetzes erhalten wir aber über die Spannungs/Strom Kurve:

$$U = I * R \tag{5.7}$$

Der Wert R, also die Steigung der Spannungs/Strom Kurve, ist bekannt als elektrischer Widerstand und wird gemessen in Ohm kurz Ω. Es gilt natürlich der Zusammenhang

$$G = \frac{1}{R} \tag{5.8}$$

zwischen Widerstand und Leitfähigkeit.

Das elektrische Bauteil assoziiert mit dem Ohm'schen Widerstand ist der Widerstand mit dem Schaltsymbol.

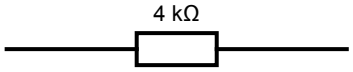


Figure 5.13: Schaltsymbol 4000 Ω Widerstand.

Eine lineare Funktion erfüllt die selbe Symmetrie wie die Glühbirne (Abbildung 4.8). die Flußrichtungen können daher beliebig gewählt werden.

Ein elektrischer Widerstand stellt einen quantitativen Zusammenhang zwischen Strom und Spannung her und kann wie der Name schn sagt als Widerstand des Bauteiles gegen die Verursachung von elektrischem Strom durch eine gegebene elektrische Spannung angesehen werden.
Innerhalb unserer Anschauung der Flüssigkeit, die sich durch Röhren bewegt, wäre dies so etwas wie ein Gewebe oder ein poröses Material, durch die die Flüßigkeit mit Druck gepresst werden muss.

5.3.1 Reihen und parallele Schaltungen

Bestimmte häufig auftretende Teilstrukturen elektrischer Stromkreise werden als elektrische Bauteile abstrahiert. Hierbei errechnet sich, abhängig von der Teilstruktur die Charakteristik des abstrahierten Bauteiles aus den Charakteristiken der darin enthaltenen Bauteile.

Gegeben sei eine Teilstruktur bestehend aus mehreren zweipoligen elektrischen Bauteilen, die nacheinander geschaltet sind, eine so genannte **Reihenschaltung**.

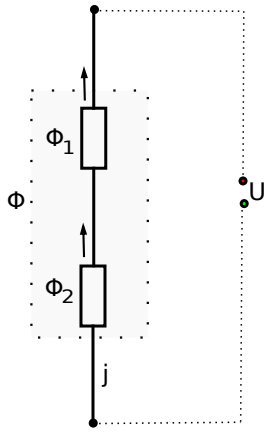


Figure 5.14: Reihenschaltung.

Hier haben wir ein Beispiel mit zwei Bauteilen, mit Charakteristiken Φ_1 und Φ_2. Man beachte das gepunkte Rechteck um die beiden Bauteile. Dies soll unser zusammengesetztes Bauteil darstellen, zusammen mit dessen Charakteristik Φ. Gesucht ist Φ. Wir wissen, dass Wegen der Knotenregel die Stromstärke überall im Stromkreis gleich sein muss, weil es ja keine Verzweigung gibt. Wir setzten den Potentialwert der oberen Kante auf Null, was wir ja dürfen, weil das Potential immer bis auf eine Konstante nur eindeutig

ist und daher ein Potentialwert frei gewählt werden darf. Die untere Kannte hat damit laut Abbildung den Potentialwert U. Es verbleibt der Potentialwert, nennen wir ihn u' der verbleibenden Kante zwischen den Bauteilen zu bestimmen.
Dieser muss erfüllen, dass $\Phi_1(u'-0) = \Phi_1(u') = j$ ist und das $\Phi_2(U-u') = j$ ist. Wenn wir zu einem gegebenen U das u' und damit j, da $j = \Phi_1(u')$ bestimmen können haben wir unsere Charakteristik Φ gefunden. Diese Gleichungsystem ist im allgemeinen nicht linearen Fall das Best, dass wir tun können. Nehmen wir an Φ_1 und Φ_2 sind linear, daher $\Phi_1(u) = u/R_1$ und $\Phi_2(u) = u/R_2$. Wobei R_1 und R_2 die Widerstände der Bauteile sind.
Das Problem vereinfacht sich stark und es ist leicht zu zeigen, dass $\Phi(u) = u/R$ ist, wobei

$$R = R_1 + R_2. \tag{5.9}$$

Übungsbeispiel: Zeige die Gültigkeit von 4.9.

Gegeben sei eine Teilstruktur bestehend aus mehreren zweipoligen elektrischen Bauteilen, die nebeneinander geschaltet sind, eine so genannte **Paralellschaltung**. In der folgenden Abbildung sehen wir ein Beispiel mit zwei paralell geschalteten Bauteilen.

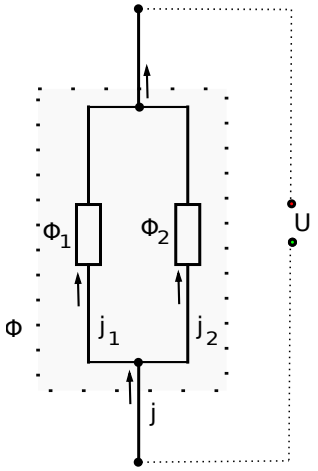


Figure 5.15: Paralellschaltung.

Wie im letzten Beispiel haben wir hier zwei Bauteile mit Charakteristiken Φ_1 und Φ_2, sowie ein gepunktetes Rechteck um diese, welches das zusammengesetzte Bauteil symbolisiert, dessen Charakteristik Φ wir suchen. Es liegt eine Spannung von U zwischen dem oberen und dem unteren Punkt an und wir suchen die zugehörige Stromstärke j. Wir legen wieder fest, dass

oben, daher der gesamte durch den Knoten verbundene Bereich oberhalb der Bauteile, den Potentialwert Null hat und analog unten der Potential wert U. Nun liegt an beiden Bauteilen dieselbe Spannung U an, daher produzieren sie jeweils auf ihre Leiterverzweigung den zugehörigen Strom $j_1 = \Phi_1(U)$ und $j_2 = \Phi_2(U)$. Nach der Knotenregel ergibt sich $j = j_1 + j_2$. Es gilt also

$$\Phi(U) = \Phi_1(U) + \Phi_2(U) \tag{5.10}$$

Es ist leicht zu sehen, dass wenn wir annehmen, dass unsere Bauteile lineare Charakteristik haben, daher $\Phi_1(U) = U/R_1$ und $\Phi_2(U) = U/R_2$ ist, dann ist $\Phi(U) = U/R$, wobei R sich errechnet aus

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2}$$

5.3.2 Lineare Systeme

Die Gleichungen in (4.3) reduzieren sich im Fall von Bauteilen mit linearer Charakteristik zu linearen Gleichungen und mit Hilfe von (4.4) erhalten wir ein lineares Gleichungssystem, das mit klassischen Methoden wie dem Gaußschen Eliminationsverfahren gelöst werden kann.
Wir zeigen in diesen Abschnitt wie dies an einem konkreten Beispiel funktioniert und dass wir im Fall von zweipoligen linearen Bauteilen immer eine eindeutige Lösung haben.

Gegeben sei ein elektrischer Stromkreis bestehend aus einer quadratischen Anordnung von Widerständen. Die Kanten des Quadrats sind Widerstände, die Ecken sind Knoten. Eine der Diagonalen ist ein Widerstand, die Andere ist eine ideale Spannungsquelle.

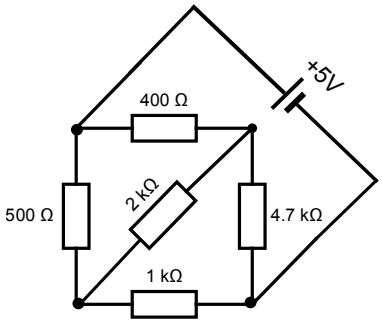


Figure 5.16: Beispiel.

Wir starten mit unserer Analyse des Stromkreises indem wir für jede Kante eine Vorzugsrichtung auswählen und die zugehörigen Ströme benennen. Die

an der Spannungsquelle liegenden Kanten haben bereits eine Vorzugsrichtung (4.5), und für alle an Widerstände grenzenten Kanten, kann die Flußrichtung beliebig gewählt werden (4.10), solange beachtet wird, dass jeder Widerstand eine zufließende und eine abfließende Kante haben muss.

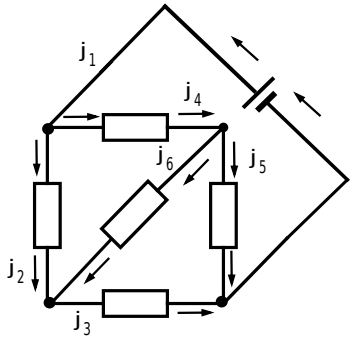


Figure 5.17: Ströme und Richtungen.

Um ein wenig mit den Variablen zu sparen, haben wir hier beiden an ein Bauteil geschlossenen Kanten denselben Strom zugeordnet, weil sie ja laut Knotenregel ohnehin gleich sind.

Als nächstes ordnen wir jedem durch Knoten verbundenen Bereich von Kanten und einzelnen nicht an Knoten grenzende Kanten eine Potentialvariable zu. Die Potentialvariablen stehen für die unbekannten Potentialwert an den entsprechenden Bereichen.

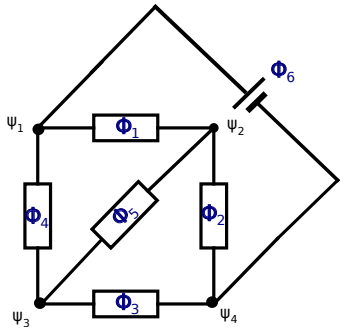


Figure 5.18: Potentialvariablen.

Anstelle von Spannungswerten bei der Spannungsquelle und Widerstandswerten bei den elektrischen Widerständen haben wir Φ_x für die Charakteristiken der Bauteile geschrieben. Dies geschah damit wir zu Anfangs im selben Schema wie zuvor arbeiten.

Die Charakteristiken geben uns non folgende Gleichungen.

$$
\begin{aligned}
\psi_1 - \psi_4 &= 5V \\
j_2 = \Phi_4(\psi_1 - \psi_3) &= \frac{\psi_1 - \psi_3}{500\Omega} \\
j_3 = \Phi_3(\psi_3 - \psi_4) &= \frac{\psi_3 - \psi_4}{1000\Omega} \\
j_4 = \Phi_1(\psi_1 - \psi_2) &= \frac{\psi_1 - \psi_2}{400\Omega} \\
j_5 = \Phi_2(\psi_2 - \psi_4) &= \frac{\psi_2 - \psi_4}{4700\Omega} \\
j_6 = \Phi_5(\psi_2 - \psi_3) &= \frac{\psi_2 - \psi_3}{2000\Omega}
\end{aligned}
\tag{5.11}
$$

Beachte hierbei die erste Gleichung, welche wir erhalten da sich wie schon gezeigt die Charakteristik der idealen Spannungsquelle nicht als Funktion schreiben lässt. Links stehen die Stromstärken; in der Mitte dann die Charakteristik als Funktionensymbol und rechts dann die Definition der linearen Charakteristik ausgeschrieben. Wir sehen, dass es keine Gleichung für j_1 gibt, aber dies wird sich ändern sobald wir die Knotenregeln aufschreiben.

Wir wissen bereits, dass Potentialfunktionen immer nur bis auf eine Konstante festgelegt sind. Das bedeutet wir können einen der Potentialwerte frei wählen und es empfliehlt sich bei nur einer Spannungsquelle den Minuspol auf Null zu setzen. In unserem Fall ist das ψ_4. Man sagt auch diese Kante ist **Erde** und man kann diesen Sachverhalt im Schaltplan folgendermaß darstellen.

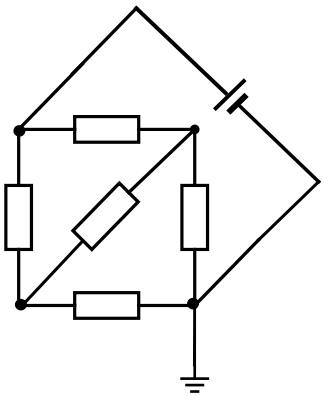


Figure 5.19: Erde.

Sobald wir $\psi_4 = 0$ gesetzt haben folgt aus (4.11), dass $\psi_1 = 5V$ ist. Wir haben also nur zwei Unbekannte ψ_2 und ψ_3, welche im weiteren Verlauf

dieser Analyse noch eindeutig bestimmt werden.

Als nächstes kommen wir zu den Gleichungen, die aus der Knotenregel resultieren.

$$j_1 = j_2 + j_4 \tag{5.12}$$

$$j_2 + j_6 = j_3 \tag{5.13}$$

$$j_3 + j_5 = j_1 \tag{5.14}$$

$$j_4 = j_6 + j_5 \tag{5.15}$$

Die unbekannten Potentialvariablen bestimmen über die Charakteristiken die Ströme. Die Ströme wiederum stehen über die Knotenregel in Beziehung zu einander. Daher können wir durch Einsetzen der rechten Seiten in (4.11) in die Gleichunge (4.12-4.15) ein Gleichungssystem der unbekannten Potentialvariablen aufstellen.

Bevor wir Einsetzen beachte, dass es zwei Unbekannte gibt und vier Gleichungen. Ein lineares Gleichungssystem kann nur dann eindeutig lösbar sein, wenn die Anzahl der Gleichungen gleich der Anzahl der Unbekannten ist.
Wir können nicht einfach zwei Gleichungen rausnehmen und uns damit begnügen, weil wir damit Information ignorieren könnten.
Der korrekte Weg ist es das Gleichungsystem (4.12-4.15) durch Umformung in ein logisch äquivalentes System mit möglichst wenigen Gleichungen umzuformen.
Der allgemeine Weg dies zu tun, ist das Gaußschen Eliminationsverfahren anzuwenden, aber wir werden einen individuelleren Weg gehen.

Der Strom j_1 kommt nicht in (4.11) vor, aber kommt dafür in (4.12) und (4.14) vor. Die einzige Anforderung, die j_1 an die übrigen Ströme damit stellt um aus ihnen bestimmbar zu sein erhalten wir durch Gleichsezten der rechten Seite von (4.12) mit (4.14)

$$j_2 + j_4 = j_3 + j_5 \tag{5.16}$$

Damit haben wir (4.12) und (4.14) auf eine Gleichung reduziert, aber es kommt noch besser, denn wenn wir (4.13) und (4.15) addieren kommen wir auf genau dasselbe Ergebnis (4.16).

$$(j_2 + j_6) + (j_4) = (j_3) + (j_6 + j_5)$$

Als Konsequenz können wir Gleichungen (4.12) und (4.14) streichen und durch Einsetzen von (4.11) in (4.13) und (4.15) erhalten wir unser gesuchtes

Gleichungssystem.

$$\begin{aligned} \frac{5V-\psi_3}{500\Omega}+\frac{\psi_2-\psi_3}{2000\Omega} &= \frac{\psi_3}{1000\Omega} \\ \frac{5V-\psi_2}{400\Omega} &= \frac{\psi_2-\psi_3}{2000\Omega}+\frac{\psi_2}{4700\Omega} \end{aligned} \tag{5.17}$$

Es ist nun eine einfache Aufgabe dieses Gleichungssystem aufzulösen. Die Stromstärke j_1 ist zum Abschluss noch mit Hilfe von (4.12) zu bestimmen.

Übungsaufgabe: Stelle das Gleichungssystem für folgenden Schaltkreis auf.

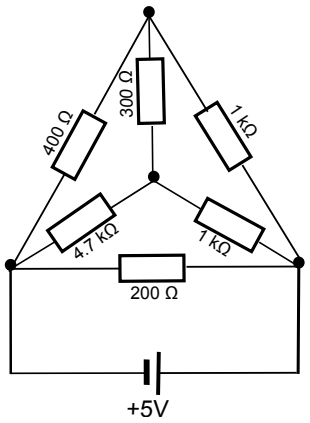


Figure 5.20: Übungsbeispiel.

5.4 Dynamische Stromkreise

Bisher haben wir die Annahme getroffen, dass Strom und Spannung in einem elektrischen Netzwerk sich nicht mit der Zeit verändert. Diese Annahme ist allerdings bis auf Netze, die sich nach dem Einschalten in einem konstanten Gleichgewichtzustand einfinden in physischen Netzwerken nie gegeben. Nach dem Einschalten eines elektrischen Gerätes braucht es immer eine gewisse Zeit bis die Spannung und der Strom von den Quellen zu den Bauteilen gelangt sind. Es gibt Anordnungen von elektrischen Bauteilen, die Schwingungen unterschiedlicher Formen erzeugen, sowie einen Effekt der bei Abschalten der Spannungsquelle beziehungsweise des Stromes dazu führt, dass dieser nicht sofort wegfällt sondern langsam abklingt. All diese Effekte werden in diesem Abschnitt anhand von idealisierten Bauteilen erklärt. Abschließend werden Halbleiterbauteile wie Dioden und Feldeffekttransistoren behandelt, die uns dann ermöglichen im letzten Abschnitt dieses Kapitels die CMOS Technologie und darauf basierende logische Gatter zu verstehen.

5.4.1 Der Kondensator

Ein Kondensator ist ein elektrisches Bauteil, dass Strom in eine dem Strom entgegengerichete Spannung umwandelt. Wir wollen in diesem Abschnitt nicht auf den Aufbau oder unterliegende physikalische Prinzip ein, sondern beschränken uns auf eine Beschreibung basierend auf Strom und Spannung allein. Wer mehr wissen will kann über die darunterliegende Physik im Anhang lesen oder in einem entsprecheden Fachbuch.

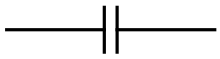


Figure 5.21: Schaltsymbol eines Kondensator.

Betrachten wir einen Schaltplan in dem ein Kondensator über einen Widerstand und eine Glühbiren an eine Spannungsquelle geschlossen ist.

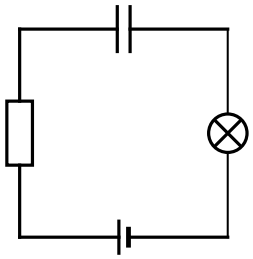


Figure 5.22: Stromkreis mit Kondensator.

Eine mögliche Realisierung dieses Schaltplanes wäre eine Batterie die anuf einem Pol an einen Kondensator und am anderen Pol über eine Glühbirne am anderen Pol des Kondensators geschlossen ist.
Sobald wir den Stromkreis geschlossen haben wird entsprechend der Spannung an der Quelle, dem Widerstand der Leiter und der Glühbirne ein Strom im Uhrzeigersinn durch den Stromkreis fließen. Der Kondensator wird diesen Strom in eine dem Strom und somit der Spannungsquelle (unten) entgegengerichtete Spannung umwandeln. Umso mehr Strom geflossen ist umso höher wird diese umgekehrte Spannung und umso kleiner wird der Strom, da der Kondensator die an den Bauteilen anliegende Spannung verringert. Dies geht so lange bis der Kondensator betragsmässig dieselbe Spannung wie die Spannungsquelle hat und der Strom Null wird. Wir beobachten ensprechend beim Schließen des Stromkreises ein helles Aufleuchten der Glühbirne, das schnell dimmt und verschwindet.

Entfernen wir nun die Spannungsquelle aus dem Stromkreis, und schließen den Kondensator direkt an die Glühbiren, sehen wir dasselbe Resultat. Die

Glühbirne leuchtet auf, dimmt schnell und erlischt.

Ein Kondensator speichert also Strom als Spannung, ähnlich wie eine aufladbare Batterie. Er hat daher einen Zustand, nämlich seine Spannung.
Die Charakteristik eines Kondensators entspricht der einer idealen Spannungsquelle, deren Spannung von der Summe, der durch Ströme, zugeflossenen elektrischen Ladung abhängt. Hierbei tragen Ströme in eine Richtung als positiver Beitrag und Ströme in die andere Richtung als negativer Beitrag zur Summe bei.
Wir werden daher jedem Kondensator, genauso wie einer Kante, eine Vorzugsrichtung zuordnen. Die Charakteristik soll einer idealen Spannungsquelle mit Plus Pol in Vorzugsrichtung entsprechen.

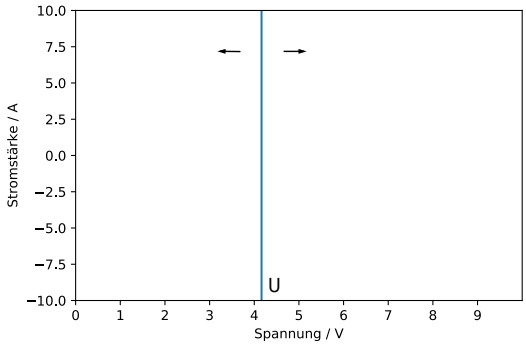
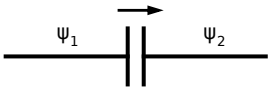


Figure 5.23: Charakteristik variable Spannungsquelle

Jedem Kondensator in einem elektrischen Netz wird ein Zustand, seine Spannung U zugeordnet. Um diese Spannung zu interpretieren müssen eine Vorzugsrichtung und anliegende Potentiale gegeben sein



Wir schreiben nun, unter Beachtung der Vorzugsrichtung.

$$\psi_2 - \psi_1 = U$$

Dies entspricht genau der Gleichung, die wir für eine ideale Spannungsquelle mit Pluspol in Vorzugsrichtung erhalten würden. Es wird ein linearer Zusammenhang zwischen dem Differentialquotienten der Spannung U nach der Zeit t und dem Strom j angenommen.

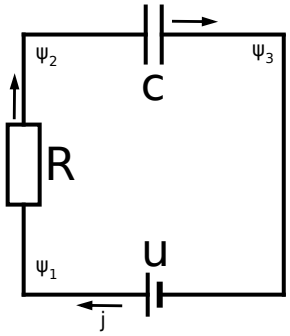
$$\frac{dU}{dt} = -\frac{j}{c} \tag{5.18}$$

Das negative Vorzeichen berücksichtigt, dass die durch den Strom aufgebaute Spannung im Kondensator immer gegen den Strom wirkt.Die Konstante c nennt man auch die Kapazität des Konsensators. Sie gibt an wie viel elektrische Ladung q zufließen muss um eine gegebene den Strom antreibende Spannung zu erreichen.

$$c = \frac{q}{U} \tag{5.19}$$

Als Konsequenz von (4.18) wird das zu einem Schaltplan gehörende Gleichungssystem zu einem Differentialgleichungssystem. Also zu einem System von Gleichungen, dass nicht einen Satz von reelen Zahlenwerten für jede Potentialvariable festlegt, sondern Funktionen, die jedem Zeitpunkt einen reelen Zahlenwert zuordnet.

Wir werden im Rest dieses Abschnittes das in Worten beschriebene Experiment zu Abbildung (4.22) nachrechnen. Hierzu vereinfachen wir den Stromkreis indem wir die Glühbirne und den Widerstand zu einem einzigen Widerstand verschmelzen. Dies führt zum selben Ergebnis, wenn wir annehmen, dass die Glühbirne eine lineare Charakteristik hat und der resultierende Widerstand nach der Reihenschaltungsformel(4.9) berechnet wird.



In der Abbildung wurden bereits alle Vorzugsrichtungen, Potentiale und Ströme, sowie Paramter c, R und u eingezeichnet. Als Gleichungssystem ausformuliert bedeutet dies.

$$\psi_1 = \psi_3 + u \tag{5.20}$$

$$j = \frac{\psi_1 - \psi_2}{R} \tag{5.21}$$

$$\frac{d(\psi_3 - \psi_2)}{dt} = -\frac{j}{c} \tag{5.22}$$

Wir werden wie gehabt annehmen, dass $\psi_3 = 0$, wodurch $\psi_1 = u$ ist und wir das Problem auf eine Unbekannte, nämlich ψ_2 reduziert haben.

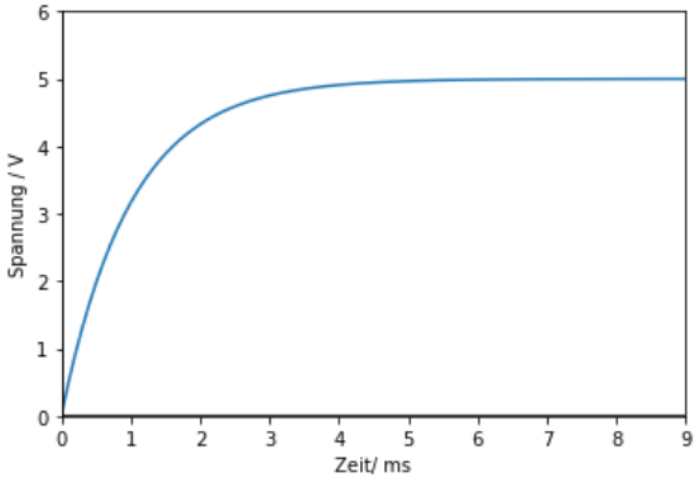
$$\frac{d\psi_2}{dt} = \frac{u - \psi_2}{Rc} \tag{5.23}$$

Dies ist eine inhomogene lineare Differentialgleichung, die beschreibt wie sich das Potential ψ_2 mit der Zeit entwickelt. Es fehlt uns nur noch ein Startwert für die Spannung am Kondensator U und wir können eine eindeutige Löesung aufschreiben.

Dies führt uns zu den Szenarien unterhalb von Abbildung (4.22). Starten wir mit dem Szenario wo wir eine Spannungsquelle an einen Kondensator schließen. Der Kondensator ist zu Beginn ungeladen also $U(0) = 0$. Dies führt zur eindeutigen Lösung:

$$\psi_2(t) = u * (1 - e^{-\frac{t}{Rc}}) \tag{5.24}$$

Setzen wir den Widerstand R gleich $10k\Omega$, die Spannung u auf $5V$ und die Kapazität c gleich $10nAs/V$ und betrachten die Kurve.

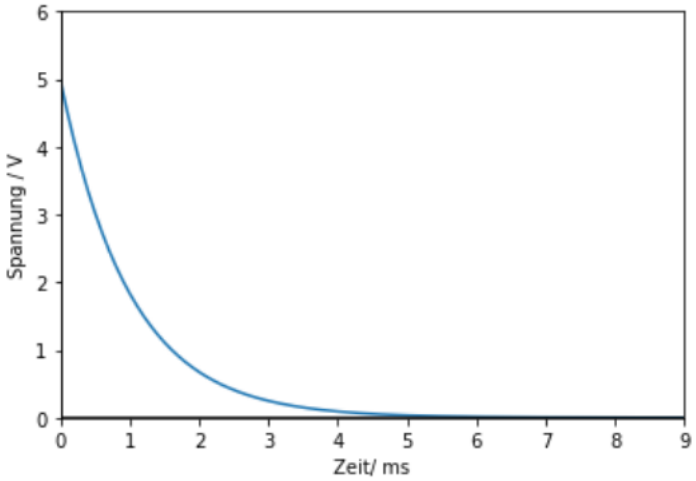


Zu Beginn steigt die Spannung sehr rasch an und es stellt sich schnell ein Gleichgewicht mit der Spannungsquelle ein wodurch nur mehr ein kleiner werdender schwacher Strom fließt.

Im zweiten Szenario nehmen wir an, dass wir den nun geladenen Kondensator von der Spannungsquelle nehmen und direkt an den Widerstand schließen. Hierzu setzen wir u einfach Null, was die Spannungsquelle aus dem Stromkreis eliminiert und setzen die Spannung des Kondensators U gleich $5V$.

$$\psi_2(t) = 5 * e^{-\frac{t}{Rc}} \tag{5.25}$$

Wir setzen die Paramter c und R wie gehabt und erhalten, einen exponentiellen Abfall der Spannung.



Wie im ersten Szenario folgt auf eine schnelle Änderung der Spannung eine Abflachung und Sättigung. Der Strom verhält sich entsprechend dem Ohmschen Gesetz hier analog zur Spannungskurve.

Damit haben wir die Ladesituation sowie die Entladungssituation eines Kondensators im Detail beschrieben und nachgerechnet.

In einem physikalsichen elektrischen Netzwerk verhalten sich so gut wie alle elektrischen Bauteile und somit auch die Kabelverbindungen zwischen Diesen wie ein Kondensator mit sehr kleiner Kapazität. Dies fällt kaum auf solange im Netzwerk keine all zu hoch frequenten Prozesse stattfinden. Kann aber zum Problem werden, wenn zum Beispiel unsere Rechenmaschine eine sehr hohe Taktfrequenz hat. Man spricht hier von kapazitiven Effekten und wenn wir sie berücksichtigt wollen, dann kann dies einfach gemacht werden indem man einen Kondensator in parallel mit dem ensprechenden Bauteil im Schaltplan zeichnet. Wenn zwei Bauteile kapazitiv wechselwirken, also sich zusammen wie ein Kondensator verhalten, dann verbinden wir sie über einen Kondensator mit der entsprechenden Kapazität.

5.4.2 Die Spule

Eine Spule ist ein zweipoliges elektrisches Bauteil, dass sich einer Änderung des elektrischen Stromes durch eine, der Änderung entgegen gerichtete, Spannung widersetzt. Man sagt die Stromänderung induziert eine Spannung in der Spule. Das zugrunde liegende physikalsiche Prinzip nennt man Induktion beziehungsweise Selbstinduktion und wird im Anhang näher erklärt. Wir wollen aber hier, wie bereits beim Kondensator, das Bauteil anhand der für uns interessanten Phänomene Spannung und Stromstärke beschreiben.

Figure 5.24: Schaltsymbol einer Spule.

Starten wir wieder mit einem einfachen Experiment. Wir nehmen eine Spule und verbinden sie an einer Seite mit einer Glühbirne und schlieẞen dies in parallel mit einer weiteren Glühbirne an eine Spannungsquelle. Der beschriebene Sachverhalt entspricht dem in Abbildung (4.25) dargestellten Schaltpan.

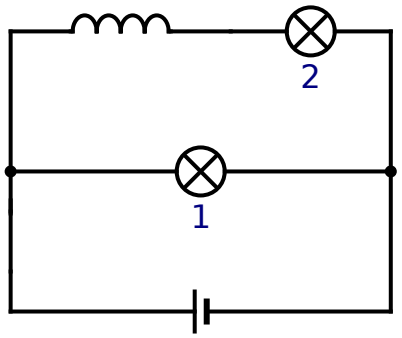


Figure 5.25: Experiment zu Induktion.

Wenn eine stark genuge Spule[6] gewählt wurde, sehen wir beim schließ der Spannungsquelle an den Stromkreis Lampe Nummer eins sofort aufleuchten, aber Lampe Nummer zwei leuchtet anfangs nur gedimmt und erreicht seine volle Leuchtstärke erst einen Moment später. Unterbrechen wir nun den Stromkreis indem wir die Spannungsquelle entfernen, beobachten wir, dass Lampe eins und zwei noch einen Moment nachleuchten.

Im Gegensatz zum Kondensator speichert eine Spule also anscheinend elektrischen Strom. Zuerst will sie ihn bei Null lassen und nachdem er seinen Sättungsstrom nach dem Ohmschen Gesetz erreicht hat, versucht es diesen beizubehalten, auch nachdem die treibende Spannung genommen wurde.

Woher kommt aber dieser Strom, nachdem die Spannungsquelle entnommen wurde? Es gelten ja noch immer die Grundgesetze elektrischer Netzwerke, also muss der Strom von einer Spannung begleitet werden. Diese Spannung nennt man induzierte Spannung und sie ist immer der Richtung der Änderung des Stromes entgegengerichtet. Wenn wir also beim Entfernen des Stromkreises die an der Spule anliegende Spannung gemessen hätten, so wäre sie der Richtung der gerade entfernten Spannungsquelle entegegenge-

[6] Was das heiß sehen wir im Laufe dieses Abschnittes.

setzt gewesen.

Analog zum Kondensator nehmen wir an, dass die Spule die Charakteristik einer idealen Spannungsquelle (4.23) mit veränderlicher Spannung U hat. Die Spule hat ebenfalls wie der Kondensator eine Zustand und zwar die durch sie fließende Stromstärke. Entsprechend wird einer Spule eine Vorzugsrichtung gegeben wie einer Kante und es gilt wie beim Kondensator.

Da wir die Spule wie eine ideale Spannungsquelle behandeln gilt wieder.

$$\psi_2 - \psi_1 = U$$

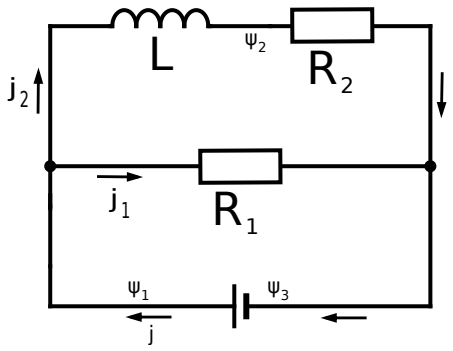
Die induzierte Spannung U hängt linear vom Differentialquotienten der Stromstärke j nach der Zeit t ab.

$$\frac{dj}{dt} = -\frac{U}{L} \tag{5.26}$$

Vergleiche Gleichung (4.26) mit (4.18) und erkenne, dass sich Kondensator und Spule analog verhalten, wobei die Rolle von Strom und Spannung vertauscht ist.

Die Konstante L nennen wir die Induktivität der Spule und sie wird in Vs/A gemessen. Genauso wie beim Fall des Kondensators erhalten wir bei einer Spule ein Differentialgleichungssystem, dass einen Satz von Funktionen für jeden Potentialwert bestimmt.

Kommen wir nun zurück zum Experiment zu Abbildung (4.25) zurück und rechnen dieses mit Hilfe der nun bekannten Gesetzmässigkeiten nach. Der Schaltplan wird hierzu wieder vereinfacht indem jede Glühbirne durch einen ohmschen Widerstand ersetzt wird.



In der Abbildung sind bereits alle Vorzugsrichtungen, Potentiale und Parameter eingezeichnet. Es verbleibt nur noch das resultierende Gleichungssystem aufzuschreiben.

Zuerst die Bauteilgleichungen:

$$\psi_1 = \psi_3 + u \tag{5.27}$$

$$j_1 = \frac{\psi_1 - \psi_3}{R_1} \tag{5.28}$$

$$j_2 = \frac{\psi_2 - \psi_3}{R_2} \tag{5.29}$$

$$\frac{dj_2}{dt} = -\frac{\psi_2 - \psi_1}{L} \tag{5.30}$$

und dann die Knotenregel:

$$j = j_1 + j_2$$

Wir setzen wieder $\psi_3 = 0$, woraus $\psi_1 = u$ folgt und erhalten durch Umformung:

$$\frac{d\psi_2}{dt} = -\frac{R_2}{L}(\psi_2 - u) \tag{5.31}$$

Dies entspricht der Differentialgleichung (4.23), die wir bei unserer Diskussion des Kondensators kennen gelernt haben. Der Anfangszustand unseres Stromkreises ist $j_2(0) = 0$. Wir haben aber unseren Stromkreis über das Potential ψ_2 beschrieben. Daher müssen wir unsere Anfangsbedingung über (4.29) umwandeln in eine die ψ_2 betrifft und zwar $\psi_2(0) = \psi_3(0) = 0$.

Unser erstes Experiment entspricht also bis auf die Konstanten genau dem ersten Experiment, dass wir beim Kondensator kennen gelernt haben. Wir können daher die Löesung (4.24) kopieren.

$$\psi_2(t) = u * (1 - e^{-\frac{R_2 t}{L}}) \tag{5.32}$$

Nehmen wir wie beim Kondensator ein konkretes Beispiel heran. Die Spule hat eine Induktivität $L = 0.1Vs/A$, $R_1 = 100\Omega$ und $R_2 = 100\Omega$ und die Spannungsquelle eine Spannung $u = 5V$.

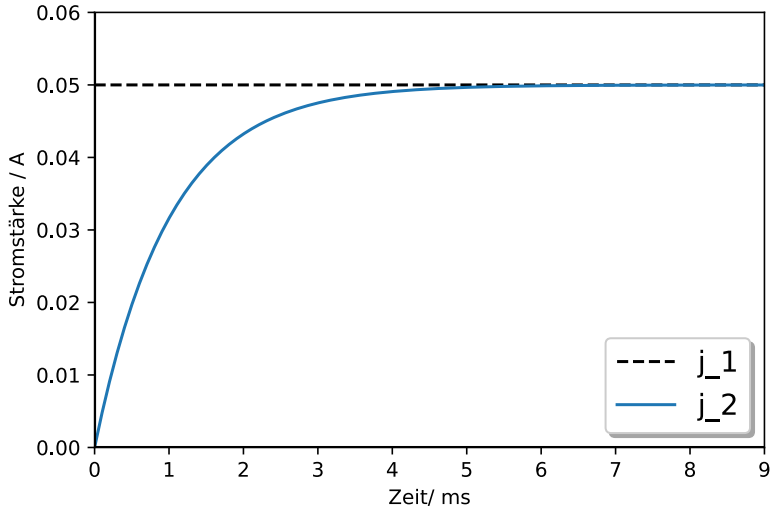
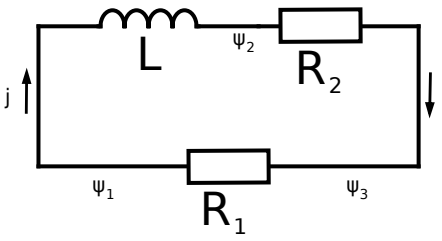


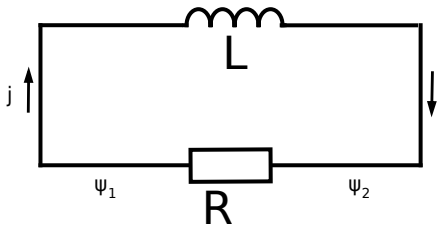
Figure 5.26: Stromstärken zu Experiment 1.

Die Abbildung zeigt die Stromstärken der Glühbirne 1 und 2. Es zeigt sich ganz analog zu unserem Experiment, dass der Strom durch Glühbirne 2 etwas länger braucht um seine volle Stärke zu erreichen. Dies entspricht einem Ladevorgang der Spule.

Kommen wir nun zum zweiten Experiment, wo wir von dem Stromkreis (4.25), nachdem beide Glühbirnen konstant leuchten, die Spannungsquelle aus dem Stromkreis entfernen. Hierzu müssen wir etwas an unseren Variablen ändern und erhalten folgenden Schaltplan, der den Sachverhalt nach der Entfernung widerspiegelt.



Im Abschnitt über Reihenschaltung haben wir gesehen, dass sich dies vereinfachen lässt, sei hierzu $R = R_1 + R_2$.



Wir haben damit beide Glühbirnen zu einem Bauteil vereinigt und erhalten folgendes Gleichungssytem, wobei wir ψ_2 schon Null gesetzt haben:

$$j = \frac{\psi_1}{R} \tag{5.33}$$

$$\frac{dj}{dt} = \frac{\psi_1}{L} \tag{5.34}$$

Durch Einsetzen erhalten wir die Differentialgleichung,

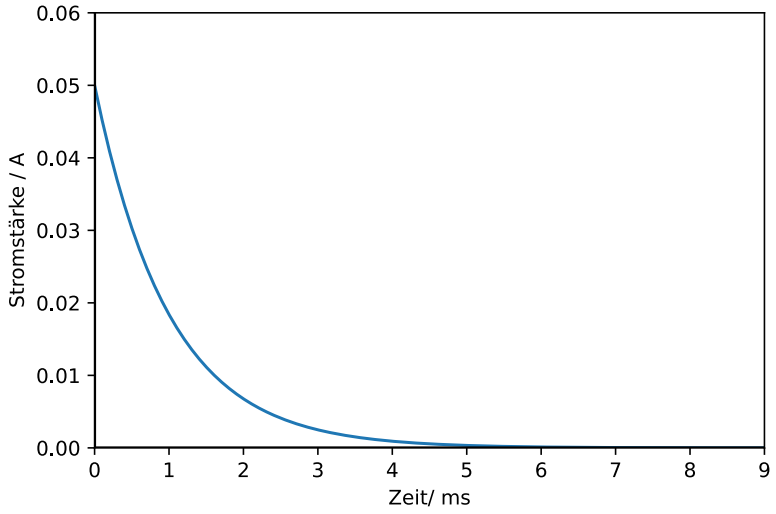
$$\frac{d\psi_1}{dt} = \frac{R}{L}\psi_1 \tag{5.35}$$

die unter der Annahme, dass $j(0) = u/R$ und somit $\psi_1(0) = u$ zur Lösung

$$\psi_1(t) = u * e^{-\frac{Rt}{L}})$$

führt.

Betrachten wir abschließend noch den Verlauf von j in diesem Experiment.



Genauso wie beim ersten Experiment, sehen wir auch beim zweiten Experiment ein analoges Verhalten zum zweiten Experiment im Kondensator Abschnitt.
Die Stromkurve fällt zunächst rasch und stellt sich dann langsam gegen Null ein. Dies entspricht genau dem Verhalten, dass wir beim Experiment beschrieben haben, denn bei $t = 0$ wird hier die Spannungsquelle entfernt und die Spule die mit einer Stromstärke von u/R_2 geladen war, entlädt sich.

Zum Abschluss soll noch einmal auf die Analogie zwischen Spule und Kondensator hingewiesen werden. Man kann sich Experiment eins und zwei jeweils als Ladevorgang und Entladevorgang der Spule denken. Es wurden beide beschrieben und nachgerechnet.

Die Analogie zieht sich aber noch weiter, denn genauso wie sich in einem physikalischen elektrischen Netzwerk jedes Bauteil sowie auch die Kabelverbindungen wie ein Kondensator mit schwacher Kapazität verhalten, verhalten sie sich auch wie eine Spule mit schwacher Induktivität.
Entsprechend muss man auch so genannte induktive Effekte berücksichtigen, wenn man ein elektrisches Netzwerk plant. Ein bekannter Effekt der Induktvität ist der Funkenüberschlag beim öffnen eines Schalters oder die zeitliche Verzögerung bei sehr langen Netzwerken.
Will man das induktive Verhalten eines elektrischen Bauteiles im Schaltplan berücksichtigen, wird eine Spule mit entsprechender Induktivität vor oder nach dem Bauteil in Reihe geschaltet. Es gibt noch induktive Effekte die über die hier besprochenen Effekte hinausgehen. Hierzu gehören allgemeine elektromagnetische Effekte, die im Anhang besprochen werden.

5.4.3 Der Schwingkreis

In den letzten beiden Abschnitten haben wir einen Speicher für Strom, die Spule und einen Speicher für Spannung, den Kondensator kennen gelernt. Nun ist es so, dass ein Kondensator durch einen Strom geladen wird, und sich über einen Strom entlädt. Auf der anderen Seite ist es so, dass eine Spule über eine Spannung geladen wird und sich über eine Spannung entlädt.

Es ist daher leicht zu sehen, dass wenn wir eine Spule und einen Kondensator kombinieren in einer Schleife, die eine Komponente die andere Komponente beim entladen auflädt und umgekehrt. Dies führt zu einem zyklischen oder periodischen Verhalten des Stromkreises.
In diesem Abschnitt werden wir uns entwas näher mit diesem Phänomen auseinandersetzen und nachrechnen was genau passiert und wie ein ohmscher Widerstand das Verhalten beeinflußt.

Betrachten wir zunächst den einfacheren Fall, den so genannten ungedämpften

Schwingkreis, nämlich eines Kondensators, der direkt an eine Spule geschlossen wird. Nehmen wir an, dass der Kondensator anfangs geladen ist und kein Strom fließt und berechnen wie sich der Strom und die Spannung mit der Zeit entwickelt.

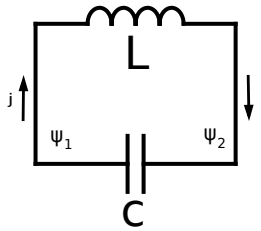


Figure 5.27: Ungedämpfter Schwingkreis.

Wir nehmen wieder an, dass $\psi_2 = 0$ und können sofort die Differentialgleichungen aufschreiben.

$$\frac{dj}{dt} = \frac{\psi_1}{L} \tag{5.36}$$

$$\frac{d\psi_1}{dt} = -\frac{j}{c} \tag{5.37}$$

Durch Einsetzen von (4.37) in (4.36) erhalten wir die Gleichung:

$$\frac{d^2\psi_1}{dt^2} = -\frac{\psi_1}{Lc} \tag{5.38}$$

Diese Gleichung ist bekannt als der klassische harmonische Oszillator und besitzt die allgemeine Lösung:

$$\psi_1(t) = a\cos(\frac{t}{Lc}) + b\sin(\frac{t}{Lc}) \tag{5.39}$$

Wir haben also zwei Unbekannte, die durch die Anfangsbedingung bestimmt werden sollen. Die Anfangsbdingung lautet $\psi_1(0) = u$ und $j(0) = 0$. Betrachten wir zunächst die allemeine Lösung und ihre Ableitung zum Zeitpunkt $t = 0$.

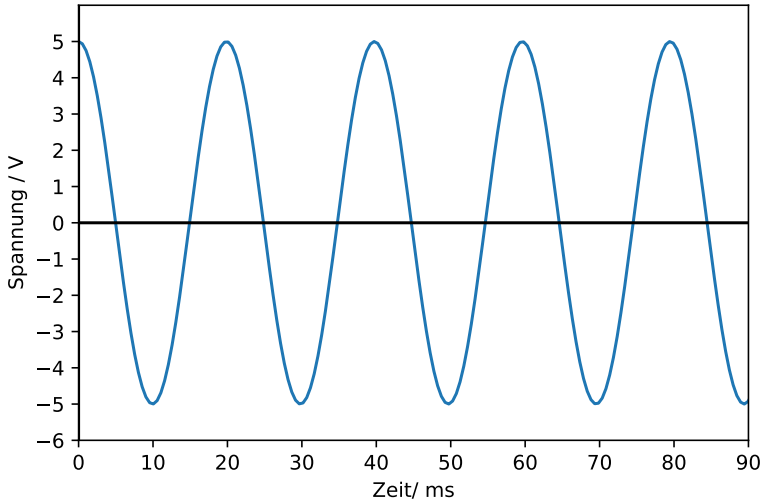
$$\psi_1(0) = a\cos(\frac{0}{Lc}) + b\sin(\frac{0}{Lc}) = a \tag{5.40}$$

$$\frac{d\psi_1}{dt}(0) = -\frac{a}{Lc}\sin(\frac{0}{Lc}) + \frac{b}{Lc}\cos(\frac{0}{Lc}) = \frac{b}{Lc} \tag{5.41}$$

Aus GLeichung (4.37) und $j(0) = 0$ folgt zusammen mit (4.41), dass $b = 0$ und aus der Anfangsbedingung $\psi_1(0) = u$ können wir mit (4.40) schließen, dass $a = u$. Die Lösung unseres Problems lautet also:

$$\psi_1(t) = u * \cos(\frac{t}{Lc}) \tag{5.42}$$

In graphischer Form dargestellt sieht das bei einem Kondensator mit Kapazität $c = 1nF$ und einer Spule mit Induktivität $L = 1H$ folgendermaßen aus.



Man sieht ein periodisches Verhalten des Spannungssignals, was analog auch für den Strom gilt. Dieser periodische Vorgang würde sich unendlich fortsetzen wenn man ihn ließe. In der Praxis ist eine derartige Schaltung aber unmöglich, da der Kondensator und die Spule nur Idealisierungen darstellen. Vor allem ist in einem echten Schaltkreis auch immer die Charakteristik der verbindenden Drähte, also ein ohmscher Widerstand, beteiligt.

Warum nennen wir diese Schaltung einen **gedämpften** Schwingkreis? Um diese Frage zu beantworten betrachten wir die Energie, die ein Kondensator bei einer bestimmten Spannung U hält.

$$E_c(U) = \frac{1}{2} c * U^2 \tag{5.43}$$

Die Energie erhält man durch Integration der Leistung nach der Zeit. In diesem Fall verwendeten wir Gleichung (4.18) und die Formel für die elektrische Leistung (4.1).
Analog errechnet sich die Energie die bei einer bestimmten Stromstärke I in einer Spule gespeichert ist aus (4.26) und erhält.

$$E_L(I) = \frac{1}{2} L * I^2 \tag{5.44}$$

Wir können E_L als die kinetische Energie und E_c als die potentielle Energie in Analogie zur Mechanik. Die Gesamtenergie eines geschlossenen Systems

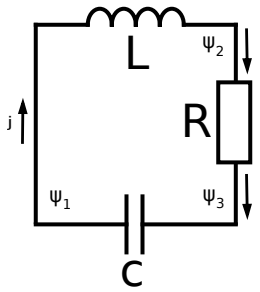
bleibt erhalten und da wir Verluste durch Wärme und elektromagnetischen Wechselwirkung mit der Umgebung vernachlässigt wurde, gilt dies auch hier. Also wenn wir die Lösung (4.42) und den sich daraus ergebenden Strom nach (4.37) einsetzen erhalten wir.

$$E_{ges}(\psi_1) = E_L(-c\frac{d\psi_1}{dt}) + E_c(\psi_1)$$
$$= \frac{1}{2}(Lc^2(\frac{d\psi_1}{dt})^2 + c(\psi_1)^2) = \frac{1}{2}cu^2$$

Dies zeigt, dass die Gesamtenergie E_{ges} konstant bleibt für alle Zeiten und zwar mit dem Startwert der Energie des geladenen Kondensators.

Dämpfung ist ein Resultat von Termen in der dynamischen Gleichung, die Energie an die Umgebung abfüren. Ein Ohmscher Widerstand gibt uns einen solchen Term. Er wandelt elektrische Energie bei Stromdurchfluss in Wärme um, die so genannte Joulesche Wärme.

Widmen wir uns um näher an die Realität zu kommen, kurz dem gedämpften Schwingkreis in dem einfach zusätzlich ein Widerstand zwischen Kondensator und Spule geschaltet ist.



Wir setzen wieder $\psi_3 = 0$ und lesen direkt das zugehörige Differentialgleichungssystem ab.

$$\frac{dj}{dt} = -\frac{\psi_1 - j * R}{L} \tag{5.45}$$

$$\frac{d\psi_1}{dt} = -\frac{j}{c} \tag{5.46}$$

Durch Einsetzen von (4.45) in (4,46) erhalten wir wieder eine Differentialgleichung mit einer zweiten Ableitung.

$$\frac{d^2\psi_1}{dt^2} + \frac{R}{L}\frac{d\psi_1}{dt} + \frac{1}{Lc}\psi_1 = 0 \tag{5.47}$$

Dies ist eine lineare Differentialgleichung mit konstanten Koeffizienten, genauso wie der harmonische Oszillator, nur eben mit einem Dämpfungsterm. Eine

solche Gleichung lässt sich mit einem Exponentialansatz lösen. Setzen wir hierzu eine Funktion ψ_1 der Form

$$\psi_1(t) = a * e^{\lambda * t} \tag{5.48}$$

in die Gleichung (4.47) ein wobei λ eine unbekannte komplexe Zahl ist. Exponentialfunktionen sind immer positiv also können wir nach Einsetzen auf eine quadratische Gleichung reduzieren.

$$\lambda^2 + \frac{R}{L}\lambda + \frac{1}{Lc} = 0 \tag{5.49}$$

Es gibt entweder zwei komplexe Zahlen λ_1 und λ_2, oder nur eine komplexe Zahl λ, die diese Gleichung auflöst.

Übungsbeispiel: Verwende die Lösungsformel für quadratische Gleichungen um die Gleichung (4.49) zu lösen.

Wir können direkt an der Lösungsformel ablesen, dass wir bis auf einen Grenzfall und zwar wenn $R^2 * c = 4 * L$ gilt, zwei Lösungen erwarten. Heben wir uns den Grenzfall erstmal auf und behandeln den Fall mit zwei Lösungen. Unser Anfangswertproblem zu (4.47) verlangt zwei unabhängige Lösungsfunktionen, da wir zwei Anfangsbedingungen haben, $j(0) = 0$ und $\psi_1(0) = u$.
Setzen wir nun für ψ_1 in (4.47)

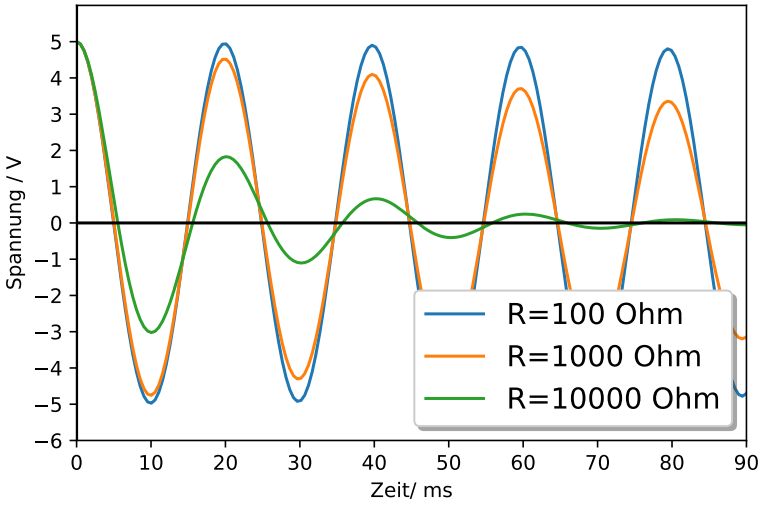
$$\psi_1(t) = a * e^{\lambda_1 * t} + b * e^{\lambda_2 * t} \tag{5.50}$$

und suchen Werte für die Unbekannten a und b, sodass die Anfangsbedingung erfüllt sind. Wir gehen hier genauso vor wie beim ungedämpften Fall und erhalten die Lösung.

$$\psi_1(t) = \frac{\lambda_2 * u}{\lambda_2 - \lambda_1} * e^{\lambda_1 * t} + \frac{\lambda_1 * u}{\lambda_2 - \lambda_1} * e^{\lambda_2 * t} \tag{5.51}$$

Übungsbeispiel: Führe die Berechnung durch, die zu gegebenen Anfangswerten $j(0)$ und $\psi_1(0)$ Werte für a und b liefert.

Betrachten wir die graphische Darstellung des zeitlichen Verlaufes der Spannungskurve mit den selben Werten für Induktivität und Kapazität wie in der vorherigen Darstellung im ungedämpften Fall, für drei Werte des Widerstandes R.



Wir sehen, dass eine Dämpfung stattfindet, daher die vormals periodische Funktion mit der Zeit abklingt. Dieses Abklingen ist umso schneller, desto größer der elektrische Widerstand im Schwingkreis ist.

Wenn wir das Ganze von der Perspektive der Energie im Schwingkreis aus betrachten, verhält sich der Widerstand wie ein Leck, das mit der Zeit Energie an die Umwelt abgibt. Diese Jouleschen Verluste sind für den Dämpfungseffekt verantwortlich. Der Energieverlust erfolgt durch Umwandlung der elektrischen Energie in Wärme nach der Formel:

$$E_R(t) = j^2 * R * t \tag{5.52}$$

Im Fall wenn es nur eine Lösung der Gleichung (4.50) gibt, in der Literatur als aperiodischer Grenzfall bezeichnet, müssen wir den Ansatz

$$\psi_1(t) = a * e^{\lambda * t} + b * t * e^{\lambda * t} \tag{5.53}$$

verwenden.

Übungsbeispiel: Finde die Lösung im aperiodischen Grenzfall.

Das Thema Schwingkreis ließe sich noch wesentlich detailierter darstellen. Der gedämpfte Schwingkreis ist immer noch eine Idealisierung und wir sind nicht auf ein äußeres Spannungssignal eingegangen, dass womöglich den Schwingkreis treibt.

5.4.4 Die Diode

Eine Diode ist ein zweipoliges nichtlineares elektrisches Bauteil, dass Strom in eine Richtung sehr gut und in die entgegengesetzte Richtung bei moderat niedrigen Spannungen kaum leitet. Es gibt sehr viele verschiedene Bauformen und Implementierungen in verschiedenen Substraten. An sich ist eine Diode kein dynamisches Bauteil, aber Implementierungsabhängig kann es eine zeitliche Charakteristik haben und unabhängig von der Implementierung ist sie in Sperrrichtung immer auch ein Kondensator. Wir werden uns in der Diskussion von Dioden auf normale Halbleiter Dioden beschränken.

Figure 5.28: Schaltsymbol einer Diode.

Am Schaltsymbol einer Diode sehen wir sofort, dass es sich um ein Bauteil mit asymmetrischer Charakteristik handelt. Der Pfeil gibt hierbei die bevorzugte Stromrichtung an und die vertikale Linie ist auf der Seite aus welcher der Strom nicht kommen darf.

5.4.5 Der Feldeffekt Transitor

5.4.6 Kippstufen

5.4.7 Logische Gatter

Kapitel 6

Ein möglichst einfacher Digitalrechner

Ein Digitalrechner zeichnet sich dadurch aus, dass er genau zwei Buchstaben hat, die in der Machine durch ein hohe Spannung und eine niedrige Spannung dargestellt sind. Wir schreiben diese Buchstaben normalerweise als Null und Eins.
Innerhalb der Rechenmaschine werden diese Buchstaben in Worten einer festen Länge angeordnet. Jede Stelle in diesen Anordnungen von Nullen und Einsen nennen wir Bits.
Ein moderner 64Bit Prozessor arbeitet entsprechend auf Worten der Länge 64. Diese Worte stellen die eigentlichen elementaren Objekte im Computer dar. Jede Operation wird auf einem ganzen Wort ausgefürt, jeder vergleich ist ein Vergleich zwischen ganzen Worten.

Diese Worte können natürliche Zahlen, ganze Zahlen, rationale Zahlen oder auch komplexe Zahlen symbolisieren. Genauso ist es aber auch möglich, dass es eine Kodierung gibt, die einem Wort einen Buchstaben zuordnet, sodass die Worte im Speicher zusammen einen Text repräsentieren.

Die einfachste Möglichkeit wäre es eine Rechenmaschine mit Wortlänge Eins zu bauen, also jedes Wort ist genau einen Bit lang. Aber dies würde der Maschine eine Komplexität nehmen, die ich für essentiell halte.

Ich habe mich entschieden für uns eine 4Bit Rechner zu bauen. Daher wir arbeiten mit der Wortlänge vier, wodurch wir zumindest natürliche Zahlen von 0 bis 15 oder ganze Zahlen von -7 bis +7 darstellen können. Vier Bit werden auch ein Nibble genannt, genauso wie acht Bit auch ein Byte genannt werden.

6.1 Aufbau der Rechenmaschine

In diesem Abschnitt wird der grobe Aufbau unserer Rechenmaschine vorgestellt. Dies geschieht auf der Ebene von Modulen. Ein Modul ist ein Bauelement, dass eine einfach zu verstehende Funktion erfüllt und bis auf eine streng definiertes Eingabe-Ausgabe Verhalten in sich geschlossen ist. Jedes der Module erfüllt eine grundlegende Aufgabe der Rechenmaschine und besteht aus einer Verknüpfung von logischen Gattern.

Das Herz der Rechenmaschine ist die Kontrolleinheit. Sie hat die Aufgabe aus dem Programmspeicher den nächsten Befehl abzurufen, und den anderen Modulen zu sagen, was von ihnen verlangt wird um dem Befehl folge zu leisten. Der Speicher dient wie nicht anders zu erwarten ist dem Speichern von vier Bit Worten. Worte im Speicher können im Rechenkern durch mathematische Operationen verknüpft werden. Das Ergebnis der Operation landet dann im Register. Abhängig vom Ausgang der Operation wird auch ein Flag, dass sich ebenfalls im Register befindet auf Eins oder auf Null gesetzt. Das Register kann aber auch direkt Inhalte aus dem Speicher annehmen oder an Speicheradressen im Speicher senden. Der BUS dient dabei als Kommunikationskanal für vier Bit Übertragung zwischen den Modulen. Das ist im Prinzip alles was wir benötigen. Betrachten wir kurz eine Skizze des Aufbaus unserer Rechenmaschine.

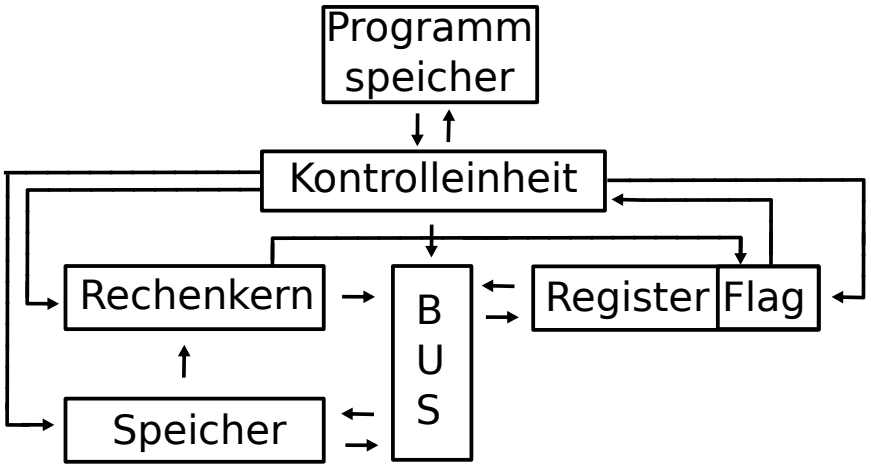


Figure 6.1: Grober Plan der Rechenmaschine.

Die Pfeile symbolisieren Kommunikation de zwischen den Modulen stattfinden kann. Wir sehen zum Beispiel, dass die Kontrolleinheit allen Modulen Nachrichten schickt, aber nur vom Register Nachrichten zurückbekommen kann. Die Kontrolleinheit braucht nämlich das Flag um Informationen über den Ausgang von Rechenoperationen zu bekommen. Dies ist notwendig,

wenn die Kontrolleinheit eine bedingte Verzweigung ausführen soll. Der Rechenkern bekommt seine Eingabe vom Speicher und sendet sie an den BUS. Der BUS kann mit dem Speicher und dem Register in beide Richtungen kommunizieren. Wir geben der Rechenmaschine aber nicht die Möglichkeit direkt die Ausgabe vom Rechenkern in den Speicher zu schicken, da es sonst zu Instabilitäten kommen kann. Dies kann passieren, wenn an die Speicheradresse geschrieben wird von der gerade gelesen wird. Die Kommunikation zwischen der Kontrolleinheit und dem Programmspeicher geht auch in beide Richtungen, da die Kontrolleinheit die Reihenfolge der Befehle verändern können muss.

In den folgenden Abschnitten werden die Module einzeln vorgestellt und im Detail beschrieben. Am Ende dieses Kapitels haben wir dann eine vollständige Rechenmaschine konstruiert.

6.1.1 Speicher

Die Rechenmaschine hat einen Daten Speicher von 16 Worten. Er kann sich also gleichzeitig 16 vier Bit lange Worte merken. Jeder dieser ein Wort fassenden Speicherblöcke hat eine 4 Bit lange Addresse mit der er angesprochen werden kann.

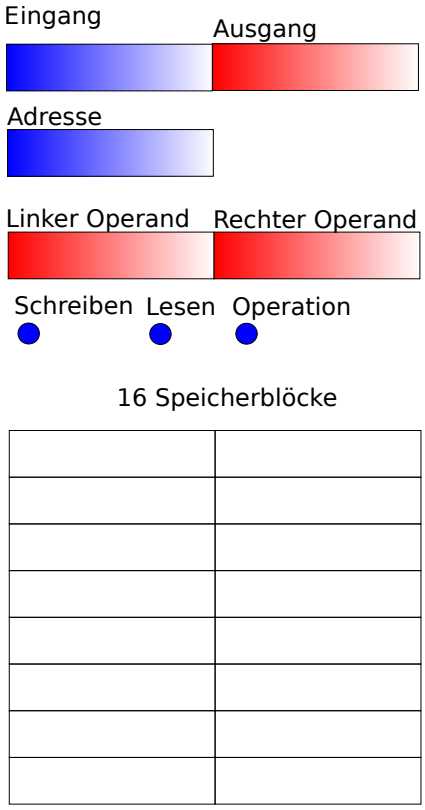


Figure 6.2: Speichermodul.

Die Abbildung zeigt das was wir Schema des Moduls nennen werden. Es fasst alle als Eingang verwendeten Anschlüsse durch blau unterlegte Symbole zusammen und alle Ausgänge durch rot hinterlegte Symbole zusammen. Ein Bit Anschlüsse werden durch kleine Kreise dargestellt und vier Bit Anschlüsse durch rechteckige Kästchen. Die übrigen nicht farblich hinterlegten Symbole stellen in diesem Fall interne Komponenten des Moduls dar, die nicht direkt manipulierbar sind.

Das Speicher Modul hat einen vier Bit Ausgang/Eingang Anschluss um Daten zwischen Register und Speicher hin und her zu transportieren. Zusätzlich gibt es einen vier Bit Anschluss an dem die Addresse angelegt wird. Es gibt

noch drei ein Bit Eingänge mit den Namen **Schreiben**, **Lesen** und **Operation**.
Wenn wir beabsichtigen ein bestimmtes Wort $(1, 1, 1, 1)$ in Speicheradresse $(0, 1, 1, 1)$ zu schreiben, dann legen wir das Wort an den Eingang/Ausgang an, setzen die Adresse auf den entsprechenden Wert und setzen den ein Bit Eingang mit dem Namen **Schreiben** auf Eins und die anderen ein Bit Eingänge auf Null.
Die Spannungssignale müssen bevor der Schreiben Eingang auf Eins gesetzt wird immer stabilisiert sein, damit nicht an die falsche Adresse geschrieben wird.
Falls wir aber ein Wort an einer bestimmten Adresse lesen wollen, setzen wir die Adresse auf den gewünschten Speicherblock und den Lesen Eingang auf Eins. Alle anderen ein Bit Eingänge sollen dabei wieder auf Null gesetzt sein.

Man unterscheidet Speicherblöcke deren Adresse auf Null endet und solche deren Adresse auf Eins endet. Erstere wollen wir linker Speicher und letzere rechter Speicher nennen. Wenn eine binäre Operation ausgeführt wird, ist jeweils der linke Operand aus dem Linken und der rechte Operand aus dem rechten Speicher.
Deshalb gibt es einen **Operation** Modus, der aktiv ist wenn am Operation Eingang Eins anliegt. Im Operation Modus wird das letzte Bit der Adresse $(a_1, a_2, a_3, a_4))$ ignoriert und stattdessen am Ausgang **Linker Operand** der Inhalt vom Speicherblock mit der Adresse $(a_1, a_2, a_3, 0))$ ausgegeben und am Ausgang **Rechter Operand** der Inhalt der Speicheradresse $(a_1, a_2, a_3, 1))$ ausgegeben.

6.1.2 Das Register

Normalerweise übernehmen Register in einer Rechenmaschine eine Aufgabe analog zu Schmierzetteln beim händisch Rechnen. Es gibt nur einige wenige davon im Gegensatz zum großen Speicher, ihr Inhalt kann schnell manipuliert werden und in vielen Fällen können Operanden und Resultate von Rechenoperationen nur Register sein. Bei unserer Rechenmaschine gilt dies nur teilweise. Auf jeden Fall gibt es nur ein einzelnes vier Bit Register, aber die Zugriffszeiten des Registers unterscheiden sich nicht von denen des Hauptspeichers.

Eine Besonderheit des Registers ist es, dass sie zusätzlich noch ein fünftes Bit enhält, dass unser einziges Flag darstellt.
Das Register hat einen vier Bit Eingang/Ausgang, wie der Speicher, einen ein Bit Eingang zum Setzen des Flags und einen ein Bit Ausgang zum Lesen des Flags. Zusätzlich hat es noch drei ein Bit Eingänge, mit Namen **Lesen**, **Schreiben** und **Operation**.
Wenn wir ein Wort in das Register geschrieben werden soll, wird das Wort

an den Eingang/Ausgang angelegt und der ein Bit Eingang mit Namen **Schreiben** auf Eins gesetzt und der **Lesen** Eingang auf Null gesetzt.
Im Gegenzug wenn wir das Wort im Register lesen wollen, setzen wir den **Lesen** Anschluss auf Eins und den **Operation** sowie **Schreiben** Anschluss auf Null, sodass dann am Eingang/Ausgang das gesuchte Wort vorliegt.
Falls eine Operation stattfindet, wird der **Operation** Anschluss auf Eins geschalten und an den fünften Eingangs/Ausgangsbit der Flag Zustand des Rechenkerns angelegt. Der Zustand des Flags kann zu jeder Zeit vom Flag Ausgang gelesen werden.

Betrachten wir als nächstes das Eingabe-Ausgabe Schema des Registers.

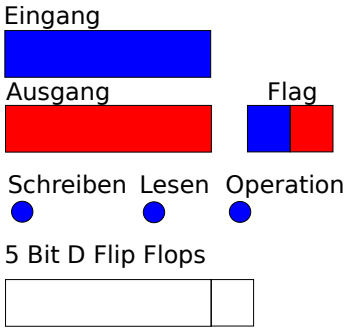


Figure 6.3: Register.

Der Bau unseres Register lässt sich sehr einfach mit fünf D-Flip-Flops und vier UND-Gattern umsetzen. Betrachte hierzu den logischen Schaltplan.

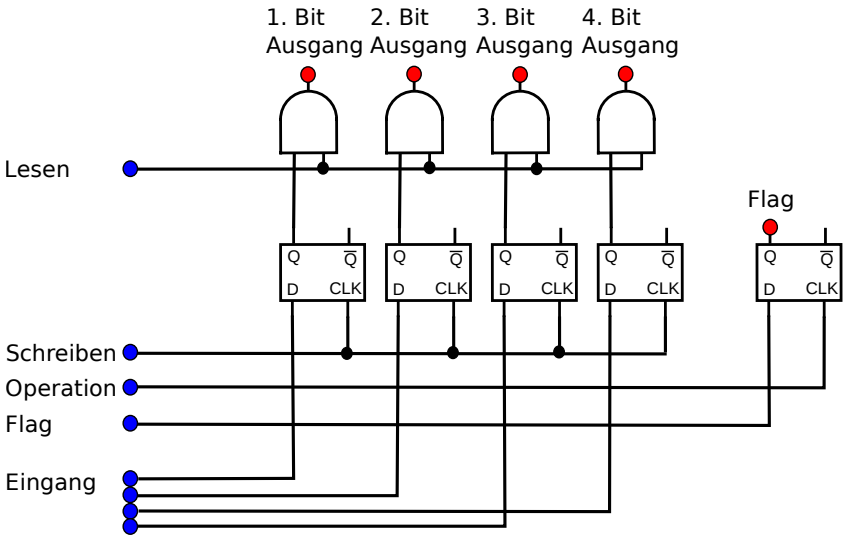


Figure 6.4: Plan Register.

Wir sehen im Plan im Gegensatz zum Schema jeweils einen vier Bit Eingang und einen vier Bit Ausgang. Beim Zusammenbau der Rechenmaschine werden später alle Ausgänge der unterschiedlichen Module über ein gemeinsames ODER-Gatter in den BUS eingleitet, der direkt an den Eingängen liegt. Dementsprechend werden die Ein- und Ausgänge verbunden.

6.1.3 Der Rechenkern

Die harte Arbeit im Rechner, also die Implementierung der Grundoperationen übernimmt der Rechenkern. Er ist dabei dafür verantwortlich abhängig vom einkommenden Steuersignal die richtige Operation auszuwählen, und das korrekte Resultat der Operation, angewendet auf die einkommenden Operanden, auszugeben

In unserem Fall implementiert er eine stellenweise NAND Operation und eine Links Verschiebung.
Die stellenweise NAND Operation nimmt zwei 4 Bit lange Worte und wendet auf jedes Bit das NAND an. Falls das Resultat Null ist wird das **Flag** auf Eins gesetzt, ansonsten wird es Null.
Die Links Verschiebung ignoriert den rechten Operand und schiebt jeden 0-1 Wert des Wortes auf den nächst höheren Bit. Der niedrigste Bit erhält dabei den Wert Null und der Wert des höchsten Bits geht auf das Flag üeber.

Betrachten wir das Eingabe/Ausgabe Schema unseres Rechenkerns.

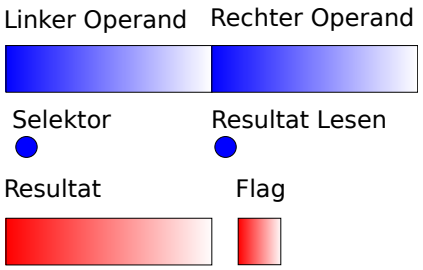


Figure 6.5: Rechenkern.

Wir haben zwei vier Bit Eingänge für den **linken** und den **rechten Operand**, zwei ein Bit Eingänge namens **Selektor** und **Resultat Lesen**. Zusätzlich haben wir einen vier Bit Ausgang für das Ergebnis der Operation und einen ein Bit Ausgang für das Flag.

Der ein Bit Eingang namens **Resultat Lesen** setzt den Ausgang **Resultat** in jedem Fall auf Null, falls er selbst gleich Null ist. Falls der Eingang jedoch Eins ist, haben wir das im Folgenden beschriebene Verhalten.

Wenn wir die NAND Operation ausführen wollen, stellen wir den Selektor auf Null und legen den linken sowie den rechten Operand an die entsprechenden vier Bit Eingänge. Das Resultat erscheint am 4 Bit Ausgang **Resultat**. Falls das Ergebnis der Operation gleich Null sein sollte, wird der **Flag** Ausgang gleich Eins, ansonsten Null.

Will man allerdings die Links Verschiebung anwenden, so legt man am Selektor die Eins and und an den vier Bit Eingang namens **linker Operand** das zu bearbeitende Wort an. Wie im ersten Fall steht das Resultat im Ausgang **Resultat** und das **Flag** trägt den vormals höchsten Bit des linken Operanden.

Das beschriebene Verhalten lässt sich leicht durch folgenden logischen Schaltplan darstellen.

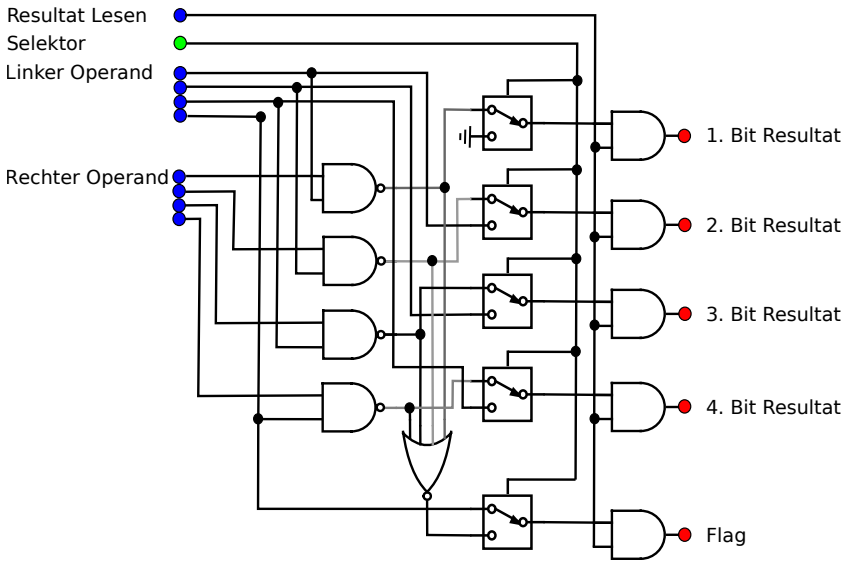


Figure 6.6: Logischer Schaltplan Kern.

Übungsbeispiel : Analysiere den logischen Schaltplan und identifiziere welche Komponente welche Aufgabe übernimmt.

6.1.4 Der BUS

Ein BUS wird in einer Rechenmaschine verwendet um die Kommunikation mehrerer Komponenten miteinander zu ermöglichen. Der BUS selbst besteht dabei aus einem mehrere Bit großen Kanal, durch den genau zwei Komponenten miteinander reden können. Es gibt dabei immer einen Sender und einen Empfänger. Falls man mehr als zwei Komponenten hat oder bloß Komminikation in beide Richtugen zwischen zwei Komponenten zu ermöglichen will, muss man präzise steuern wann eine Komponente redet, wann sie schweigt und wann sie zuhören soll. Das Prinzip ist, dass immer genau eine Komponente reden soll und alle anderen zu dieser Zeit schweigen sollen.
Im Fall unserer Rechenmaschine ist der BUS ein vier Bit breiter Kommunikationskanal über den, wie in der Skizze zu Beginn dieses Kapitels zu erkennen ist, beinahe jeglicher Datenaustausch läuft.

Der BUS hat entsprechend folgendes Eingabe/Ausgabe Schema.

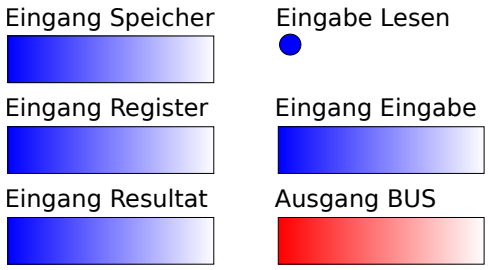


Figure 6.7: BUS Modul.

Wir sehe die vier Module, die im BUS senden dürfen, aber nur ein Steuersignal **Lesen**, dass kontrolliert ob die Eingabe sendet oder nicht. Die Eingabe ist dabei ein von der Kontrolleinheit kommendes vier Bit Signal, dass direkt aus dem Programm im Programmspeicher ausgelesen wurde.
Die übrigen Module wurden bereits mit einem Kontrolleingang geplant, der festlegt ob sie gerade in den BUS senden oder nicht. Im Prinzip ließe sich dieses Kontrollsignal auch direkt in der Kontrolleinheit implentieren, aber diese ist ohnehin schon komplex genug, wie wir gleich sehen werden.
Der vier Bit Ausgang, namens **Ausgang BUS**, dient als Anschluss für alle Module, die empfangen sollen. Die korrekten Anschlüsse finden sich in der Skizze zu Beginn des Kapitels.

Ein möglicher logischer Schaltplan um das erwünschte Verhalten zu bekommen, ist der Folgende.

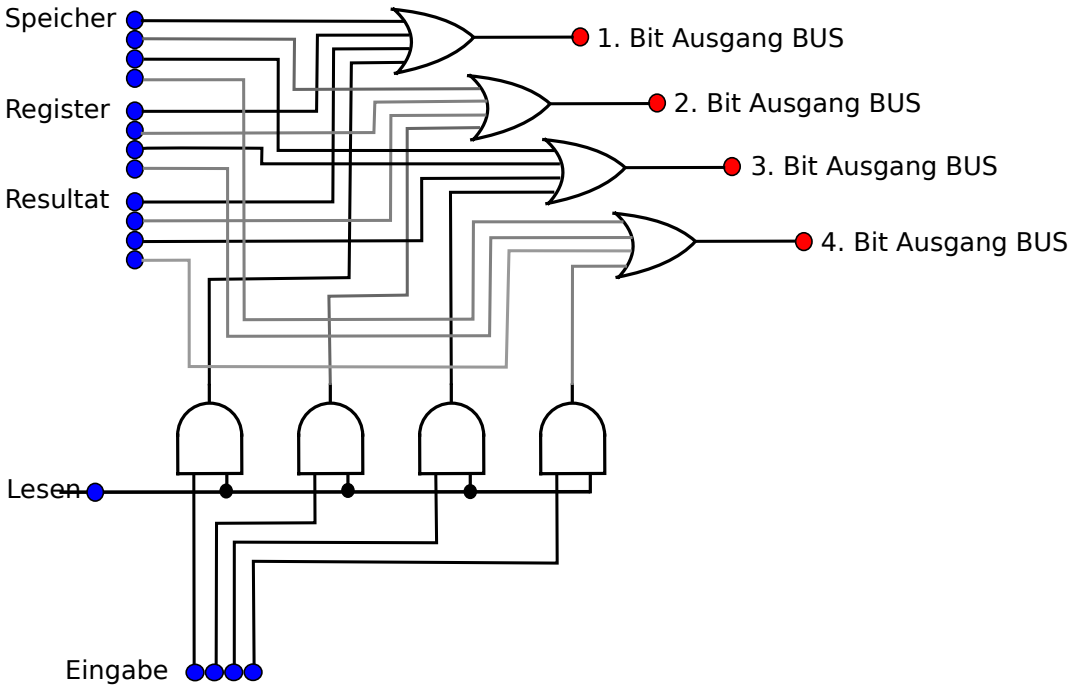


Figure 6.8: Logischer Schaltplan.

Die Kontrolle über die Eingabe wird durch UND-Gatter ermöglicht. Wir erkennen, dass nur, dann ein Signal durchgelassen wird wenn auch das **Lesen** Signal Eins ist. Das resultierende vier Bit Signal und die Signale der anderen Eingänge werden dann über Oder-Gatter kombiniert.

6.1.5 Kontrolleinheit

Die Kontrolleiheit ist der zentrale Kern des Rechners. Sie kommuniziert mit dem Programmspeicher, um die nächste Anweisung zu erhalten und übersetzt diese dann in zeitlich abgestimmte Signale an die einzelnen Module. Die Module führen dann die durch die Anweisung bestimmte Berechnung durch.

Wir können die Kontrolleinheit in zwei Teilmodule aufspalten, nämlich einem **Taktgeber** und einem **Ausführer**. Der Taktgeber erzeugt drei Taktsignale. Die Taktsignale sind drei Rechteckswellen ohne Überlappung, die der Reihe nach in der sie auftreten, Taktsignal, Schreibsignal und Steuersignal genannt werden. Es ist dabei zu beachten, dass zwischen diesen Signalen abhängig von den Charakteristiken der elektrischen Bauteile mindestens eine bestimmte Zeit vergehen muss und die Signale eine bestimmte Zeit lang sein müssen.

Dies wird durch einen so genannten Ring Counter und einen Rechteckswellen Generator implementiert. Der Ring Counter besitzt drei Zustände durch die bei Eintreffen einer aufsteigenden Rechtecksflancke am Takteingang zyklische weitergeschalten werden. Das Resultat wird in der folgenden Abbildung dargestellt. Das Rechteckssignal von Generator wird in drei Taktsignal umgewandelt. Eine gute Analogie wäre hier ein Dirigent der den verschiedenen Teilen des Orchesters sagt wann sie Spielen und wann sie nicht Spielen sollen.

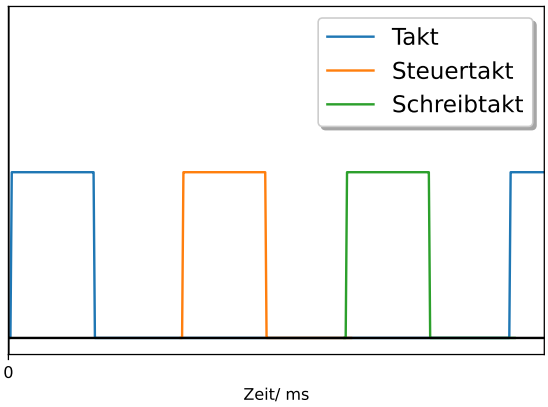


Figure 6.9: Taktsignale.

Der Grund warum wir ein extra Schreibsignal und Steuersignal brauchen liegt in einer Verzögerung der jede elektronische Komponente im Rechner unterliegt. Erst nach einer kurzen Zeit wird jede Komponente den korrekten Spannungswert liefern und ein Schreiben bevor diese Zeit vergangen ist, führt zu Fehlern. Das Steuersignal dient dazu den internen Zustand der Kontrolleinheit zu manipulieren und das Zurücksetz Signal an den Programmspeicher zu erzeugen. Es ist notwendig, dass nichts mehr geschrieben wird nachdem das Steuersignal entweder den internen Zustand ändert oder ein Zurücksetzsignal sendet, da sich ansonsten die Machine in einem unspezifischen Zustand befindent und gleichzeitig Schreiben darf.

Ausführer

Am Anfang des Kapitels haben wir gesehen wie die Kontrolleinheit mit den unterschiedlichen Modulen im Rechner kommuniziert. Anschließend haben wir in jedem der bisherigen Module gesehen welche Steuersignale jedes von ihnen benötigt. In diesem Abschnitt wird gezeigt wie der Ausführer mit dem Programmspeicher kommuniziert, und abhängig von seinem internen Zustand, der vom Programmspeicher gesendeten Anweisung und dem Flag diese Steuersignale erzeugt.

Der Ausführer hat fünf ein Bit Eingänge. Einen Eingang für das Flag, einen Eingang für das Taktsignal, einen für den Schreib Takt, einen für den Steuertakt und einen Eingang zum Einschalten der Rechenmaschine.

Die Kommunikation mit dem Programmspeicher ist sehr einfach gestaltet. Es gibt einen ein Bit Ausgang namens **Zurücksetzen**. Dieser teilt dem Programmspeicher mit, dass er zurück zur ersten Anweisung gehen soll.
Der ein Bit Ausgang **Takt an** steuert den **Taktgeber**, der wiederum dafür verantwortlich ist dem Programmspeicher mitzuteilen wann er zur nächsten Anweisung über gehen soll. Als Antwort erhält der Ausführer an einem acht Bit Eingang, namens **Anweisung von Programmspeicher** die gesuchte Anweisung in Form eines acht Bit Wortes.

Intern hat der Ausführer zwei Zustände, die durch RS-Flip-Flops implementiert sind. Einen **AN** Bit, der signalisiert, dass die Maschine einen Rechenprozess ausführt und einen **WENN** Bit, der signalisiert, dass sich der Rechner in einer IF-THEN-END Verzweigung befindet.

Es folgt wie in jedem Modul bisher ein bildliche Zusammenfassung. Da es eine Menge Ausgänge gibt mit selbsterklärendem Namen, verzichte ich darauf sie alle hier im Text aufzuzählen, sondern verweise auf die Abbildung.

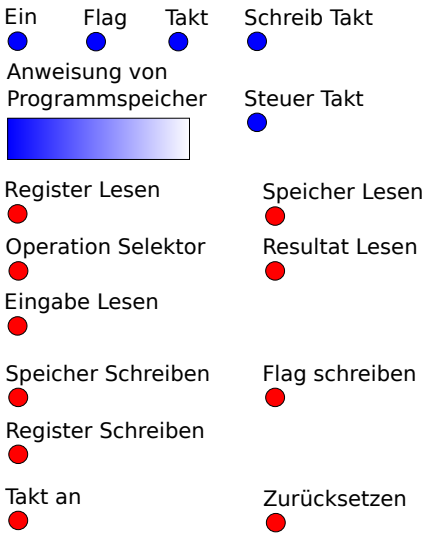


Figure 6.10: Ausführer.

Der ein Bit Ausgang namens **Operation Selektor** teilt dem Rechenkern mit welche der beiden Operationen ausgeführt werden soll. Die übrigen Ausgänge dienen zur Kontrolle des BUS.

6.2 Entwicklungsentscheidungen

Die Idee einer digitalen Rechenmaschine mit der kleinst möglichen Anzahl an arithmetischen und logischen Operationen, die in Kombination eine universelle Rechenmaschine ergeben, hat mich fasziniert, seit ich mir Gedanken über den Bau von Rechenmaschinen gemacht habe. Solche Minimal Konstruktionen sind in der Regel nur in der Theorie interessant, da sie natürlich mehr Rechenschritte benötigen als Maschinen mit mehreren Rechenoperationen. Diese zusätzlichen Rechenoperationen sind, zwar redundant was für bestimmte Menschen ein Schönheitsfehler sein kann, aber Schnelligkeit und Praktikabilität sind in der echten Welt wichtiger.

Ebenso galt es die Kontrolleinheit so einfach wie möglich zu gestalten um eine physikalische Realisierung mit so wenigen Bauteilen wie möglich zu gewährleisten. Dies musste natürlich unter der Bedingung geschehen, dass unsere Rechenmaschine mit universeller Berechenbarkeit ausgestattet ist.

Um zu zeigen, dass unsere Rechenmaschine ein universelle Programmiersprache implementiert, müssen wir lediglich einen Algorithmus finden, der die Additions Operation mit unseren beiden Grundoperationen und den zu verfügung stehenden bedingten Verzweigungen ausdrückt.

Bevor wir dies tun können müssen wir den Rest unserer Programmiersprache definieren. Meine Wahl fiel hierbei auf WHILE-Programme mit IF Verzweigungen. Insbesondere ließ ich mich einschränken durch die Tatsache, dass die Kontrolleinheit der Rechenmaschine selbst keine Additionsoperationen ausführen soll, da dies die Sinnhaftigkeit der Einschränkung auf die beiden gewählten Grundoperationen zu absurd scheinen lässt. Dies führte mich zur Kleenschen Normalform. Jedes WHILE-Programm, aber auch jedes GOTO- Programm lässt sich umschreiben sodass nur einer WHILE Schleife beziehungsweise GOTO Aufruf verwendet wird.
Die Kontrolleinheit muss in diesem Fall immer nur den jeweils nächsten Befehl ausspucken, oder im Fall einer IF Verzweigung einige Befehle überspringen und am Ende des Programms zurückspulen zum Anfang. Es ist weder die Eingabe einer absoluten Addresse noch einer relativen Addresse notwendig. Die Kontrolleinheit muss also nicht Addressen aus den Befehlen extrahieren und dem Befehlzeiger setzten, noch einen Teil des Befehls auf den derzeitigen Befehlszeiger hinzuaddieren. Als Konsequenz ist zum einen die Absurdität der Rechenmaschine nicht all zu offensichtlich und zum Anderen müssen unsere Befehle auch nicht all zu lange sein und jede Opertion benötigt genau einen Maschinenzyklus.
In unserem Fall wird ein Befehl ein Byte sein, wobei der Befehlsraum hierbei nicht ausgelastet wird.

Die Realisierung des Programmspeichers kann ein Lochstreifenlesegerät oder ein binär Counter zusammen mit einem 8 Bit Parallelspeicher (EEPROM oder FLASH) sein.

6.3 Maschinensprache

In diesem Abschnitt definieren wir die Programmiersprache, die unsere Rechenmaschine ausführen wird. Es wird dabei besonders viel Wert auf Einfachheit gelegt, wobei wir einige Eigenschaften ausnutzen die wir in den vorangegangenen Kapiteln kennen gelernt haben

Eine dieser Eigenschaften ist, dass sich jede aussagenlogische Formel durch NAND Operation ausdrücken lässt.
Am Ende dieses Kapitels werden wir sehen, dass sich sowohl die Addition als auch die Gleichheit in unserer Maschinensprache ausdrücken lässt.

Bei den bedingten Verzweigungen greifen wir auf die Kleenesche Normalform zurück. Hierzu benötigen wir eine Implementierung einer IF-THEN-END Verzweigung und einer einzigen WHILE Schleife. Die WHILE-Schleife soll dabei durch bedingte Sprünge zum Anfang des Programms dargestellt

werden. Hierdurch erhalten wir bedingte Verzweigungen, die equivalent zu GOTO Verzweigungen sind.

Die letzte Eigenschaft die wir ausnutzen ist, dass verschachtelte IF-THEN-END Verzweigungen nicht notwendig sind und wir werden demnach auf diese verzichten.

Gießen wir den Gedanken nun in eine konkrete Form. Jeder Befehl ist einen Byte lang und hat die Form:

$$(b_0, b_1, b_2, a_0, a_1, a_2, a_3, F)$$

Die Befehle, die unsere Rechenmaschine kennt sind die Folgenden:

HALT: $(b_0, b_1, b_2) = (0, 0, 0)$ Der Zustand der Rechenmaschine ändert sich nicht und kein weiterer Befehl wird mehr ausgeführt.

IF: $(b_0, b_1, b_2) = (0, 0, 1)$ Wenn das Flag auf Eins steht wird der nächste Befehl in der Reihe als nächstes ausgeführt, ansonsten wenn das Flag auf Null steht wird der nächste Befehl ausgeführt dessen letztes Bit, der F Bit, Eins ist.

NAND + 3 bit Addresse: $(b_0, b_1, b_2) = (0, 1, 0)$ Berechnet die NAND Operation des Wortes an der Addresse $(a_0, a_1, a_2, 0)$ mit dem Wort an der Stelle $(a_0, a_1, a_2, 1)$ und schreibt das Ergebnis in das Register. Falls das Ergebnis der Operation Null ist wird das Flag auf Eins gesetzt, sonst auf Null.

LSHIFT + 3 bit Addresse: $(b_0, b_1, b_2) = (0, 1, 1)$ Wendet die LSHIFT auf das Wort an der Stelle $(a_0, a_1, a_2, 1)$ and und schreibt das Ergebnis in das Register. Hierbei wird das nullte Bit des Wortes auf Null gesetzt und der Wert des dritten Bits wird auf das Flag übertragen.

POP + 4 bit Addresse: $(b_0, b_1, b_2) = (1, 0, 0)$ Schreibt das Wort im Register an die Addresse (a_0, a_1, a_2, a_3).

PUSH + 4 bit Addresse: $(b_0, b_1, b_2) = (1, 0, 1)$ Schreibt das Wort an der Addresse (a_0, a_1, a_2, a_3) in das Register.

LOAD + 4 bit Wort: $(b_0, b_1, b_2) = (1, 1, 0)$ Schreibt das Wort (a_0, a_1, a_2, a_3) in das Register.

RESET: $(b_0, b_1, b_2) = (1, 1, 1)$ Der Zähler wird auf Null gesetzt.

Nun sind wir in der Lage die Additionsoperation in der Maschinensprache unserer Rechenmaschine auszudrücken.

6.4 Funktionsweise der Kontrolleinheit

Nachdem wir die Machinensprache die in unserem Rechner zur Anwendung kommt definiert haben, können wir die Kontrolleinheit im Detail konstruieren.

Es gibt drei Befehle, die direkt die Kontrolleinheit betreffen. Die HALT Anweisung, die der Machine mitteilt, dass das Ende der Berechnung erreicht wurde und sie nun ruhen kann, die bedingte Verzweigungen, also der IF Befehl, und der RESET Befehl.
Diese Befehle sind in der Lage den internen Zustand der Kontrolleinheit zu verändern.

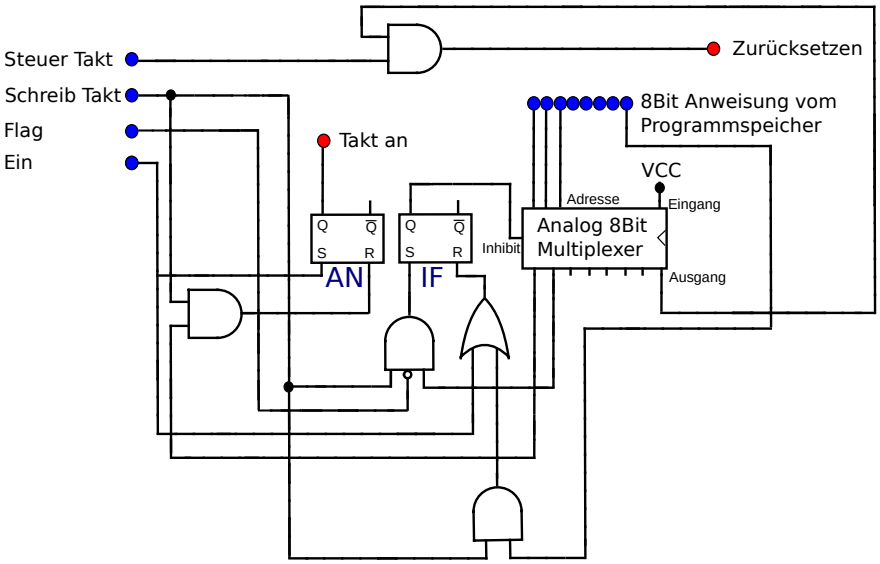


Figure 6.11: Logischer Schaltplan.

Das obige Bild stellt eine Möglichkeit dar den Ausführer als logischen Schaltplan darzustellen. Man beachte, dass nur die drei oben erwähnten Befehle in diesem Schaltplan implementier sind. Dies dient der Übersichtlichkeit und was fehlt wird später nachgeholt.

Starten wir beim Einschaltvorgang. Wenn wir den ein Bit Eingang namens **Ein** auf eins setzen, wird dadurch der AN Flip-Flop gesetzt. Dies

signalisiert der Machine, dass sie nun arbeiten soll.
Der aktive Ausgang des AN FLip-Flops steuert direkt über den **Takt an** Ausgang die Verbindung zwischen dem Rechteckswellengenerator und dem Ring Counter im Taktgeber. Hierdurch wird sicher gestellt, dass die Machine hält wenn der AN Flip-Flop inaktiv ist. Zusätzlich wird der IF Flip-Flop durch das AN Signal zurückgesetzt.

Der acht Bit Eingang vom Programmspeicher teilt sich in drei Segmente. Die ersten drei Bit sind an die drei Adresseingänge eines acht Bit analog Multiplexers gebunden. Die nächsten vier Bit bilden direkt den Ausgang **Datenadresse** und der letzte Bit wird über ein UND Gatter mit dem **Schreib Takt** verbunden. Das letztere Signal vom Ausgang des UND Gatters setzt den IF Flip-Flop zurück.

Der erste Ausgang des Multiplexers ist mit de **Schreib Takt** verknüpft durch ein UND-Gatter dessen Ausgang den **AN** Flip-Flop zurücksetzt. Hierdurch wird **HALT** implementiert.

Der zweite Ausgang des Multiplexers wird mit dem Schreibtakt und dem verneinten Flag Signal durch ein UND-Gatter verknüpft. Das resultierende Signal setzt den **IF** Flip-Flop. Dies implementiert den **IF** Befehl.

Der letzte Ausgang des Multiplexers ist über ein UND-Gatter mit dem Steuertakt verbunden. Der Ausgang des UND-Gatters bildet dann den Ausgang **Zurücksetzen** oder RESET.

Die übrigen Anweisungen betreffen alleine Steuersignale für den BUS und den Rechenkern und werden im folgenden präsentiert. Man beachte, dass alle Steuersignale, die ein Schreiben steuern nur Eins sind, wenn der Schreibtakt Eins ist.

Multiplexer Ausgang	Register Lesen	Speicher Lesen	Eingabe Lesen
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	1	0	0
5	0	1	0
6	0	0	1
7	0	0	0

Operation Selektor	Operation	Register schreiben	Speicher schreiben
0	0	0	0
0	0	0	0
1	1	1	0
0	1	1	0
0	0	0	1
0	0	1	0
0	0	1	0
0	0	0	0

Resultat Lesen	Flag Schreiben
0	0
0	0
1	1
1	1
0	0
0	0
0	0
0	0

Kommen wir zur Implementierung dieses Teils des Ausführers. Für jeden Ausgang des Multiplexers betrachten wir die entsprechende Zeile in der Tabelle und verbinden den Multiplexerausgang mit jedem durch Eins gekennzeichneten Ausgang des Ausführers. Falls mehrere Multiplexer Ausgänge mit dem selben Ausführer Ausgang verbunden werden sollen, müssen Diese zuerst durch ein ODER-Gatter mit entsprechend vielen Eingängen verknüpft werden. Der Ausgang des ODER-Gatters führt dann zum gewollten Ausführer Ausgang.

Abschließend muss noch **Register Schreiben**, **Speicher Schreiben** und **Flag Schreiben** jeder für sich mit dem **Schreib Takt** per UND-Gatter verknüpft werden. Hierdurch erzeugt man die eigentlichen Schreib Takt gesteuerte Schreibsignale.

Die gewünschte Funktionalität lässt sich mit nur wenig Aufwand implementieren.

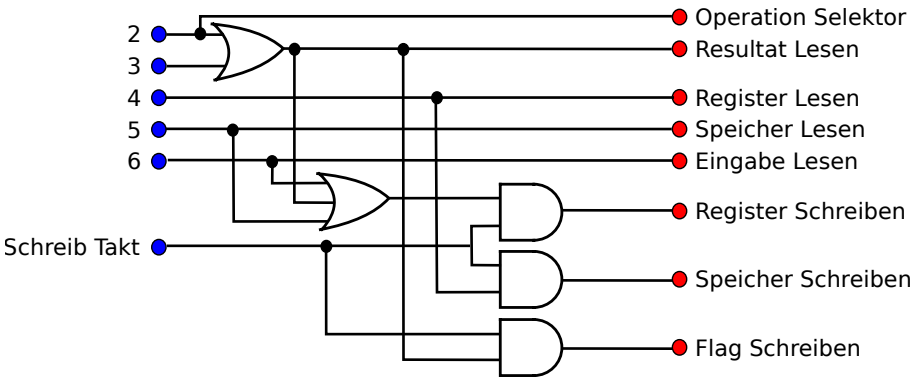


Figure 6.12: restlicher Ausführer.

Die Symbole 2, 3, 4 ,5 und 6 stehen hierbei für die entsprechenden Ausgänge des Multiplexers.

Kapitel 7

Programmierung der Rechenmaschine

Wir kommen nun endlich zur Ernte der Früchte unserer Bemühungen. In diesem Kapitel vergewissern wir uns, dass unsere Rechenmaschine tatsächlich eine universelle Programmiersprache implementiert, soweit dies überhaupt dies überhaupt für eine physikalische Machine möglich ist.
Dies wird in mehreren Schritten gezeigt. Zuerst vergewissern wir uns, dass wir mit unseren Grundfunktionen Vier Bit Addition und Gleichheit darstellen können. Damit haben wir bereits gezeigt, dass sich beliebige WHILE-Programme in unsrer Maschinensprache ausdrücken lassen. Natürlich mit der Einschränkung, dass die Variablen und Kontrollstrukturen, die notwendig sind in den Speicher passen, aber diese Einschränkung hat jede Rechenmaschine. Um zu zeigen, dass sich interessante Programme für unsere Rechenmaschine schreiben lassen, geben wir zum Abschluss ein paar Beispiele. Hierzu gehören Kontrollstrukturen für IF-THEN-END Anweisungen, acht Bit Addition, Multiplikation und ein Programm, dass Fibonacci Zahlen ausspuckt.

7.1 Addition

Die Addition zweier binärer Zahlen funktioniert analog zu dem Verfahren, dass wir in der Schule für dekadische Zahlen gelernt haben. Nehmen wir als Beispiel der Addition der Zahl 0111 mit der Zahl 0110, oder in gewohnter dekadischer Darstellung 7 plus 6. Wir schreiben die beiden Zahlen wie gewohnt übereinander, sodass die Stellen rechtsbündig übereinander angeordnet sind.

$$\begin{array}{cccc} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ \hline \end{array}$$

Wir addieren die erste Stelle. Falls dies den Wert 0 oder 1 ergibt, ist das auch der Wert der ersten Stelle des Ergebnisses. Der Überlauf zur nächsten

Stelle ist dann Null. Falls es jeddoch 2 ergibt, dann kommt der Wert 0 an die erste Stelle des Ergebnisses und wir haben einen Überlauf von 1 zur nächsten Stelle.

$$\begin{array}{cccc} 0 & 1 & 1 & 1 \\ 0 & 1 & 1_0 & 0 \\ \hline & & & 1 \end{array}$$

Wir fahren fort und rechnen die Werte an der zweiten Stelle und den Überlauf zusammen. Falls das Ergebnis 0, 1 oder 2 ist, handeln wir wie bei der ersten Stelle. Falls wir aber als Ergebnis 3 erhalten, ist der Überlauf zur nächsten Stelle, sowie der Wert des Resultats an der zweiten Stelle gleich Eins.

$$\begin{array}{cccc} 0 & 1 & 1 & 1 \\ 0 & 1_1 & 1 & 0 \\ \hline & & 0 & 1 \end{array}$$

Die übrigen Stellen werden dann ganz analog zur zweiten behandelt.

$$\begin{array}{cccc} 0 & 1 & 1 & 1 \\ 0_1 & 1 & 1 & 0 \\ \hline & 1 & 0 & 1 \end{array}$$

$$\begin{array}{cccc} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ \hline 1 & 1 & 0 & 1 \end{array}$$

Modifizieren wir dieses Verfahren ein wenig. Hierzu bilden wir zunächst die Summe der beiden Zahlen ohne Berücksichtigung der Überläufe.

$$\begin{array}{cccc} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}$$

Diese Operation entspricht einer stellenweisen XOR Operation. Um das korrekte Resultat zu erhalten müssen wir hierzu für die ignorierten Überläufe kompensieren. Bei jeder Stelle haben wir einen Überlauf ignoriert, wenn das logische UND der beiden Stellen 1 ergeben hat.

$$\begin{array}{ccccc} & 0 & 1 & 1 & 1 \\ & 0 & 1 & 1 & 0 \\ \hline & 0 & 1 & 1 & 0 \\ \hline 0 & 1 & 1 & 0 & 0 \end{array}$$

Wir sehen also, dass ein hinzuaddieren des Resultats der um eine Stelle nach links verschobenen stellenweise UND Operation genau für den begangenen Fehler kompensiert.

Satz 7.1. *Seien x und y zwei binäre Zahlen dann gilt:*

$$x + y = XOR(x, y) + LSHIFT(UND(x, y))$$

Im einfachsten Fall ergibt das logische UND für jede Stelle Nullen und wir sind fertig. Im Fall des obigen Beispiels von $7+6$ muss die Umwandlung der Summe ein zweites mal durchgeführt werden und wir haben das Ergebnis bereits errechnet.

$$\begin{array}{cccc} 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ \hline 1 & 1 & 0 & 1 \end{array}$$

Die Frage die sich jetzt noch stellt ist, ob mehrmaliges hintereinander Ausführen dieser Umwandlung immer dazu führt, dass einer der Summanden Null wird. Dies lässt sich mit ja beantworten.

Satz 7.2. *Seien x und y binäre Zahlen der Länge n, dann ergibt n maliges anwenden der Umwandlung aus Satz 7.1 einen linken Summanden der gleich der gesuchten Summe ist und einen rechten Summanden der gleich Null ist.*

Der Satz 7.2 und der Satz 7.1 ermöglichen es uns, die arithmetische Addition mit folgendes Programm durch UND, XOR und Linksverschiebung auszudrücken.

```
WHILE Y
        Z = XOR(X, Y)
        Y = UND(X, Y)
        X = Z
        Y = LSHIFT(Y)
```

Die XOR Operation und die UND Operation lassen sich ja so wie jede binäre Operation durch NAND Operationen ausdrücken. Somit haben wir gezeigt, dass unsere ALU zusammen mit einer Implementierung von WHILE Schleifen die Additions Operation durchführen kann.

In Wahrheit kommen wir auch ohne WHILE-Schleife aus, da wir ja wissen, dass die Umwandlung bei zwei n-Bit lange Zahlen maximal n mal durchgeführt werden muss.

Übungsbeispiel: Zeige, dass sich die Subtraktion analog zur Addition mit Hilfe von logischen Verknüpfungen und links Verschiebung implementieren lässt.

7.2 Gleichheit

Offensichtlich können wir die XOR Operation ausnutzen um in unserer Maschinensprache eine IF Anweisung bedingt auf die Gleichheit oder Ungleichheit zweier Wörter zu verwenden.

7.3 Ein WHILE Program Komplierer

7.4 Beispiele

7.4.1 Acht Bit Addition

7.4.2 Multiplikation

7.4.3 Fibonacci Zahlen