Lernmaterial für das vorgestellte PCD mit integrierter Entwicklungsumgebung

Fabio Jain

Inhaltsverzeichnis

1 Programmaufbau

Das Board kann durch eine eigene Programmiersprache direkt am Display programmiert werden. Die Befehle werden dir in den kommenden Kapiteln vorgestellt. Insgesamt kann der Speicher des Boards bis zu 1000 Befehle Speichern. Es kann sinnvoll sein, das Programm nicht direkt einzugeben, sondern erst einmal eine Zeichnung für den Programmablauf zu erstellen. Diese Zeichnungen siehst du auch in den Beispielen und Lösungen und sie sollen dir helfen, den Programmablauf besser zu verstehen. Ein Programmablauf beginnt immer mit dem:


Startsymbol

Am Programmende steht das:


Endsymbol

Dazwischen gibt es Anweisungen, die dem Board sagen, was es tun soll:

Und Verzweigungen, die bestimmte Teile des Programms nur ausführen, wenn bestimmte Bedingungen erfüllt sind:

In den folgenden Kapiteln wirst du sehen, wie genau sich ein Programmablaufdiagramm aufbaut.

2 Aufgaben

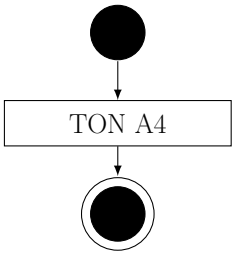
Die Aufgaben zeigen dir Schritt für Schritt, wie du die Befehle benutzen kannst, um das Board zu programmieren. Versuche die Aufgaben in der vorgegebenen Reihenfolge zu bearbeiten. In *Abschnitt 3* findest du eine Übersicht aller Befehle. Im Normalfall sollten aber alle Befehle, die du für eine Aufgabe brauchst in der Aufgabe selbst oder vorherigen Aufgaben erklärt werden.

2.1 Abspielen von Tönen

Dein Board ist mit einem kleinen Lautsprecher ausgestattet, der Töne abspielen kann. Du musst den Board nur sagen, welchen Ton es abspielen soll. Wenn du im Programmiermodus bist, kannst du über die Tastatur Befehle eingeben, die du im Display siehst. Links wird dir außerdem die Zeilennummer angezeigt. Erst einmal möchten wir uns nur den Befehl **TON** anschauen. Dieser aktiviert den Lautsprecher. Hinter dem Befehl geben wir noch den TON an, den der Lautsprecher ausgeben soll. Wir können zwischen den Tönen C, D, E, F, G, A und H wählen. Dazu geben wir noch die Tonhöhe an. C5 ist zum Beispiel eine Oktave höher als C4. Wenn wir den Ton um einen Halbton erhöhen möchten können wir hinter den Buchstaben noch ein Kreuzchen (#) setzen. Aus C5 wird dann C#5. Es macht nichts, wenn du gerade zum ersten mal etwas von Oktaven und Halbtönen hörst. Das ist nicht wirklich wichtig für die kommenden Aufgaben und wir konzentrieren uns erst mal auf einige wenige Töne.

2.1.1 Ein einzelner Ton

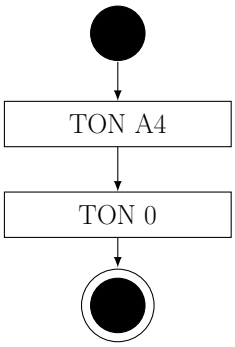
Das einfachste, was wir jetzt machen können, ist dem Board zu sagen, es soll einen einzelnen Ton abspielen. Nehmen wir mal den Ton A4. Um diesen abzuspielen den Befehl **TON A4**. Wenn wir nur diesen Befehl ausführen, sieht unser Programmablauf so aus:



Schalte das Board in den Programmiermodus, gib den Befehl **ABZ 45** ein und gehe wieder in den ausführen-Modus. Jetzt hast du das Programm schon fertig einprogrammiert und solltest den Ton hören. Das Startsymbol und das Endsymbol müssen nicht eingegeben werden, sie vereinfachen nur das Lesen der Ablaufdiagramme. Du hörst den Ton jetzt unendlich lange, weil wir dem Board nicht gesagt haben, dass es den Lautsprecher auch wieder ausschalten soll. Wenn das Programm beendet wird, obwohl der Lautsprecher noch aktiviert ist, bleibt er auch aktiviert. Das wird mit der Zeit schnell nervig, also am besten direkt wieder in den Programmiermodus...

2.1.2 Den Lautsprecher wieder ausschalten

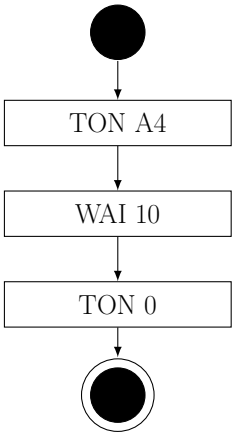
Der Befehl **TON 0** (das ist die Zahl 0 und nicht der Buchstabe O) schaltet den Lautsprecher wieder aus. Vielleicht kann uns dieses Programm helfen, einen Ton abzuspielen und den Lautsprecher wieder auszuschalten?



Gib dafür die Befehle **TON A4** und **TON 0** im Programmiermodus nacheinander ein. Wechsle jetzt in den ausführen-Modus. Du hörst nichts? Kannst du dir vorstellen, warum?

2.1.3 Wartezeiten

Wenn wir den Lautsprecher wie in Kapitel 2.1.2 beschrieben anschalten und daraufhin direkt wieder ausschalten, ist der Ton so kurz, dass wir ihn wahrscheinlich gar nicht hören können. Damit wir etwas hören, müssen wir also nicht nur den Lautsprecher einschalten und ausschalten, sondern zwischendurch auch noch kurz warten. Der Befehl, um das Programm warten zu lassen heißt **WAI** und auch hier geben wir dem Programm noch einen Zahlenwert vor. Nämlich wie lange gewartet werden soll. Dieser Wert ist in $\frac{1}{10}$ Sekunden. Das heißt, der Befehl **WAI 10** wartet genau eine Sekunde und der Befehl **WAI 5** wartet eine halbe Sekunde. Wir wollen den Ton jetzt genau eine Sekunde abspielen und den Lautsprecher danach wieder ausschalten. Das geht durch das Folgende Programm:



2.1.4 Eine Melodie

Jetzt bist du an der Reihe. Versuche für die folgende Aufgabe einen Programmablaufplan zu zeichnen und anschließend das Programm in das Board einzuprogrammieren. Die Töne **H4, G4, A4 und D4** sollen nacheinander mit jeweils einer Sekunde Länge abgespielt werden.

Tipp: Wenn du einen neuen Ton einschaltest, musst du zwischendurch nicht den Befehl **TON 0** verwenden. Der alte Ton wird automatisch deaktiviert, sobald ein neuer Ton aktiviert wird.

Die Lösung dieser Aufgabe findest du in Abschnitt 4.1.

Erkennst du die Melodie? Sie ist Teil einer Glockensequenz, dem sogenannten *Westminster Chime*, die von den Glocken des *Palace of Westminster* in London abgespielt wird. Die Melodie wird aber auch in vielen Schulen als Schulglocke und von manchen Türklingeln verwendet.

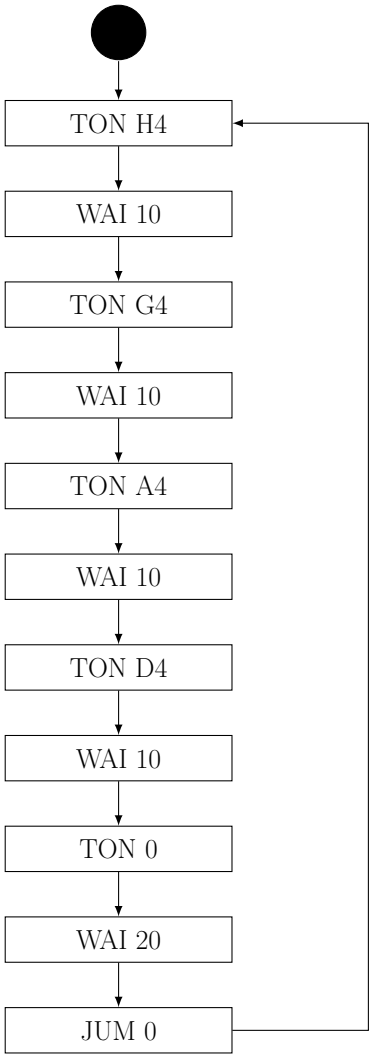
2.1.5 Eine längere Melodie

Wenn du die Melodie noch erweitern möchtest, lautet die längere Tonfolge:
H4, G4, A4, D4, D4, A4, H4, G4
Wie du siehst, ist hier zwei mal der Ton D4 hintereinander. Hier bietet es sich an, eine kleine Pause von einer Sekunde zwischen den beiden Tönen einzubauen, bei der der Lautsprecher deaktiviert ist.

Die Lösung dieser Aufgabe findest du in Abschnitt 4.2.

2.1.6 Tonfolgen wiederholen

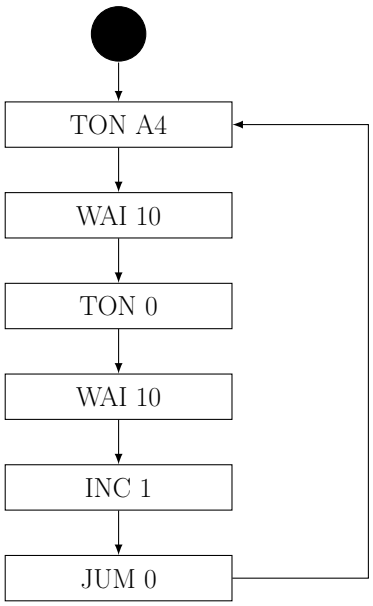
Wir wollen die Tonfolge aus Kapitel 2.1.4 jetzt nicht nur einmal abspielen, sondern unendlich wiederholen. Dazu gibt es mehrere Möglichkeiten. Wir können die Melodie zum Beispiel einfach ganz oft hintereinander einprogrammieren. Aber das kostet viel Zeit und wir haben auch gar nicht so viel Programmspeicher.
Stattdessen können wir auch den Sprungbefehl **JUM** benutzen. Dieser springt zu einer bestimmten Position im Programm. Die Positionen beginnen bei 0, das heißt, der erste Befehl im Programm hat die Position 0, der zweite die Position 1 und so weiter. Das ist später wichtig, um an die richtige Position zu springen. Du kannst die Position einfach an der Zeilennummer links auf dem Display ablesen. Nachdem unsere Melodie beendet ist, können wir einfach zurück an den Anfang springen und es geht wieder von vorne los. Wir bauen nach dem deaktivieren des Lautsprechers aber noch eine Pause von 2 Sekunden ein, damit die Melodien voneinander abgetrennt sind.



Der Befehl **JUM 0** springt jetzt zur ersten Programmposition und die Tonfolge wird unendlich wiederholt. Deswegen gibt es in diesem Programmablaufplan auch kein Endsymbol. Probiere das gleiche mal mit der längeren Tonfolge aus Kapitel 2.1.5. Die Lösung findest du in Kapitel 4.3.

2.1.7 Zählen

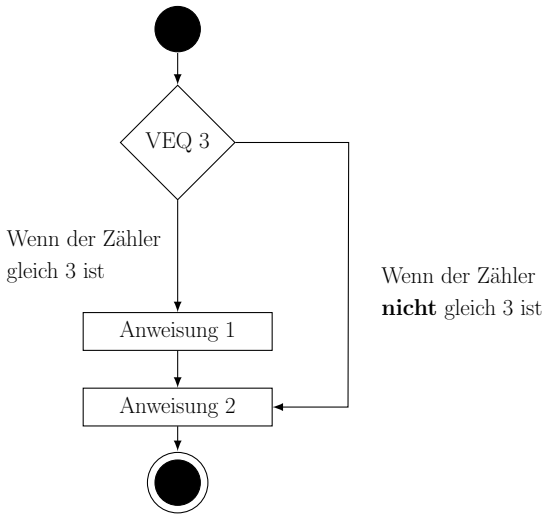
Wir können eine Tonfolge jetzt unendlich oft wiederholen. Wenn wir aber zum Beispiel eine Türklingel programmieren möchten, wäre es sehr unpraktisch, wenn sie unendlich lange klingelt. Wie können wir die Melodie jetzt also zum Beispiel 3 Mal wiederholen? Auch hier könnten wir das Programm aus Kapitel 2.1.4 einfach 3 Mal hintereinander schreiben. Aber das verbraucht wieder viel Programmspeicher und ist mühsam einzugeben. Es gibt eine bessere Möglichkeit. Dazu müssen wir aber erst einmal ein neues Konzept kennenlernen: **Register**
Stell dir vor, du spielst die Melodie selbst auf einem Klavier und zählst mit den Fingern ab, wie oft du sie schon wiederholt hast. Nach der ersten Wiederholung zählst du 1, nach der zweiten 2 und so weiter... Und du schaust nach jeder Wiederholung, ob deine Finger schon 3 zeigen. Dann hörst du auf. So hast du die Melodie genau 3 mal abgespielt.
Genau das macht auch ein Register in deinem Programm. Er ist quasi eine Hand, an der das Programm zählen kann. Zu Anfang des Programms ist das Register immer automatisch auf 0. Wie eine geschlossene Hand. Mit dem Befehl **INC** können wir jetzt eine bestimmte Zahl zum Wert des Registers dazuaddieren. **INC 1** erhöht den Wert des Registers um den Wert 1. So können wir jetzt also Zählen. Angenommen, wir wollen nur einen Ton (A4) 3 Mal hintereinander abspielen. Wie wäre es jetzt mit diesem Code:



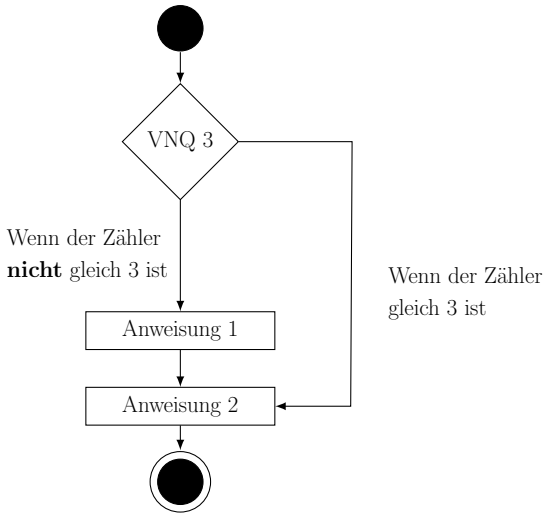
Wenn du den Code zu diesem Diagramm eingibst, wirst du merken, dass der Ton sich trotzdem unendlich wiederholt. Das liegt daran, dass wir die Wiederholungen zwar mitzählen, aber das Programm gar nicht wissen kann, bei welchem Wert es aufhören soll, sich zu wiederholen. Das ist also noch nicht die ganze Lösung...

2.1.8 Ein Register abfragen

Erinnern wir uns noch mal an das Beispiel, bei dem du eine Melodie mit den Fingern abzählst. Hier musst du auch immer schauen, ob die gewünschte Anzahl an Wiederholungen schon erreicht ist. Das müssen wir mit dem Register auch. Dazu gibt es zwei Befehle. **VEQ** prüft, ob das Register einen bestimmten Wert hat. **VNQ** prüft, ob das Register einen bestimmten wert **nicht** hat.
In beiden Fällen springt das Programm zur nächsten Programmposition, wenn die Bedingung erfüllt ist und zur übernächsten, wenn die Bedingung nicht erfüllt ist. Nehmen wir einmal die Abfrage **VEQ 3**, mit der wir Prüfen können, ob das Register gleich 3 ist. Wenn wir diese Abfrage ausführen, passiert das hier:



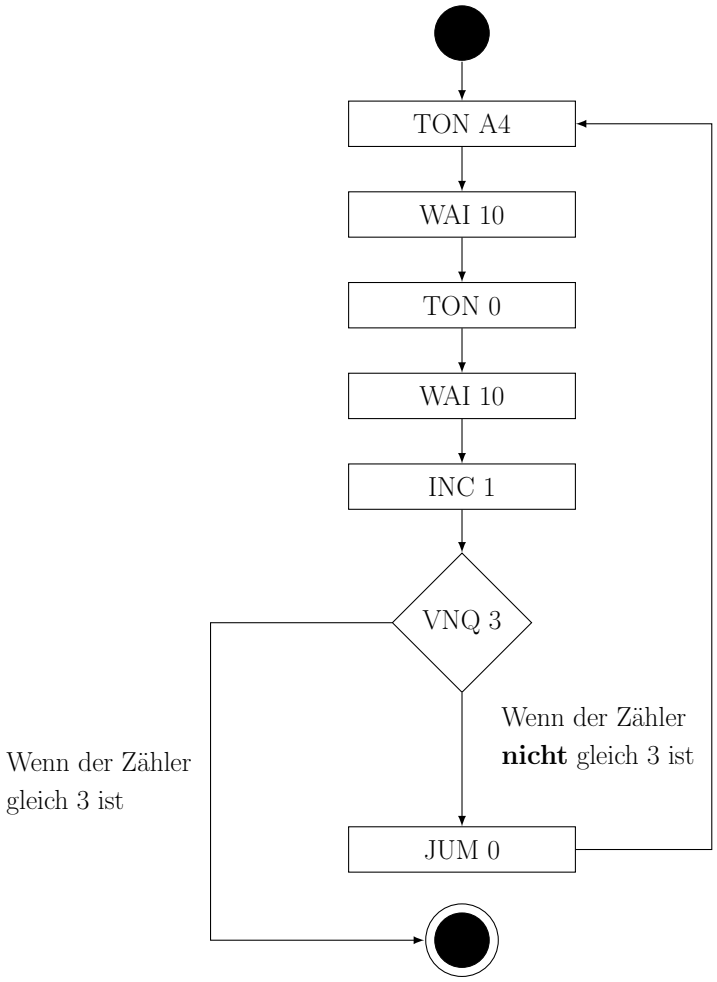
Und für den Befehl **VNQ 3** passiert fast das gleiche, bloß springt das Programm nur dann zur nächsten Position, wenn das Register **nicht** den Wert 3 hat.



Wenn die Bedingung wahr ist, springt das Programm in diesem Beispiel also zu **Anweisung 1** und **Anweisung 2** wird anschließend ausgeführt. Ist die Bedingung nicht erfüllt, springt das Programm direkt zu Anweisung 2. Wenn **Anweisung 1** aber zum Beispiel ein Sprungbefehl ist, kann verhindert werden, dass Anweisung 2 ausgeführt wird.

2.1.9 Zählen und Wiederholen

Die Abfragen aus Kapitel 2.1.8 ergeben nur dann Sinn, wenn sie in Kombination mit Sprungbefehlen (JUM) und Addierbefehlen (INC) verwendet werden. Wir wollen jetzt also endlich versuchen, einen einzelnen Ton 3 mal zu wiederholen. Dazu erhöhen wir das Register nach jeder Wiederholung um 1 und fragen dann ab, ob der Wert 3 erreicht ist. Im folgenden Beispiel verwenden wir **VNQ 3**, weil wir das Programm immer dann wiederholen wollen, wenn das Register **noch nicht** den Wert 3 erreicht hat.



So können wir den Ton 3 mal abspielen. Im Programm, das du in das Board eingibst, ist **JUM 0** der letzte Befehl. Wenn das Register 3 ist, springt das Programm an die Position dahinter, an der nichts mehr steht. Dadurch ist das Programm automatisch beendet.

Versuche jetzt, die Melodie aus Aufgabe 2.1.4 5 Mal hintereinander abzuspielen. Die Lösung findest du in Kapitel 4.4

2.2 Steuerung der LEDs

Jetzt kennst du schon viele der Grundbefehle. Um die beiden LEDs an deinem Board anzusteuern, sind nur zwei weitere Befehle notwendig. **LD1** und **LD2**. Die beiden LEDs sind sogenannte RGB-LEDs. Das heißt sie können nicht nur in einer Farbe, sondern die Farben **R**ot, **G**rün und **B**lau leuchten und sogar Kombinationen dieser Farben anzeigen. Zum Beispiel kann man die LEDs in Violett leuchten lassen, in dem man die Farben Rot und Blau aktiviert. Aber wie geht das jetzt genau?

2.2.1 Einzelne Farben aktivieren und deaktivieren

Um die linke LED zu aktivieren, benutzen wir den Befehl **LD1** und für die rechte LED den Befehl **LD2**. Dazu geben wir noch eine Farbe ein:

LD1 R	Links, Rot
LD1 G	Links, Grün
LD1 B	Links, Blau
LD1 V	Links, Violett
LD1 T	Links, Türkis
LD1 O	Links, Orange
LD1 A	Links, Aus

Diese Befehle funktionieren genau so mit **LD2** für die rechte LED. Mit **LD1 A** bzw. **LD2 A** schaltest du die jeweilige LED wieder aus. Versuche einmal, die linke LED mit jeweils einer Sekunde Pause in grün blinken zu lassen. Die Lösung findest du in Kapitel 4.5.

2.2.2 Alarm

Wir versuchen jetzt die LEDs und den Lautsprecher aus dem vorherigen Kapitel zu kombinieren. Dazu sollen beide LEDs in rot blinken (Zeitabstand: $\frac{1}{2}$ Sekunde) und gleichzeitig immer wenn die LEDs aktiviert sind der Lautsprecher mit dem Ton D4 aktiviert werden.

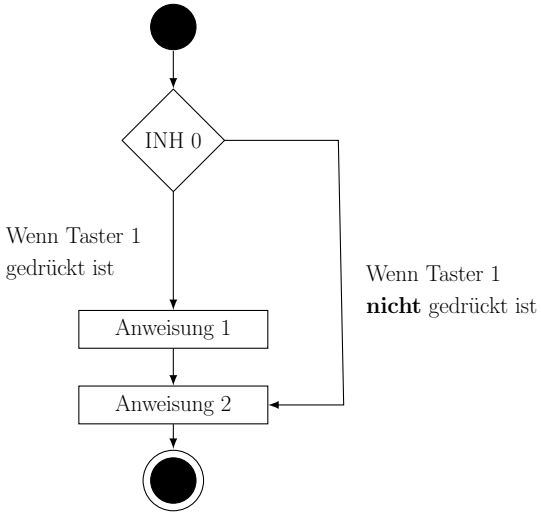
Die Lösung findest du in Kapitel 4.6

2.3 Abfragen der Taster

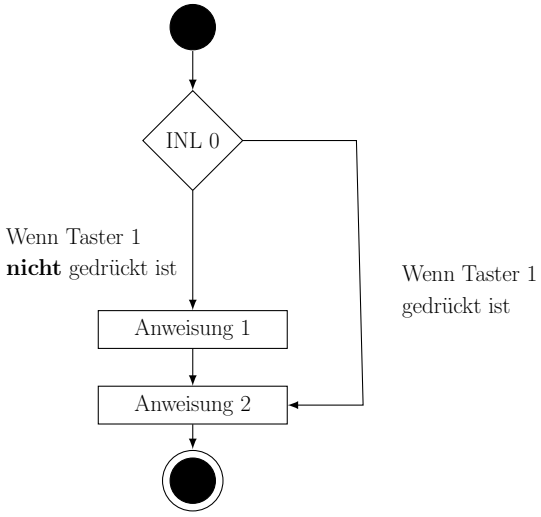
In diesem Kapitel lernst du, wie man die vier Taster auf der Platine abfragt.

Um die Taster auslesen zu können, gibt es zwei Befehle. **INH** prüft, ob ein Taster gedrückt ist. **INH 0** prüft dabei den ersten Taster, **INH 1** den zweiten und so weiter.

Zusätzlich gibt es den Befehl **INL**. Dieser funktioniert genau wie **INH**, prüft aber, ob der Taster **nicht** gedrückt ist. Die Befehle verhalten sich genau wie **VEQ** und **VNQ**. Sie springen zur nächsten Programmposition, wenn die Bedingung wahr ist und zur übernächsten, wen die Bedingung nicht wahr ist.



Und für den Befehl **INL 0** passiert fast das gleiche, bloß springt das Programm nur dann zur nächsten Position, wenn das Register **nicht** den Wert 3 hat.



2.3.1 LEDs mit Taster steuern (1)

Mit diesen Informationen soll jetzt folgendes Programm geschrieben werden:
Die linke LED soll solange deaktiviert sein, bis Taster 1 gedrückt ist. Dann soll sie in grün aktiviert werden und auch aktiviert bleiben, egal wann der Taster wieder losgelassen wird.

Die Lösung findest du in Kapitel 4.7.

2.3.2 LEDs mit Taster steuern (2)

Kapitel 2.3.1 soll so erweitert werden, dass die linke LED immer dann in grün aktiviert wird wenn der Taster gedrückt ist und deaktiviert wird, wenn der Taster nicht gedrückt ist.

Die Lösung findest du in Kapitel 4.8.

2.3.3 LEDs mit Taster steuern (3)

Als nächstes soll die linke LED anfangs in grün leuchten. Wenn Taster 1 gedrückt wird, soll die linke LED ausgeschaltet und die rechte LED in grün eingeschaltet werden. Wird der Taster erneut gedrückt, soll wieder die linke LED aktiviert und die rechte deaktiviert werden usw.

Die Lösung findest du in Kapitel 4.9.

2.3.4 Zählen mit Taster

Wie du im vorherigen Kapitel gesehen hast, ist es wichtig, beim Auslesen von Tastern auch wieder darauf zu warten, dass der Taster losgelassen wird. Diese Schleife braucht man fast immer, wenn man Taster ausliest. Um den Taster und das Register zu verbinden, versuche mal folgendes: Die linke LED soll in rot leuchten, wenn der Taster 10 Mal gedrückt wurde.

Die Lösung zu dieser Aufgabe findest du in Kapitel 4.10

3 Befehlsübersicht

3.1 INC

Erhöht das Register um den gegebenen Wert.
Beispiel: **INC 5** erhöht das Register um den Wert 5. Das Register darf den Wert 255 nicht überschreiten und den Wert 0 nicht unterschreiten.

3.2 DEC

Verringert das Register um den gegebenen Wert.
Beispiel: **DEC 3** verringert das Register um den Wert 3. Das Register darf den Wert 255 nicht überschreiten und den Wert 0 nicht unterschreiten.

3.3 SET

Setzt das Register auf den gegebenen Wert.
Beispiel: **SET 1** setzt den Wert des Registers auf 1. Das Register hat zu Anfang des Programms automatisch den Wert 0.

3.4 VEQ

Prüft, ob das Register gleich dem angegebenen Wert ist. Ist dies der Fall, springt das Programm zur nächsten Programmposition, sonst zur übernächsten.
Beispiel: **VEQ 10** prüft, ob das Register gleich 10 ist.

3.5 VNQ

Prüft, ob das Register **nicht** gleich dem angegebenen Wert ist. Ist dies der Fall, springt das Programm zur nächsten Programmposition, sonst zur übernächsten.
Beispiel: **VNQ 8** prüft, ob das Register **nicht** gleich 8 ist.

3.6 LD1 / LD2

Aktiviert eine bestimmte Farbe der linken oder rechten LED gemäß der folgenden Tabelle:

LD1 R	Links, Rot	LD2 R	Rechts, Rot
LD1 G	Links, Grün	LD2 G	Rechts, Grün
LD1 B	Links, Blau	LD2 B	Rechts, Blau
LD1 V	Links, Violett	LD2 V	Rechts, Violett
LD1 T	Links, Türkis	LD2 T	Rechts, Türkis
LD1 O	Links, Orange	LD2 O	Rechts, Orange
LD1 A	Links, Aus	LD2 A	Rechts, Aus

Figure 1: Befehlstabelle LD1 / LD2

3.7 TON

Aktiviert den Lautsprecher mit dem vorgegebenen Ton bestehend aus Buchstabe (C, D, E, F, G, A, H), Kreuzchen (, optional) und Tonhöhe (1-7). **TON 0** deaktiviert den Lautsprecher.
Beispiel: Der Befehl **TON A4** spielt den Ton A in der Tonhöhe 4.

3.8 INH

Prüft, ob einer der drei Taster gedrückt ist. Ist dies der Fall, springt das Programm zur nächsten Position, sonst zur übernächsten.
INH 0 überprüft den ersten Taster, **INH 1** den zweiten, **INH 2** den dritten und **INH 3** den vierten.

3.9 INL

Prüft, ob einer der drei Taster **nicht** gedrückt ist. Ist dies der Fall, springt das Programm zur nächsten Position, sonst zur übernächsten.
INL 0 überprüft den ersten Taster, **INL 1** den zweiten, **INL 2** den dritten und **INL 3** den vierten.

3.10 WAI

Lässt das Programm in 1/10 Sekunden warten. Der Wert darf 255 nicht überschreiten.
Beispiel: Der Befehl **WAI 10** lässt das Programm eine Sekunde warten.

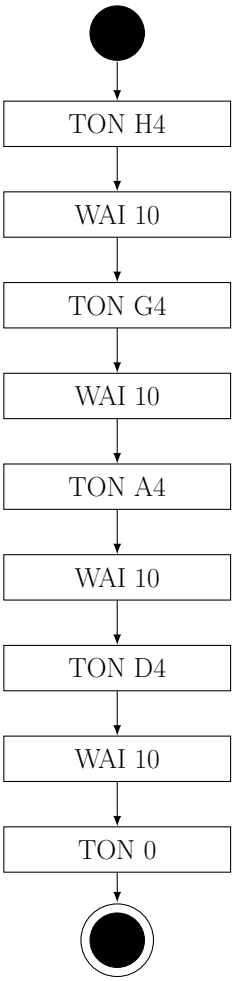
3.11 JUM

Springt zu der gegebenen Programmposition, beginnend bei 0.
Beispiel: Der Befehl **JUM 0** springt zur ersten Programmposition, der Befehl **JUM 1** zur zweiten, usw.

4 Lösungen

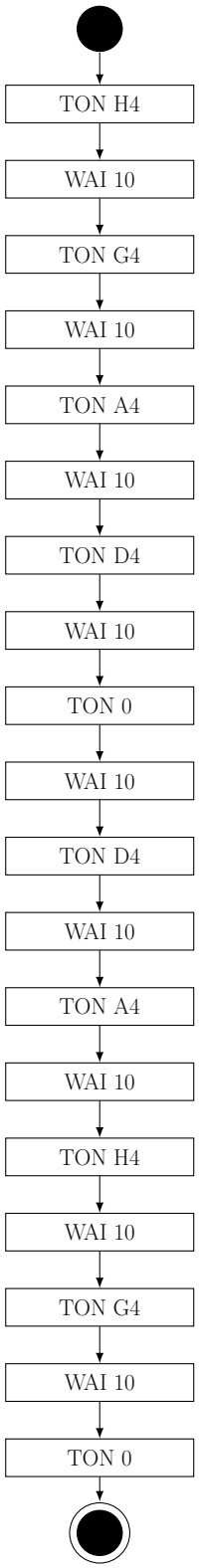
Die Lösungen zeigen jeweils eine Möglichkeit, die Aufgabe zu lösen. Das heißt nicht, dass die gezeigte Lösung die einzig richtige ist!

4.1 Lösung zu Aufgabe 2.1.4



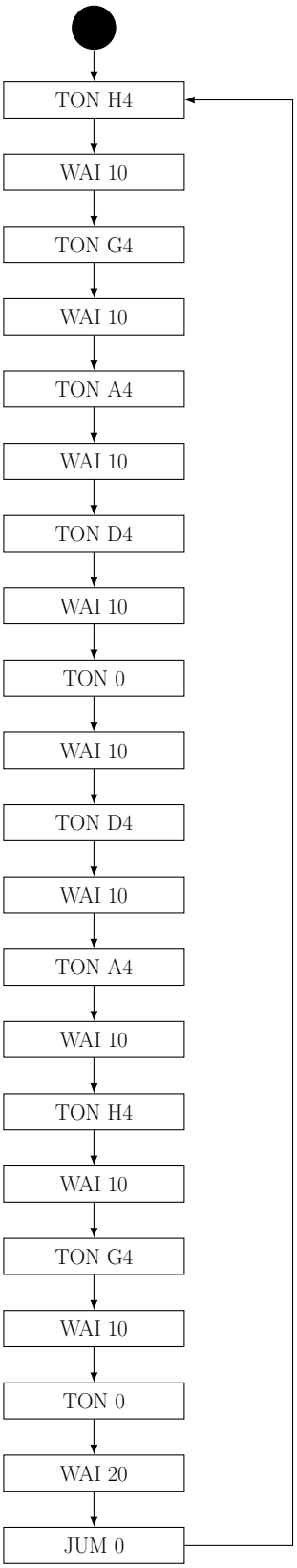
Programmposition	Befehl
0	TON H4
1	WAI 10
2	TON G4
3	WAI 10
4	TON A4
5	WAI 10
6	TON D4
7	WAI 10
8	TON 0

4.2 Lösung zu Aufgabe 2.1.5



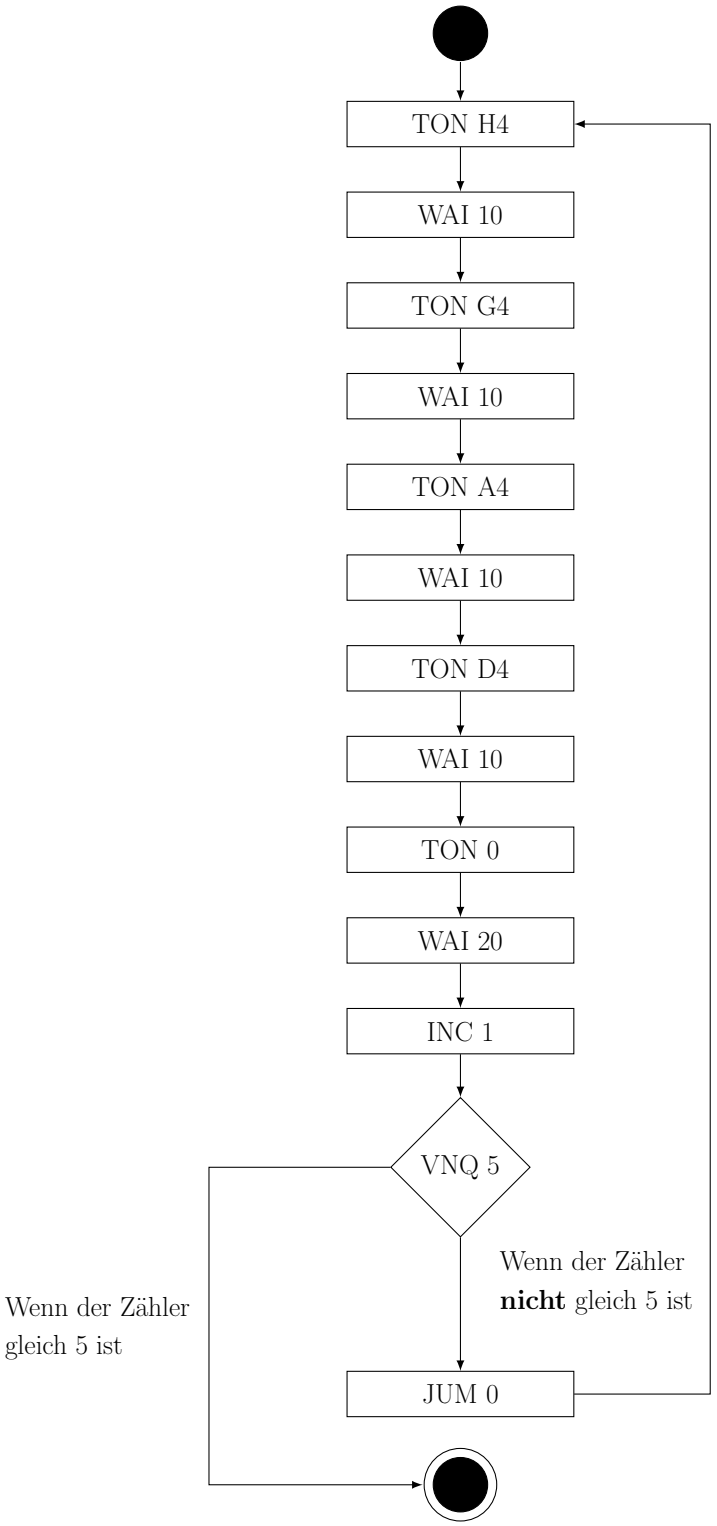
Programmposition	Befehl
0	TON H4
1	WAI 10
2	TON G4
3	WAI 10
4	TON A4
5	WAI 10
6	TON D4
7	WAI 10
8	DBZ
9	WAI 10
10	TON D4
11	WAI 10
12	TON A4
13	WAI 10
14	TON H4
15	WAI 10
16	TON G4
17	WAI 10
18	TON 0

4.3 Lösung zu Aufgabe 2.1.6



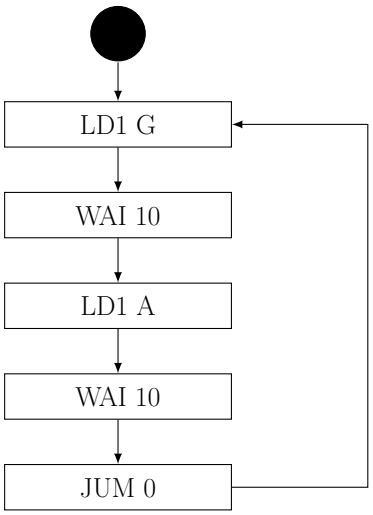
Programmposition	Befehl
0	TON H4
1	WAI 10
2	TON G4
3	WAI 10
4	TON A4
5	WAI 10
6	TON D4
7	WAI 10
8	DBZ
9	WAI 10
10	TON D4
11	WAI 10
12	TON A4
13	WAI 10
14	TON H4
15	WAI 10
16	TON G4
17	WAI 10
18	TON 0
19	WAI 20
20	JUM 0

4.4 Lösung zu Aufgabe 2.1.9



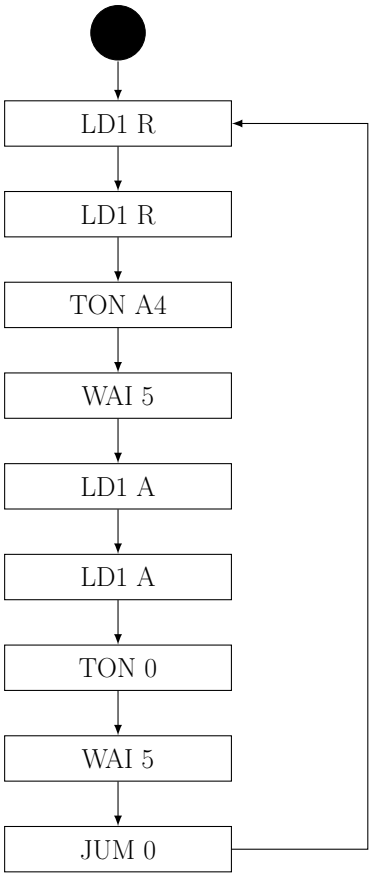
Programmposition	Befehl
0	TON H4
1	WAI 10
2	TON G4
3	WAI 10
4	TON A4
5	WAI 10
6	TON D4
7	WAI 10
8	TON 0
9	WAI 20
10	INC 1
11	VNQ 5
12	JUM 0

4.5 Lösung für Kapitel 2.2.1



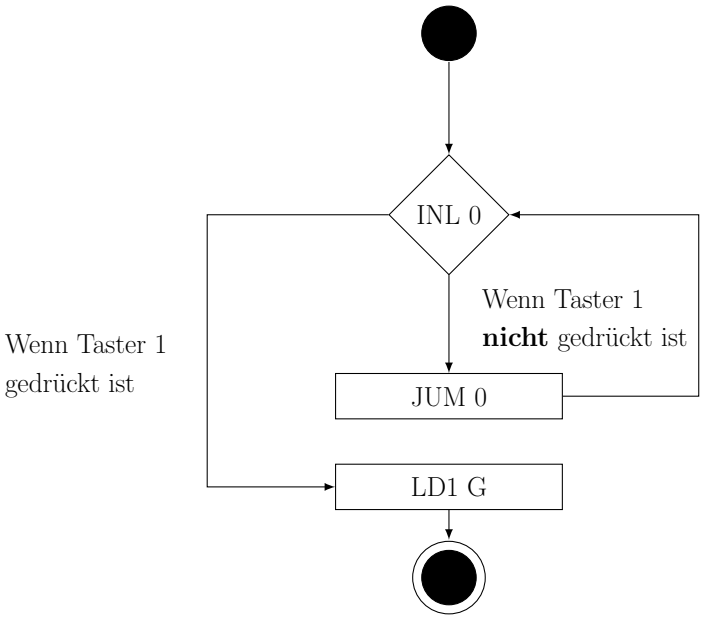
Programmposition	Befehl
0	LD1 G
1	WAI 10
2	LD1 A
3	WAI 10
4	JUM 0

4.6 Lösung für Kapitel 2.2.2



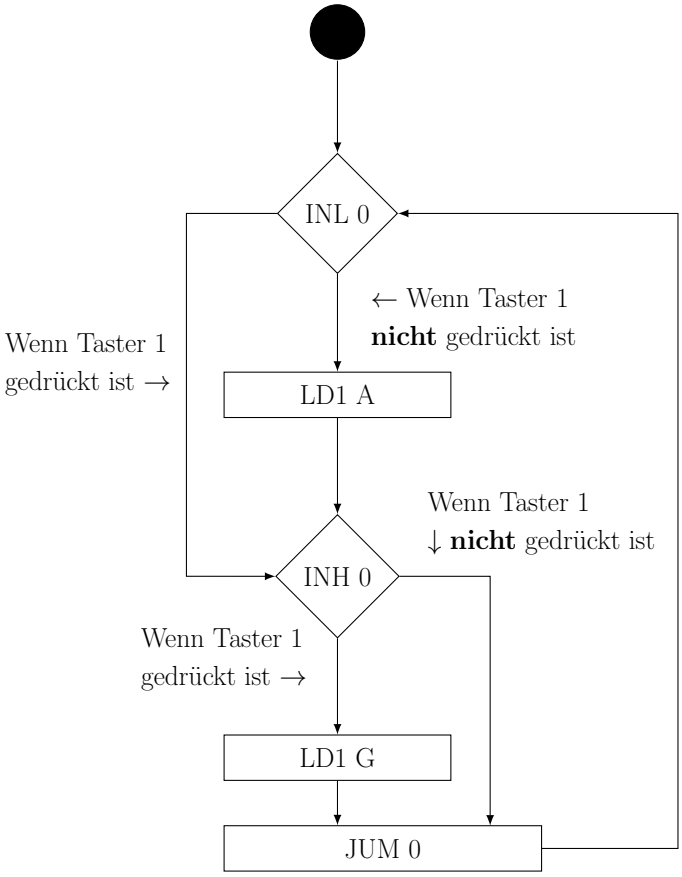
Programmposition	Befehl
0	LD1 R
1	LD2 R
2	TON A4
3	WAI 5
4	LD1 A
5	LD2 A
6	TON 0
7	WAI 5
8	JUM 0

4.7 Lösung zu Aufgabe 2.3.1



Programmposition	Befehl
0	INL 0
1	JUM 0
2	LD1 G

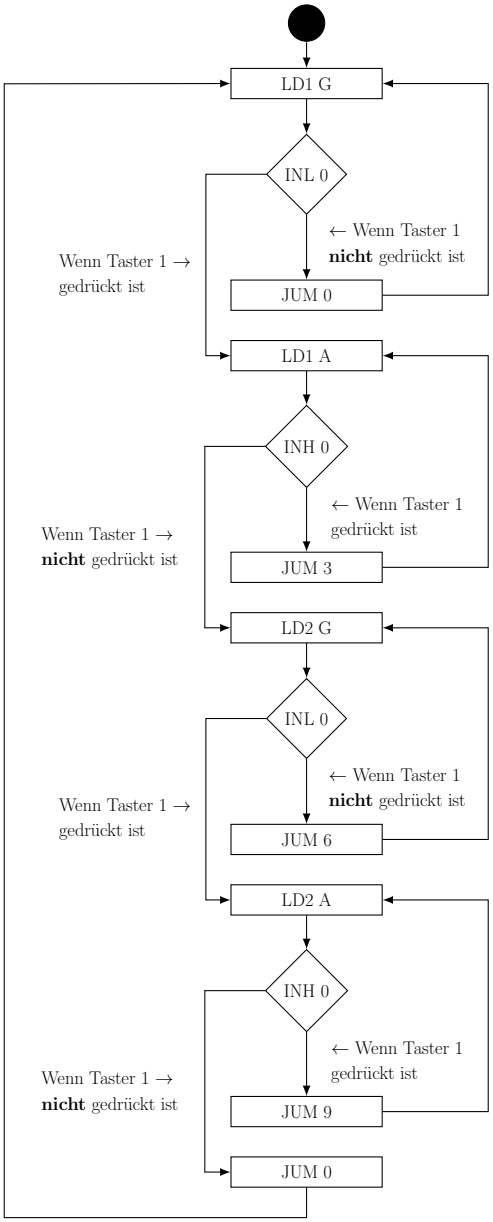
4.8 Lösung zu Aufgabe 2.3.2



Programmposition	Befehl
0	INL 0
1	LD1 A
2	INH 0
3	LD1 G
4	JUM 0

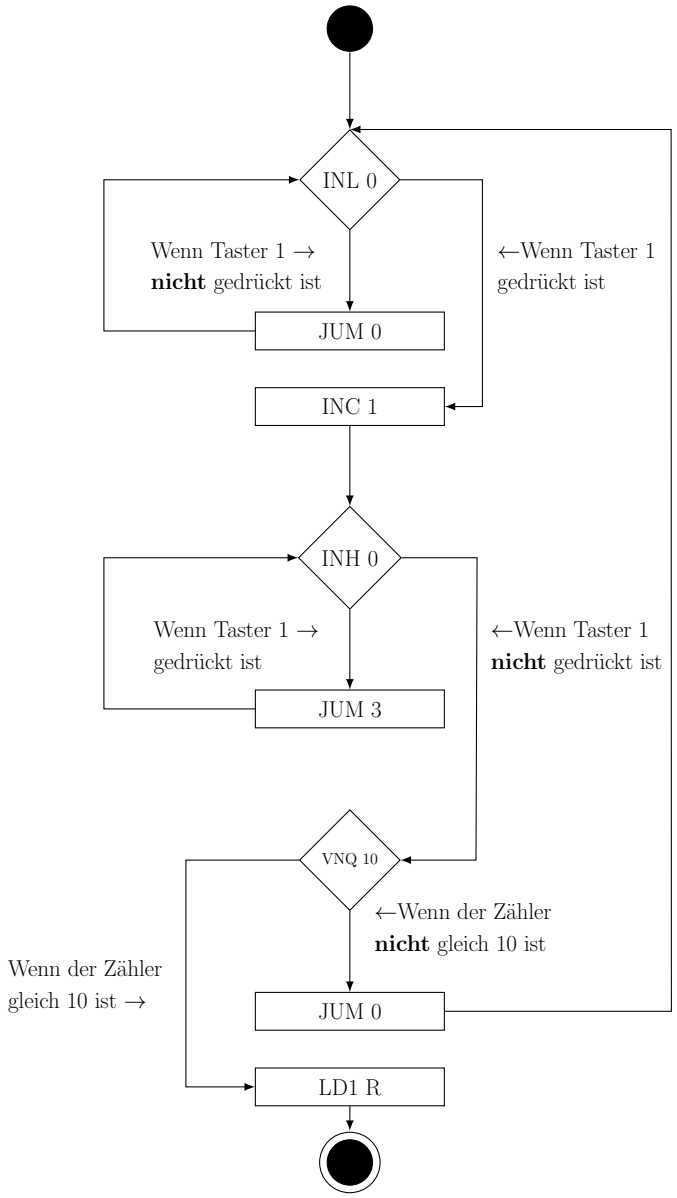
4.9 Lösung zu Aufgabe 2.3.3

Es gibt viele verschiedene Möglichkeiten, diese Aufgabe zu lösen. In diesem Fall geht die LED, die gerade aktiviert ist immer aus, wenn der Taster gedrückt wird und die nächste LED geht erst dann an, wenn der Taster wieder losgelassen wird. Man könnte es genau so gut auch so programmieren, dass das gleichzeitig passiert, wenn der Taster losgelassen wird.
Die Lösung scheint vielleicht unnötig kompliziert. Die Schleifen mit **INH 0** werden benötigt, um sicherzugehen, dass der Taster auch wieder losgelassen wird. Sonst würde sich der Zustand der beiden LEDs sehr schnell hin und her schalten und man wüsste nie, welche der LEDs gerade aktiviert ist, wenn man den Taster wieder loslässt.



Programmposition	Befehl
0	LD1 G
1	INL 0
2	JUM 0
3	LD1 A
4	INH 0
5	JUM 3
6	LD2 G
7	INL 0
8	JUM 6
9	LD2 A
10	INH 0
11	JUM 9
12	JUM 0

4.10 Lösung zu Aufgabe 2.3.4



Programmposition	Befehl
0	INL 0
1	JUM 0
2	INC 1
3	INH 0
4	JUM 3
5	VNQ 10
6	JUM 0
7	LD1 R