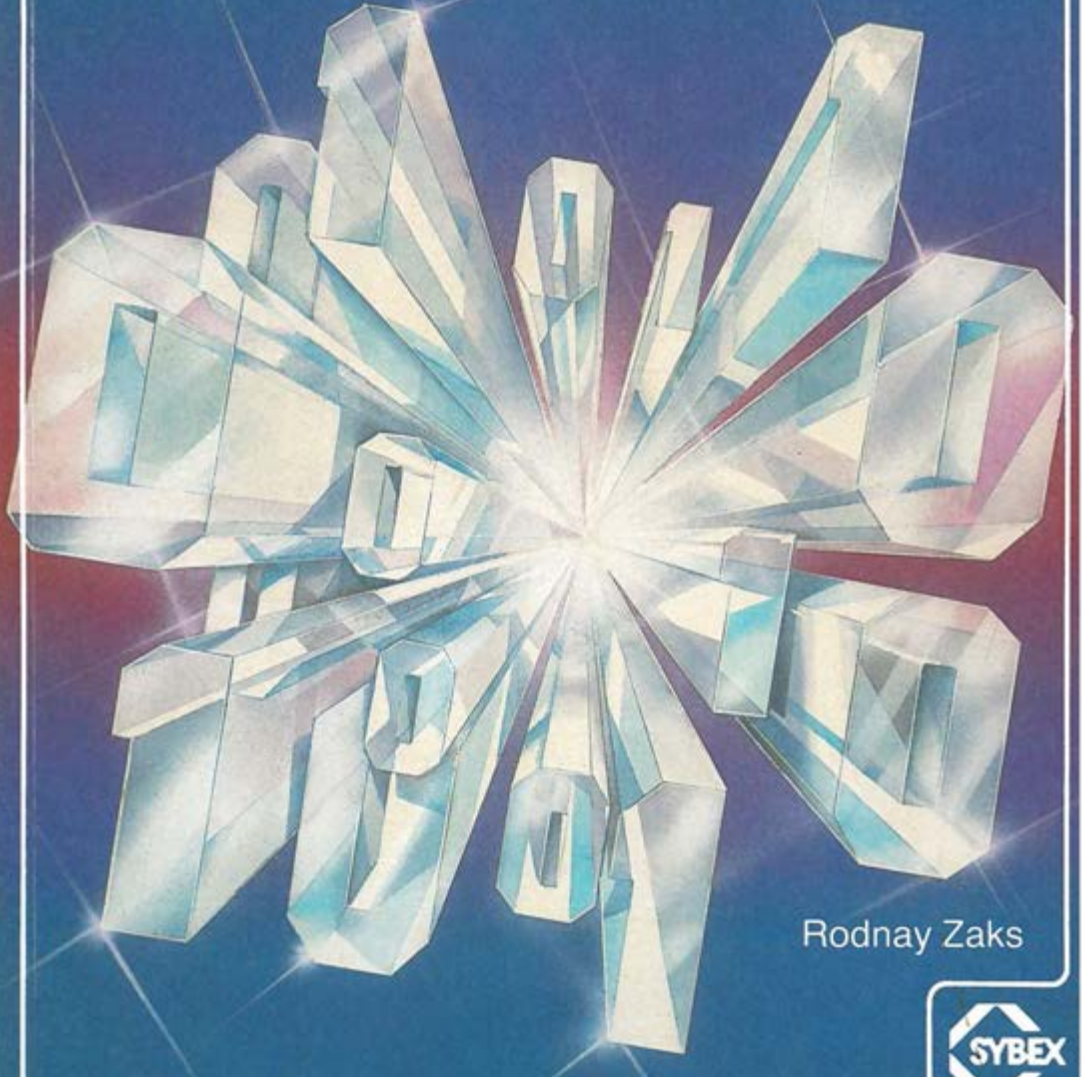
# FORTGESCHRITTENE 6502 PROGRAMMIERUNG



Rodnay Zaks

SYBEX

# Fortgeschrittene 6502-Programmierung

6502 Serie – Band III

# Fortgeschrittene 6502-Programmierung

**Rodnay Zaks**

BERKELEY · PARIS · DÜSSELDORF

# Die 6502-Serie

I Programmierung des 6502
Band II 6502 Anwendungen
Band III Fortgeschrittene 6502-Programmierung

**Anmerkungen:**

SYM ist ein Markenzeichen von Synertek System, Inc.
KIM ist ein Warenzeichen von MOS Technology, Inc.
AIM65 ist ein Warenzeichen von Rockwell International, Inc.

Originalausgabe in Englisch
Titel der englischen Ausgabe: "Advanced 6502 Programming"


**Deutsche Übersetzung: Rainer Gebauer**
Umschlagentwurf: Daniel Le Noury
Satz: tgr – typo-grafik-repro gmbh, remscheid
Gesamtherstellung: Druckerei Hub. Hoch, Dusseldorf



ISBN 3-88745-047-7
1 Auflage 1984

# Vorwort

Dieses Buch soll Ihnen fortgeschrittene Programmiertechniken für den 6502 Mikroprozessor vermitteln, wobei eine systematisch fortschreitende Darstellungsmethode angewendet wird. Die Entwicklung eines Programms besteht aus dem Entwurf eines brauchbaren Algorithmus, geeigneten Datenstrukturen und dem Kodieren des Algorithmus. Bei einem Mikroprozessor wie dem 6502 ist die Entwicklung dieses Algorithmus und der Datenstrukturen normalerweise durch drei Faktoren beeinflußt:

1. Der verfügbare Speicherplatz ist oft begrenzt, das Programm muß also entsprechend gestrafft werden.
2. Es kann erforderlich sein, daß die höchstmögliche Laufgeschwindigkeit für das Programm verwirklicht wird, was eine möglichst effiziente Assemblerkodierung wünschenswert macht, insbesondere durch optimalen Gebrauch der Register.
3. Die spezifische Ein/Ausgabe-Struktur verlangt das Verstehen dieser Bausteine und ihrer Programmierung.

Beim Beurteilen eines Algorithmusentwurfs muß der Programmierer also die Vor- und Nachteile der unterschiedlichen Techniken daran messen, wie gut seine Programmierkenntnisse sind, wieviel Speicherplatz verfügbar ist, welche Laufzeiten erforderlich und wie die allgemeinen Erfolgschancen sind.

Eine fortgeschrittene 6502-Programmierung beinhaltet demnach die Kenntnis aller vom Programm benötigten Chips zusätzlich zu den Programmierungskenntnissen, die die Algorithmen, Datenstrukturen und eine effiziente Registerbenutzung betreffen. Dieses Buch gibt daher einen umfassenden Überblick über alle wichtigen Techniken zur effizienten Programmierung eines 6502-Systems. Das Buch ist als Lehrbuch konzipiert, und so führt jedes Kapitel neue Konzepte, Chips oder Techniken ein. In den letzten Kapiteln werden dann immer komplexere Algorithmen vorgestellt, die die bis dahin erarbeiteten Techniken integrieren.

Aus Gründen einer klaren konzeptionellen Linie bezieht sich das Buch auf ein ganz bestimmtes 6502-System, auf dem alle vorgestellten Programme lauffähig sind; die Einzelheiten dieses Systems sind in Kapitel 1 beschrieben. Trotzdem sind die Programme und Techniken grundsätzlich auf allen 6502-Systemen anwendbar. Ebenso sind alle Programme dieses Buches zwar „nur" Spielprogramme, die vorkommenden Ein/Ausgabe-Techniken bis hin zu den ausgefeilten Echtzeitsimulationen und Interrupt-Anwendungen sind jedoch für die allermeisten Problemstellungen relevant.

Die Vorgehensweise bezieht sich auf „Fallstudien", und jedes Kapitel setzt sich aus folgenden Teilen zusammen:

1. einer Beschreibung der Konzepte und Techniken, die studiert werden sollen;
2. der „Verhaltensweise" des Programms mit typischem Programmablauf, der die zu lösende Aufgabe verdeutlicht;
3. dem Algorithmus, seiner theoretischen Funktionsweise, seinem Aufbau und seinen Vor- und Nachteilen;
4. dem Programm selbst, seinen Datenstrukturen, Programmierkniffen, speziellen Unterprogrammen, alternativen Möglichkeiten und natürlich einem vollständigen Listing.

Dazu kommen Änderungsvorschläge und Übungsaufgaben in jedem Kapitel. Sie werden also zuerst die Definition eines Problems studieren, danach das Programmverhalten kennenlernen und sich schließlich eine mögliche Problemlösung erarbeiten und diese in ein komplettes Programm in der 6502-Assemblersprache übertragen, wobei die Schwerpunkte gesetzt werden in der Datenorganisation, der effizienten Registerverwendung und der richtigen Ein/Ausgabe-Technik.

Beim Entwickeln Ihrer Ein/Ausgabe-Fertigkeiten werden Sie auch die Zeitgeber und Interrupts genau kennenlernen. Vor allem aber werden Sie lernen, wie Sie leichtes Programmieren, Speicherausnutzung, Laufzeit und Algorithmusoptimierung „unter einen Hut" bringen, indem Sie Hardware- und Software-Techniken ganz gezielt anwenden.

Um die fortgeschrittenen Programmiertechniken dieses Buches zu erlernen, ist es nicht notwendig, daß Sie sich die erforderliche Hardware bauen. Sie sollten aber unbedingt Ihre eigenen Programme schreiben, während Sie das Buch durcharbeiten. Der Autor hofft, daß sein Versuch, das Entwerfen von Programmen detailliert zu zeigen und zu erklären, Ihnen ein Stück weiterhilft auf dem Weg zum eigenen Programmieren.

# Inhaltsverzeichnis

# 1

# Einführung

Um die Techniken zu lernen und die Programmbeispiele zu studieren, die in diesem Buch vorgestellt werden, ist an sich keine spezifische Geräteausrüstung erforderlich. Die Verfügbarkeit eines 6502-Systems dürfte jedoch zum Entwikkeln und Testen eigener 6502-Programme von großem Vorteil sein. Sie sollten sich klarmachen, daß alle 6502-Systeme etwas unterschiedliche Ein/Ausgabe-Konfigurationen aufweisen. Die hier vorgeführten Techniken sind jedoch in jedem Fall anwendbar, und die Programme können leicht angepaßt werden, wenn Sie Ein/Ausgabe-Operationen erst einmal verstanden haben.

Zum Lesen dieses Buches sollten Sie mit dem Befehlssatz des 6502 vertraut sein und die grundlegenden Programmiertechniken des Bandes PROGRAMMIERUNG DES 6502 kennen. Gewisse Grundkenntnisse über Ein/Ausgabe-Techniken wären ebenfalls empfehlenswert. (Dieses Thema wurde in den 6502 ANWENDUNGEN umfassend behandelt.)

Die in den Kapiteln 2 bis 11 vorgestellten Programme reichen von einfach bis komplex. Um diese Programme zu implementieren, werden Algorithmen entwickelt und Datenstrukturen entworfen. Diesen Weg muß jeder disziplinierte Programmierer gehen, wenn er für ein vorgegebenes Problem ein Lösungsprogramm entwickelt. Die zehn hier vorgeführten „Fallstudien" werden Sie auch mit gebräuchlichen Ein/Ausgabe-Techniken vertraut machen. Gegen Ende dieses Buches werden Sie feststellen, daß die vorgelegten Aufgaben wachsende intellektuelle Anforderungen stellen, um effiziente Lösungen zu finden. Alle vorgestellten Strategien, auch die in Kapitel 11 auf das Tic-Tac-Toe-Spiel angewandten, sind vermutlich Originale, und diese Strategien und ihr Entwicklungsprozeß werden detailliert analysiert. Als zusätzliche Auflage, die Ihnen effizientes Planen nahebringen soll, wurden alle Algorithmen und Datenstrukturen so konzipiert, daß jedes Programm weniger als 1K freien Speicherplatz braucht. Alle Programme in diesem Buch sind von vielen Anwendern auf der aktuellen Hardware getestet worden, und alle liefen unter den jeweiligen Testbedingungen fehlerfrei. Wie es jedoch in jedem umfangreichen Satz von Programmen der Fall sein kann, wird sich auch hier manches finden, was unangebracht oder verbesserungswürdig ist.

## VERWENDBARE HARDWARE

Die in diesem Buch enthaltenen Programme können auf jedem 6502-System entwickelt werden, für ihre Ausführung benötigen sie allerdings eine spezifische Ein/Ausgabe-Umgebung. Der Einfachheit halber wurde für dieses Buch durchgehend eine einheitliche Hardware-Konfiguration gewählt: der SYM (ein 6502 von SYNERTEK SYSTEMS) und eine zusätzliche Ein/Ausgabe-Platine, das leicht selbst zu bauende „Spielbrett". Der Vollständigkeit halber werden in diesem Kapitel ein Überblick über den SYM und eine ausführliche Beschreibung des Spielbretts gegeben. Für das Verständnis der in diesem Buch vorgestellten Informationen ist es jedoch nicht erforderlich, diese Geräte zu kaufen oder zu bauen. Das Spielbrett kann auch leicht an andere 6502-Systeme angeglichen werden, etwa an APPLE- oder COMMODORE-Computer. Die Programme können dabei im wesentlichen unverändert übernommen werden, lediglich die Ein/Ausgabe-Einrichtungen müssen entsprechend angepaßt werden.

Das Spielbrett kann auch auf einem üblichen Terminal simuliert werden, indem die Informationen auf einem CRT-Schirm wiedergegeben werden und die Eingabe über ein normales alphanumerisches Tastenfeld erfolgt.

Ein Foto des Spielbretts ist in Bild 1.1 gezeigt. Das Tastenfeld rechts wird für Eingaben an den Mikrocomputer benutzt, die LEDs links dienen als Display für die vom Programm ausgegebenen Informationen. Die jeweilige Verwendung der Tasten und LEDs wird in jedem Kapitel erläutert. Für akustische Effekte ist ein Lautsprecher vorgesehen, der für bessere Tonqualität in ein Gehäuse eingebaut werden kann (siehe Bild 1.2). Dieses Ein/Ausgabe-Brett kann mit wenigen preiswerten Bauteilen selbst zu Hause zusammengebaut werden.

## DAS SYSTEM WIRD ANGESCHLOSSEN

Wenn Sie das System und das Ein/Ausgabe-Brett tatsächlich zusammenbauen möchten, lesen Sie bitte weiter. Interessiert Sie dagegen der eigentliche Hardware-Aufbau nicht, so lesen Sie bitte bei der Beschreibung der Tasten-Eingabe-Routine weiter, einem wichtigen Unterprogramm, das in diesem Buch immer wieder verwendet wird.

Das Spielbrett ist aus vier wesentlichen Komponenten zusammengesetzt:

1. der Stromversorgung
2. der SYM-Platine
3. dem Spielbrett
4. einem Kassettenrecorder (empfehlenswert)

Als erstes soll die Stromversorgung angeschlossen werden. Wenn das entsprechende Netzteil nicht bereits damit ausgestattet ist, so ist der Anschluß von zwei Kabelpaaren erforderlich (siehe Bild 1.3): Erstens muß ein Netzkabel angeschlossen werden, zweitens müssen Erde und +5V-Verbindung (entsprechend

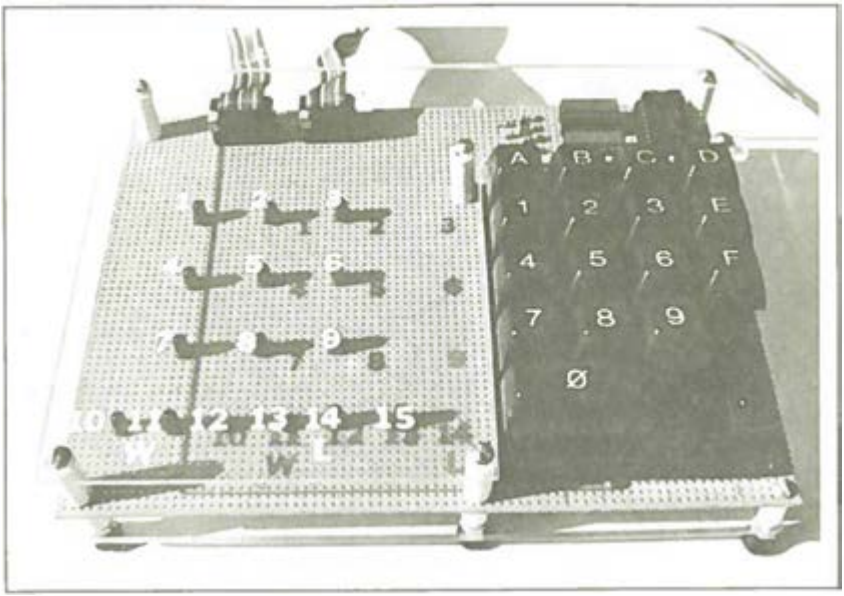



*Abb. 1.1: Das Spielbrett*

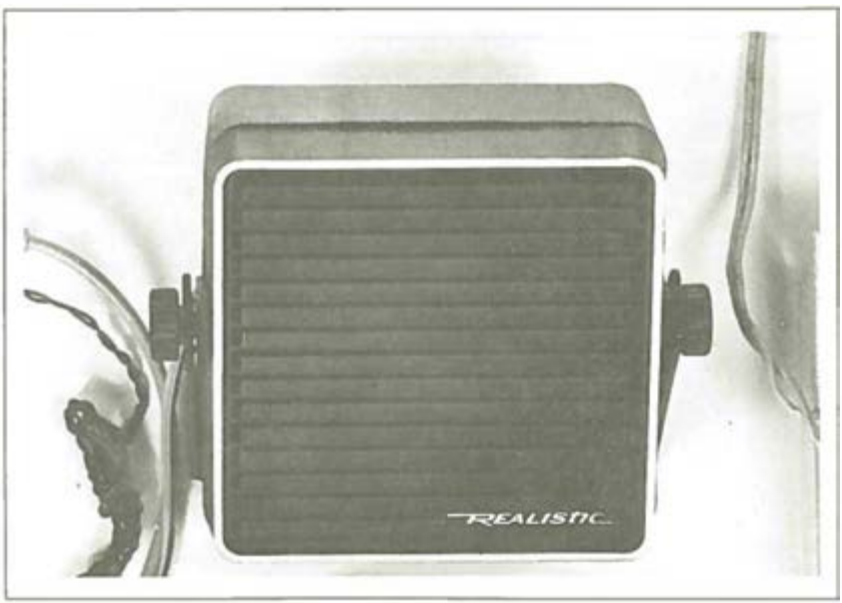



*Abb. 1.2: Lautsprecherchassis zur Klangverbesserung*

der Beschreibung des Herstellers) zum Netzanschlußteil des SYM hergestellt werden.

Als nächstes sollte das Spielbrett mit dem SYM verbunden werden, wobei sowohl der A-Anschluß als auch der AA-Anschluß des SYM benutzt werden (Bild 1.4). Ein Netzanschluß ist beim SYM ebenfalls vorgesehen.

Beim Zusammenstecken der Anschlüsse sollten Sie sorgfältig darauf achten, daß die richtige (meist die beschriftete) Seite nach oben zeigt. Ein falsches Zusammenstecken kann höchst unangenehme Folgen haben. Das gilt besonders für den Netzanschluß, während beim Ein/Ausgabe-Anschluß im allgemeinen kein großer Schaden entsteht.

Soll auch ein Recorder Verwendung finden (was sehr zu empfehlen ist), so muß schließlich auch dieser an den SYM angeschlossen werden. Hier sollten wenigstens der MONITOR- oder der Kopfhörer-Ausgang, vorzugsweise auch der REMOTE-Ausgang angeschlossen werden. Möchte man neue Programme auch auf Band speichern, ist der Anschluß des RECORD- oder des Mikrophon-Ausgangs unerläßlich (Bild 1.5). Einzelheiten zu diesen Anschlüssen finden sich im SYM-Handbuch.

Unser System ist jetzt startklar (Bild 1.6). Wenn Sie eine Spielkassette haben, legen Sie sie nun in den Recorder ein. Schalten Sie den SYM ein, drücken Sie die RST-Taste, um das entsprechende Spiel in den SYM zu laden, und es kann losgehen. Andernfalls geben Sie den hexadezimalen Objekt-Code des Spiels über die SYM-Tastatur ein. Alle Spiele beginnen bei Adresse 200 („GO 200“).

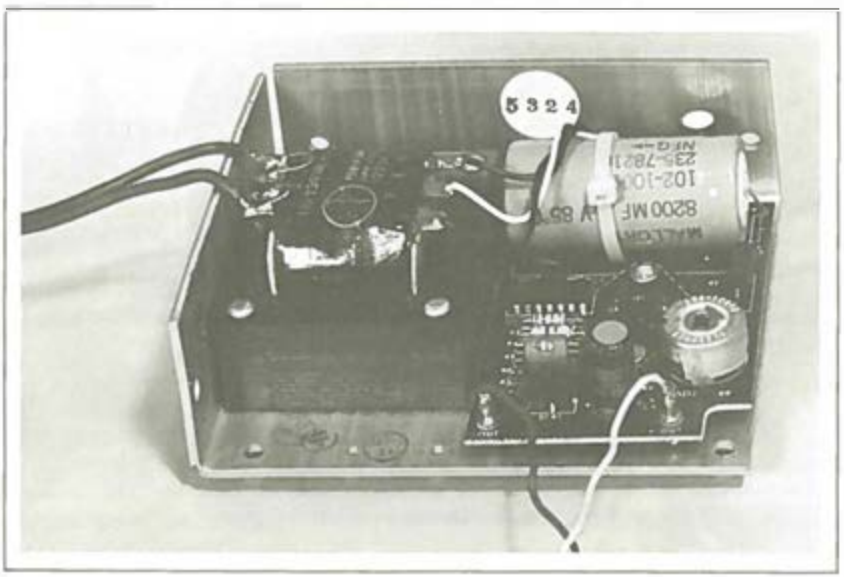



*Abb. 1.3: Zwei Drähte gehen zur Stromversorgung*

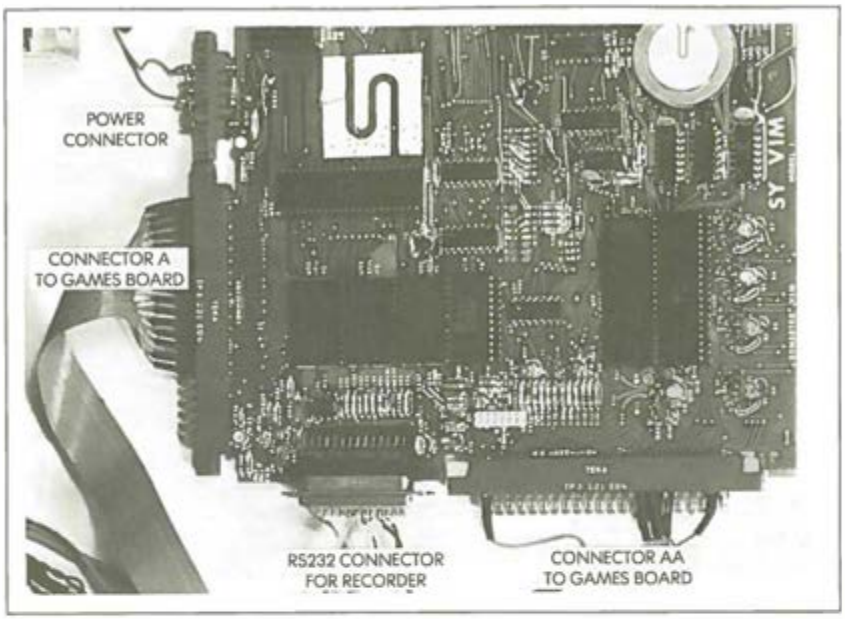



*Abb. 1.4: Zwei Verbindungen zwischen Spielbrett und SYM (dazu Stromversorgungs- und Kassettenrecorderkabel)*

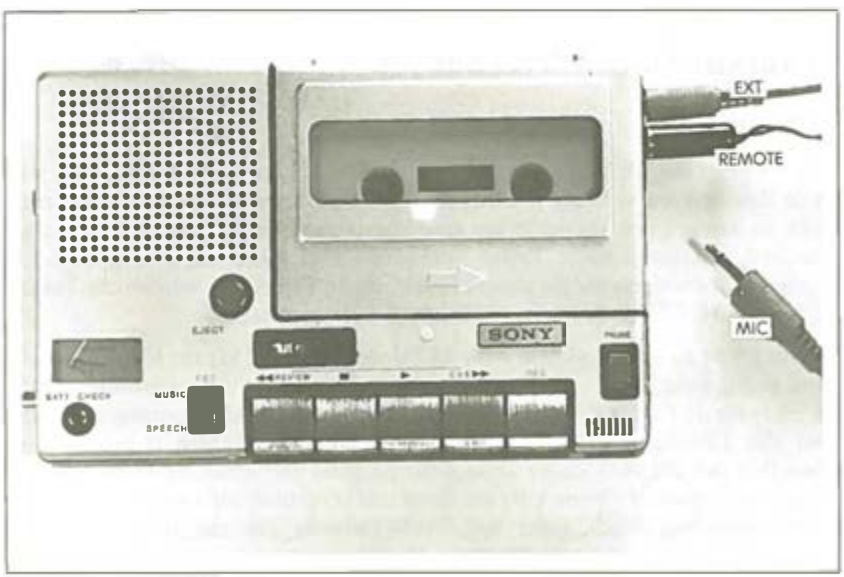



*Abb. 1.5: Verbindung des Recorders*

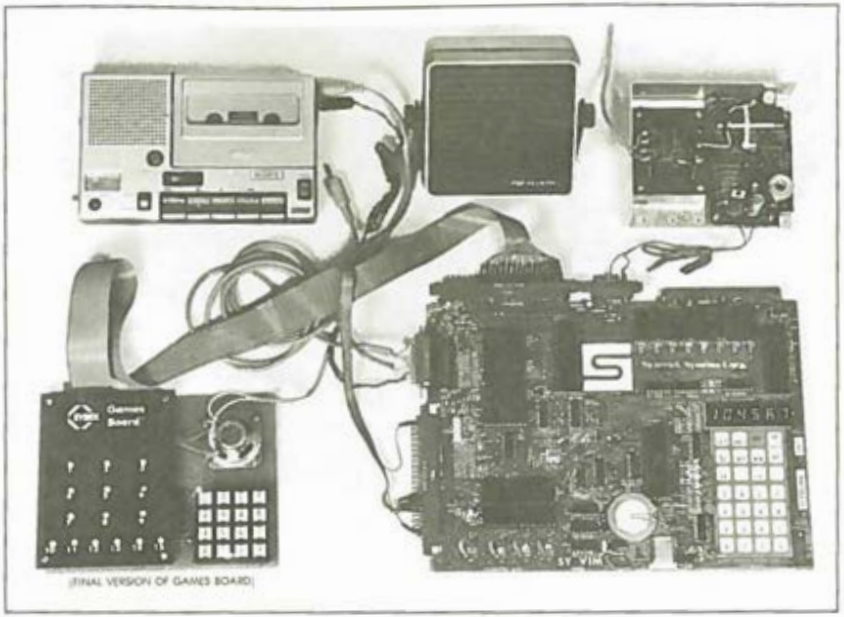



*Abb. 1.6: Das fertige System*

## VERBINDUNG DES SPIELBRETTS

### Die Tastatur

Bild 1.7 zeigt die Elemente des Spielbretts, Bild 1.8 die LED-Anordnung, wie sie in den Spielen verwendet wird. Das Tastenfeld verwendet pro Taste eine Zeile, es handelt sich also nicht um eine Matrix-Anordnung. Obwohl auf vielen Standard-Tastaturen mehr Tasten vorhanden sind (etwa auf der in Bild 1.7 gezeigten), benötigen wir für unsere Spiele nur 16 Tasten. So werden die Tasten H, L und SHIFT des Prototyps in Bild 1.7 nicht benutzt.

In Bild 1.9 ist zu sehen, wie ein 4-bis-16-Decoder (der 74154) zur Identifizierung einer gedrückten Taste verwendet wird, wobei nur vier Ausgabeleitungen (PB0 bis PB3) für 16 Codes verwendet werden. Das Tastatur-Abfrageprogramm gibt über die Leitungen PB1 bis PB3 nacheinander die Zahlen 0 bis 15 aus, woraufhin der 74154-Decoder seine 4-Bit-Eingabe nacheinander in jede der 16 Ausgaben decodiert. Wenn z.B. die Binärzahl 0000 über die Leitungen PB0 bis PB3 ausgegeben wird, erdet der 74154-Decoder die zur 0-Taste korrespondierende Leitung 1 (siehe Bild 1.9). Nach Ausgabe jeder 4-Bit-Kombination liest das Abfrageprogramm den Wert von PA7. War die gerade geerdete Taste nicht gedrückt, so ist PA7 logisch hoch. War die Taste gedrückt, so ist PA7

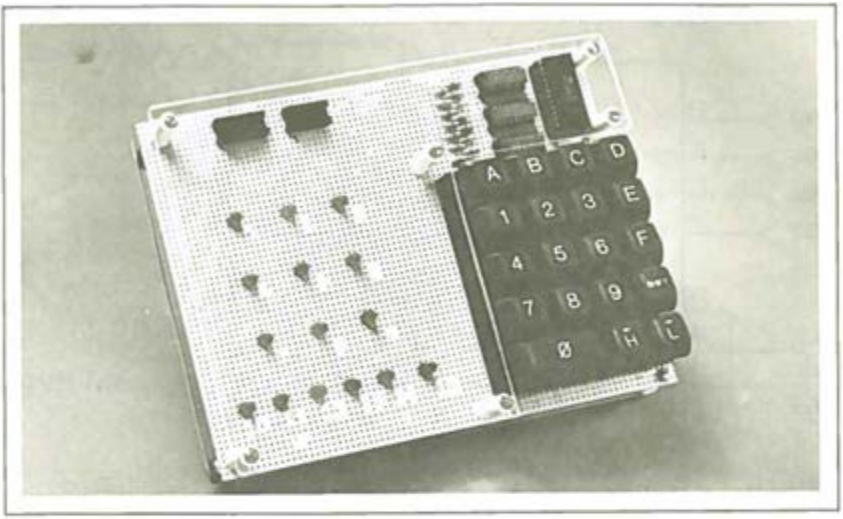



*Abb. 1.7: Das Spielbrett*

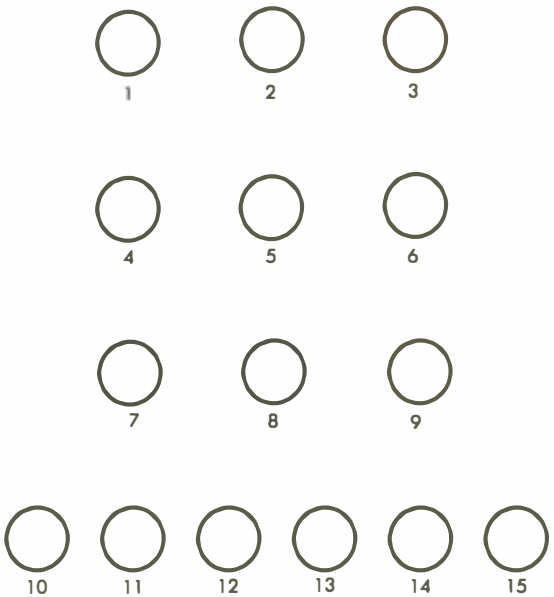



*Abb. 1.8: Die LEDs*

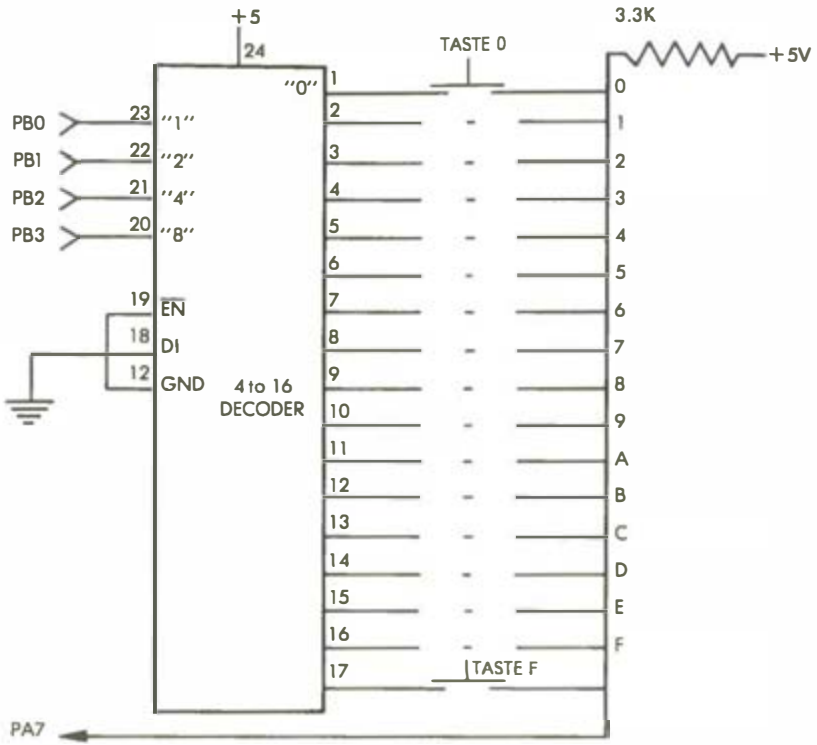



*Abb. 1.9: Dekoder-Tastatur-Verbindung*

geerdet, und es wird eine logische 0 gelesen. In Bild 1.10 ist beispielsweise erkannt worden, daß Taste 1 gedrückt wurde. Bei jedem Abfrage-Algorithmus wird ein gutes Programm die Tastenschlüsse durch eine Verzögerung „entprellen". Für eine detailliertere Beschreibung von speziellen Tastatur-Schnittstellen-Techniken sei auf das Buch MIKROPROZESSOR INTERFACE TECHNIKEN (Ref. Nr. 3012) verwiesen.

In unserer Konfiguration sind die vier Eingabeleitungen zum 74154 (PB0 bis PB3) mit dem VIA 3 des SYM verbunden, mit dem auch PA7 verbunden ist. Der 3.3 K Widerstand rechts oben in Bild 1.9 hält PA7 hoch und garantiert eine logische 1, solange nicht geerdet wird.
Das GETKEY-Programm oder eine ähnliche Routine wird in allen Programmen dieses Buches benutzt. Eine Beschreibung folgt weiter unten.

### Die LEDs

Wie die 15 LEDs verbunden werden, zeigt Bild 1.10. Den nötigen Strom von 16 mA liefern drei 7416 LED-Treiber.

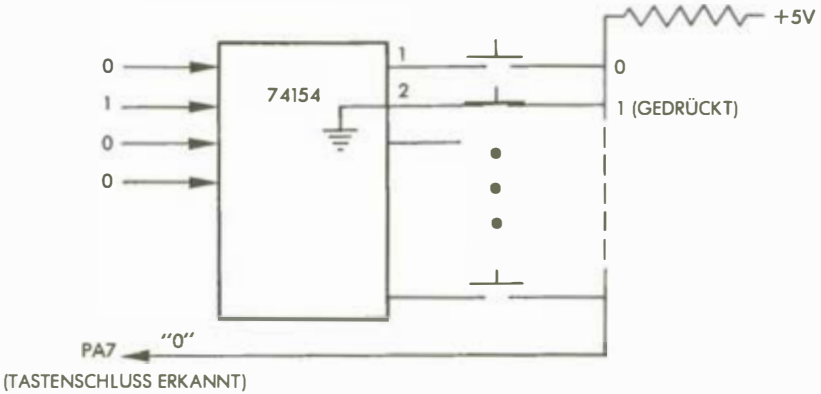



*Abb. 1.10: Erkennen eines Tastenschlusses*

Die LEDs sind mit den Leitungen PA0 bis PA7 und PB0 bis PB7 – ausgenommen PB6 – verbunden. Diese Tore gehören zum VIA 1 des SYM. Eine LED wird dadurch zum Leuchten gebracht, indem einfach der richtige Eingangskontakt zum jeweiligen Treiber ausgewählt wird. Die sich daraus ergebende Anordnung wird in den Bildern 1.12 und 1.13 gezeigt. Die in Bild 1.11 angegebenen Widerstände sind 330 Ohm-Widerstände, die als Strom-Begrenzer für die 7416-Tore dienen.

Die Ausgabe-Routinen werden im Zusammenhang mit spezifischen Spielen beschrieben.

**Erforderliche Teile**

ein 6 × 9 inch Vektorbrett
ein 4-bis-16-Decoder (74154)
drei invertierende Hex-Treiber
eine 24er Steckerplatte
drei 14er Steckerplatten (für die Treiber)
eine 16er Tastatur (uncodiert)
15 Widerstände à 330 Ohm
ein 3.3 K Ohm-Widerstand
ein Entkopplungskondensator (0.1 mF)
15 LEDs
ein Lautsprecher
ein 50 Ohm- oder 110 Ohm-Widerstand (für den Lautsprecher)
zwei 40 bis 50 cm lange 16-adrige Bandkabel
eine Packung Drahtwickelanschlüsse
Wickeldraht
Lötmetall

Ebenfalls erforderlich sind ein Lötkolben und Wicklungswerkzeug.

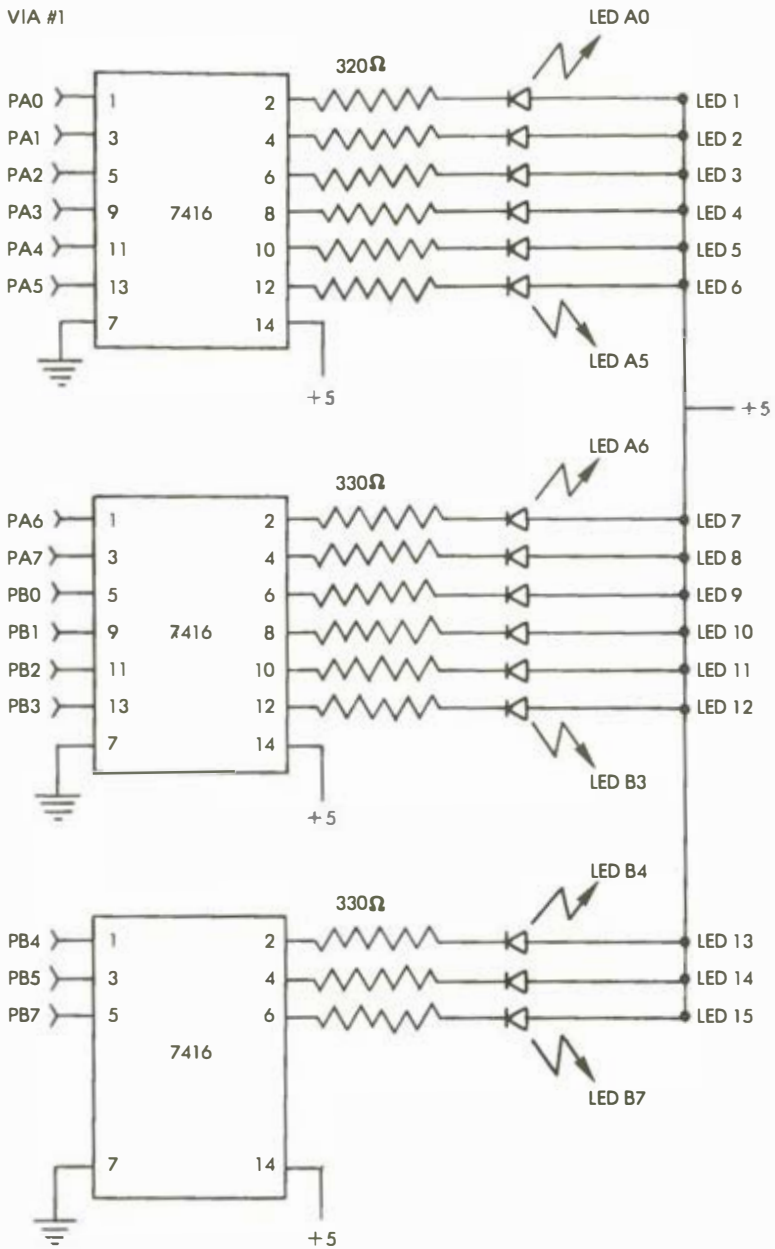



*Abb. 1.11: Die LED-Verbindungen*

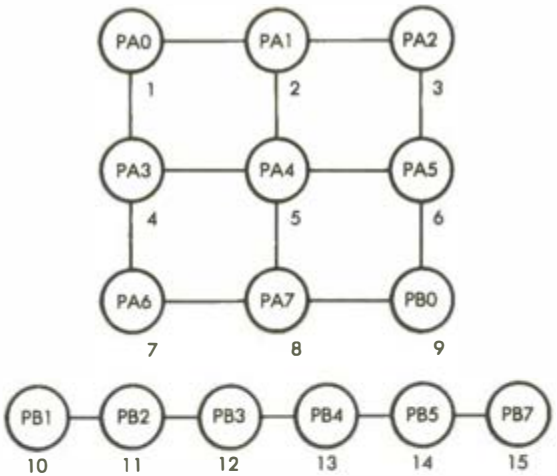



*Abb. 1.12: LED-Anordnung auf dem Spielbrett*

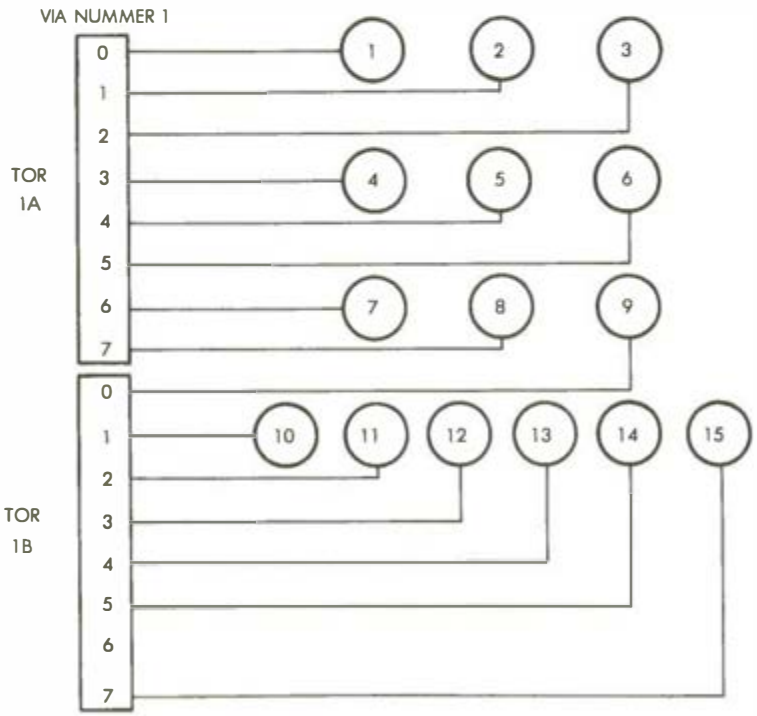



*Abb. 1.13: LED-Verbindungen zu den Toren im Detail*

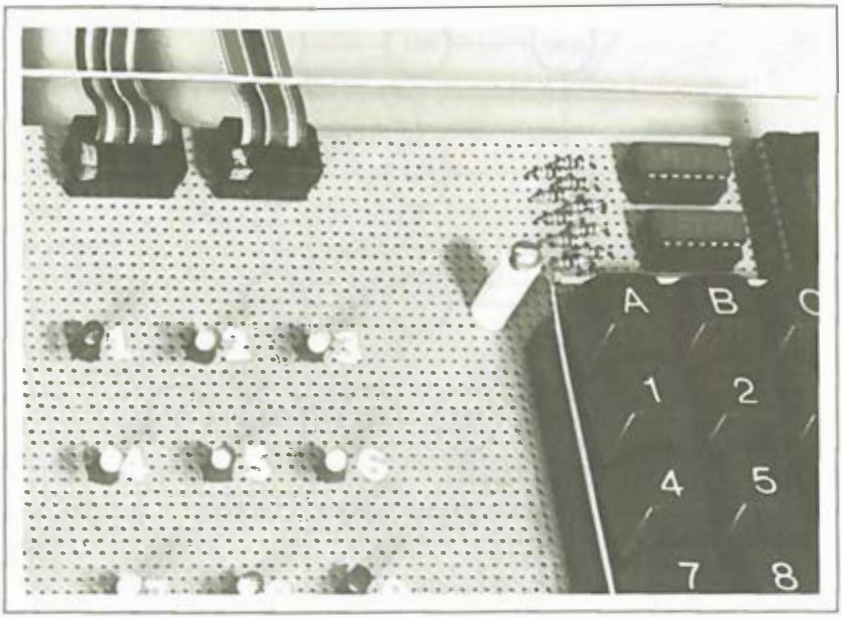



*Abb. 1.14: Das Spielbrett im Detail*

**Zusammenbau**

Eine mögliche Bauweise ist folgende: Die Tastatur kann direkt auf die Lochplatte geklebt werden. Die auf der Platte positionierten Stecker und LEDs können vorübergehend mit Klebestreifen befestigt werden, danach können alle Verbindungen gewickelt werden. Beim vorliegenden Prototyp sind alle Tastaturverbindungen gelötet worden, um zuverlässige Verbindungen zu schaffen. Für gewöhnliche Verbindungen wurden Drahtwickelanschlüsse verwendet.

Beim Prototyp wurden außerdem zwei Stecker vorgesehen, die dem leichteren Anschluß der Bandkabelverbindung an das Spielbrett dienen. Sie sind zwar nicht unverzichtbar, zu ihrer Verwendung sei jedoch dringend geraten, um das Stecken und Ziehen der Kabel zu erleichtern (sie sind in Bild 1.14 links oben zu sehen). Verwendet wurden ein 14er und ein 16er Stecker. Statt der Stecker können auch Drahtwickelanschlüsse benutzt werden, um die Bandkabel direkt auf der Lochplatte zu befestigen. Das andere Ende des Bandkabels wird einfach an den Kantenverbindern des SYM angebracht. Beim Anschluß des Bandkabels an eins der Enden ist sorgfältig darauf zu achten, daß die Stifte richtig sitzen (das Kabel darf nicht verkehrt herum eingesteckt werden). Die Stromversorgung des Spielbretts erfolgt vom SYM aus über die Bandkabelverbindung, und verkehrtes Stecken würde mit Sicherheit unerfreuliche Folgen haben.

Der Lautsprecher kann an jeden Ausgangstreiber des VIA 3 (PB4, PB5, PB6 oder PB7) angeschlossen werden. Jedes dieser Ausgangstore ist mit einem

Transistor-Puffer ausgestattet. Ein 110 Ohm-Widerstand ist mit dem Lautsprecher in Serie geschaltet.

## DIE TASTEN-EINGABE-ROUTINE

Diese Routine, die wir GETKEY nennen wollen, ist eine Dienstroutine, die überprüft, ob eine Taste gedrückt wurde und, wenn ja, um welche Taste es sich handelt. Der dazu gehörige Code wird im Akkumulator abgelegt. Prell-, Wiederholungs- und Rollover-Effekte werden ebenfalls berücksichtigt.

Prellvorgänge werden durch eine Verzögerung von 50 ms beim Registrieren eines Tastenkontaktschlusses vermieden.

Das Wiederholungsproblem (Dauertastendruck) wird gelöst, indem auf das Loslassen einer gerade gedrückten Taste gewartet wird, ehe ein neuer Wert akzeptiert wird. Dies ist z.B. der Fall, wenn eine Taste für längere Zeit gedrückt bleibt. Beim Einsprung in die GETKEY-Routine kann es auch sein, daß eine Taste bereits gedrückt ist. Dies wird so lange ignoriert, bis das Programm feststellt, daß die Taste losgelassen worden ist, erst dann wartet es auf den nächsten Tastendruck. Wenn das die GETKEY-Routine benutzende Programm längere Berechnungen durchführt, besteht die Möglichkeit, daß der Anwender eine neue Taste drückt, ehe GETKEY erneut aufgerufen werden kann. Auch ein solcher Tastendruck wird von GETKEY ignoriert, und die Tasteneingabe muß wiederholt werden.

Die meisten der in diesem Buch vorgestellten Programme geben akustische Signale: Wenn vom Spieler eine Eingabe erwartet wird, wird ein Ton erzeugt. Dazu ist anzumerken, daß bei der Tonerzeugung und auch beim Durchlaufen von Warteschleifen Tastendrücke völlig wirkungslos bleiben.

Bild 1.9 zeigt die Hardware-Konfiguration für die GETKEY-Routine. Der entsprechende Ein/Ausgabe-Chip des SYM ist ein Bild 1.15 dargestellt. Für die Kommunikation zwischen Mikroprozessor und Spielbrett wird der VIA 3 des SYM benutzt. Das für die Ausgabe konzipierte Tor B des VIA führt mit den Leitungen 0 bis 3 zum 74154 (4-bis-16-Decoder), der seinerseits die Verbindung zur Tastatur selbst darstellt. Die GETKEY-Routine gibt die hexadezimalen Zahlen 0 bis F in Folge an den 74154 weiter, was zur Erdung des zugehörigen Ausgabekanals des 74154 führt. Ist eine Taste gedrückt, so wird Bit 7 des Tores 3A des VIA geerdet. Die Programmlogik ist also recht einfach; Bild 1.16 zeigt das zugehörige Flußdiagramm.

Das Programm selbst ist in Bild 1.17 abgebildet. Wir wollen es uns nun genauer ansehen. Die GETKEY-Routine ist frei verschiebbar, kann also an beliebiger Stelle im Speicher untergebracht werden. Aus Platzgründen wurde sie in den Speicherbereich 100 bis 12E gelegt, wobei es sich zu erinnern gilt, daß dies der untere Bereich des Stapelspeichers ist: Ein Anwenderprogramm, das den kompletten Stapelspeicher benutzt, würde die Routine also überschreiben und folglich zerstören. Um dem vorzubeugen, könnte die Routine zwar auch an anderer Stelle untergebracht werden, für alle Programme jedoch, die in diesem Buch dargestellt werden, ist der angesprochene Speicherbereich richtig gewählt.

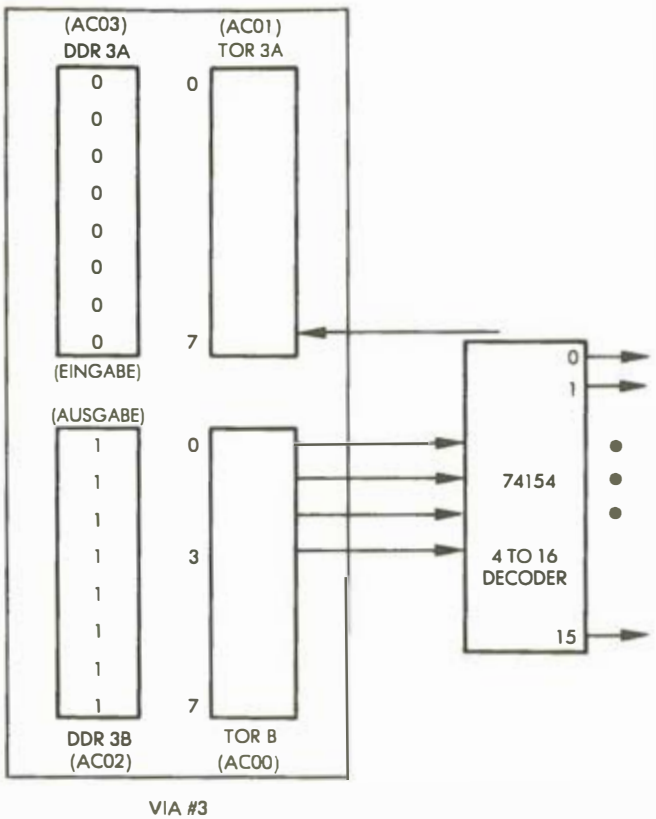



*Abb. 1.15: VIA-Verbindung zum Tastatur-Dekoder*

Die vier ersten Befehle der Routine konditionieren die Datenrichtungsregister des VIA 3. Das Datenrichtungsregister von Tor A wird auf Eingabe gesetzt (nur Nullen), das von Tor B auf Ausgabe (nur Einsen). In Bild 1.15 ist dies dargestellt.

```
LDA #0
STA DDR3A
LDA #$FF
STA DDR3B
```

Um Bit 7 von Tor 3A zu testen – was Aufschluß darüber gibt, ob ein Tastenkontakt geschlossen wurde oder nicht –, sind zwei Befehle erforderlich:

```
START   BIT TOR3A
        BPL START
```

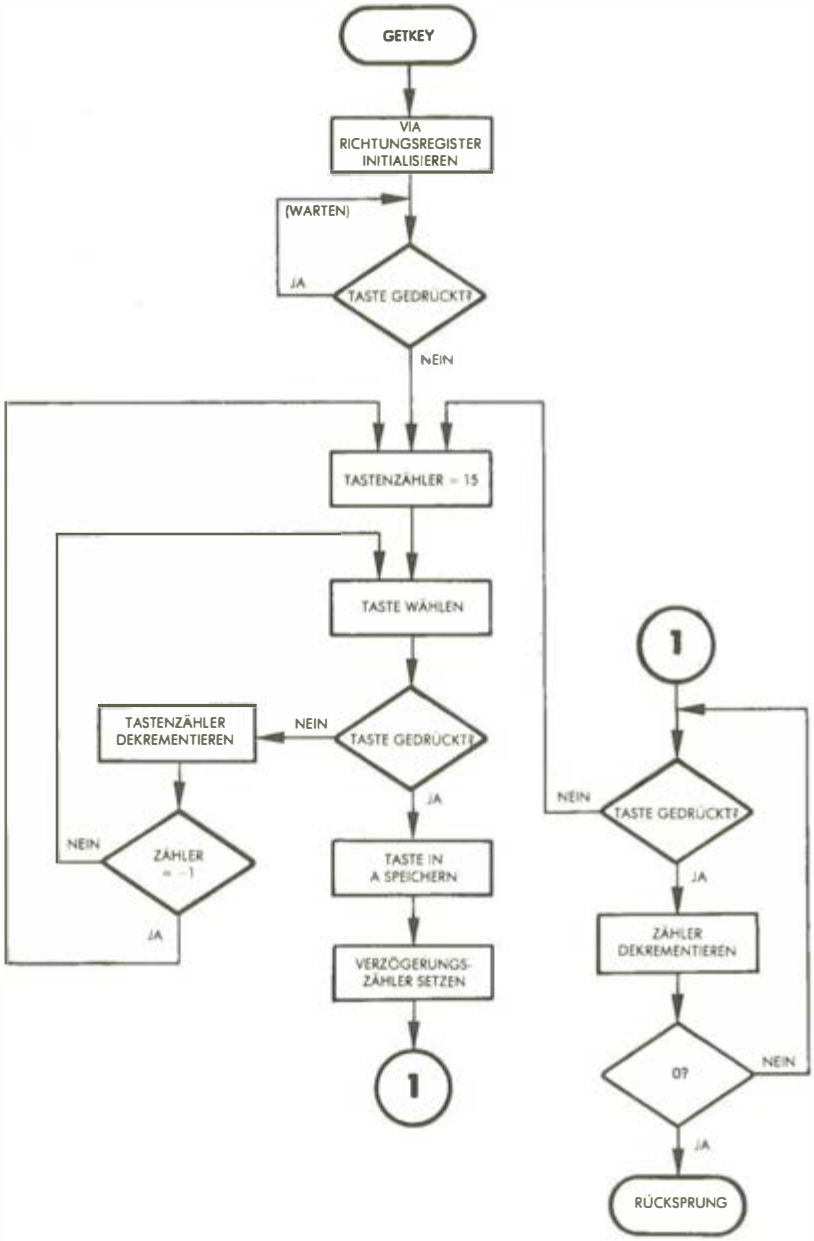



*Abb. 1.16: Flußdiagramm GETKEY*

```
                ;'GETKEY' TASTENEINGABE-ROUTINE
                ;LIEST UND ENTPRELLT TASTENEINGABE, KEHRT MIT TASTEN-NR.
                ;IM AKKUMULATOR ZURUECK.
                ;AUSFUEHRUNG: SENDET ZAHLEN 0 BIS F AN 74154 (4-BIS-16 DECODER),
                ;DER JEWEILS EINE SEITE DER TASTENSCHALTER NACHEINANDER ERDET.
                ;WENN TASTE GEDRUECKT IST, WIRD PA7 VON VIA 3 GEERDET, UND DER
                ;GERADE AN 74154 UEBERGEBENE WERT IST DIE TASTENNUMMER.
                ;ENTDECKT DAS PROGRAMM EINE GEDRUECKTE TASTE, WIRD FUER 50 MS
                ;WEITERER TASTENSCHLUSS ZWECKS PRELLUNTERDRUECKUNG GEPRUEFT.
                ;ANMERKUNG: IST KEINE TASTE GEDRUECKT, WARTET 'GETKEY'.
                ;

0100: A9 00             LDA #0
0102: 8D 03 AC          STA DDR3A       ;TASTEN-EINBLENDUNGSTOR AUF EINGABE
0105: A9 FF             LDA #$FF
0107: 8D 02 AC          STA DDR3B       ;TOR FUER TASTENNUMMER AUF AUSGABE
010A: 2C 01 AC START    BIT TOR3A       ;PRUEFEN, OB TASTE NOCH VOM LETZTEN MAL
                                        ;GEDRUECKT: TASTENEINBLENDUNG IM STATUS-
                                        ;BIT 'N'
010D: 10 FB             BPL START       ;WENN JA, AUF TASTENFREIGABE WARTEN
010F: A2 0F     RSTART  LDX #15         ;TASTENNUMMER-ZAEHLER AUF 15 SETZEN
0111: 8E 00 AC NXTKEY   STX TOR3B       ;TASTENNUMMER AN 74154 AUSGEBEN
0114: 2C 01 AC          BIT TOR3A       ;TASTE GEDRUECKT? EINBLENDUNG IN 'N'
0117: 10 05             BPL PRELLEN     ;WENN JA, ENTPRELLEN
0119: CA                DEX             ;TASTENZAEHLER DEKREMENTIEREN
011A: 10 F5             BPL NXTKEY      ;NEIN, NAECHSTE TASTE
011C: 30 F1             BMI RSTART      ;VON VORN
011E: BA        PRELLEN TXA             ;TASTENNUMMER NACH A RETTEN
011F: A0 12             LDY #$12        ;AUSSENSCHLEIFENZAEHLER FÜR 50 MS
                                        ;VERZOEGERUNG LADEN
0121: A2 FF     SCHL1   LDX #$FF        ;INNENSCHLEIFE
0123: 2C 01 AC SCHL2    BIT TOR3A       ;TASTE NOCH GEDRUECKT?
0126: 30 E7             BMI RSTART      ;WENN NICHT, UNGUELTIG: NEUANFANG
0128: CA                DEX
0129: D0 FB             BNE SCHL2       ;SCHLEIFENFORMEL 2115*5
012B: 88                DEY
012C: D0 F3             BNE SCHL1       ;AUSSENSCHLEIFE: INSGESAMT 50 MS
012E: 60                RTS             ;FERTIG: TASTENNUMMER IN A.


SYMBOLTABELLE:
 DDR3A      AC03        DDR3B      AC02        TOR3A      AC01
 TOR3B      AC00        START      010A        RSTART     010F
 NXTKEY     0111        PRELLEN    011E        SCHL1      0121
 SCHL2      0123
```

*Abb. 1.17: Programm GETKEY*

Der Tastenzähler steht ursprünglich auf dem Wert 15, und er wird so lange dekrementiert, bis ein Tastenschluß festgestellt wird. Der Zählwert steht im X-Register, was mit dem DEX-Befehl leicht dekrementiert werden kann.

```
RSTART      LDX #15
```

Dieser Wert (15) wird an den 74154 ausgegeben, was zur Auswahl der Leitung 17 führt, die mit Taste 15 (F) verbunden ist. Der obige BIT-Befehl testet den Zustand von Bit 7 des Tores 3A daraufhin, ob die Taste gedrückt worden ist.

```
NXTKEY      STX TOR3B
            BIT TOR3A
            BPL PRELLEN
```

Bei gedrückter Taste würde eine Verzweigung nach PRELLEN stattfinden, wo eine Verzögerung ein Prellen unterbindet. Bei nicht gedrückter Taste wird der Zähler dekrementiert und auf Unterlauf getestet: Solange der Zähler nicht negativ wird, erfolgt der Rücksprung zur Marke NXTKEY. Die Schleife wird durchlaufen, bis entweder eine gedrückte Taste erkannt oder der Zähler negativ wird. In diesem Fall springt die Routine zur Marke RSTART, wo der Prozeß erneut beginnt:

```
DEX
BPL NXTKEY
BMI RSTART
```

Zu beachten ist, daß im Fall von mehreren gleichzeitig gedrückten Tasten immer die jeweils höchste festgestellt wird. Wären etwa die Tasten F und 3 zusammen gedrückt, würde also nur F erkannt, 3 dagegen ignoriert. Man kann dieses Problem dadurch umgehen, daß man einen sogenannten Mehrfachtasten-Rollover-Schutz vorsieht, was als kleine Übung nahegelegt werden soll:

**Übung 1-1:** *Um das Mehrfachtasten-Rollover-Problem zu umgehen, ändern Sie die GETKEY-Routine so, daß alle 15 Tastenschlüsse gehandhabt werden können. Bei mehr als einer gedrückten Taste sollen Tastendrücke so lange ignoriert werden, bis nur mehr eine gedrückt ist.*

Ist ein Tastendruck identifiziert, wird die entsprechende Tastennummer im Akkumulator gespeichert. Es folgt die 50 ms lange Verzögerung für die Prellunterdrückung. Innerhalb dieser Schleife wird der Tastenschluß dauernd überwacht. Wird die Taste losgelassen, beginnt die Routine von vorn. Die Verzögerung selbst ist eine gewöhnliche, doppelt verschachtelte Schleife.

```
PRELLEN   TXA
          LDY #$12
SCHL1     LDX #$FF
SCHL2     BIT PORT3A
          BMI RSTART
          DEX
          BNE SCHL2
          DEY
          BNE SCHL1
```

**Übung 1-2:** *Der für den Zähler der äußeren Schleife gewählte Wert $12 (12 hexadezimal) ist möglicherweise nicht ganz genau. Berechnen Sie die exakte Verzögerungszeit der oben angeführten Befehlsfolge, indem Sie die im Anhang aufgeführten Befehlsausführungszeiten zugrundelegen.*

## ZUSAMMENFASSUNG

Um die Spielprogramme laufen lassen zu können, ist ein einfaches Spielbrett mit den wesentlichen Ein/Ausgabe-Einrichtungen erforderlich. Die benötigte Hardware- und Softwareschnittstelle ist in diesem Kapitel beschrieben worden. Fotos des aus dem Prototyp entwickelten Spielbrettes zeigen die Bilder 1.18 und 1.19.

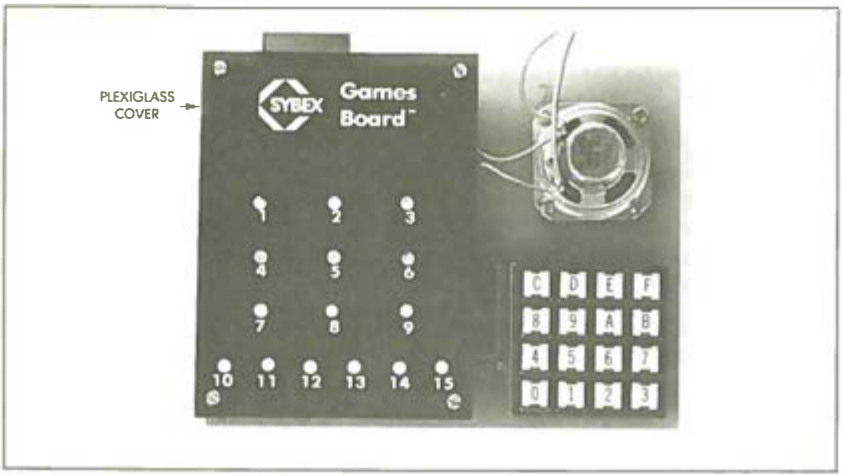



*Abb. 1.18: „Produktions"-Spielbrett*

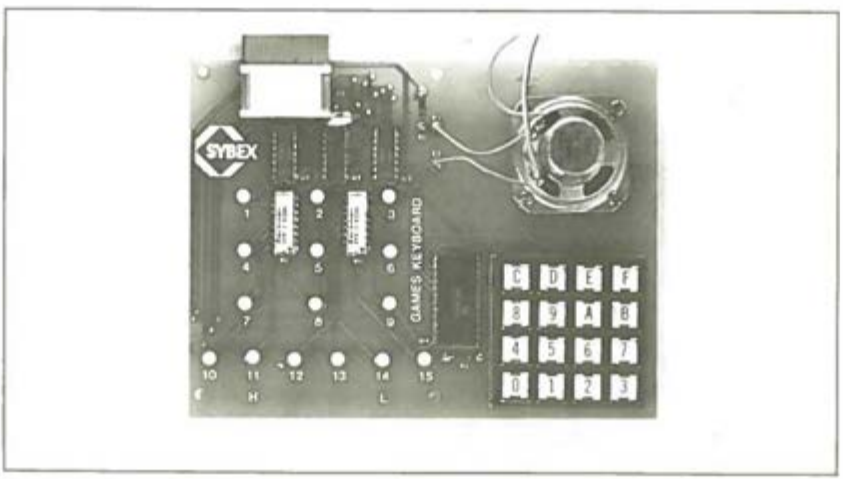



*Abb. 1.19: Entfernen der Abdeckplatte*

# 2

# Erzeugung von Rechteck-Wellen (Musikmacher)

## EINFÜHRUNG

Bei diesem Programm werden Sie lernen, wie der Mikrocomputer als Synthesizer zur Erzeugung von Rechteck-Wellen verwendet wird. Zur Tonerzeugung und zum Musikmachen wird ein tabellengesteuerter Algorithmus verwendet. Von der Technik der indizierten Adressierung wird systematisch Gebrauch gemacht.

## DIE REGELN

Das Spiel ermöglicht das Musizieren durch unmittelbaren Gebrauch der Tastatur, wobei das Programm die gespielten Töne gleichzeitig aufzeichnet, um sie auf Anfrage wiederzugeben. Die Tasten 0 bis C erzeugen die eigentlichen Töne (Bild 2.1). Die D-Taste wird zum Halten eines Tones benutzt, die E-Taste zum Abspielen der gespeicherten Tonfolge. Taste F schließlich löscht den Speicherinhalt, um ein neues Spiel beginnen zu können. Den normalen Spielablauf schildert der folgende Abschnitt.

| | | | |
|---|---|---|---|
| A (A) | B (H) | C (C) | D (REST) |
| 1 (A) | 2 (H) | 3 (C) | E (PBK) |
| 4 (D) | 5 (E) | 6 (F) | F (RST) |
| 7 (F#) | 8 (G) | 9 (G#) | 0 (G) |

| TASTEN-NUMMER | NOTE |
|---|---|
| 0 | G |
| 1 | A |
| 2 | H |
| 3 | C |
| 4 | D |
| 5 | E |
| 6 | F |
| 7 | F# |

| TASTEN-NUMMER | NOTE |
|---|---|
| 8 | G |
| 9 | G# |
| A | A |
| B | H |
| C | C |
| D | REST |
| E | PLAY BACK |
| F | RESTART |

*Abb. 2.1: Musikmachen auf der Tastatur*

**9. Sinfonie (Beethoven):**

5 — 5 — 6 — 8 — 8 — 6 — 5 — 4 — 3 — 3 — 4 — 5 — 5 — 4 — 4 — D — 5 —
5 — 6 — 8 — 8 — 6 — 5 — 4 — 3 — 3 — 4 — 5 — 4 — 3 — 3 — D — 4 — 4 —
5 — 3 — 4 — 6 — 5 — 3 — 4 — 6 — 5 — 4 — 3 — 4 — D

**Clementine:**

3 — 3 — 3 — D — 2 — D — 5 — 5 — 5 — D — 3 — D — 3 — 5 — 8 — D — D —
8 — 6 — 5 — 4 — D — D — D — 4 — 5 — 6 — D — 6 — D — 5 — 4 — 5 — D —
3 — D — 3 — 5 — 4 — D — D — 2 — 3 — 4 — 3

**Frere Jacques:**

3 — 4 — 5 — 3 — 3 — 4 — 5 — 3 — 5 — 6 — 8 — D — 5 — 6 — 8 — D — 8 —
A — 8 — 6 — 5 — D — 3 — D — 8 — A — 8 — 6 — 5 — D — 3 — D — 3 — D —
2 — D — 3 — D — D — D — 3 — D — 2 — D — 3

**Jingle Bells:**

5 — 5 — 5 — D — 5 — 5 — 5 — D — 5 — 8 — 3 — 4 — 5 — D — D — D — 6 —
6 — 6 — 6 — 6 — 5 — 5 — 5 — 8 — 8 — 6 — 4 — 3

**London Bridge:**

8 — A — 8 — 6 — 5 — 6 — 8 — D — 4 — 5 — 6 — D — 5 — 6 — 8 — D — 8 —
A — 8 — 6 — 5 — 6 — 8 — D — 4 — D — 8 — D — 5 — 3

**Mary Had a Little Lamb:**

5 — 4 — 3 — 4 — 5 — 5 — 5 — D — 4 — 4 — 4 — D — 5 — 8 — 8 — D — 5 —
4 — 3 — 4 — 5 — 5 — 5 — 5 — 4 — 4 — 5 — 4 — 3

**Row Row Row Your Boat:**

3 — D — 3 — D — 3 — 4 — 5 — D — 5 — 4 — 5 — 6 — 8 — D — D — D — C —
C — 8 — 8 — 5 — 5 — 3 — 3 — 8 — 6 — 5 — 4 — 3

**Stille Nacht:**

8 — D — D — A — 8 — D — 5 — D — D — D — 8 — D — D — A — 8 — D — 5 —
D — D — D — 3 — D — D — 3 — D — B — D — D — D — C — D — D — C —
D — 8 — D — D — C — D — 8 — 5 — 8 — D — 6 — D — 4 — D — 3

**Twinkle Twinkle Little Star:**

3 — 3 — 8 — 8 — A — A — 8 — D — 6 — 6 — 5 — 5 — 4 — 4 — 3 — D — 8 —
8 — 6 — 6 — 5 — 5 — 4 — D — 3 — 3 — 8 — 8 — A — A — 8 — D — 6 — 6 —
5 — 5 — 4 — 4 — 3

*Abb. 2.2: Einfache Melodien*

## EIN TYPISCHER SPIELVERLAUF

Um ein neues Spiel zu starten, drücken Sie die F-Taste. Es erklingt ein dreitöniger Triller, der anzeigt, daß der Speicher gelöscht ist. Mit den Tasten 0 bis D (die die Tonhöhen und die Tondauer festlegen) geben Sie nun Ihr Lied ein. Bis zu 254 Noten können gespielt und abgespeichert werden. Zu jeder Zeit kann die Wiedergabetaste E gedrückt werden, um die bis dahin gespeicherte Melodie wieder erklingen zu lassen, dies kann beliebig oft geschehen. Beispiele für einfache Melodien oder Tonsequenzen zeigt Bild 2.2.

## DIE VERBINDUNGEN

Bei diesem Spiel werden die Tastatur und der Lautsprecher benutzt. Dieser ist in Serie geschaltet mit einer der gepufferten Ausgabeleitungen von Tor B des VIA 3, und zwar über einen 110 Ohm-Widerstand. Benutzt werden PB4, PB5, PB6 oder PB7 des VIA 3, wie die Steuerung durch den Transistor-Puffer des SYM erfolgt. Zur Verbesserung der Tonqualität wird empfohlen, den Lautsprecher in einem Kasten oder dergleichen unterzubringen. Soll die Lautstärke vergrößert werden, kann auch ein kleinerer Widerstand (allerdings nicht unter 50 Ohm) verwendet werden.

## DER ALGORITHMUS

Ein Ton (eine Note) wird dadurch erzeugt, daß ein Rechtecksignal mit der gewünschten Frequenz zum Lautsprecher gesendet wird, d.h., diese Frequenz wird einfach an- und wieder abgeschaltet (Bild 2.3). Die Zeitdauer, für die der Lautsprecher ein- bzw. ausgeschaltet ist, nennt man Halbperiode. In diesem Programm ist ein Frequenzbereich von 195 bis 523 Hertz vorgesehen. Ist N die Frequenz, so ist die Schwingungsdauer T deren Kehrwert:

$$T = 1/N$$

Daraus folgt, daß die Halbperioden zwischen den Werten $1/(2 \times 195) = 0.002564$ und $1/(2 \times 523) = 0.000956$ Sekunden liegen. Eine klassische Warteschleife wird die jeweils gewünschte Frequenz erzeugen. Die eigentlichen Berechnungen der verschiedenen Programmparameter werden weiter unten vorgestellt.

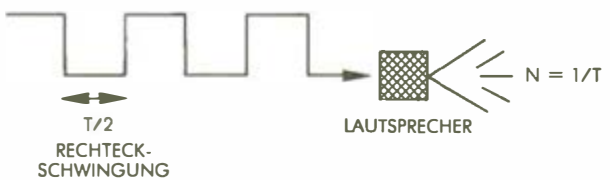



*Abb. 2.3: Tongenerierung*

## DAS PROGRAMM

Das Programm ist im Speicherbereich zwischen 200 und 2DD abgelegt, aufgezeichnete Tonfolgen ab Speicherstelle 300. Bis zu 254 Noten können in 127 Bytes gespeichert werden.

### Datenstrukturen

In diesem Programm werden drei Tabellen benutzt, sie sind in Bild 2.4 wiedergegeben. Die aufgezeichnete Melodie befindet sich in in einer Tabelle, die ab Adresse 300 gespeichert ist. Die Tonkonstanten, die die Frequenz erzeugen, bei der der Lautsprecher ein- und ausgeschaltet wird, sind in einer 16-Byte-Tabelle ab Adresse 2C4 gespeichert. Die Tondauern, also die Anzahl der

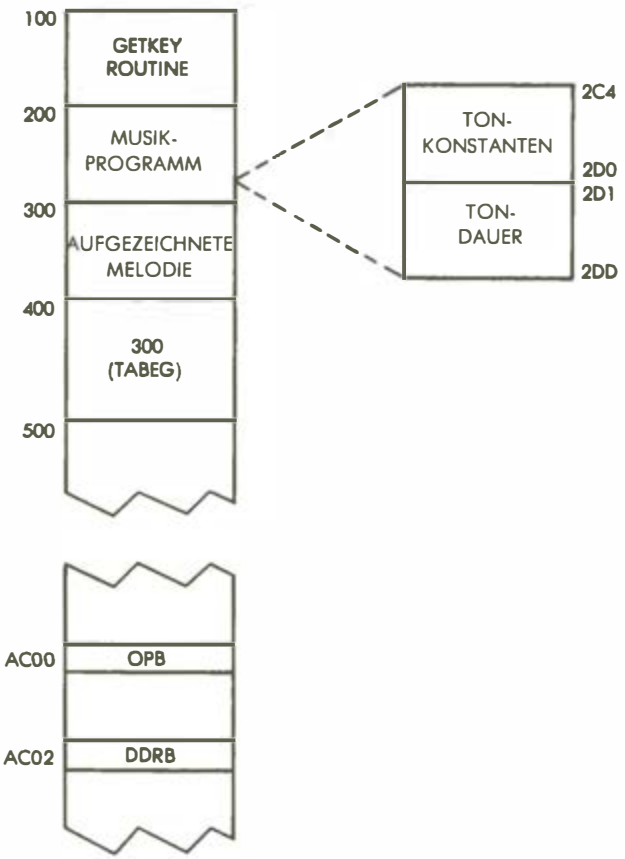



*Abb. 2.4: Speicheraufteilung*

Halbperioden, die einèn gleichbleibenden Ton von etwa 0.21 Sekunden erzeugen, liegen in einer 16-Byte-Tabelle ab Adresse 2D1. Innerhalb der Melodie-Tabelle tauchen zwei „Nibble"-(Halbbyte-)Zeiger auf: PILEN bei Eingaben und PTR bei Ausgaben. (Jedes 8-Bit-Byte dieser Tabelle enthält also zwei Noten.) Um den jeweils aktuellen Tabelleneinsprung aus dem Nibble-Zeiger zu erhalten, wird dieser einfach um eine Bitposition nach rechts verschoben. Der verbleibende Wert wird zu einem Byte-Zeiger, während das in die C-Flagge geschobene Bit die linke bzw. rechte Bytehälfte markiert. Die zwei mit CONSTANTS und TONDAUERN bezeichneten Tabellen sind einfache Referenztabellen zur Bestimmung der Halbfrequenz eines Tones und der Häufigkeit, mit der der Lautsprecher anzusprechen ist, sobald ein Ton identifiziert und eingeordnet worden ist. Beide Tabellen werden durch indirekte Adressierung mit dem X-Register angesprochen.

**Ein wenig Musiktheorie**

Ein kleiner Abriß musikalischer Begriffe sei an dieser Stelle eingeflochten. Die Frequenzen, die zur Erzeugung der gewünschten Töne benutzt werden, sind von der sogenannten temperierten Tonleiter abgeleitet, wobei die Frequenzen aufeinanderfolgender Halbtöne im Verhältnis

$$1:\sqrt[12]{2}$$

stehen.

Die Frequenzen für die mittlere C-Oktave sind in Bild 2.5 wiedergegeben. Um die entsprechenden Frequenzen der darüber bzw. darunter liegenden Oktaven zu berechnen, ist die Multiplikation mit bzw. Division durch 2 erforderlich.

| NOTE | FREQUENZ (HERTZ) |
|---|---|
| A | 220.00 |
| A# | 223.08 |
| H | 246.94 |
| C | 261.62 |
| C# | 277.18 |
| D | 293.66 |
| D# | 311.13 |
| E | 329.63 |
| F | 349.23 |
| F# | 369.99 |
| G | 391.99 |
| G# | 415.30 |

*Abb. 2.5: Frequenzen für die mittlere C-Oktave*

### Die Tonerzeugung

Um die Halbperiodenverzögerung zu erhalten, mit der das Rechtecksignal an den Lautsprecher gesendet wird, wird eine Programmschleife implementiert mit einem Grundzyklus von 10 $\mu$s. Das Programm benutzt einen „Schleifenindex" oder Schrittzähler, um die Anzahl von 10-$\mu$s-Zyklen zu zählen. Die Gesamtverzögerung errechnet sich zu

(Schleifenindex) × 10 − 1

Mikrosekunden.

Beim letzten Schleifenschritt (Schleifenindex = 0) wird der Verzweigungsbefehl nicht mehr ausgeführt. so daß eine Mikrosekunde (bei Annahme einer 1 MHz Uhr) von der Gesamtverzögerung subtrahiert werden muß. Nachfolgend aufgeführt wird die Tonerzeugungsroutine:

```
TON     STA FREQ
        LDA #$FF
        STA DDRB
        LDA #$00
        LDX DAUER
FL2     LDY FREQ
FL1     DEY
        CLC
        BCC +2
        BNE FL1
        EOR #$FF
        STA OPB
        DEX
        BNE FL2
        RTS
```

INNEN-SCHLEIFE

AUSSEN-SCHLEIFE

Zu beachten ist die „klassische" Schachtelschleifen-Struktur. Jedesmal, wenn die Außenschleife angesteuert wird, bewirkt sie eine zusätzliche Verzögerung von 13 Mikrosekunden: 14 Mikrosekunden für die zusätzlichen Befehle (LDY, EOR, STA, DEX und BNE) abzüglich eine Mikrosekunde für die Reaktion auf die erfolglose Innenschleifenverzweigung. Die Gesamtverzögerung der Außenschleife ist also

(Schleifenindex) × 10 + 13

Mikrosekunden. Es sei daran erinnert, daß ein Außenschleifendurchlauf nur eine Halbperiode für den Ton darstellt.

### Berechnung der Tonkonstanten

Wir wollen die Innenschleifenverzögerung IV und die zusätzliche Außenschlei-

fenverzögerung AV nennen. Wie im vorangegangenen Abschnitt gezeigt, ist die Halbperiode T/2 = (Schleifenindex) × 10 + 13, also

$$T/2 = (\text{Schleifenindex}) \times IV + AV$$

Die in der Tabelle gespeicherte Tonkonstante ist der Index-Wert, der vom Programm gebraucht wird. Aus der Gleichung läßt sich leicht ableiten, daß gilt

$$\text{Tonkonstante} = \text{Schleifenindex} = (T - 2 \times AV)/2 \times IV$$

Die Periode als Funktion der Frequenz ist T = 1/N, das ergibt in Mikrosekunden:

$$T = 10^6/N$$

Aus der oberen Gleichung wird also:

$$\text{Tonkonstante} = (10^6/N - 2 \times AV)/2 \times IV$$

Als Beispiel wollen wir die Tonkonstante berechnen, die zur Frequenz des mittleren C korrespondiert, sie ist in Bild 2.5 abgebildet und beträgt 261.62 Hertz. Als AV-Verzögerung erhielten wir oben 13, als IV-Wert 10 Mikrosekunden. Die Tonkonstantengleichung lautet dann:

$$\begin{aligned}\text{Tonkonstante} &= (10^6/N - 2 \times 13)/2 \times 10 \\ &= \frac{1000000/261.62-26}{20} \\ &= 190 \text{ (hexadezimal BE)}\end{aligned}$$

Man kann sich überzeugen, daß diese Zahl mit der vierten Tabelleneintragung bei Adresse NOTAB (in Bild 2.9 bei Adresse 02C4) übereinstimmt. In Bild 2.6 sind alle Tonkonstanten aufgeführt.

| NOTE | | NOTE | KONSTANTE | NOTE | KONSTANTE |
|---|---|---|---|---|---|
| UNTERHALB MITTLERER C-OKTAVE G | FE | MITTLERE C-OKTAVE C | BE | OBERHALB MITTLERER C-OKTAVE C | 5E |
| A | E2 | D | A9 | | |
| H | C9 | E | 96 | | |
| | | F | 8E | | |
| | | F# | 86 | | |
| | | G | 7E | | |
| | | G# | 77 | | |
| | | A | 70 | | |
| | | H | 64 | | |

*Abb. 2.6: Frequenzkonstanten*

**Übung 2-1:** *Berechnen Sie anhand der Tabelle in Bild 2.6 die korrespondierenden Frequenzen, und prüfen Sie, ob die Konstanten korrekt gewählt wurden.*

**Berechnung der Tondauern**

Die Tabelle DAUERTAB enthält die Tondauern als Zahlen, die der Anzahl von Halbzyklen für jede Note entsprechen. Diese Tondauern wurden so berechnet, daß eine Einheitsnote etwa 0.2175 Sekunden lang erklingt. Ist D die Dauer und T die Periode, so gilt die Gleichung

$$D \times T = 0.2175$$

wobei D in Perioden ausgedrückt ist. Da in der Praxis Halbperioden verwendet werden, lautet die Anzahl D' von Halbperioden:

$$D' = 2D = 2 \times 0.2175 \times N$$

Für das Beispiel des mittleren C ergibt das:

$$D = 2 \times 0.2175 \times 261.62 = 113.8 \approx 114$$

114 dezimal entspricht 72 hexadezimal.

**Übung 2-2:** *Berechnen Sie die Tondauern, indem Sie die obige Gleichung und die (zu erweiternde) Frequenztabelle in Bild 2.5 verwenden. Überprüfen Sie die Ergebnisse anhand der DAUERTAB-Tabelle ab Adresse 2D1 (Bild 2.9).*

**Implementierung des Programms**

Die Programmstruktur besteht aus zwei logischen Teilen. Bild 2.7 zeigt das zugehörige Flußdiagramm. Der erste Programmteil ist für das Erfassen der Noten verantwortlich und beginnt bei der Marke NUMKEY. (Das Programm selbst erscheint in Bild 2.9.) Der zweite Programmteil beginnt bei der Marke PLAYEM, er spielt die gespeicherten Noten. Beide Programmteile benutzen das Unterprogramm PLAYNOTE, das die Ton- und Tondauerkonstanten ermittelt und die Töne spielt. Diese Routine, deren Flußdiagramm in Bild 2.8 dargestellt ist, beginnt bei der Marke PLAYIT.

Die Hauptroutinen des Notenempfang-Programms heißen NXKEY, NUMKEY und BEEP3, die des Notenspiel-Programms heißen PLAYEM und DELAY. Allgemeine Dienstprogramme sind TON und PLAYIT.

Wir wollen uns diese Routinen nun genauer ansehen. Das Programm ist ab Speicheradresse 200 abgelegt. Es sei daran erinnert, daß dieses Programm, wie auch die meisten anderen in diesem Buch, die Verfügbarkeit der GETKEY-Routine aus Kapitel 1 voraussetzt.

Die NXKEY-Routine erfragt ohne viel Umschweife die nächste gedrückte Taste durch Aufruf des GETKEY-Unterprogramms:

```
START    LDA #0
         STA PILEN      Listenlänge initialisieren
         CLC
NXKEY    JSR GETKEY
```

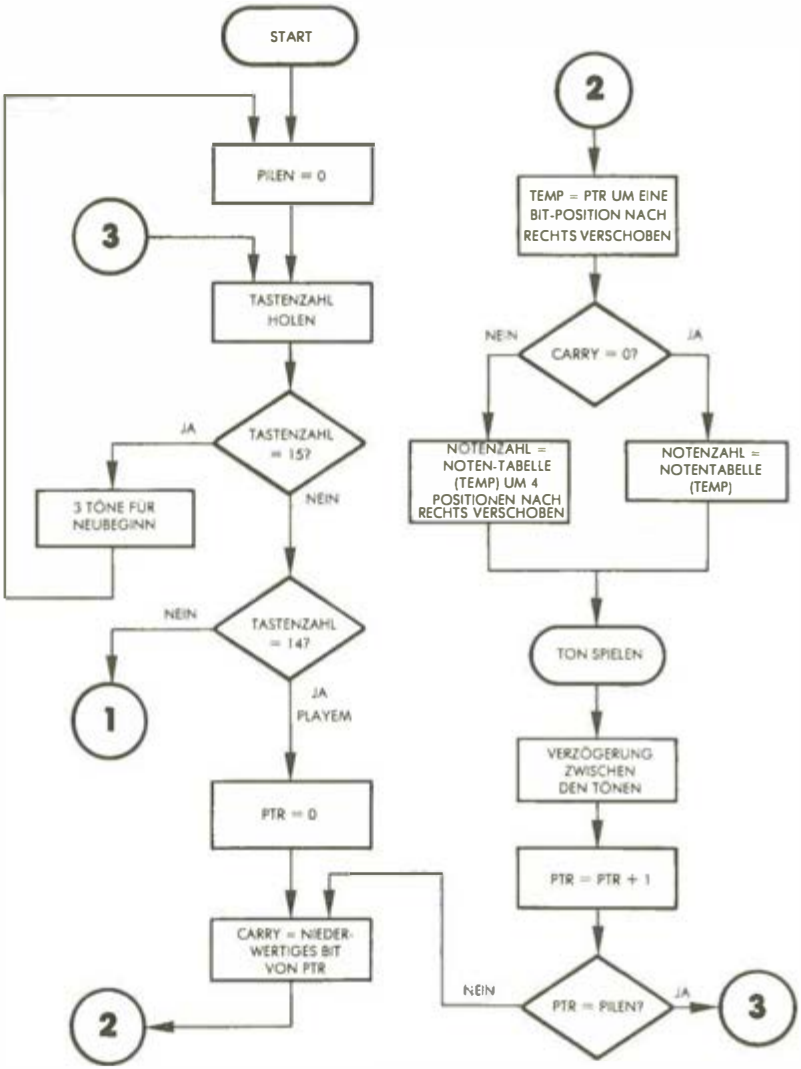



*Abb. 2.7: Flußdiagramm MUSIKMACHER*

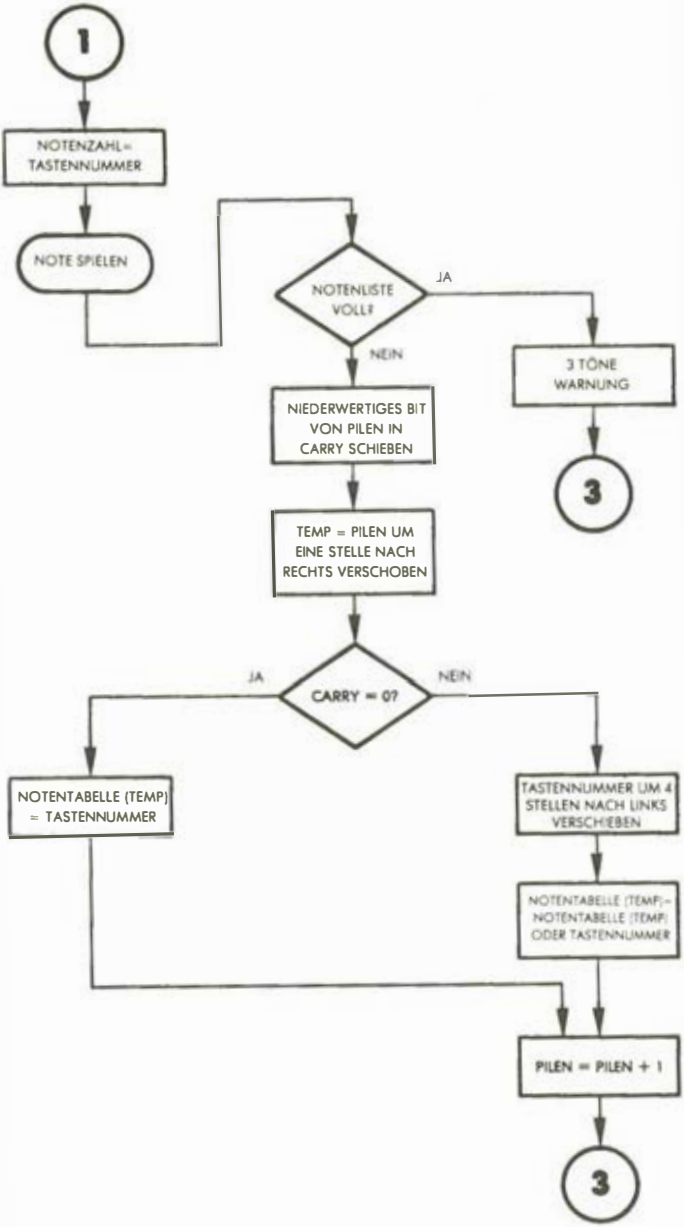



*Abb. 2.7: Flußdiagramm MUSIKMACHER (Fortsetzung)*

Der gelesene Wert wird nun mit den Konstanten 15 und 14 verglichen, um zu sehen, ob Sonderaktionen erforderlich sind. Ist das nicht der Fall, wird die Konstante mit Hilfe der NUMKEY-Routine in der Notenliste abgespeichert.

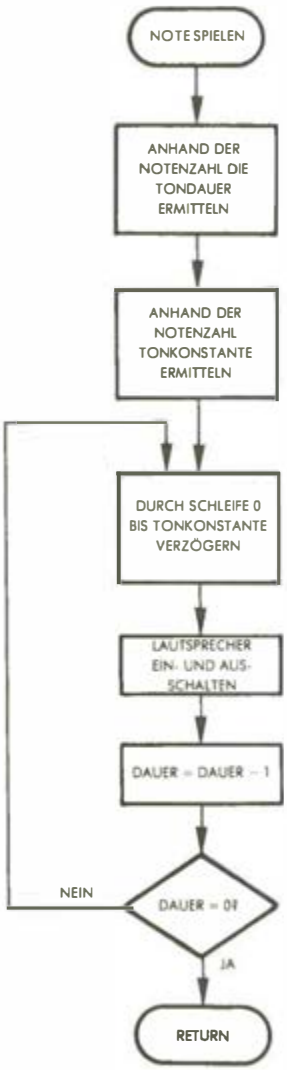



*Abb. 2.8: Flußdiagramm PLAYIT*

```
                ;   PROGRAMM 'MUSIKMACHER'
                ;   BENUTZT WERDEN 16-ER TASTATUR UND GEPUFFERTEN LAUTSPRECHER
                ;DAS PROGRAMM SPIELT GESPEICHERTE MUSIKNOTEN.
                ;ES GIBT 2 BEDIENUNGSMODI: EINGEBEN UND ABSPIELEN.
                ;VORGEGEBEN IST DER EINGABE-MODUS, UND ALLE NICHT-KOMMANDO-TASTEN
                ;(0-D) WERDEN BEIM DRUECKEN FUR DIE WIEDERGABE GESPEICHERT. IM
                ;FALL EINES UEBERLAUFS ERKLINGT EIN DREITOENIGES WARNSIGNAL.
                ;DASSELBE TRILLERSIGNAL KUENDIGT AUCH DAS ERNEUTE STARTEN DES
                ;PROGRAMMS AN.
                ;
                GETKEY = $100
                PILEN  = $00            ;LAENGE DER NOTENLISTE
                TEMP   = $01            ;ZWISCHENSPEICHER
                PTR    = $02            ;AKTUELLE LISTENPOSITION
                FREQ   = $03            ;ZWISCHENSPEICHER FUER FREQUENZ
                DAUER  = $04            ;ZWISCHENSPEICHER FUER TONDAUER
                TABEG  = $300           ;MELODIE-SPEICHERTABELL
                OPB    = $AC00          ;VIA AUSGANG TOR B
                DDRB   = $AC02          ;VIA TOR B DATENRICHTUNGSREGISTER
                *      = $200           ;ANFANG
                ;
                ;BEFEHLSINTERPRETATION
                ;   $F = ZEIGER ZURUECKSETZEN, NEUBEGINN
                ;   $E = GESPEICHERTE MELODIE ABSPIELEN
                ;   ALLE ANDEREN TASTEN WERDEN FUER DAS ABSPIELEN GESPEICHERT.
                ;
0200: A9 00     START   LDA #0          ;NOTENLISTENLAENGE NULLEN
0202: 85 00             STA PILEN
0204: 18                CLC             ;NIBBLE-ZEIGER NULLEN
0205: 20 00 01 NXKEY    JSR GETKEY
0208: C9 0F             CMP #15         ;TASTENNUMMER = 15 ?
020A: D0 05             BNE NXTST       ;NEIN, NAECHSTER VERSUCH
020C: 20 B7 02          JSR BEEP3       ;SIGNAL: GELOESCHT
020F: 90 EF             BCC START       ;ZEIGER ZURUECKSETZEN, NEUBEGINN
0211: C9 0E     NXTST   CMP #14         ;TASTENNUMMER = 14 ?
0213: D0 06             BNE NUMKEY      ;NEIN, TASTE IST NOTENNUMMER
0215: 20 48 02          JSR PLAYEM      ;NOTEN SPIELEN
0218: 18                CLC
0219: 90 EA             BCC NXKEY       ;NAECHSTE TASTE HOLEN
                ;
                ;ROUTINE ZUM LADEN DER NOTENLISTE
                ;
021B: 85 01     NUMKEY  STA TEMP        ;TASTE SPEICHERN, A FREIMACHEN
021D: 20 70 02          JSR PLAYIT      ;NOTE SPIELEN
0220: A5 00             LDA PILEN       ;LISTENLAENGE HOLEN
0222: C9 FF             CMP #$FF        ;UEBERLAUF ?
0224: D0 05             BNE OK          ;NEIN, NOTE AN LISTE ANFUEGEN
0226: 20 B7 02          JSR BEEP3       ;JA, WARNSIGNAL SPIELEN
0229: 90 DA             BCC NXKEY       ;ZUM EINGABEMODUS
022B: 4A        OK      LSR A           ;LOW BIT WIRD NIBBLE-ZEIGER
022C: A8                TAY             ;NIBBLE-ZEIGER WIRD BYTE-INDEX
022D: A5 01             LDA TEMP        ;TASTENNUMMER ZURUECKHOLEN
022F: B0 09             BCS FINBYT      ;WENN BYTE SCHON NIBBLE HAT, AUFFUELLEN
                                        ;UND ABSPEICHERN
0231: 29 0F             AND #%00001111  ;1. NIBBLE, HIGH NIBBLE MASKIEREN
0233: 99 00 03          STA TABEG,Y     ;UNVOLLSTAENDIGES HALBBYTE SPEICHERN
0236: E6 00             INC PILEN       ;ZEIGER AUF NAECHSTEN NIBBLE
0238: 90 CB             BCC NXKEY       ;NAECHSTE TASTE HOLEN
023A: 0A        FINBYT  ASL A           ;NIBBLE HOEHERWERTIG VERSCHIEBEN
023B: 0A                ASL A
023C: 0A                ASL A
023D: 0A                ASL A
023E: 19 00 03          ORA TABEG,Y     ;2 NIBBLES ZU BYTE VERBINDEN
0241: 99 00 03          STA TABEG,Y     ;UND ABSPEICHERN
0244: E6 00             INC PILEN       ;ZEIGER AUF NAECHST. NIBBLE,NAECHST. BYTE
0246: 90 BD             BCC NXKEY       ;ZURUECK
                ;
                ;ROUTINE ZUM SPIELEN DER NOTEN
                ;
0248: A2 00     PLAYEM  LDX #0          ;ZEIGER INITIALISIEREN
024A: 86 02             STX PTR
024C: A5 02             LDA PTR         ;AKKU MIT AKTUELLEM PTR-WERT LADEN
```

*Abb. 2.9: Programm MUSIKMACHER*

```
024E: 4A        SCHLEIFE LSR A           ;NIBBLE-ZEIGER INS CARRY-BIT
024F: AA                 TAX             ;NIBBLE-ZEIGER WIRD BYTE-ZEIGER
0250: BD 00 03           LDA TABEG,X     ;ZU SPIELENDE NOTE LADEN
0253: B0 04              BCS ENDBYT      ;LOW NIBBLE BENUTZT: HIGH NIBBLE NEHMEN
0255: 29 0F              AND #%00001111  ;HIGH BITS AUSBLENDEN
0257: 90 06              BCC FERTIG      ;NOTE SPIELEN
0259: 29 F0     ENDBYT   AND #%11110000  ;LOW NIBBLE ELIMINIEREN
025B: 4A                 LSR A           ;IN LOW-POSITION SCHIEBEN
025C: 4A                 LSR A
025D: 4A                 LSR A
025E: 4A                 LSR A
025F: 20 70 02 FERTIG    JSR PLAYIT      ;KONSTANTEN BERECHNEN UND SPIELEN
0262: A2 20              LDX #$20        ;VERZOEGERUNG ZWISCHEN TOENEN
0264: 20 9C 02           JSR DELAY
0267: E6 02              INC PTR         ;EIN NIBBLE BENUTZT
0269: A5 02              LDA PTR
026B: C5 00              CMP PILEN       ;LISTENENDE ?
026D: 90 DF              BCC SCHLEIFE    ;NEIN, NAECHSTE NOTE HOLEN
026F: 60                 RTS             ;FERTIG
                ;
                ;TABELLENSUCHROUTINE, HALTETON AUSSORTIEREN
                ;
0270: C9 0D     PLAYIT   CMP #13         ;HALTETON ?
0272: D0 06              BNE KLANG       ;NEIN
0274: A2 54              LDX #$54        ;VERZOEGERUNG=TONDAUER=0.21 SEK.
0276: 20 9C 02           JSR DELAY
0279: 60                 RTS
027A: AA        KLANG    TAX             ;TASTENNUMMER=INDEX
027B: BD D1 02           LDA DURTAB,X    ;...FUER TONDAUER
027E: 85 04              STA DAUER       ;ABSPEICHERN
0280: BD C4 02           LDA NOTAB,X     ;NOTENWERT LADEN
0283: 20 AB 02           JSR TON
0286: 60                 RTS
                ;
                ;ROUTINE FUER 3-TOENE-SIGNAL
                ;
0287: A9 FF     BEEP3    LDA #$FF        ;PIEPTONDAUER
0289: 85 04              STA DAUER
028B: A9 4B              LDA #$4B        ;KODE FUER E2
028D: 20 AB 02           JSR TON         ;1. TON
0290: A9 3B              LDA #$3B        ;KODE FUER D2
0292: 20 AB 02           JSR TON
0295: A9 4B              LDA #$4B
0297: 20 AB 02           JSR TON
029A: 1B                 CLC
0298: 60                 RTS
                ;
                ;VARIABLEN LAENGEN VERZOEGERUNG
                ;
029C: A0 FF     DELAY    LDY #$FF
029E: EA        DLY      NOP
029F: D0 00              BNE +2
02A1: BB                 DEY
02A2: D0 FA              BNE DLY         ;SCHLEIFE 10
02A4: CA                 DEX
02A5: D0 F5              BNE DELAY       ;SCHLEIFENZEIT = 2556*(X)
02A7: 60                 RTS
                ;
                ;TONERZEUGUNGSROUTINE, HALBZYKLENZAHL IST IN 'DAUER', HALBZYKLUS-
                ;ZEIT IN A, SCHLEIFENZEIT=20*(A)+26, WEIL ZWEI AUSSENSCHLEIFEN-
                ;DURCHLAEUFE EINEN ZYKLUS FUER DEN TON ERGEBEN.
                ;
02A8: 85 03     TON      STA FREQ        ;ZYKLENZAHL ZWISCHENSPEICHERN
02AA: A9 FF              LDA #$FF        ;DATENRICHTUNGSREGISTER SETZEN
02AC: 8D 02 AC           STA DDRB
02AF: A9 00              LDA #0          ;A ZUM TOR, BEGINN MIT HIGH
0281: A6 04              LDX DAUER
0283: A4 03     FL2      LDY FREQ
0285: 8B        FL1      DEY
0286: 1B                 CLC
0287: 90 00              BCC +2
0289: D0 FA              BNE FL1         ;INNENSCHLEIFE
```

*Abb. 2.9: Programm MUSIKMACHER (Fortsetzung)*

```
02BB: 49 FF             EOR #$FF        ;I/O-TOR KOMPLEMENTIEREN
02BD: 8D 00 AC          STA OPB         ;...UND SETZEN
02C0: CA                DEX
02C1: D0 F0             BNE FL2         ;AUSSENSCHLEIFE
02C3: 60                RTS
                ;TONKONSTANTENTABELLE ENTHAELT:
                ;UNTER MITTEL-C-OKTAVE: G, A, H
                ;MITTEL-C-OKTAVE: C,D,E,F,F#,G,G#,A,H
                ;UEBER MITTEL-C-OKTAVE: C
                ;
02C4: FE        NOTAB   .BYT $FE,$E2,$C9,$BE,$A9,$96,$8E
02C5: E2
02C6: C9
02C7: BE
02C8: A9
02C9: 96
02CA: 8E
02CB: B6                .BYT $B6,$7E,$77,$70,$64,$5E
02CC: 7E
02CD: 77
02CE: 70
02CF: 64
02D0: 5E
                ;TONDAUERTABELLE IN HALBZYKLEN,FUER TONDAUER ETWA 0.21 SEK
                ;
02D1: 55        DURTAB  .BYT $55,$60,$6B,$72,$B0,$BF,$94
02D2: 60
02D3: 6B
02D4: 72
02D5: B0
02D6: BF
02D7: 94
02D8: A1                .BYT $A1,$AA,$B5,$BF,$D7,$E4
02D9: AA
02DA: B5
02DB: BF
02DC: D7
02DD: E4

SYMBOLTABELLE:
 GETKEY         0100        PILEN       0000        TEMP        0001
 PTR            0002        FREQ        0003        DAUER       0004
 TABEG          0300        OPB         AC00        DDRB        AC02
 START          0200        NXKEY       0205        NXTST       0211
 NUMKEY         0218        OK          0228        FINBYT      023A
 PLAYEM         0248        SCHLEIFE    024E        ENDBYT      0259
 FERTIG         025F        PLAYIT      0270        KLANG       027A
 BEEP3          0287        DELAY       029C        DLY         029E
 TON            02AB        FL2         0283        FLI         0285
 NOTAB          02C4        DURTAB      02D1
```

*Abb. 2.9: Programm MUSIKMACHER (Fortsetzung)*

```
                CMP #15
                BNE NXTST
                JSR BEEP3
                BCC START
NXTST           CMP #14
                BNE NUMKEY
                JSR PLAYEM
                CLC
                BCC NXKEY
```

**Übung 2-3:** *Warum werden die beiden letzten Befehle statt eines unbedingten Sprungbefehls verwendet? Was sind die Vor- und Nachteile dieser Technik?*

Jedesmal, wenn Taste 15 gedrückt wurde, wird eine spezielle 3-Töne-Routine BEEP3 aufgerufen, die bei Adresse 0287 beginnt. Sie spielt drei rasch aufeinanderfolgende Töne, um anzuzeigen, daß die bisher gespeicherten Noten gelöscht worden sind. Dieses Löschen geschieht durch „Nullen" der Listenlänge PILEN. Nachfolgend diese Routine:

```
BEEP3   LDA #$FF        Tondauerkonstante
        STA DAUER
        LDA #$4B        Code für E2
        JSR TON         1. Note
        LDA #$38        Code für D2
        JSR TON         2. Note
        LDA #$4B        Code für E2
        JSR TON         3. Note
        CLC
        RTS
```

Die NUMKEY-Routine legt den Code der eingegebenen Note im Speicher ab. Dabei wird wie bei einem Fernschreiberprogramm das gedrückte Zeichen durch ein akustisches Signal auch hörbar gemacht. Mit anderen Worten: Jedesmal, wenn eine Taste gedrückt wird, läßt das Programm den dazugehörigen Ton erklingen. Dies wird durch die beiden nächsten Befehle bewirkt:

```
NUMKEY  STA TEMP
        JSR PLAYIT
```

Jetzt wird die Listenlänge auf Überlauf geprüft. Hat ein Überlauf stattgefunden, wird der Spieler durch die 3-Töne-Folge von BEEP3 darauf aufmerksam gemacht:

```
        LDA PILEN       Listenlänge holen
        CMP #$FF        Überlauf?
        BNE OK          Nein: Note an Liste anfügen
        JSR BEEP3       Ja: Spieler warnen
        BCC NXKEY       nächste Taste einlesen
```

Andernfalls wird der Nibble (4 Bits), der zu der Identifikationsnummer des neuen Tons gehört, in die Liste eingefügt:

```
OK      LSR A           niedrigstes Bit in den
                        Nibble-Zeiger
        TAY             Byte-Index übertragen
        LDA TEMP        Tastennummer zurückholen
```

Zu beachten ist, wie der Nibble-Zeiger durch 2 dividiert und zum Byte-Index umfunktioniert wird, woraufhin er ins Y-Register wandert. Dieses wird später zur Indizierung beim Zugriff auf die richtige Tabellenposition des entsprechenden Bytes benutzt (STA TABEG,Y).

Je nach dem Wert, der ins Carry-Bit gerutscht ist, wird der Nibble im oberen oder im unteren Bereich des Tabelleneintrags gespeichert. Immer wenn der Nibble in der höherwertigen Hälfte abgelegt werden soll, ist ein viermaliges Verschieben nach links erforderlich, was vier derartige Befehle nötig macht:

```
        BCS FINBYT      Byte auf Nibble prüfen
        AND #%00001111  höherwertigen Nibble maskieren
        STA TABEG,Y     speichern
        INC PILEN       nächster Nibble
        BCC NXKEY
FINBYT  ASL A
        ASL A
        ASL A
        ASL A
```

Jetzt kann er bei der entsprechenden Tabellenadresse abgelegt werden:

```
ORA TABEG,Y
STA TABEG,Y
```

Nach der Erhöhung des Zeigers wird die nächste Taste untersucht:

```
INC PILEN
BCC NXKEY
```

Machen wir uns diese Technik an einem Beispiel klar. Es sei

PILEN = 9 (Listenlänge)
TEMP = 6 (gedrückte Taste)

Die Befehle wirken sich dann folgendermaßen aus:

```
OK      LSR A       A ist 4, und C ist 1
        TAY         Y ist 4
        LDA TEMP    A ist 6
        BCS FINBYT  C ist 1, und die Verzweigung findet
                    statt
```

Wie die Liste aussieht, zeigt Bild 2.10:

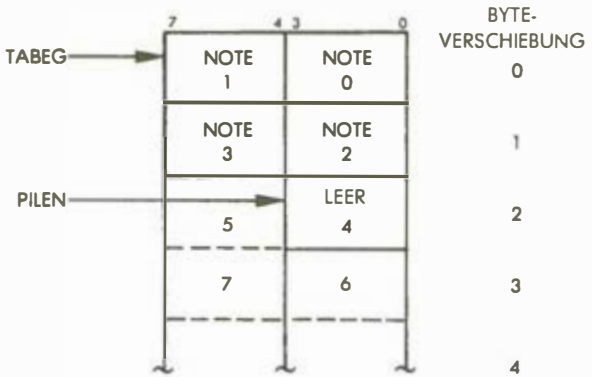



*Abb. 2.10: Abspeichern einer Note*

Die 6 wandert in die höherwertige Hälfte von A:

```
FINBYT   ASL A
         ASL A
         ASL A
         ASL A          A = 60 (hexadezimal)
```

A wandert in die Tabelle:

```
         ORA TABEG, Y   A = 6X (mit X = Wert des vorher
                        vorhandenen Nibble an dieser Ta-
                        bellenposition)
         STA TABEG, Y   alten und neuen Nibble abspei-
                        chern
```

**Die Unterprogramme**

***Unterprogramm PLAYEM***

Auch die PLAYEM-Routine ist äußerst geradlinig. Als laufender Nibble-Zeiger für die Notentabelle wird die Speicherstelle PTR benutzt. Wie gehabt, wird der Inhalt des Zeigers einmal nach rechts geschoben, woraufhin er als Byte-Zeiger fungiert. Durch indizierte Adressierung wird der korrespondierende Tabelleneintrag geladen:

```
PLAYEM      LDX #0
            STX PTR          PTR = 0
            LDA PTR
SCHLEIFE    LSR A
            TAX
            LDA TABEG,X
            BCS ENDBYT
            AND #%00001111
            BCC FERTIG
ENDBYT      AND #%11110000
            LSR A
            LSR A
            LSR A
            LSR A
```

Je nach dem Wert, der im Carry-Bit steht, wird wieder der höherwertige bzw. der niederwertige Nibble ausgesondert und im Akkumulator nach links justiert. Die weiter unten aufgeführte PLAYIT-Routine besorgt die passenden Konstanten und spielt den Ton:

```
FERTIG      JSR PLAYIT       Note spielen
```

Zwischen zwei aufeinanderfolgenden Noten wird nun eine Verzögerung eingeschaltet, der laufende Zeiger wird erhöht, es wird auf Listenende geprüft, dann geht es zurück in die Schleife:

```
            LDX #$20         Verzögerungskonstante
            JSR DELAY        Verzögerung zwischen den Noten
            INC PTR          1 Nibble verwendet
            CMP PILEN        auf Listenende prüfen
            BCC SCHLEIFE     nein: nächste Note
            RTS              fertig
```

### *Unterprogramm PLAYIT*

Die PLAYIT-Routine spielt eine Note oder hält einen Ton, je nach dem Wert des im Akkumulator mitgebrachten Nibble. Dieses Unterprogramm, das im Flußdiagramm mit PLAYNOTE bezeichnet ist, holt sich lediglich aus der DURTAB-Tabelle die entsprechende Tondauer und legt sie bei Adresse DAUER (Speicherstelle 4) ab. Danach lädt es den zugehörigen Halbperiodenwert aus der Tabellenadresse NOTAB ins A-Register (mittels indirekter Adressierung) und ruft das Unterprogramm TON auf, um den Ton zu spielen:

```
PLAYIT      CMP #13          auf Dauerton prüfen
            BNE KLANG        nein
```

```
        LDX #$54        Verzögerung = 0.21 Sek.
                        (Tondauer)
        JSR DELAY       Verzögerung
        RTS
KLANG   TAX             Tastennummer als Index
        LDA DURTAB,X    Tondauer holen
        STA DAUER
        LDA NOTAB,X
        JSR TON
        RTS
```

***Unterprogramm TON***

Diese Routine implementiert die weiter oben beschriebene Prozedur der richtigen Wellen-Erzeugung, um den Lautsprecher auf der gewünschten Frequenz „swingen" zu lassen. Zur Verwendung kommt eine doppelt geschachtelte Verzögerungsschleife, die den Lautsprecher an- und abschaltet, indem das Ausgangstor nach jeder spezifizierten Verzögerung komplementiert wird:

```
TON     STA FREQ
```

Beim Einsprung enthält A die Halbzyklus-Zeit, die in FREQ gespeichert wird. Das Schleifen-Timing bestimmt die ausgegebene Wellenlänge zu:

$(20 \times A + 26)\ \mu s$

Der Ausgang ist Tor B:

```
        LDA #$FF
        LDX DAUER
```

Der Zähler für die Innenschleife Y wird dann auf FREQ gesetzt, die Frequenz-Konstante:

```
FL2     LDY FREQ
```

Und die Verzögerung der Innenschleife entsteht wie bekannt:

```
FL1     DEY
        CLC
        BCC +2
        BNE FL1         10 Mikrosekunden Innenschleife
```

Durch Komplementieren wird das Ausgangstor geschaltet:

```
        EOR #$FF
        STA OPB
```

Schließlich wird die Außenschleife komplettiert:

```
DEX
BNE FL2
RTS
```

Das DELAY-Unterprogramm findet sich in Bild 2.9 ab Speicherstelle 29C und bleibt Ihnen zum Üben überlassen.

## ZUSAMMENFASSUNG

Dieses Programm benutzt einen einfachen Algorithmus zum Merken und Spielen von Melodien. Sämtliche Daten und Konstanten werden in Tabellen gespeichert. Gezielte Zeitverzögerungen werden durch geschachtelte Schleifen bewirkt. Zum Speichern und Aufgreifen von Daten werden indizierte Adressierungstechniken verwendet. Töne werden durch Rechteck-Schwingungen generiert.

### Übungen

**Übung 2-4:** *Ändern Sie die Tonkonstanten, um einen anderen Tonumfang zu erzielen.*

**Übung 2-5:** *Speichern Sie eine Melodie im voraus ab, und rufen Sie sie mit der 0-Taste ab.*

**Übung 2-6:** *Schreiben Sie das Programm so um, daß es die Ton- und Tondauerkonstanten sofort bei der Eingabe abspeichert, um sie nicht erst beim Abspielen der Melodie aufsuchen zu müssen. Was sind die Nachteile dieser Methode?*

# 3

# Erzeugung von Pseudo-Zufallszahlen (Übersetzen)

## EINFÜHRUNG

In diesem Programm wird ein Pseudo-Zufallszahlen-Generator vorgestellt, es erscheinen Figuren auf der Anzeige, Zeitabläufe werden gemessen, und es werden Verzögerungen generiert. Ehe wir danach zu noch komplexeren Entwürfen weitergehen, wird ihr Wissen über grundlegende Ein/Ausgabe-Techniken überprüft.

## DIE SPIELREGELN

Dieses Spiel ist als Wettbewerb zwischen zwei Spielern konzipiert, die beide versuchen, die vom Computer codierten Zahlen zu entschlüsseln. Die Spieler sind abwechselnd an der Reihe, eine vom Programm ausgegebene vierstellige Binärzahl durch Drücken der zugehörigen hexadezimalen Taste zu identifizieren. Das Programm führt – bis zu einem Höchstwert von etwa 17 Sekunden – Buch über die Gesamtratezeit jedes Spielers. Hat jeder Spieler seine Zahl richtig decodiert, so entscheidet die bessere Zeit darüber, wer die Runde gewonnen hat. Derjenige, der zuerst zehn Runden gewonnen hat, ist Gesamtsieger.

Wer gerade an der Reihe ist, signalisiert das Programm mit einem Pfeil auf der Anzeige, der nach rechts oder links zeigt. Der Spieler rechts erhält das Zeichen bei Spielbeginn. Die „Aufforderung" des Programms ist in Bild 3.1 abgebildet.

Nach der Aufforderung vergeht zunächst eine zufallsgesteuerte Zeitspanne, dann leuchtet die unterste LED-Reihe auf. Die LED ganz links (LED 10) zeigt dem Spieler, daß es losgehen kann, während die vier rechten LEDs (12, 13, 14 und 15) den zu entschlüsselnden Binärcode anzeigen. Bild 3.2 zeigt die Situation, in der Spieler 1 die Taste 5 drücken sollte. Rät der Spieler richtig, wendet sich das Programm dem Spieler 2 zu, andernfalls erhält Spieler 1 so lange einen neuen Versuch, bis seine 17 Sekunden abgelaufen sind. Ist diese Grenze erreicht, so erlischt die unterste Zahl, und eine neue Zahl wird präsentiert.

Das Programm signalisiert Spieler 2 (dem Spieler links), daß er an der Reihe ist, indem es den nach links zeigenden Pfeil ausgibt (Bild 3.3). Sobald beide Spieler ihren Versuch gehabt haben, zeigt das Programm den Sieger dadurch an, daß entweder die drei linken oder die drei rechten LEDs der untersten Zeile

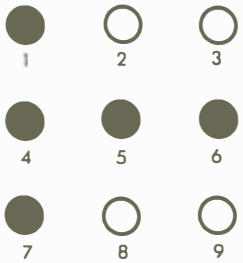



*Abb. 3.1: Aufforderungssignal für rechten Spieler*


*Abb. 3.2: Zu ratende Zahl in der unteren LED-Reihe*

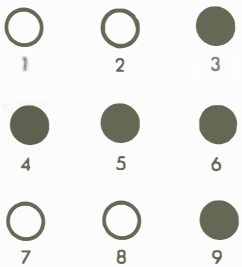



*Abb. 3.3: Der linke Spieler ist an der Reihe*

aufleuchten. Gewonnen hat der Spieler mit der kürzeren Bedenkzeit. Das Spiel geht so lange weiter, bis ein Spieler zehn Siege errungen hat, was für diesen Spieler den Gesamtsieg bedeutet. Dieser Gesamtsieger wird vom Programm dadurch angezeigt, daß die drei LEDs des Spielers zehnmal hintereinander aufleuchten. Bei Spielende geht die Kontrolle zurück an den SYM-Monitor.

## EIN TYPISCHER SPIELVERLAUF

Der Pfeil nach rechts leuchtet auf, und folgende LEDs der unteren Reihe gehen an: 10, 13, 14, 15. Der rechte Spieler (Spieler 1) drückt die C-Taste, worauf die

untere LED-Reihe erlischt, denn die Antwort war falsch. Da Spieler 1 nach dem mißglückten Versuch noch Zeit übrig hat, wird eine neue Zahl angeboten: LEDs 10, 13, 14 und 15 leuchten wieder auf. Jetzt drückt der Spieler Taste 7, was richtig ist. Nun erscheint der Pfeil nach links, und das bedeutet, daß Spieler 2 an der Reihe ist. Diesmal ist die vorgestellte Zahl in den LEDs 10, 12 und 15 codiert. Der linke Spieler drückt Taste 9, woraufhin die LEDs 10, 11 und 12 aufleuchten, um anzuzeigen, daß der Spieler diese Runde gewonnen hat, weil er weniger Zeit für die richtige Antwort gebraucht hat.

Es geht weiter, und der Pfeil nach rechts erscheint. Die zu übersetzende Zahl verbirgt sich in den LEDs 10, 13, 14 und 15. Spieler 1 drückt Taste 7, was den Pfeil nach links aufleuchten läßt. Die nächste Zahl illuminiert LEDs 10 und 14, und Spieler 2 drückt Taste 2. Wieder war Spieler 2 schneller mit der richtigen Antwort bei der Hand, wie die drei linken unteren LEDs unmißverständlich anzeigen.

## DER ALGORITHMUS

Das Flußdiagramm für das Programm erscheint in Bild 3.4. Eine erste Warteschleife sorgt dafür, daß die Zeit richtig gemessen wird, die Spieler 1 für die korrekte Antwort gebraucht hat. Wenn er richtig geraten hat, findet sich die verbrauchte Gesamtzeit in einer Variablen namens TEMP. Für den nun an der Reihe befindlichen Spieler 2 gibt es eine ähnliche Warteschleife. Sobald beide Spieler ihre Antworten übergeben haben, werden ihre jeweiligen Denkzeiten miteinander verglichen, und der Spieler mit der besseren Zeit gewinnt. Wie die Marken 1 und 2 im Flußdiagramm (Bild 3.4) zeigen, geht die Kontrolle nun entweder nach links oder nach rechts. Eine untergeordnete Variable PLYR1 oder PLYR2 zählt die vom jeweiligen Spieler gewonnenen Spiele. Hat ein Spieler eine Runde gewonnen, wird die entsprechende Variable um 1 erhöht und dann mit dem Wert 10 verglichen. Ist dieser Wert noch nicht erreicht, beginnt eine neue Spielrunde, andernfalls wird der entsprechende Spieler zum Sieger erklärt.

## DAS PROGRAMM

Es gibt eigentlich nur eine relativ signifikante Datenstruktur. Sie ist mit NUMTAB bezeichnet und erzeugt die LED-Anzeige der binären Zufallszahlen. Sie erinnern sich, daß LED 10, die „grünes Licht" gibt, immer leuchten muß, während LED 11 immer dunkel bleibt. LEDs 12 bis 15 leuchten oder leuchten nicht, der dargestellten Binärzahl entsprechend. Nicht zu vergessen ist auch, daß Position 6 von Tor 1B nicht benutzt wird. Daraus folgt, daß eine 0 als Binärmuster 00000010 ausgegeben wird, eine 1 als 10000010, eine 2 als 00100010, eine 3 als 10100010 usw. (vgl. Bild 3.5).

Die vollständige Liste von allen 16 möglichen Mustern ist in der NUMTAB-Tabelle des Programms (Bild 3.6) abgespeichert. Betrachten wir als Beispiel

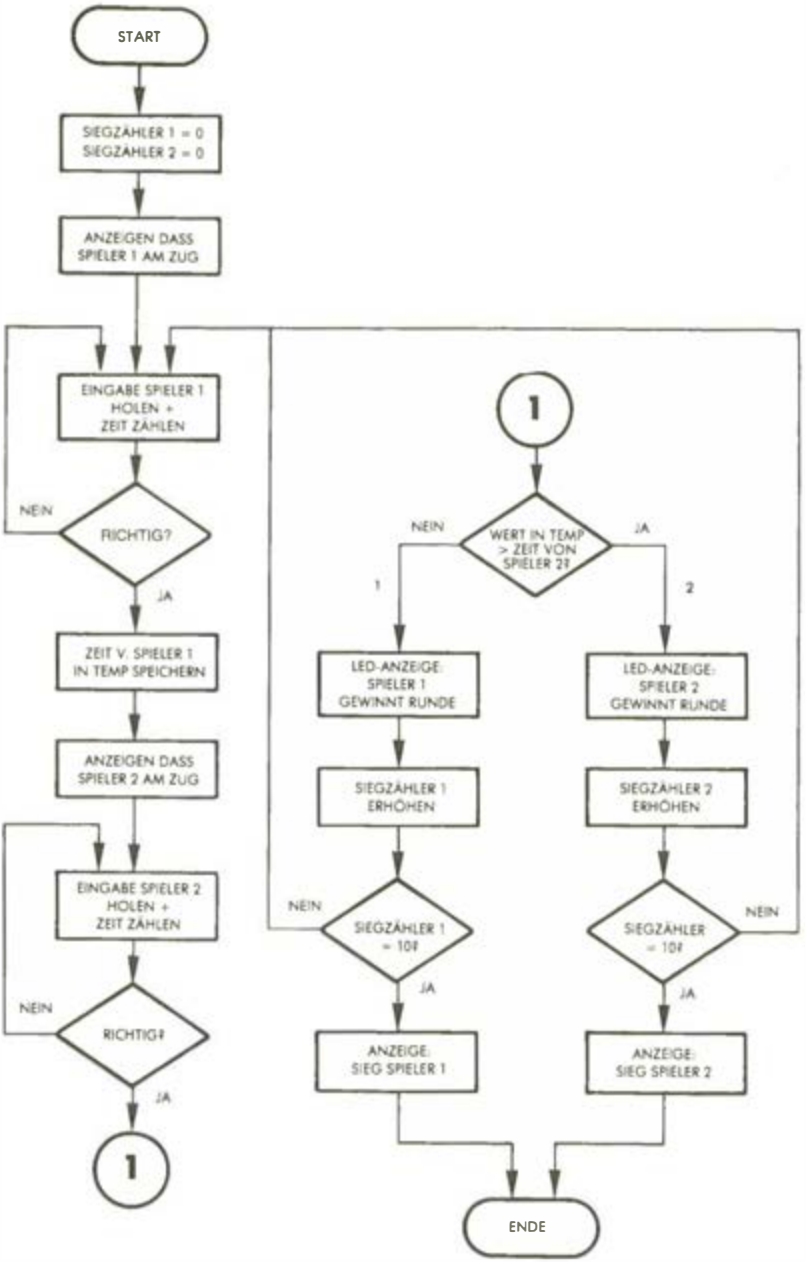
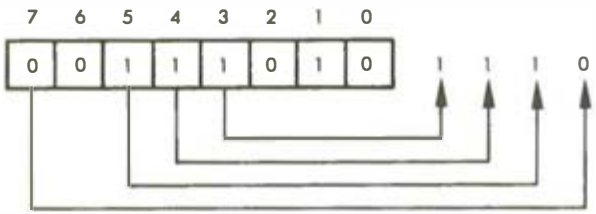



*Abb. 3.4: Flußdiagramm ÜBERSETZEN*

Eintragung 14 in NUMTAB (Programmzeile 0060): 00111010. Die dazugehörige anzuzeigende Binärzahl ist folglich 00111.




Es wird 1110, also 14 daraus. Nochmals: Bit 6 dieses Tores ist immer 0.

**Unterer Speicherbereich**

Im Speicherbereich 0 bis 1D sind die Zwischenspeicher-Variablen und die NUMTAB-Tabelle untergebracht. Nachfolgend die Funktionen der einzelnen Variablen:

| | |
|---|---|
| TEMP | Zufall-Verzögerungszeit |
| CNTHI, CNTLO | Spieler-Zeit bis zur Zugausführung |
| CNT1H, CNT1L | Dauerspeicher für Zeit von Spieler 1 bis zur Zugausführung |
| PLYR1 | bisher erreichte Punktzahl (Rundengewinne) von Spieler 1 (Maximum 10) |
| PLYR2 | dasselbe für Spieler 2 |
| ZAHL | zu ratende Zahl |
| SCR folgende | Arbeitsspeicher für den Zufallszahlen-Generator |

Die Methode im Assembler-Listing, die Speicherplatz bereitstellt, ist bei diesem Programm eine andere als beim Programm des Kapitels 2. Im Musik-Programm wurde Speicherplatz einfach dadurch reserviert, daß die Werte der Variablensymbole mit Hilfe eines Statements

<Variablenname> = <Speicheradresse>

definiert wurden.

In diesem Programm wird der Zuordnungszähler des Assemblers in der Form

```
* = * + n
```

inkrementiert. Die Symbole für die Variablen-Orte werden in diesem Programm also als „Marken" deklariert, während sie im Musik-Programm „Symbole" oder „Konstanten-Symbole" sind.

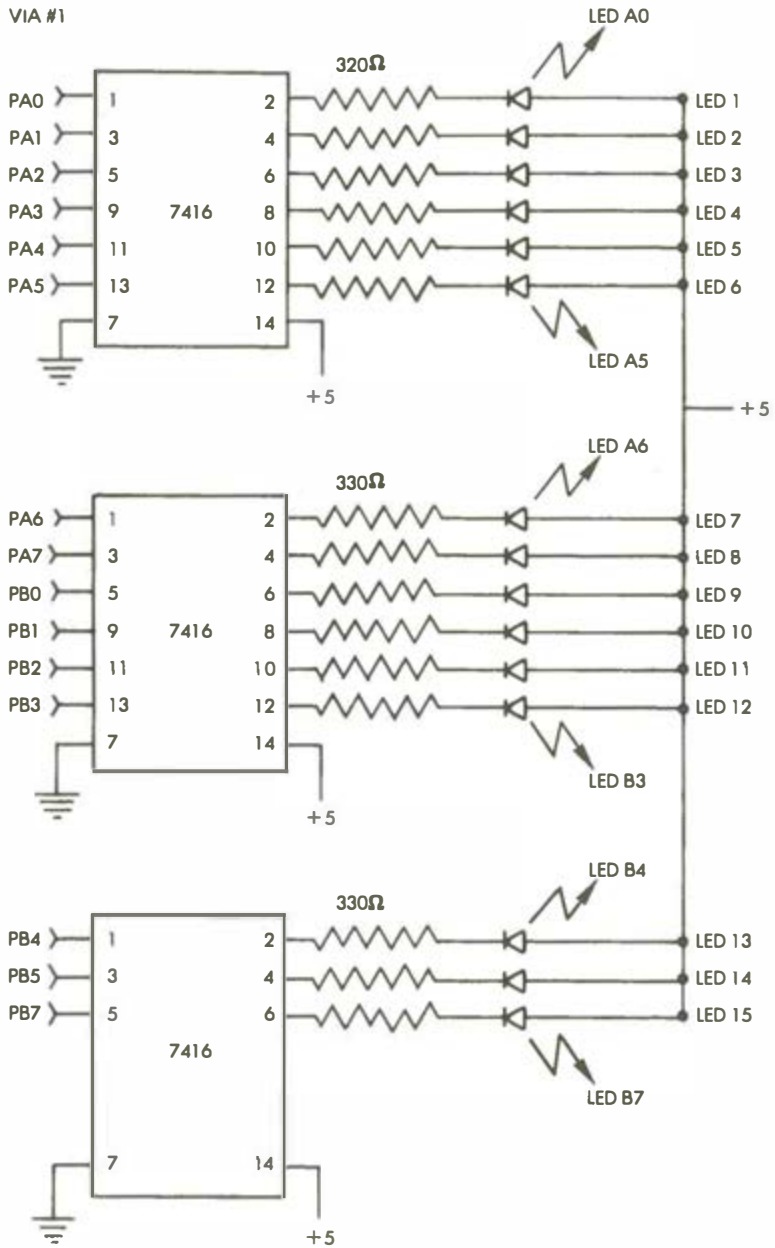



*Abb. 3.5: LED-Verbindungen*

Das Programm dieses Kapitels besteht aus einer Haupt-Routine mit dem Namen MOVE und fünf Unterprogrammen: PLAY, COUNTER, BLINK, DELAY und RANDOM. Nehmen wie sie nun genauer unter die Lupe. Die Datenrichtungsregister A und B für VIA 1 und VIA 3 müssen zunächst initialisiert werden. Als Ausgänge dienen DDR1A, DDR1B und DDR3B:

```
START   LDA #$FF
        STA DDR1A
        STA DDR1B
        STA DDR3B
```

DDR3A wird als Eingang konditioniert:

```
        LDA #0
        STA DDR3A
```

Die Variablen PLYR1 und PLYR2, die die Siege jedes Spielers aufsummieren, werden initialisiert („genullt"):

```
        STA PLYR1
        STA PLYR2
```

Jetzt geht es mit dem Hauptteil von MOVE weiter. Der Pfeil nach rechts erscheint und signalisiert, daß Spieler 1 an der Reihe ist. Zur Erinnerung zeigt Bild 3.5 die LED-Verbindungen. Um einen Pfeil nach rechts zu erzeugen, müssen die LEDs 1, 4, 5, 6 und 7 aufleuchten (siehe auch Bild 3.1). Dies wird bewerkstelligt durch die Ausgabe des entsprechenden Codes an Tor 1A:

```
MOVE    LDA #%01111001
        STA TOR1A       Ausgabe Pfeil nach rechts
```

Die untere LED-Reihe wird gelöscht:

```
        LDA #0
        STA TOR1B
```

Schließlich müssen auch die Zeitmessungszähler auf Null gestellt werden:

```
        STA CNTLO
        STA CNTHI
```

Und das Spiel kann beginnen:

```
        JSR PLAY
```

Die PLAY-Routine wird weiter unten beschrieben, sie kehrt zum aufrufenden Programm zurück mit der gemessenen Laufzeit in CNTLO und CNTHI.

Kehren wir nun zurück zum Hauptprogramm (Zeile 0082 in Bild 3.6). Die in CNTLO und CNTHI von der PLAY-Routine akkumulierte verstrichene Zeit wird in einem Dauerspeicherbereich CNT1L und CNT1H abgelegt, der für Spieler 1 reserviert ist:

```
LDA CNTLO
STA CNT1L
LDA CNTHI
STA CNT1H
```

Jetzt ist Spieler 2 an der Reihe, was der Pfeil nach links anzeigt, der von den LEDs 3, 4, 5, 6 und 9 geformt wird:

```
LDA #%00111100    Pfeil nach links ausgeben
STA TOR1A
```

LED 9, die den Pfeil komplettiert, geht über TOR1B:

```
LDA #1
STA TOR1B
```

Wie vorher wird der Zeitzähler zurück auf Null gestellt:

```
LDA #0
STA CNTLO
STA CNTHI
```

Und Spieler 2 kann loslegen:

```
JSR PLAY
```

Nach der Rückkehr aus dem Unterprogramm werden die von den Spielern verbrauchten Zeiten miteinander verglichen: Gewinnt Spieler 2, wird nach PLR2 verzweigt, andernfalls nach PLR1. Verglichen werden zuerst die höherwertigen Bytes, sind diese gleich, dann auch die niederwertigen:

```
          LDA CNTHI
          CMP CNT1H       High Bytes vergleichen
          BEQ GLEICH
          BCC PLR2        weniger Zeit für Spieler 2?
          BCS PLR1        nein Spieler 1
GLEICH    LDA CNTLO       Low Bytes vergleichen
          CMP CNT1L
          BCC PLR2
          BCS PLR1
```

```
                ;'ÜBERSETZEN'
                ;TESTPROGRAMM FÜR 2 SPIELER, WER ZUERST EINE 4-STELLIGE BINÄR-
                ;ZAHL IN DIE ZUGEHÖRIGE HEXZAHL ÜBERSETZT. DIE SPIELER SIND AB-
                ;WECHSELND AN DER REIHE, WAS DURCH EINEN PFEIL NACH LINKS ODER
                ;RECHTS ANGEZEIGT WIRD. DIE ZAHL ERSCHEINT IN LEDS 12-15, WENN
                ;10 EBENFALLS LEUCHTET. DER SPIELER MUSS DANN DIE RICHTIGE TASTE
                ;DRÜCKEN. DAS ERGEBNIS ERSCHEINT NACH JEDER RUNDE IN DER UNTEREN
                ;LED-REIHE. NACH 10 RUNDENSIEGEN WIRD DER GESAMTSIEGER ANGEZEIGT,
                ;DANACH BEGINNT EIN NEUES SPIEL.
                ;
                ;EIN/AUSGABE
                ;
                TOR1A = $A001           ;LEDS 1-8
                TOR1B = $A000           ;LEDS 9-15
                DDR1A = $A003
                DDR1B = $A002
                TOR3A = $AC01           ;TASTENIMPULS EINGANG
                TOR3B = $AC00           ;TASTENNUMMER AUSGANG
                DDR3A = $AC03
                DDR3B = $AC02
                ;
                ;VARIABLENSPEICHER:
                ;
                        * = $0
                ;
0000:           TEMP    *=*+1
0001:           CNTHI   *=*+1           ;ZWISCHENSPEICHER FÜR RATEZEIT
0002:           CNTLO   *=*+1
0003:           CNT1H   *=*+1           ;ZWISCHENSPEICHER FÜR RATEZEIT SPIELER 1
0004:           CNT1L   *=*+1
0005:           PLYR1   *=*+1           ;RUNDENSIEGE SPIELER 1
0006:           PLYR2   *=*+1           ;RUNDENSIEGE SPIELER 2
0007:           ZAHL    *=*+1           ;SPEICHER FÜR ZU RATENDE ZAHL
0008:           SCR     *=*+6           ;ARBEITSSPEICHER F. ZUFALLSZAHLERZEUGUNG
                ;
                ;TABELLE DER 'UMGEKEHRTEN' ZAHLEN FÜR AUSGABE IN BITS 3-8 VON
                ;TOR1B ODER LEDS 12-15.
                ;
000E:02         NUMTAB .BYTE %00000010
000F:82                .BYTE %10000010
0010:22                .BYTE %00100010
0011:A2                .BYTE %10100010
0012:12                .BYTE %00010010
0013:92                .BYTE %10010010
0014:32                .BYTE %00110010
0015:B2                .BYTE %10110010
0016:0A                .BYTE %00001010
0017:8A                .BYTE %10001010
0018:2A                .BYTE %00101010
0019:AA                .BYTE %10101010
001A:1A                .BYTE %00011010
001B:9A                .BYTE %10011010
001C:3A                .BYTE %00111010
001D:BA                .BYTE %10111010
                ;
                ;HAUPTPROGRAMM
                ;
                        * = $200
                ;
0200:A9 FF      START   LDA #$FF        ;TORE SETZEN
0202:8D 03 A0           STA DDR1A
0205:8D 02 A0           STA DDR1B
0208:8D 02 AC           STA DDR3B
020B:A9 00              LDA #0
020D:8D 03 AC           STA DDR3A
0210:85 05              STA PLYR1       ;RUNDENGEWINNE INITIALISIEREN
0212:85 06              STA PLYR2
0214:A9 79      MOVE    LDA #%01111001
0216:8D 01 A0           STA TOR1A       ;AUSGABE PFEIL NACH RECHTS
0219:A9 00              LDA #0
021B:8D 00 A0           STA TOR1B
021E:85 02              STA CNTLO       ;ZÄHLER INITIALISIEREN
```

*Abb. 3.6: Programm ÜBERSETZEN*

```
0220:85 01              STA CNTHI
0222:20 8C 02           JSR PLAY        ;ZEIT VON SPIELER 1 HOLEN
0225:A5 02              LDA CNTLO       ;ZWISCHENSPEICHER IN DAUERSPEICHER
0227:85 04              STA CNT1L
0229:A5 01              LDA CNTHI
022B:85 03              STA CNT1H
022D:A9 C3              LDA #%00111100  ;AUSGABE PFEIL NACH LINKS
022F:8D 01 A0           STA TOR1A
0232:A9 01              LDA #1
0234:8D 00 A0           STA TOR1B
0237:A9 00              LDA #0
0239:85 02              STA CNTLO       ;ZÄHLER LÖSCHEN
023B:85 01              STA CNTHI
023D:20 8C 02           JSR PLAY        ;ZEIT VON SPIELER 2 HOLEN
0240:A5 01              LDA CNTHI       ;WERT VON SPIELER 2 HOLEN...
0242:C5 03              CMP CNT1H       ;...MIT SPIELER 1 VERGLEICHEN
0244:F0 04              BEQ GLEICH      ;NIEDERWERTIGE BYTES PRÜFEN
0246:90 27              BCC PLR2        ;WENIGER FÜR SPIELER 2 ?
0248:B0 08              BCS PLR1        ;NEIN: SPIELER 1
024A:A5 02      GLEICH  LDA CNTLO       ;HI BYTES GLEICH: LO BYTES PRÜFEN
024C:C5 04              CMP CNT1L       ;VERGLEICHEN
024E:90 1F              BCC PLR2        ;SPIELER 2 BESSER!
0250:B0 00              BCS PLR1        ;SPIELER 1 BESSER!
0252:A9 F0      PLR1    LDA #%11110000  ;RECHTE UNTERE ZEILENHÄLFTE ERLEUCHTEN
0254:8D 00 A0           STA TOR1B
0257:A9 00              LDA #0
0259:8D 01 A0           STA TOR1A       ;UNTERE LEDS LÖSCHEN
025C:A9 40              LDA #$40        ;FÜR SIEGANZEIGE WARTEN
025E:20 E3 02           JSR DELAY
0261:E6 05              INC PLYR1       ;1 PUNKT MEHR FÜR SPIELER 1
0263:A9 0A              LDA #10         ;10 SIEGE ?
0265:C5 05              CMP PLYR1
0267:D0 AB              BNE MOVE        ;WENN NICHT, NEUE RUNDE
0269:A9 F0              LDA #%11110000  ;WENN JA, BLINKMUSTER LADEN
026B:20 C8 02           JSR BLINK       ;SIEGERSEITE BLINKEN LASSEN
026E:60                 RTS             ;SPIELENDE: ZURÜCK ZUM MONITOR
026F:A9 0E      PLR2    LDA #%1110      ;LINKE UNTERE REIHE ERLEUCHTEN
0271:8D 00 A0           STA TOR1B
0274:A9 00              LDA #0
0276:8D 01 A0           STA TOR1A       ;UNTERE LEDS LÖSCHEN
0279:A9 40              LDA #$40        ;FÜR SIEGANZEIGE WARTEN
027B:20 E3 02           JSR DELAY
027E:E6 06              INC PLYR2       ;PUNKTGEWINN SPIELER 2
0280:A9 0A              LDA #10         ;SIND ES 10 PUNKTE?
0282:C5 06              CMP PLYR2
0284:D0 8E              BNE MOVE        ;WENN NICHT, NEUE RUNDE
0286:A9 0E              LDA #%1110      ;WENN JA, BLINKMUSTER LADEN
0288:20 C8 02           JSR BLINK       ;...UND BLINKEN
028B:60                 RTS             ;ENDE
                ;
                ;UNTERPROGRAMM 'PLAY'
                ;ZÄHLT ZEIT FÜR JEDEN SPIELER. BEI FALSCHER ZAHL: NEUER VERSUCH
                ;UND NEUE ZEIT WIRD ZUR ALTEN ADDIERT
                ;
028C:20 F4 02   PLAY    JSR RANDOM      ;ZUFALLSZAHL HOLEN
028F:20 E3 02           JSR DELAY       ;ZUFALLSVERZÖGERUNG
0292:20 F4 02           JSR RANDOM      ;NOCH EINE ZUFALLSZAHL
0295:29 0F              AND #$0F        ;ZU RATENDE ZAHL KLEINER 16
0297:85 07              STA ZAHL
0299:AA                 TAX             ;ALS INDEX VERWENDEN...
029A:B5 0E              LDA NUMTAB,X    ;...UMGEKEHRTES MUSTER AUS TABELLE HOLEN
029C:0D 00 A0           ORA TOR1B       ;...UND IN LEDS 12-15 DARSTELLEN
029F:8D 00 A0           STA TOR1B
02A2:20 B5 02           JSR CNTSUB      ;TASTENEINGABE HOLEN
02A5:C4 07              CPY ZAHL        ;RICHTIG GERATEN?
02A7:F0 0B              BEQ FERTIG      ;WENN JA: FERTIG
02A9:A9 01              LDA #1          ;NEIN: ALTE ZAHL LÖSCHEN
02AB:2D 00 A0           AND TOR1B
02AE:8D 00 A0           STA TOR1B
02B1:4C BC 02           JMP PLAY        ;NEUER VERSUCH MIT ANDERER ZAHL
02B4:60         FERTIG  RTS             ;RÜCKSPRUNG MIT ZEIT IN CNTLO+CNTHI
```

*Abb. 3.6: Programm ÜBERSETZEN (Fortsetzung)*

```
                ;
                ;UNTERPROGRAMM 'COUNTER'
                ;HOLT TASTENEINGABE UND ZÄHLT ZEIT BIS DAHIN
                ;
02B5:A0 0F      CNTSUB  LDY #$F         ;TASTENZÄHLER SETZEN
02B7:8C 00 AC   SCHLIFE STY TOR3B       ;TASTENNUMMER AUSGEBEN
02BA:2C 01 AC           BIT TOR3A       ;TASTE GEDRÜCKT?
02BD:10 0B              BPL FERTIG      ;WENN JA: FERTIG
02BF:88                 DEY             ;NÄCHSTE TASTE
02C0:10 F5              BPL SCHLIFE
02C2:E6 02              INC CNTLO       ;ALLE TASTEN DURCH: ZÄHLER ERHÖHEN
02C4:D0 EF              BNE CNTSUB      ;NEUE ABFRAGE, WENN KEIN ÜBERLAUF
02C6:E6 01              INC CNTHI       ;ÜBERLAUF: HIGH BYTE INKREMENTIEREN
02C8:D0 EB              BNE CNTSUB      ;NEUE ABFRAGE
02CA:60         FERTIG  RTS             ;ZEIT ABGELAUFEN ODER TASTE GEDRÜCKT
                ;
                ;UNTERPROGRAMM 'BLINK'
                ;LEDS MIT BEIM EINSPRUNG IM AKKU GESETZTEN BITS BLINKEN
                ;
02CB:A2 14      BLINK   LOX #20         ;20 MAL BLINKEN
02CD:86 01              STX CNTHI       ;BLINKZÄHLER SETZEN
02CF:85 02              STA CNTLO       ;BLINKREGISTER
02D1:A5 02      SCHLEFE LDA CNTLO       ;BLINKMUSTER LADEN
02D3:4D 00 A0           EOR TOR1B       ;BLINKEN
02D6:8D 00 A0           STA TOR1B
02D9:A9 0A              LDA #10         ;KURZE VERZÖGERUNG
02DB:20 E3 02           JSR DELAY
02DE:C6 01              OEC CNTHI
02E0:D0 EF              BNE SCHLEFE     ;SCHLEIFE FERTIG?
02E2:60                 RTS
                ;
                ;UNTERPROGRAMM 'DELAY'
                ;INHALT VON BESTIMMT DIE VERZÖGERUNGSZEIT
                ;
02E3:85 00      DELAY   STA TEMP
02E5:A0 10      DL1     LDY #$10
02E7:A2 FF      DL2     LDX #$FF
02E9:CA         DL3     DEX
02EA:D0 FD              BNE DL3
02EC:88                 DEY
02ED:D0 F8              BNE OL2
02EF:C6 00              DEC TEMP
02F1:D0 F2              BNE OL1
02F3:60                 RTS
                ;
                ;UNTERPROGRAMM 'RANDOM'
                ;ZUFALLSZAHLENGENERATOR. RÜCKSPRUNG MIT ZUFALLSZAHL IN AKKU.
                ;
02F4:38         RANDOM  SEC
02F5:A5 09              LDA SCR+1
02F7:65 0C              ADC SCR+4
02F9:65 0D              ADC SCR+5
02FB:85 08              STA SCR
02FD:A2 04              LDX #4
02FF:B5 08      RNDSCHL LDA SCR,X
0301:95 09              STA SCR+1,X
0303:CA                 DEX
0304:10 F9              BPL RNDSCHL
0306:60                 RTS

SYMBOLTABELLE

BLINK    02CB    SCHLEFE  02D1    CNT1H    0003    CNT1L    0004
CNTHI    0001    CNTLO    0002    CNTSUB   02B5    DDR1A    A003
DDR1B    A002    DDR3A    AC03    DDR3B    AC02    DELAY    02E3
DL1      02E5    DL2      02E7    DL3      02E9    FERTIG   02B4
GLEICH   024A    FERTIG   02CA    SCHLIFE  02B7    MOVE     0214
ZAHL     0007    NUMTAB   000E    PLAY     02BC    PLR1     0252
PLR2     026F    PLYR1    0005    PLYR2    0006    TOR1A    A001
TOR1B    A000    TOR3A    AC01    TOR3B    AC00    RANDOM   02F4
RNOSCHL  02FF    SCR      0008    START    0200    TEMP     0000
```

*Abb. 3.6: Programm ÜBERSETZEN (Fortsetzung)*

Wenn der Sieger ermittelt ist, leuchten entsprechend die linken oder rechten LEDs der unteren Reihe auf. Verfolgen wir als Beispiel den Fall des Sieges von Spieler 1, zu dem die drei rechten LEDs (13 bis 15) gehören:

```
PLR1    LDA #%11110000
        STA TOR1B
```

Alle anderen LEDs werden gelöscht:

```
        LDA #0
        STA TOR1A
```

Danach kommt eine Verzögerung, und wir sind bereit für eine neue (bis zur 10.) Runde:

```
        LDA #$40
        JSR DELAY
```

Die Punktzahl von Spieler 1 wird um 1 erhöht

```
        INC PLYR1
```

und dann mit 10 verglichen: Ist die Zahl kleiner als 10, erfolgt die Rückkehr zur Hauptroutine MOVE:

```
        LDA #10
        CMP PLYR1
        BNE MOVE
```

Im anderen Fall wurde die Höchstpunktzahl 10 erreicht, das Spiel ist zu Ende, und die LEDs auf der Gewinnerseite blinken:

```
        LDA #%11110000    Blinkmuster
        JSR BLINK
        RTS
```

Die entsprechende Befehlsfolge für Spieler 2 steht ab Adresse PLR2 (Zeile 117 in Bild 3.6):

```
PLR2    LDA #%1110
        STA TOR1B
        LDA #0
        STA TOR1A
        LDA #$40
        JSR DELAY
        INC PLYR2
```

```
LDA #10
CMP PLYR2
BNE MOVE
LDA #%1110
JSR BLINK
RTS
```

**Die Unterprogramme**

***Unterprogramm PLAY***

Die Play-Routine wartet zunächst für eine kurze, unbestimmte Zeitspanne, indem sie das Unterprogramm RANDOM (zur Ermittlung der Dauer) und dann das Unterprogramm DELAY (zur Ausführung) aufruft:

```
PLAY    JSR RANDOM
        JSR DELAY
```

Jetzt liefert das weiter unter beschriebene RANDOM-Unterprogramm eine weitere Zufallszahl, die auf den Bereich von 0 bis 15 „zurechtgestutzt" und als zu entschlüsselnde Binärzahl angezeigt wird. Gespeichert wird sie bei Adresse ZAHL:

```
JSR RANDOM
AND #$0F        höherwertigen Nibble maskieren
STA ZAHL
```

Aus der anfangs beschriebenen NUMTAB-Tabelle wird durch indizierte Adressierung das richtige Binärmuster für die LEDs geholt, wobei die Zahl zwischen 0 und 15 im X-Register steht:

```
TAX             Index-Übertrag nach X
LDA NUMTAB,X    Binärmuster holen
```

Das Muster im Akkumulator wandert ins Ausgaberegister, um die entsprechenden LEDs aufleuchten zu lassen. Beachten Sie die OR-Anweisung mit dem vorherigen Inhalt des Ausgaberegisters, damit LED 9 unverändert bleibt:

```
ORA TOR1B
STA TOR1B
```

Nachdem die Zufallszahl als binäres LED-Muster ausgegeben ist, wartet die Routine auf einen Tastendruck des Spielers. Dies erledigt das CNTSUB-Unterprogramm,

```
JSR CNTSUB
```

desssen Beschreibung später folgt.

Der aus dieser Routine mitgebrachte Wert im Y-Register wird nun mit der zu ratenden Zahl verglichen, die bei Adresse ZAHL abgelegt ist. Verläuft der Vergleich positiv, so erfolgt der Rücksprung, andernfalls werden mit einem AND-Befehl (er sorgt dafür, daß LED 9 unverändert bleibt) alle LEDs gelöscht, und die Routine wird erneut durchlaufen. Zu beachten ist, wie bei jedem CNTSUB-Durchlauf der Zeitzähler dekrementiert wird, so daß bei Erreichen des Nullwertes dem Spieler eine neue Zahl präsentiert werden kann.

```
                CPY ZAHL        Richtig geraten?
                BEQ FERTIG
                LDA #1          Nein: alte Zahl löschen
                AND TOR1B
                STA TOR1B
                JMP PLAY        neuer Versuch
FERTIG          RTS
```

**Übung 3-1:** *Modifizieren Sie PLAY und/oder CNTSUB so, daß bei abgelaufener Zeit die laufende Runde für den Spieler verloren ist (als sei für den Versuch die Maximalzeit verwendet worden).*

***Unterprogramm CNTSUB***

Die vom gerade besprochenen PLAY-Unterprogramm aufgerufene CNTSUB-Routine handhabt eine Tasteneingabe des Spielers und führt Buch über die bis hierhin verstrichene Zeit. Die Tastenabfrage offeriert wenig Neues:

```
CNTSUB      LDY #$0F
SCHLEIFE    STY TOR3B
            BIT TOR3A
            BPL FERTIG
            DEY             Tastenzähler dekrementieren
            BPL SCHLEIFE    nächste Taste
```

Sind alle Tasten abgefragt und wurde keine gedrückt, so wird der Zeitzähler inkrementiert (CNTLO, CNTHI):

```
                INC CNTLO
                BNE CNTSUB
                INC CNTHI
                BNE CNTSUB
FERTIG          RTS
```

Beim Rücksprung aus der Routine ist im Y-Register die eventuell gedrückte Taste gespeichert.

**Übung 3-2:** *Fügen Sie irgendwelche „Nichts tun"-Befehle in die CNTSUB-Routine ein, um die Bedenkzeit zu vergrößern.*

### *Unterprogramm BLINK*

Durch diese Routione werden die im Akkumulator spezifizierten LEDs zum Blinken gebracht: Sie leuchten und erlöschen zehnmal hintereinander. Als Arbeitsspeicher werden die Adressen CNTHI und CNTLO benutzt, so daß deren vorheriger Inhalt verlorengeht. Aufleuchten und Erlöschen müssen sich abwechseln, was durch die Komplementierfähigkeit des EOR-Befehls bewirkt wird. Da für einen Blinkzyklus (Ein/Aus) zwei Komplementierungen der LED-Zustände erforderlich sind, wird die Schleife zwanzigmal durchlaufen. Beachtet werden muß dabei auch, daß das LED-Blinken wahrnehmbar wird, daß die Verzögerung zwischen an und aus also nicht zu kurz ist. Hier das Programm:

```
BLINK       LDX #20          20 Durchläufe
            STX CNTHI        Blinkzähler
            STA CNTLO        Blinkregister
SCHLEIFE    LDA CNTLO        Blinkmuster laden
            EOR TOR1B        Blinken
            STA TOR1B
            LDA #10          kurze Verzögerung
            JSR DELAY
            DEC CNTHI
            BNE SCHLEIFE     Schleife fertig?
            RTS
```

### *Unterprogramm DELAY*

Die DELAY-Routine ist eine dreifach geschachtelte Schleife. Zähler für die Innenschleife ist das auf den Maximalwert $FF gesetzte X-Register. Die mittlere Schleife wird vom Y-Register (hexadezimaler Anfangswert 10) gezählt. Die Außenschleife regelt die Verzögerungszeit mit dem bei Adresse TEMP abgelegten Wert ein drittes Mal:

```
DELAY       STA TEMP
DL1         LDY #$10
DL2         LDX #$FF
DL3         DEX
            BNE DL3
            DEY
            BNE DL2
            DEC TEMP
            BNE DL1
            RTS
```

**Übung 3-3:** *Berechnen Sie die genaue Verzögerungszeit dieser Routine als Funktion der in TEMP befindlichen Zahl.*

### *Unterprogramm RANDOM*

Dieser einfache Zufallszahlengenerator legt eine halbzufällige Zahl im Akkumulator ab. Als „Zahlenküche" dienen dem Generator sechs Speicherstellen ab Adresse 0008 („SCR"). Die Zufallszahl wird folgendermaßen berechnet: Zum Wert 1 werden nacheinander die Inhalte von (SCR + 1), (SCR + 4) und (SCR + 5) addiert:

```
RANDOM    SEC
          LDA SCR +1
          ADC SCR +4
          ADC SCR +5
          STA SCR
```

Die Inhalte der Arbeitsadressen ab SCR werden nun für die nächste „Würfelrunde" präpariert, und zwar durch Abwärtsverschieben der Speicherinhalte:

```
          LDX #4
RNDSCHL   LDA SCR,X
          STA SCR +1,X
          DEX
          BPL RNDSCHL
          RTS
```

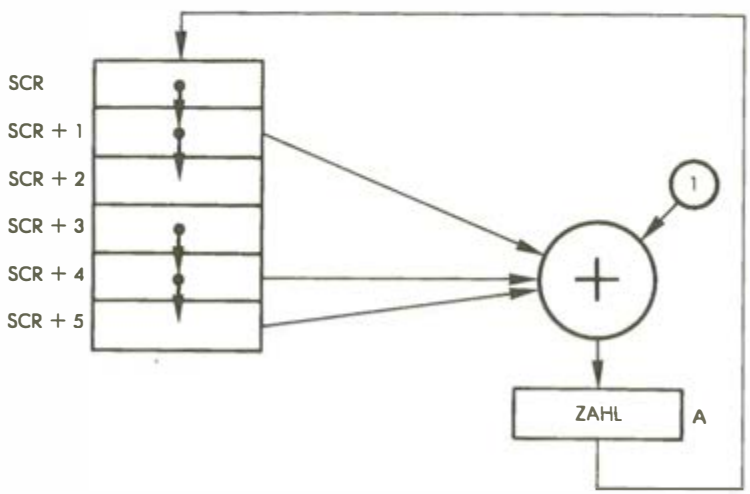



*Abb. 3.7: Zufallszahlen-Generator*

Der Verlauf dieses Vorgangs ist in Bild 3.7 illustriert. Man erkennt die zyklische sieben-Speicher-Verschiebung. Die errechnete Zufallszahl wird in SCR zurückgeschrieben, und alle an SCR anschließenden Speicherinhalte wandern eine Position nach unten, wodurch der Wert von SCR + 5 verlorengeht. Auf diese Weise entstehen einigermaßen „zufällige" Zahlen.

## ZUSAMMENFASSUNG

In diesem Spiel tragen zwei Spieler einen Wettkampf gegeneinander aus. Mit Hilfe von mehrfach geschachtelten Schleifen wird Zeit gemessen. Die zu ratenden Zahlen werden von einem Pseudo-Zufallszahlen-Generator erzeugt. Die Anzeige der Binärzahlen erfolgte unter Verwendung einer speziellen Tabelle. Zur Kennzeichnung des an der Reihe befindlichen Spielers und des Siegers werden die LEDs des Spielbrettes verwendet.

**Übung 3-4:** *Was geschieht, wenn alle Speicherstellen von SCR bis (SCR + 5) zu Beginn den Wert 0 haben?*

# 4

# Hardware-Zufallszahlen-Generator (Hexraten)

## EINFÜHRUNG

In diesem Kapitel werden Zufallszahlen erzeugt, indem der Zeitgeber ein Ein/Ausgabe-Chip schaltet. Algorithmen größerer Komplexität und simultane Licht- und Toneffekte werden entwickelt.

## DIE REGELN

Bei diesem Spiel sollen computererzeugte zweistellige Zahlen geraten werden. Nach der Eingabe einer Versuchszahl zeigt der Computer an, wie nahe sie der geheimen Zahl ist, so daß der Bereich nach und nach eingegrenzt wird. Zu Beginn zeigt das Programm durch einen hohen Ton an, daß es auf eine zweistellige Zahleneingabe wartet. Nach einer richtigen Eingabe des Spielers antwortet das Programm mit einem „Sieg"-Signal. Hat der Spieler danebengegriffen, leuchten 1 bis 9 LEDs auf, um den Abstand zur richtigen Zahl anzudeuten. Bei einem Licht ist die eingegebene Zahl ziemlich weit von der richtigen entfernt, bei neun leuchtenden LEDs dagegen sehr nahe daran.

Wurde richtig geraten, so ertönt ein Trillersignal, und die LEDs flackern. Maximal zehn Versuche sind möglich. Wird die richtige Zahl innerhalb dieser Spanne nicht gefunden, so erklingt ein tiefer Ton, und ein neues Spiel beginnt.

## EIN TYPISCHER SPIELVERLAUF

Ein Piepton des Computers fordert uns auf, eine Zahl einzugeben.

| | |
|---|---|
| Wir versuchen es mit 40 | |
| Es leuchten 4 LEDs auf | Wir liegen etwas daneben |
| Nächster Versuch: C0 | |
| Computerantwort: 3 LEDs | Wir bewegen uns weiter weg |
| Nächster Versuch: 20 | |
| Computerantwort: 3 LEDs | Zahl muß zwischen C0 und 20 liegen |
| Nächster Versuch: 80 | |
| Antwort: 5 | Wir kommen näher |
| Nächster Versuch: 75 | |
| Antwort: 5 | Also nicht unmittelbar unter 80 |

```
Nächster Versuch: 90
Antwort: 4                    Wir entfernen uns
Nächster Versuch: 65
Antwort: 7                    Jetzt kommen wir der Sache näher
Nächster Versuch: 60
Antwort: 9
Nächster Versuch: 5F
Antwort: 8
Nächster Versuch: 61
```

Alle LEDs flackern, und ein Trillerton erklingt – wir haben gewonnen!

## DER ALGORITHMUS

Bild 4.1 zeigt das Flußdiagramm für Hexraten. Der Algorithmus geht ohne viel Schnörkel zur Sache:

- Erzeugung einer Zufallszahl
- Eingabe einer Versuchszahl
- Auswertung, wie nahe der Versuch der geheimen Zahl ist. Neun Näherungsgrade werden von den LEDs angezeigt. Verwendet wird eine Näherungstabelle (Proximitätstabelle).
- Anzeige für Sieg oder nicht
- Weitere Versuche, falls Höchstzahl 10 nicht erreicht.

## DAS PROGRAMM

### Die Datenstrukturen

Das Programm besteht aus einem Hauptteil GETGES und zwei Unterprogrammen LICHT und TON. Die Datenorganisation geschieht mit einer Tabelle LIMITS. Bild 4.1 zeigt das Flußdiagramm, Bild 4.2 das Programm-Listing.

Die LIMITS-Tabelle enthält neun Werte, an denen die Nähe der Versuchs- zur Computerzahl gemessen wird. Die Sequenz ist, bis auf den letzten Wert, eine Exponentialreihe: 1, 2, 4, 8, 16, 32, 64, 128 und 200.

### Implementierung des Programms

Werfen wir einen Blick auf das eigentliche Programm. Es beginnt bei Adresse 200 und ist nicht ohne weiteres verschiebbar. Auf der 0-Seite (zero-page) befinden sich fünf Variable:

VRSUCH enthält die augenblickliche Versuchszahl
VRSUCH# ist die laufende Versuchs-Nr.
DAUER und FREQ sind die bekannten Parameter für die Tonerzeugungs-Routine (TON-Unterprogramm)
ZAHL bezeichnet die zu suchende Zahl des Computers

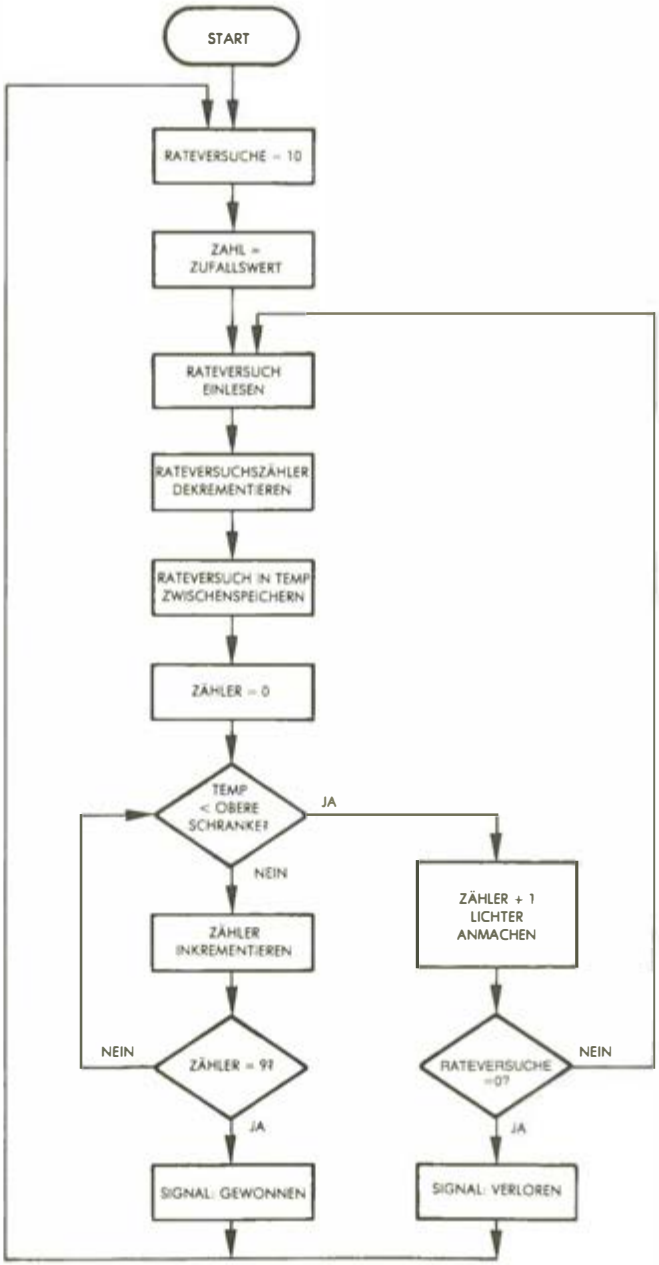



*Abb. 4.1: Flußdiagramm HEXRATEN*

```
                ;'HEXRATEN'
                ;HEXADEZIMALZAHLEN-RATESPIEL
                ;ZIEL DES SPIELS IST, EINE VOM COMPUTER 'ERDACHTE' HEXADEZIMAL-
                ;ZAHL ZU RATEN. NACH ERKLINGEN EINES PIEPTONS WIRD EINE ZWEI-
                ;STELLIGE HEXZAHL EINGEGEBEN, UND DER COMPUTER ZEIGT AN, WIE
                ;NAHE DIESE DER COMPUTERZAHL IST, INDEM EINE PROPORTIONALE ZAHL
                ;VON LEDS AUFLEUCHTET. WIRD DIE ZAHL INNERHALB VON 10 VERSUCHEN
                ;GERATEN, SO BLINKEN ALLE LEDS UND EIN TRILLERSIGNAL ERTONT.
                ;DIE STARTADRESSE DES PROGRAMMS IST BEI $200.
                !
                GETKEY   = $100
                ;6522 VIA 1 ADRESSEN:
                ZEITGEBR = $A004        ;ZWISCHENSPEICHER 0 VON ZEITGEBER 1
                DDR1A    = $A003        ;DATENRICHTUNGSREGISTER TOR A
                DDR1B    = $A002        ;DATENRICHTUNGSREGISTER TOR B
                TOR1A    = $A001        ;TOR A
                TOR1B    = $A000        ;TOR B
                ;6522 VIA 3 ADRESSEN:
                DDR3B    = $AC02        ;DATENRICHTUNGSREGISTER TOR B
                TOR3B    = $AC00        ;TOR B
                ;SPEICHERSTELLEN:
                VRSUCH   = $00
                VRSUCH#  = $01
                DAUER    = $02
                FREQ     = $03
                ZAHL     = $04
                !

                        * = $200
0200: A9 FF               LDA #$FF      ;DATENRICHTUNGSREGISTER SETZEN
0202: 8D 03 A0            STA DDR1A
0205: 8D 02 A0            STA DDR1B
0208: 8D 02 AC            STA DDR3B
020B: 85 02               STA DAUER     ;TONDAUERN SETZEN
020D: A9 0A     START     LDA #$0A      ;10 RATEVERSUCHE
020F: 85 01               STA VRSUCH#
0211: A9 00               LDA #00       ;LEDS LÖSCHEN
0213: 8D 01 A0            STA TOR1A
0216: 8D 00 A0            STA TOR1B
0219: AD 04 A0            LDA ZEITGEBR  ;ZUFALLSZAHL HOLEN
021C: 85 04               STA ZAHL      ;...UND ABSPEICHERN
021E: A9 20     GETGES    LDA #$20      ;KURZER HOHER TON ALS SIGNAL FÜR EINGABE
0220: 20 96 02            JSR TON       ;PIEPTON ERZEUGEN
0223: 20 00 01            JSR GETKEY    ;1. ZIFFER VON RATEVERSUCH HOLEN
0226: 0A                  ASL A         ;IN HÖHERWERTIGEN NIBBLE SCHIEBEN
0227: 0A                  ASL A
0228: 0A                  ASL A
0229: 0A                  ASL A
022A: 85 00               STA VRSUCH    ;SPEICHERN
022C: 20 00 01            JSR GETKEY    ;2. ZIFFER VON RATEVERSUCH HOLEN
022F: 29 0F               AND #%00001111 ;HÖHERWERTIGEN NIBBLE MASKIEREN
0231: 05 00               ORA VRSUCH    ;HÖHERWERTIGEN NIBBLE HINZUFÜGEN
0233: 85 00               STA VRSUCH    ;ENDERGEBNIS SPEICHERN
0235: A5 04               LDA ZAHL      ;ZAHL ZUM VERGLEICHEN HOLEN
0237: 38                  SEC
0238: E5 00               SBC VRSUCH    ;RATEVERSUCH SUBTRAHIEREN, UM RELATIVE
                                        ;NÄHE FESTZUSTELLEN
023A: B0 05               BCS OK        ;POSITIVER WERT: KEINE KORREKTUR
023C: 49 FF               EOR #%11111111 ;NEGATIVWERT ABSOLUT SETZEN
023E: 38                  SEC           ;ZWEIERKOMPLEMENT
023F: 69 00               ADC #00       ;...NICHT EINERKOMPLEMENT
0241: A2 00     OK        LDX #00       ;'NÄHENZÄHLER' SETZEN
0243: DD AD 02  SCHLEIFE  CMP LIMITS,X  ;'NÄHENTABELLE' LIEFERT ZAHL ZU ERLEUCH-
                                        ;TENDER LEDS
0246: B0 27               BCS SIGNAL    ;WERT ZU GROSS: LICHTSIGNAL GEBEN
0248: E8                  INX           ;NÄCHSTE 'NÄHENSTUFE'
0249: E0 09               CPX #9        ;ALLE 9 STUFEN DURCH?
024B: D0 F6               BNE SCHLEIFE  ;WENN NICHT: NÄCHSTE STUFE
024D: A9 0B     SIEG      LDA #11       ;WENN JA: SIEG, BLINKZÄHLER LADEN
024F: 85 00               STA VRSUCH    ;ZWISCHENSPEICHERN
0251: A9 FF               LDA #$FF      ;LEDS AN!
0253: 8D 01 A0            STA TOR1A
```

*Abb. 4.2: Programm HEXRATEN*

```
0256: 8D 00 A0          STA TOR1B
0259: A9 32     WOW     LDA #50         ;TONHOHE
025B: 20 96 02          JSR TON         ;SIEGSIGNAL ERZEUGEN
025E: A9 FF             LDA #$FF
0260: 4D 01 A0          EOR TOR1A       ;TORE KOMPLEMENTIEREN
0263: 8D 01 A0          STA TOR1A
0266: 8D 00 A0          STA TOR1B
0269: C6 00             DEC VRSUCH      ;BLINKEN + TONE FERTIG?
026B: D0 EC             BNE WOW         ;WENN NICHT: NOCHMAL
026D: F0 9E             BEQ START       ;WENN JA: NEUES SPIEL
026F: E8        SIGNAL  INX             ;'NAHENSTUFE' ERHOHEN
0270: A9 00             LDA #0          ;OBERES LED-TOR LOSCHEN
0272: 8D 00 A0          STA TOR1B
0275: 20 BE 02          JSR LICHT       ;LED-MUSTER HOLEN
0278: 8D 01 A0          STA TOR1A       ;...UND SETZEN
027B: 90 05             BCC CC          ;WENN CARRY GESETZT: PB0 = 1
027D: A9 01             LDA #01
027F: 8D 00 A0          STA TOR1B
0282: C6 01     CC      DEC VRSUCH#     ;EIN VERSUCH AUFGEBRAUCHT
0284: D0 98             BNE GETGES      ;WEITERE VERSUCHE: NACHSTEN HOLEN
0286: A9 BE             LDA #$BE        ;TIEFER TON SIGNALISIERT: VERLUST
0288: 20 96 02          JSR TON
028B: 4C 0D 02          JMP START       ;NEUES SPIEL
                ;
                ;ROUTINE ZUM ERZEUGEN VON LED-MUSTERN. EINE SERIE VON LINKSVER-
                ;SCHIEBUNGEN BRINGT '1'-WERTE IN DIE RICHTIGEN POSITIONEN IM
                ;AKKUMULATOR, ZAHL DER VERSCHIEBUNGEN STEHT IN X
                ;
028E: A9 00     LICHT   LDA #0          ;AKKU FÜR MUSTERÜBERNAHME LÖSCHEN
0290: 38        SHIFT   SEC             ;BIT SETZEN
0291: 2A                ROL A           ;...UND HINEINSCHIEBEN
0292: CA                DEX             ;EIN BIT FERTIG
0293: D0 FB             BNE SHIFT       ;WENN NICHT FERTIG: NACHSTES BIT
0295: 60                RTS             ;RÜCKSPRUNG
                ;
                ;TONERZEUGUNGSROUTINE
                ;
0296: 85 03     TON     STA FREQ
0298: A9 00             LDA #00
029A: A6 02             LDX DAUER
029C: A4 03     FL2     LDY FREQ
029E: 88        FL1     DEY
029F: 18                CLC
02A0: 90 00             BCC +2
02A2: D0 FA             BNE FL1
02A4: 49 FF             EOR #$FF
02A6: 8D 00 AC          STA TOR3B
02A9: CA                DEX
02AA: D0 F0             BNE FL2
02AC: 60                RTS
                ;
                ;BEGRENZUNGSTABELLE FUR 'NAHENSTUFEN' (PROXIMITAT)
                ;
02AD: CB        LIMITS  .BYTE  200,128,64,32,16,8,4,2,1
02AE: 80
02AF: 40
02B0: 20
02B1: 10
02B2: 08
02B3: 04
02B4: 02
02B5: 01

SYMBOLTABELLE:

GETKEY   0100    ZEITGEBR  A004    DDR1A    A003    DDR1B    A002
TOR1A    A001    TOR1B     A000    DDR3B    AC02    TOR3B    AC00
VRSUCH   0000    VRSUCH#   0001    DAUER    0002    FREQ     0003
ZAHL     0004    START     020D    GETGES   021E    OK       0241
SCHLEIFE 0243    SIEG      024D    WOW      0259    SIGNAL   026F
CC       02B2    LICHT     028E    SHIFT    0290    TON      0296
FL2      029C    FL1       029E    LIMITS   02AD
```

*Abb. 4.2: Programm HEXRATEN (Fortsetzung)*

Für die LED-Ausgaben und die Tastenfeld-Eingaben werden zunächst wieder die Datenrichtungsregister VIA 1 und VIA 2 konditioniert:

```
        LDA #$FF
        STA DDR1A       Ausgabe
        STA DDR1B       Ausgabe
        STA DDR3B       Ausgabe
```

Die Dauer des von der TON-Routine zu erzeugenden Tons wird bei Adresse DAUER abgelegt. Anfangswert ist FF (hexadezimal):

```
        STA DAUER
```

Speicher VRSUCH#, wo die laufende Versuch-Nr. steht, erhält den Anfangswert 10:

```
START   LDA #$0A
        STA VRSUCH#
```

Die LEDs des Spielbrettes werden abgeschaltet:

```
        LDA #0
        STA TOR1A
        STA TOR1B
```

Jetzt wird die zu ratende Zufallszahl erzeugt. Eine brauchbar zufällige Zahl erhalten wir hier durch Lesen des Wertes von Zeitgeber 1 des VIA 3. Der Wert wird bei Adresse ZAHL gespeichert:

```
        LDA TIMER
        STA ZAHL
```

Ein eigentlicher Zahlengenerator wird nicht benötigt, da die Zufallszahlen – anders als bei den meisten anderen Spielen, die hier beschrieben werden – zu unbestimmten Zeitpunkten abgerufen werden. Man muß sich klarmachen, daß T1C-L des 6522 VIA zwar oft als Verriegelung bezeichnet wird, daß es sich jedoch um einen Zähler handelt. Sein Inhalt wird bei einem Lesevorgang nämlich *nicht* „eingefroren", wie es bei einer Verriegelung der Fall ist, sondern fortlaufend dekrementiert. Wird der Wert 0 erreicht, lädt der tatsächliche Verriegelungsschalter den Zähler neu auf.

Beachten Sie, daß T1L-L in Bild 4.3 zweimal auftaucht, bei Adresse 04 und bei Adresse 06. Das kann eine gewisse Verwirrung auslösen, und man sollte sich deshalb folgendes klarmachen: Adresse 4 korrespondiert mit dem Zähler, und Adresse 6 korrespondiert mit der Verriegelung. Was hier gelesen wird, ist Adresse 4.

Wir sind jetzt startklar. Ein hoher Ton wird erzeugt, um dem Spieler zu signalisieren, daß er seine Versuchszahl eingeben kann. Die Tondauer ist bei

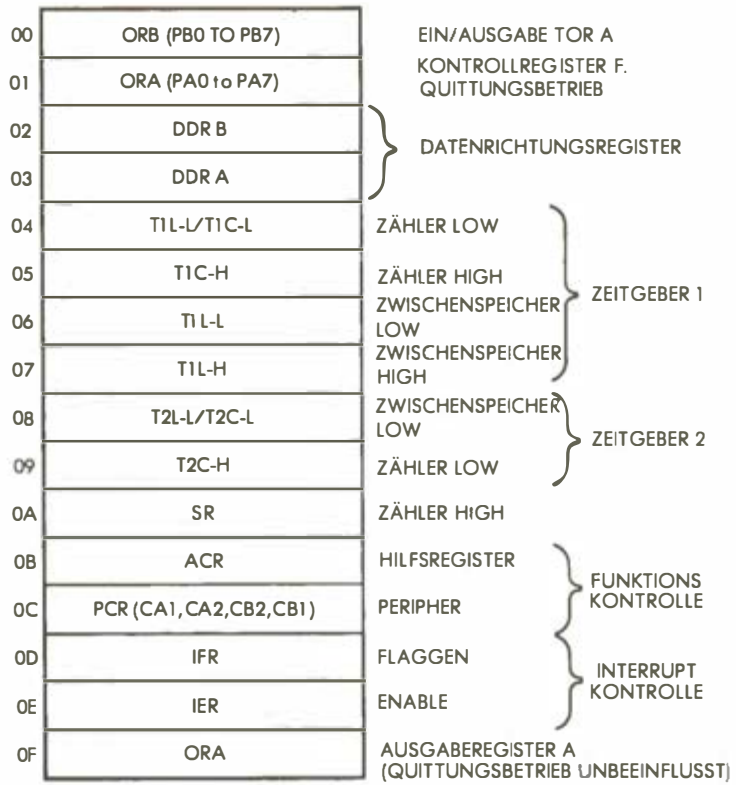



*Abb. 4.3: 6522 VIA Speicheraufteilung*

Adresse DAUER abgespeichert, während sich die Frequenz im Akkumulator befindet:

```
GETGES      LDA #$20        hoher Ton
            JSR TON
```

Bei jedem Ratedurchgang müssen zwei Tasteneingaben gesammelt werden. Die gedrückte Taste wird mit Hilfe der GETKEY-Routine ermittelt, die Tastennummer geht in den Akkumulator. Sobald das erste Zeichen hereingekommen ist, wird es viermal nach links in den höherwertigen Nibble geschoben. Dann wird die zweite Taste geholt (siehe Bild 4.4).

```
            JSR GETKEY
            ASL A
            ASL A
```

```
ASL A
ASL A
STA VRSUCH
JSR GETKEY
```

Ist der Code der zweiten Taste im Akkumulator, wird der Code der ersten Taste aus VRSUCH wiedergeholt und mittels ORA richtig eingepaßt:

```
AND #%00001111
ORA VRSUCH
```

und das Ergebnis geht wieder nach VRSUCH:

```
STA VRSUCH
```

Die Versuchszahl hätten wir also, sie muß jetzt mit der Zufallszahl verglichen werden, die der Computer bei Adresse ZAHL hinterlegt hat. Das besorgt eine einfache Subtraktion:

```
LDA ZAHL
SEC
SBC VRSUCH
```

Zu beachten ist, daß bei negativem Ergebnis eine Komplementierung erforderlich ist:

```
BCS OK          positiv?
EOR #%11111111  negativ: komplementieren
SEC             Zweierkomplement
ADC #0          1 addieren
```

Der „Abstand" der Versuchszahl von der wirklichen Zufallszahl ist also ermittelt, und es muß der Proximitätszähler auf einen Wert zwischen 1 und 9 (nur 9 LEDs werden benutzt) gesetzt werden. Dies leistet eine Schleife, die den absoluten „Abstand" der Versuchszahl von der Zufallszahl einem Vergleichswert der LIMITS-Tabelle gegenüberstellt. Der entsprechende Vergleichswert ist dann die gesuchte Kennziffer für die Proximität, die relative Nähe der beiden Zahlen. Indexregister X erhält den Anfangswert 0, und mittels indizierter Adressierung werden die Vergleichswerte abgerufen, so lange, wie der „Abstand" kleiner als der Vergleichswert ist, oder aber bis X den höchsten Tabellenwert 9 überschreitet:

```
OK          LDX #0
SCHLEIFE    CMP LIMITS,X    Tabellenwert lesen
            BCS SIGNAL
            INX             Wert zu klein
            CPX #9
            BNE SCHLEIFE
```

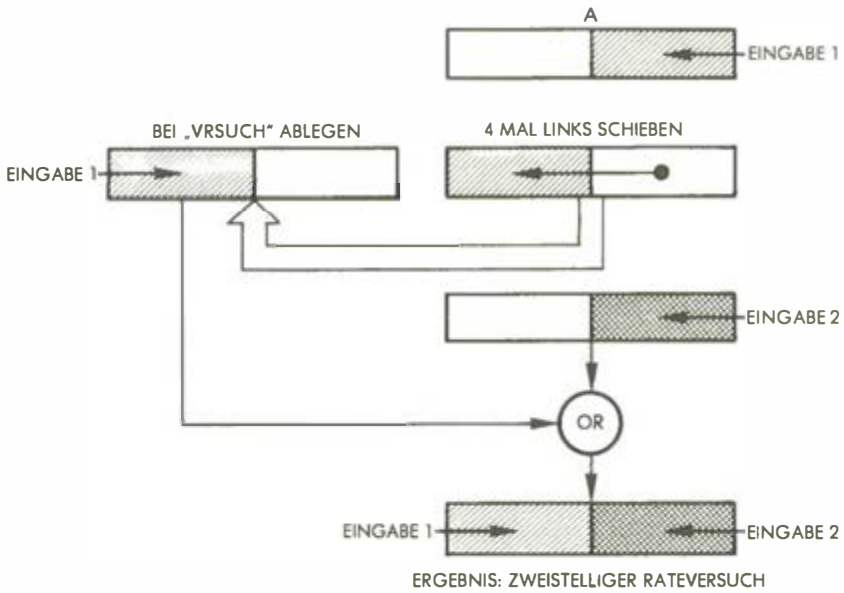



*Abb. 4.4: „Holen" des Spieler-Versuchs*

Hat die Verzweigung nach Signal *nicht* stattgefunden, so ist der „Abstand" 0, der Spieler hat gewonnen. Dies signalisieren die blinkenden LEDs und eine „Siegerfanfare":

```
SIEG    LDA #11
        STA VRSUCH      zwischenspeichern
        LDA #$FF
        STA TOR1A
        STA TOR1B
WOW     LDA #50         Tonhöhe
        JSR TON         Ton erzeugen
```

Der Flackereffekt wird durch wiederholtes Komplementieren der LEDs erzeugt:

```
        LDA #$FF
        EOR TOR1A       Tore komplementieren
        STA TOR1A
        STA TOR1B
```

Das Ganze noch einmal:

```
        DEC VRSUCH
        BNE WOW
```

Wenn der Schleifenindex (VRSUCH) 0 wird, geht es zu START zurück:

```
        BEQ START
```

Betrachten wir jetzt den Fall, daß *nicht* richtig geraten wurde. Es erfolgt dann ein Sprung nach SIGNAL, wobei X den Proximitätswert enthält, also die Anzahl der zu erleuchtenden LEDs. Je nach der Nähe von geratener und zu ratender Zahl gehen LEDs zwischen den Nummern 1 bis 9 an:

```
SIGNAL  INX          Proximität inkrementieren
        LDA #0       oberes LED-Tor löschen
        STA TOR1B
        JSR LICHT    LED-Muster holen
        STA TOR1A
        BCC CC       bei Carry = 1, PB0 = 1
        LDA #1
        STA TOR1B
```

X enthält die Zahl der zu erleuchtenden LEDs, die in das entsprechende, an die Ausgabetore zu sendende Muster verwandelt werden muß. Dies erledigt das Unterprogramm LICHT, das weiter unten beschrieben wird. Soll LED 9 aufleuchten, so setzt LICHT die Carry-Flagge. Um diesen Fall zu berücksichtigen, testet der obige Programmabschnitt eigens das C-Bit (01 wird dann an TOR1B gesendet).
Jetzt wird die laufende Nummer des Ratedurchgangs dekrementiert. Bei 0 hat der Spieler verloren, und das entsprechende Signal ertönt, danach beginnt ein neues Spiel. Andernfalls wird der nächste Rateversuch abgerufen.

```
CC      DEC VRSUCH#
        BNE GETGES   noch Versuche übrig?
        LDA #$BE     tiefer Ton
        JSR TON
        JMP START    neues Spiel
```

**Die Unterprogramme**

***Unterprogramm LICHT***

Diese Routine erzeugt entsprechend dem Wert im X-Register das Muster für die zu erleuchtenden LEDs 1 bis 8. Die erforderlichen 1-Bits werden im Akkumulator einfach so lange nach links geschoben, bis X auf 0 dekrementiert ist. Bild 4.5 zeigt ein Beispiel.

Beim Rücksprung aus der Routine ist im Akkumulator das gewünschte LED-Muster. Ist auch LED 9 dabei, so besteht A aus lauter Einsen, und Carry ist gesetzt:

```
LICHT   LDA #0
SHIFT   SEC          Beginn 1
        ROL A        1 in Position drehen
        DEX          fertig?
        BNE SHIFT
        RTS
```

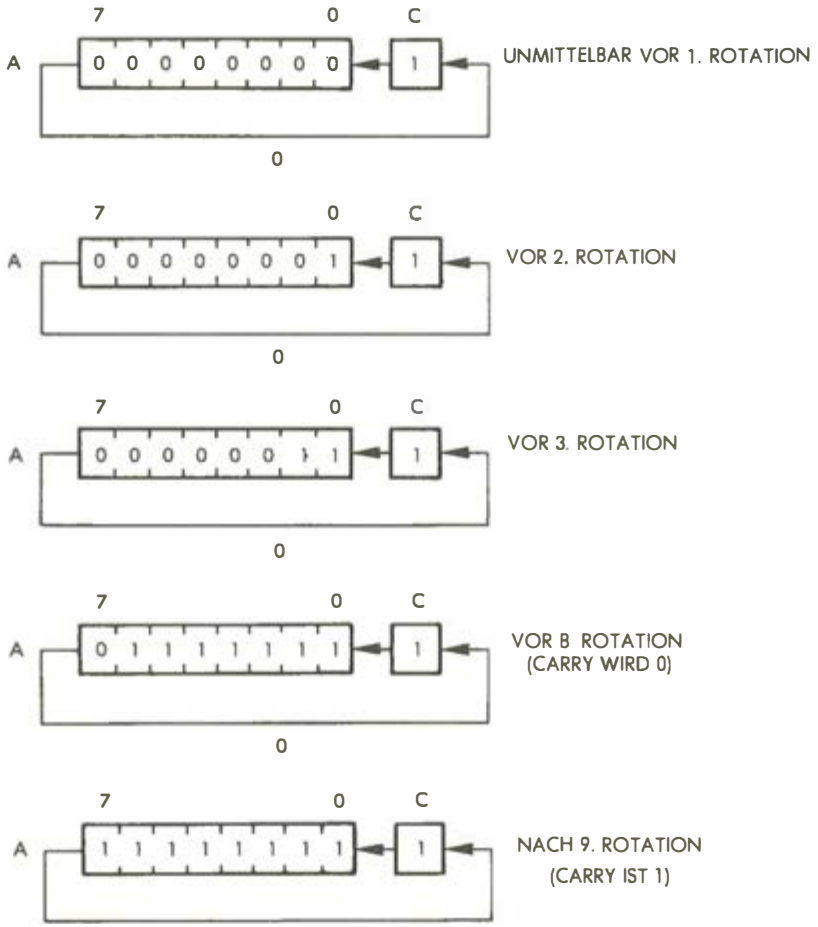



*Abb. 4.5: Erstellen der LED-Muster für 8 LEDs*

### *Unterprogramm TON*

Das Unterprogramm TON erzeugt einen Ton, dessen Dauer in Adresse DAUER fixiert ist und dessen Frequenz der Akkumulator enthält. Als Zähler

für die Innenschleife dient das Y-Register. Der Ton entsteht wieder durch Ein- und Ausschalten des an TOR3B angeschlossenen Lautsprechers:

```
TON     STA FREQ
        LDA #0
        LDX DAUER
FL2     LDY FREQ
FL1     DEY
        CLC
        BCC +2
        BNE FL1
        EOR #$FF
        STA TOR3B
        DEX
        BNE FL2
        RTS
```

## ZUSAMMENFASSUNG

Als Zufallszahlengenerator benutzt dieses Programm den Zwischenspeicher des Zeitgebers 1 (also ein Hardware-Register) und keine Software-Routine. Eine einfache LICHT-Routine zeigt Zahlenwerte an, und die normale TON-Routine erzeugt Töne.

## Übungen

**Übung 4-1:** Verbessern Sie das Programm „Hexraten" durch folgende Erweiterung. Wenn der Spieler verloren hat, zeigt das Programm für etwa drei Sekunden die Zahl an, die der Spieler hätte raten sollen. Erst dann beginnt ein neues Spiel.

**Übung 4-2:** Was würde geschehen, wenn der SEC-Befehl bei Adresse 290 (hexadezimal) weggelassen würde?

**Übung 4-3:** Was sind die Vor- und Nachteile der Zufallszahlenerzeugung mit dem Zeitgeber? Was ist mit aufeinanderfolgenden Zahlen? Stehen sie in Beziehung zueinander? Sind sie identisch?

**Übung 4-4:** Wie oft blinken die LEDs, wenn sie den Spielgewinn signalisieren?

**Übung 4-5:** Studieren Sie die SIEG-Routine. Ertönt der Sieg-Klang ein oder mehrere Male?

**Übung 4-6:** Was ist der Sinn der Befehle bei Adressen 29F und 2A0? (Hinweis: Schauen Sie in Kapitel 2.)

**Übung 4-7:** Sollte das Programm den Zeitgeber starten?

**Übung 4-8:** Steht die Anzahl der erleuchteten LEDs nach einem Rateversuch in einem linearen Verhältnis zur Güte des Versuchs?

# 5

# Simultane Ein/Ausgabe (Magisches Quadrat)

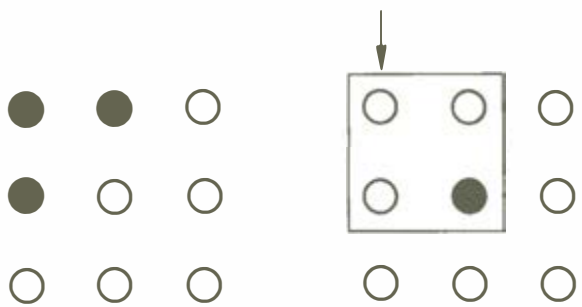
## EINFÜHRUNG

Dieses Programm befaßt sich mit der Erstellung bestimmter visueller Muster. Eine Hardware-Einrichtung, der Zeitgeber, erzeugt Zufallszahlen. Verzögerungen, Blinklichter und Zählmechanismen kommen zur Verwendung.

## DIE SPIELREGELN

Ziel des Spieles ist es, ein quadratisches LED-Muster zu erzeugen, d.h. die LEDs 1, 2, 3, 6, 9, 8, 7 und 4, nicht aber LED 5 in der Mitte zum Leuchten zu bringen.

Zu Spielbeginn erscheint zunächst ein Zufallsmuster. Dieses kann der Spieler nun durch Tastendrücke verändern, wobei jede Taste einen zugehörigen Teilbereich komplementiert („ein" wird „aus", und „aus" wird „ein"). Zum Beispiel: Die zu den LEDs der vier Ecken korrespondierenden Tasten (1, 3, 9, 7) komplementieren jeweils die Teilquadrate, zu denen sie gehören. Taste 1 komplementiert also die LEDs des Teilquadrats 1, 2, 4 und 5. Sind etwa LEDs 1, 2 und 4 erleuchtet, so sieht es nach Drücken von Taste 1 folgendermaßen aus: 1 ist aus, 2 ist aus, 4 ist aus, 5 ist an.

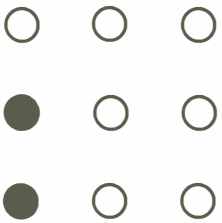


Das Lichtmuster aus LEDs 1, 2, 4 und 5 wurde also durch Taste 1 komplementiert, d.h., die vorher erleuchteten LEDs 1, 2, 4 sind aus, während die vorher

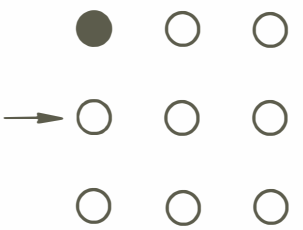
dunkle LED 5 jetzt leuchtet. Wird Taste 1 erneut gedrückt, so leuchten LEDs 1, 2 und 4 wieder auf, und 5 erlischt. Das zweimalige Drücken derselben Taste, eine doppelte Komplementierung also, stellt somit den ursprünglichen Zustand wieder her.

Analog komplementiert Taste 9 das rechte untere Teilquadrat (LEDs 5, 6, 8, 9), Taste 3 das rechte obere Teilquadrat (2, 3, 5, 6), Taste 7 das linke untere (4, 5, 7, 8).

Die „Grenztasten" (gehörig zu den LEDs 2, 4, 6, 8) komplementieren jeweils die drei LEDs der Großquadratseite, zu der sie gehören. Taste 2 etwa komplementiert die LED-Konfiguration 1, 2 und 3. Sind z.B. alle drei LEDs (1, 2, 3) an, so bewirkt Taste 2 ihr Erlöschen. Oder betrachten wir das Beispiel, daß von der linken Quadratseite die LEDs 4 und 7 leuchten:



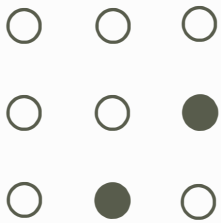
Wird nun Taste 4 gedrückt, so erlöschen LEDs 4 und 7, während LED 1 aufleuchtet:



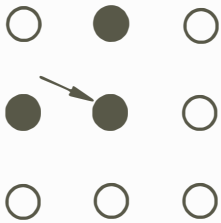
TASTE 4 WURDE GEDRUCKT

Genauso werden die LEDs 7, 8, 9 durch Taste 8 komplementiert und LEDs 3, 6, 9 durch Taste 6.

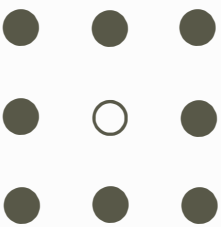
Taste 5 schließlich, die zur zentralen LED gehört, komplementiert das orthogonale LED-Kreuz 2, 4, 5, 6 und 8. Betrachten wir als Beispiel das Anfangsmuster mit nur LEDs 6 und 8 erleuchtet:



Drücken von Taste 5 bewirkt dann folgendes Muster:



Durch eine geeignete Tastenfolge soll schließlich aus dem vorgegebenen LED-Muster das Gewinnmuster gemacht werden, bei dem die acht umsäumenden Außenlichter leuchten:

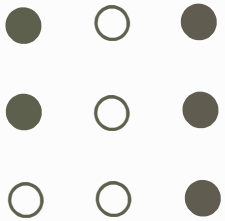


Die mathematische Beweisführung, daß jedes mögliche Ausgangsmuster ins Gewinnmuster überführbar ist, sei dem Leser als Übungsaufgabe überlassen. Hat der Spieler es geschafft, das Gewinnmuster zu erzeugen, quittiert das Programm diesen Erfolg durch Blinken.

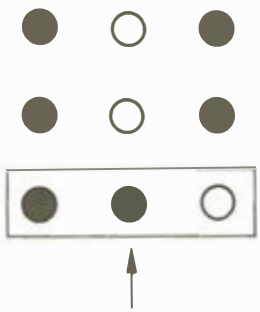
Um ein neues Spiel zu beginnen, muß die 0-Taste gedrückt werden, es erscheint dann ein neues Zufallsmuster erleuchteter LEDs. Alle anderen Tasten werden vom Programm ignoriert.

## EIN TYPISCHER SPIELVERLAUF

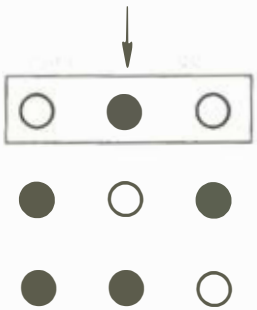
Das Anfangsmuster ist 1-3-4-6-9:



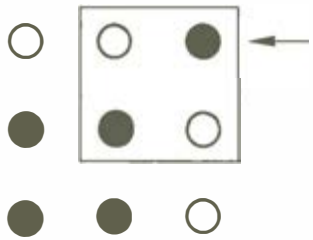
Erster Zug: Taste 8
Es erscheint Muster 1-3-4-6-7-8:



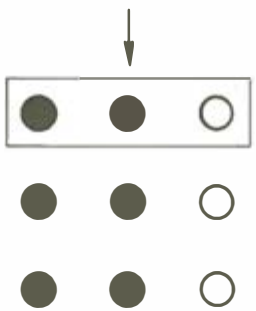
Nächster Zug: Taste 2
Es erscheint Muster 2-4-6-7-8:



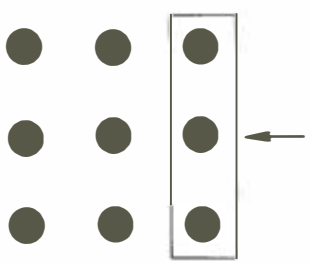
Nächster Zug: Taste 3
Es erscheint Muster 3-4-5-7-8:



Nächster Zug: Taste 2
Es erscheint Muster 1-2-4-5-7-8:



Nächster Zug: Taste 6
Es erscheint Muster 1-2-3-4-5-6-7-8-9:



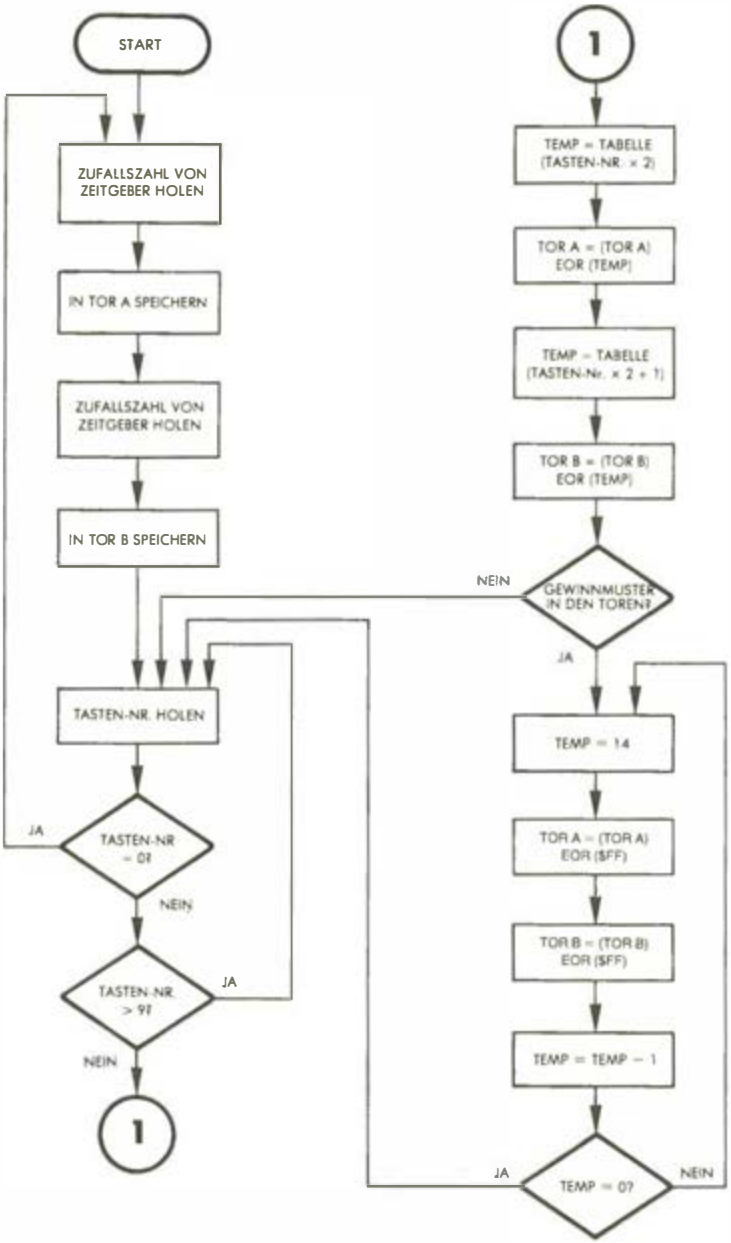



*Abb. 5.1: Flußdiagramm MAGISCHES QUADRAT*

Dies ist zwar ein „klassisches" Muster (alle 9 LEDs leuchten), das Gewinnmuster würde jedoch erfordern, daß die zentrale LED 5 aus ist. Wie das Spiel fortzusetzen ist, möge der Leser selbst ausknobeln. Hier sollten lediglich die Auswirkungen der verschiedenen Tastendrücke demonstriert werden. Für Ungeduldige jedoch ein möglicher Lösungsweg: 2-4-6-8-5.

Hier noch ein allgemeiner Hinweis für die richtige Vorgehensweise: Versuchen Sie als erstes, ein symmetrisches Muster zu erzeugen, aus einem solchen läßt sich die endgültige Lösung meist recht einfach entwickeln. Ein symmetrisches Muster ist im allgemeinen dadurch zu erzielen, daß man die LEDs anmacht, die aus sind, die aber für ein solches Muster an sein sollten.

## DER ALGORITHMUS

Die Muster werden durch Zufallszahlen erzeugt. Dann wird die gedrückte Taste identifiziert, und die entsprechenden LEDs werden komplementiert. Die LED-Gruppen, die zu bestimmten Tastendrücken korrespondieren, werden in einer Tabelle definiert.
Das resultierende Muster wird dann mit dem Gewinnmuster verglichen. Verläuft der Vergleich positiv, gewinnt der Spieler, andernfalls geht das Spiel weiter. Das detaillierte Flußdiagramm erscheint in Abb. 5.1.

## DAS PROGRAMM

### Datenstrukturen

Das Hauptproblem ist, eine effiziente Methode zu finden, mit der ein bestimmter Tastendruck in die Komplementierung des zugehörigen LED-Musters übersetzt werden kann. Die Komplementierung selbst kann leicht mit dem Exklusiv-OR bewerkstelligt werden: Das Muster für die EOR-Anweisung muß da eine 1 haben, wo eine LED-Position zu komplementieren ist, alle anderen Bits müssen 0 sein. Die Aufgabe wird einfach mit einer neunzeiligen Tabelle namens TABELLE bewältigt, wobei jeder Tabelleneintrag einer Taste entspricht. Da neun LEDs zu berücksichtigen sind, werden zwei Bytes benötigt, nur neun der 16 Bits finden allerdings Verwendung. Jedes der neun Bits zeigt durch 1-Werte die LEDs an, die von der gedrückten Taste beeinflußt werden.

Betrachten wir ein Beispiel. Wir haben gesehen, daß die Taste 1 die LEDs 1, 2, 4 und 5 komplementiert. Demnach ergibt sich folgende Tabelleneintragung: 00011011. Entsprechend der Tastennumerierung mit 1 beginnend, erhalten Bits 1, 2, 4 und 5 also den Wert 1. Tatsächlich müssen wir natürlich das komplette 16-Bit-Muster betrachten:

```
0000000000011011
```

Die komplette Tabelle ist in Bild 5.2 aufgeführt.

| KEY = TASTE | PATTERN = BINÄRMUSTER | |
|---|---|---|
| 1 | 00011011 | 00000000 |
| 2 | 00000111 | 00000000 |
| 3 | 00110110 | 00000000 |
| 4 | 01001001 | 00000000 |
| 5 | 10111010 | 00000000 |
| 6 | 00100100 | 00000001 |
| 7 | 11011000 | 00000000 |
| 8 | 11000000 | 00000001 |
| 9 | 10110000 | 00000001 |

*Abb. 5.2: Komplementierungstabelle*

**Implementierung des Programms**

Zu Beginn des Spiels müssen die LEDs in einem Zufallsmuster aufleuchten. Wie im letzten Kapitel wird das durch Einlesen des Wertes vom Zeitgeber des VIA 1 besorgt. Wäre kein Zeitgeber verfügbar, müßte ein Zufallszahlen-Generator eingesetzt werden.

Als Ausgabetore zu den LEDs dienen die Datenrichtungsregister DDRA und DDRB:

```
LDA #$FF
STA DDRA
STA DDRB
```

Jetzt werden die „Zufallszahlen" aus dem Zeitgeber 1 geholt, wobei für 16 Bits (von denen nur neun gebraucht werden) zwei Zahlen gelesen werden müssen:

```
START   LDA T1CL        erste Zahl holen
        STA TOR1        verwenden
        LDA T1CL        zweite Zahl holen
        AND #1          nur Bit 0 wird gebraucht
        STA TOR2        verwenden
```

Die Benutzung von T1CL wurde im letzten Kapitel beschrieben. Das Programm überwacht nun die Tastatur und wartet auf eine Eingabe. Akzeptiert werden nur die Tasten 0 bis 9:

```
TASTE   JSR GETKEY
        CMP #0          Taste 0 gedrückt?
        BEQ START
        CMP #10
        BPL TASTE       Bei Eingabe >9 neue Abfrage
```

```
                    ; 'MAGISCHES QUADRAT'
                    ;JEDE DER TASTEN 1 BIS 9 IST BESTIMMTEN NUMMERN DES LED-FELDES
                    ;FELDES ZUGEORDNET: WIRD EINE TASTE GEDRÜCKT, SO VERÄNDERT SICH
                    ;DAS LED-MUSTER. ZIEL DES SPIELES IST, EIN ANFÄNGLICHES ZUFALLS-
                    ;MUSTER DURCH EINE GEEIGNETE TASTENSEQUENZ IN EIN QUADRAT VON
                    ;LEUCHTENDEN LEDS UMZUWANDELN. GELINGT DAS, SO BLINKEN ALLE
                    ;LEDS. MIT DER '0'-TASTE KANN JEDERZEIT EIN NEUES SPIEL MIT
                    ;NEUEM STARTMUSTER BEGONNEN WERDEN.
                    ;
                    GETKEY = $100
                    T1CL   = $A004          ;LOW REGISTER DES ZEITGEBERS IM 6522 VIA
                    TOR1   = $A001          ;6522 VIA TOR A
                    TOR2   = $A000          ;6522 VIA TOR B
                    TEMP   = $0000          ;ZWISCHENSPEICHER
                    DDRA   = $A003          ;DATENRICHTUNGSREGISTER FÜR TOR A
                    DDRB   = $A002          ;DATENRICHTUNGSREGISTER FÜR TOR B
                           * = $200
                    ;
                    ;ERLAUTERUNGEN: DAS PROGRAMM BENUTZT EIN ZEITGEBERREGISTER ALS
                    ;  ALS ZUFALLSZAHLENQUELLE. IST KEINS VOEHANDEN, KANN EIN ZU-
                    ;  FALLSZAHLENGENERATOR VERWENDET WERDEN, DER AUFGRUND VON WIE-
                    ;  DERHOLUNGSEFFEKTEN JEDOCH NICHT SO GEEIGNET WÄRE. DAS LED-
                    ;  MUSTER WIRD IN DEN REGISTERN VON TOR A GESPEICHERT. DA DER
                    ;  PROZESSOR DIE POLARITÄTEN DER AUSGANGSLEITUNGEN LIEST, WÜRDE
                    ;  EIN ÜBERLADEN DER LEITUNGEN DAZU FÜHREN, DAß DAS PROGRAMM
                    ;  NICHT RICHTIG ARBEITET
                    ;
0200: A9 FF                 LDA #%FF        ;TORE AUF AUSGABE SETZEN
0202: BD 03 A0              STA DDRA
0205: BD 02 A0              STA DDRB
0208: AD 04 A0  START       LDA T1CL        ;1. ZUFALLSZAHL HOLEN
020B: BD 01 A0              STA TOR1
020E: AD 04 A0              LDA T1CL        ;...UND DIE 2.
0211: 29 01                 AND #01         ;UNTERE LED-REIHE AUSMASKIEREN
0213: BD 00 A0              STA TOR2
0216: 20 00 01  TASTE       JSR GETKEY
0219: C9 00                 CMP #0          ;TASTE VON 1-9: IST ES 0?
021B: F0 EB                 BEQ START       ;WENN JA: NEUES SPIEL MIT NEUEM BRETT
021D: C9 0A                 CMP #10         ;TASTE KLEINER 10?
021F: 10 F5                 BPL TASTE       ;TASTE > 9: NEUE TASTE EINLESEN
                    ;
                    ;DER FOLGENDE PROGRAMMTEIL BENUTZT DIE TASTENNUMMER ALS INDEX,
                    ;UM IN DER TABELLE EIN BITMUSTER ZUR LED-KOMPLEMENTIERUNG ZU
                    ;FINDEN.
                    ;
0221: 38                    SEC             ;A FÜR TABELLENZUGRIFF DEKREMENTIEREN
0222: E9 01                 SBC #1
0224: 0A                    ASL A           ;MAL 2, DA TABELLENEINTRAGUNGEN 2 BYTES
0225: AA                    TAX             ;A WIRD INDEX
0226: AD 01 A0              LDA TOR1        ;TORINHALT FÜR KOMPLEMENTIERUNG HOLEN
0229: 5D 6B 02              EOR TABELLE,X   ;EOR VON TOR UND MUSTER
022C: BD 01 A0              STA TOR1        ;TOR 1 WIEDERHERSTELLEN
022F: AD 00 A0              LDA TOR2        ;DASSELBE FÜR TOR 2
0232: 5D 6C 02              EOR TABLLE+1,X  ;...MIT NÄCHSTEM TABELLENEINTRAG
0235: 29 01                 AND #01         ;UNTERE LED-REIHE MASKIEREN
0237: BD 00 A0              STA TOR2        ;...UND WIEDERHERSTELLEN
                    ;
                    ;DIESER ABSCHNITT ÜBERPRÜFT LEDS AUF GEWINNMUSTER.
                    ;
023A: 4A                    LSR A           ;BIT 0 VON TOR 1 INS CARRY-BIT
023B: 90 D9                 BCC TASTE       ;WENN NICHT GEWINNMUSTER: NÄCHSTER ZUG
023D: AD 01 A0              LDA TOR1        ;TOR 1 LADEN FÜR GEWINNTEST
0240: C9 EF                 CMP #%11101111  ;AUF GEWINNMUSTER PRÜFEN
0242: D0 D2                 BNE TASTE       ;NICHT GEWONNEN: NÄCHSTER ZUG
                    ;
                    ;SIEG LÄSST LEDS JEDE HALBE SEK. BLINKEN (4 MAL)
                    ;
0244: A9 0E                 LDA #14
0246: 85 00                 STA TEMP        ;BLINKZAHL LADEN
0248: A2 20     BLINK       LDX #$20        ;VERZÖGERUNGSKONSTANTE FÜR 0.00 SEK.
024A: A0 FF     DELAY       LDY #$FF        ;AUSSENSCHLEIFENVERZÖGERUNG = 2556 * X
024C: EA        DLY         NOP             ;MIKROSEKUNDEN
```

*Abb. 5.3: Programm MAGISCHES QUADRAT*

```
024D: D0 00             BNE +2
024F: 88                DEY
0250: D0 FA             BNE DLY
0252: CA                DEX
0253: D0 F5             BNE DELAY
0255: AD 01 A0          LDA TOR1        ;TORE KOMPLEMENTIEREN
0258: 49 FF             EOR #$FF
025A: 8D 01 A0          STA TOR1
025D: AD 00 A0          LDA TOR2
0260: 49 01             EOR #1
0262: 8D 00 A0          STA TOR2
0265: C6 00             DEC TEMP        ;BLINKZAHL DEKREMENTIEREN
0267: D0 DF             BNE BLINK       ;WENN NICHT FERTIG: NOCHMAL
0269: F0 AB             BEQ TASTE       ;NACHSTER ZUG
                ;
                ;KODETABELLE FÜR DIE LED-KOMPLEMENTIERUNG
                ;
0268: 1B        TABELLE .BYT %00011011, %00000000
026C: 00
026D: 07                .BYT %00000111, %00000000
026E: 00
026F: 36                .BYT %00110110, %00000000
0270: 00
0271: 49                .BYT %01001001, %00000000
0272: 00
0273: BA                .BYT %10111010, %00000000
0274: 00
0275: 24                .BYT %00100100, %00000001
0276: 01
0277: DB                .BYT %11011000, %00000000
0278: 00
0279: C0                .BYT %11000000, %00000001
027A: 01
027B: B0                .BYT %10110000, %00000001
027C: 01


SYMBOLTABELLE:
GETKEY   0100    TICL    A004    TOR1    A001    TOR2    A000
TEMP     0000    DDRA    A003    DDRB    A002    START   0208
TASTE    0216    BLINK   0248    DELAY   024A    DLY     024C
TABELLE  026B
```

*Abb. 5.3: Programm MAGISCHES QUADRAT (Fortsetzung)*

Wurde die 0-Taste gedrückt, beginnt das Program von vorn und präsentiert ein neues LED-Muster. Liegt die Eingabe zwischen 1 und 9, so muß die entsprechende Veränderung beim LED-Muster stattfinden. Der Tastencode selbst wird als Index für die Suche nach dem entsprechenden Komplementierungscode benutzt. Da die Tastennummern von 1 bis 9 gehen, muß zunächst 1 subtrahiert werden, dann ist eine Multiplikation mit 2 erforderlich, da die Tabelle Doppelbytes enthält. Die drei folgenden Instruktionen führen dies aus:

```
SEC
SBC #1          1 subtrahieren
ASL A           mit 2 multiplizieren
```

(Sie erinnern sich: eine Verschiebung nach links ist im Binärsystem gleichbedeutend mit einer Multiplikation mit 2.)

Jetzt wird der Wert als Index ins X-Register übertragen:

```
TAX
```

Das LED-Muster ist im Datenregister von Tor A gespeichert, und es wird nun mit einem EOR-Befehl komplementiert. Dasselbe geschieht mit Tor 2:

```
LDA TOR1
EOR TABELLE,X    Tor 1 komplementieren
STA TOR1
LDA TOR2         dasselbe mit Tor 2
EOR TABELLE+1,X
AND #1           unbenutzte Bits entfernen
STA TOR2
```

Beachten Sie die Assemblerzeit-Arithmetik, mit der das zweite Tabellen-Byte angesprochen wird:

```
EOR TABELLE+1,X
```

Ist das Muster komplementiert, wird auf das Gewinnmuster hin geprüft. Dazu müssen die Inhalte von Tor 2 und Tor 1 mit dem korrekten LED-Muster verglichen werden. Bei Tor 2 ist das 00000001, für Tor 1 ist das 11101111. Bit 0 von Tor 2 ist zufällig auch gerade im Akkumulator und kann nach einer Rechtsverschiebung unmittelbar getestet werden:

```
LSR A            Bit 0 von Tor 2 verschieben
BCC TASTE
```

Der Inhalt von Tor 1 muß dagegen explizit verglichen werden:

```
LDA TOR1
CMP #%11101111
BNE TASTE
```

Um den Sieg anzuzeigen, werden die LEDs zum Flackern gebracht. Zählvariable ist die Adresse TEMP, X fixiert die Verzögerungszeit, und Y zählt in der innersten Schleife. Nach jeder Verzögerung werden die Tore komplementiert:

```
        LDA #14
        STA TEMP     Blinkzahl laden
BLINK   LDX #$20     Verzögerungskonstante 0.08 Sek.
DELAY   LDY #$FF     Außenschleife der Variablenverzö-
                     gerung ergibt 2556 x (X-Inhalt bei
                     Eintritt) Mikrosekunden-Schleife
```

```
DLY     NOP
        BNE +2
        DEY
        BNE DLY
        DEX
        BNE DELAY
        LDA TOR1        Tore komplementieren
        EOR #$FF
        STA TOR1
        LDA TOR2
        EOR #1
        STA TOR2
        DEC TEMP        Blinkzähler dekrementieren
        BNE BLINK       wenn nicht fertig, noch einmal
        BEQ TASTE
```

## ZUSAMMENFASSUNG

Dieses Denkspiel braucht eine Spezialtabelle zu verschiedenen Komplementierungsvorgängen. Der Zeitgeber besorgt Pseudo-Zufallszahlen, nicht ein spezielles Programm. Das LED-Muster wird unmittelbar in den Registern des Ein/Ausgabe-Chips gespeichert.

## Übungen

**Übung 5-1:** *Schreiben Sie für den Schluß des Programms eine Verzögerungsroutine.*

**Übung 5-2:** *Ist das Ausgangsmuster wirklich einigermaßen zufällig?*

**Übung 5-3:** *Implementieren Sie auch Toneffekte.*

**Übung 5-4:** *Ermöglichen Sie eine weitere Veränderung durch Verwendung der „A"-Taste, etwa eine Total-Komplementierung.*

**Übung 5-5:** *(schwierig): Schreiben Sie ein Programm, das auch den Computer spielen und gewinnen läßt.*

**Übung 5-6:** *Erweitern Sie die vorige Übung durch folgendes: Führen Sie Buch über die Zugzahl des Computers, und spielen Sie gegen ihn. Sie gewinnen, wenn Sie weniger Züge brauchen. Ein für beide identisches Anfangsmuster muß verfügbar sein. Fangen Sie an, und dann lassen Sie den Computer „es Ihnen zeigen". Braucht der Computer mehr Züge als Sie, sind Sie entweder ein besonders guter oder glücklicher Spieler oder aber ein schlechter Programmierer. Vielleicht benutzen Sie einen ungeeigneten Algorithmus.*

# 6

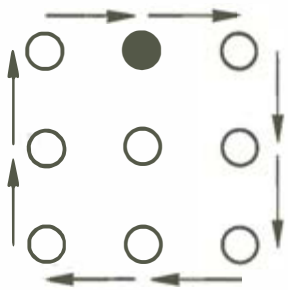
# Eine einfache Realzeit-Simulation (Rundlauf)

## EINFÜHRUNG

Dieses Programm reagiert auf Anwendereingaben in Realzeit. Unterschiedliche Schwierigkeitsstufen werden durch ausgefeiltere Schleifenzählmechanismen verwirklicht.

## DIE SPIELREGELN

Das Quadrat aus den LEDs 1, 2, 3, 6, 9, 8, 7 und 4 wird im Uhrzeigersinn von einem Lichtsignal umkreist:



Ziel des Spieles ist es, den Licht-Rundlauf zu stoppen, indem man die zu irgendeiner LED korrespondierende Taste genau dann drückt, wenn das Lichtsignal diese LED durchläuft. Jedesmal, wenn das gelingt, beginnt ein neuer, etwas schnellerer Rundlauf. Hat der Spieler innerhalb von 32 Rundläufen das Licht nicht anhalten können, so hält es kurz bei LED 4 an und beginnt danach mit einer neuen, langsameren Umrundung. Ein guter Spieler wird den Licht-Rundlauf also immer weiter beschleunigen, bis die Höchstgeschwindigkeit erreicht ist. Dies wird durch gleichzeitiges Aufleuchten aller LEDs von 1 bis 15 angezeigt. Nach einem solchen Erfolg beginnt ein neues Spiel.

Jedesmal, wenn der Spieler das Licht durch Drücken der richtigen Taste „erwischt" hat, bleibt es kurz an der entsprechenden LED-Position stehen.

Das Spiel kann dazu dienen, die Reflexe und das Reaktionsvermögen zu schärfen. Es kann vorkommen, daß die Reaktion eines Spielers auch bei der kleinsten Rundlaufgeschwindigkeit nicht ausreicht, um das Licht anzuhalten. In diesem Fall kann dem Spieler die Möglichkeit gegeben werden, zwei oder gar drei aufeinanderfolgende Tasten gleichzeitig zu drücken, was einer Verlängerung der Reaktionszeit gleichkommt. Drückt der Spieler so z.B. gleichzeitig die Tasten 7, 8 und 9, so hält das Licht an, wenn es sich an einer der drei Positionen befindet.

## DER ALGORITHMUS

Bild 6.1 zeigt das Flußdiagramm für dieses Programm. Es gibt acht Schwierigkeitsgrade, die den steigenden Geschwindigkeiten entsprechen, mit denen der „Lichtpunkt" um das LED-Quadrat wandert. Ein 8-Bit-Zählregister hat gleichzeitig zwei Funktionen (Bild 6.2): Während die drei niederwertigen Bits die augenblickliche Lichtpunkt-Position auf dem LED-Quadrat festschreiben (mit drei Bits kann eine von acht Positionen eindeutig fixiert werden), fungieren die fünf restlichen Bits als Zähler für die Anzahl der Umkreisungen (fünf Bits können also bis zu 32 Runden zählen). Durch Inkrementieren dieses Zählers werden die LEDs nacheinander eingeschaltet. Immer wenn der Lichtpunktzähler von 8 auf 0 springt, wandert ein Übertrag in den Schleifenzähler, wodurch dieser automatisch inkrementiert wird. Die Zuordnung der 8 Bits des Y-Registers zu zwei verschiedenen Zählmechanismen vereinfacht die Programmierung, eine andere Verfahrensweise ist jedoch möglich.

Jedesmal, wenn eine LED aufleuchtet, wird die Tastatur abgefragt, um festzustellen, ob die entsprechende Taste gedrückt wurde. Wesentlich ist dabei, daß ein Tastendruck *vor* dem Aufleuchten der LED ignoriert wird. Dies wird mit Hilfe einer „Ungültig"-Flagge erreicht: Der Algorithmus prüft dabei, ob eine Taste bereits gedrückt war. Ist das der Fall, werden weitere Tastenschlüsse für diese Taste ignoriert. Eine Verzögerungskonstante ergibt sich daraus, daß man den Schwierigkeitsgrad mit 4 multipliziert. Während dieser Verzögerungszeit (die entsprechende LED leuchtet) wird erneut auf Tastenschluß geprüft, es sei denn, eine Taste war zu Beginn der Routine bereits gedrückt. In diesem Fall würde der Tastendruck als Fehlschlag gewertet, und das Programm würde diese Taste nicht mehr überprüfen, weil die „Ungültig"-Flagge gesetzt wäre.

Wird während der Verzögerung die richtige Taste gedrückt (der linke Zweig im Flußdiagramm, in der Mitte von Bild 6.1), wird der Schwierigkeitsgrad dekrementiert, was einer höheren Rotationsgeschwindigkeit des Lichtpunktes entspricht. Bei jedem Fehlversuch dagegen erhöht sich der Schwierigkeitswert (die Geschwindigkeit verringert sich) bis zum Maximum von 15. Beim Erreichen des Schwierigkeitswertes 0 ist das Spiel gewonnen, und alle LEDs leuchten anerkennend auf.

## DAS PROGRAMM

### Datenstrukturen

Vom Programm werden zwei Tabellen benutzt. Die TATAB-Tabelle enthält die Tastennummern, die dem kreisförmigen Lichtumlauf auf den LEDs entsprechen: 1, 2, 3, 6, 9, 8, 7 und 4. TATAB liegt im Speicherbereich 0B bis 12 (siehe Programm-Listing Bild 6.3).
Die zweite Tabelle LITAB enthält die Bitmuster, die zur sequentiellen Erleuchtung der LEDs an das VIA-Tor gesendet werden müssen. Um z.B. LED 1 anzuschalten, muß das Bitmuster 00000001 (01 hexadezimal) übertragen werden, LED 2 erfordert das Muster 00000010 (02 hexadezimal). Die hexadezimalen Codes für die übrigen LEDs sind dann 04, 20, 00, 80, 40 und 08. Zu beachten ist wieder die Sonderbehandlung von LED 9: Tor 1 erhält den Wert 0, aber zusätzlich muß Bit 0 von Tor 2 gesetzt werden. Wir werden weiter unten näher auf diesen Fall eingehen.

### Implementierung des Programms

Die zero-page enthält drei Variable:

| | |
|---|---|
| DURAT | Verzögerung zwischen zwei aufeinanderfolgenden LED-Erhellungen |
| SCHGRD | „umgekehrter" Schwierigkeitsgrad |
| DNTST | Flagge zur Kennzeichnung unzulässiger Tastenschlüsse während der Tastaturabfrage |

Wie gewöhnlich werden zunächst die drei erforderlichen Datenrichtungsregister initialisiert: Tore A und B von DDR1 für die LEDs und DDR3B für die Tastatur:

```
START   LDA #$FF
        STA DDR1A
        STA DDR1B
        STA DDR3B
```

Der Schwierigkeitsgrad wird auf einen mittleren Wert 8 gesetzt:

```
        LDA #8
        STA SCHGRD
```

Das Tor für die Tasteneinblendung steht auf Eingabe:

```
        STA DDR3A
```

Das Y-Register, unser kombinierter Schleifen- und Lichtpunktzähler, erhält den Wert 0:

```
NEUSP   LDY #0
```

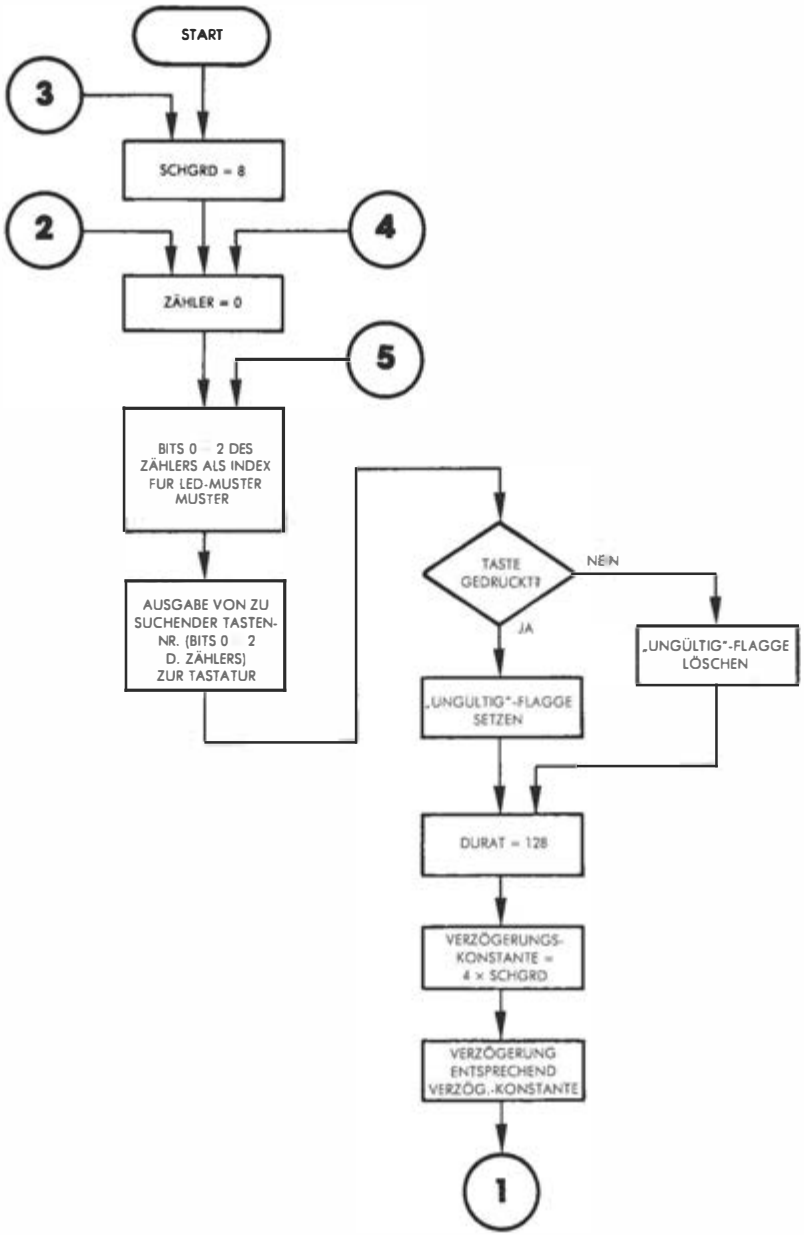



*Abb. 6.1: Flußdiagramm RUNDLAUF*

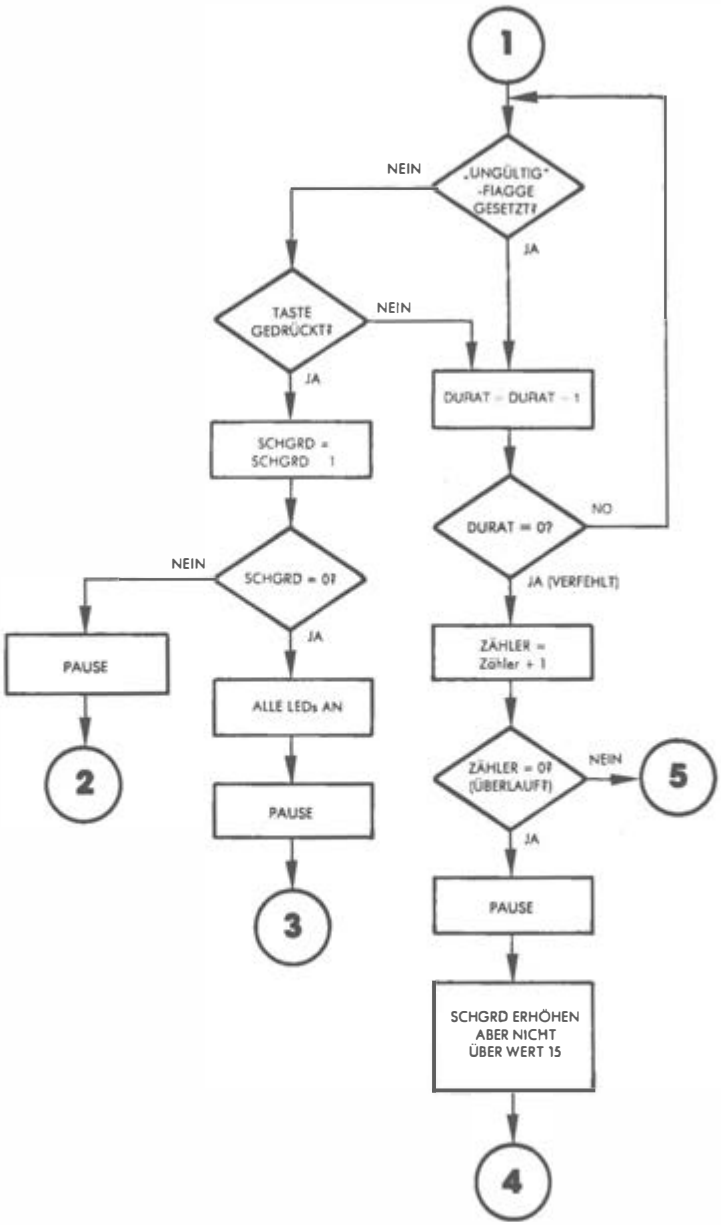



*Abb. 6.1: Programm RUNDLAUF (Fortsetzung)*

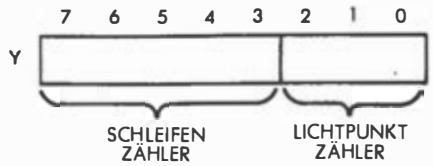



*Abb. 6.2: Doppelter Zähler*

Die „Ungültig"-Flagge wird ebenfalls auf 0 gesetzt:

```
SCHLEIFE    LDA #0
            STA DNTST
```

LED 9 ist dunkel:

```
            STA TOR1B
```

Wir holen nun die drei niederwertigen Bits des Y-Registers, den Lichtpunktzähler, der als Index für die LED-Muster-Tabelle dient:

```
            TYA             Zähler ist Y
            AND #$07        3 untere Bits holen
            TAX             ... und als Index verwenden
```

Durch X-indizierte Adressierung erhalten wir das Muster aus der LITAB-Tabelle und geben es über Tor 1A aus, um die entsprechende LED anzumachen:

```
            LDA LITAB,X     Muster holen
            STA TOR1A       damit LED anmachen
```

Wie bereits angedeutet, braucht das 0-Muster eine Sonderbehandlung: Bit 0 von Tor 1B muß gesetzt werden, um LED 9 anzusprechen:

```
            BNE CHECK       Muster = 0?
            LDA #1          wenn ja: LED 9
            STA TOR1B
```

Die richtige LED leuchtet auf, und es muß die Tastatur daraufhin abgefragt werden, ob die entsprechende Taste vorher schon gedrückt war. Geprüft wird nur die zur gegenwärtig erleuchteten LED korrespondierende Taste:

```
CHECK       LDA TATAB,X     entsprechender Zähler in X
            STA TOR3B
            BIT TOR3A       Abtastimpuls?
            BMI DELAY       wenn nicht, springen
```

```
                        ;    RUNDLAUF'
                        ;REAKTIONSTEST-PROGRAMM
                        ;EIN LICHTPUNKT ROTIERT UM EIN 3X3-LED-QUADRAT, UND DER SPIELER
                        ;MUSS VERSUCHEN, DIEJENIGE TASTE ZU DRÜCKEN, DIE GERADE DURCH-
                        ;LAUFEN WIRD. GELINGT DIES NACH EINER GEWISSEN ZAHLL VON RUND-
                        ;LÄUFEN NICHT, GEHT ES IN DER NÄCHSTE PHASE LANGSAMER, ANDERN-
                        ;FALLS SCHNELLER. HAT MAN SICH ZUR MAXIMALGESCHWINDIGKEIT 'HOCH-
                        ;GEARBEITET', LEUCHTEN ALLE LEDS AUF.
                        ;
                        TOR1A     = $A001       ;LEDS 1 - 8
                        TOR1B     = $A000       ;LEDS 9 - 15
                        DDR1A     = $A003
                        DDR1B     = $A002
                        TOR3A     = $AC01       ;TASTENEINGABE
                        TOR3B     = $AC00       ;TASTENNR. AUSGABE
                        DDR3A     = $AC03
                        DDR3B     = $AC02
                        DURAT     = $0000       ;LAUFZEIT ZWISCHEN 2 NACHBAR-LEDS
                        SCHGRD    = $0001       ;SCHWIERIGKEITSGRAD
                        DNTST     = $0002       ;FLAGGE F. TASTE ZU FRÜH GEDRÜCKT
                        ;
                        ;TABELLE DER BITMUSTER, DIE ZUR SEQUENTIELLEN ERLEUCHTUNG DER
                        ;LEDS (IM UHRZEIGERSINN) AN DAS VIA-TOR GESENDET WERDEN MÜSSEN
                        ;
0003: 01                LITAB     .BYTE $01,$02,$04,$20,$00,$80,$40,$08
0004: 02
0005: 04
0006: 20
0007: 00
0008: 80
0009: 40
000A: 08
                        ;
                        ;TABELLE DER TASTENNUMMERN, UM ZU PRÜFEN, OB DIE ENTSPRECHENDEN
                        ;LEDS JEWEILS AN SIND
                        ;
000B: 01                TATAB     .BYTE 1,2,3,6,9,8,7,4
000C: 02
000D: 03
000E: 06
000F: 09
0010: 08
0011: 07
0012: 04
                        ;
                        ;HAUPTPROGRAMM
                        ;
                                  * = $200
0200: A9 FF     START     LDA #$FF      ;EIN/AUSGABE-REGISTER SETZEN
0202: 8D 03 A0            STA DDR1A
0205: 8D 02 A0            STA DDR1B
0208: 8D 02 AC            STA DDR3B
020B: A9 08               LDA #8
020D: 85 01               STA SCHGRD    ;SCHWIERIGKEITSGRAD = 8
020F: 8D 03 AC            STA DDR3A     ;TASTENEINGABETOR SETZEN
0212: A0 00     NEUSP     LDY #0        ;SCHLEIFEN/LICHTPUNKTZÄHLER SETZEN
0214: A9 00     SCHLEIFE  LDA #0
0216: 85 02               STA DNTST     ;'UNGÜLTIG'-FLAGGE AUF 0
0218: 8D 00 A0            STA TOR1B     ;OBERE LED-TORE LÖSCHEN
021B: 98                  TYA           ;DREI NIEDERWERTIGE BITS ALS ZÄHLER
021C: 29 07               AND #$07      ;FÜR RICHTIGES BITMUSTER HOLEN
021E: AA                  TAX
021F: B5 03               LDA LITAB,X   ;LED-MUSTER AUS TABELLE HOLEN...
0221: 8D 01 A0            STA TOR1A     ;...UND AUSGEBEN
0224: D0 05               BNE CHECK     ;MUSTER = '0'?
0226: A9 01               LDA #1        ;WENN JA: OBERES BIT SETZEN
0228: 8D 00 A0            STA TOR1B
022B: B5 0B     CHECK     LDA TATAB,X   ;TASTE HOLEN
022D: 8D 00 AC            STA TOR3B     ;ABSPEICHERN
0230: 2C 01 AC            BIT TOR3A     ;ABTASTIMPULS?
0233: 30 04               BMI DELAY     ;WENN NICHT: ÜBERSPRINGEN
0235: A9 01     UNGLTG    LDA #01       ;TASTE GEDRÜCKT: FLAGGE SETZEN
```

*Abb. 6.3: Programm RUNDLAUF*

```
0237: 85 02              STA DNTST
0239: A9 80     DELAY    LDA #$80       ;VERZÖGERUNGSLÄNGE SETZEN
023B: 85 00              STA DURAT
023D: A5 01     DL1      LDA SCHGRD     ;SCHWIERIGKEITSGRAD MIT 4 MULTIPLIZIE-
023F: 0A                 ASL A          ;REN, UM VERZÖGERUNG ZU BESTIMMEN
0240: 0A                 ASL A
0241: AA                 TAX
0242: 26 02     DL2      ROL DNTST      ;VERZÖGERUNG ENTSPRECHEND SCHWIERIG-
0244: 66 02              ROR DNTST      ;KEITSGRAD
0246: CA                 DEX
0247: D0 F9              BNE DL2        ;BIS 0 ZÄHLEN
0249: A5 02              LDA DNTST      ;TASTE-GEDRÜCKT-FLAGGE HOLEN
024B: D0 05              BNE NICHT      ;WENN TASTE VORHER GEDRÜCKT:NICHT TESTEN
024D: 2C 01 AC           BIT TOR3A      ;TASTENDRUCK PRÜFEN
0250: 10 19              BPL TREFFR     ;WENN TASTE KORREKT GEDRÜCKT: TREFFER
0252: C6 00     NICHT    DEC DURAT      ;VERZÖGERUNGSZÄHLER HERABSETZEN
0254: D0 E7              BNE DL1        ;SCHLEIFE, WENN NICHT 0
0256: C8                 INY            ;HAUPTDREHZÄHLER ERHÖHEN
0257: D0 BB              BNE SCHLEIFE   ;WENN WENIGER ALS 32 RUNDEN: NÄCHSTE
0259: A6 01              LDX SCHGRD     ;KEIN TREFFER: NÄCHSTE RUNDE LEICHTER
025B: E8                 INX
025C: 8A                 TXA            ;SCHWIERIGKEITSGRAD NICHT ÜBER 15
025D: C9 10              CMP #16
025F: D0 02              BNE OK
0261: A9 0F              LDA #15
0263: 85 01     OK       STA SCHGRD
0265: 20 80 02           JSR WARTEN     ;KLEINE PAUSE
0268: 4C 12 02           JMP NEUSP      ;NEUES SPIEL
026B: 20 80 02  TREFFR   JSR WARTEN
026E: C6 01              DEC SCHGRD     ;NÄCHSTES SPIEL SCHWERER
0270: D0 A0              BNE NEUSP      ;WENN NICHT HÖCHSTSCHWIER.: NEUES SPIEL
0272: A9 FF              LDA #$FF       ;SPIELER HAT HÖCHSTSCHWIERIGKEITSGRAD
0274: 8D 01 A0           STA TOR1A      ;ERREICHT: ALLE LICHTER AN
0277: 8D 00 A0           STA TOR1B
027A: 20 80 02           JSR WARTEN     ;KLEINE PAUSE
027D: 4C 00 02           JMP START      ;NEUES SPIEL BEGINNT
                         ;
                         ;UNTERPROGRAMM 'WARTEN'
                         ;KURZE VERZÖGERUNG
                         ;
0280: A0 FF     WARTEN   LDY #$FF
0282: A2 FF     LP1      LDX #$FF
0284: 66 00     LP2      ROR DURAT
0286: 26 00              ROL DURAT
0288: 66 00              ROR DURAT
028A: 26 00              ROL DURAT
028C: CA                 DEX
028D: D0 F5              BNE LP2
028F: 88                 DEY
0290: D0 F0              BNE LP1
0292: 60                 RTS

SYMBOLTABELLE:
CHECK    022B    DDR1A    A003    DDR1B    A002    DDR3A    AC03
DDR3B    AC02    DELAY    0239    SCHGRD   0001    DL1      023D
DL2      0242    DNTST    0002    DURAT    0000    TREFFR   026B
UNGLTG   0235    TATAB    000B    SCHLEIFE 0214    LP1      0282
LP2      0284    LITAB    0003    NICHT    0252    NEUSP    0212
OK       0263    TOR1A    A001    TOR1B    A000    TOR3A    AC01
TOR3B    AC00    START    0200    WARTEN   0280
```

*Abb. 6.3: Programm RUNDLAUF (Fortsetzung)*

Ist die Taste gedrückt (Impuls von Tor 3A), wird die „Ungültig"-Flagge auf 1 gesetzt:

```
UNGLTG    LDA #1
          STA DNTST
```

Der Tastendruck wird also nicht gewertet. Die Verzögerung, die die LED erhellt läßt, entsteht durch Laden eines Wertes in Adresse DURAT, die als Schleifenzähler dient und später dekrementiert wird, um den Rücksprung nach DL1 zu ermöglichen:

```
DELAY   LDA #$80
        STA DURAT
```

Die Multiplikation des Schwierigkeitswertes in SCHGRD mit 4 geschieht durch zweimaliges Schieben nach links:

```
DL1     LDA SCHGRD
        ASL A
        ASL A
        TAX
```

Das Ergebnis wandert ins X-Register und bemißt die Verzögerungsdauer. Je niedriger der Schwierigkeitswert, desto kürzer ist die Verzögerung:

```
DL2     ROL DNTST
        ROR DNTST
        DEX
        BNE DL2         Schleife bis Zähler = 0
```

Nun wird die „Ungültig"-Flagge geholt und getestet. War die Taste bei Beginn der Routine bereits gedrückt, verzweigt das Programm nach NICHT. Andernfalls, bei Entdeckung eines Tastenschlusses, wird bei TREFFER ein Treffer registriert:

```
        LDA DNTST
        BNE NICHT
        BIT TOR3A       prüfen
        BPL TREFFER
```

Bei NICHT läuft die äußere Verzögerungsschleife weiter: DURAT wird dekrementiert, und es folgt ein Rücksprung nach DL1, solange DURAT nicht 0 wird. Geschieht dies ohne einen Treffer, wird der Hauptzähler Y um 1 erhöht, was die drei unteren Lichtpunktzähler-Bits auf die nächste LED einstellt. Zeigt der Zähler dabei nun auf LED 4 (die letzte in unserer Folge), so erfolgt ein Übertrag nach Bit 3, was automatisch den Schleifenzähler in den oberen 5 Bits inkrementiert (siehe Bild 6.2). Erreicht dieser Zähler wiederum den Wert 32, so wird Y zu 0, und es erfolgt ein Überlauf ins Carry-Bit. Nach diesem Ereignis wird explizit gefragt:

```
NICHT   DEC DURAT
        BNE DL1         Schleife wenn nicht 0
        INY             Zähler erhöhen
        BNE SCHLEIFE    32 Umrundungen?
```

Bei erfolgtem Überlauf des Y-Registers, also nach 32 Durchgängen, wird der Schwierigkeitswert erhöht, d.h. die Umlaufgeschwindigkeit verringert:

```
            LDX SCHGRD      kein Treffer: leichter machen
            INX
```

Nun wird auf den höchsten Schwierigkeitswert 15 getestet:

```
            TXA             nur A wird verglichen
            CNP #16
            BNE OK
            LDA #15         15 ist Maximum
OK          STA SCHGRD
```

Es folgt eine kleine Pause,

```
            JSR WARTEN
```

und es beginnt ein neuer Rundlauf:

```
            JMP NEUSP
```

Auch im Fall, daß ein Treffer gelandet wurde, folgt die Pause:

```
TREFFER     JSR WARTEN
```

– allerdings wird die nächste Runde schwerer, der Schwierigkeitswert SCHGRD also erniedrigt:

```
            DEC SCHGRD
```

Ein Test dieses Wertes auf 0 (höchste Umlaufgeschwindigkeit) folgt. Wurde dieser Wert erreicht, hat der Spieler gewonnen, und alle LEDs gehen an:

```
            BNE NEUSP       wenn nicht 0, nächstes Spiel
            LDA #$FF        gewonnen
            STA TOR1A       „Festbeleuchtung“
            STA TOR1B
```

Auch hier folgt eine Pause, ehe ein neues Spiel beginnt:

```
            JSR WARTEN
            JMP START
```

Die Pause wird durch eine normale Verzögerungsroutine WARTEN verwirk-

licht, eine klassische doppelte Schleife mit zusätzlichen „Untätigkeits"-Befehlen ab Adresse 0286, um Zeit zu gewinnen:

```
WARTEN    LDY #$FF
LP1       LDX #$FF
LP2       ROR DURAT
          ROL DURAT
          ROR DURAT
          ROL DURAT
          DEX
          BNE LP2
          DEY
          BNE LP1
          RTS
```

## ZUSAMMENFASSUNG

Dieses Programm stellt ein Geschicklichkeitsspiel vor, dessen verschiedene Schwierigkeitsgrade den Spieler unterschiedlich stark herausfordern. Da menschliche Reaktionen langsam sind, werden verschiedene Verzögerungsschleifen verwendet. Aus Gründen der Effektivität wird ein zweifacher Zähler in nur einem Register untergebracht: ein Lichtpunkt- und Schleifenzähler.

## Übungen

**Übung 6-1:** *Bei diesem Programm gibt es verschiedene Möglichkeiten zu „mogeln". So kann jede Taste mehrmals schnell hintereinander gedrückt werden, oder es können mehrere Tasten gleichzeitig gedrückt werden, um die Trefferchancen zu erhöhen. Ändern Sie das Programm so ab, daß diese Möglichkeiten nicht mehr bestehen.*

**Übung 6-2:** *Ändern Sie die Umlaufgeschwindigkeit des Lichtpunktes durch Modifizierung des geeigneten Speicherwertes. (Hinweis: Die richtige Adresse ist zu Beginn des Kapitels namentlich genannt.)*

**Übung 6-3:** *Fügen Sie Toneffekte hinzu.*

# 7

# Echtzeit-Simulation (Spielautomat)

## EINFÜHRUNG

Dieses Programm ist eine Echtzeit-Simulation eines bekannten elektromechanischen Apparates. Ausgeführt werden komplexe Auswertungen (mittels indizierter Adressierung) und eine spezielle Datenorganisation für einen flüssigen Programmablauf.

## DIE SPIELREGELN

Simuliert wird ein Glücksspielautomat, der in Spielerstädten wie Las Vegas weit verbreitet ist. Das Drehen der Glücksräder wird von den drei vertikalen LED-Reihen 1-4-7, 2-5-8 und 3-6-9 imitiert, die „rotieren" und irgendwann anhalten (Bild 7.1). Die mittlere horizontale LED-Zeile 4-5-6 zeigt in diesem Fall an, ob der Spieler gewonnen hat.

Wieviel Punkte der Spieler hat, wird jeweils durch die entsprechende LED angezeigt. So ist bei Spielbeginn LED 8 erleuchtet, um das „Startkapital" von 8 Punkten sichtbar zu machen.

Der Spielautomat wird durch Drücken irgendeiner Taste in Gang gesetzt, und die drei vertikalen LED-Reihen beginnen zu rotieren. Sobald sie anhalten, zeigt die LED-Zeile 4-5-6, ob und wieviel gewonnen wurde: Ist eine oder keine LED dieser Zeile erleuchtet, hat man verloren, und das derzeitige Punktekonto wird um 1 verringert. Leuchten zwei LEDs, wird die Punktzahl um 1 erhöht. Leuchten alle drei LEDs der Mittelzeile, so wächst der Punktestand um 3 Punkte.

Immer wenn der Punktestand auf 0 absinkt, ist die gesamte Spielrunde verloren. Erreicht man dagegen die Punktzahl 16, so hat man die jeweilige Runde gewonnen. Alle Ereignisse während des Spielverlaufs werden von akustischen Signalen begleitet. Während der Rotationsphase der LEDs ertönt ein dauerndes Klicken, um den Bewegungsvorgang nachzuahmen. Bleiben die LEDs schließlich stehen, zeigt ein hoher Lautsprecherton eine Gewinnsituation an, ein tiefer Ton dagegen eine Verlustkombination. Besonders auffallend wird die Gewinnstellung mit drei erleuchteten LEDs in der Mittelreihe „kommentiert". Der Lautsprecher gibt in diesem Fall einen dreifachen hohen Piepton von sich, um den Gewinn von 3 zusätzlichen Punkten zu markieren. Beim Erreichen der

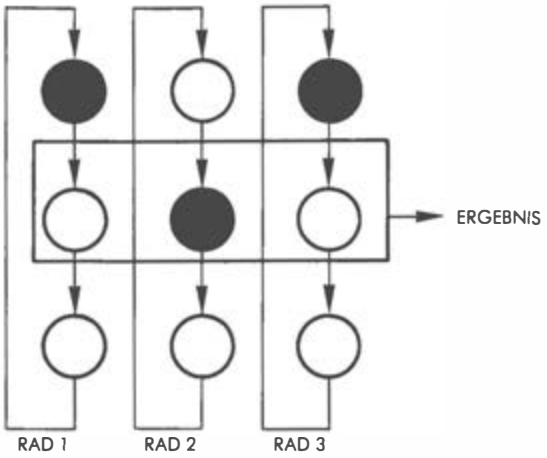



*Abb. 7.1: Der Spielautomat*

maximalen Punktzahl 16 hat der Spieler einen „Haupttreffer" gelandet: Alle LEDs leuchten gleichzeitig auf, und ein aufsteigender Sirenenton erklingt. Im Gegensatz dazu begleitet ein absteigender Heulton das Erreichen der Untergrenze von 0 Punkten.

Anders als die wirklichen „Spielhöllen"-Maschinen läßt dieser Automat Sie dauernd gewinnen, zu Ihrem Glück. Sie werden jedoch wissen, daß dies weniger eine Sache des Glücks als der Programmierung derartiger Automaten ist. Sie werden feststellen, daß sich der Zählvorgang und die Gewinnchancen innerhalb dieses Programms unschwer modifizieren lassen.

## EIN TYPISCHER SPIELVERLAUF

Wenn das Spiel beginnt, zeigt LED 8 die Anfangspunktzahl 8 an, und der Spieler kann nun irgendeine Taste drücken, um den Automaten in Gang zu setzen. Wir drücken z.B. die 0-Taste, und es geht los. Nach Abschluß der Rotationsphase sind LEDs 4, 5 und 9 erleuchtet (Bild 7.2). Das ist eine Gewinnkombination, das Punktekonto erhöht sich um 1, und ein hoher Ton ist die zugehörige Begleitmusik. Als nächstes leuchtet LED 9 auf, um den neuen Punktestand von 8 + 1 = 9 sichtbar zu machen. Wieder drücken wir die 0-Taste, und diesmal leuchtet nur LED 5 in der mittleren Reihe auf. Wie weiter oben besprochen, ist ein oder kein Licht eine Verluststellung, unser Punktestand vermindert sich also wieder um den Wert 1 auf 8 Punkte. Erneutes Drücken der 0-Taste läßt diesmal LEDs 5 und 6 leuchten, was die Gesamtpunktzahl wieder auf 9 bringt.

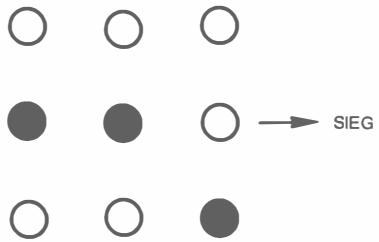



*Abb. 7.2: Gewinnstellung*

Und wieder die 0-Taste: LED 4 bleibt an, und danach zeigt LED 8 den aktuellen Punktestand.
Nochmals 0 gedrückt, und LED 7 zeigt uns den Gesamtstand 7. Und so weiter.

## DER ALGORITHMUS

Das Flußdiagramm im Bild 7.3 zeigt den prinzipiellen Ablauf des Spielautomatenprogramms. Nach der Anzeige der Anfangspunktzahl startet ein Tastendruck des Spielers das Spiel, und die LEDs rotieren. Danach folgt die Ergebnisauswertung: Der Gesamtpunktestand wird entsprechend korrigiert, und die Sieg- bzw. Verlustsituation wird angezeigt.

Die LED-Positionen einer senkrechten Spalte sind von oben nach unten mit 0, 1 und 2 bezeichnet. Der Rotationseffekt entsteht durch das sequentielle Aufleuchten der Positionen 0, 1 und 2 und wieder 0, 1 und 2 usw. Mit der Zeit läßt die Drehgeschwindigkeit dann nach, und schließlich bleiben die LEDs stehen. Dieser Verlangsamungseffekt wird dadurch erzielt, daß die Verzögerungszeiten zwischen den einzelnen LED-Aktivierungen in einer Spalte stetig größer werden. Jedem „Rad", also jeder dreizeiligen LED-Spalte, ist ein Zählregister zugeordnet. Die Anfangswerte dieser Zähler für Rad 1, Rad 2 und Rad 3 werden von einem Zufallszahlengenerator gesetzt. Eine gewisse „Zufallskontrolle" übernimmt ein programmierbarer Werterahmen LOLIM und HILIM, der in einem Zwischenspeicher kontinuierlich bis auf den Wert 0 gebracht wird. Ist dieser Wert erreicht, leuchtet die nächste LED des entsprechenden Rades auf. Zusätzlich wird der ursprüngliche Zählerwert inkrementiert, wodurch sich das Aufleuchten der nächstfolgenden LED verzögert. Beim Überlauf des Zählers (Wert 0) kommt das jeweilige Rad schließlich zum Stillstand. Eine synchrone Aktualisierung der Zwischenspeicher resultiert also in einem asynchronen Aufleuchten der LED-Spalten. Stehen alle drei Räder schließlich still, erfolgt die Auswertung der entstandenen Lichterkombination.

Bild 7.4 zeigt das Flußdiagramm dieser DISPLAY-Routine, die wir nun genauer analysieren wollen. Die ersten drei Schritte setzen die LED-Zeiger auf die oberste Reihe (0-Position). Dann werden die drei Zähler für die Steuerung der Zeitintervalle jedes Rades mit Zufallszahlen gefüllt, wobei diese Zufallszah-

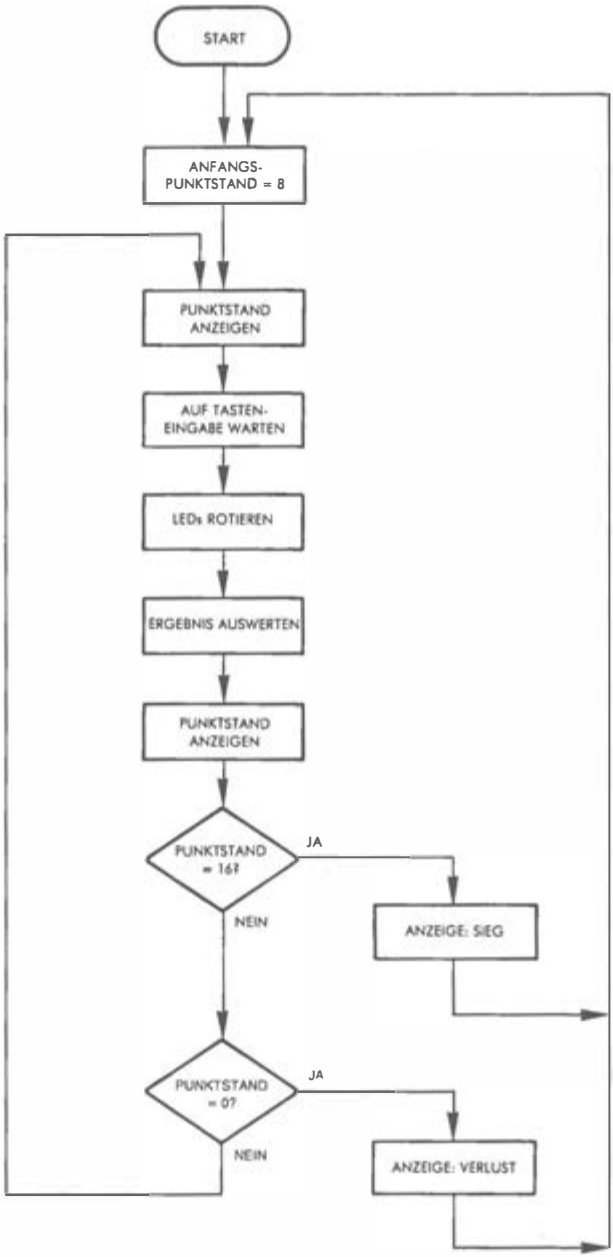



*Abb. 7.3: Flußdiagramm (Grobstruktur)*

len innerhalb bestimmter Grenzen gehalten werden. Die drei Zählerinhalte werden schließlich in die Zwischenspeicher kopiert, die für die Dekrementierung der Verzögerungskonstanten verantwortlich sind.

Werfen wir nun einen Blick auf die nächsten Schritte, die in Bild 7.4 dargestellt sind:

4. Der Rad-Zeiger X deutet auf die äußerste rechte Spalte: X = 3.
5. Der zur aktuellen Spalte (jetzt also Spalte 3) korrespondierende Zähler wird auf den Wert 0 getestet, um zu sehen, ob das Rad angehalten hat. Beim ersten Mal ist der Wert natürlich nicht 0.

6,7. Die Verzögerungskonstante für die vom Radzeiger angegebene LED-Spalte wird erneut auf den Wert 0 geprüft. Ist die Verzögerung nicht 0, geschieht nichts weiter in dieser Spalte, und wir gehen eine Spalte weiter nach links:

16. Spaltenzeiger X wird dekrementiert: X = X − 1
17. Ist X = 0, so erfolgt ein Sprung zu Schritt 5, denn jedesmal, wenn X gleich 0 ist, kann das in allen drei Spalten der Fall sein. Alle Radzähler müssen also auf diesen Wert getestet werden.
18. Sind tatsächlich alle Zähler 0, ist die Rotationsphase beendet, und die Routine wird verlassen. Sind nicht alle Zähler 0, findet (nach einer Verzögerung) der Rücksprung nach 4. statt.

Zurück zu Schritt 7:

7. Hat die Verzögerungskonstante den Wert 0 erreicht, muß die nächstuntere LED in der Spalte aufleuchten.
8. Der LED-Zeiger des Rades, dessen Nummer im Radzeiger steht, wird um 1 erhöht.
9. Der LED-Zeiger wird auf den Wert 4 geprüft: Ist 4 nicht erreicht, geht es weiter, andernfalls wird der Wert auf 1 zurückgesetzt. (Extern erscheinen die LEDs von oben nach unten in den Positionen 1, 2 und 3. Nach der dritten LED leuchtet also wieder die erste auf.)

10,11. Die LED auf dem Brett muß angemacht werden. Das richtige Muster wird der Tabelle LICHTAB entnommen.

12. Der Zähler des entsprechenden Rades wird um 1 erhöht. Zu beachten ist, daß hier kein Test auf den Wert 0 stattfindet, das ist nur der Fall, wenn das Programm links von Rad 1 ankommt. Dies findet Berücksichtigung bei Schritt 18 im Flußdiagramm, wo die Zähler auf 0 geprüft werden.
13. Der neue Zählerwert wird in den Verzögerungskonstantenspeicher übernommen, mit der Folge, daß die Aktivierung der nächsten LED verzögert stattfindet.
14. Die aktuellen Lichtmuster aller Spalten werden für die Ausgabe kombiniert und ausgegeben.
15. Während die LEDs der Reihe nach „gezündet" werden, erhält gleichzeitig der Lautsprecher seine Ein/Aus-Signale.
16. Es geht eine Spalte weiter nach links und dann weiter wie gehabt.

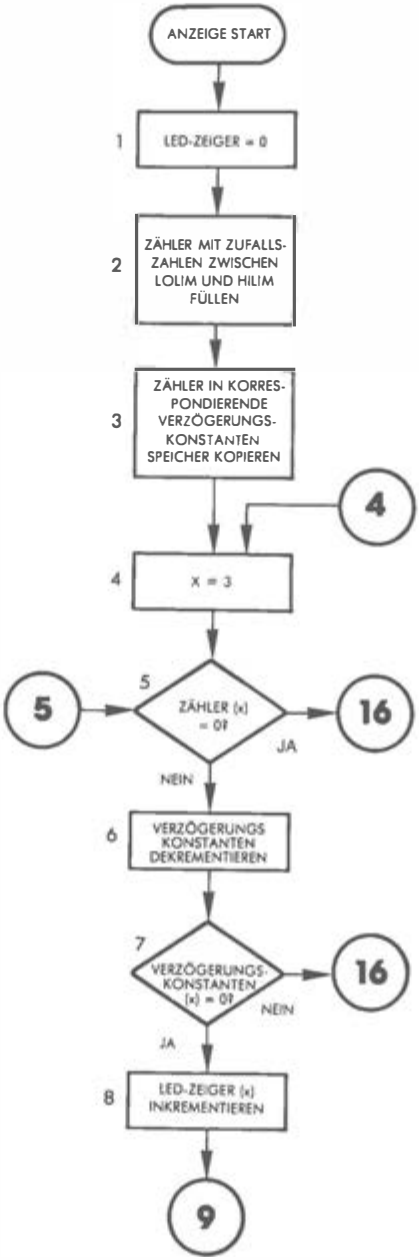



*Abb. 7.4: Flußdiagramm DISPLAY*

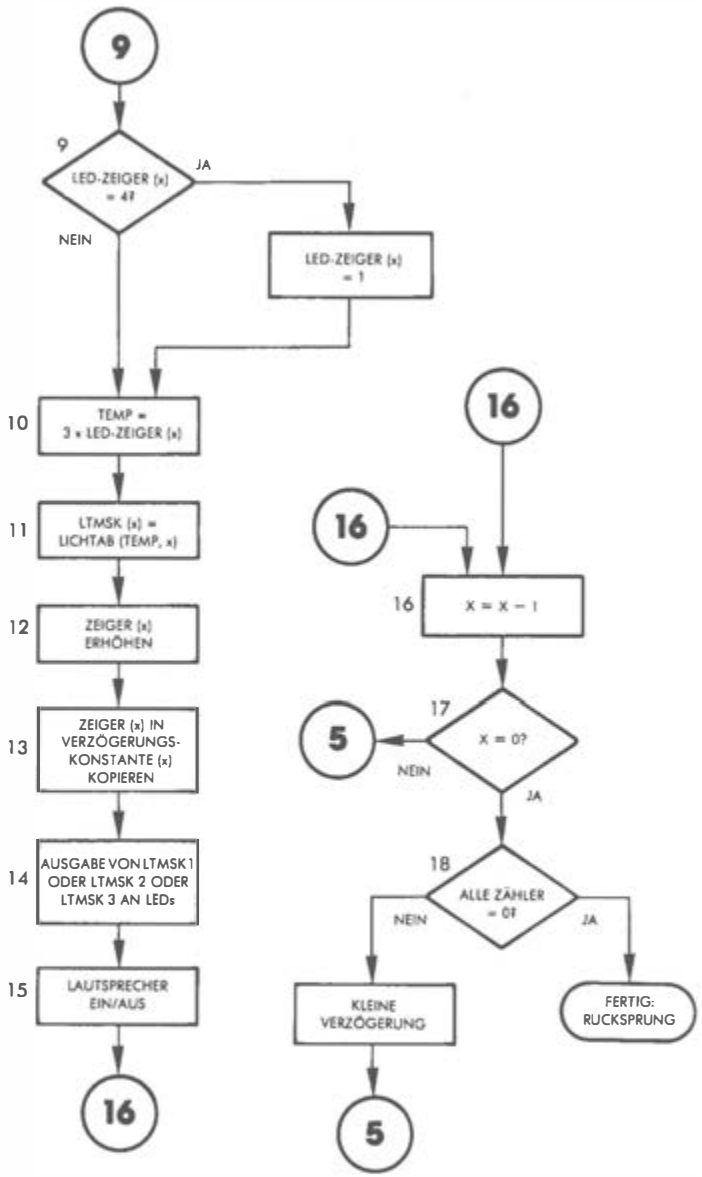



*Abb. 7.4: Flußdiagramm DISPLAY (Fortsetzung)*

Gehen wir noch einmal zum Testschritt 5 im Flußdiagramm:

5. Immer wenn der Zählerwert einer Spalte 0 wird, ist die LED-Bewegung hier zum Stillstand gekommen, und nichts weiter geschieht. Im Flußdiagramm deutet dies der Pfeil rechts vom Entscheidungskästchen bei 5 an: Die Verzweigung geht nach 16, und der Spaltenzeiger wird dekrementiert. In der Spalte, deren Zeiger 0 wurde, ändert sich also nicht.

Als nächstes muß ein Auswertungsalgorithmus die Situation beurteilen, die nach dem Stillstand der Räder vorliegt, und das Ergebnis muß dem Spieler mitgeteilt werden. Wenden wir unsere Aufmerksamkeit also diesem Programmteil zu:

**Die Auswertung**

Das Flußdiagramm des Algorithmus AUSWTG ist in Bild 7.5 dargestellt. Die Vorgehensweise illustriert Bild 7.6, wo die neun LEDs und die damit verknüpften Einheiten erkennbar sind: X als Zeilenzeiger und Y als Spalten-, also als Radzeiger. Mit jeder Zeile ist ein Wertezähler assoziiert, der die Anzahl der erleuchteten LEDs in dieser Zeile ausweist. Dieser Zähler wird nach bestimmten Regeln für jede Zeile in eine Punktzahl verwandelt. Bisher hatten wir ja lediglich Zeile 2 betrachtet und eine Gewinnposition so definiert, daß zwei oder drei LEDs in dieser Zeile erleuchtet sein müssen. Es sind jedoch auch eine ganze Reihe anderer Kombinationen möglich und durch diesen Mechanismus zugelassen. Weiter unten sind Übungsaufgaben für andere Gewinnmuster vorgeschlagen. Die Gesamtsumme aus den Ergebnissen aller Zeilen wird in PUNKTE zusammengetragen (rechts unten in Bild 7.6).

Konzentrieren wir uns nun auf das Flußdiagramm in Bild 7.5. Der Spalten- oder Radzeiger Y zeigt anfangs auf das rechte äußere Rad: Y = 3.

2. Die Zwischenzähler werden auf den Anfangswert 0 gesetzt.
3. Innerhalb der aktuellen Spalte (also 3) müssen wir nur die Zeile berücksichtigen, die eine leuchtende LED enthält. Auf diese Zeile zeigt LEDZEIGR. Die Zeilennummer wird gespeichert in X = LEDZEIGR(Y)
4. Da in der von X angezeigten Zeile eine LED an ist, wird der entsprechende Wertezähler um 1 erhöht.

Wenn wir die LED-Situation in Bild 7.7 annehmen, wird der Wertezähler von Zeile 2 auf den Wert 1 gesetzt.

5. Die nächste Spalte wird untersucht: Y = Y − 1.

Ist Y nicht 0, geht es zurück zu 3., andernfalls kann der Auswertungsprozeß in die nächste Phase gehen.

**Übung 7-1:** *Ermitteln Sie die Endwerte in den Wertezählern, wenn der Test bei (6) im Flußdiagramm Bild 7.5 schließlich abgeschlossen ist. Benutzen Sie dazu das Flußdiagramm und das Beispiel 7.7.*

Die Zahl der erleuchteten LEDs in jeder Reihe muß nun also in eine Punktzahl umgewandelt werden. Hierzu wird die PKTAB-Tabelle benutzt. Werden die Punktvergaberegeln dieser Tabelle geändert, ergibt sich im übrigen ein völlig anderer Spielverlauf.

Die Punktetabelle enthält in jeder Zeile vier Bytes, wobei jede Bytezahl sich jeweils auf die Punkte bezieht, die der Spieler für 0, 1, 2 oder 3 erleuchtete LEDs in dieser Reihe erhält. Bild 7.8 zeigt die logische Organisation der Tabelle. Die Tabelleneintragungen beziehen sich auf die Punktwertung, die zu Beginn dieses Kapitels für dieses Spiel ausgewählt worden ist: In den Zeilen 1 und 3 zählt jede LED-Kombination 0 Punkte; in Zeile 2 zählt jede Zweierkombination 1 Punkt, drei erleuchtete LEDs dagegen 3 Punkte. Praktisch heißt das, daß der Punktwert von Reihe 1 durch einfachen inidzierten Zugriff erhalten wird, als Index dient die Zahl der erleuchteten LEDs; in Zeile 2 kommt eine Verschiebung von 4 hinzu; in Zeile 3 eine weitere Verschiebung von 4. Mathematisch ausgedrückt, heißt das:

$$\text{PUNKTE} = \text{PKTAB}((X - 1) \times 4 + 1 + Y)$$

Hierin ist X die Zeilennummer und Y die Zahl der in dieser Zeile erleuchteten LEDs. Da diese Vorgehensweise es jeder Zeile möglich macht, Punkte zu erzielen, muß das Programm auch die Wertezähler aller Reihen bei der Berechnung der Gesamtpunktzahl berücksichtigen. Dies tun die Schritte 7 und 8: der Zeilenzeiger wird auf 3 initialisiert, und der PKTAB-Versetzungszeiger ergibt sich zu:

$$\text{TEMP} = (X - 1) \times 4 + 1$$

9. Jetzt wird aus der Tabelle der Punktwert entnommen:

   Q = PKTAB(Wertezähler(X),TEMP)

Den Punktwert dieser Zeile erhalten wir durch indiziertes Ansprechen der Tabelle, als Index dient die Zahl der erleuchteten LEDs, die im Wertezähler für diese Reihe enthalten ist, vermehrt um die Versetzung TEMP. Der Punkte-Zwischenstand ist jeweils der bisher aufgelaufene Wert plus diesem Teilwert:

10. PKTEMP = PKTEMP + Q
11. Schließlich wird die Zeilennummer dekrementiert, und der Vorgang wird wiederholt, bis X den Wert 0 erreicht.
12. Erreicht X den Wert 0, so ist die Gesamtpunktzahl für diesen Durchlauf ermittelt und in PKTEMP gespeichert.
13. Die eben erhaltene Punktzahl in PKTEMP wird nun vom Programm geprüft, wobei es zwei mögliche Ergebnisse gibt: Ist PKTEMP 0, findet eine Verzweigung nach 20 statt, wo die Gesamtpunktzahl um 1 vermindert wird. Ist PKTEMP nicht 0, so wird das Gesamtkonto um den Gewinn bei diesem Durchlauf vermehrt, das ist genau PKTEMP.
    Verfolgen wir diesen Fall zuerst.

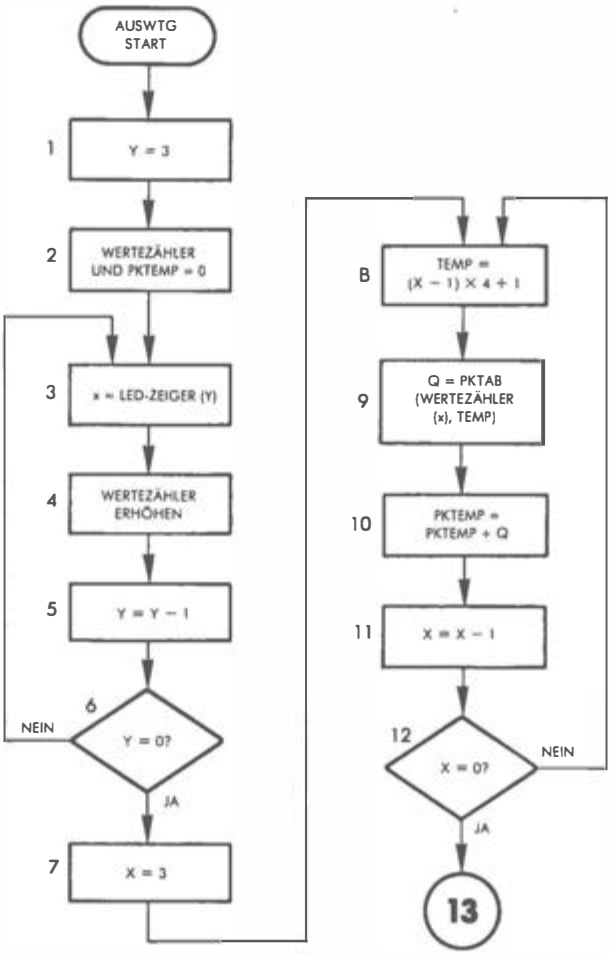



*Abb. 7.5: Flußdiagramm AUSWTG*

14. Die Gesamtpunktzahl wird um 1 erhöht.
15. Es wird auf den Maximalwert 16 hin überprüft.
16. Ist dieses Maximum in Schritt 15 erreicht worden, wird der Spieler audiovisuell darüber informiert, und ein neues Spiel kann beginnen.
17. Wurde der vorige Schritt 16 von 15 aus nicht erreicht, wird der neue Punktestand dem Spieler bekanntgegeben, begleitet von einem hohen Tonsignal.
18. Die Anzahl von Malen, die der Gesamtpunktstand erhöht werden muß, nämlich PKTEMP, wird um 1 erniedrigt.

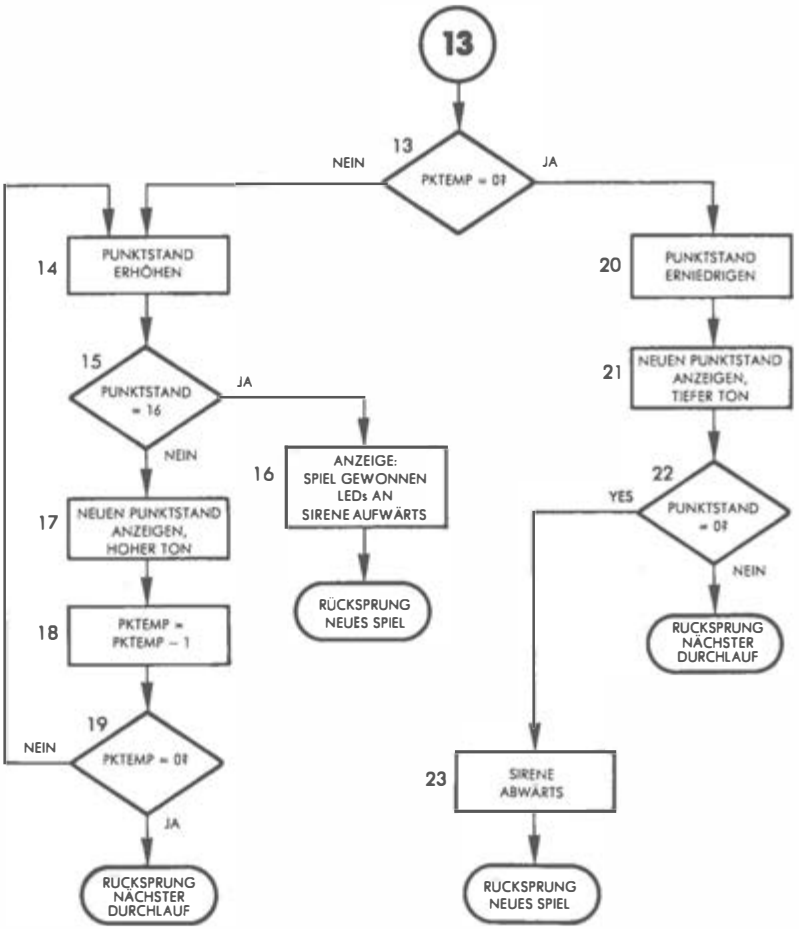



*Abb. 7.5: Flußdiagramm AUSWTG (Fortsetzung)*

19. Ist PKTEMP nicht 0, muß weiter der Gesamtpunktstand erhöht werden, die Verzweigung geht nach 14. Im andern Fall kann der Spieler den nächsten Durchlauf starten.

Folgen wir nun dem anderen Pfad von Position 14 im Flußdiagramm aus, wo die Gesamtpunktzahl getestet wurde:

20. Es gab 0 Punkte für diesen Durchlauf: Der Gesamtpunktstand wird um 1 erniedrigt.

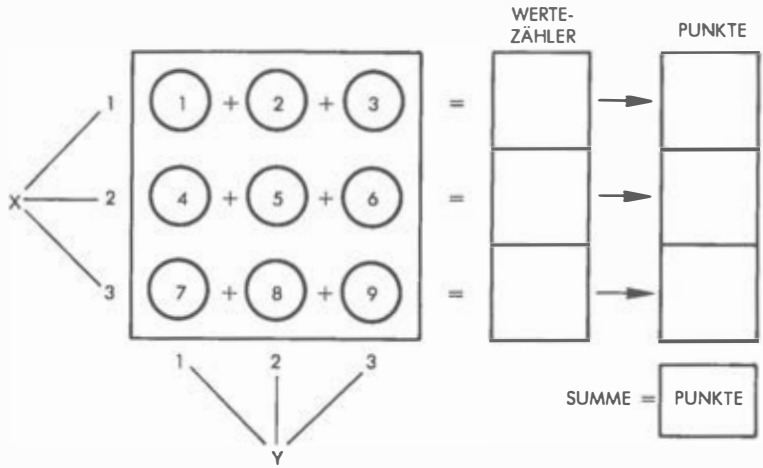



*Abb. 7.6: Auswertung des Brettes*

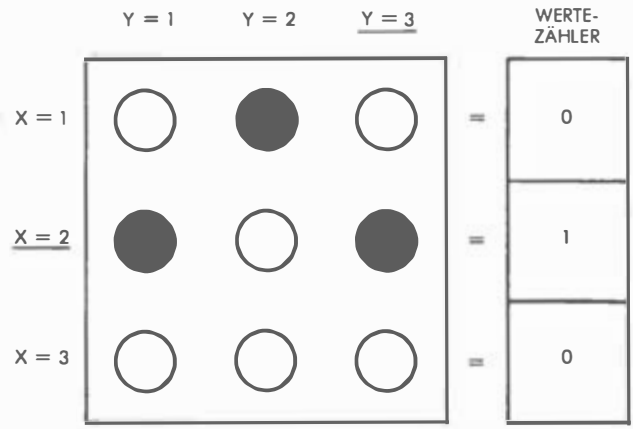



*Abb. 7.7: Auswertungsbeispiel*

21. Der Spieler erfährt den neuen Punktstand, begleitet von einem tiefen Ton.
22. Der neue Punktstand wird auf den Minimalwert 0 getestet: Ist er erreicht, hat der Spieler verloren, andernfalls kann er weiterspielen.
23. Ein absteigender Sirenenton wird erzeugt, um die Niederlage akustisch zu untermalen, und das Spiel ist zu Ende.

| 0 | 1 | 2 | 3 | ANZAHL ERLEUCHTETER LEDs |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | REIHE 1 |
| 0 | 0 | 1 | 3 | REIHE 2 |
| 0 | 0 | 0 | 0 | REIHE 3 |

*Abb. 7.8: Punktetabelle*

## DAS PROGRAMM

### Datenstrukturen

Das Programm benutzt zwei Tabellen. Die eine ist die Punktetabelle, mit der eine Punktzahl aus der Zahl der erleuchteten LEDs in jeder Zeile errechnet wird; sie wurde soeben beschrieben. Die andere ist LEDTAB, mit Hilfe derer der richtige Code zur Erleuchtung einer spezifischen LED im Ein/Ausgabe-Tor erzeugt wird. Jeder Eintrag in dieser Tabelle ist ein spezifisches, zum Ein/Ausgabe-Tor zu übertragendes LED-Muster.

In vertikaler Richtung entsprechen die Tabellenwerte der 1., 2. und 3. LED-Spalte, im Programm beziehen sich somit z.B. die Zahlen in den Adressen 1A – 1C auf die erste LED-Spalte. Die Hexadezimalzahl 40 (64 dezimal) in 1C korrespondiert also zur dritten LED in Spalte 1 des Spielbrettes, das ist LED 7.

### Zero-page-Variable

In der Nullseite sind die folgenden Variablen gespeichert:

- TEMP Arbeitsspeicher
- PKTEMP Zwischenspeicher für Gewinn- oder Verlustpunkte während eines Durchlaufs
- PUNKTE Gesamtpunktstand
- DAUER und FREQ sind die üblichen Tonerzeugungskonstanten
- SPEEDS (3 Adressen) Umdrehungsgeschwindigkeiten in den 3 Spalten
- INDX (3 Adressen) Verzögerungszähler für die LED-Rotationen
- INCR (3 Adressen) Zeiger auf die LED-Positionen in jeder Spalte für den Zugriff auf die Tabellen
- LTMSK (3 Adressen) Muster für erleuchtete LEDs
- WERTE (3 Adressen) Zahl erleuchteter LEDs in jeder Spalte
- RND (6 Adressen) Arbeitsspeicher des Zufallszahlengenerators

```
                ; SPIELAUTOMAT -SIMULATIONSPROGRAMM
                ;DRUCKEN IRGENDEINER TASTE SETZT 'RÄDER' IN BEWEGUNG.
                ;PUNKTAUSWERTUNG ANHAND DER TABELLE 'PKTAB'.
                ; STARTKAPITAL SIND 8 PUNKTE. DER 'EINSATZ' IST 1 PUNKT.
                ;
                TEMP      = $0000       ;ZWISCHENSPEICHER
                PKTEMP    = $0001       ;PUNKTE-ZWISCHENSPEICHER
                PUNKTE    = $0002       ;PUNKTSTAND
                DAUER     = $0003       ;TONDAUERKONSTANTE
                FREQ      = $0004       ;FREQUENZKONSTANTE
                SPEEDS    = $0005       ;LED-DREHGESCHWINDIGKEITEN (3 ADRESSEN)
                INDX      = $0008       ;LED-VERZÖGERUNGSZÄHLER (3ADRESSEN)




                ; SPIELAUTOMAT'-SIMULATIONSPROGRAMM
                ;DRUCKEN IRGENDEINER TASTE SETZT 'RÄDER' IN BEWEGUNG.
                ;PUNKTAUSWERTUNG ANHAND DER TABELLE 'PKTAB'.
                ; STARTKAPITAL SIND 8 PUNKTE. DER 'EINSATZ' IST 1 PUNKT.
                ;
                TEMP      = $0000       ;ZWISCHENSPEICHER
                PKTEMP    = $0001       ;PUNKTE-ZWISCHENSPEICHER
                PUNKTE    = $0002       ;PUNKTSTAND
                DAUER     = $0003       ;TONDAUERKONSTANTE
                FREQ      = $0004       ;FREQUENZKONSTANTE
                SPEEDS    = $0005       ;LED-DREHGESCHWINDIGKEITEN (3 ADRESSEN)
                INDX      = $0008       ;LED-VERZÖGERUNGSZÄHLER (3 ADRESSEN)
                INCR      = $000B       ;LED-POSITIONSZEIGER ZUM HERAUSGREIFEN
                                        ;DER MUSTER AUS DEN TABELLEN (3 ADR.)
                LTMSK     = $000E       ;MUSTER DER ERLEUCHTETEN LEDS (3 ADR.)
                WERTE     = $0011       ;ERLEUCHTETE LEDS PRO 'RAD' (3 ADR.)
                RND       = $0014       ;ARBEITSSPEICHER DES ZUFALLSGENERATORS
                                        ;(6 ADRESSEN)
                TOR1A     = $A001       ;LEDS 1 BIS 7
                TOR1B     = $A000       ;LEDS 8 BIS 15
                TOR3B     = $AC00       ;LAUTSPRECHER
                DDR1A     = $A003       ;DATENRICHTUNGSREGISTER
                DDR1B     = $A002
                DDR3B     = $AC02
                T1CL      = $A004       ;ZÄHLREGISTER ZEITGEBER 1
                ;
                ;LEDMUSTER-TABELLE
                ;DIE ZEILEN DES FELDES ENTSPRECHEN DEN LED-SPALTEN ('RÄDERN')
                ;UND DIE SPALTEN DEN LED-ZEILEN. BEISPIEL: 3. BYTE IN ZEILE 1
                ;ERLEUCHTET LED 7.
                ;
001A: 01        LEDTAB .BYTE 2,16,128
001B: 08
001C: 40
001D: 02               .BYTE 4,32,0
001E: 10
001F: 80
0020: 04               .BYTE 4,32,0
0021: 20
0022: 00

                ;PUNKTETABELLE FÜR ERZIELEN BESTIMMTER LED-STELLUNGEN
                ;TABELLENZEILEN ENTSPRECHEN ERLEUCHTETEN LED-ZEILEN, TABELLEN-
                ;SPALTEN ENTSPRECHEN DER ZAHL ERLEUCHTETER LEDS DIESER SPALTE.
                ;BEISPIEL: 3 LEDS IN DER MITTLEREN REIHE = 3 PUNKTE.
                ;
0023: 00        PKTAB  .BYTE 0,0,0.0
0024: 00
0025: 00
0026: 00
0027: 00               .BYTE 0,0,1,3
0028: 00
0029: 01
002A: 03
002B: 00               .BYTE 0,0,0.0
```

*Abb. 7.9: Programm SPIELAUTOMAT*

```
002C: 00
002D: 00
002E: 00
                    ;
                    ;HAUPTPROGRAMM
                    ;
                    GETKEY      = $100
                                * = $200
0200: A9 FF                     LDA #$FF        ;TORE SETZEN
0202: 8D 03 A0                  STA DDR1A
0205: 8D 02 A0                  STA DDR1B
0208: 8D 02 AC                  STA DDR3B
020B: AD 04 A0                  LDA TICL        ;STARTZAHL FüR ZUFALLSZAHLENGENERATOR
020E: 85 15                     STA RND+1
0210: A9 08         START       LDA #8          ;STARTPUNKTZAHL = 8
0212: 85 02                     STA PUNKTE
0214: A8                        TAY             ;STARTPUNKTZAHL ANZEIGEN
0215: 20 3D 03                  JSR LICHT
0218: 20 00 01      TASTE       JSR GETKEY      ;BELIEBIGE TASTE FüR PROGRAMMSTART
021B: 20 27 02                  JSR DISPLY      ;RäDER DREHEN SICH
021E: 20 A7 02                  JSR AUSWTG      ;PUNKTSTAND BERECHNEN UND ANZEIGEN
0221: A5 02                     LDA PUNKTE
0223: D0 F3                     BNE TASTE       ;BEI PUNKTZAHL<>0, NäCHSTER DURCHLAUF
0225: F0 E9                     BEQ START       ;BEI PUNKTZAHL=0, NEUBEGINN
                    ;
                    ;UNTERPROGRAMM 'DISPLY'
                    ;'LAUFENDE RäDER' DARSTELLEN UND ERGEBNIS ERMITTELN
                    ;
                    LOLIM       = 90
                    HILIM       = 135
                    SPDPRM      = 80
0227: A9 00         DISPLY      LDA #0          ;ZEIGER ZURüCKSETZEN
0229: 85 0B                     STA INCR
022B: 85 0C                     STA INCR+1
022D: 85 0D                     STA INCR+2
022F: A0 02         LDRND       LDY #2          ;INDEX FüR 3 SCHRITTE
0231: 20 80 03      GETRND      JSR RANDOM      ;ZUFALLSZAHL HOLEN
0234: C9 87                     CMP #HILIM      ;ZU GROSS?
0236: B0 F9                     BCS GETRND      ;WENN JA: NEUE ZAHL
0238: C9 5A                     CMP #LOLIM      ;ZU KLEIN?
023A: 90 F5                     BCC GETRND      ;WENN JA: NEUE ZAHL
023C: 99 0B 00                  STA INDX,Y      ;INDIZES UND GESCHWINDIGKEITSZäHLER AB-
023F: 99 05 00                  STA SPEEDS,Y    ;SPEICHERN
0242: 88                        DEY
0243: 10 EC                     BPL GETRND      ;NäCHSTE ZUFALLSZAHL
0245: A2 02         UPDATE      LDX #2          ;INDEX FüR 2 SCHRITTE SETZEN
0247: B4 05         UPDTLP      LDY SPEEDS,X    ;GESCHWINDIGKEIT(X)=0?
0249: F0 44                     BEQ NXTUPD      ;WENN JA: NäCHSTE SPALTE AUF NEUEN STAND
024B: D6 0B                     DEC INDX,X      ;SCHLEIFENINDEX(X) DEKREMENTIEREN
024D: D0 40                     BNE NXTUPD      ;WENN SCHLEIFENINDEX<>0: AKTUALISIEREN
024F: B4 0B                     LDY INCR,X      ;ZEIGER(X) ERHöHEN
0251: C8                        INY
0252: C0 03                     CPY #3          ;ZEIGER = 3?
0254: D0 02                     BNE NORST       ;WENN NICHT: ZEIGERRüCKSETZUNG üBER-
0256: A0 00                     LDY #0          ;SPRINGEN
0258: 94 0B         NORST       STY INCR,X      ;ZEIGER(X) ZURüCKSETZEN
025A: 86 00                     STX TEMP        ;MULTIPLIKATION MIT 3 ZWECKS ZUGRIFF AUF
025C: 8A                        TXA             ;RICHTIGE TABELLENPOSITION
025D: 0A                        ASL A
025E: 18                        CLC
025F: 65 00                     ADC TEMP
0261: 75 0B                     ADC INCR,X      ;SPALTEN-NR. + ZEILENZEIGER
0263: A8                        TAY             ;üBERTRAGEN FüR INDIZIERUNG
0264: B9 1A 00                  LDA LEDTAB,Y    ;LEDMUSTER HOLEN
0267: 95 0E                     STA LTMSK,X     ;IN MASKE(X) ABSPEICHERN
0269: B4 05         SPDUPD      LDY SPEEDS,X    ;GESCHWINDIGKEITSKONSTANTE ERHöHEN
026B: C8                        INY
026C: 94 05                     STY SPEEDS,X    ;UND ABSPEICHERN
026E: 94 08                     STY INDX,X      ;INDEX JUSTIEREN
0270: A9 00         LEDUPD      LDA #0          ;LICHTER AKTUALISIEREN
0272: 8D 00 A0                  STA TOR1B       ;SONDERBEHANDLUNG LED 9
0275: A5 10                     LDA LTMSK+2     ;MUSTER FüR AUSGABE KOMBINIEREN
```

*Abb. 7.9: Programm SPIELAUTOMAT (Fortsetzung)*

```
0277: D0 07             BNE AUSLD9      ;WENN MASKE 3 <> 0: LED 9 AUS
0279: A9 01             LDA #1          ;LED 9 AN
027B: 8D 00 A0          STA TOR1B
027E: A9 00             LDA #0          ;A ZURÜCKSETZEN FÜR RICHTIGES MUSTER
0280: 05 0E     AUSLD9  ORA LTMSK       ;RESTLICHE MUSTER KOMBINIEREN
0282: 05 0F             ORA LTMSK+1
0284: 8D 01 A0          STA TOR1A       ;LICHTER SETZEN
0287: AD 00 AC          LDA TOR3B       ;LAUTSPRECHER IN BETRIEB SETZEN
028A: 49 FF             EOR #$FF
028C: 8D 00 AC          STA TOR3B
028F: CA        NXTUPD  DEX             ;X FÜR NÄCHSTEN SCHRITT DEKREMENTIEREN
0290: 10 B5             BPL UPDTLP      ;WENN X=>0: NÄCHSTE AKTUALISIERUNG
0292: A0 50             LDY SPDPRM      ;KLEINE VERZÖGERUNG
0294: 88        WARTEN  DEY             ;LEDS KURZ AN
0295: D0 FD             BNE WARTEN
0297: A5 05             LDA SPEEDS      ;PRÜFEN, OB ALLE RÄDER STEHEN
0299: 05 06             ORA SPEEDS+1
029B: 05 07             ORA SPEEDS+2
029D: D0 A6             BNE UPDATE      ;WENN NICHT: NÄCHSTE AKTUALISIERUNG
029F: A9 FF             LDA #$FF
02A1: 85 03             STA DAUER       ;ANZEIGE ZUM LESEN KURZ ANHALTEN
02A3: 20 30 03          JSR DELAY
02A6: 60                RTS             ;RÄDER STEHEN: FERTIG
                ;
                ;UNTERPROGRAMM ZUR AUSWERTUNG EINES DURCHLAUFS UND AUDIOVISUELLE
                ;ANZEIGE DES ERGEBNISSES (GEWONNEN ODER VERLOREN)
                ;
                HITON   = $20
                LOTON   = $F0
02A7: A9 00     AUSWTG  LDA #0          ;VARIABLE ZURÜCKSETZEN
02A9: 85 11             STA WERTE
02AB: 85 12             STA WERTE+1
02AD: 85 13             STA WERTE+2
02AF: 85 01             STA PKTEMP
02B1: A0 02             LDY #2          ;ERLEUCHTETE LEDS/REIHE 3 MAL ZÄHLEN
02B3: B6 0B     CNTLP   LDX INCR,Y      ;AUFSUMMIEREN DER ERLEUCHTETEN LEDS
02B5: F6 11             INC WERTE,X     ;LEDZAHL/REIHE ERHÖHEN
02B7: 88                DEY
02B8: 10 F9             BPL CNTLP       ;WENN NICHT FERTIG: NOCHMAL
02BA: A2 02             LDX #2          ;PUNKTE IN 3 SCHRITTEN BERECHNEN
02BC: 8A        PKTSCHL TXA             ;MULTIPLIZIEREN M. 4 FÜR TABELLENZUGRIFF
02BD: 0A                ASL A
02BE: 0A                ASL A
02BF: 18                CLC             ;LEDS/REIHE(X) ADDIEREN: ERGIBT SPALTEN-
02C0: 75 11             ADC WERTE,X     ;ADRESSE FÜR TABELLENZUGRIFF
02C2: A8                TAY             ;IN ZEIGER ÜBERTRAGEN
02C3: B9 23 00          LDA PKTAB,Y     ;PUNKTE F. DIESEN DURCHLAUF ABLESEN
02C6: 18                CLC
02C7: 65 01             ADC PKTEMP      ;ZUM BISHER AUFSUMMIERTEN HINZUFÜGEN
02C9: 85 01             STA PKTEMP      ;UND WIEDER ABSPEICHERN
02CB: CA                DEX
02CC: 10 EE             BPL PKTSCHL     ;SCHLEIFE WENN NICHT FERTIG
02CE: A9 60             LDA #$60        ;TONDAUERKONSTANTE SETZEN
02D0: 85 03             STA DAUER
02D2: A5 01             LDA PKTEMP      ;PUNKTE FÜR DIESEN DURCHLAUF HOLEN
02D4: F0 34             BEQ VERLUST     ;BEI 0: PUNKTEKONTO ERNIEDRIGEN
02D6: E6 02     GEWINN  INC PUNKTE      ;PUNKTEKONTO ERHÖHEN
02D8: A4 02             LDY PUNKTE      ;PUNKTEKONTO HOLEN
02DA: C0 10             CPY #16         ;PUNKTEKONTO 16?
02DC: F0 10             BEQ GEWENDE     ;WENN JA: SIEG-ROUTINE ANSPRINGEN
02DE: 20 3D 03          JSR LICHT       ;PUNKTEKONTO ANZEIGEN
02E1: A9 20             LDA #HITON      ;HÖHER TON
02E3: 20 64 03          JSR TON
02E6: 20 30 03          JSR DELAY       ;KURZE VERZÖGERUNG
02E9: C6 01             DEC PKTEMP      ;ZU ADDIERENDE PUNKTE DEKREMENTIEREN
02EB: D0 E9             BNE GEWINN      ;WENN NICHT FERTIG: NOCHMAL
02ED: 60                RTS             ;FERTIG: ZURÜCK ZUM HAUPTPROGRAMM
02EE: A9 FF     GEWENDE LDA #$FF        ;SIEGSIGNAL: ALLE LEDS AN
02F0: 8D 01 A0          STA TOR1A
02F3: 8D 00 A0          STA TOR1B
02F6: 85 00             STA TEMP        ;FREQUENZPARAMETER SETZEN
02FB: A9 00             LDA #0
```

*Abb. 7.9: Programm SPIELAUTOMAT (Fortsetzung)*

```
02FA: 85 02              STA PUNKTE     ;SIGNAL FÜR NEUBEGINN
02FC: A9 04              LDA #4
02FE: 85 03              STA DAUER      ;EINZELTONVERZÖGERUNG IN TONLEITER
0300: A5 00     AUF      LDA TEMP       ;FREQUENZ FÜR TON HOLEN
0302: 20 64 03           JSR TON        ;UND TON SPIELEN
0305: C6 00              DEC TEMP       ;NÄCHSTER TON ETWAS HÖHER
0307: D0 F7              BNE AUF        ;WENN NICHT FERTIG: NÄCHSTER TON
0309: 60                 RTS            ;RÜCKSPRUNG FÜR NEUBEGINN
030A: C6 02     VERLUST  DEC PUNKTE     ;SCHLECHTER DURCHLAUF: PUNKTE=PUNKTE-1
030C: A4 02              LDY PUNKTE     ;PUNKTE ANZEIGEN
030E: 20 3D 03           JSR LICHT
0311: A9 F0              LDA #LOTON     ;TIEFER VERLUSTTON
0313: 20 64 03           JSR TON
0316: A4 02              LDY PUNKTE     ;KONTO AUF SPIELENDE PRÜFEN
0318: F0 01              BEQ VERLENDE
031A: 60                 RTS            ;WENN SPIEL NICHT ZUENDE: RÜCKSPRUNG
031B: A9 00     VERLENDE LDA #0         ;FREQUENZPARAMETER FÜR FALLENDE TON-
031D: 85 00              STA TEMP       ;LEITER ZWISCHENSPEICHERN
031F: BD 01 A0           STA TOR1A      ;LED 1 LÖSCHEN
0322: A9 04              LDA #4
0324: 85 03              STA DAUER
0326: A5 00     AB       LDA TEMP
0328: 20 64 03           JSR TON        ;TON SPIELEN
032B: E6 00              INC TEMP       ;NÄCHSTER TON ETWAS TIEFER
032D: D0 F7              BNE AB
032F: 60                 RTS            ;RÜCKSPRUNG ZUM NEUBEGINN
                ;
                ;UNTERPROGRAMM FÜR VARIABLE VERZÖGERUNG.
                ;VERZÖGERUNG = (2046*(DAUER)+10)
                ;
0330: A4 03     DELAY    LDY DAUER      ;VERZÖGERUNGSLÄNGE EINLESEN
0332: A2 FF     DL1      LDX #$FF       ;INNENSCHLEIFENVERZÖGERUNG 2040 US
0334: D0 00     DL2      BNE +2         ;'ZEIT VERSCHWENDEN'
0336: CA                 DEX            ;INNENSCHLEIFENZÄHLER
0337: D0 FB              BNE DL2        ;INNENSCHLEIFE ZUENDE ZÄHLEN
0339: 88                 DEY            ;AUSSENSCHLEIFENZÄHLER
033A: D0 F6              BNE DL1        ;AUSSENSCHLEIFE ZUENDE ZÄHLEN
033C: 60                 RTS            ;RÜCKSPRUNG
                ;
                ;UNTERPROGRAMM ZUM ERLEUCHTEN DER IN Y MITGEBRACHTEN LED
                ;
033D: A9 00     LICHT    LDA #0         ;A FÜR SCHIEBEVORGANG LÖSCHEN
033F: 85 00              STA TEMP       ;ÜBERLAUF LÖSCHEN
0341: 8D 01 A0           STA TOR1A      ;UNTERE LEDS LÖSCHEN
0344: 8D 00 A0           STA TOR1B      ;OBERE LEDS LÖSCHEN
0347: C0 0F              CPY #15        ;KODE FÜR UNVERBUNDENES BIT?
0349: F0 01              BEQ +3         ;WENN JA: KEINE ÄNDERUNG
034B: 88                 DEY            ;PASSEND MACHEN
034C: 38                 SEC            ;ZU VERSCHIEBENDES BIT SETZEN
034D: 2A        LTSHFT   ROL A          ;BIT NACH LINKS VERSCHIEBEN
034E: 90 05              BCC LTCC       ;BEI CARRY=1: ÜBERLAUF IN OBERES
0350: A2 FF              LDX #$FF       ;ÜBERLAUFFLAGGE SETZEN
0352: 86 00              STX TEMP
0354: 2A                 ROL A          ;BIT AUS CARRY SCHIEBEN
0355: 88        LTCC     DEY            ;EIN BIT WENIGER ZU VERSCHIEBEN
0356: 10 F5              BPL LTSHFT     ;WENN NICHT FERTIG: WEITERSCHIEBEN
0358: A6 00              LDX TEMP       ;ÜBERLAUFFLAGGE HOLEN
035A: D0 04              BNE HIBYTE     ;BEI ÜBERLAUF: HIGH BYTE IN A
035C: BD 01 A0  LOBYTE   STA TOR1A      ;A IN UNTERE LEDS
035F: 60                 RTS            ;...UND RÜCKSPRUNG
0360: BD 00 A0  HIBYTE   STA TOR1B      ;A IN OBERE LEDS
0363: 60                 RTS
                ;
                ;UNTERPROGRAMM ZUR TONERZEUGUNG
                ;
0364: 85 04     TON      STA FREQ
0366: A9 FF              LDA #$FF
0368: BD 00 AC           STA TOR3B
036B: A9 00              LDA #0
036D: A6 03              LDX DAUER
036F: A4 04     FL2      LDY FREQ
0371: 88        FL1      DEY
```

*Abb. 7.9: Programm SPIELAUTOMAT (Fortsetzung)*

```
0372: 18                CLC
0373: 90 00             BCC +2
0375: D0 FA             BNE FL1
0377: 49 FF             EOR #$FF
0379: 8D 00 AC          STA TOR3B
037C: CA                DEX
037D: D0 F0             BNE FL2
037F: 60                RTS
                ;
                ;UNTERPROGRAMM ZUR ERZEUGUNG VON ZUFALLSZAHLEN
                ;
0380: 38        RANDOM  SEC
0381: A5 15             LDA RND+1
0383: 65 18             ADC RND+4
0385: 65 19             ADC RND+5
0387: 85 14             STA RND
0389: A2 04             LDX #4
038B: B5 14     RNDSH   LDA RND,X
038D: 95 15             STA RND+1,X
038F: CA                DEX
0390: 10 F9             BPL RNDSH
0392: 60                RTS


SYMBOLTABELLE:
CNTLP   02B3    DDR1A   A003    DDR1B   A002    DDR3B   AC02
DELAY   0330    DISPLY  0227    DL1     0332    DL2     0334
DAUER   0003    AUSWTG  02A7    AB      0326    FL1     0371
FL2     036F    FREQ    0004    GETKEY  0100    GETRND  0231
HIBYTE  0360    HILIM   0087    HITON   0020    INCR    0008
INDX    0008    TASTE   0218    LDRND   022F    LEDUPD  0270
LICHT   033D    LOBYTE  035C    LOLIM   005A    VERLUST 030A
VERLENDE 031B   LOTON   00F0    LEDTAB  001A    LTCC    0355
LTMSK   000E    LTSHFT  034D    NORST   0258    NXTUPD  028F
AUSLD9  0280    TOR1A   A001    TOR1B   A000    TOR3B   AC00
RANDOM  0380    AUF     0300    RND     0014    RNDSH   038B
PUNKTE  0002    PKTSCHL 02BC    PKTAB   0023    PKTEMP  0001
SPDPRM  0050    SPDUPD  0269    SPEEDS  0005    START   0210
TICL    A004    TEMP    0000    TON     0364    UPDATE  0245
UPDTLP  0247    WERTE   0011    WARTEN  0294    GEWINN  02D6
GEWENDE 02EE
```

*Abb. 7.9: Programm SPIELAUTOMAT (Fortsetzung)*

### Implementierung des Programms

Das Programm besteht aus einem Hauptteil und zwei Hauptunterprogrammen, DISPLY und AUSWTG, dazu kommen einige Dienstleistungsroutinen: DELAY (variable Verzögerung), LICHT (Erleuchtung bestimmter LEDs), TON (Tonerzeugung) und RANDOM (Zufallszahlenerzeugung).

Das Hauptprogramm ist ab Speicherstelle 200 gespeichert. Wie gewöhnlich werden zunächst die drei Datenrichtungsregister für die Tore A und B von VIA 1 und Tor B von VIA 3 auf Ausgabe gesetzt:

```
LDA #$FF
STA DDR1A
STA DDR1B
STA DDR3B
```

Wie in vorangegangenen Kapiteln wird das Zählregister des Zeitgebers 1 zur Erzeugung einer Startzahl für den Zufallszahlengenerator benutzt. Diese „Urzufallszahl" wird in Adresse RND + 1 abgespeichert, worauf das Zufalls-Unterprogramm dann später immer wieder zurückgreift:

```
        LDA T1CL
        STA RND +1
```

Bei Beginn eines neuen Spiels wird das Punktekonto auf den Wert 8 gesetzt

```
START   LDA #8
        STA PUNKTE
```

und anschließend ausgegeben:

```
        TAY             Übergabe an Y
        JSR LICHT
```

Das Unterprogramm LICHT zeigt den Punktstand an, indem die dem Wert im Y-Register entsprechende LED angemacht wird. Die genaue Beschreibung folgt weiter unten.

Jetzt ist das Spielautomatenprogramm bereit, auf eine Tasteneingabe des Spielers zu reagieren:

```
TASTE   JSR GETKEY
```

Sobald eine Taste gedrückt ist, beginnen sich die Räder zu drehen:

```
        JSR DISPLY
```

Wenn die Räder stehengeblieben sind, erfolgt die Punktauswertung, das Ergebnis wird angezeigt und „musikalisch untermalt":

```
        JSR AUSWTG
```

Ist die Endpunktzahl ungleich 0, beginnt der Prozeß von neuem,

```
        LDA PUNKTE
        BNE TASTE
```

und der Spieler kann die Räder wieder in Bewegung setzen. Bei Punktzahl 0 dagegen beginnt ein neues Spiel:

```
        BEQ START
```

Damit ist das Hauptprogramm abgeschlossen. Seine Einfachheit und Kürze beruht auf der Strukturierung durch Unterprogramme.

**Die Unterprogramme**

Die Algorithmen der beiden Unterprogramme DISPLY und AUSWTG wurden im letzten Abschnitt beschrieben. Betrachten wir nun die Implementierung dieser Programme.

***Unterprogramm DISPLY***

Drei wesentliche Unterprogrammparameter sind LOLIM, HILIM und SPDPRM, und die Veränderung dieser Parameter hat unterschiedliche Auswirkungen. So resultiert z.B. ein kleinerer LOLIM-Wert in einer längeren Rotationszeit der LEDs. Man könnte auch die Werte so wählen, daß fast jeder Durchlauf ein Gewinnmuster ergibt. Wir haben folgende Werte gewählt: LOLIM = 90, HILIM = 134, SPDPRM = 80 (dezimal).

Adresse INCR enthält einen Zeiger auf die aktuelle LED-Position, der später dazu benutzt wird, das geeignete Bitmuster aus der Tabelle zu holen. Die möglichen Werte 0, 1 und 2 zeigen auf die LED-Positionen 1, 2 und 3. Die drei Zeiger auf die LEDs in jeder Spalte befinden sich in den Adressen INCR, INCR + 1 und INCR + 2. Sie werden auf 0 gestellt:

```
DISPLY    LDA #0
          STA INCR
          STA INCR +1
          STA INCR +2
```

Zu beachten ist, daß in den vorangegangenen Beispielen (etwa Bild 7.7) zur einfacheren Erläuterung die Zeiger X und Y die Werte 1 bis 3 hatten. Die unkompliziertere Indizierung erfolgt jetzt natürlich mit den Werten von 0 bis 2. Der Radzeiger weist auf die Spalte rechts außen:

```
LDRND     LDY #2
```

Das Unterprogramm RANDOM erzeugt die Startzufallszahl:

```
GETRND    JSR RANDOM
```

Die Zahl, die die Routine zurückbringt, wird mit den gesetzten Über- und Untergrenzen verglichen. Paßt sie da nicht hinein, wird der Vorgang so lange wiederholt, bis sie paßt:

```
          CMP #HILIM     Zahl zu groß?
          BCS GETRND     wenn ja: neue Zahl
          CMP #LOLIM     zu klein?
          BCC GETRND     wenn ja: neue Zahl
```

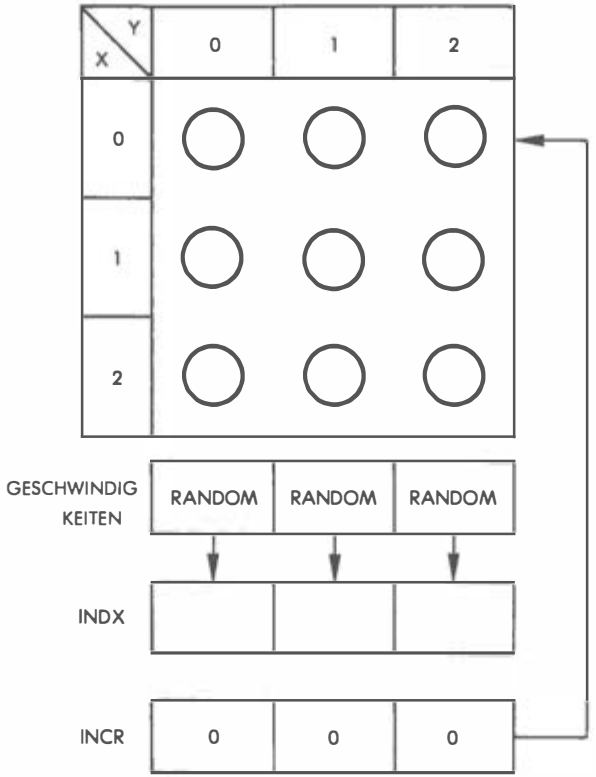



*Abb. 7.10: Die Räder drehen sich*

Die erste zulässige Zufallszahl wandert in die Speicherstellen INDX und SPEEDS:

```
STA INDX,Y
STA SPEEDS,Y
```

Dasselbe geschieht für Spalten 1 und 0:

```
DEY
BPL GETRND        nächste Zufallszahl
```

Haben alle drei Spalten ihre Indizes und Geschwindigkeiten, beginnt eine neue Iterationsschleife, wobei das X-Register als Radzähler fungiert:

```
UPDATE    LDX #2          Zähler für 3 Iterationen setzen
```

Überprüfung auf Geschwindigkeit 0:

```
UPDTLP   LDY SPEEDS,X   Geschwindigkeit (X) = 0?
         BEQ NXTUPD     wenn ja: nächste Spalte auf neuen
                        Stand
```

Solange die Geschwindigkeit nicht 0 ist, muß die nächste LED dieser Spalte aufleuchten und der Verzögerungszähler dekrementiert werden:

```
         DEC INDX,X     Schleifenindex X dekrementieren
```

Erreicht die Verzögerung nicht 0, wird nach NXTUPD verzweigt (siehe weiter unten), andernfalls muß die nächste LED aufleuchten. Bei der Inkrementierung des LED-Zeigers ist das Erreichen des Wertes 3 mit einem zyklischen Rücksprung gekoppelt:

```
         BNE NXTUPD     wenn X ungleich 0: nächste Aktua-
                        lisierung
         LDY INCR,X     Zeiger erhöhen
         INY
         CPY #3         Zeiger auf 3?
         BNE NORST      wenn nicht, überspringen
         LDY #0         zurücksetzen auf 0
NORST    STY INCR,X     Zeiger wiederherstellen
```

Der neue Wert des LED-Zeigers geht zurück in die spaltenspezifische INDX-Adresse (in der UPDATE-Routine zeigt X auf die Spalte). Um die richtige LED zu erleuchten, brauchen wir jetzt aus der LEDTAB-Tabelle das entsprechende Bitmuster. Beachten Sie die prinzipielle Zweidimensionalität des Feldes LEDTAB (wie auch des Feldes PKTAB), also die Zeilen-/Spaltenstruktur, obwohl beide Tabellen im Programm als fortlaufende Zahlenserien erscheinen. Um die Adresse eines bestimmten Feldelementes zu bekommen, muß also die Zeilennummer mit der Spaltenanzahl multipliziert und das Ergebnis zur Spaltennummer addiert werden.

Die Tabelle wird mittels Y-indizierter Adressierung angesprochen. Dazu muß X zunächst mit 3 multipliziert werden, dazu wird dann der Wert des LED-Zeigers in INCR addiert. Die Multiplikation mit 3 wird mit einer Linksverschiebung (Multiplikation mit 2) und anschließender Addition von 1 verwirklicht:

```
         STX TEMP       multipliziere X mit 3
         TXA
         ASL A          Linksverschiebung
         CLC
         ADC TEMP       plus 1
```

der Wert von INCR wird addiert, und das Ergebnis wandert für die indizierte

Adressierung ins Y-Register. Dann kann der entspechende Wert aus LEDTAB entnommen werden:

```
        ADC INCR,X
        TAY
        LDA LEDTAB,Y    LED-Muster holen
```

Das Muster wird nun in eine der drei Adressen ab LTMSK übertragen, in welche, das hängt von der Spalte ab, in der eine LED-Bewegung gerade stattfindet. Angemacht werden die entsprechenden Lichter erst dann, wenn das vollständige Muster für alle drei Spalten bereitsteht. Nach der LED-Bewegung muß schließlich noch die Geschwindigkeitskonstante inkrementiert werden:

```
        STA LTMSK,X
SPDUPD  LDY SPEEDS,X
        INY
        STY SPEEDS,X
```

Der Index wird auf die neue Geschwindigkeit justiert:

```
        STY INDX,X
```

Es ist nun die Spezialbehandlung von LED 9 zu berücksichtigen. Das Muster der ersten acht LEDs ist in LEDTAB gespeichert. Daß LED 9 erleuchtet werden muß, ist leicht daran zu erkennen, daß das Muster von Spalte 3 nur Nullen zeigt. Eine LED in dieser Spalte muß aber leuchten, in diesem Fall kann es folglich nur LED 9 sein:

```
LEDUPD  LDA #0
        STA TOR1B       LED 9 setzen
```

Das Muster für die dritte Spalte wird aus Adresse LTMSK + 2 geholt und auf dem Wert 0 getestet:

```
        LDA LTMSK +2
        BNE AUSLD9
```

Ist dieses Muster 0, muß LED 9 aufleuchten,

```
        LDA #1
        STA TOR1B
```

andernfalls wird nach AUSLD9 verzweigt, und die übrigen LEDs werden erleuchtet. Das aus LTMSK + 2 in den Akkumulator geholte Muster wird nacheinander einem ORA mit der zweiten und der ersten Spalte unterzogen:

```
        LDA #0
AUSLD9  ORA LTMSK
        ORA LTMSK +1
```

Jetzt endlich ist das fertige, zum Ausgangstor zu sendende LED-Muster im Akkumulator:

```
STA TOR1A
```

Zusätzlich wird der Lautsprecher in Betrieb gesetzt:

```
LDA TOR3B
EOR #$FF
STA TOR3B
```

Folgendes sollten Sie sich unbedingt klarmachen: Obwohl nur in einer der drei Spalten eine LED bewegt wurde, müssen alle drei Spalten „gezündet" werden, oder die beiden restlichen bleiben dunkel.

Nachdem Spalte 3 gebührend berücksichtigt worden ist, kommt die nächste an die Reihe: Zeiger X rutscht 1 nach unten:

```
NXTUPD    DEX
          BPL UPDTLP      wenn X> = 0: nächste Aktualisie-
                          rung
```

Wenn auch Spalten 2 und 1 abgehandelt sind, folgt eine Verzögerung, die ein zu rapides LED-Flackern verhindert, gesteuert wird sie vom Geschwindigkeitsparameter SPDPRM:

```
          LDY #SPDPRM
WARTEN    DEY
          BNE WARTEN
```

Nach Beendigung der Schleife folgt die Null-Probe für die Geschwindigkeiten aller drei Räder; eine dreimalige 0 heißt: Durchlauf beendet,

```
LDA SPEEDS
ORA SPEEDS +1
ORA SPEEDS +2
BNE UPDATE
```

oder es erfolgt der Sprung nach UPDATE. Stehen die Räder, muß der Anwender das Muster sehen können, ein Verweilen ist also nötig:

```
LDA #$FF
STA DAUER
JSR DELAY
```

Jetzt fehlt nur noch der Rücksprung:

```
RTS
```

**Übung 7-2:** *Die Inhalte der drei SPEEDS-Adressen wurden mittels ORA-Befehlen auf 0 geprüft. Wäre ihre Addition eine gleichwertige Vorgehensweise gewesen?*

### *Unterprogramm AUSTWG*

Dieses Unterprogramm ist die Schnittstelle für die Anwenderausgabe. Sie berechnet den Spielstand und erzeugt die audiovisuellen Effekte. Zu Beginn werden die Konstanten für den hochfrequenten Ton bei Spielgewinn und den niedrigfrequenten Ton bei Spielverlust definiert:

```
HITON = $20
LOTON = $F0
```

Die Berechnungsmethode für die Zahl der je Zeile erleuchteten LEDs wurde bereits erörtert und in Bild 7.7 illustriert. Zu Beginn sind alle drei Zeilen dunkel:

```
AUSWTG    LDA #0
          STA WERTE
          STA WERTE +1
          STA WERTE +2
```

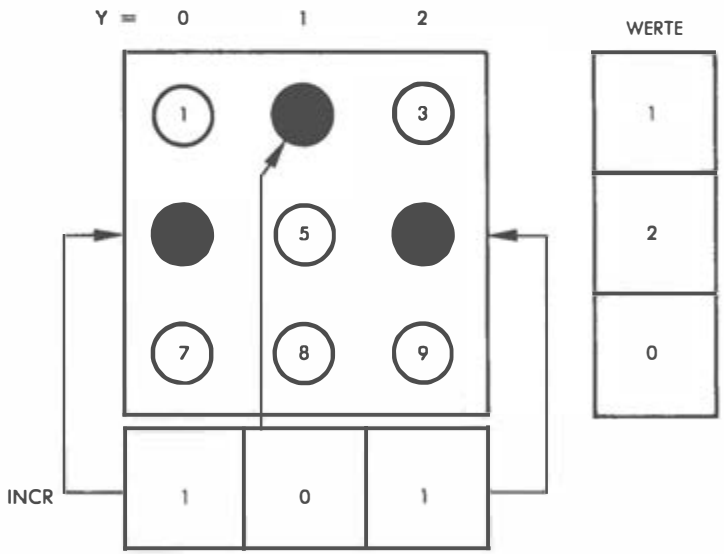



*Abb. 7.11: Endauswertung eines Durchlaufs*

Die Übergangspunktzahl ist ebenfalls 0:

```
        STA PKTEMP
```

Als Spaltenzeiger dient das Y-Register, und für jede Zeile wird die Zahl der erleuchteten LEDs bestimmt. Wie viele LEDs in der aktuellen Spalte leuchten, wird aus dem entsprechenden INCR-Speicher entnommen (siehe Beispiel 7.11). Jeder der drei INCR-Werte ist eine Zeilennummer, die im X-Register gespeichert wird und als Index für die Inkrementierung des entsprechenden Wertes in der WERTE-Tabelle dient. Beachten Sie, wie dies mit nur zwei Befehlen geschieht, wobei die Indizierungsfähigkeiten des 6502 zweimal genutzt werden:

```
CNTLP   LDY #2          3 Schritte
        LDX INCR,Y
        INC WERTE,X
```

Nach Spalte 2 kommen analog Spalten 1 und 0 an die Reihe:

```
        DEY
        BPL CNTLP
```

Jetzt folgt eine weitere Iteration, mit der die endgültigen Zahlen in der WERTE-Tabelle in die aktuellen Punktzahlen verwandelt werden, Bezug wird dabei auf die Punktetabelle PKTAB genommen.

```
        LDX #2
```

Da die PKTAB-Tabelle pro Zeile vier Byte-Eintragungen aufweist, muß zum Erreichen des richtigen Bytes die Zeilennummer mit 4 multipliziert werden, ehe dazu der entsprechende Wert (gleich Zahl der erleuchteten LEDs) dieser Zeile addiert wird. Die Multiplikation besorgen natürlich zwei Verschiebungen nach links:

```
PKTSCHL TXA
        ASL A
        ASL A
```

Die gegenwärtig im Akkumulator enthaltene Zahl ist das Vierfache des X-Registers bzw. des Zeilenregisters. Um die endgültige Versetzung innerhalb der PKTAB-Tabelle zu erhalten, müssen wir nun die in der gerade betrachteten Zeile erleuchteten LEDs addieren, also die in der WERTE-Tabelle enthaltene Zahl. Wir besorgen sie uns mittels indizierter Adressierung:

```
        CLC
        ADC WERTE,X     Spaltenadresse im WERTE-Feld
```

Damit haben wir nun auch die richtige Stelle im PKTAB-Feld, und wir können uns den entsprechenden Wert holen. Indexregister Y braucht dazu diesen Wert,

```
TAY
```

und wir können „zugreifen“:

```
LDA PKTAB,Y     Punktzahl für diesen Durchlauf
                holen
```

Die richtige Punktzahl für die erleuchteten LEDs in der Zeile, auf die das X-Register zeigt, steht jetzt im Akkumulator. Dieses Teilergebnis muß jetzt zur laufenden Zwischensumme aus allen schon betrachteten Zeilen addiert werden:

```
CLC
ADC PKTEMP      Punkte aufsummieren
STA PKTEMP      und wieder ablegen
```

Der Zeilenzähler wird dekrementiert, um die nächste Zeile untersuchen zu können. Geht Zähler X dabei ins Minus, sind wir fertig, andernfalls bleiben wir in der Schleife:

```
DEX
BPL PKTSCHL
```

Die Gesamtpunktzahl für diesen Durchlauf haben wir jetzt, und der Spieler wird akustisch und visuell über Erfolg oder Mißerfolg informiert. In Anbetracht des zu aktivierenden Lautsprechers versorgen wir Adresse DAUER mit der Länge des Tones:

```
LDA #$60
STA DAUER
```

Der Punktstand muß nun auf den Wert 0 getestet werden, um eine Verlust- bzw Gewinnsituation zu erkennen: 0 verliert und führt in die VERLUST-Routine,

```
LDA PKTEMP
BEQ VERLUST
```

während „nicht-0“ in die GEWINN-Routine überleitet. Betrachten wir nun diese beiden Routinen.

*GEWINN-Routine*

Das aktuelle Punktekonto (aus allen bisherigen Durchläufen) befindet sich in Adresse PUNKTE, wo es nach jeder Erhöhung auf den Maximalwert 16 getestet wird:

```
GEWINN    INC PUNKTE
          LDY PUNKTE
          CPY #16
```

Ist dieser Maximalwert erreicht, bedeutet das das Ende des Spiels, das Programm springt nach GEWENDE:

```
          BEQ GEWENDE
```

Andernfalls wird der Punktstand angezeigt

```
          JSR LICHT
```

und ein Piepton erzeugt

```
          LDA #HITON
          JSR TON
```

Diese beiden Unterprogramme werden weiter unten erläutert. Es folgt eine Verzögerung,

```
          JSR DELAY
```

und die Punktzahl wird dekrementiert:

```
          DEC PKTEMP
```

Ist 0 erreicht (d.h. alle Punkte sind zum Gesamtpunktstand addiert), ist der Zählvorgang beendet, andernfalls geht es zurück in die Schleife:

```
          BNE GEWINN
          RTS
```

*GEWENDE-Routine*

Diese Routine wird dann angesprungen, wenn die Gesamtpunktzahl 16 das Spiel beendet. Alle LEDs gehen gleichzeitig an und ein aufsteigender Sirenenton erklingt. Danach kann das Spiel von vorn beginnen. Zur „Festbeleuchtung" wird das entsprechende LED-Muster an Tore 1A und 1B gesendet:

```
GEWENDE   LDA #$FF
          STA TOR1A        alle LEDs an
          STA TOR1B
```

Jetzt müssen die Variablen wieder initialisiert werden: PUNKTE wird auf 0 gesetzt, um dem Hauptprogramm den Beginn eines neuen Spiels zu signalisie-

ren; DAUER erhält den Wert 4 zur Kontrolle über die Länge der Pieptöne; der Frequenzparameter in TEMP wird zu FF:

```
        STA TEMP        Frequenzparameter
        LDA #0
        STA PUNKTE      Signal für Neubeginn
        LDA #4
        STA DAUER       Pieptondauer
```

Den Ton erzeugt das TON-Unterprogramm:

```
AUF     LDA TEMP        Frequenz holen
        JSR TON         Piepton erzeugen
```

Die Piepfrequenz wird dekrementiert, und der nächste Ton ist etwas höher:

```
        DEC TEMP
        BNE AUF
```

Kommt die Frequenzkonstante bei 0 an, ist das Sirenengeheul beendet, und das Programm springt zurück:

```
        RTS
```

*VERLUST-Routine*

Die Verlustsituation ruft Ereignisse hervor, die in etwa symmetrisch zu denen der Gewinnsituation verlaufen. Die Punktzahl wird allerdings nur einmal durch Dekrementieren justiert:

```
VERLUST DEC PUNKTE
```

Der Spieler erfährt sein neues Punktekonto:

```
        LDY PUNKTE
        JSR LICHT
```

Ein Ton wird erzeugt:

```
        LDA #LOTON
        JSR TON
```

Die Gesamtpunktzahl wird nun auf den Wert 0 getestet, um festzustellen, ob dieser ganze Durchgang verloren und das Spiel zu Ende ist, oder ob der nächste Durchlauf gestartet werden kann:

```
        LDY PUNKTE
        BEQ VERLENDE
        RTS
```

Nun wollen wir sehen, was passiert, wenn der Punktestand 0 (VERLENDE) erreicht ist: Ein absteigender Sirenenton wird erzeugt, und alle LEDs gehen aus:

```
VERLENDE   LDA #0
           STA TEMP
           STA TOR1A        LED 1 löschen
```

Wieder erhält Tonlängenspeicher DAUER den Wert 4:

```
           LDA #4
           STA DAUER
```

der Piepton wird erzeugt:

```
AB         LDA TEMP
           JSR TON
```

Der abwärts gerichtete Heulton entsteht durch Inkrementieren der Frequenzkonstante, Schluß ist bei einem Überlauf des TEMP-Registers:

```
           INC TEMP         nächster Ton etwas höher
           BNE AB
           RTS
```

Das beschließt die Beschreibung des Hauptprogramms. Es folgen noch die vier angesprochenen Unterprogramme DELAY, LICHT, TON und RANDOM.

### *Unterprogramm DELAY*

Die Verzögerung, die diese Routine bewirkt, ist eine Funktion des Wertes, der in Adresse DAUER steht, und beträgt (206 × DAUER + 10) Mikrosek. Bei der vertrauten doppelt geschachtelten Schleife kontrolliert das X-Register die Innenschleife und das Y-Register die Außenschleife, wobei letzteres den Anfangswert DAUER erhält:

```
DELAY      LDY DAUER
```

Es folgt die Innenschleife

```
DL1        LDX #$FF
DL2        BNE +2           Zeit „verschwenden“
           DEX              Innenschleifenzähler
           BNE DL2
```

und dann die Außenschleife:

```
DEY
BNE DL1
RTS
```

**Übung 7-3:** *Verifizieren Sie die genaue Verzögerungszeit der DELAY-Routine.*

***Unterprogramm LICHT***

Diese Routine erleuchtet die LED, die der im Y-Register stehenden Zahl entspricht. Wir erinnern uns dabei, daß die 15 LEDs des Spielbretts extern zwar von 1 bis 15 durchnumeriert sind, daß sie aber auf die zweimal acht Bits von Tor 1A und Tor 1B aufgeteilt sind. Soll z.B. der Punktstand 1 angezeigt werden, muß Bit 0 von Tor 1A an sein. Allgemein gilt für Tor 1A: Bit N muß eingeschaltet werden, wenn der Punktstand N-1 angezeigt werden soll. Eine Ausnahmesituation ergibt sich aus Bild 1.13, wo die LED-Verbindungen dargestellt sind. Bit 6 von Tor 1B ist offensichtlich an keine LED angeschlossen. Soll also der Punktstand 15 angezeigt werden, muß Bit 7 von Tor 1B angesprochen werden. Diese Ausnahme wird einfach so gehandhabt, daß beim Punktstand 15 die Dekrementierung entfällt.

Das korrekte Muster zur Erleuchtung einer spezifischen LED entsteht, indem eine 1 in die richtige Bitposition im Akkumulator geschoben wird. Eine andere Vorgehensweise wird in den Übungsaufgaben weiter unten vorgeschlagen.

Initialisieren wir zunächst:

```
LICHT   LDA #0
        STA TEMP
        STA TOR1A
        STA TOR1B
```

Der oben angesprochene Wert 15 im Y-Register wird durch „Nichtstun" berücksichtigt:

```
        CPY #15     Kode für unkorrigiertes Bit?
        BEQ +3      wenn ja: nichts tun
```

Jede andere Zahl wird vor der Verschiebung dekrementiert:

```
        DEY         interner Kode durch Dekrementieren
        SEC         zu verschiebendes Bit setzen
LTSHFT  ROL A
```

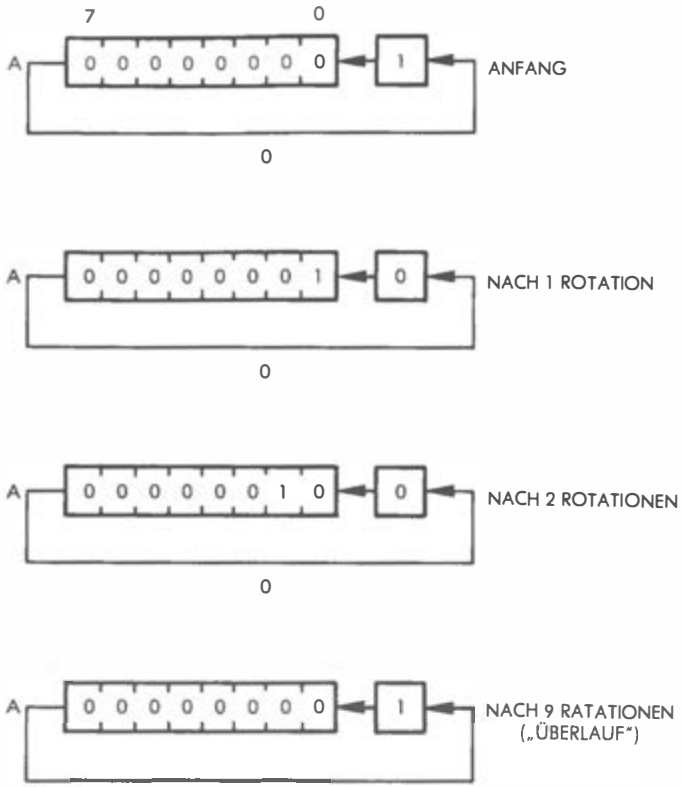



*Abb. 7.12: Erstellung des LED-Musters*

Der Akkumulatorinhalt wurde in der ersten Anweisung auf 0 gesetzt. Nun wird das Carrybit gesetzt und zunächst in die rechte äußere Position geschoben (siehe Bild 7.12) und dann weiter bis zur gewünschten Stelle. Da wir von 1 bis 14 bzw. von 0 bis 13 zählen müssen, findet ein Überlauf statt, wenn die 1 über die äußerste linke Position in Akkumulator hinausgeschoben wird. Solange das nicht geschieht, wird der Schiebevorgang als fortgesetzt und dann nach LTCC verzweigt:

```
BCC LTCC
```

Fällt die 1 jedoch nach links heraus, wird der Vorgang durch Einschreiben von FF in Adresse TEMP registriert, die vom zweiten Befehl der LICHT-Routine her ja noch den Wert 0 hatte:

```
LDX #$FF
STX TEMP
```

Die 1 im Carry-Bit wandert jetzt wieder rechts außen in den Akkumulator. Je nach dem Wert in TEMP kann nun später entschieden werden, ob das Akkumulator-Muster an Tor 1A oder Tor 1B gesendet werden muß. Der Schiebevorgang geht jetzt so lange weiter, bis der Zähler den Wert 0 hat, dann sind wir fertig:

```
        ROL A
LTCC    DEY
        BPL LTSHFT
```

Wie angekündigt, wird jetzt TEMP überprüft. Bei „0" hat kein Überlauf stattgefunden und Tor 1A muß benutzt werden, bei „FF" dagegen Tor 1B:

```
        LDX TEMP        Überlaufflagge holen
        BNE HIBYTE
LOBYTE  STA TOR1A       A an untere LEDs
        RTS             Rücksprung
HIBYTE  STA TOR1B       A an obere LEDs
        RTS
```

### *Unterprogramm TON*

Diese Routine erzeugt einen Piepton, dessen Frequenz durch den im Akkumulator mitgebrachten Wert bestimmt wird, und dessen Dauer in Adresse DAUER fixiert ist. Eine genaue Beschreibung fand in Kapitel 2 statt.

### *Unterprogramm RANDOM*

Dieser einfache Zufallszahlengenerator wurde in Kapitel 3 beschrieben.

**Übung 7-4:** *Erarbeiten Sie eine andere Methode zur Erzeugung des richtigen LED-Musters im Akkumulator, die nicht aus einer Rotationsfolge besteht.*

### Spielvariationen

Die drei LED-Reihen auf dem Spielbrett können auch anders verwendet werden als zu Beginn des Kapitels dargestellt. Zeile 1 könnte z.B. als „Kirschen", Zeile 2 als „Sterne" und Zeile 3 "als „Orangen" interpretiert werden. So würden z.B eine erleuchtete LED in Zeile 1 und zwei erleuchtete LEDs in Zeile 3 die Spielstellung „Kirsche-Orange-Orange" repräsentieren. Die Bewertungstabelle könnte dann so geändert werden, daß entsprechend bestimmten „Bild"-Kombinationen eine unterschiedliche Anzahl von Punkten gesetzt wird. Es kommt also darauf an, die Punktetabelle mit anderen Zahlen zu bestücken, um ein gänzlich anderes Bewertungssystem zu erhalten. Das Programm selbst müßte nicht verändert werden.

## ZUSAMMENFASSUNG

Dieses an sich einfach erscheinende Programm ist doch relativ komplex und kann zu vielerlei Spielvarianten führen, indem nur die Bewertungsmethode nach Stillstand der Lichter abgeändert wird. Die Strukturierung in mehrere getrennte Routinen macht das Programm transparent und ermöglicht ein gutes Einzelstudium der Programmteile.

# 8

# Echtzeitstrategien (Echo)

## EINFÜHRUNG

Zum Sammeln von Informationen wird eine Stapeltechnik benutzt. Sie ist vergleichbar mit einer Arbeitsspeichermethode.

## DIE SPIELREGELN

Bei diesem Spiel kommt es darauf an, Licht- und Tonfolgen, die vom Computer generiert wurden, zu erkennen und zu reproduzieren. Eine Reihe ähnlicher Spiele verschiedener Hersteller ist im Spielwarenhandel erhältlich (SIMON von MILTON BRADLEY CO. oder FOLLOW ME von ATARI). Bei unserer Version muß der Spieler zu Beginn die Länge der zu erkennenden Sequenz durch Drücken einer Taste zwischen 1 und 9 festlegen. Der Computer generiert dann eine Zufallsfolge dieser Länge, die mit einer der Tasten A bis F gehört und gesehen werden kann.

Nach Drücken einer dieser Tasten wird die vom Computer entworfene Sequenz visuell mit den LEDs 1 bis 9 und gleichzeitig akustisch durch eine entsprechende Tonfolge vorgestellt. Der Spieler muß diese Licht- und/oder Tonsequenz zu identifizieren versuchen und anschließend über die Tastatur wieder richtig eingeben. Bei jeder richtigen Tasteneingabe leuchtet die entsprechende LED am Spielbrett auf, bei jeder falschen Eingabe erklingt dagegen ein tiefer Ton.

Hat der Spieler zum Schluß alles richtig erkannt und eingegeben, so leuchten alle LEDs auf, und eine aufsteigende Tonleiter erklingt. Hat der Spieler Fehler gemacht, leuchtet eine einzelne LED auf, die die Fehlerzahl angibt, und eine Tonleiter abwärts wird gespielt.

Wenn die Sequenz richtig war, beginnt ein neues Spiel. Andernfalls wird die Fehlerzahl gelöscht, und der Spieler erhält einen weiteren Versuch, die Sequenz zu rekonstruieren.

Während des Spiels kann durch Drücken einer Buchstabentaste jederzeit die Sequenz neu abgefragt werden. Alle vorherigen Versuche werden dabei gelöscht, und der Spieler beginnt demnach von vorn.

Als „Kommunikations-LEDs“ zwischen Spieler und Computer dienen zwei LEDs der untersten Reihe:

LED 10 (links außen) zeigt an, daß der Computer bereit ist für die Eingabe der gewünschten Sequenzlänge.
LED 11 leuchtet unmittelbar nach dieser Eingabe auf und bleibt danach erleuchtet als Aufforderung, einen Versuch einzugeben.

Hierbei hat der Spieler nun drei Möglichkeiten:

1. Drücken einer Zifferntaste als Code für die vermeintliche nächste Zahl.
2. Drücken der 0-Taste zum Starten eines neuen Spiels.
3. Drücken einer Taste von A bis F, um den Computer zu veranlassen, die Sequenz zu wiederholen und dem Spieler einen weiteren Versuch zu ermöglichen.

**Variationen**

Das Programm stellt Ihr musikalisches Gehör auf die Probe, und Sie sollten versuchen, eine neue Sequenz nur akustisch im Gedächtnis zu behalten, ohne dabei auf die LEDs zu schauen. Da die LEDs auf dem Spielbrett durchnumeriert sind, ist die Sequenz als Lichtfolge relativ einfach zu behalten, indem man sich lediglich die Ziffernfolge merkt, und so leicht wollen Sie es sich doch sicher nicht machen!

Am besten beginnen Sie mit einem einzelnen Ton, dann versuchen Sie eine Zweitonfolge, dann drei Töne usw. Vielleicht können Sie sich auch mit anderen Spielern messen: Es gewinnt, wer die längste Sequenz richtig rekonstruiert. Es gibt übrigens Spieler, die auch Neuntonfolgen noch recht problemlos im Ohr behalten.

Um den Ratevorgang zu erleichtern, können Sie auch nach einer bestimmten Zeit (z.B. nach dem 5. Ton) dem Spieler einen Blick auf die LEDs erlauben. Oder Sie erlauben ihm, zu jedem beliebigen Zeitpunkt eine Buchstabentaste zu drücken, um die Sequenz nochmals zu hören, wobei dies eine Strafe kosten sollte, z.B. daß der Spieler vor dem Versuch mit einer längeren Sequenz zunächst nochmals die gleiche Länge zu bewältigen hätte. Ein Beispiel: Ein Spieler ist gerade mit einer Fünftonfolge beschäftigt, wird bei einem Fehler nervös und ist daraufhin so durcheinander, daß er sich die Sequenz nochmals anhört. Wenn er jetzt die Folge korrekt reproduziert, muß er als Strafe eine weitere Fünftonfolge bewältigen, um dann mit einer Sechstonfolge fortzufahren.

Wollen Sie den Wettkampf weiter verschärfen, können Sie auch die Zahl zulässiger „Wiederaufführungen" auf zwei, drei oder fünf begrenzen. Während einer Spielrunde darf diese „Reserve" dann nur bis zur festgesetzten Grenze in Anspruch genommen werden.

***Ein ASW-Tester***

Eine andere Spielvariante wäre, Sequenzen zu reproduzieren, ohne sie vorher zu hören oder zu sehen, was Sie dann nur noch mit Hilfe Ihrer ASW (außersinnliche Wahrnehmung) bewerkstelligen können. Setzen Sie dazu den Längenzäh-

ler auf 1 und versuchen Sie den Ton/die Zahl des Programms zu „erspüren". Ein Maß für Ihr ASW-Potential erhalten Sie aus einer Folge von Versuchen. Fehlt es Ihnen am 7. oder 8. Sinn, so wird ihre Trefferquote dem statistischen Mittelwert von 11.11 % entsprechen (im Schnitt raten Sie jedes neunte Mal richtig), wobei die Versuchszahl natürlich genügend groß sein muß.

Wenn Sie diese 11 Prozent aber übertreffen, dann schlummern vielleicht wirklich ASW-Kräfte in Ihnen. Im Bereich über 50 Prozent steht Ihnen wahrscheinlich eine Karriere als Politiker oder Wirtschaftsboß bevor. Bleiben Sie dagegen nennenswert unter 11 Prozent, so ist Ihre ASW negativ, vielleicht sollten Sie dann vor dem Überqueren einer Straße mehrmals in beide Richtungen schauen.

Für Leser mit Statistikkenntnissen wird folgende Aufgabe gestellt:

**Übung 8-1:** *Berechnen Sie die statistischen Wahrscheinlichkeiten, eine Zweitonfolge und eine Viertonfolge richtig vorherzusagen.*

## EIN TYPISCHER SPIELVERLAUF

Das Programm beginnt bei Adresse 200. LED 10 leuchtet auf (Bild 8.1), und wir „bestellen" eine Zweitonfolge durch Drücken von Taste 2. Das LED-Muster in Bild 8.2 erscheint und fordert uns auf, unseren Vorschlag einzugeben.

Da wir die Sequenz kennenlernen wollen, drücken wir Taste F, und prompt leuchten LEDs 5 und 2 kurz nacheinander auf, jeweils begleitet von den entsprechenden Tönen im Lautsprecher (Bild 8.3). Jetzt geben wir ein, was wir erkannt zu haben glauben, als erstes 5. LED 11 erlischt kurz und LED 5 leuchtet ebenso kurz auf, während gleichzeitig der entsprechende Ton erklingt: Wir lagen richtig!

Als nächstes drücken wir Taste 2, und tatsächlich: LED 2 und der zugehörige Ton bestätigen uns, daß auch das richtig war. Kurz darauf gehen alle LEDs an und beglückwünschen uns, dazu erklingt eine aufwärtssteigende Tonleiter. Danach beginnt ein neues Spiel, was LED 10 anzeigt (wieder Bild 8.1).

Verfolgen wir nun auch einen mißglückten Versuch. Nach dem Aufleuchten von LED 10 drücken wir die Taste 1, um eine Eintonfolge zu bekommen. LED 11 leuchtet als Aufforderung, und mittels Taste F entlocken wir dem Lautsprecher einen Ton. Diesmal schauen wir nicht auf die LEDs, um nur das Gehör zu testen. Jetzt wird Taste 3 gedrückt ... und es erklingt ein „Verloren"-Ton, während LED 1 die Fehlerzahl 1 gleichzeitig bekanntgibt. Danach läuft eine Tonleiter abwärts und bestätigt den mißglückten Rateversuch. Anschließend bekommen wir mit dem LED-Muster von Bild 8.2 eine neue Chance.

An diesem Punkt können wir durch Drücken der 0-Taste das Spiel neu starten, um die Sequenzlänge neu festzulegen oder um eine neue Folge einzugeben. Die Situation nach Eingabe von 0 ist dann wieder die in Bild 8.1 dargestellte.


*Abb. 8.1: Eingabe der Sequenzlänge*


*Abb. 8.2: Eingabe eines Versuchs*

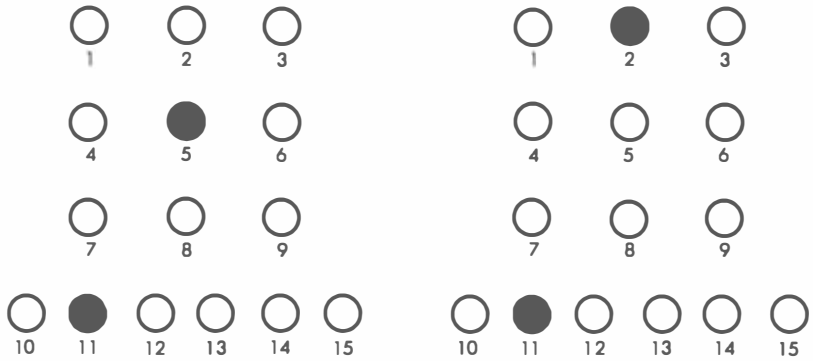



*Abb. 8.3: „Folge mir"*

## DER ALGORITHMUS

Betrachten wir das Flußdiagramm (Bild 8.4) Schritt für Schritt:

1. Das Programm fordert den Spieler mit LED 10 auf, die Sequenzlänge zu wählen.
2. Die Länge wird von der Tastatur eingelesen (Tasten 0 und A bis F werden hierbei ignoriert).
3. Die zwei Hauptvariablen, Versuchszahl und Fehlerzahl, werden auf den Anfangswert 0 gesetzt.
4. Eine Sequenztabelle der erforderlichen Länge wird erstellt und mit Zufallszahlen zwischen 1 und 9 gefüllt.
5. LED 11 wird erleuchtet und die Tasteneingabe wird gelesen.
6. Ist es die 0-Taste, wird ein neues Spiel gestartet, andernfalls geht es weiter.

7. Ist der Tastenwert größer als 9, handelt es sich um eine Buchstabentaste, und wir verzweigen in den rechten Flußdiagrammteil zu den Schritten 8 und 9: Die Sequenz wird dem Spieler vorgeführt, die Variablen zurück auf 0 gesetzt, und der Rateprozeß beginnt von vorn. Lag die Tastennummer zwischen 1 und 9, muß sie mit dem aktuellen Sequenzwert verglichen werden, das ist Schritt 10 im Flußdiagramm.
10. Ist richtig geraten worden, geht es mit Schritt 11 weiter.
11. Da Tastennummer und gespeicherter Sequenzwert übereinstimmen, muß die entsprechende LED auf dem Spielbrett erleuchtet und der passende Ton erzeugt werden.
12. Die Versuchsnummer wird erhöht und mit der fixierten Maximallänge der Sequenz verglichen.
13. Ist die Maximallänge nicht erreicht, geht es zurück zu Schritt 5, und der nächste Tastendruck wird geholt. Andernfalls geht es mit Schritt 14 weiter.
14. Die Gesamtfehlerzahl wird geprüft. Ist die Variable FEHLER = 0, so hat der Spieler gewonnen, und Schritt 15 folgt.
15. Alle LEDs auf dem Spielbrett werden erleuchtet, eine aufsteigende Tonleiter generiert, und es geht zum Start zurück.

Gehen wir zurück zu 14. War die Fehlerzahl größer als 0, verzweigt das Programm nach 16:

16. Die Fehlerzahl wird angezeigt und eine abwärtsschreitende Tonleiter gespielt.
17. Alle Variablen gehen zurück auf 0, und der nächste Programmschritt liegt bei 5, wo der Spieler einen neuen Versuch hat.

Wenden wir uns nun Schritt 10 im Flußdiagramm zu, wo der Tastenwert mit dem aktuellen Sequenzwert verglichen wurde. War der Versuch ein Fehlversuch, geht es links von Kasten 10 weiter:

18. Die Fehlerzahl wird um 1 erhöht.
19. Ein tiefer Ton kennzeichnet die Verlustsituation, dann geht es zurück nach 12.

## DAS PROGRAMM

Das vollständige Programm finden Sie in Bild 8.5. Zur Verwendung kommen zwei Tabellen: NOTAB für die benötigten Tonfrequenzen und DURTAB für die Tondauer; beide wurden in Kapitel 2 ausführlich beschrieben und werden hier nur kurz umrissen. Im wesentlichen enthalten sie die Verzögerungskonstanten zur Erzeugung von Tönen der gewünschten Dauer und Frequenz. Sie können übrigens den Schwierigkeitsgrad des Spiels leicht dadurch modifizieren, daß Sie die Tondauer verlängern (was das Spiel leichter macht) oder verkürzen (wodurch es schwieriger wird). Experimentieren Sie ruhig etwas damit.

Die wesentlichen Variablen des Programms sind folgende:
DIGITS enthält die Zahl der Töne in der zu ratenden Sequenz.

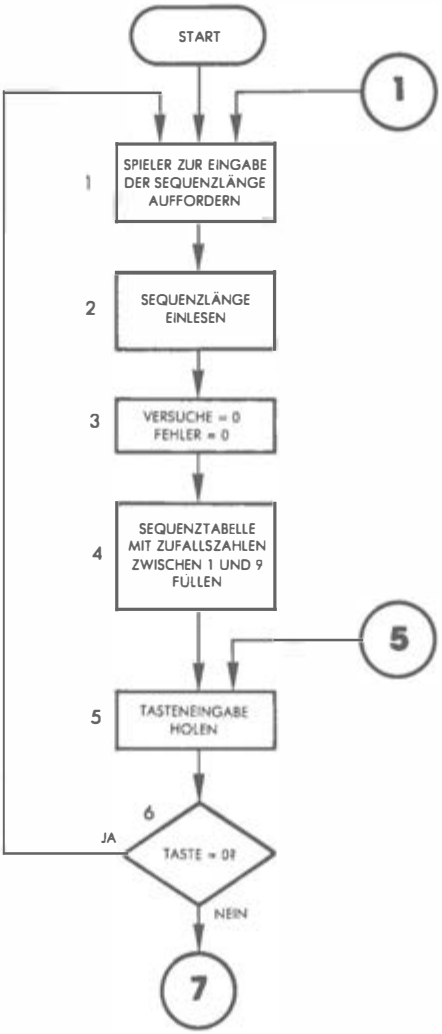



*Abb. 8.4: Flußdiagramm ECHO*

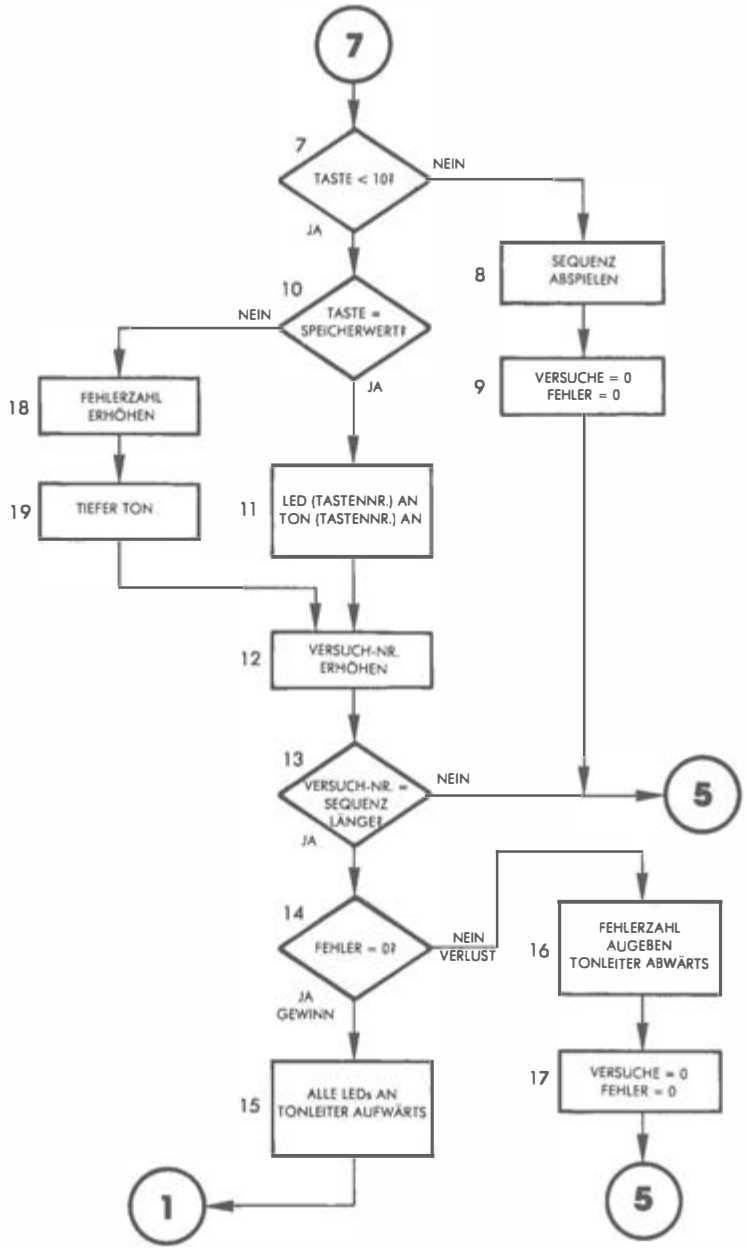



*Abb. 8.4: Flußdiagramm ECHO (Fortsetzung)*

```
                        ;   ECHO
                        ;AUDIOVISUELLES GEDAECHTNIS- UND  ASW -TESTPROGRAMM
                        ;DER ANWENDER VERSUCHT, LED-MUSTER UND DAMIT GEKOPPELTE TON-
                        ;SEQUENZEN ZU RATEN. DIE SEQUENZEN KÖNNEN ABGESPIELT WERDEN, UND
                        ;DER SPIELER MUSS SIE DANN IM GEDÄCHTNIS BEHALTEN UND KORREKT
                        ;WIEDERZUGEBEN VERSUCHEN.
                        ;   BEDIENUNG DES PROGRAMMS
                        ;STARTADRESSE IST $200. DIE UNTERSTE LEDREIHE ZEIGT DEN PRO-
                        ;GRAMMSTATUS AN: LED 10 FORDERT AUF, DIE GEWÜNSCHTE SEQUENZ-
                        ;LÄNGE FESTZULEGEN. LED 11 ZEIGT DANACH AN, DASS DAS PROGRAMM
                        ;EINEN VON DREI MÖGLICHEN BEFEHLEN ERWARTET: EINEN RATEVERSUCH
                        ;(TASTEN 1-9), NEUBEGINN (TASTE 0) ODER ABSPIELEN DER GESPEI-
                        ;CHERTEN SEQUENZ (TASTEN A-F). DIE TASTEN 1-9 ENTSPRECHEN DEN
                        ;LEDS 1-9. EIN ABFRAGEN DER SEQUENZ WÄHREND DES RATENS LÖSCHT
                        ;ALLE VORHERIGEN VERSUCHE (VARIABLE 'VRSNR' UND 'FEHLER').
                        ;NACH EINEM ERFOLG BEGINNT DAS PROGRAMM VON VORN.
                        ;
                        ;GEKOPPELTE TASTENEINGABE-ROUTINE:
                        GETKEY    = $100
                        ;
                        ;VARIABLENSPEICHER:
                        DIGITS    = $00         ;ANZAHL DER STELLEN (SEQUENZLÄNGE)
                        VRSNR     = $01         ;NUMMER DES LAUFENDEN RATEVERSUCHS
                        FEHLER    = $02         ;ANZAHL BISHERIGER RATE-FEHLVERSUCHE
                        DAUER     = $03         ;ZWISCHENSPEICHER FÜR TONDAUERKONSTANTE
                        FREQ      = $04         ;ZWISCHENSPEICHER FÜR FREQUENZKONSTANTE
                        TEMP      = $05         ;ZWISCHENSPEICHER FÜR X-REGISTER
                        TABLE     = $06         ;SEQUENZSPEICHER
                        RND       = $0F         ;ARBEITSSPEICHER ZUFALLSZAHLENGENERATOR
                        ;6522 VIA 1 ADRESSEN:
                        TOR1A     = $A001
                        DDR1A     = $A003
                        TOR1B     = $A000
                        DDR1B     = $A002
                        T1CL      = $A004
                        ;6522 VIA 3 ADRESSEN:
                        TOR3B     = $AC00
                        DDR3B     = $AC02
                        ;
                                  * = $200
                        ;
0200: A9 FF     START     LDA #$FF        ;DATENRICHTUNGSREGISTER SETZEN
0202: 8D 03 A0            STA DDR1A
0205: 8D 02 A0            STA DDR1B
0208: 8D 02 AC            STA DDR3B
020B: A9 00               LDA #0          ;VARIABLENSPEICHER LÖSCHEN
020D: 8D 01 A0            STA TOR1A       ;LEDS LÖSCHEN
0210: 85 02               STA FEHLER
0212: 85 01               STA VRSNR
0214: AD 04 A0            LDA T1CL        ;AUSGANGSWERT FÜR ZUFALLSZAHLENGENERATOR
0217: 85 10               STA RND+1       ;IM ARBEITSSPEICHER ABLEGEN
0219: 85 13               STA RND+4
021B: A9 02               LDA #%010       ;LED 10 AN: SIGNAL ZUR EINGABE DER GE-
021D: 8D 00 A0            STA TOR1B       ;WÜNSCHTEN SEQUENZLÄNGE
0220: 20 00 01  DIGKEY    JSR GETKEY      ;SEQUENZLÄNGE EINLESEN
0223: C9 00               CMP #0          ;EINGABE = 0 ?
0225: F0 F9               BEQ DIGKEY      ;WENN JA: NEUE EINGABE
0227: C9 0A               CMP #10         ;SEQUENZLÄNGE > 9 ?
0229: 10 F5               BPL DIGKEY      ;WENN JA: NEUE EINGABE
022B: 85 00               STA DIGITS      ;ZULÄSSIGE EINGABE ABSPEICHERN
022D: AA                  TAX             ;LÄNGE-1 ALS INDEX FÜR ERZEUGUNG DER
022E: CA                  DEX             ;EINZELNEN ZUFALLSZAHLEN VERWENDEN
022F: 86 05     FILL      STX TEMP        ;X ZWISCHENSPEICHERN
0231: 20 E7 02            JSR RANDOM
0234: A6 05               LDX TEMP        ;X WIEDER ZURÜCKHOLEN
0236: F8                  SED             ;DEZIMAL EINPASSEN
0237: 18                  CLC
0238: 69 00               ADC #0
023A: D8                  CLD
023B: 29 0F               AND #$0F        ;AUF ZAHL ZWISCHEN 0 UND 9 'BESCHNEIDEN
023D: F0 F0               BEQ FILL        ;WERT 0 WIRD NICHT AKZEPTIERT
023F: 95 06               STA TABLE,X     ;ZAHL IN TABELLE ABLEGEN
```

*Abb. 8.5: Programm ECHO*

```
0241: CA                DEX             ;DEKREMENTIEREN FÜR NÄCHSTE ZIFFER
0242: 10 EB             BPL FILL        ;WENN NICHT FERTIG: NÄCHSTE ZIFFER
0244: A9 00     TASTE   LDA #0          ;LEDS LÖSCHEN
0246: 8D 01 A0          STA TOR1A
0249: A9 04             LDA #%0100      ;LED 12 ALS EINGABESIGNAL AN
024B: 8D 00 A0          STA TOR1B
024E: 20 00 01          JSR GETKEY      ;RATEVERSUCH ODER ABSPIELBEFEHL HOLEN
0251: C9 00             CMP #0          ;TASTE 0 ?
0253: F0 AB     STRTJP  BEQ START       ;WENN JA: NEUBEGINN
0255: C9 0A             CMP #10         ;TASTE < 10 ?
0257: 30 22             BMI AUSWTG      ;WENN JA: RATEVERSUCH AUSWERTEN
                ;
                ;ROUTINE ZUM AKUSTISCHEN UND VISUELLEN VORFÜHREN DER SEQUENZ
                ;
0259: A2 00     VORFRG  LDX #0
025B: 86 01             STX VRSNR       ;ALLE BISHERIGEN RATEVERSUCHE LÖSCHEN
025D: 86 02             STX FEHLER      ;ALLE BISHERIGEN FEHLER LÖSCHEN
025F: B5 06     VFGSCHL LDA TABLE,X     ;X. TABELLENEINTRAG HOLEN
0261: 86 05             STX TEMP        ;X ZWISCHENSPEICHERN
0263: 20 CF 02          JSR LICHT       ;LED ENTSPRECHEND TABELLE(X) AN
0266: 20 FA 02          JSR SPIEL       ;TON ENTSPRECHEND TABELLE(X) SPIELEN
0269: A0 FF             LDY #$FF        ;SCHLEIFENZÄHLER FÜR VERZÖGERUNG
026B: 66 03     DELAY   ROR DAUER       ;'ZEITVERSCHWENDUNG'
026D: 26 03             ROL DAUER
026F: 88                DEY             ;HERUNTERZÄHLEN
0270: D0 F9             BNE DELAY       ;WENN NICHT FERTIG: SCHLEIFE
0272: A6 05             LDX TEMP        ;X WIEDERHOLEN
0274: E8                INX             ;NÄCHSTE ZIFFER
0275: E4 00             CPX DIGITS      ;ALLES VORGEFÜHRT?
0277: D0 E6             BNE VFGSCHL     ;WENN NICHT: WEITER
0279: F0 C9             BEQ TASTE       ;FERTIG: NÄCHSTE EINGABE HOLEN
                ;
                ;AUSWERTUNG DER RATEVERSUCHE DES SPIELERS
                ;
027B: A6 01     AUSWTG  LDX VRSNR       ;NR. DES VERSUCHS HOLEN
027D: D5 06             CMP TABLE,X     ;RICHTIG GERATEN?
027F: F0 0D             BEQ RICHTG      ;WENN JA: ANZEIGEN
0281: E6 02     FALSCH  INC FEHLER      ;FALSCH GERATEN: FEHLERZAHL ERHÖHEN
0283: A9 80             LDA #$80        ;TONDAUER FÜR TIEFEN TON SETZEN, UM
0285: 85 03             STA DAUER       ;FEHLVERSUCH ANZUZEIGEN
0287: A9 FF             LDA #$FF        ;FREQUENZKONSTANTE
0289: 20 04 03          JSR SPLTON      ;TON SPIELEN
028C: F0 06             BEQ ENDTST      ;AUF SPIELENDE TESTEN
028E: 20 CF 02  RICHTG  JSR LICHT       ;ERFOLGREICHEN VERSUCH ANZEIGEN
0291: 20 FA 02          JSR SPIEL
0294: E6 01     ENDTST  INC VRSNR       ;EIN ZUSÄTZLICHER VERSUCH
0296: A5 00             LDA DIGITS
0298: C5 01             CMP VRSNR       ;ALLE ZIFFERN DURCH?
029A: D0 AB             BNE TASTE       ;WENN NICHT: NÄCHSTE TASTE
029C: A5 02             LDA FEHLER      ;BISHERIGE FEHLERZAHL HOLEN
029E: C9 00             CMP #0          ;SCHON FEHLER GEMACHT?
02A0: F0 15             BEQ SIEG        ;WENN NICHT: SPIELER GEWINNT
02A2: 20 CF 02  VRLST   JSR LICHT       ;FEHLERZAHL ANZEIGEN
02A5: A9 09             LDA #9          ;8 ABSTEIGENDE TÖNE SPIELEN
02A7: 48        VLSTSCHL PHA
02A8: 20 FA 02          JSR SPIEL
02AB: 68                PLA
02AC: 38                SEC
02AD: E9 01             SBC #1
02AF: D0 F6             BNE VLSTSCHL
02B1: 85 01             STA VRSNR       ;VARIABLE LÖSCHEN
02B3: 85 02             STA FEHLER
02B5: F0 8D             BEQ TASTE       ;NÄCHSTE RATEFOLGE HOLEN
02B7: A9 FF     SIEG    LDA #$FF        ;SIEG: ALLE LICHTER AN
02B9: 8D 01 A0          STA TOR1A
02BC: 8D 00 A0          STA TOR1B
02BF: A9 01             LDA #1          ;8 AUFSTEIGENDE TÖNE SPIELEN
02C1: 48        SIEGSCHL PHA
02C2: 20 FA 02          JSR SPIEL
02C5: 68                PLA
02C6: 18                CLC
```

*Abb. 8.5: Programm ECHO (Fortsetzung)*

```
02C7: 69 01             ADC #1
02C9: C9 0A             CMP #10
02CB: D0 F4             BNE SIEGSCHL
02CD: F0 94             BEQ STRTJP      ;ZWEIFACHSPRUNG ZUM NEUBEGINN
                        ;
                        ;UNTERPROGRAMM ZUM ERLEUCHTEN VON LED 'N', WOBEI 'N' DIE IM
                        ;AKKUMULATOR MITGEBRACHTE ZAHL IST
                        ;
02CF: 48        LICHT   PHA             ;A AM STAPEL ZWISCHENSPEICHERN
02D0: AB                TAY             ;UND ALS ZÄHLER ÜBERTRAGEN
02D1: A9 00             LDA #0          ;A FÜR BITVERSCHIEBUNG LÖSCHEN
02D3: 8D 00 A0          STA TOR1B       ;OBERE LEDS LÖSCHEN
02D6: 38                SEC             ;BIT ZUM VERSCHIEBEN SETZEN
02D7: 2A        LNKSHFT ROL A           ;GESETZTES BIT NACH LINKS SCHIEBEN
02D8: 88                DEY             ;ZÄHLER DEKREMENTIEREN
02D9: D0 FC             BNE LNKSHFT     ;SCHIEBEVORGANG BEENDET?
02DB: 8D 01 A0          STA TOR1A       ;RICHTIGES BITMUSTER SENDEN
02DE: 90 05             BCC LTCC        ;FERTIG, WENN BIT 9 NICHT GESETZT
02E0: A9 01             LDA #1
02E2: 8D 00 A0          STA TOR1B       ;LED 9 AN
02E5: 68        LTCC    PLA             ;A VOM STAPEL ZURÜCKHOLEN
02E6: 60                RTS             ;FERTIG
                        ;
                        ;ZUFALLSZAHLENGENERATOR. RÜCKSPRUNG MIT ZAHL IM AKKUMULATOR
                        ;
02E7: 38        RANDOM  SEC
02E8: A5 10             LDA RND+1
02EA: 65 13             ADC RND+4
02EC: 65 14             ADC RND+5
02EE: 85 0F             STA RND
02F0: A2 04             LDX #4
02F2: B5 0F     RNDLP   LDA RND,X
02F4: 95 10             STA RND+1,X
02F6: CA                DEX
02F7: 10 F9             BPL RNDLP
02F9: 60                RTS
                        ;
                        ;TONERZEUGUNGSROUTINE. TONPARAMETER IST IM AKKUMULATOR BEREIT-
                        ;GESTELLT. BEIM EINSPRUNG BEI 'SPLTON' WIRD TON GESPIELT, DESSEN
                        ;LÄNGE IN 'DAUER' UND DESSEN FREQUENZ IM AKKUMULATOR STEHT.
                        ;
02FA: AB        SPIEL   TAY             ;TON-NR. ALS INDEX ÜBERNEHMEN
02FB: BB                DEY             ;FÜR TABELLENZUGRIFF DEKREMENTIEREN
02FC: B9 27 03          LDA DURTAB,Y    ;TONDAUER AUS TABELLE HOLEN
02FF: 85 03             STA DAUER       ;UND ABSPEICHERN
0301: B9 1E 03          LDA NOTAB,Y     ;FREQUENZKONSTANTE HOLEN
0304: 85 04     SPLTON  STA FREQ        ;UND ABSPEICHERN
0306: A9 00             LDA #0          ;LAUTSPRECHERTOR 3B AUF 0
0308: 8D 00 AC          STA TOR3B
030B: A6 03             LDX DAUER       ;TONDAUER ALS ZAHL VON HALBZYKLEN
030D: A4 04     FL2     LDY FREQ        ;FREQUENZ HOLEN
030F: 88        FL1     DEY             ;HERUNTERZÄHLEN
0310: 18                CLC             ;'ZEITVERSCHWENDUNG'
0311: 90 00             BCC +2
0313: D0 FA             BNE FL1         ;VERZÖGERUNGSSCHLEIFE
0315: 49 FF             EOR #$FF        ;TOR KOMPLEMENTIEREN
0317: 8D 00 AC          STA TOR3B
031A: CA                DEX             ;HERUNTERZÄHLEN
031B: D0 F0             BNE FL2         ;SCHLEIFE BIS TON ZUENDEGESPIELT
031D: 60                RTS             ;FERTIG
                        ;
                        ;FREQUENZKONSTANTEN-TABELLE
                        ;
031E: C9        NOTAB   .BYTE $C9,$BE,$A9,$96,$8E,$7E,$70,$64,$5E
031F: BE
0320: A9
0321: 96
0322: 8E
0323: 7E
0324: 70
0325: 64
0326: 5E
```

*Abb. 8.5: Programm ECHO (Fortsetzung)*

```
                ;
                ;TONDAUERKONSTANTEN-TABELLE
                ;
0327: 6B        DURTAB    .BYTE $6B,$72,$80,$8F,$94,$AA,$BF,$D7,$E4
032B: 72
0329: 80
032A: 8F
032B: 94
032C: AA
032D: BF
032E: D7
032F: E4

SYMBOLTABELLE:

RICHTG   028E    DDR1A    A003    DDR1B    A002    DDR3B    AC02
DELAY    026B    DIGITS   0000    DIGKEY   0220    DAUER    0003
DURTAB   0327    ENDTST   0294    FEHLER   0002    AUSWTG   027B
FILL     022F    FL1      030F    FL2      030D    FREQ     0004
VRSNR    0001    GETKEY   0100    TASTE    0244    LICHT    02CF
VRLST    02A2    VLSTSCHL 02A7    LTCC     02E5    LNKSHFT  02D7
NOTAB    031E    SPIEL    02FA    SPLTON   0304    TOR1A    A001
TOR1B    A000    TOR3B    AC00    RANDOM   02E7    RND      000F
RNDLP    02F2    VORFRG   0259    VFGSCHL  025F    START    0200
STRTJP   0253    TICL     A004    TABLE    0006    TEMP     0005
SIEG     02B7    SIEGSCHL 02C1    FALSCH   02B1
```

*Abb. 8.5: Programm ECHO (Fortsetzung)*

VRSNR ist die augenblickliche Versuchsnummer und bezeichnet den Ton der Sequenz, der gerade zu raten ist.
In FEHLER steht die bisher aufgelaufene Fehlerzahl.
TABLE enthält die zu rekonstruierende Tonfolge.
Dazu kommen verschiedene andere Adressen, die als Zwischen- und Arbeitsspeicher dienen. Sie werden im Zusammenhang mit den Routinen besprochen, in denen sie gebraucht und verwendet werden.

Wie gewöhnlich werden zuerst die Datenrichtungsregister für die Tore 1A, 1B und 3A auf Ausgabe gesetzt:

```
START   LDA #$FF
        STA DDR1A
        STA DDR1B
        STA DDR3B
```

Dann werden alle LEDs auf dem Spielbrett gelöscht:

```
        LDA #0
        STA TOR1A
```

Die beiden Variablen VRSNR und FEHLER werden genullt:

```
        STA FEHLER
        STA VRSNR
```

Der Zufallszahlengenerator erhält seinen Anfangswert, Speicherstellen sind RND + 1 und RND + 4:

```
        LDA T1CL        Zeitgeberzähler einlesen
        STA RND + 1
        STA RND + 4
```

Das Spiel kann beginnen, und LED 10 wird erleuchtet:

```
        LDA #%010       Muster für LED 10
        STA TOR1B       Länge festlegen
```

Unsere bekannte GETKEY-Routine aus Kapitel 1 fragt die Tastatur ab:

```
DIGKEY  JSR GETKEY
```

Test auf den Wert 0:

```
        CMP #0          bei 0: neue Abfrage
        BEQ DIGKEY
```

Geht 0 ein, so wartet das Programm weiter, andernfalls wird auf den Wert 10 geprüft:

```
        CMP #10         Sequenz größer als 9?
        BPL DIGKEY
```

Damit wird jede Sequenzlänge größer 9 zurückgewiesen. Werden auf diese Weise zulässige Wertbereiche abgesteckt, so spricht man auch von Grenzwertfilterung („bracket-filtering").

Jetzt haben wir eine zulässige Sequenzlänge, sie wird in DIGITS gespeichert:

```
        STA DIGITS      Sequenzlänge
```

Ein laufender Zeiger auf die aktuelle Sequenzposition erhält den Wert dieser Länge minus 1 und geht in die Speicherstelle TEMP:

```
        TAX             mit X wird berechnet
        DEX             minus 1
FILL    STX TEMP
```

Nun besorgt das Unterprogramm RANDOM eine Zufallszahl:

```
        JSR RANDOM
```

Aus TEMP wird der Positionszeiger nun zurückgeholt und ans X-Register

übergeben, das später als Index für die Abspeicherung der einzelnen Zahlen in TABLE dient:

```
LDX TEMP
```

Die nun erforderliche Umwandlung der Zufallszahl in einen Dezimalwert zwischen 0 und 9 kann auf unterschiedliche Weise bewerkstelligt werden. Wir wollen hier den speziellen Dezimalmodus des 6502 dazu ausnutzen, also setzen wir die D-Flagge

```
SED             Dezimalmodus setzen
```

und löschen die Carryflagge für eine folgende Addition:

```
CLC             Carry-Bit = 0
```

Nun kommt der spezielle Kunstgriff: Zu der im Akkumulator stehenden Zahl wird schlicht 0 addiert, was im Dezimalmodus mit Sicherheit dazu führt, daß im niederwertigen Halbbyte des A-Registers eine Zahl zwischen 0 und 9 steht. Natürlich könnten wir auch eine andere Zahl addieren, um zu einer Dezimalzahl zu kommen, allerdings würde der Bereich dadurch nach unten beschnitten, und Zahlen wie 0, 1 oder 2 würden niemals vorkommen. Danach wird der Dezimalbetrieb wieder abgeschaltet.

```
ADC #0          im Dezimalbetrieb 0 addieren
CLD             Dezimalbetrieb abschalten
```

Wir haben damit eine sehr leistungsfähige Eigenschaft des 6502 recht vorteilhaft ausgenutzt. Damit wir endgültig eine Zahl zwischen 0 und 9 im Akkumulator haben, werfen wir den höherwertigen Nibble durch Maskieren hinaus:

```
AND #$0F
```

Da wir den Wert 0 nicht brauchen können, muß in diesem Fall eine neue Zahl generiert werden:

```
BEQ FILL
```

**Übung 8-2:** *Könnte der Wert 0 auch dadurch umgangen werden, daß etwas anderes als 0 zum Akkumulator addiert wird?*

Ist A nicht 0, so haben wir unsere Zufallszahl zwischen 1 und 9, sie kann also gespeichert werden. Dazu erinnern wir uns, daß im X-Register noch die aktuelle Sequenzposition steht, dies ist also unser Index:

```
STA TABLE,X     Zufallszahl ablegen
```

Für die nächste Position muß der Zeiger nun dekrementiert werden,

```
DEX
```

und falls wir nicht fertig sind, kommt die nächste Zahl:

```
                BPL FILL
```

Alles ist bereit für die Entgegennahme der Spielereingabe. LED 12 wird erleuchtet:

```
TASTE           LDA #0
                STA TOR1A
                LDA #%0100
                STA TOR1B
```

Die Spielertaste wird eingelesen

```
                JSR GETKEY      Zahl holen
```

und auf 0 oder einen Buchstabenwert getestet. Zuerst auf 0:

```
                CMP #0          Taste 0?
STRTJP          BEQ START       wenn ja: Neubeginn
```

Ist keine 0 eingegeben worden (was zu START führen würde), folgt der Test auf eine Buchstabentaste:

```
                CMP #10         Zahl kleiner 10?
                BMI AUSWTG      wenn ja: auf Übereinstimmung
                                prüfen
```

Ist die Eingabe kleiner als 10, so haben wir einen zulässigen Rateversuch, der ab AUSWTG ausgewertet wird. Andernfalls muß die gespeicherte Sequenz dem Spieler vorgeführt werden, dies macht die Routine VORFRN.

### *Routine VORFRN*

Es ist also eine Buchstabentaste gedrückt worden, die BMI-Verzweigung hat nicht stattgefunden. Der VORFRN-Programmteil zeigt nun „in Bild und Ton" die abgespeicherte Sequenz. Immer wenn dieser Programmteil angesprungen wird, beginnt auch eine neue Ratephase, die entsprechenden Variablen müssen also auf 0 zurückgestellt werden:

```
VORFRG          LDX #0
                STX VRSNR
                STX FEHLER      Variable zurücksetzen
```

Der erste Tabellenwert wird nun geholt, die passende LED erleuchtet und die passende Note gespielt:

```
VFGSCHL   LDA TABLE,X    Xten Tabellenwert holen
          STX TEMP       X aufbewahren
          JSR LICHT      LED laut TABLE,X an
          JSR SPIEL      Ton laut TABLE,X an
```

Es folgt eine Verzögerung, Zähler ist Y. Zwei Blindbefehle verlängern die Verzögerungszeit:

```
          LDY #$FF
DELAY     ROR DAUER
          ROL DAUER      Blindbefehle
          DEY            herunterzählen
          BNE DELAY      Test auf Schleifenende
```

Jetzt ist die nächste Note in der aktuellen Tabelle an der Reihe. Der Indexzeiger wird geholt und inkrementiert:

```
          LDX TEMP       X wiederholen ...
          INX            ... und um 1 erhöhen
```

Es muß nun überprüft werden, ob die Sequenz komplett wiedergegeben wurde (dann geht es zurück zur Marke TASTE) oder nicht (dann geht die „Aufführung" bei Marke VFGSCHL weiter):

```
          CPX DIGITS     Sequenz vollständig?
          BNE VFGSCHL
          BEQ TASTE      Fertig: nächste Eingabe holen
```

***Routine AUSWTG***

Dieser Programmteil wertet aus, ob der Spieler richtig geraten hat oder nicht; sehen wir uns das genauer an.
Der Tabellenwert für die aktuelle Sequenzposition wird mit der Eingabe des Spielers verglichen:

```
AUSWTG    LDX VRSNR      Versuchszahl nach X holen
          CMP TABLE,X    mit tatsächlichem Wert vergleichen
          BEQ RICHTG     wenn richtig: anzeigen
```

Stimmen Spielerglaube und Wirklichkeit überein, geht es nach RICHTG, andernfalls selbstverständlich nach FALSCH. Betrachten wir diesen Fall zuerst. Zunächst wird die Fehlerzahl aktualisiert,

```
FALSCH    INC FEHLER
```

und dann erklingt ein tiefer Ton:

```
        LDA #$80
        STA DAUER
        LDA #$FF
        JSR SPLTON      „intonieren“
```

Danach erfolgt ein Sprung nach ENDTST:

```
        BEQ ENDTST      auf Spielende testen
```

**Übung 8-3:** *Was ist mit obigem BEQ-Befehl? Führt er immer zu Adresse ENDTST? (Überprüfen Sie, ob die Z-Flagge an dieser Stelle gesetzt ist oder nicht.)*

**Übung 8-4:** *Worin besteht der Vorteil dieses BEQ-Befehls gegenüber einem JMP-Befehl?*

Nun der Fall, daß der Spieler richtig geraten hat: Die entsprechende LED muß erleuchtet und der zugehörige Ton gespielt werden. Beide Unterprogramme erwarten die richtige Zahl im Akkumulator.

```
RICHTG  JSR LICHT       LED an
        JSR SPIEL       Ton dazu
```

Wieder müssen wir überprüfen, ob das Ende der Sequenz erreicht ist oder nicht. Die Versuchzahl wird also mit der Gesamtlänge der Sequenz verglichen:

```
ENDTST  INC VRSNR       noch ein Versuch
        LDA DIGITS
        CMP VRSNR       alle Positionen durch?
        BNE TASTE       wenn nicht: nächste Taste
```

War die Sequenz noch nicht vollständig, geht es zurück nach TASTE. Andernfalls sind wir fertig und müssen den Spieler über sein Abschneiden informieren. Dazu wird die Fehlerzahl abgefragt:

```
        LDA FEHLER      Fehlerzahl holen
        CMP #0          Kein Fehler?
        BEQ SIEG        wenn kein Fehler: Spieler gewinnt
```

Marke SIEG folgt weiter unten. Hat der Spieler verloren, geht es mit VRLST weiter:

```
VRLST   JSR LICHT       Fehlerzahl anzeigen
```

Die Fehlerzahl erscheint in Form von Aufleuchten der entsprechenden LED. Sie erinnern sich, daß der Inhalt von FEHLER beim Eintritt in diesen Programmteil noch im Akkumulator steht.
Es erklingt jetzt eine abwärtsfallende Achttonfolge. Die noch zu spielenden Töne werden im Stapelspeicher aufbewahrt.

```
            LDA #9          8 fallende Töne spielen
VLSTSCHL    PHA             A auf den Stapel
            JSR SPIEL       Ton spielen
            PLA             A zurückholen
```

Ist ein Ton gespielt, wird die noch zu spielende Zahl von Tönen um 1 vermindert und auf 0 überprüft:

```
            SEC             C-Flagge für Subtraktion setzen
            SBC #1          1 subtrahieren
            BNE VLSTSCHL
```

**Übung: 8-5:** *Beachten Sie die Verwendung des Stapelspeichers als Zwischenspeicher. Können Sie eine entsprechende Vorgehensweise ohne Benutzung des Stapels vorschlagen?*

**Übung 8-6:** *Erläutern Sie die Vor- und Nachteile der Stapelbenutzung gegenüber anderen Techniken zur Schaffung von Arbeitsspeicherplatz. Ist die Stapelbenutzung mit irgendwelchen Risiken verbunden?*

Nun werden acht aufeinanderfolgende Töne gespielt, und die Variablen VRSNR und FEHLER werden auf 0 zurückgesetzt. Dann erfolgt der Rücksprung zum Programmanfang:

```
            STA VRSNR       Variable auf 0
            STA FEHLER
            BEQ TASTE       nächste Ratesequenz holen
```

Betrachten wir nun die Sieg-Situation. Es gehen gleichzeitig alle Lichter an:

```
SIEG        LDA #$FF        Sieg: „Festbeleuchtung“
            STA TOR1A
            STA TOR1B
```

Und es erklingt eine aufsteigende Folge von acht Tönen. Die Tonnummer im Akkumulator dient während der Spiel-Routine als Index, und wieder wird der obere Teil der Stapelspeicher als Arbeitsspeicher benutzt:

```
            LDA #1          A steigt bis 9
SIEGSCHL    PHA             am Stapel zwischenspeichern
            JSR Spiel
            PLA
```

Die Zahl der gespielten Töne erhöht sich um 1 und wird auf den Endwert 9 getestet:

```
CLC          Carry für Addition löschen
ADC #1
CMP #10
```

Ist Endwert 9 noch nicht erreicht, geht die Schleife bei SIEGSCHL weiter,

```
BNE SIEGSCHL
```

oder aber ein neues Spiel beginnt:

```
BEQ STRTJP   Doppelsprung über STRTJP nach
             START
```

Nach dem Hauptprogramm wollen wir nun auch die drei Unterprogramme näher untersuchen.

**Die Unterprogramme**

***Unterprogramm LICHT***

Dieses Unterprogramm erwartet im Akkumulator die Nummer der LED, die es erleuchten soll. Dies geschieht wie immer dadurch, daß eine 1 an die richtige Stelle im Akkumulator geschoben wird, dessen Inhalt anschließend zum richtigen Ausgabetor geschickt wird: für LEDs 1 bis 8 Tor 1A, für LED 9 Tor 1B. Die Anzahl der Verschiebungen des 1-Musters bis an die richtige Position entspricht der jeweiligen LED-Nummer. Als Zähler für die Verschiebungen dient das Y-Register. Die LED-Nummer wird am Beginn des Unterprogramms im Stapelspeicher abgelegt und vor dem Rücksprung dort wieder abgeholt. Dies ist die klassische Methode, den Inhalt eines wichtigen Registers während eines Unterprogramms aufzubewahren und beim Rücksprung wieder parat zu haben. Würde man anders programmieren, müßte hier das LICHT aufrufende Programm den Akkumulator vorher explizit zwischenspeichern, und vor dem Aufruf beispielsweise des SPIEL-Unterprogramms müßte der Akkumulator wieder neu geladen werden. Da LICHT und SPIEL normalerweise nacheinander aufgerufen werden, ist es effizienter, den Akkumulator vom Unterprogramm aufbewahren zu lassen. Tun wir das also:

```
LICHT   PHA          Akkumulator aufbewahren
```

Jetzt wird der Verschiebungszähler gesetzt

```
        TAY          Y wird Zählregister
```

und der Akkumulator auf 0:

```
        LDA #0       A löschen
```

Sollte LED 9 an gewesen sein, wird gelöscht:

```
        STA TOR1B
```

Es folgt die Verschiebungsschleife: Das Carrybit wird gesetzt und entsprechend weit in den Akkumulator geschoben:

```
        SEC             Carry setzen
LNKSHFT ROL A
        DEY
        BNE LNKSHFT
```

Das adäquate Bitmuster ist jetzt im Akkumulator und wird zum Spielbrett gesendet:

```
        STA TOR1A
```

Nun muß wieder der Spezialfall von LED 9 berücksichtigt werden: Soll diese LED angesprochen werden, so ist der Akkumulator zum jetzigen Zeitpunkt 0 und das Carrybit gesetzt. Das muß gepüft werden,

```
        BCC LTCC        Bit 9 gesetzt?
```

und wenn es der Fall ist, geht der Wert 1 an Tor 1B:

```
        LDA #1
        STA TOR1B       LED 9 an
```

Vor dem Rücksprung wird der Akkumulator wie angedeutet vom Stapel zurückgeholt:

```
LTCC    PLA             A zurückholen
        RTS
```

**Übung 8-7:** *Führen Sie auf, wie die Register durch dieses Unterprogramm verändert werden.*

**Übung 8-8:** *Welche Programmänderungen sind erforderlich, wenn das Y-Register beim Verlassen dieses Unterprogrammes unverändert sein soll?*

***Unterprogramm RANDOM***

Dieses Unterprogramm erzeugt eine neue Zufallszahl und bringt sie im Akkumulator zurück. Eine Beschreibung fand in Kapitel 4 statt.

***Unterprogramm SPIEL***

Diese Routine spielt im Normalfall den Ton, der im Akkumulator repräsentiert wird. Sie kann aber auch durch Einsprung bei SPLTON einen Ton erzeugen,

dessen Frequenz im Akkumulator enthalten ist und dessen Dauer in DAUER fixiert ist. Schauen wir sie uns an.

Als Index für den Zugriff auf die beiden Tabellen mit den Tonhöhen und -längen dient das Y-Register. Bis zu neun Töne können gespielt werden, entsprechend den LEDs und Tasten 1 bis 9:

```
SPIEL   TAY     Tonnummer wird Index
        DEY     auf internen Wert justieren
```

Die Korrektur des Index ist deshalb nötig, weil die 0. Eintragung der Tastennummer 1 entspricht (usw.). Mittels indizierter Adressierung werden nun die richtigen Werte aus den Tabellen DURTAB und NOTAB entnommen und in DAUER und FREQ abgelegt:

```
        LDA DURTAB,Y    Tondauer holen
        STA DAUER       ablegen
        LDA NOTAB,Y     Frequenz holen
SPLTON  STA FREQ        ablegen
```

Dann wird der Lautsprecher abgestellt:

```
        LDA #0
        STA TOR3B       Lautsprecher 3B setzen
```

Zwei Schleifen werden jetzt implementiert. Eine Innenschleife, gezählt vom Y-Register, sorgt für die richtige Frequenz; eine Außenschleife, gezählt von X, sorgt für die gewünschte Tondauer. Zuerst werden die beiden Zähler geladen:

```
        LDX DAUER       Anzahl der Halbzyklen
FL2     LDY FREQ        Frequenzkonstante
```

Es folgt die Innenschleife:

```
FL1     DEY
        CLC             Zeit „verschwenden“
        BCC +2
        BNE FL1         Verzögerungsschleife
```

Beachten Sie, daß die zwei „Nichtstun“-Befehle innerhalb der Schleife dazu dienen, eine längere Verzögerung zu erzielen. Nach der Innenschleifenverzögerung werden die Inhalte der Lautsprecher-Ausgangstore komplementiert, um einen Rechteckimpuls zu generieren (beachten Sie auch den Komplementierbefehl EOR #$FF):

```
        EOR #$FF        Ton komplementieren
        STA TOR3B
```

Dann wird die Außenschleife komplettiert:

```
DEX
BNE FL2          Außenschleife
RTS
```

## ZUSAMMENFASSUNG

Dieses Programm zeigt, wie einfach elektronische Tastenspiele zu programmieren sind, die bei der Ein/Ausgabe Töne erzeugen und durchaus auch Erwachsene fordern können.

**Übung 8-9:** *Die Tondauer- und Frequenzkonstanten sind in Bild 8.6 aufgelistet. Was sind die tatsächlichen Frequenzen, die dieses Programm erzeugt?*

| NOTE | FREQUENZ KONSTANTE | TONDAUER KONSTANTE |
|---|---|---|
| 1 | C9 | 6B |
| 2 | BE | 72 |
| 3 | A9 | 80 |
| 4 | 96 | 8F |
| 5 | 8E | 94 |
| 6 | 7E | AA |
| 7 | 70 | BF |
| 8 | 64 | D7 |
| 9 | 5E | E4 |

*Abb. 8.6: Frequenz- und Tondauerkonstanten*

# 9

# Verwendung von Interrupts (Hirnverdreher)

## EINFÜHRUNG

Mit dem Unterbrechungs-Zeitgeber des 6522 VIA (ein gebräuchlicher Ein/Ausgabe-Chip des 6502) werden Programmunterbrechungen - Interrupts - erzeugt. Dieser programmierbare Unterbrechungszeitgeber wird im Freilaufmodus benutzt und erzeugt Wellenformen.

## DIE SPIELREGELN

Die Grundidee dieses Spiels ist die des INVICTA-Spiels „Superhirn". Der Computer „denkt" sich eine Ziffernfolge aus - etwa eine fünfstellige Zahl aus Ziffern zwischen 0 und 9 - und der Spieler versucht, diese Zahl zu erraten. Dabei wird jeder Rateversuch vom Computer „kommentiert", indem die Anzahl richtiger Ziffern an der richtigen Stelle und die Anzahl richtiger Zahlen an falscher Stelle angezeigt werden.

Die Antwort des Computers erscheint in den LEDs 1 bis 9. Eine blinkende LED bezeichnet dabei einen „Volltreffer", also richtige Ziffer an richtiger Position, und eine konstant leuchtende LED kennzeichnet einen „Treffer", eine richtige Ziffer an falscher Position. Mehrere Spieler können dabei miteinander konkurrieren, wobei der gewinnt, der für die Entschlüsselung einer Zahl bestimmter Länge die wenigsten Versuche benötigt.

Man kann das Spiel auch mit „Handicaps" spielen, indem ein Spieler eine Zahl mit n Ziffern raten muß und der andere eine Zahl mit n-1 Ziffern. Dies ist schon ein nennenswertes Handicap, denn der Schwierigkeitsgrad steigt mit zunehmender Stellenzahl rapide.

## EIN TYPISCHER SPIELVERLAUF

Das Spiel wird sowohl von visuellen als auch von akustischen Signalen begleitet.

### Akustische Signale

Jedesmal wenn der Spieler seine Versuchszahl eingegeben hat, antwortet der Computer mit einem tiefen Ton, wenn die eingegebene Zahl falsch ist, und mit einen hohen Ton, wenn sie richtig ist.

### Visuelle Signale

Zu Beginn jedes Spieles fordert LED 10 die Eingabe der gewünschten Länge der zu ratenden Zahl (Bild 9.1) und wartet auf eine Zahl zwischen 1 und 9. Andere Eingaben werden ignoriert.

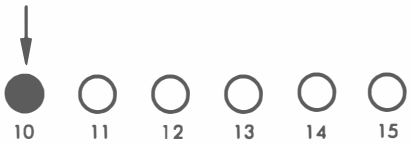



*Abb. 9.1: Sequenzlänge eingeben*

Nachdem die Länge festgelegt ist, beispielsweise 2, leuchtet LED 11 auf und fordert den Spieler auf, seine Versuchszahl einzugeben (Bild 9.2). Der Spieler gibt daraufhin in unserem Beispiel zwei Ziffern nacheinander ein. Machen wir also ein kleines Spiel.

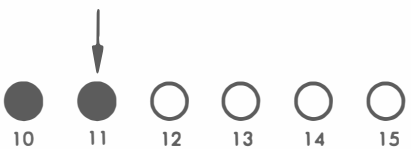



*Abb. 9.2: Versuchszahl eingeben*

Der Spieler gibt die Ziffernfolge 12 ein. Ein tiefer Ton wird hörbar, LEDs 10 und 11 gehen für kurze Zeit aus, aber sonst geschieht nichts weiter (Bild 9.3).

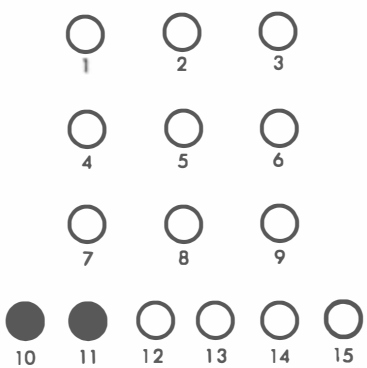



*Abb. 9.3: Fehlversuch*

Da LEDs 1 bis 9 dunkel bleiben, gibt es also weder Treffer noch Volltreffer. Weder 1 noch 2 kommt also in der Zahl des Computers vor. Versuchen wir es weiter.

Wir geben 34 ein. Wieder erklingt ein tiefer Ton, aber diesmal bleibt LED 1 stetig erleuchtet (Bild 9.4). Wir wissen nun, daß entweder 3 oder 4 vorkommt, allerdings in anderer Position. Anders ausgedrückt: Wenn wir als nächstes 43 eingeben, müßten wir eigentlich ein Volltreffer-Signal bekommen. Machen wir die Probe.

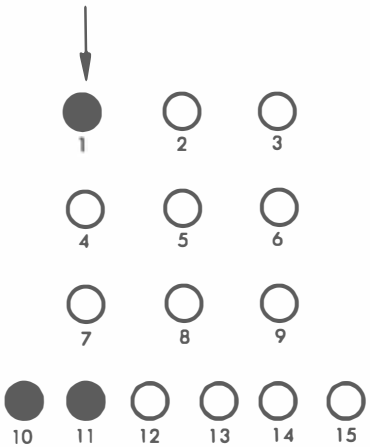



*Abb. 9.4: Volltreffer*

Nach Eingabe von 43 kommt zwar wieder der tiefe Ton (43 ist nicht richtig), aber diesmal blinkt LED 1. Unsere Überlegung war also richtig. Wir versuchen als nächstes 45. Ein hoher Ton erklingt, und LEDs 1 und 2 leuchten kurz auf. Wir haben die Zahl richtig erraten und unser erstes Spiel gewonnen.

Das Spiel ist zu Ende, und es erscheint wieder LED 10 wie in Bild 9.1. Zu bemerken ist noch, daß die Eingabe eines Wertes, der nicht zwischen 1 und 9 liegt, das Spiel von vorn beginnen läßt.

Es gilt, eine Besonderheit bei diesem Spiel zu beachten: Wenn die zu erratende Zahl mehrmals die gleichen Ziffern enthält, und der Spieler hat eine solche Ziffer in seinem Versuch in einer Volltrefferposition, so kennzeichnet das Programm sowohl den Treffer als auch den Volltreffer.

## DER ALGORITHMUS

Das Flußdiagramm für dieses Programm ist in Bild 9.5 dargestellt. Die Interrupts werden automatisch vom programmierbaren Zeitgeber des VIA 1 etwa jede Fünfzehntelsekunde erzeugt.

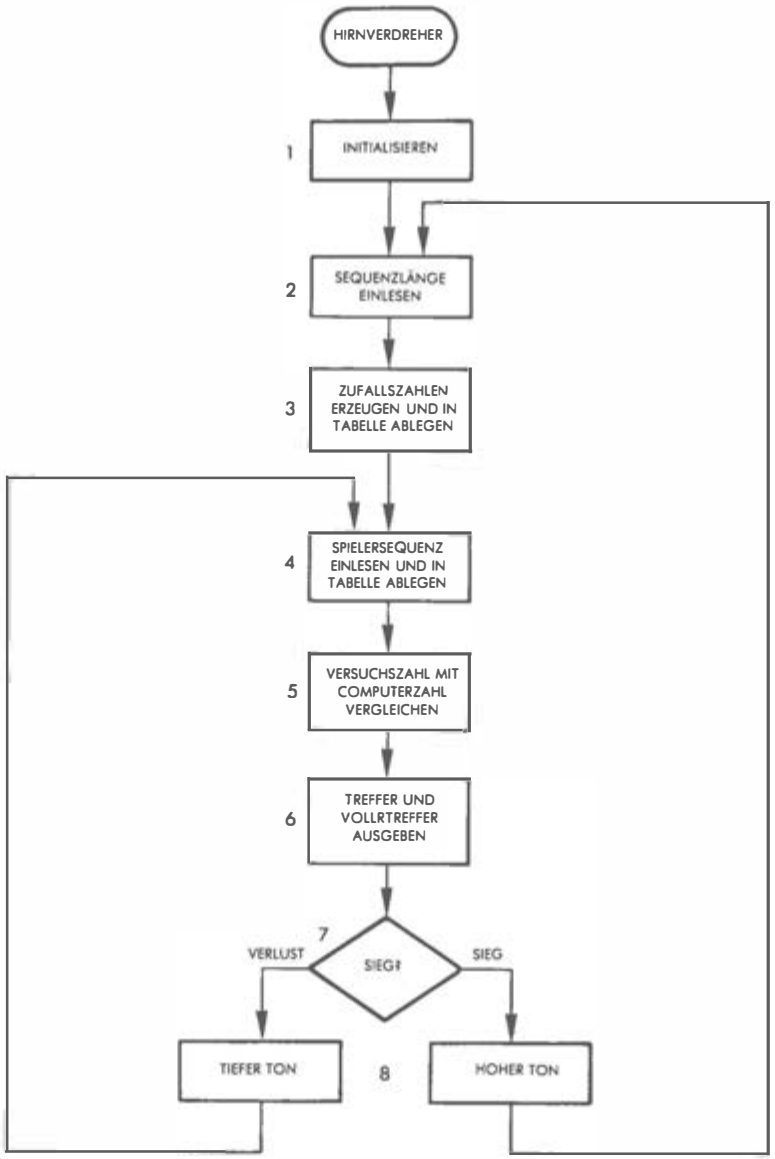



*Abb. 9.5: Flußdiagramm HIRNVERDREHER*

Wie in Bild 9.5 zu sehen ist, werden zunächst alle erforderlichen Register und Speicherstellen initialisiert. Dann (Kasten 2) wird die gewünschte Zahlenlänge von der Tastatur eingelesen, wobei die zulässigen Grenzwerte 1 und 9 wie im letzten Kapitel „herausgefiltert" werden. Als nächstes wird nun eine Zufallszahlenfolge erzeugt (Kasten 3) und in einer Zifferntabelle ab Adresse COM0 abgelegt.

Bei Kasten 5 wird die Computerzahl ziffernweise mit der Versuchszahl des Spielers verglichen. Der Algorithmus nimmt der Reihe nach eine Ziffer nach der anderen aus seiner Zahl und stellt sie der entsprechenden Ziffer in der Versuchszahl gegenüber. Wie bereits angedeutet, können dabei zwei LEDs für dieselbe Ziffer aufleuchten, dann nämlich, wenn diese Ziffer zwei oder mehrere Male in der zu ratenden Ziffernfolge vorkommt. Sie wird in diesem Fall u.U. als Treffer und als Volltreffer angezeigt.

Es ist zu bemerken, daß auch ein anderer Vergleichsalgorithmus eingesetzt werden könnte, bei dem jede Ziffer der Spielerzahl nacheinander mit jeder Ziffer der Computerzahl verglichen wird.

Nach dem Vergleichen der Ziffern wird das Ergebnis bekanntgegeben (Kasten 6) und anschließend auf eine Gewinnsituation hin getestet (Kasten 7). Schließlich wird der passende Begleitton erzeugt (Kasten 8).

## DAS PROGRAMM

### Datenstrukturen

Die (zu ratende) Computerzahl und die (eingegebene) Spielerzahl werden in zwei Tabellen festgehalten, die ab Adressen COM0 und SPI0 gespeichert werden (Bild 9.6).

### Die Variablen

Die Speicherseite 0 (zero-page) wird wie üblich dazu benutzt, zusätzliche Arbeitsregister zur Verfügung zu stellen. Wie die Variablen in der 0-Seite organisiert sind, zeigt Bild 9.6. Die ersten neun Adressen enthalten die Programmvariablen, deren Funktionen im Bild angedeutet sind und die im weiteren Verlauf genau beschrieben werden. Adressen 09 bis 0E sind für die Erzeugung der Zufallszahlen reserviert. Die COM0-Tabelle mit der Computerzahl ist von 0F bis 17 gespeichert, die SPI0-Tabelle mit der Speicherzahl ab Adresse 18.

Der Speicherbereich für die Ein/Ausgabe-Adressierung und Unterbrechungssteuerung ist in Bild 9.7 dargestellt. Über A000 bis A005 werden die Tore A und B des VIA 1 und der Zeitgeber T1 adressiert. Bild 9.8 zeigt die Registeraufteilung beim 6522 VIA.

Über Adresse A00B wird das Hilfskontrollregister angesprochen, über A00B das Unterbrechungszulassungsregister (interrupt-enable). Eine detaillierte

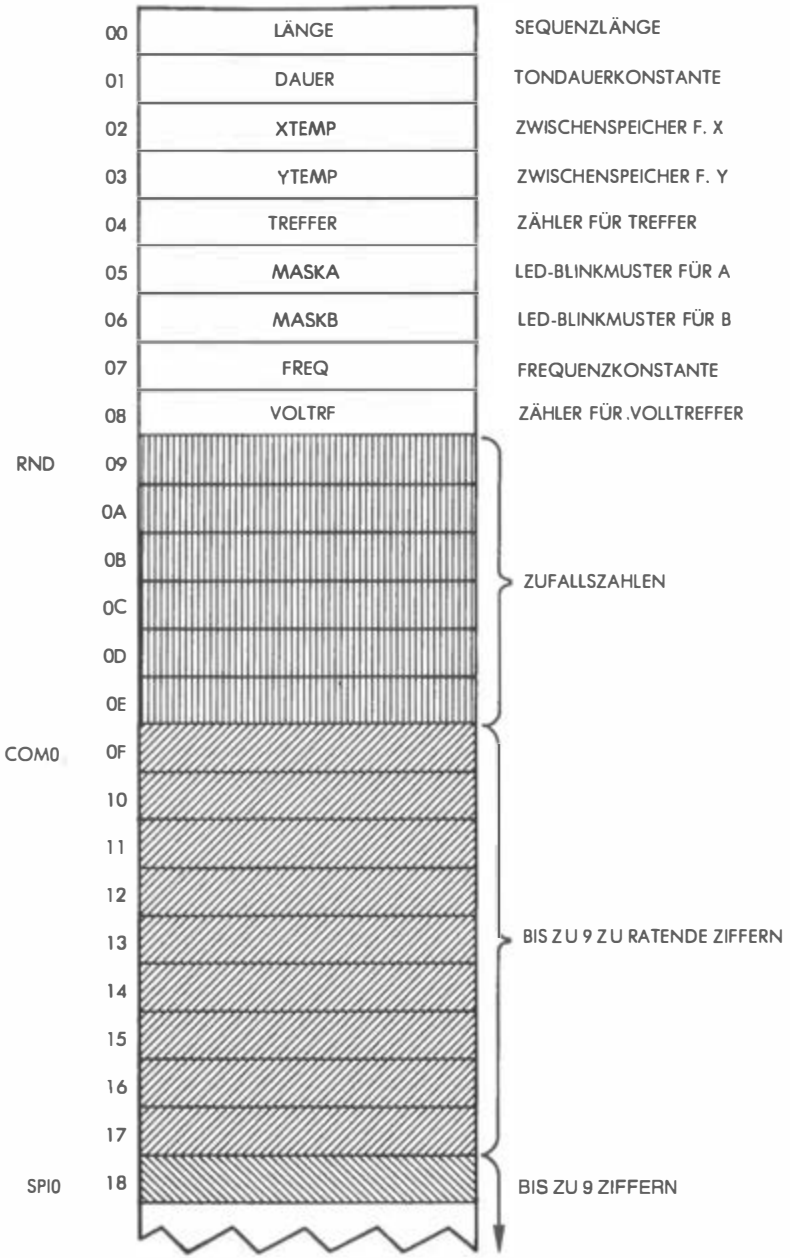



*Abb. 9.6: Unterer Speicherbereich*

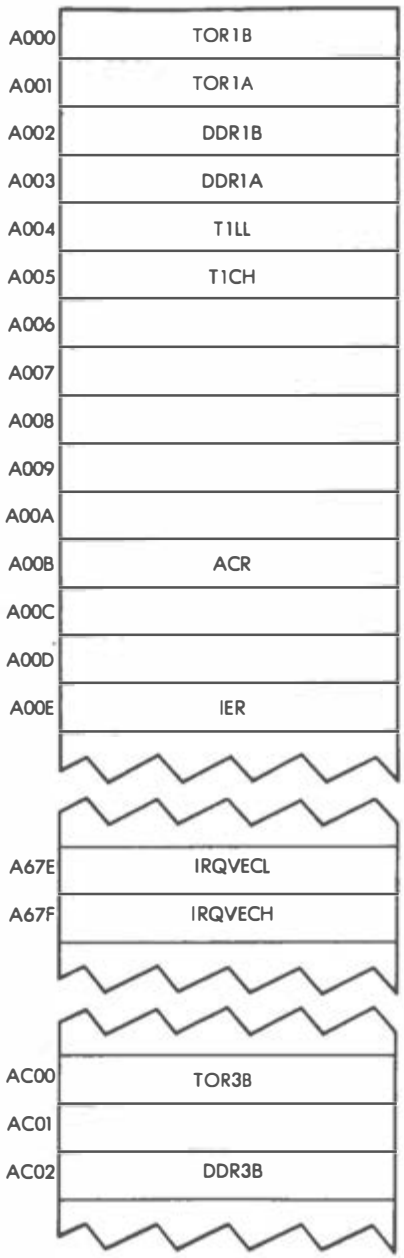



*Abb. 9.7: Oberer Speicherbereich*

Beschreibung dieser Register findet sich in den „6502 Anwendungen“ (Ref-Nr. 3014 von SYBEX).

Adressen A67E und A67F werden zum Setzen des Unterbrechungsvektors benutzt: Hier befindet sich die Startadresse für die Unterbrechungsroutine, in unserem Fall ist das Adresse 03EA. Diese Routine handhabt das Blinken der LEDs und wird weiter unten beschrieben. Tor 3 schließlich wird über die Adressen AC00 und AC02 angesprochen.

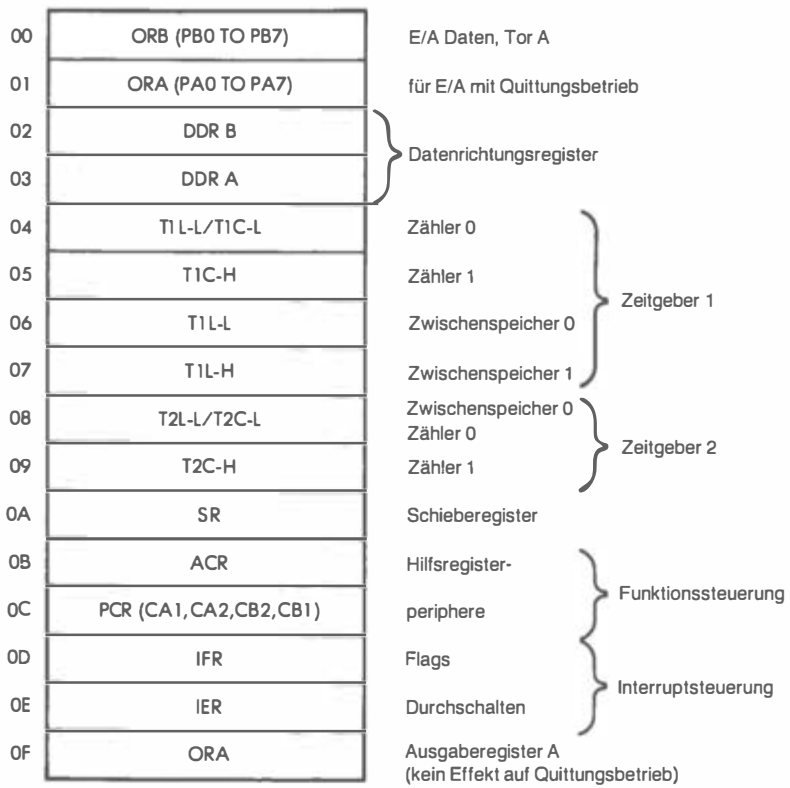



*Abb. 9.8: 6522 VIA Speicheraufteilung*

### Implementierung des Programms

Ein detailliertes Flußdiagramm des „Hirnverdreher“-Programms ist in Bild 9.9 wiedergegeben. Nehmen wir das Programm nun genau unter die Lupe (Programm-Listing in Bild 9.13).

Der Initialisierungsteil liegt im Speicherbereich 0200 bis 0239 (hexadezimal) und konditioniert Unterbrechung und Ein/Ausgabe. Vor der Modifizierung des Unterbrechungsvektors (in Adressen A67E und A67F, siehe Bild 9.7) muß der Zugang zu diesem geschützten Speicherbereich ermöglicht werden. Dies geschieht mit dem Unterprogramm ACCESS, das zum SYM-Monitor gehört:

```
JSR ACCESS
```

Jetzt kann der Unterbrechungsvektor geladen werden, Adresse IRQVEC erhält den Wert 03EA:

```
LDA #$EA        niederwertiges Byte
STA IRQVECL
LDA #$03        höherwertiges Byte
STA IRQVECH
```

Für die Unterbrechungskontrolle müssen nun die internen Register des 6522 VIA gesetzt werden. Das Unterbrechungszulassungsregister (IER) läßt Interrupts zu oder nicht, wobei jedes IER-Bit einem Bit im Unterbrechungsflaggenregister (IFR) entspricht. Immer wenn ein Bit 0 ist, wird der entsprechende Interrupt gesperrt. Bit 7 des IER spielt eine besondere Rolle (Bild 9.10): Wenn es 0 ist, löscht jede 1 der übrigen Bitpositionen des IER die entsprechende Zulassungsflagge; wenn es 1 ist, setzt jede 1 normal die korrespondierende Zulassungsflagge. Wird Bit 7 des IER also auf 0 und alle anderen Bits auf 1 gesetzt, so sind alle Unterbrechungen gesperrt:

```
LDA #$7F
STA IER
```

Als nächstes wird Bit 6, das zum Interrupt des Zeitgebers 1 gehört, auf „zulassen" gesetzt. Dazu muß neben Bit 6 auch Bit 7 des IER auf 1 gesetzt werden:

```
LDA #$C0
STA IER
```

Jetzt erhält Zeitgeber 1 den Freilaufmodus (wir erinnern uns, daß ja auch ein Einzelbetrieb möglich ist). Für den Modus sind Bits 6 und 7 des Hilfskontrollregisters verantwortlich (siehe Bild 9.11). In unserem Fall wird Bit 7 zu 0 und Bit 6 zu 1:

```
LDA #$40
STA ACR
```

Ehe der Zeitgeber im Ausgabemodus verwendet werden kann, muß sein Zählregister einen 16-Bit-Wert erhalten, der die Dauer des zu erzeugenden

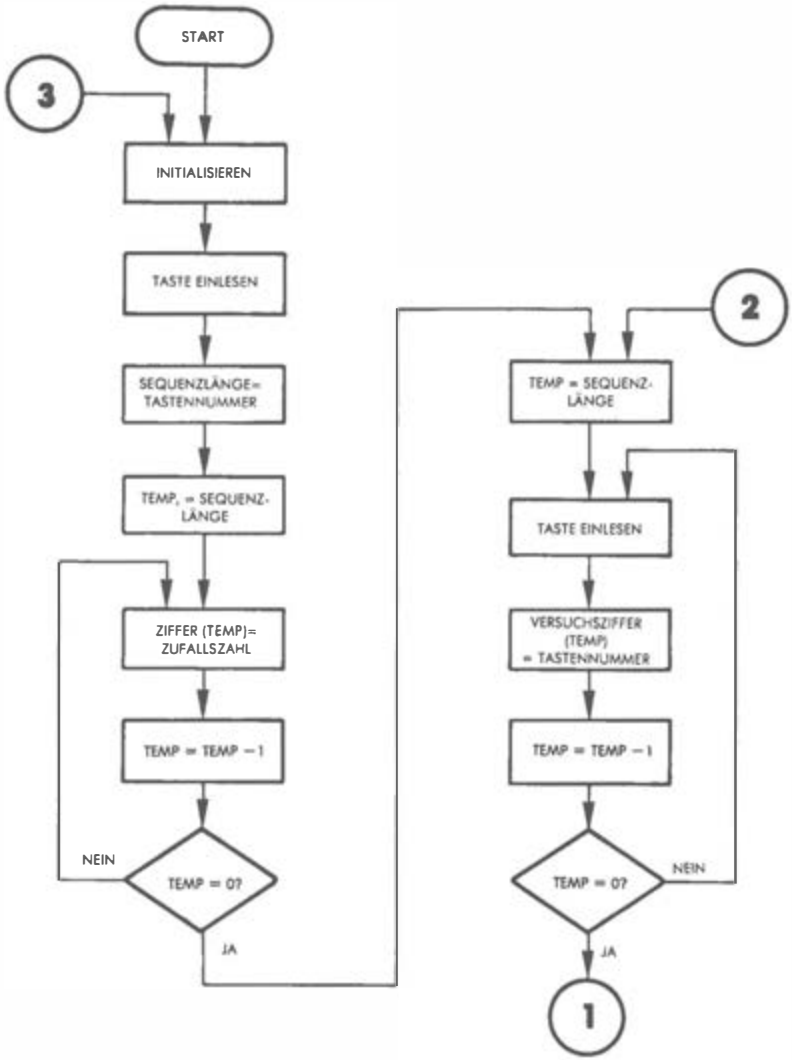



*Abb. 9.9: Detailliertes Flußdiagramm HIRNVERDREHER*

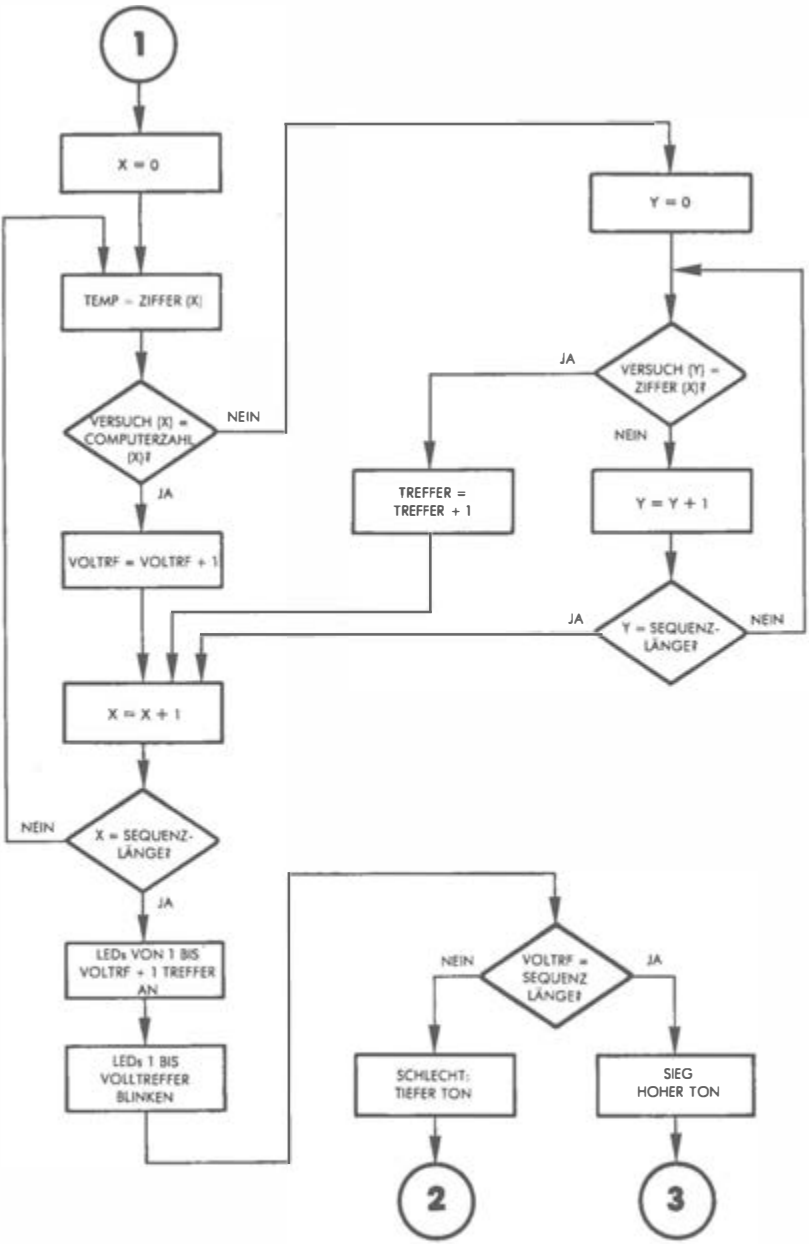



*Abb. 9.9: Detailliertes Flußdiagramm HIRNVERDREHER (Fortsetzung)*

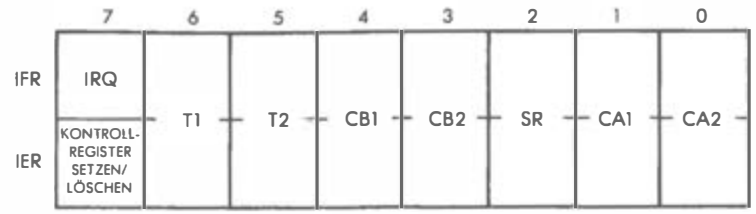



*Abb. 9.10: Interrupt-Register*

Rechtecksignals spezifiziert. Wir nehmen hier den Maximalwert FFFF:

```
LDA #$FF
STA T1LL
STA T1CH
```

Die tatsächlich von Zeitgeber 1 ausgesendete Signalform ist in Bild 9.12 dargestellt. Um die genaue Impulsdauer zu bestimmen, machen wir uns klar, daß, je nach dem Anfangswert n im Zählregister, der Wert zwischen n+1.5 und n+2 Zyklen pendeln wird.

Jetzt werden die Unterbrechungen zugelassen,

```
CLI
```

und die drei vom Programm benutzten Ausgangstore gerichtet:

```
STA DDR1A      Ausgabe
STA DDR1B      Ausgabe
STA DDR3B      Ausgabe
```

| ACR7 OUTPUT ENABLE | ACR6 INPUT ENABLE | MODUS |
|---|---|---|
| 0 | 0 EINZEL BETRIEB | „TIME OUT" – UNTERBRECHUNG WENN T1 DURCH PB7 GESPERRT |
| 0 | 0 FREILAUF | BLEIBENDE UNTERBRECHUNG PB7 GESPERRT |
| 1 | 1 EINZEL BETRIEB | UNTERBRECHUNG UND AUSGANGSSIGNAL AUF PB7 WENN T1 GELADEN = EINZELBETRIEB UND IMPULS MIT PROGRAMMIERBARER BREITE |
| 1 | 1 FREILAUF | BLEIBENDE UNTERBRECHUNG UND RECHTECK-SIGNAL AUF PB7 |

*Abb. 9.11: Arbeitsmodus von Zeitgeber 1, bestimmt durch 6522 Hilfsregister*

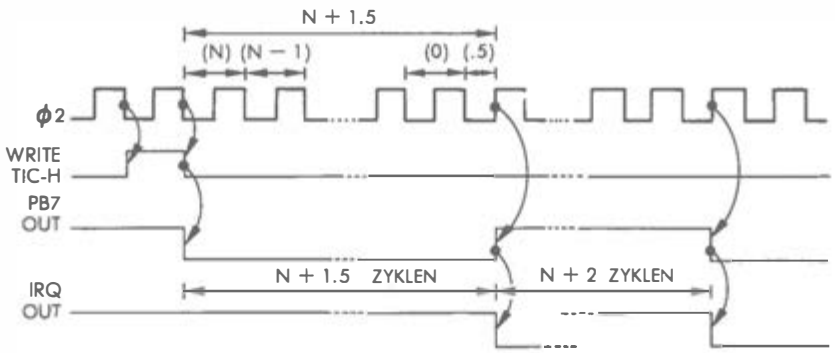



*Abb. 9.12: Zeitgeber 1 im Freilaufmodus*

Alle LEDs werden gelöscht,

```
TAST1   LDA #0
        STA TOR1A
        STA TOR1B
```

und die Blinkmasken auf 0 initialisiert:

```
        STA MASKA
        STA MASKB
```

LED 10 leuchtet jetzt auf und gibt dem Spieler zu verstehen, daß er die Länge der zu ratenden Zahl eingeben soll:

```
        LDA #%00000010   LED 10 ...
        STA TOR1B        ... an
```

Unsere GETKEY-Routine holt die gedrückte Taste:

```
        JSR GETKEY       Zahlenlänge holen
```

Die Tasteneingabe muß auch hier „gefiltert" werden: Alle Eingaben kleiner 1 und größer 9 werden ignoriert:

```
        CMP #10
        BPL TAST1
        CMP #0
        BEQ TAST1
```

Eine im zugelassenen Rahmen liegende Zahlenlänge wird in LÄNGE abgespeichert:

```
STA LÄNGE
```

Eine Zufallszahlenreihe wird nun erzeugt.

***Erzeugung einer Zufallszahlenreihe***

Die erste Zufallszahl, mit der der Generator in Gang gesetzt wird, wird dem Zähler entnommen. Die Theorie zu dieser Vorgehensweise ist bereits beschrieben worden. Die „Urzahl" geht in die Speicherstellen RND+1, RND+4 und RND+5:

```
LDA T1LL
STA RND+1
STA RND+4
STA RND+5
```

Die zu ratende Zufallsfolge wird von dem Unterprogramm RANDOM zusammengesetzt:

```
        LDY LÄNGE       Zahlenlänge holen
        DEY             bis auf 0 zählen
RAND    JSR RANDOM      Wert holen
```

Die Zahl muß nun in den BCD-Kode übersetzt werden, damit wir einen Wert zwischen 0 und 9 bekommen (vgl. Kapitel 8),

```
SED
ADC #00         Dezimalwert
CLD
```

und die obere Bytehälfte wird „gekappt":

```
AND #%00001111
```

Jetzt haben wir eine brauchbare Zahl, sie wird in der Tabelle am richtigen Platz abgelegt, indem wir das Y-Register als Index benutzen:

```
STA COM0,Y
```

Y wird dekrementiert, bis die Zahl die gewünschte Länge hat:

```
DEY
BPL RAND
```

***„Sammeln" der Spielerziffern***

Der laufende Zeiger auf die SPI0-Tabelle, in der die Spieler-Eingaben zusam-

mengetragen werden, ist das X-Register. Er wird auf 0 initialisiert und in XTEMP aufbewahrt:

```
EINGABE   LDA #0          Zeiger auf 0
          STA XTEMP
```

LEDs 10 und 11 werden nun eingeschaltet, um dem Spieler die Bereitschaft zur Entgegennahme seiner Versuchszahl anzuzeigen:

```
          LDA #%00000110
          ORA TOR1B
          STA TOR1B
```

Die GETKEY-Routine holt wie üblich die Eingabe ab:

```
TAST2     JSR GETKEY
```

Ist die eingegebene Zahl größer als 9, wird dies als Aufforderung zum Neubeginn des Spiels interpretiert:

```
          CMP #10
          BPL TAST1
```

Andernfalls können wir uns den Zeigerwert für die aktuelle Tabellenposition aus XTEMP ins X-Register holen und die Spielerziffer passend abspeichern:

```
          LDX XTEMP
          STA SPI0,X      Ziffer ablegen
```

Der Zeiger wird inkrementiert und wandert zurück:

```
          INX
          STX XTEMP
```

Jetzt wird er mit der Gesamtlänge verglichen, um zu sehen, ob weitere Eingaben kommen. Wenn das der Fall ist, geht es zurück zu TAST2:

```
          CPX LÄNGE       alle Ziffern eingegeben?
          BNE TAST2       wenn nicht: nächste Ziffer holen
```

Die Versuchszahl des Spielers ist komplett, und sie muß nun mit der vom Computer „erfundenen" Ziffernfolge verglichen werden. Für den Fall, daß der Spieler die richtige Lösung hat, müssen zuvor die LEDs und Masken gelöscht werden:

```
          LDX #0
          STX TOR1A
          STX TOR1B
          STX MASKA
          STX MASKB
```

Die zwei Speicher, in denen die Anzahl von Treffern und Volltreffern summiert werden, erhalten den Anfangswert 0:

```
          STX TREFFER     Zahl der Treffer
          STX VOLTRF      Zahl der Volltreffer
```

Nun werden der Reihe nach alle Werte der COM0-Tabelle mit allen Werten der SPI0-Tabelle verglichen:

```
ZIFSCHL   LDA COM0,X
          CMP SPI0,X
```

Bei Nichtübereinstimmung haben wir keinen Volltreffer. Um eventuelle Treffer zu finden, müssen wir die aktuelle Ziffer noch mit den anderen SPI0-Werten vergleichen, das geschieht ab TREFSUCH:

```
          BNE TREFSUCH
```

Andernfalls haben wir einen Volltreffer, der entsprechende Zähler wird erhöht, dann kommt die nächste Ziffer:

```
          INC VOLTRF
          BNE NCHSTZIF
```

Betrachten wir nun den Fall „nicht Volltreffer". Die gerade bearbeitete Ziffer der Computersequenz ist noch im Akkumulator. Sie wird nacheinander den Ziffern der Spielersequenz gegenübergestellt, Y dient dabei als laufender Zeiger in die SPI0-Tabelle:

```
TREFSUCH  LDY #0
TREFSCHL  CMP SPI0,Y
          BNE NCHSTREF
```

Wird eine Übereinstimmung gefunden, so erhöht sich TREFFER, und die nächste Ziffer ist an der Reihe:

```
          INC TREFFER
          BNE NCHSTZIF
```

Andernfalls erhöht sich Indexregister Y. Ist die Sequenz zu Ende, erfolgt der Sprung nach NCHSTZIF. Oder aber es folgt der Rücksprung nach TREFSCHL:

```
NCHSTREF  INY             Zeiger erhöhen
          CPY LÄNGE       alle Ziffern getestet?
          BNE TREFSCHL    wenn nicht: nächste Ziffer
```

Jetzt muß die nächste COM0-Ziffer geprüft werden. Zähler ist das X-Register, er wird erhöht und auf die Gesamtlänge hin überprüft:

```
NCHSTZIF    INX             Ziffernzeiger erhöhen
            CPX LÄNGE       alle Ziffern durch?
```

Ist noch nicht alles überprüft, geht es wieder in die äußere Schleife bei ZIFSCHL:

```
            BNE ZIFSCHL
```

Ober aber wir können den Spieler über seinen Erfolg oder Mißerfolg informieren, indem wir die entsprechenden LEDs aufleuchten lassen.

***Ergebnisanzeige***

Die Summe der zu erleuchtenden LEDs ist TREFFER+VOLTRF:

```
            CLC             fertig für Addition
            LDA TREFFER
            ADC VOLTRF
```

Diese Summe wird vom Akkumulator ins Y-Register übertragen, das im LICHT-Unterprogramm als Index fungiert:

```
            TAY
            JSR LICHT
```

Die Beschreibung der LICHT-Routine folgt weiter unten. Sie lädt das passende Bitmuster in den Akkumulator, um die richtigen LEDs zu erleuchten. Dieses Muster wandert jetzt in die Maske:

```
            STA TOR1A
```

Beim bekannten Spezialergebnis 9 ist die Carryflagge gesetzt, dies muß überprüft werden:

```
            BCC CC          wenn Carry=0: nicht PB0 er-
                            leuchten
```

Ist Carry gesetzt, wird auch Tor 1B angesprochen:

```
            LDA #1          Tor 1B = 1
            STA TOR1B
```

Wir erinnern uns jetzt, daß, wenn die Masken A und B stehen, auch die Unterbrechungsroutine darauf zurückgreift, um die entsprechenden LEDs zum Blinken zu bringen:

```
CC      LDY VOLTRF
        JSR LICHT
        STA MASKA
        BCC TEST
        LDA #1
        STA MASKB
```

Schließlich muß das Programm noch testen, ob der Spieler gewonnen hat oder nicht.

### *Test auf Gewinn oder Verlust*

Die in VOLTRF stehende Zahl von Volltreffern wird einfach mit der Sequenzlänge verglichen. Sind diese Zahlen gleich, hat der Spieler gewonnen:

```
TEST    LDX VOLTRF
        CPX LÄNGE
        BEQ SIEG
```

Ist dem nicht so, erklingt ein tiefer Ton, dessen Tondauer auf 72 und dessen Frequenzkonstante auf BE gesetzt wird,

```
SCHLECHT  LDA #$72
          STA DAUER
          LDA #$BE
```

ehe das Unterprogramm TON aufgerufen wird:

```
          JSR TON
```

Dann geht es zurück zum Programmanfang:

```
          BEQ EINGABE
```

Hat der Spieler Erfolg gehabt, wird ein hoher Ton erzeugt, Dauerkonstante = FF, Frequenzkonstante = 54:

```
SIEG    LDA #$FF
        STA DAUER
        LDA #$54
        JSR TON
```

Und es beginnt ein neues Spiel:

```
        JMP TAST1
```

**Die Unterprogramme**

Es gibt vier Routinen, die von diesem Programm aufgerufen werden: LICHT, RANDOM, TON und INTERRUPT. RANDOM und TON können bei Bedarf in früheren Kapiteln nachgeschlagen werden. Betrachten wir hier nur die beiden anderen.

***Unterprogramm LICHT***

Beim Einsprung in diese Routine steht im Y-Register die Zahl der LEDs, die zum Blinken gebracht werden sollen. Dazu muß das entsprechende Muster in die Masken MASKA und MASKB geladen werden, damit dort die Bits „richtig sitzen". Zunächst wird Y jedoch auf 0 geprüft. Hat Y diesen Wert, werden Akkumulator und Carrybit genullt (die Carryflagge war der Zeiger für den Fall Y = 9):

```
LICHT    BNE STRTSH     Y = 0?
         LDA #0
         CLC
         RTS
```

Andernfalls wird der Akkumulator ebenfalls gelöscht, und der bekannte Verschiebevorgang von 1-Werten über das Carrybit in den Akkumulator findet statt, wobei Y der Zähler ist, der Schritt für Schritt auf 0 gebracht wird:

```
STRTSH   LDA #0
SHIFT    SEC
         ROL A          in Position schieben
         DEY
         BNE SHIFT      Schleife
         RTS
```

Beachten Sie, daß diesmal ein Rotations- und kein Schiebebefehl verwendet wird: Wäre Y = 9, würden lauter 1-Werte in den Akkumulator wandern, und das Carrybit wäre beim Rücksprung ebenfalls 1.

***Der INTERRUPT***

Diese Routine komplementiert die LEDs jedesmal, wenn ein Unterbrechungssignal kommt, also wenn Zeitgeber 1 ausläuft. Sie liegt ab Adresse 03EA. Da der Akkumulator als Arbeitsregister benutzt wird, muß er zunächst auf dem Stapel in Sicherheit gebracht werden:

```
         PHA
```

Nun werden die Inhalte der Tore 1A und 1B eingelesen und dann komplemen-

tiert, MASKA und MASKB bezeichnen die entsprechenden Bits. Da der 6502 keinen Komplementierbefehl kennt, benutzen wir die EOR-Anweisung:

```
LDA TOR1A
EOR MASKA
STA TOR1A
LDA TOR1B
EOR MASKB
STA TOR1B
```

Wir erinnern uns auch, daß das Interrupt-Bit im 6522 nach jeder Unterbrechung explizit gelöscht werden muß. Dies geschieht durch Einlesen des Zwischenspeichers:

```
LDA T1LL
```

Schließlich wird der Akkumulatorinhalt wiederhergestellt, und es geht zurück ins Hauptprogramm:

```
PLA
RTI
```

## ZUSAMMENFASSUNG

Dieses Programm benutzt zwei neue Hardware-Fähigkeiten des 6522 Ein/Ausgabe-Chips: die Unterbrechungskontrolle und den programmierbaren Intervallzeitgeber. Unterbrechungen werden benutzt, um simultanes Rechnen und LED-Blinken zu ermöglichen, während das Programm weiter auswertet.

**Übung 9.1:** *Könnten Sie dasselbe auch ohne eine Unterbrechungsbenutzung implementieren?*

```
;     HIRNVERDREHER
;BEIM SPIEL  HIRNVERDREHER  GIBT MAN ZUERST DIE LÄNGE DER ZU
;RATENDEN ZAHL EIN. NACH JEDEM RATEVERSUCH GIBT DER COMPUTER
;DANN AN, WIEVIELE ZIFFERN DER VERSUCHSZAHL IN DER ZU RATENDEN
;ZAHL VORKOMMEN (TREFFER) UND WIEVIELE ZIFFERN AN DER RICHTIGEN
;STELLE STEHEN (VOLLTREFFER). BLINKENDE LEDS KENNZEICHNEN VOLL-
;TREFFER UND STETIG ERLEUCHTETE LEDS TREFFER.
;DIE UNTERSTE LEDREIHE ZEIGT DEN PROGRAMM-MODUS AN: LEUCHTET DIE
;LED LINKS AUSSEN, ERWARTET DAS PROGRAMM DIE EINGABE DER GE-
;WÜNSCHTEN ZAHLENLÄNGE. LEUCHTEN DIE ZWEI LINKEN UNTEREN LEDS,
;WIRD DIE EINGABE EINER VERSUCHSZAHL ERWARTET. LÄNGENEINGABEN
;KLEINER 1 ODER GRÖSSER 9 WERDEN NICHT AKZEPTIERT, VERSUCHS-
;ZIFFERN GRÖSSER ALS 9 STARTEN EIN NEUES SPIEL. EIN TIEFER TON
;KENNZEICHNET EINEN FEHLVERSUCH, EIN HOHER TON EINEN SIEG. NACH
;EINEM SIEG, BEGINNT DAS PROGRAMM VON VORN. DAS BLINKEN DER LEDS
;BEWIRKT EINE UNTERBRECHUNGSROUTINE.
```

*Abb. 9.13: Programm HIRNVERDREHER*

```
                        ;
                        *=$200
                GETKEY  =$100
                ACCESS  =$BB86          ;ÖFFNET GESCHÜTZTEN SPEICHERBEREICH
                LÄNGE   =$00            ;ANZAHL ZU RATENDER ZIFFERN
                DAUER   =$01            ;TONDAUERKONSTANTE
                XTEMP   =$02            ;ZWISCHENSPEICHER FÜR X-REGISTER
                YTEMP   =$03            ;ZWISCHENSPEICHER FÜR Y-REGISTER
                TREFFER =$04            ;ZÄHLER FÜR TREFFERZAHL
                MASKA   =$05            ;BLINKMUSTER FÜR TOR A
                MASKB   =$06            ;BLINKMUSTER FÜR TOR B
                FREQ    =$07            ;FREQUENZKONSTANTE
                VOLTRF  =$08            ;ZÄHLER FÜR VOLLTREFFER
                RND     =$09            ;ERSTER ZUFALLSZAHLENSPEICHER
                COM0    =$0F            ;SPEICHERBEGINN FÜR 9 ZU RATENDE ZIFFERN
                SP10    =$18            ;SPEICHERBEGINN FÜR 9 VERSUCHSZIFFERN
                IRQVECL =$A67E          ;UNTERBRECHUNGSVEKTOR, LOW BYTE
                IRQVECH =$A67F          ;UNTERBRECHUNGSVEKTOR HIGH BYTE
                IER     =$A00E          ;UNTERBRECHUNGSZULASSUNGSREGISTER
                ACR     =$A00B          ;HILFSKONTROLLREGISTER
                T1LL    =$A004          ;ZWISCHENSPEICHER LOW ZEITGEBER 1
                T1CH    =$A005          ;ZÄHLER HIGH ZEITGEBER 1
                TOR1A   =$A001          ;VIA 1 TOR A EIN/AUSGABE-REGISTER
                DDR1A   =$A003          ;VIA 1 TOR A DATENRICHTUNGSREGISTER
                TOR1B   =$A000          ;VIA 1 TOR B EIN/AUSGABE-REGISTER
                DDR1B   =$A002          ;VIA 1 TOR B DATENRICHTUNGSREGISTER
                TOR3B   =$AC00          ;VIA 3 TOR B EIN/AUSGABE-REGISTER
                DDR3B   =$AC02          ;VIA 3 TOR B DATENRICHTUNGSREGISTER
                        ;
                        ;EINRICHTEN DER VARIABLEN UND DES UNTERBRECHUNGSZEITGEBERS ZUM
                        ;BLINKEN DER LEDS
                        ;
0200: 20 86 BB                  JSR ACCESS      ;GESCHÜTZTEN BEREICH ZUGÄNGLICH MACHEN
0203: A9 EA                     LDA #$EA        ;UNTERBRECHUNGSVEKTOR (LOW BYTE) LADEN
0205: 8D 7E A6                  STA IRQVECL     ;...UND SPEICHERN
0208: A9 03                     LDA #$03        ;DASSELBE MIT HIGH BYTE
020A: 8D 7F A6                  STA IRQVECH
020D: A9 7F                     LDA #$7F        ;INTERRUPT ENABLE REGISTER LÖSCHEN
020F: 8D 0E A0                  STA IER
0212: A9 C0                     LDA #$C0        ;ZEITGEBER 1 INTERRUPT ZULASSEN
0214: 8D 0E A0                  STA IER
0217: A9 40                     LDA #$40        ;FREILAUFMODUS ZEITGEBER 1
0219: 8D 0B A0                  STA ACR
021C: A9 FF                     LDA #$FF
021E: 8D 04 A0                  STA T1LL        ;MAXIMALWERT $FFFF FÜR ZÄHLREGISTER
0221: 8D 05 A0                  STA T1CH        ;DES ZEITGEBERS
0224: 58                        CLI             ;INTERRUPTS ZULASSEN
0225: 8D 03 A0                  STA DDR1A       ;VIA 1 TOR A AUF AUSGABE
0228: 8D 02 A0                  STA DDR1B       ;VIA 1 TOR B AUF AUSGABE
022B: 8D 02 AC                  STA DDR3B       ;VIA 3 TOR B AUF AUSGABE
022E: A9 00             TAST1   LDA #0          ;LEDS LÖSCHEN
0230: 8D 01 A0                  STA TOR1A
0233: 8D 00 A0                  STA TOR1B
0236: 85 05                     STA MASKA       ;BLINKMASKEN LÖSCHEN
0238: 85 06                     STA MASKB
                        ;
                        ;VOM SPIELER GEWÜNSCHTE ZIFFERNANZAHL HOLEN UND ENTSPRECHEND
                        ;VIELE ZUFALLSZAHLEN ZWISCHEN 0 UND 9 ERZEUGEN
                        ;
023A: A9 02                     LDA #%00000010;LED-SIGNAL ZUR EINGABE DER GEWÜNSCHTEN
023C: 8D 00 A0                  STA TOR1B       ;ZIFFERNANZAHL ERZEUGEN...
023F: 20 00 01                  JSR GETKEY      ;...UND EINGABE HOLEN
0242: C9 0A                     CMP #10         ;WENN TASTE > 9: NEUSTART
0244: 10 E8                     BPL TAST1
0246: C9 00                     CMP #0          ;ZIFFERNANZAHL 0 NICHT ERLAUBT!
0248: F0 E4                     BEQ TAST1
024A: 85 00                     STA LÄNGE       ;GEWÜNSCHTE LÄNGE ABSPEICHERN
024C: AD 04 A0                  LDA T1LL        ;ZUFALLSZAHL HOLEN UND GENERATORSPEICHER
024F: 85 0A                     STA RND+1       ;FÜLLEN
0251: 85 0D                     STA RND+4
0253: 85 0E                     STA RND+5
0255: A4 00                     LDY LÄNGE       ;ZAHLENLÄNGE HOLEN UND AUF 0 HERUNTER-
```

*Abb. 9.13: Programm HIRNVERDREHER (Fortsetzung)*

```
0257: 88                    DEY             ;ZÄHLEN
0258: 20 FF 02  RAND        JSR RANDOM      ;NÄCHSTE ZUFALLSZAHL HOLEN
025B: F8                    SED
025C: 69 00                 ADC #0          ;AUF DEZIMALZAHL JUSTIEREN
025E: D8                    CLD
025F: 29 0F                 AND #%00001111;ZIFFER KLEINER 10 HALTEN
0261: 99 0F 00              STA COM0,Y      ;IN TABELLE ABLEGEN
0264: 88                    DEY
0265: 10 F1                 BPL RAND        ;NÄCHSTE ZIFFER
                    ;
                    ;SPIELERZIFFERN IN TABELLE ABLEGEN
                    ;
0267: A9 00         EINGABE LDA #0          ;TABELLENZEIGER INITIALISIEREN
0269: 85 02                 STA XTEMP
026B: A9 06                 LDA #%00000110;EINGABE-SIGNAL GEBEN
026D: 0D 00 A0              ORA TOR1B
0270: 8D 00 A0              STA TOR1B       ;FELD UNVERÄNDERT
0273: 20 00 01  TAST2       JSR GETKEY      ;VERSUCHSZIFFER EINLESEN
0276: C9 0A                 CMP #10         ;GRÖSSER 9?
0278: 10 B4                 BPL TAST1       ;WENN JA: NEUSTART
027A: A6 02                 LDX XTEMP       ;INDIZIERUNGSZEIGER HOLEN
027C: 95 10                 STA SPI0,X      ;VERSUCHSZIFFER IN TABELLE ABLEGEN
027E: E8                    INX             ;ZEIGER ERHÖHEN
027F: 86 02                 STX XTEMP
0281: E4 00                 CPX LÄNGE       ;ALLE ZIFFERN GEHOLT?
0283: D0 EE                 BNE TAST2       ;WENN NICHT: NÄCHSTE HOLEN
                    ;
                    ;VERGLEICHEN DER SPIELERZIFFERN MIT DEN ZIFFERN DER COMPUTER-
                    ;ZAHL. VOLLTREFFER DURCH BLINKENDE LED UND TREFFER DURCH NICHT
                    ;BLINKENDE LED ANZEIGEN
                    ;
0285: A2 00                 LDX #0          ;LÖSCHEN VON...
0287: 8E 01 A0              STX TOR1A       ;...LEDS
028A: 8E 00 A0              STX TOR1B
028D: 86 05                 STX MASKA       ;...BLINKMASKEN
028F: 86 06                 STX MASKB
0291: 86 04                 STX TREFFER     ;...TREFFERN
0293: 86 0B                 STX VOLTRF      ;...UND VOLLTREFFERN
0295: B5 0F     ZIFSCHL     LDA COM0,X      ;ZIFFER AUF VOLLTREFFER PRÜFEN
0297: D5 10                 CMP SPI0,X      ;VOLLTREFFER?
0299: D0 04                 BNE TREFSUCH    ;WENN NICHT: AUF TREFFER PRÜFEN
029B: E6 0B                 INC VOLTRF      ;WENN JA: VOLLTREFFERZAHL ERHÖHEN
029D: D0 10                 BNE NCHSTZIF    ;...UND NÄCHSTE ZIFFER
029F: A0 00     TREFSUCH    LDY #0          ;ZEIGER INITIALISIEREN FÜR TREFFERSUCHE
02A1: D9 1B 00  TREFSCHL    CMP SPI0,Y      ;TREFFER?
02A4: D0 04                 BNE NCHSTREF    ;WENN NICHT: NÄCHSTE ZIFFER PRÜFEN
02A6: E6 04                 INC TREFFER     ;WENN JA: TREFFERZÄHLER ERHÖHEN
02A8: D0 05                 BNE NCHSTZIF    ;...UND NÄCHSTE ZIFFER
02AA: C8        NCHSTREF    INY             ;ZEIGER ERHÖHEN
02AB: C4 00                 CPY LÄNGE       ;ALLE ZIFFERN DURCH?
02AD: D0 F2                 BNE TREFSCHL    ;WENN NICHT: NÄCHSTE ZIFFER
02AF: E8        NCHSTZIF    INX             ;ZIFFERNZEIGER ERHÖHEN
02B0: E4 00                 CPX LÄNGE       ;ALLE ZIFFERN AUSGEWERTET?
02B2: D0 E1                 BNE ZIFSCHL     ;WENN NICHT: NÄCHSTE ZIFFER
02B4: 18                    CLC             ;TREFFER UND VOLLTREFFER FÜR LED-ANZEIGE
02B5: A5 04                 LDA TREFFER     ;ADDIEREN
02B7: 65 0B                 ADC VOLTRF
02B9: AB                    TAY             ;FÜR LICHT-ROUTINE NACH Y ÜBERTRAGEN
02BA: 20 F1 02              JSR LICHT       ;RICHTIGES LEDMUSTER HOLEN
02BD: 8D 01 A0              STA TOR1A       ;LEDS AN!
02C0: 90 05                 BCC CC          ;WENN CARRY=0: PB0 DUNKEL
02C2: A9 01                 LDA #1
02C2: 8D 00 A0              STA TOR1B       ;PB0 AN
02C7: A4 0B     CC          LDY VOLTRF      ;ZU BLINKENDE LEDZAHL LADEN
02C9: 20 F1 02              JSR LICHT       ;MUSTER HOLEN
02CC: 85 05                 STA MASKA       ;LEDS ZUM BLINKEN FERTIGMACHEN
02CE: 90 04                 BCC TEST        ;BEI CARRY=0: PB0 BLINKT NICHT
02D0: A9 01                 LDA #1
02D2: 85 06                 STA MASKB
                    ;
                    ;AUF SIEGSITUATION PRÜFEN DURCH VERGLEICH: ZIFFERNANZAHL=VOLL-
                    ;TREFFERZAHL? BEI SIEG HOHEN TON ERZEUGEN, WENN NICHT ALLE
```

*Abb. 9.13: Programm HIRNVERDREHER (Fortsetzung)*

```
                    ;ZIFFERN RICHTIG, TIEFEN TON ERZEUGEN.
                    ;
02D4: A6 08         TEST        LDX VOLTRF      ;VOLLTREFFERANZAHL HOLEN
02D6: E4 00                     CPX LäNGE       ;ALLES RICHTIG GERATEN?
02D8: F0 0B                     BEQ SIEG        ;WENN JA: SPIELER GEWINNT
02DA: A9 72         SCHLECHT    LDA #$72
02DC: 85 01                     STA DAUER       ;TONDAUER SETZEN
02DE: A9 BE                     LDA #$BE        ;TIEFER TON
02E0: 20 12 03                  JSR TON         ;TONSIGNAL: SCHLECHT GERATEN
02E3: F0 82                     BEQ EINGABE     ;NäCHSTER VERSUCH
02E5: A9 FF         SIEG        LDA #$FF        ;TONDAUER FüR HOHEN TON
02E7: 85 01                     STA DAUER
02E9: A9 54                     LDA #$54        ;HOHER TON
02EB: 20 12 03                  JSR TON         ;SIEGSIGNAL
02EE: 4C 2E 02                  JMP TAST1       ;NEUES SPIEL
                    ;
                    ;UNTERPROGRAMM 'LICHT'. SCHIEBT '1'-BITS VON RECHTS IN AKKUMULA-
                    ;TOR BIS ZUR POSITION, DIE DER ZU ERLEUCHTENDEN LED ENTSPRICHT.
                    ;
02F1: D0 04         LICHT       BNE STRTSH      ;WENN Y<>0: '1'-BITS EINSCHIEBEN
02F3: A9 00                     LDA #0          ;SPEZIALFALL: KEINE '1'-BITS
02F5: 18                        CLC
02F6: 60                        RTS
02F7: A9 00         STRTSH      LDA #0          ;AKKUMULATOR FERTIGMACHEN
02F9: 38            SHIFT       SEC             ;...BIT SETZEN
02FA: 2A                        ROL A           ;...UND IN AKKUMULATOR SCHIEBEN
02FB: 88                        DEY             ;Y-ZäHLER ERNIEDRIGEN
02FC: D0 FB                     BNE SHIFT       ;WENN NICHT FERTIG: WEITERSCHIEBEN
02FE: 60                        RTS
                    ;
                    ;ZUFALLSZAHLENGENERATOR
                    ;BENUTZT DIE IN RND BIS RND+5 ENTHALTENEN ZAHLEN A,B,C,D,E,F:
                    ;ADDIERT B+E+F+1 UND SPEICHERT ERGEBNIS IN A, DANN GEHT A NACH
                    ;B, B NACH C USW. ZUFALLSZAHL ZWISCHEN 0 UND 255 IST BEIM RüCK-
                    ;SPRUNG IM AKKUMULATOR.
                    ;
02FF: 38            RANDOM      SEC             ;CARRY GESETZT: ADDIERT 1
0300: A5 0A                     LDA RND+1       ;A,B,E UND 1 ADDIEREN
0302: 65 0D                     ADC RND+4
0304: 65 0E                     ADC RND+5
0306: 85 09                     STA RND
0308: A2 04                     LDX #4          ;ZAHLEN WEITERSCHIEBEN
030A: B5 09         RPL         LDA RND,X
030C: 95 0A                     STA RND+1,X
030E: CA                        DEX
030F: 10 F9                     BPL RPL
0311: 60                        RTS
                    ;
                    ;UNTERPROGRAMM ZUR TONERZEUGUNG. TONDAUER (=SCHLEIFENZäHLER)
                    ;MUSS BEIM EINSPRUNG IN 'DAUER' BEREITGESTELLT SEIN, FREQUENZ-
                    ;KONSTANTE IM AKKUMULATOR.
                    ;
0312: 85 07         TON         STA FREQ
0314: A9 FF                     LDA #$FF
0316: 8D 00 AC                  STA TOR3B
0319: A9 00                     LDA #0
031B: A6 01                     LDX DAUER
031D: A4 07         FL2         LDY FREQ
031F: 88            FL1         DEY
0320: 18                        CLC
0321: 90 00                     BCC +2
0323: D0 FA                     BNE FL1
0325: 49 FF                     EOR #$FF
0327: 8D 00 AC                  STA TOR3B
032A: CA                        DEX
032B: D0 F0                     BNE FL2
032D: 60                        RTS
                    ;
                    ;UNTERBRECHUNGSROUTINE ZUM BLINKEN DER LEDS
                    ;
                            * = $3EA        ;IM HOHEN SPEICHERBEREICH LOKALISIERT
03EA: 48                    PHA             ;AKKUMULATOR RETTEN
```

*Abb. 9.13: Programm HIRNVERDREHER (Fortsetzung)*

```
03EB: AD 01 A0          LDA TOR1A    ;TOR ZUM KOMPLEMENTIEREN HOLEN
03EE: 45 05             EOR MASKA    ;ENTSPRECHENDE BITS KOMPLEMENTIEREN
03F0: 8D 01 A0          STA TOR1A    ;UND WIEDER SPEICHERN
03F3: AD 00 A0          LDA TOR1B    ;DASSELBE MIT TOR B
03F6: 45 06             EOR MASKB
03F8: 8D 00 A0          STA TOR1B
03FB: AD 04 A0          LDA TILL     ;UNTERBRECHUNGSBIT IN VIA LÖSCHEN
03FE: 68                PLA          ;AKKUMULATOR WIEDERHOLEN
03FF: 40                RTI          ;FERTIG

SYMBOLTABELLE:
GETKEY   0100   ACCESS   0BB6   LÄNGE    0000   DAUER    0001
XTEMP    0002   YTEMP    0003   TREFFER  0004   MASKA    0005
MASKB    0006   FREQ     0007   VOLTRF   000B   RND      0009
COM0     000F   SPI0     001B   IRQVECL  A67E   IRQVECH  A67F
IER      A00E   ACR      A00B   TILL     A004   T1CH     A005
TOR1A    A001   DDR1A    A003   TOR1B    A000   ODR1B    A002
TOR3B    AC00   DDR3B    AC02   TAST1    022E   RAND     025B
EINGABE  0267   TAST2    0273   ZIFSCHL  0295   TREFSUCH 029F
TREFSCHL 02A1   NCHSTREF  02AA   NCHSTZIF 02AF   CC       02C7
TEST     02D4   SCHLECHT 02DA   SIEG     02E5   LICHT    02F1
STRTSH   02F7   SHIFT    02F9   RANDOM   02FF   RPL      030A
TON      0312   FL2      031D   FL1      031F
```

*Abb. 9.13: Programm HIRNVERDREHER (Fortsetzung)*

# 10

# Komplexe Auswertungstechnik (17 und 4)

## EINFÜHRUNG

Bei der in diesem Kapitel besprochenen Problemstellung kommt ein vielschichtiges Auswertungssystem zum Zug, das nur eine einfache Ein/Ausgabe-Umgebung und wenig Speicherplatz braucht. Das Programm arbeitet mit Licht- und Toneffekten in Realzeit.

## DIE SPIELREGELN

Bei dem allgemein bekannten Kartenspiel „17 und 4" versuchen der Spieler und der Kartengeber, durch Sammeln von Kartenwerten möglichst viele Punkte („Augen") zu bekommen, um mehr zu haben als der Gegner. Der Clou ist dabei, daß, wer beim Sammeln über 21 (17 + 4) Augen kommt, verloren hat. Erreicht ein Spieler mit nur zwei Karten genau 21 Augen, so hat er in jedem Fall gewonnen. Die Karten haben Werte von 1 bis 11 Augen. In der Standardversion des Spieles muß der Geber (die Bank) so lange Karten nehmen, bis er 17 oder mehr Augen hat.

Bei der Version, die wir hier auf dem Spielbrett spielen wollen, enthält das „Blatt" nur die Werte 1 bis 10 (es gibt also kein „As" mit 11 Augen), und die höchste erlaubte Augenzahl ist 13 statt 21. Die Bank (der Geber) ist der Computer.

Eine Spielrunde beginnt damit, daß Geber und Spieler je eine Karte bekommen. Eine kontinuierlich leuchtende LED repräsentiert dabei den Kartenwert des Gebers (des Computers), eine blinkende LED den des Spielers. Möchte der Spieler nun eine weitere Karte, muß er Taste A drücken, Er kann dies mehrmals tun, kommt er jedoch dabei über 13 Augen, so hat er das Spiel „geschmissen", also verloren, er erhält keine weiteren Karten und hat die Runde verloren. In diesem Fall, und wenn der Spieler sich entschieden hat, keine weiteren Karten zu nehmen, ist der Geber an der Reihe: Solange er zusammen weniger als 10 Augen hat, muß er eine weitere Karte nehmen. Ist er noch nicht über 13 Augen, prüft er, ob er eine weitere Karte ziehen muß. Auch er verliert, wenn er über 13 kommt. Der erwähnte sichere Gewinn bei 13 Augen mit nur zwei Karten wird bei unserer Version nicht berücksichtigt, dies sei dem Leser überlassen. Sobald der Geber seine Karten hat, ohne über 13 Augen

gekommen zu sein, werden die Augen der Gegner verglichen, und die höhere Augenzahl gewinnt.

Zu Beginn jeder Serie erhält der Spieler 5 Punkte als Spielkapital. Bei jeder verlorenen Runde wird dieses Kapital um einen Punkt verringert. Bei Erreichen von 0 Punkten hat der Spieler die Serie verloren, und das Spiel ist zu Ende. Erreicht der Spieler 10 Punkte, ist umgekehrt der Computer der Verlierer. Das aktuelle Punktekonto des Spielers wird nach jeder Runde mit der entsprechenden LED angezeigt. Gewinnt der Spieler eine Runde, leuchten die drei LEDs links in der unteren Reihe auf, die rechten drei LEDs in dieser Reihe signalisieren einen Rundengewinn für den Geber (Bild 10.1).

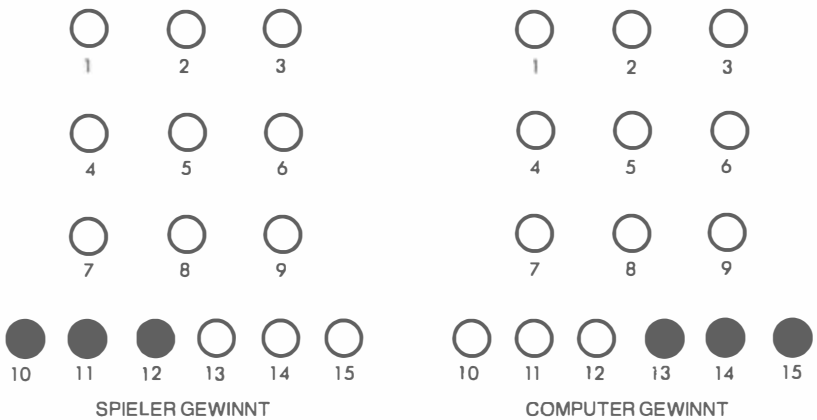



*Abb. 10.1: Anzeigen des Gewinners*

## EIN TYPISCHER SPIELVERLAUF

Um bei einem Spiel gegen den Geber eine weitere Karte zu bekommen, muß der Spieler Taste A drücken. Er kann das tun, bis er entweder über die Gesamtaugenzahl von 13 kommt, oder bis er der Meinung ist, daß er „genug" hat, also dicht genug bei 13 ist, um den Geber zu besiegen. Im letztgenannten Fall drückt er Taste C (alle anderen Tasten werden ignoriert), und der Geber ist an der Reihe. Während der Computer jetzt seine Karten bekommt, leuchten verschiedene LEDs auf. Schließlich hat der Geber mehr als 10, genau 13 oder über 13 Augen. Der Spieler kann dann irgendeine Taste drücken, um das Ergebnis und den neuen Punktstand zu erfahren. Nach etwa einer Sekunde beginnt dann ein neues Spiel.

Zu beachten ist, daß bei 10 oder mehr Augen des Gebers so lange nichts geschieht, bis eine Taste gedrückt wird. Folgen wir nun einem „typischen" Spiel.

Bild 10.2 zeigt die Anfangssituation: Ein Dauerlicht (voll ausgefüllter Kreis) zeigt 1 Punkt für den Geber, und ein Blinklicht (halb ausgefüllter Kreis) zeigt 4 Punkte für den Spieler. Dieser drückt nun Taste A, um eine weitere Karte zu erhalten: Es ist eine 9, und die Anzeige erscheint wie in Bild 10.3. Das ist die Idealsituation, 13 Punkte. Das Beste, was der Geber jetzt noch erreichen kann, ist ein Gleichstand mit ebenfalls 13 Punkten. Sehen wir also, was passiert. Dazu drücken wir die C-Taste, und einen Augenblick später leuchtet LED 3 auf. Damit hat der Computer 1 + 3 = 4 Augen, und er muß sich eine weitere Karte geben. Kurz darauf geht LED 7 an, was bedeutet, daß der Geber jetzt 4 + 7 = 11 Augen hat. Da er keine Karte mehr nimmt, hat er diese Runde verloren. Wir überzeugen uns durch Drücken irgendeiner Taste (etwa 0): LEDs 10, 11 und 12 gehen an (Sieg des Spielers) und danach LED 6, das neue Punktekonto (Bild 10.4). Schließlich gehen alle LEDs aus, und eine neue Runde beginnt. Bei einem Gleichstand bleiben übrigens alle unteren LEDs dunkel, und das Punktekonto bleibt unverändert. „Schmeißt" der Spieler (mehr als 13 Augen), so wird der Gebersieg unmittelbar angezeigt.

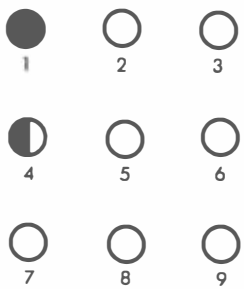



*Abb. 10.2: Erste Blattverteilung*

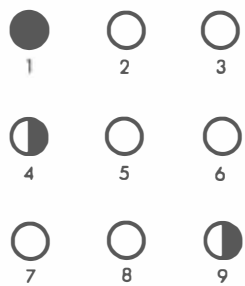



*Abb. 10.3: 2. Karte für den Spieler: gewonnen*

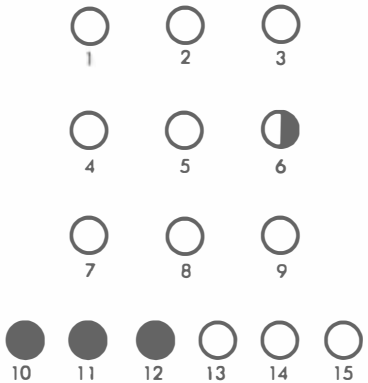



*Abb. 10.4: Ende der Runde: Geber verliert*

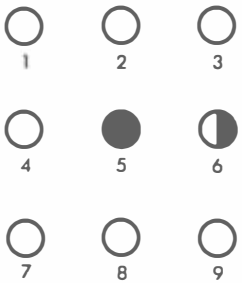



*Abb. 10.5: 2. Blatt*

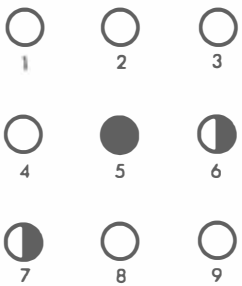



*Abb. 10.6: Wieder gewonnen*

Zu Beginn der neuen Runde hat der Computer sich 5 und dem Spieler 6 Augen gegeben (Bild 10.5). Mit Taste A fordern wir eine weitere Karte und bekommen eine 7. Unglaublich aber wahr: wir haben wieder 13 Augen (Bild 10.6), und Taste C bringt den Computer ans Spiel. LED 10 geht an und bringt dem Computer die Gesamtaugenzahl 15, er hat geschmissen (Bild 10.7). Nach Drücken einer Taste sehen wir uns bestätigt (Bild 10.8): Die drei linken unteren LEDs (10, 11 und 12) zeigen unseren Sieg, und LED 7 markiert unser neues Punktekonto. Kurz darauf erlischt die Anzeige, und wieder kann eine neue Runde beginnen.

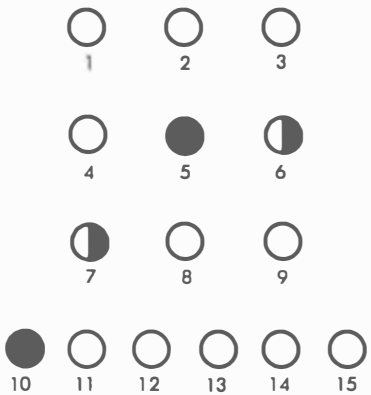



*Abb. 10.7: Geber hat „geschmissen"*

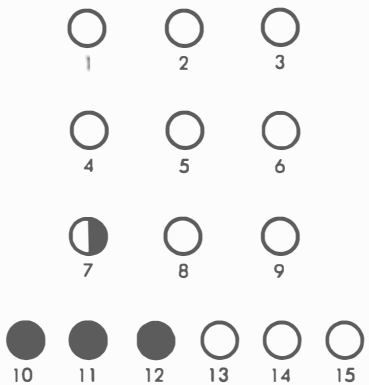



*Abb. 10.8: Punktstand 7*

## DAS PROGRAMM

Bild 10.9 zeigt das detaillierte Flußdiagramm für das „17 und 4"-Spiel, das Listing steht am Schluß des Kapitels. Wieder sind die Variablen und Flaggen, die keinen dauernden Platz in den internen Registern des 6502 haben, auf der Nullseite untergebracht, Bild 10.10 zeigt die Speicheraufteilung:

FERTIG: Diese Flagge hat zu Beginn des Spiels den Wert 0, geht der Spieler „pleite", erhält sie den Wert 11111111, und kommt er auf das Siegkonto 10, so wird sie 1. Nach jeder Runde wird diese Flagge von der Routine BEENDEN getestet, die das Endergebnis eines Spiels anzeigt: entweder eine stetig leuchtende LED-Reihe oder ein blinkendes Quadrat.
CHIPS: Diese Variable enthält das Punktekonto des Spielers (Anfangswert 5). Bei jedem Sieg des Spielers wird CHIPS um 1 erhöht, bei jeder Niederlage um 1 erniedrigt. Wird CHIPS 0 oder 10, endet das Spiel.
MASKA, MASKB: In diesen beiden Variablen stehen die Masken (die Muster) für das Blinken der mit Tor A bzw. Tor B verbundenen LEDs.
SPAUG: Hier ist die aktuelle Augensumme des Spielers gespeichert. Sie wird entsprechend jeder neugezogenen Karte erhöht.
COMAUG: die entsprechende Gesamtaugenzahl des Computers (des Gebers).
TEMP: Dies ist der Zwischenspeicher, in dem die RANDOM-Routine die nächste Karte für Spieler oder Geber ablegt.
RND bis RND+5: Diese sechs Speicherstellen sind für den Zufallszahlengenerator (Unterprogramm RANDER) bereitgestellt.
SIEGER: Der Gewinner der laufenden Runde wird in dieser Statusflagge festgeschrieben: Der Anfangswert 0 wird bei einem Spielersieg inkrementiert, bei einer Spielerniederlage dekrementiert.

Am oberen Speicherende (Bild 10.11) sind die Adressen für die VIA-Kontrolle, die ACCESS-Routine des SYM-Monitors und der Unterbrechungsvektor untergebracht.

Verfolgen wir nun den allgemeinen Programmverlauf, am besten anhand des Flußdiagrammes von Bild 10.9.

### Initialisierung

Der Zeitgeber des 6522 VIA 1 erzeugt die Unterbrechungen für das Blinken der LEDs. Die Interrupt-Verzweigung geht nach 03EA, wo die Unterbrechungsroutine untergebracht ist. Als erstes laden wir also den Unterbrechungsvektor 03EA:

```
'17+4'      JSR ACCESS      Systemspeicher zugänglich machen
            LDA #$EA        Low Byte des Unterbrechungsvektors
            STA INTVECL
            LDA #$03        High Byte
            STA INTVECH
```

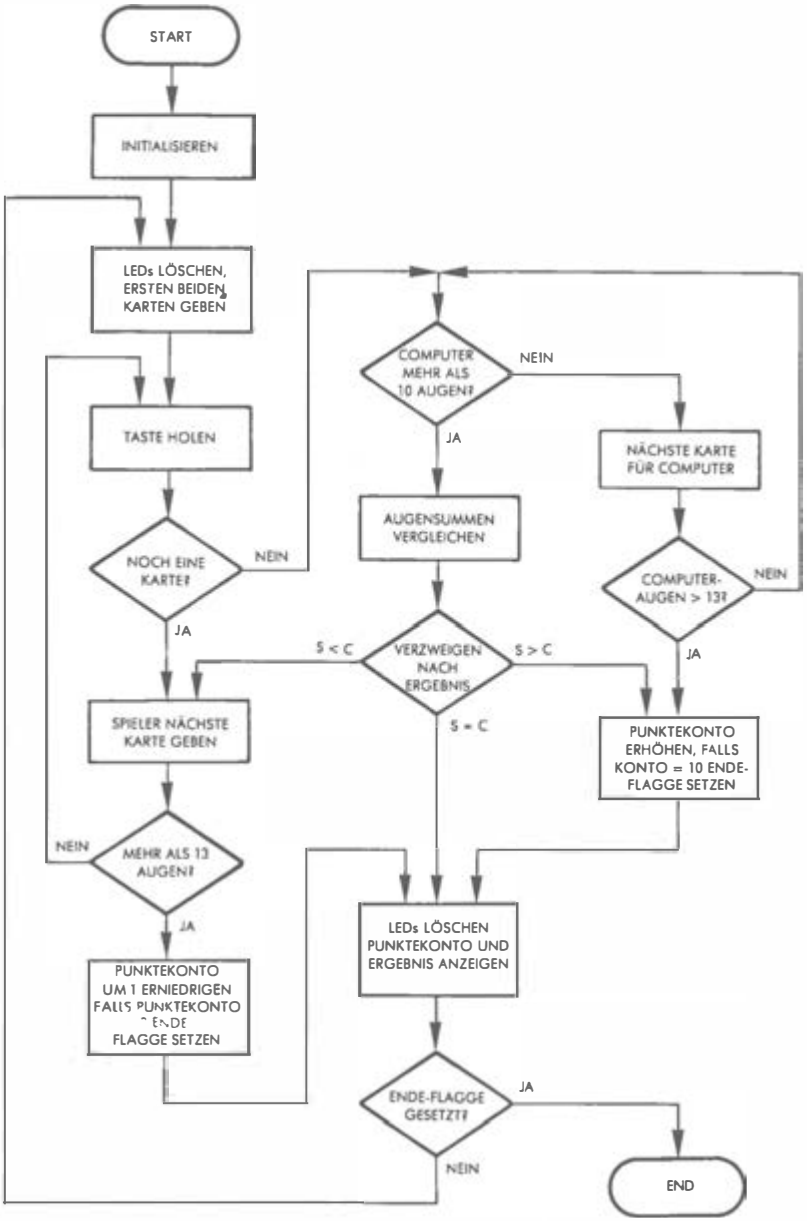



*Abb. 10.9: Flußdiagramm „17+4"*

Wie im vorigen Kapitel beschrieben, wird das Unterbrechungszulassungs-Register (IER) zuerst mit 01111111, dann mit 11000000 geladen:

```
LDA #$7F        Zeitgeber-Interrupt löschen
STA IER
LDA #$C0        Zeitgeber-Interrupt zulassen
STA IER
```

Laden des Wertes 7F löscht Bits 0 bis 6, was alle Unterbrechungen sperrt. Dann setzt C0 Bit 6, das Unterbrechungsbit für Zeitgeber 1 (vgl. Bild 9.10). Wie im vorigen Kapitel wird Zeitgeber 1 auf Freilauf-Modus geschaltet. Er erzeugt dann automatisch die Unterbrechungen für das LED-Blinken. Im Freilaufmodus ist Bit 6 von ACR 1:

```
LDA #$40        Zeitgeber 1...
STA ACR         ... in Freilaufmodus
```

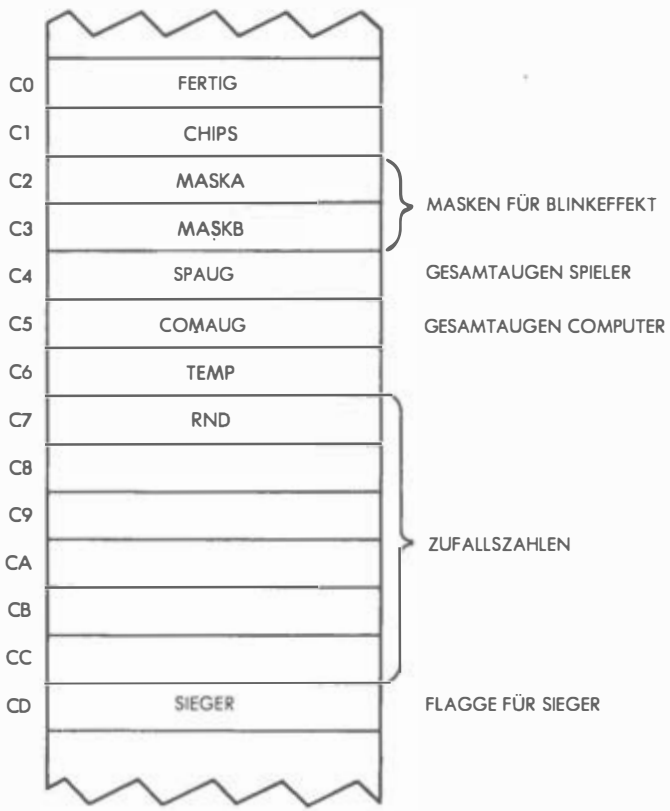



*Abb. 10.10: Unterer Speicherbereich*

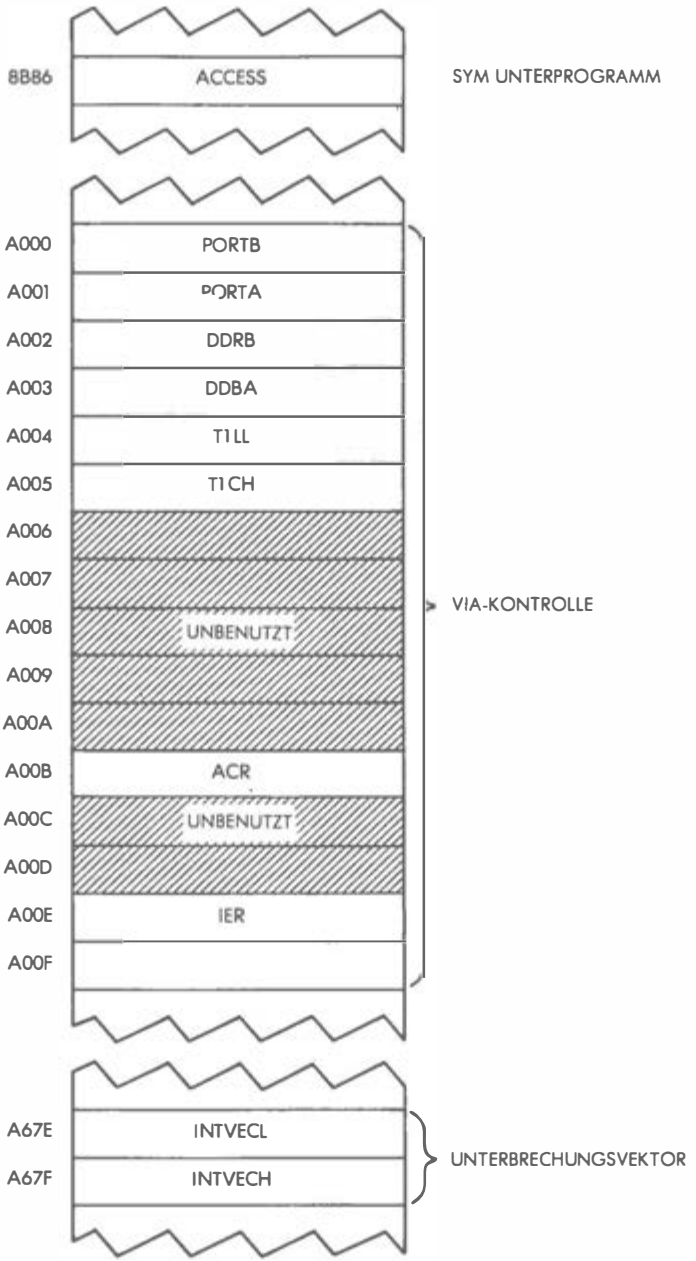



*Abb. 10.11: Oberer Speicherbereich*

Die Zwischenspeicher von Zeitgeber 1 erhalten wieder den Höchstwert FFFF:

```
LDA #$FF
STA T1LL        Low Byte
STA T1CH        High Byte und Zeitgeberstart
```

Der Zeitgeber ist richtig initialisiert, und es werden Unterbrechungen zugelassen:

```
CLI             Interrupts zugelassen
```

Die LED-Tore A und B gehen auf Ausgabe ($FF ist noch in A):

```
STA DDRA
STA DDRB
```

Das Löschen der Dezimalflagge ist eine Vorsichtsmaßnahme:

```
CLD
```

Nun kommen 5 Punkte auf das Spielerkonto,

```
LDA #5          Spielerkonto = 5
STA CHIPS
```

die FERTIG-Flagge wird genullt,

```
LDA #0
STA FERTIG
```

und alle LEDs werden gelöscht:

```
STA MASKA
STA MASKB
STA TORA
STA TORB
```

Und auch die SIEGERflagge wird gelöscht:

```
STA SIEGER
```

### *Geben des ersten Blattes*

Es kann also losgehen: Geben wir Spieler und Geber je eine Karte; dazu benutzen wir die Routinen LICHT und BLINKEN. Jedes dieser Unterprogramme holt sich eine Zufallszahl und erleuchtet die entsprechende LED, wobei LICHT ein bleibendes und BLINKEN ein blinkendes Licht erzeugt (die

Beschreibung der beiden Routinen folgt weiter unten). Zunächst erhält der Spieler ein blinkendes Licht,

```
        JSR BLINKEN     erste Spielerkarte ausgeben
```

und der Kartenwert wird als Augenzahl gespeichert:

```
        STA SPAUG       Augen mitzählen
```

Für den Computer geschieht das Analoge:

```
        JSR LICHT       Zufalls-Dauerlicht
        STA COMAUG      Geberaugen mitzählen
```

***Karte nehmen oder passen?***

Es folgt die Tastaturabfrage: Drückt der Spieler A, erhält er eine weitere Karte, drückt er C, so paßt er und der Computer ist an der Reihe. Alle anderen Tasten werden ingnoriert. Die gedrückte Taste besorgt uns die GETKEY-Routine:

```
FRAGEN  JSR GETKEY
```

Der zurückgebrachte Wert muß mit A und C verglichen werden:

```
        CMP #$0A
        BEQ SPLNMT
        CMP #$0C        Computer an der Reihe?
        BEQ GEBER
```

Wurde irgendeine andere Taste gedrückt, muß neu eingelesen werden:

```
        JMP FRAGEN      ungültige Taste
```

Verfolgen wir zuerst den Fall, daß der Spieler eine Karte nimmt, dann muß eine weitere LED blinken. Natürlich muß dafür gesorgt werden, daß weder die LICHT- noch die BLINK-Routine eine Karte erwischt, die schon gegeben wurde. Wie das geschieht, werden wir später sehen, wenn wir die SETBIT-Routine besprechen.

```
SPLNMT  JSR BLINKEN     Zufalls-LED setzen
```

Hat der Spieler seine neue Karte, wird seine neue Augenzahl berechnet:

```
        CLC
        ADC SPAUG       Augen addieren
        STA SPAUG
```

Die neue Summe muß nun auf den Wert 13 getestet werden. Hat der Spieler 13 oder weniger, darf er eine weitere Karte nehmen oder passen. Liegt er über 13, so hat er „geschmissen" und verloren:

```
CMP #14          auf Maximum 13 testen
BCC FRAGEN       auf 13 oder weniger testen
JMP VRLORN       „geschmissen"
```

An dieser Stelle ist der Geber an der Rheihe. Da der Computer erheblich schneller „überlegt" als der Spieler, wollen wir seine Bedenkzeit künstlich etwas verlängern:

```
GEBER    JSR DELAY
```

Diese Verzögerung verlängert auch die Zeitspannen zwischen den einzelnen Computerentscheidungen, was ihn etwas „menschlicher" agieren läßt.

Ehe wir den Computer eine weitere Karte nehmen lassen, soll er seine bisher gesammelten Augen überprüfen. Als Faustregel wollen wir ihn bei mehr als 10 Augen keine weitere Karte nehmen lassen („17 und 4"-Experten mögen auch eine andere „Verhaltensweise" erproben). Das Computerblatt wird also auf die Augenzahl 10 getestet: Liegt es über diesem Wert, wird nach BESSER verzweigt, wo die höhere Augenzahl ermittelt wird. Andernfalls erhält der Computer eine weitere Karte:

```
LDA COMAUG
CMP #10          Höchstrisiko erreicht?
BCS BESSER       wenn ja: bessere Augenzahl berechnen
```

Solange der Geber unter der Risikoschwelle liegt, nimmt er eine weitere Karte. Dies geht vor sich wie weiter oben beim Spieler:

```
JSR LICHT        neue Karte = Zufalls-LED
```

Und die neue Augenzahl wird errechnet:

```
CLC
ADC COMAUG       Augen summieren
STA COMAUG
```

Wie beim Spieler wird auf 13 getestet („geschmissen" oder nicht?):

```
CMP #14          Augenzahl größer als 13?
BCC GEBER        wenn nicht: weitere Karte
JMP GWONNEN      „geschmissen": Spieler gewinnt
```

Hat der Computer über 13, also geschmissen, geht es bei GWONNEN weiter, wo der Spieler seinen Sieg erfährt. Andernfalls erfolgt der Rücksprung nach GEBER, wo der Computer erneut zu entscheiden hat, ob er eine weitere Karte nimmt. Wir müssen jetzt ermitteln, wer gewonnen hat, die beiden Augenzahlen werden also verglichen:

```
BESSER    LDA COMAUG
          CMP SPAUG      Augen vergleichen
```

Es gibt drei mögliche Ergebnisse: Sieg des Spielers, Gleichstand, Sieg des Gebers:

```
          BEQ ANZEIGE
          BCC GWONNEN
```

Sind die Augenzahlen gleich, wird bei ANZEIGE der neue (und alte) Stand ausgegeben. Bei einem Spielersieg geht es bei GWONNEN (siehe weiter unten) weiter. Hat der Computer gewonnen, tritt der Programmteil VRLORN in Aktion. Schauen wir uns dies zuerst an.

***Spieler verliert***

Der Ausgang einer Runde wird in der Flagge SIEGER festgehalten: Hat der Spieler verloren, wird diese Flagge dekrementiert

```
VRLORN    DEC SIEGER
```

und das Spielerkonto um 1 vermindert:

```
          DEC CHIPS
```

Dieses Konto muß jetzt auf 0 getestet werden, um festzustellen, ob der Spieler pleite ist. Ist das der Fall, wird auch die FERTIG-Flagge dekrementiert (auf 11111111 gesetzt), andernfalls bleibt sie unverändert. Danach geht es zu ANZEIGE:

```
          BNE ANZEIGE    Spieler pleite?
          DEC FERTIG     wenn ja: Verloren-Flagge setzen
          JMP ANZEIGE
```

***Spieler gewinnt***

Hat der Spieler gewonnen, wird die SIEGER-Flagge inkrementiert (auf 1 gesetzt),

```
GWONNEN   INC SIEGER
```

das Punktekonto erhöht

```
          INC CHIPS
```

und auf Endstand 10 getestet:

```
LDA CHIPS
CMP #10          Konto = 10?
```

Beim Endstand 10 „steigt" die FERTIG-Flagge, anstatt zu sinken:

```
BNE ANZEIGE
INC FERTIG       FERTIG-Flagge setzen
```

Der Programmteil ANZEIGE gibt den neuen Punktstand des Spielers bekannt. Wir erinnern uns, daß dies erst auf Anfrage geschieht, nachdem der Spieler also irgendeine Taste gedrückt hat. Darauf müssen wir also zunächst warten:

```
ANZEIGE   JSR GETKEY
```

Vor der Anzeige werden zunächst alle LEDs gelöscht,

```
LDA #0
STA MASKA
STA MASKB
STA TORA
STA TORB
```

und dann wird die Punktzahl eingelesen:

```
LDX CHIPS
BEQ BEENDEN
```

Ist der Spieler pleite (CHIPS=0), so wird das Spiel bei BEENDEN beendet, andernfalls erscheint der neue Punktstand im entsprechenden LED. Da die LEDs intern von 0 bis 7 numeriert sind, obwohl sie extern als 1 bis 8 erscheinen, muß X zunächst um 1 vermindert werden:

```
DEX
```

Erleuchtet wird die passende LED dann vom Unterprogramm SETMASK, wobei die LED-Nummer im Akkumulator zu stehen hat:

```
TXA
JSR SETMASK
```

Das richtige Muster steht also bereit. Je nachdem, ob der Spieler oder der Computer gewonnen hat, müssen die linken oder die rechten drei unteren LEDs zusätzlich erleuchtet werden (bei einem Unentschieden bleibt die ganze Reihe dunkel). Sehen wir uns das an:

```
            LDA SIEGER
            BEQ BEENDEN     Unentschieden: LEDs unverändert
            BMI SC
```

Hat der Spieler verloren, geht es bei Marke SC weiter, hat er gewonnen, gehen die drei linken unteren LEDs an:

```
            LDA #$0E        Spielersieg: linke LEDs an
            JMP SC0
```

Der Spieler hat verloren:

```
SC          LDA #$B0        Spielerniederlage: rechte LEDs an
```

Das Muster für die linken oder rechten LEDs steht in A und kann jetzt ausgegeben werden:

```
SC0         ORA TORB
            STA TORB
```

***Ende des Spiels***

Die BEENDEN-Routine setzt den Schlußpunkt: Ist das Konto weder 0 noch 10, werden die Karten neu verteilt:

```
BEENDEN     JSR DELAY2
            LDA FERTIG
            BNE EN0
            JMP START
```

Andernfalls muß die FERTIG-Flagge getestet werden, um festzustellen, ob der Spieler gewonnen oder verloren hat. Im letzten Fall leuchtet die untere LED-Reihe auf, und das Spiel wird beendet:

```
EN0         BPL EN1         Flagge=1: Siegsituation
            LDA #$BE        komplette untere LED-Reihe
            STA TORB
            RTS             Rücksprung in den Monitor
```

In der Siegsituation blinkt ein LED-Quadrat vor dem Rücksprung in den Monitor:

```
EN1         LDA #$FF
            STA MASKA
            LDA #$01
            STA MASKB
            RTS
```

**Die Unterprogramme**

***Unterprogramm SETBIT***

Diese Routine erzeugt das erforderliche Muster zum Erleuchten einer bestimmten LED, deren Nummer (von 0 bis 9) beim Einsprung im Akkumulator steht. Beim Rücksprung enthält der Akkumulator dann das entsprechend positionierte Bit zum Erleuchten dieser LED. Ist die logische LED-Nummer größer als 7, so ist das Carrybit gesetzt, damit die Ausgabe zu Tor B und nicht zu Tor A geht. Außerdem steht die externe LED-Nummer (1 bis 10) im Y-Register.

Zunächst wird die LED-Nummer ins Y-Register gerettet

```
SETBIT    TAY          logische Nummer aufbewahren
```

und dann mit dem „Grenzwert" 7 verglichen:

```
          CMP #8
          BCC SB0
```

Bei einem Wert größer als 7 subtrahieren wir 8:

```
          SBC #8       wenn > 7: subtrahieren
```

**Übung 10-1:** *Müßte vor dem SBC-Befehl nicht das Carrybit gesetzt werden?*

Jetzt ist die Zahl im Akkumulator mit Sicherheit zwischen 0 und 7, sie wandert ins X-Register:

```
SB0       TAX
```

Es folgt die bekannte Bitverschiebung in den Akkumulator. Dazu muß das Carrybit gesetzt

```
          SEC          Vorbereitung zum Rotieren
```

und der Akkumulator leer sein:

```
          LDA #0
```

Und das Rangiermanöver kann beginnen:

```
SBSCHL    ROL A
          DEX
          BPL SBSCHL
```

Beachten Sie, daß X als Bitzähler fungiert. Der Akkumulator ist nun richtig konditioniert. Die externe LED-Nummer ist allerdings um 1 größer als der

ursprünglich mitgebrachte Wert, der in Y bereitsteht:

```
INY                 LED-Nr. justieren
```

Müssen LED 9 oder 10 erleuchtet werden, muß dies wie angedeutet aus der Carry-Flagge ersichtlich sein, damit Tor B statt Tor A angesprochen werden kann:

```
CPY #9              Carry für Tor B setzen
RTS
```

**Übung 10-2:** *Vergleichen Sie diese Routine mit der LICHT-Routine des vorigen Kapitels.*

**Übung 10-3:** *Wie wurde im vorletzten Befehl das Carrybit gesetzt?*

***Unterprogramm LICHT***

Diese Routine teilt die nächste Karte an den Geber (Computer) aus. Sie muß sich zunächst eine Zufallszahl besorgen und dann prüfen, ob die dazu gehörige LED nicht bereits erleuchtet ist. Ist sie das nicht, so kann diese Zahl verwendet werden, um das den Geber charakterisierende Dauerlicht zu erzeugen:

```
LICHT       JSR RANDOM
```

Bei der Diskussion der RANDOM-Routine weiter unten werden wir sehen, wie eine Zufallszahl nicht nur erzeugt, sondern auch daraufhin überprüft wird, ob die entsprechende LED nicht bereits leuchtet. Hier müssen wir also nur noch das richtige Bit im Akkumulator setzen. Das leistet die oben besprochene SETBIT-Routine:

```
JSR SETBIT
```

Wieder muß die Carry-Flagge auf Tor A bzw. Tor B getestet werden:

```
BCS LL0
```

Ist Tor A gemeint, so wird das neue Bit mittels ORA dort „hineinoperiert“,

```
ORA TORA
STA TORA
```

und anschließend wird der Kartenwert im Akkumulator wiederhergestellt: Die SETBIT-Routine hatte diesen Wert ins Y-Register gerettet:

```
TYA
RTS
```

Im Fall, daß Tor B gemeint ist, passiert das Entsprechende:

```
LL0     ORA TORB
        STA TORB
        TYA             Wert wiederholen
        RTS
```

### *Unterprogramm BLINKEN*

Die Arbeitsweise dieser Routine entspricht genau derjenigen der obigen LICHT-Routine, mit dem Unterschied, daß BLINKEN ein Blinklicht setzt. Wie Sie sehen, ist in dieser Routine auch die SETMASK-Routine enthalten, die die jeweilige LED zum Blinken bringt und mit der Nummer dieser LED im Akkumulator zurückspringt:

```
BLINKEN JSR RANDOM      Zufallszahl holen
SETMASK JSR SETBIT
        BCS BL0         verzweigen nach Tor B
        ORA MASKA
        STA MASKA
        TYA             Wert wiederherstellen
        RTS
BL0     ORA MASKB
        STA MASKB
        TYA
        RTS
```

### *Unterprogramm RANDOM*

Diese Routine erzeugt eine Zufallszahl zwischen 0 und 9, die noch nicht benutzt wurde, die also keiner bereits erleuchteten LED entspricht. Der entsprechende Zahlenwert befindet sich beim Rücksprung im Akkumulator.

```
RANDOM  JSR RANDER      Zahl zwischen 0 und 255 holen
```

Das Unterprogramm RANDER ist der aus vorangegangenen Kapiteln bekannte Zufallszahlengenerator. Um wieder den Zahlenbereich 0 bis 9 einzugrenzen, bedienen wir uns diesmal einer anderen Methode: Alle Zahlen über 9 werden einfach nicht berücksichtigt:

```
AND #$0F
CMP #10
BCS RANDOM
```

**Übung 10-4:** *Haben Sie eine andere Idee, wie ein Zahl zwischen 0 und 9 erzeugt werden könnte? (In früheren Kapiteln wurde eine entsprechende Methode vorgestellt.)*

Nachdem wir nun eine Zahl zwischen 0 und 9 haben, müssen wir das entsprechende Bitmuster erzeugen und in TEMP speichern:

```
        JSR SETBIT      Bitposition setzen
        STA TEMP
```

Es folgt die Probe, ob das entspechende Bit schon gesetzt ist. Dazu müssen wir Tor A und Tor B gesondert betrachten:

```
        BCS RN0         Tor A oder B?
```

Wir müssen nun für Tor A feststellen, welche LEDs bereits leuchten. Dazu kombinieren wir die Muster für blinkende (Maske A) und stetig leuchtende (Maske B) LEDs:

```
        LDA MASKA
        ORA TORA        Tor und Maske vereinigen
```

Es folgt die Probe, ob das Bit, das wir setzen wollen, schon gesetzt ist:

```
        JMP RN1
```

Ist das der Fall, brauchen wir eine neue Zahl zwischen 0 und 9,

```
RN1     AND TEMP
        BNE RANDOM
```

andernfalls können wir mit dem internen LED-Wert im Akkumulator zurückspringen:

```
        DEY
        TYA
        RTS
```

Sollte eine LED von Tor B erleuchtet werden, verläuft der Vorgang entsprechend:

```
RN0     LDA MASKB
        ORA TORB
RN1     AND TEMP
        BNE RANDOM
        DEY
        TYA
        RTS
```

***Unterprogramm RANDER***

Diese Routine, die eine Zufallszahl zwischen 0 und 255 erzeugt, wurde in früheren Kapiteln bereits erläutert.

*Verzögerungsprogramme*

Wir haben in diesem Programm zwei Verzögerungsroutinen: DELAY erzeugt eine Verzögerung von etwa einer halben Sekunde, DELAY2 das Doppelte, also etwa eine Sekunde. Die Indexregister X und Y erhalten für die Doppelschleife den Wert FF:

```
DELAY2    JSR DELAY
DELAY     LDA #$FF
          TAY
D0        TAX
D1        DEX
          LDA #$FF
          BNE D1
          DEY
          BNE D0
          RTS
```

**Übung: 10-5:** *Berechnen Sie die genauen Verzögerungszeiten dieser Routinen.*

*Der Interrupt*

Die Unterbrechungsroutine bringt mit Hilfe von MASKA und MASKB die LEDs zum Blinken. Die Arbeitsweise dieses Programmteils, bei dem keine Register verändert wurden, wurde im letzten Kapitel erläutert.

```
          PHA
          LDA TORA
          EOR MASKA
          STA TORA
          LDA TORB
          EOR MASKB
          STA TORB
          LDA T1LL
          PLA
          RTI
```

## ZUSAMMENFASSUNG

Trotz der einfachen Geber-Strategie ist dieses Programm komplexer als die meisten anderen. Licht- und Toneffekte begleiten die meisten logischen Schritte des Algorithmus. Bemerkenswert ist, mit wie wenig Speicherplatz ein recht komplexes Spiel verwirklicht wird.

**Übung 10-6:** *Das Programm setzt voraus, daß bei Spielbeginn der RND-Inhalt brauchbar zufällig ist. Können Sie zur Vergrößerung der Zufälligkeit einen zusätzlichen Befehl in den Initialisierungsteil dieses Programms implementieren? (Hinweis: Siehe vorangegangene Programme.)*

**Übung 10-7:** *Werden in der BEENDEN-Routine beide Befehle „BNE EN0" und „JMP START" gebraucht? Wenn nicht: Unter welchen Voraussetzungen würden sie gebraucht?*

**Übung 10-8:** *Eine Routine, die sich selbst aufruft, bezeichnet man als rekursiv. Ist DELAY2 rekursiv?*

```
                    ;  '17 UND  4'
                    ;BENUTZT WIRD EIN  BLATT  MIT 10 KARTEN. DIE KARTEN, DIE DER
                    ;SPIELER ERHÄLT, WERDEN DURCH BLINKENDE LEDS DARGESTELLT, UND
                    ;DIE KARTEN DES COMPUTERS ERSCHEINEN ALS STETIG LEUCHTENDE LEDS.
                    ; GEMISCHT' WIRD VON EINEM ZUFALLSGENERATOR. DIE GESAMTAUGEN-
                    ;ZAHLEN VON SPIELER UND COMPUTER WERDEN IN DEN ZERO-PAGE-ADRES-
                    ;SEN 'SPAUG' UND 'COMAUG' GESPEICHERT. TOR A UND TOR B SIND DIE
                    ;AUSGANGSTORE FüR DIE LED-ANZEIGE. DEN BLINKEFFEKT ERZEUGT EINE
                    ;UNTERBRECHUNGSROUTINE MIT DEN MASKEN 'MASKA' UND 'MASKB'.
                    ; FERTIG' UND 'SIEGER' SIND STATUSFLAGGEN FÜR DIE ERMITTLUNG DES
                    ;SPIELENDES UND DES GEWINNERS DER JEWEILIGEN SPIELRUNDE.
                    ;
                    ACCESS    = $8886
                    INTVECL   = $A67E
                    INTVECH   = $A67F
                    IER       = $A00E
                    ACR       = $A00B
                    T1LL      = $A004
                    T1CH      = $A005
                    DDRA      = $A003
                    DDRB      = $A002
                    T0RA      = $A001
                    T0RB      = $A000
                    MASKA     = $C2
                    MASKB     = $C3
                    CHIPS     = $C1
                    FERTIG    = $C0
                    SPAUG     = $C4
                    COMAUG    = $C5
                    TEMP      = $C6
                    RND       = $C7
                    SIEGER    = $CD
                    GETKEY    = $100
                          * = $200
                    ;
                    ;DAS PROGRAMM BEGINNT MIT DER INITIALISIERUNG DES ZEITGEBERS
                    ;UND DES UNTERBRECHUNGSVEKTORS. DIE AUSGANGSTORE WERDEN GESETZT
                    ;UND DIE STATUSFLAGGEN GELÖSCHT.
                    ;
0200: 20 86 8B      '17+4'    JSR ACCESS     ;GESCHÜTZTEN SPEICHERBEREICH ÖFFNEN
0203: A9 EA                   LDA #$EA       ;UNTERBRECHUNGSVEKTOR LOW BYTE
0205: BD 7E A6                STA INTVECL
0208: A9 03                   LDA #3         ;UNTERBRECHUNGSVEKTOR HIGH BYTE
020A: BD 7F A6                STA INTVECH
020D: A9 7F                   LDA #$7F       ;INTERRUPT ENABLE REGISTER LÖSCHEN
020F: BD 0E A0                STA IER
0212: A9 C0                   LDA #$C0       ;ZEITGEBER 1 UNTERBRECHUNG ZULASSEN
0214: BD 0E A0                STA IER
0217: A9 40                   LDA #$40       ;FREILAUFMODUS FÜR ZEITGEBER 1
0219: BD 0B A0                STA ACR
```

*Abb. 10.12: Programm „17+4"*

```
021C: A9 FF              LDA #$FF
021E: 8D 04 A0           STA T1LL       ;HILFSREGISTER ZEITGEBER 1 SETZEN
0221: 8D 05 A0           STA T1CH       ;UND ZEITGEBER STARTEN
0224: 5B                 CLI            ;UNTERBRECHUNGEN ZULASSEN
0225: 8D 03 A0           STA DDRA       ;LEDTORE AUF AUSGABE SETZEN
0228: 8D 02 A0           STA DDRB
022B: D8                 CLD
022C: A9 05              LDA #5         ;SPIELERKONTO AUF 5 SETZEN
022E: 85 C1              STA CHIPS
0230: A9 00              LDA #0         ;'FERTIG'-FLAGGE LÖSCHEN
0232: 85 C0              STA FERTIG
                ;
                ;NEUES SPIEL: ANZEIGE WIRD GELÖSCHT, BEIDE SPIELER ERHALTEN
                ;STARTWERTE. UND DIE ENTSPRECHENDEN LEDS WERDEN ERLEUCHTET.
                ;
0234: 85 C2     START    STA MASKA      ;BLINKMASKEN LÖSCHEN (AKKUMULATOR MUSS
0236: 85 C3              STA MASKB      ;0 ENTHALTEN)
0238: 8D 01 A0           STA TORA       ;LEDS LÖSCHEN
023B: 8D 00 A0           STA TORB
023E: 85 CD              STA SIEGER     ;FLAGGE LÖSCHEN
0240: 20 0F 03           JSR BLINKEN    ;ZUFALLS-LED BLINKEN LASSEN
0243: 85 C4              STA SPAUG      ;SPIELER-AUGEN SPEICHERN
0245: 20 F7 02           JSR LICHT      ;ZUFALLS-LED STETIG LEUCHTEN LASSEN
0248: 85 C5              STA COMAUG     ;COMPUTER-AUGEN SPEICHERN
                ;
                ;TASTENDRUCK HOLEN: 'A' = NOCH EINE KARTE, 'C' = COMPUTER IST
                ;AN DER REIHE. ANDERE TASTEN WERDEN IGNORIERT
                ;
024A: 20 00 01  FRAGEN   JSR GETKEY     ;TASTENEINGABE HOLEN
024D: C9 0A              CMP #$0A       ;SPIELER NOCH EINE KARTE?
024F: F0 07              BEQ SPLNMT     ;WENN JA: VERZWEIGEN
0251: C9 0C              CMP #$0C       ;'COMPUTER-DRAN'-TASTE?
0253: F0 12              BEQ GEBER      ;JA!
0255: 4C 4A 02           JMP FRAGEN     ;UNZULÄSSIGE TASTE: VON VORN
0258: 20 0F 03  SPLNMT   JSR BLINKEN    ;ZUFALLS-LED SETZEN
0258: 18                 CLC
025C: 65 C4              ADC SPAUG      ;AUGENZAHL AKTUALISIEREN
025E: 85 C4              STA SPAUG
0260: C9 0E              CMP #14        ;AUGENZAHL ÜBERPRÜFEN
0262: 90 E6              BCC FRAGEN     ;BEI WENIGER ALS 14: OK!
0264: 4C B7 02           JMP VRLORN     ;ANDERNFALLS 'GESCHMISSEN'
0267: 20 5D 03  GEBER    JSR DELAY      ;COMPUTER 'DENKT NACH': VERZÖGERN
026A: A5 C5              LDA COMAUG     ;KANN NOCH EINE KARTE RISKIERT WERDEN?
026C: C9 0A              CMP #10
026E: B0 0F              BCS BESSER     ;NEIN: GEWINNER ERMITTELN
0270: 20 F7 02           JSR LICHT      ;JA: WEITERE ZUFALLS-LED SETZEN
0273: 18                 CLC
0274: 65 C5              ADC COMAUG     ;AUGENZAHL AKTUALISIEREN
0276: 85 C5              STA COMAUG
0278: C9 0E              CMP #14        ;WENIGER ALS 14?
027A: 90 EB              BCC GEBER      ;WENN JA: NOCH EINE KARTE?
027C: 4C 92 02           JMP GWONNEN    ;'GESCHMISSEN': SPIELER GEWINNT
                ;
                ;GEWINNER ERMITTELN: 'SIEGER'-FLAGGE ENTSPRECHEND SETZEN UND
                ;KONTOSTAND AKTUALISIEREN ('VRLORN' ODER 'GWONNEN'). BEI UNENT-
                ;SCHIEDEN GESCHIEHT NICHTS.
                ;
027F: A5 C5     BESSER   LDA COMAUG     ;AUGENZAHLEN VERGLEICHEN
0281: C5 C4              CMP SPAUG
0283: F0 19              BEQ ANZEIGE    ;GLEICHSTAND: KEINE ANDERUNG
0285: 90 0B              BCC GWONNEN    ;SPIELER HAT MEHR AUGEN
0287: C6 CD     VRLORN   DEC SIEGER     ;'VERLOREN'-ROUTINE
0289: C6 C1              DEC CHIPS      ;KONTO AKTUALISIEREN
028B: D0 11              BNE ANZEIGE    ;SPIELER PLEITE?
028D: C6 C0              DEC FERTIG     ;JA: SPIELENDE-FLAGGE SETZEN
028F: 4C 9E 02           JMP ANZEIGE
0292: E6 CD     GWONNEN  INC SIEGER     ;SIEG-ROUTINE
0294: E6 C1              INC CHIPS      ;KONTO AKTUALISIEREN
0296: A5 C1              LDA CHIPS      ;KONTO AUF MAXIMUM PRÜFEN
0298: C9 0A              CMP #10        ;HÖCHSTPUNKTZAHL 10?
029A: D0 02              BNE ANZEIGE
029C: E6 C0              INC FERTIG     ;WENN JA: SPIELENDE-FLAGGE SETZEN
```

*Abb. 10.12: Programm „17+4" (Fortsetzung)*

```
                        ;
                        ;SPIELERKONTO DURCH ENTSPRECHENDE LED ANZEIGEN. SIEGER DES
                        ;DURCHGANGS WIRD IN UNTERSTER LEDREIHE ANGEZEIGT. AUF SPIELENDE
                        ;PRüFEN, LEDS ENTSPRECHEND ERLEUCHTEN UND BEI SPIELENDE PROGRAMM
                        ;BEENDEN (VORAUSGESETZT WIRD MONITORADRESSE AUF DEM STAPEL).
                        ;
029E: 20 00 01  ANZEIGE   JSR GETKEY     ;AUF TASTENDRUCK WARTEN
02A1: A9 00               LDA #0         ;LEDS LÖSCHEN
02A3: 85 C2               STA MASKA
02A5: 85 C3               STA MASKB
02A7: 8D 01 A0            STA TORA
02AA: 8D 00 A0            STA TORB
02AD: A6 C1               LDX CHIPS      ;KONTOSTAND EINLESEN
02AF: F0 13               BEQ BEENDEN
02B1: CA                  DEX            ;INTERN JUSTIEREN
02B2: 8A                  TXA
02B3: 20 12 03            JSR SETMASK
02B6: A5 CD               LDA SIEGER     ;WER HAT GEWONNEN?
02B8: F0 0F               BEQ BEENDEN    ;BEI UNENTSCHIEDEN: KEINE LEDS
02BA: 30 05               BMI SC
02BC: A9 0E               LDA #$0E       ;SPIELERSIEG: 3 LEDS LINKS UNTEN AN
02BE: 4C C3 02            JMP SC0
02C1: A9 B0       SC      LDA #$B0       ;COMPUTERSIEG: 3 LEDS RECHTS UNTEN
02C3: 0D 00 A0    SC0     ORA TORB       ;LED-TOR SETZEN
02C6: 8D 00 A0            STA TORB
02C9: 20 5A 03    BEENDEN JSR DELAY2     ;ANZEIGE HALTEN
02CC: A5 C0               LDA FERTIG     ;AUF SPIELENDE PRüFEN
02CE: D0 03               BNE EN0
02D0: 4C 34 02            JMP START      ;FLAGGE=0: NEUES BLATT
02D3: 10 06       EN0     BPL EN1        ;FLAGGE=1: SIEGBEDINGUNG
02D5: A9 BE               LDA #$BE       ;VOLLSTÄNDIGE LEDREIHE SETZEN
02D7: 8D 00 A0            STA TORB
02DA: 60                  RTS            ;RÜCKSPRUNG IN DEN MONITOR
02DB: A9 FF       EN1     LDA #$FF       ;BLINKENDES LED-QUADRAT SETZEN
02DD: 85 C2               STA MASKA
02DF: A9 01               LDA #1
02E1: 85 C3               STA MASKB
02E3: 60                  RTS            ;ZURÜCK ZUM MONITOR
                        ;
                        ;UNTERPROGRAMM 'SETBIT'
                        ;EINSPRUNG MIT WERT ZWISCHEN 0 UND 9 IM AKKUMULATOR, RÜCKSPRUNG
                        ;MIT EXTERNEM WERT ZWISCHEN 1 UND 10 IM Y-REGISTER UND POSITIO-
                        ;NIERTEM BIT IM AKKUMULATOR. CARRY=1 ZEIGT TOR B AN.
                        ;
02E4: A8          SETBIT  TAY            ;LOGISCHEN WERT SICHERN
02E5: C9 08               CMP #8         ;WERTE KLEINER 8 AUSSONDERN
02E7: 90 02               BCC SB0
02E9: E9 08               SBC #8         ;SUBTRAHIEREN FALLS > 7
02EB: AA          SB0     TAX            ;INDEX-REGISTER SETZEN
02EC: 38                  SEC            ;BIT ZUM VERSCHIEBEN SETZEN
02ED: A9 00               LDA #0
02EF: 2A          SBSCHL  ROL A          ;BIT POSITIONIEREN
02F0: CA                  DEX
02F1: 10 FC               BPL SBSCHL
02F3: C8                  INY            ;INTERN AUF EXTERN JUSTIEREN
02F4: C0 09               CPY #9         ;CARRY FÜR TOR B SETZEN
02F6: 60                  RTS
                        ;
                        ;UNTERPROGRAMM 'LICHT
                        ;SETZT NOCH NICHT BENUTZTE ZUFALLS-LED ALS DAUERLICHT. ZUFALLS-
                        ;ZAHL SETZT BIT IM ENTSPRECHENDEN TOR UND SPRINGT MIT DIESEM
                        ;BIT-MUSTER IM AKKUMULATOR ZURÜCK.
                        ;
02F7: 20 23 03    LICHT   JSR RANDOM     ;ZUFALLSZAHL HOLEN
02FA: 20 E4 02            JSR SETBIT     ;BIT-POSITION IM AKKUMULATOR HOLEN
02FD: B0 0B               BCS LL0        ;VERZWEIGEN FALLS TOR B
02FF: 0D 01 A0            ORA TORA       ;LED SETZEN
0302: 8D 01 A0            STA TORA
0305: 98                  TYA            ;EXTERNEN WERT NACH Y RETTEN
0306: 60                  RTS
0307: 0D 00 A0    LL0     ORA TORB       ;LED IN TOR B SETZEN
030A: 8D 00 A0            STA TORB
```

*Abb. 10.12: Programm „17+4" (Fortsetzung)*

```
030D: 98                TYA
030E: 60                RTS
                ;
                ;UNTERPROGRAMM 'BLINKEN'
                ;SETZT NOCH NICHT BENUTZTE ZUFALLS-LED ALS BLINKLICHT. EXTERNER
                ;LED-WERT BEIM RUCKSPRUNG IM AKKUMULATOR. HOLT ZUFALLSZAHL UND
                ;BENUTZT DANN 'SETMASK'-ROUTINE.
                ;UNTERPROGRAMM 'SETMASK'
                ;EINSPRUNG MIT LOGISCHEM LED-WERT. ENTSPRECHENDE LED WIRD DANN
                ;GESETZT. RUCKSPRUNG MIT DIESEM WERT IM AKKUMULATOR.
                ;
030F: 20 23 03  BLINKEN   JSR RANDOM    ;ZUFALLSZAHL HOLEN
0312: 20 E4 02  SETMASK   JSR SETBIT
0315: 80 05               BCS BL0       ;VERZWEIGEN FALLS TOR B GESETZT
0317: 05 C2               ORA MASKA     ;MASKE A SETZEN
0319: 85 C2               STA MASKA
031B: 98                  TYA           ;EXTERNEN WERT ÜBERTRAGEN
031C: 60                  RTS
031D: 05 C3     BL0       ORA MASKB     ;MASKE B SETZEN
031F: 85 C3               STA MASKB
0321: 98                  TYA
0322: 60                  RTS
                ;
                ;UNTERPROGRAMM 'RANDOM'
                ;ERZEUGT ZUFALLSZAHL ZWISCHEN 0 UND 9 FÜR EINE NOCH NICHT ER-
                ;LEUCHTETE LED. WERT BEIM RÜCKSPRUNG IM AKKUMULATOR.
                ;
0323: 20 47 03  RANDOM    JSR RANDER    ;ZUFALLSZAHL ZWISCHEN 0 UND 255 HOLEN
0326: 29 0F               AND #$0F      ;HÖHERWERTIGEN NIBBLE MASKIEREN
0328: C9 0A               CMP #10       ;AUF WERT ZWISCHEN 0 UND 9 BEGRENZEN
032A: B0 F7               BCS RANDOM
032C: 20 E4 02            JSR SETBIT    ;BIT POSITIONIEREN
032F: 85 C6               STA TEMP      ;ZWISCHENSPEICHERN
0331: B0 0B               BCS RN0       ;TOR A ODER TOR B?
0333: A5 C2               LDA MASKA     ;TOR MIT MASKE KOMBINIEREN
0335: 0D 01 A0            ORA TORA
0338: 4C 40 03            JMP RN1
033B: A5 C3     RN0       LDA MASKB     ;TOR MIT MASKE KOMBINIEREN
033D: 0D 00 A0            ORA TORB
0340: 25 C6     RN1       AND TEMP      ;SPEZIFISCHES BIT BETRACHTEN
0342: D0 DF               BNE RANDOM    ;WENN BIT SCHON GESETZT: NOCHMAL
0344: 88                  DEY           ;INTERNE DARSTELLUNG
0345: 98                  TYA           ;VOR RÜCKSPRUNG: WERT NACH A
0346: 60                  RTS
                ;
                ;UNTERPROGRAMM 'RANDER'
                ;ERZEUGT ZUFALLSZAHL ZWISCHEN 0 UND 255. BENUTZT ZAHLEN A,B,C,D,
                ;E,F IN ADRESSEN RND BIS RND+5. ADDIERT B+E+F+I UND LEGT IN A
                ;AB, DANN GEHT A NACH B, B NACH C USW. ZUFALLSZAHL BEIM RÜCK-
                ;SPRUNG IM AKKUMULATOR.
                ;
0347: 38        RANDER    SEC           ;GESETZTES CARRY = ADDITION VON 1
0348: A5 C8               LDA RND+1     ;B, D UND F ADDIEREN
034A: 65 CB               ADC RND+4
034C: 65 CC               ADC RND+5
034E: 85 C7               STA RND
0350: A2 04               LDX #4        ;ZAHLEN NACH UNTEN SCHIEBEN
0352: B5 C7     RDSCHL    LDA RND,X
0354: 95 C8               STA RND+1,X
0356: CA                  DEX
0357: 10 F9               BPL RDSCHL
0359: 60                  RTS
                ;UNTERPROGRAMM 'DELAY'
                ;VERZÖGERUNGSROUTINE MIT DOPPELTER VERZÖGERUNG BEI 'DELAY2'-
                ;EINSPRUNG. VERZÖGERUNG ETWA 0.5 SEKUNDEN.
                ;
035A: 20 5D 03  DELAY2    JSR DELAY
035D: A9 FF     DELAY     LDA #$FF      ;SCHLEIFENZÄHLER SETZEN
035F: A8                  TAY
0360: AA        D0        TAX
0361: CA        D1        DEX
0362: A9 FF               LDA #$FF
```

*Abb. 10.12: Programm „17+4" (Fortsetzung)*

```
0364: D0 FB              BNE D1
0366: 88                 DEY
0367: D0 F7              BNE D0
0369: 60                 RTS

                ;
                ;UNTERBRECHUNGSROUTINE
                ;BLINKEFFEKT DURCH 'EOR' VON AUSGANGSTOREN MIT BLINKMASKEN NACH
                ;ZEITGEBER-SIGNAL. REGISTER BLEIBEN UNVERANDERT. INTERRUPT-
                ;FLAGGE WIRD VOR RÜCKSPRUNG GELÖSCHT.
                ;
                         *=$03EA
03EA: 48                 PHA             ;AKKUMULATOR AUFBEWAHREN
03EB: AD 01 A0           LDA TORA        ;TORE MIT MASKEN KOMPLEMENTIEREN
03EE: 45 C2              EOR MASKA
03F0: 8D 01 A0           STA TORA
03F3: AD 00 A0           LDA TORB
03F6: 45 C3              EOR MASKB
03F8: 8D 00 A0           STA TORB
03FB: AD 04 A0           LDA TILL        ;ZEITGEBER-UNTERBRECHUNGSBIT LÖSCHEN
03FE: 68                 PLA             ;AKKUMULATOR WIEDERHOLEN
03FF: 40                 RTI

SYMBOLTABELLE:
ACCESS    8886    INTVECL   A67E    INTVECH   A67F    IER       A00E
ACR       A00B    TILL      A004    T1CH      A005    DDRA      A003
DDRB      A002    TORA      A001    TORB      A000    MASKA     00C2
MASKB     00C3    CHIPS     00C1    FERTIG    00C0    SPAUG     00C4
COMAUG    00C5    TEMP      00C6    RND       00C7    SIEGER    00CD
GETKEY    0100    '17+4'    0200    START     0234    FRAGEN    024A
SPLNMT    025B    GEBER     0267    BESSER    027F    VRLORN    02B7
GWONNEN   02?2    ANZEIGE   029E    SC        02C1    SC0       02C3
BEENDEN   02C9    EN0       02D3    EN1       02DB    SETBIT    02E4
SB0       02EB    SBSCHL    02EF    LICHT     02F7    LL0       0307
BLINKEN   030F    SETMASK   0312    BL0       031D    RANDOM    0323
RN0       033B    RN1       0340    RANDER    0347    RDSCHL    0352
DELAY2    035A    DELAY     035D    D0        0360    D1        0361
```

*Abb. 10.12: Programm „17+4" (Fortsetzung)*

# 11

# Künstliche Intelligenz (Tic-Tac-Toe)

## EINFÜHRUNG

Dieses Kapitel zeigt den vollständigen Entwurf eines komplexen Algorithmus zur Lösung der Strategie- und Implementierungsprobleme des Spieles Tic-Tac-Toe. Das recht umfangreiche Programm bedient sich ausgefeilter Auswertungstechniken. Benutzt werden Tabellen-Algorithmen und komplexe Datenstrukturen wie verkettete Listen. Das sorgfältige Studium dieses Programms wird Ihnen einen hohen Kompetenzgrad bei der 6502-Programmierung vermitteln.

## DIE SPIELREGELN

Tic-Tac-Toe wird auf einem Quadrat mit 3 x 3 Feldern gespielt. In die zu Beginn leeren 9 Kästchen dieses Quadrats setzen die Spieler abwechselnd ihre Züge, der Spieler ein „O" und der Computer ein „X". Wer im Verlauf des Spiels zuerst eine waagerechte, senkrechte oder diagonale Dreierreihe mit seinem Symbol zustande bringt, gewinnt das Spiel. (Anmerkung des Übersetzers: Tic-Tac-Toe ist eine vereinfachte Version des asiatischen Brettspiels GOMOKU.) Bild 11.1 zeigt die acht möglichen Gewinnstellungen für eine Partei. Auf unserer LED-Anzeige wird ein kontinuierliches Licht das X-Symbol des Computers repräsentieren und eine blinkende LED das O-Symbol des Spielers.

Sowohl der Computer als auch der Spieler kann den ersten Zug bekommen. Möchte der Spieler beginnen, so muß er die F-Taste drücken. Soll der Computer den ersten Zug machen, kann eine beliebige andere Taste gedrückt werden. Wenn ein Spiel beendet ist, beginnt unmittelbar darauf ein neues. Der Computer hat einen veränderlichen „Intelligenzquotienten", der zwischen den Werten 1 und 15 liegt. Wenn der Computer gewinnt, so erhöht sich sein IQ um 1, verliert er, so vermindert sich sein IQ um 1. Auf diese Weise haben beide Spieler Gewinnchancen. Gewinnt der Spieler, so erklingt ein hoher Ton, verliert er, so zeigt dies ein tiefer Ton an.

## EIN TYPISCHER SPIELVERLAUF

Zu Beginn ist die Anzeige leer, und wir wollen den Computer beginnen lassen, dazu drücken wir irgendeine Taste außer F (bei F macht der Spieler den ersten Zug), z.B. 0. Nach kurzer Zeit ertönt ein Triller, und der Computer zieht wie in

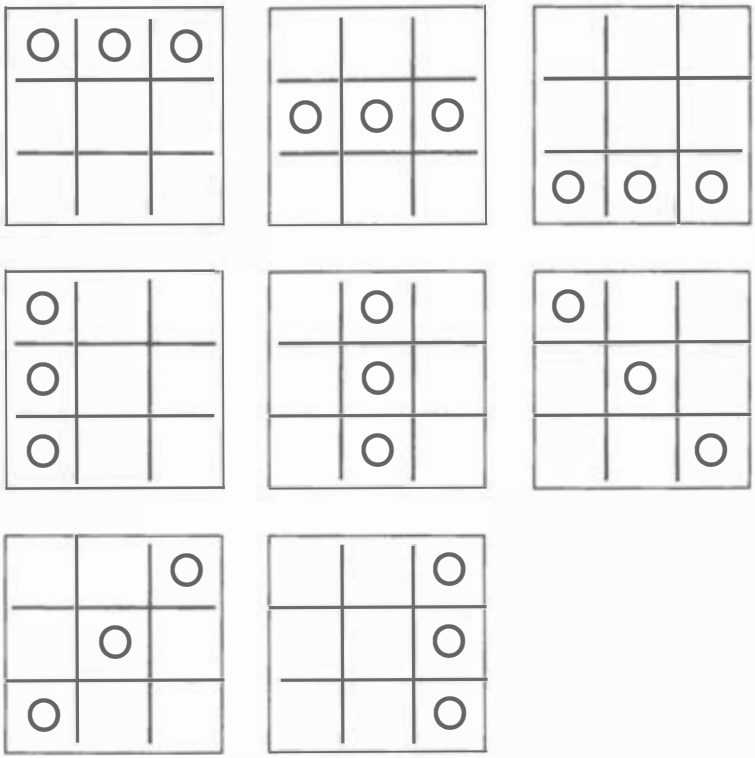


*Abb. 11.1: TIC-TAC-TOE, Gewinnstellungen*

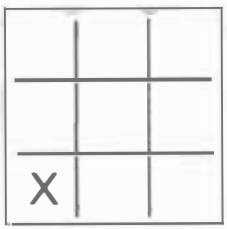



*Abb. 11.2: 1. Computerzug*

Bild 11.2 (X = Computerzug, O = Spielerzug, leer = unbesetztes Feld). Wir besetzen nun das Mittelfeld 5 (Bild 11.3), indem wir Taste „5" drücken. Kurz darauf leuchtet LED 1 auf, und ein Triller ertönt zum Zeichen, daß wir wieder am Zug sind (Bild 11.4).

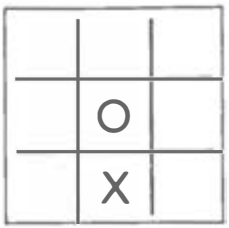



*Abb. 11.3: Unser 1. Zug*

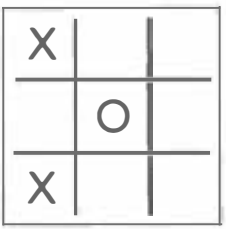



*Abb. 11.4: 2. Computerzug*

Um die Vervollständigung der linken senkrechten Dreierreihe zu verhindern, drücken wir nun „4". Kurz darauf leuchtet LED 6 als die Antwort des Computers auf (Bild 11.5).

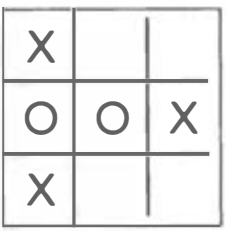



*Abb. 11.5: Nach dem 3. Computerzug*

Jetzt besetzen wir Feld 2, der Computer bald darauf Feld 8 (Bild 11.6). Erneut verhindern wir eine Gewinnstellung für den Computer, indem wir Feld 9 besetzen, worauf unser Gegner die letzte freie Position 3 belegt. (Bild 11.7). Die Schlußstellung ist ein Unentschieden. Alle LEDs blinken kurz, und dann ist das Spielbrett wieder leer. Ein neues Spiel kann beginnen.

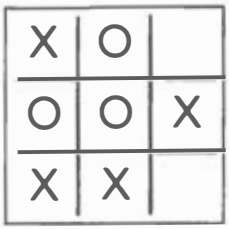
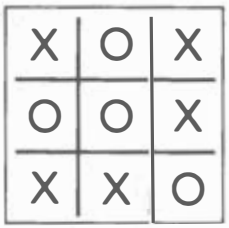



*Abb. 11.6: Nach dem 4. Computerzug*




(UNENTSCHIEDEN)

*Abb. 11.7: Nach dem 5. Computerzug*

**Ein neues Spiel**

Diesmal werden wir anfangen und hoffentlich gewinnen! Nach Drücken der F-Taste „trillert" es, der Computer wartet also auf unseren ersten Zug. Wir spielen „5", der Computer antwortet mit „3", es trillert und wir sind schon wieder dran (Bild 11.8). Auf unsere „4" kommt eine „6" vom Computer, was die Stellung in Bild 11.9 ergibt. Wir müssen nun bei „9" blockieren, und der Computer verhindert daraufhin unseren Sieg mit „1" (Bild 11.10). Jetzt müssen wir die obere waagerechte Dreierreihe verhindern: Wir spielen „2". Der Computer stört uns seinerseits mit „8" (Bild 11.11). Unser letzter Zug „7" bringt wieder nur ein Unentschieden.

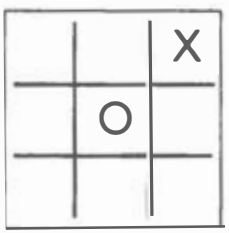



*Abb. 11.8: Zug 1*

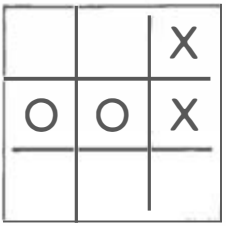



*Abb. 11.9: Zug 2*

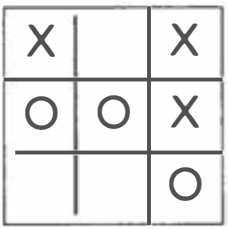



*Abb. 11.10: Zug 3*

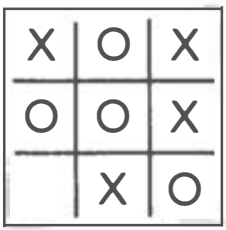



*Abb. 11.11: Zug 4*

Da der Computer so „schlau war, unseren Zug im Zentrum mit einem Diagonalzug zu beantworten, konnten wir erneut nicht gewinnen. Der Computer wird auch einmal einen der Seitenpunkte (2, 4, 6 oder 8) besetzen und uns eine Gewinnchance geben. Hier ist ein Beispiel.

Wir besetzen das Zentrum, und der Computer antwortet mit „6“ (Bild 11.12). Jetzt spielen wir „1“, und der Computer muß „9“ besetzen (Bild 11.13). „3“ ist nun für uns obligatorisch, verschafft uns aber eine „Zwickmühle“. Zug „7“ hilft dem Computer nichts mehr (Bild 11.14): Wir haben mit „2“ den Gewinnzug parat, Bild 11.15 zeigt die Siegstellung. Es sei an dieser Stelle gesagt, daß beim Tic-Tac-Toe der Spieler, der den ersten Zug macht, zumindest ein Unentschieden erzwingen kann, vorausgesetzt, er macht keinen Fehler.


*Abb. 11.12: Zug 1*

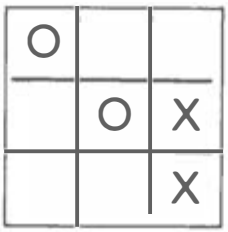
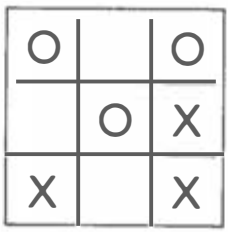



*Abb. 11.13: Zug 2*




"MOVE 3"

*Abb. 11.14: Zug 3*

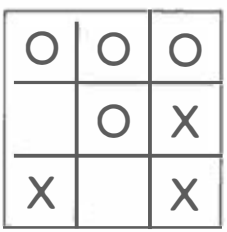



*Abb. 11.15: „Wir gewinnen"*

## DER ALGORITHMUS

Der Algorithmus für das Tic-Tac-Toe-Spiel ist der komplexeste, den wir in diesem Buch noch zu erarbeiten haben. Er gehört in den Bereich der sogenannten „künstlichen Intelligenz". Damit bezeichnet man ein Computerverhalten, das eine menschliche Aktivität simuliert, die man allgemein als „intelligent" bezeichnet. Einen guten Algorithmus für dieses Spiel zu entwickeln, der mit relativ wenig Speicherplatz auskommt, ist durchaus nicht belanglos. In der Vergangenheit sind schon viele Algorithmen vorgeschlagen worden, und es gibt sicher weitere, die gefunden werden können. Wir wollen uns hier mit zwei Strategien genauer auseinandersetzen und uns schließlich eine davon für unser Programm auswählen. Andere mögliche Strategien werden als Übungsaufgaben vorgeschlagen.

### Eine Strategie für den nächsten Zug

Eine mögliche Strategie von vielen, den nächsten Zug zu ermitteln, ist zweifellos die, alle möglichen Spielstellungen und die dazugehörigen besten Züge im Speicher griffbereit zu haben. Aus mathematischer Sicht ist dies die beste Strategie, denn sie garantiert, daß für jede Stellung der beste Zug verfügbar ist. Im Fall unseres kleinen neunfeldrigen Spielbrettes ist dies sogar eine praktische Methode, da die Zahl der möglichen Spielstellungen überschaubar ist. Da wir jedoch bei anderen Spielen bereits den Umgang mit Tabellen gelernt haben, würde uns diese Vorgehensweise kaum nennenswerte neue Programmierfertigkeiten vermitteln. Zudem könnte man diese Methode als „unfair" bezeichnen. Wir werden uns deshalb mit anderen Methoden auseinandersetzen, die auch auf andere Spiele oder auf ein größeres Brett anwendbar sind.

Ein Vorschlag für eine solche Strategie wäre, heuristisch vorzugehen, den Computer also aus Erfahrungen lernen zu lassen. Er würde dabei mit steigender Spielzahl immer besser werden, da er aus seinen Fehlern lernen könnte. Bei dieser Strategie beginnt der Computer mit Zufallsentscheidungen, aber bei genügend verfügbarem Speicherplatz wird er alle seine Entscheidungen aufzeichnen und im weiteren Verlauf feststellen können, welche Züge zu Verluststellungen führten. Diese Züge kann er dann aussortieren und in Zukunft aus dem Spiel lassen. Diese Vorgehensweise wird bei einem vernünftigen „Lern"-Algorithmus also zum Aufbau von Entscheidungstabellen führen, die allerdings das Vorhandensein von genügend Speicherkapazität voraussetzen. Da wir unser Programm innerhalb von 1 KByte unterbringen wollen, kommt diese Strategie also für uns nicht in Frage. Sehen wir uns nach einer anderen Möglichkeit um. Eine prinzipielle Problemlösung ist auch die, nach jedem Zug eine Stellungsauswertung durchzuführen. Dabei muß das Brett von zwei Gesichtspunkten aus untersucht werden. Erstens: Gibt es eine Reihe mit zwei O-Symbolen, so muß das dritte O unbedingt verhindert werden. Zweitens: Sind umgekehrt zwei X-Zeichen in einer Reihe vorhanden, so muß das Programm das dritte X setzen, denn das ist eine Gewinnstellung. Diese beiden Situationen lassen sich recht leicht erkennen. Das eigentliche Problem besteht freilich darin, die Gewinnmöglichkeiten jedes Feldes in jeder Spielsituation zu ermitteln.

### Ein analytischer Algorithmus

Wir wollen uns nun ansehen, wie ein Algorithmus anhand allgemeiner Richtlinien entwickelt wird. Wenn wir dann die Schwachstellen dieses Algorithmus erkennen, werden wir ihn zu verbessern suchen. Dies wird uns eine mögliche Vorgehensweise vor Augen führen, wie die Problemlösung bei einem Strategiespiel zu bewerkstelligen ist.

#### *Das allgemeine Konzept*

Wir gehen davon aus, daß wir das Potential jedes Brettpunktes bestimmen müssen, wobei die beiden Aspekte „Sieg" und „Drohung" entscheidend sind. Das „Sieg"-Potential bezeichnet die Möglichkeit, durch Besetzen eines Brettpunktes eine Gewinnstellung herbeizuführen. Das „Drohung"-Potential ist dann das „Sieg"-Potential für den Gegner.

Zuerst müssen wir eine Methode finden, mit der wir einem bestimmten O-X-Muster auf dem Brett einen Zahlenwert zuordnen können. Dann können wir das Potential, also den strategischen Wert eines Brettpunktes berechnen.

#### *Die Wertberechnung*

Für jede Reihe (Zeile oder Spalte oder Diagonale) gibt es vier prinzipiell verschiedene Konfigurationen (wenn wir den Fall einer bereits kompletten Reihe, in der wir ja nicht mehr spielen können, ausklammern), die in Bild 11.16 dargestellt sind.
Situation A ist die noch völlig leere Reihe, die alle Möglichkeiten offen läßt. Geben wir jeder Position in einer solchen Reihe den Wert 1. In Situation B findet sich bereits ein X in der Reihe. Wenn wir ein weiteres X hier anbringen könnten, wären wir einer Gewinnstellung sehr nahe, das Potential von Situation B ist also höher einzustufen als das von A. Tragen wir dem Rechnung, indem wir jedem freien Feld in einer solchen Reihe den Wert 2 geben.
Situation C in Bild 11.16 enthält ein X und ein O. Da wir in einer solchen Reihe nie mehr zu einer Gewinnstellung kommen können, erhält der noch freie Punkt den Wert 0.
Situation D schließlich hat bereits zwei X-Zeichen. Ein Zug auf das letzte freie Feld würde uns eine Gewinnstellung liefern, diesem Feld gebührt also der Höchstwert, sagen wir 3.

Der nächste Gesichtspunkt ist, daß jedes Feld auf dem Brett sowohl einer waagerechten als auch einer senkrechten Reihe angehört, möglicherweise sogar auch einer diagonalen. Jedes Feld sollte also in zwei oder gar drei Richtungen ausgewertet werden. Tun wir das, und addieren wir dann die Potentiale für jede Richtung. Der Übersichtlichkeit wegen bedienen wir uns einer Auswertungsmatrix, wie sie in Bild 11.17 dargestellt ist: Jedes Feld ist dort in vier Teilquadrate aufgeteilt, in denen wir das Potential dieses Feldes in allen Richtungen eintragen können. Das H-Teilquadrat soll das horizontale Potential bezeichnen, V das vertikale, D das diagonale, und T die Summe der drei (T = total). Sie

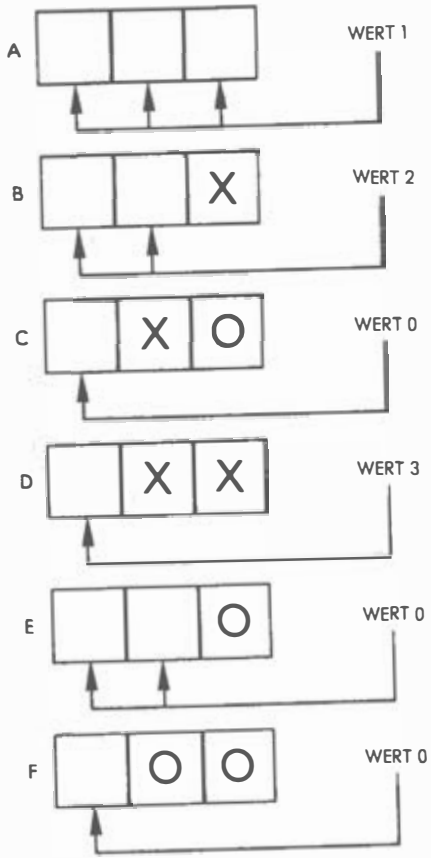



*Abb. 11.16: Die 6 Kombinationen*

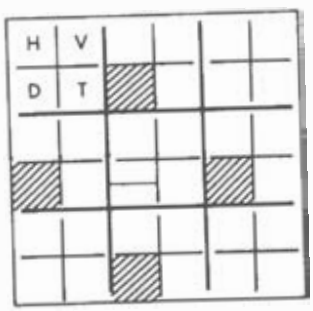



*Abb. 11.17: Auswertungsmatrix*

sehen, daß vier Felder keinen D-Wert haben, da sie keiner kompletten Diagonalen angehören, und daß das Zentrumsfeld zwei D-Werte hat, weil es beiden Diagonalen angehört.

Wenn unser Algorithmus für jedes Feld das gesamte Gewinn- und Verlustpotential ermittelt hat, muß er sich nun für den besten nächsten Zug entscheiden. Offensichtlich ist das das Feld mit dem höchsten Siegpotential.

Testen wir nun die Leistungsfähigkeit unseres Algorithmus an einigen konkreten Beispielen. Dazu betrachten wir ein paar typische Stellungen, lassen sie von unserem Algorithmus auswerten, und dann werden wir sehen, ob seine Entscheidungen sinnvoll sind.

### *Test des ersten Algorithmusansatzes*

Betrachten wir die Spielstellung in Bild 11.18, der Spieler (O) ist am Zug. Wir werten von zwei Gesichtspunkten aus: Potential für X und Drohung durch O. Das Feld mit dem höchsten Wert in jeder Auswertungsmatrix ist das Feld, wo wir unseren Zug machen müssen.

Wir beginnen mit der ersten horizontalen Reihe. Da dort ein O ist, ist das H-Potential für den Spieler dort 0 (siehe Situation E in Bild 11.16). Diese erste Matrixeintragung ist in Bild 11.19 zu sehen. In der zweiten waagerechten Reihe

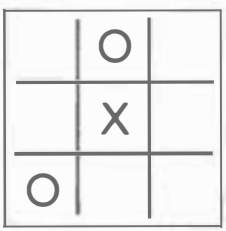



*Abb. 11.18: Testfall 1*

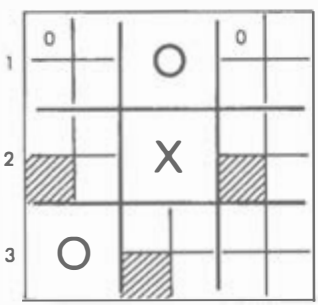



*Abb. 11.19: Auswertungsmatrix: Potential Reihe 1*

beziehen wir uns auf Situation B in Bild 11.16 und erhalten das Potential 2. Reihe 3 entspricht wie Reihe 1 der E-Situation. Nach der horizontalen Auswer-

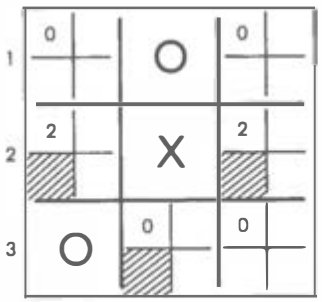



*Abb. 11.20: Auswertung des horizontalen Potentials*

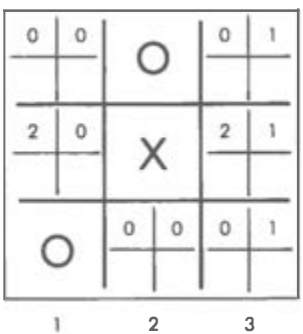



*Abb. 11.21: Auswertung des vertikalen Potentials*

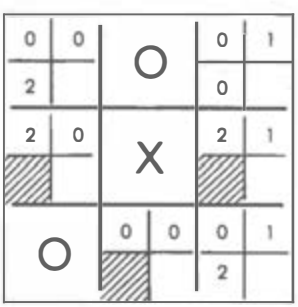



*Abb. 11.22: Auswertung des diagonalen Potentials*

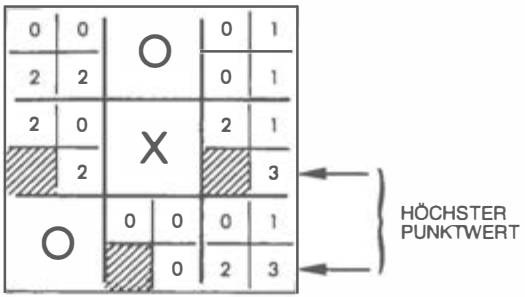



*Abb. 11.23: Endgültiges Potential*

tung ergibt sich demnach Bild 11.20. Es folgt die vertikale Auswertung mit dem Endergebnis Bild 11.21:
Die ersten beiden vertikalen Spalten enthalten 0-Potentiale (Situationen E und C in Bild 11.16), die dritte dagegen das Potential 1, es ist die Situation A.

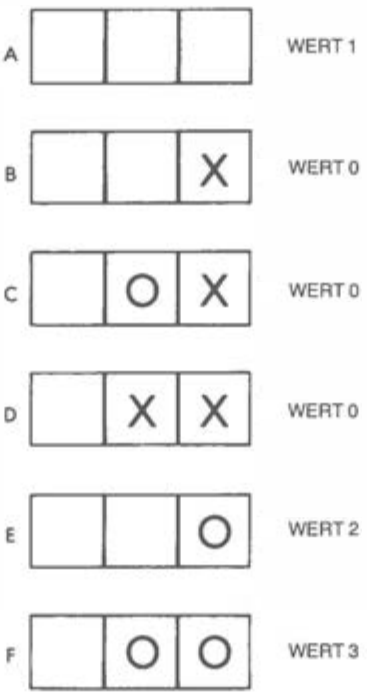



*Abb. 11.24: Auswertung für Spieler „O"*

Schließlich folgt die Auswertung der beiden Diagonalen, was die Matrix Bild 11.22 ergibt, und zum Schluß werden die Potentialsummen für jedes Feld gebildet. Das Ergebnis zeigen die Teilquadrate rechts unten in jedem Feld in Bild 11.23.

Die beiden Pfeile in Bild 11.23 deuten nun auf die Felder mit dem größten Gesamtpotential, nämlich 3. Bevor wir aber einfach dort unseren Zug machen, müssen wir auch noch das Drohungspotential unseres Gegners O berücksichtigen.

Wie zuvor stellen wir die Potentiale für die vier prinzipiellen Stellungen zusammen, diesmal nur aus der Sicht von „O". Dies ist im Bild 11.24 dargestellt. Wenden wir diese Potentiale auf unsere Stellung in Bild 11.18 an, so erhalten wir die endgültige Auswertungsmatrix in Bild 11.25, wo wieder der höchste Wert durch Pfeil markiert ist: 4. Das ist mehr als das, was wir bei Berechnung des X-Potentials als Höchstwert erhalten hatten. Unser Algorithmus wird also den Zug 1 als den besten ermitteln; in Bild 11.26 ist er ausgeführt.

Um zu sehen, ob dieser Zug tatsächlich gut war, setzen wir das Spiel mit der Maßgabe fort, daß nur noch die besten Züge gemacht werden. Wie in Bild 11.27 zu verfolgen ist, ergibt sich als Ergebnis ein Unentschieden.

Was wäre nun geschehen, wenn wir den zweiten Schritt, die Drohungsberechnung, unterlassen und gleich den Zug gemacht hätten, der sich nur aus dem

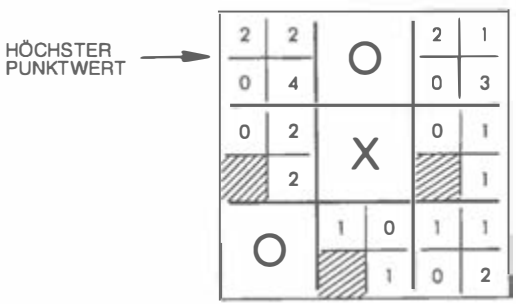



*Abb. 11.25: Potentialauswertung*

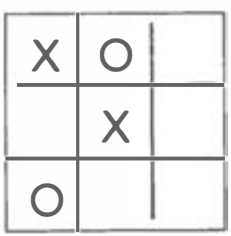



*Abb. 11.26: Zug mit höchstem Punktwert*

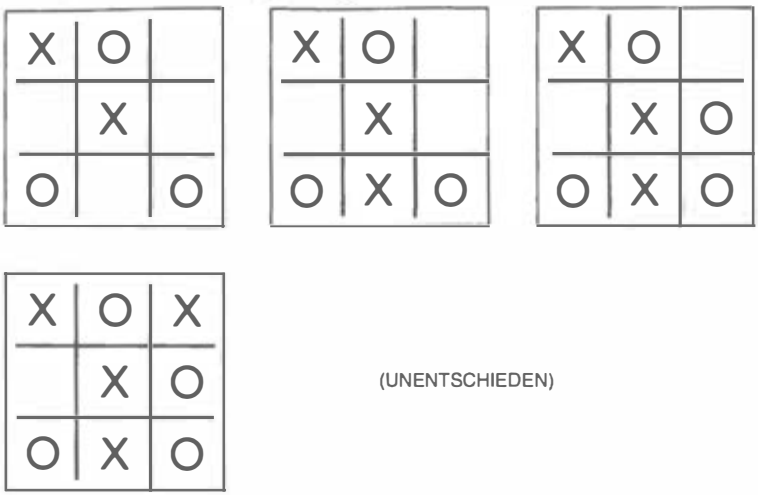



*Abb. 11.27: Spielende*

Potential für X ergab (siehe Bild 11.23)? Der alternative Spielverlauf ist in Bild 11.28 dargestellt. Auch hier ergibt sich nur ein Unentschieden, das heißt, daß bei diesem Beispiel das höhere Potential nicht tatsächlich mit einem größeren strategischen Wert einhergeht. Immerhin hat unser Algorithmus aber funktioniert. Wir wollen ihn nun auch noch unter härteren Bedingungen testen.

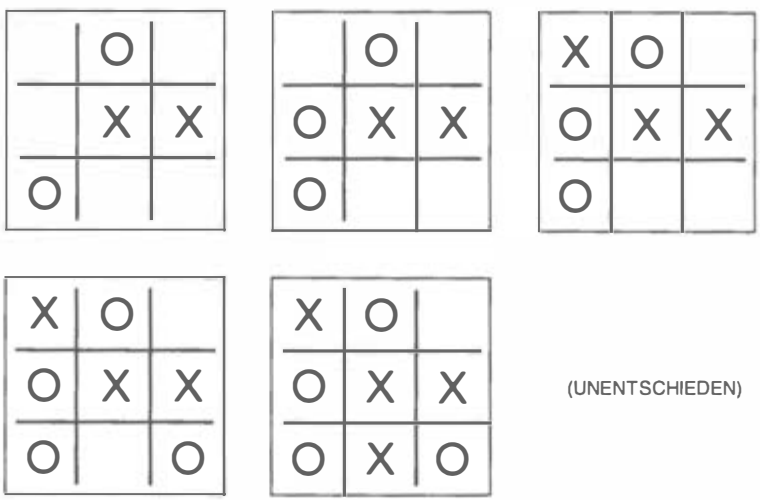



*Abb. 11.28: Alternatives Spielende*

### *Der Algorithmus wird verbessert*

Um unseren Algorithmus zu testen, sollten wir klare Situationen prüfen, in denen es genau einen besten Zug gibt. Nehmen wir an, daß der Spieler am Zuge ist. Die erste Teststellung, für „X" ausgewertet, ist in Bild 11.29 dargestellt. Bild 11.30 zeigt das Potential für „O". Bei dieser Stellung offenbart unser Algorithmus Schwächen: Das höchste Potential für „X" ist 4 im rechten unteren Feld. Würde der Computer aber tatsächlich diesen Zug machen, so würde der Spieler gewinnen (Feld oben Mitte).

Offenbar müssen wir der Situation, in der in irgendeiner Reihe bereits zwei X vorhanden sind, besonders Rechnung tragen und dem letzten Feld einer solchen Reihe ein wesentlich höheres Potential einräumen. Wenn wir z.B. den Wert auf 5 statt auf 3 festsetzen, sollte ein solcher Zug automatisch die höchste Priorität haben. Dies wäre eine erste wichtige Verbesserung unseres Algorithmus.

Die zweite Testsituation ist die in Bild 11.31. Unser Algorithmus gibt dem rechten unteren Feld nun den Wert 6 (siehe Pfeil), und das ist sicher der richtige Zug. Unsere Verbesserung funktioniert also.

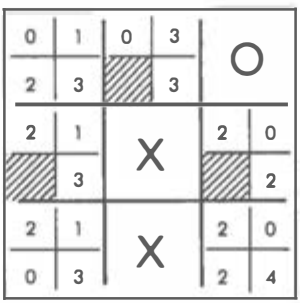



*Abb. 11.29: Testfall 1 für „X" ausgewertet*

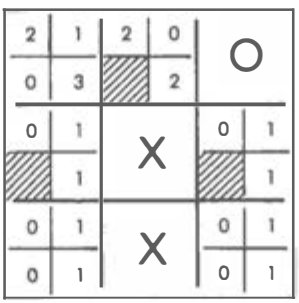



*Abb. 11.30: Testfall 1 für „O" ausgewertet*

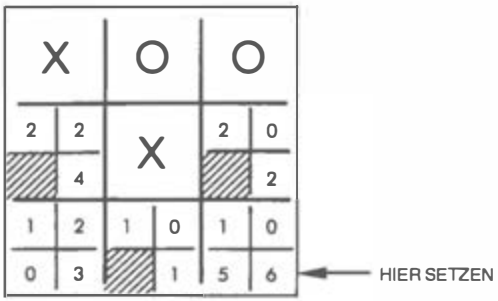



*Abb. 11.31: Testfall 2*

### *Der erste Zug*

Auch bei Spielbeginn, wenn das Brett noch leer ist, muß unser Algorithmus den besten Zug ermitteln. Bild 11.32 zeigt das Ergebnis: Es ist das Zentrumsfeld. Das ist zwar einleuchtend; es läßt sich aber zeigen, daß dieser Zug beim Tic-Tac-Toe nicht unvermeidlich ist. Zudem wäre es etwas langweilig und einfallslos, wenn der Computer tatsächlich immer mit diesem Zug beginnen würde. Auch das sollten wir also ändern.

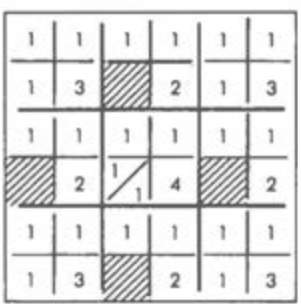



*Abb. 11.32: Zug in die Brettmitte*

### *Ein weiterer Test*

Sehen wir uns eine weitere einfache Stellung an, sie ist in Bild 11.33 dargestellt. Wieder ist der vorgeschlagene Zug offenbar vernünftig. Die umgekehrte Stellung (Bild 11.34) führt tatsächlich auch zum sicheren Sieg. Unser Algorithmus scheint also wirklich zu funktionieren. Stellen wir ihm eine letzte Falle.

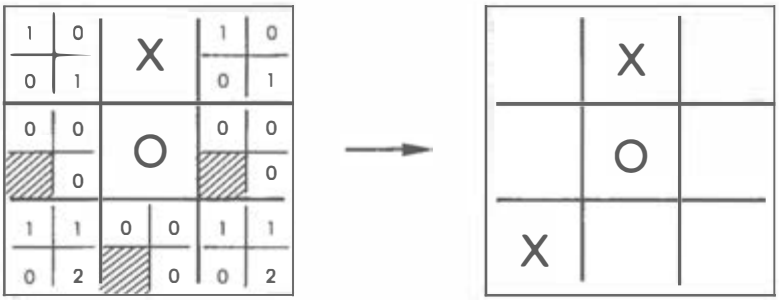


*Abb. 11.33: Eine einfache Stellung*

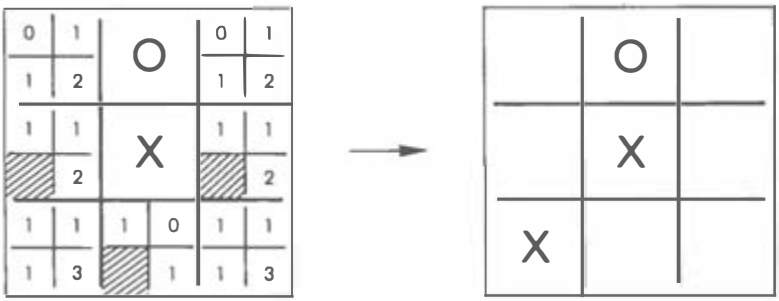


*Abb. 11.34: Die umgekehrte Situation*

### *Eine Falle*

Schauen wir uns die Situation in Bild 11.35 an; „X" ist am Zug. Unser Algorithmus würde uns eins der Felder mit dem Potential 4 empfehlen. Dies wäre jedoch ein Fehler, wie der in Bild 11.36 dargestellte weitere Spielverlauf klar ergibt: „O" würde gewinnen. Die in Bild 11.37 vorgestellte Zugfolge zeigt jedoch, daß zumindest ein Unentschieden zu erreichen ist. Unser Algorithmus hat hier also versagt. Im folgenden unterziehen wir diesem Fall einer einfachen Analyse: Gezogen wurde von „O" auf ein Feld mit dem hohen Drohpotential 4, wobei ein zweites Feld mit gleichem Drohpotential allerdings ungeschützt blieb (vgl. Bild 11.35). Das heißt im Prinzip, daß „O" wahrscheinlich gewinnt, wenn ein Feld mit dem Potential 4 für ihn frei bleibt. Anders ausgedrückt: Immer wenn eine Drohung von „O" einen bestimmten Schwellenwert erreicht, sollte der Algorithmus die Strategie wechseln. In diesem Fall wäre das, ein Feld zu besetzen, das eine Zweierreihe schafft und so eine unmittelbare Verlustdrohung für „O" darstellt, die sofort zu beantworten ist. Der Algorithmus sollte also, kurz gesagt, eine Situation noch einen Zug tiefer analysieren, also einen Zug weiter vorausdenken. Man nennt dies eine zweistufige Analyse.

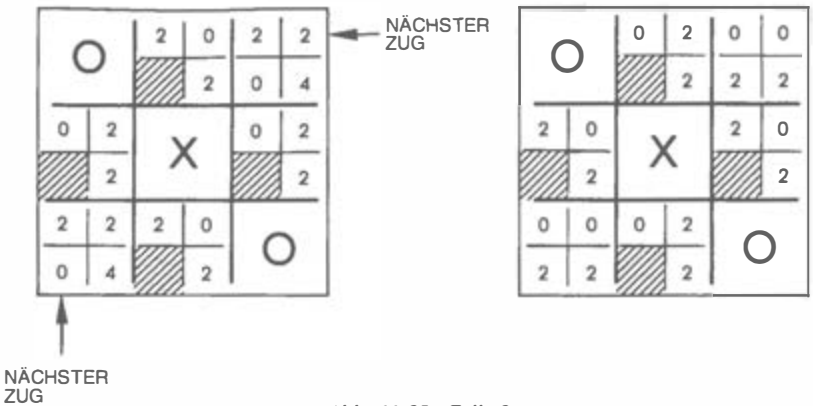



*Abb. 11.35: Falle 3*

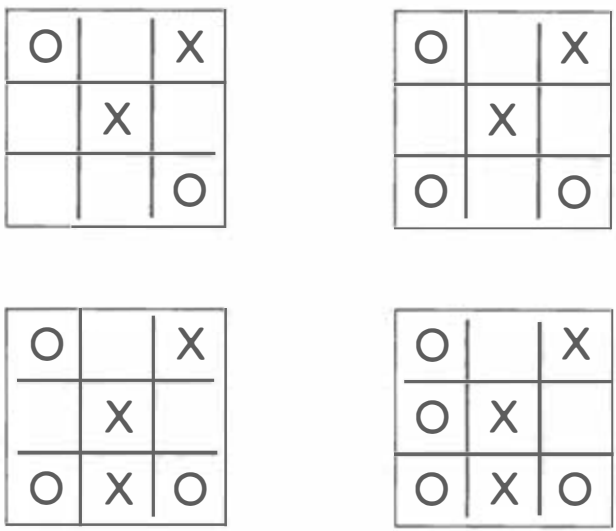


*Abb. 11.36: Spielende*

Wir fassen also zusammen, daß unser Algorithmus einfach und im Prinzip zufriedenstellend arbeitet, daß er jedoch in zumindest einem Fall (der Falle in Bild 11.35) versagt. Wir müssen nun entweder diesen Fall besonders berücksichtigen, oder aber wir müssen eine Stellung einen Zug weiter analysieren und feststellen, was auf jedes X oder O, das gesetzt werden kann, weiter passieren kann. Letzteres ist natürlich die „sauberste" Lösung, und im Idealfall sollten wir

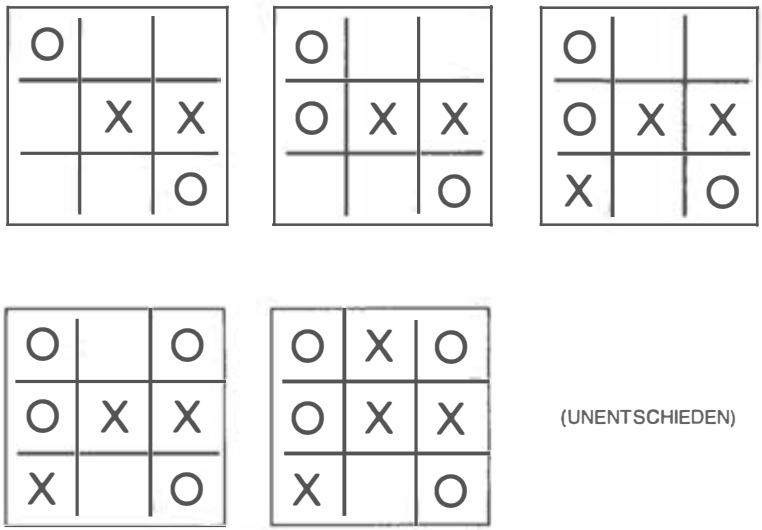



*Abb. 11.37: Ein richtiger Zug*

sogar alle möglichen Spielverläufe bis zur Schlußstellung verfolgen. Allerdings würden die zunehmende Kompliziertheit des Programms, die Speicherplatzanforderungen und die Rechenzeiten dies kaum praktikabel machen. Bei komplexeren Spielen, etwa Schach oder Dame, ist eine solche Mehrstufenanalyse unverzichtbar. Eine nur zweistufige Analyse würde z.B. ein Schachspiel weder gut noch interessant machen, hier müßten drei, vier oder mehr Stufen analysiert werden.

Wenn also die Auswertung nicht genügend in die Tiefe gehen kann, so muß der Algorithmus dazu befähigt sein, bestimmte Spezialfälle zu erkennen. Eine solche „Ad hoc"-Programmierung kann man zwar als etwas „unsauber" ansehen, sie verringert jedoch die Erfordernisse für die Programmlänge und/oder die Speichergröße. Wenn es also die Möglichkeit gibt, bestimmte Spezialfälle im voraus zu erkennen, so kann man ein Programm auch auf diese Fälle vorbereiten. Es wird dann normalerweise kürzer sein als das für den Allgemeinfall präparierte. Voraussetzung für diese Programmierweise ist, daß der Programmierer das Spiel selbst sehr gut beherrscht.

Nun ist beim Tic-Tac-Toe die Zahl der Spielsituationen sehr begrenzt, wodurch es möglich ist, alle theoretisch möglichen Spielverläufe einzeln zu verfolgen und dafür Verhaltensweisen zu entwickeln. Da wir hier im wesentlichen den verfügbaren Speicherplatz zu berücksichtigen haben, werden wir nun einen „Ad hoc"-Algorithmus für unseren 1KByte-Speicher aufstellen. Andere Vorgehensweisen werden in Übungsaufgaben vorgeschlagen.

### Der „Ad hoc"-Algorithmus

Dieser Algorithmus ordnet jedem Feld auf dem Spielbrett einen Wert zu, der davon abhängt, wer dort einen Zug plaziert hat. Zu Beginn des Spiels haben alle Felder den Wert 0. Wenn jedoch der Spieler ein Feld besetzt hat, so erhält es den Wert 1, wenn der Computer es besetzt hat, den Wert 4 (Bild 11.38). Der Wert 4 wurde deshalb gewählt, damit die Summe in irgendeiner Reihe eindeutig einer bestimmten Konstellation zugeordnet werden kann. Besteht eine Reihe z.B. aus einem O (Spielerzug) und zwei Leerfeldern, so ist die „Reihensumme" 1. Hat der Spieler schon zweimal in einer Reihe gesetzt, so ist die Summe 2, bei dreimal entsprechend 3. Da 3 die höchste Summe für eine Reihe ist, in der nur der Spieler gesetzt hat, ist einem Computerzug der Wert 4 zugeordnet worden. Damit entspricht die Reihensumme 5 der Situation, in der sowohl der Spieler als auch der Computer je einmal in dieser Reihe gesetzt haben und das dritte Feld leer ist. Die sechs möglichen Stellungsmuster sind in Bild 11.38 dargestellt. Wie sich leicht nachvollziehen läßt, sind die Reihensummen 2 und 8 gleichbedeutend mit Gewinnstellungen, während die Reihensumme 5 eine Blockierstellung

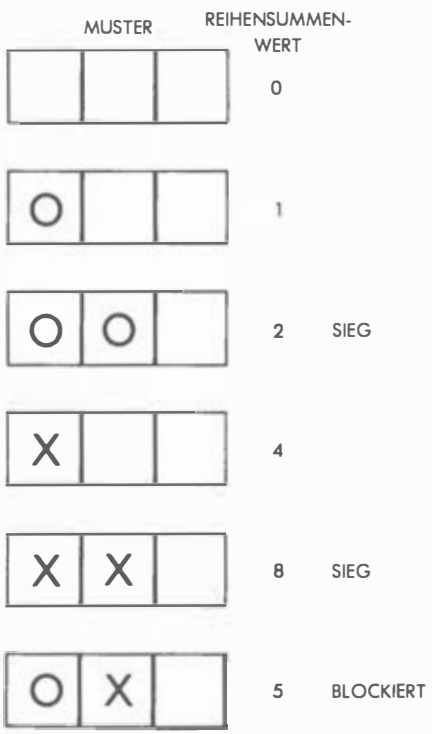



*Abb. 11.38: Reihensummen*

repräsentiert, die für den Spieler wertlos ist. Ist eine Gewinnstellung nicht möglich, so stellen die Reihensummen 1 und 4 die höchsten Potentiale dar, wobei es darauf ankommt, wer am Zug ist.

Auf diesen Beobachtungen basiert der Algorithmus. Zunächst wird er eine Reihensumme von 8 suchen und entsprechend spielen, wenn er eine solche findet. Ist das nicht der Fall, sucht er nach einer sogenannten „Fallenstellung", bei der jede von zwei sich schneidenden Reihen einen Computerzug enthält und sonst nichts. (Der Algorithmus wird immer so angewendet, daß er zu einem Vorteil für den Computer führt.) In Bild 11.39 wird ersichtlich, daß jedes unbesetzte Feld, das zwei Reihen mit der Summe 4 angehört, eine Falle charakterisiert und daß der Algorithmus dort spielen muß, was er auch tut.

Bild 11.40 zeigt das vollständige Flußdiagramm für die Brettanalyse. Sehen wir es uns nun genauer an, und denken wir daran, daß der Computer am Zug ist, wenn dieser Algorithmus zur Anwendung kommt.

Zunächst wird geprüft, ob im nächsten Zug sofort eine Gewinnstellung erreichbar ist. Praktisch geschieht dies, indem alle Reihensummen auf den Wert 8 getestet werden: In diesem Fall sind ja zwei Computerzüge in der jeweiligen Reihe bereits zu verzeichnen (Bild 11.38).

Als nächstes untersuchen wir, ob der Spieler im nächsten Zug gewinnen kann, wenn ja, so muß dieser Zug blockiert werden. Erkannt wird diese Situation an einer Reihensumme von 2, wie Bild 11.38 ausweist.

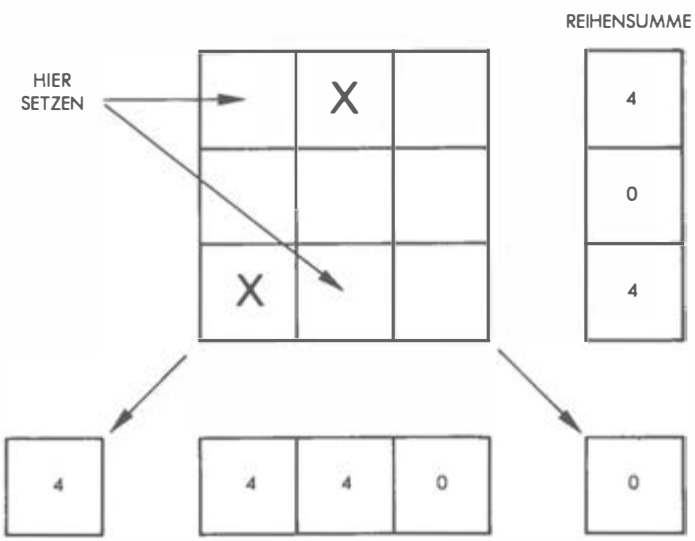



*Abb. 11.39: Eine Fallenstellung*


*Abb. 11.40: Brettanalyse, Flußdiagramm*

Im nächsten Schritt sucht der Algorithmus nach einer Fallenstellung, wie sie in Bild 11.39 vorgeführt wird, und setzt entsprechend, wenn er eine findet.

Kommen wir nun auf eine weitere Algorithmuseigenschaft zu sprechen, die der Spielbelebung dient. Es ist der veränderliche „Intelligenzquotient" des Programms. Wenn der Computer bis zur oben geschilderten dritten „Überlegungs-

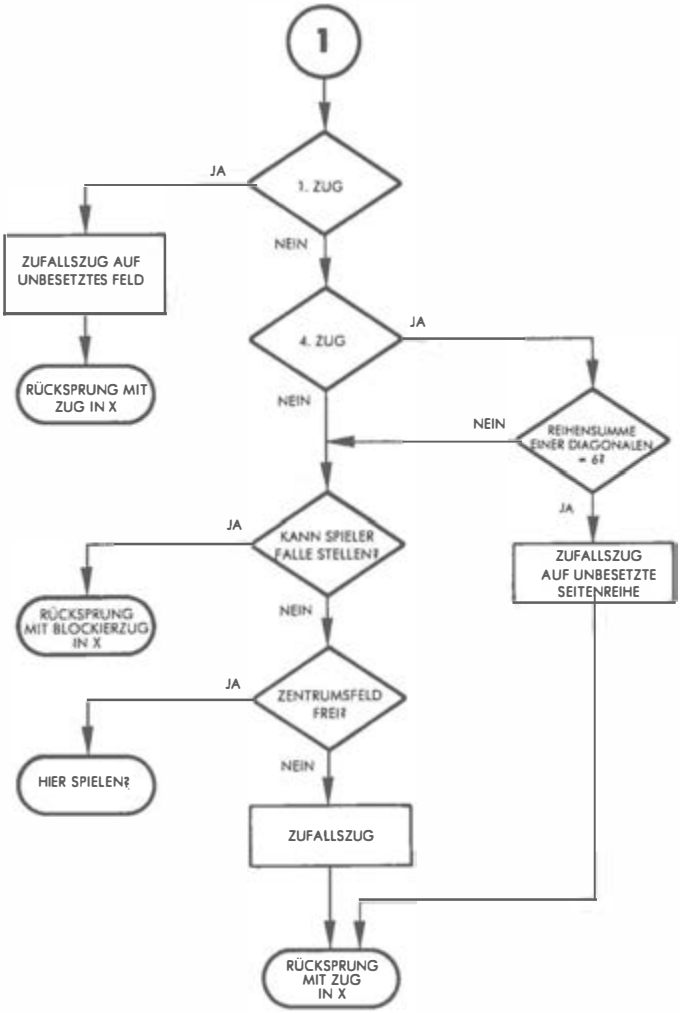



*Abb. 11.40: Brettanalyse, Flußdiagramm (Fortsetzung)*

stufe" gekommen ist, hat er die wesentlichen „vernünftigen" Züge überprüft, und er kann nun auch einmal einen unvorhersehbaren Zufallszug machen, wenn sein Intelligenzniveau gerade niedrig steht. Um dem Spiel auf diese Weise etwas Farbe zu geben, erzeugen wir eine Zufallszahl und vergleichen sie mit dem derzeitigen Intelligenzgrad. Je nach dem Ergebnis wird die Spielweise dann variieren. Ist der IQ auf seinem Höchstwert, verzweigt das Programm immer in

den rechten Teil des Flußdiagramms. Ist er aber niedriger, so geht die Verzweigung auch manchmal nach links. Verfolgen wir zunächst die Verzweigung nach rechts: Wir überprüfen dort die zwei speziellen Situationen, die dem ersten und dem vierten Zug in einem Spiel entsprechen.

In der ersten Situation, beim Eröffnungszug also, besetzt der Algorithmus irgendein beliebiges Feld. Diese unvorhersehbare Verhaltensweise wird also in gewisser Weise „intelligent" erscheinen.

Bei der zweiten Situation betrachten wir den vierten Zug, wobei der Computer „dran" ist, was bedeutet, daß der Spieler den Eröffnungszug gemacht hat, der Computer den zweiten, der Spieler den dritten. Der Spieler hat bisher also zweimal gezogen, und der Computer hat einmal gezogen und ist jetzt an der Reihe. Nun überprüfen wir, ob die ersten drei Züge alle auf einer der beiden Diagonalen liegen. Wenn das so ist, muß die Reihensumme einer Diagonalen 6 sein, denn der Spieler hat zwei Züge gemacht und der Computer einen. Der Algorithmus muß also ausdrücklich auf diesen Wert hin prüfen. Wenn die ersten drei Züge auf einer Diagonalen liegen, dann muß der Computer auf ein Seitenfeld setzen, und diese Verhaltensweise muß im Algorithmus verankert sein, damit der Computer auf höchstem IQ-Stand mit Sicherheit gewinnt. Bild 11.41 illustriert diese Situation. Zu beachten ist, daß der Computer bei streng logischem Verhalten eine der freien Ecken besetzen müßte, da der Spieler

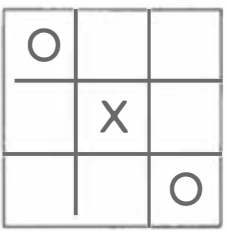



*Abb. 11.41: Die Diagonalfalle*

selbst eine Ecke zu besetzen und eine Falle aufzubauen droht. Was dabei herauskäme, zeigt Bild 11.42: Der Computer würde verlieren. Was aber passiert, wenn wir auf einem Seitenfeld unseren Zug machen, zeigt Bild 11.43: Ein Unentschieden ist das Ergebnis. Dieser Zug sollte also gemacht werden. Diese beim Tic-Tac-Toe relativ wenig bekannte Falle muß für den Algorithmus also erkennbar gemacht werden, wenn der Computer gewinnen soll.

Wenn wir uns weder im vierten Spielzug befinden, noch die Diagonalfalle gestellt wurde, sollte der Computer prüfen, ob der Spieler eine andere Falle aufbauen kann (siehe Flußdiagramm Bild 11.40). Kann er das, so blockiert der Computer einen solchen Zug. Andernfalls besetzt er das Zentrumsfeld, wenn es frei ist. Ist es nicht frei, so macht er einen Zufallszug.

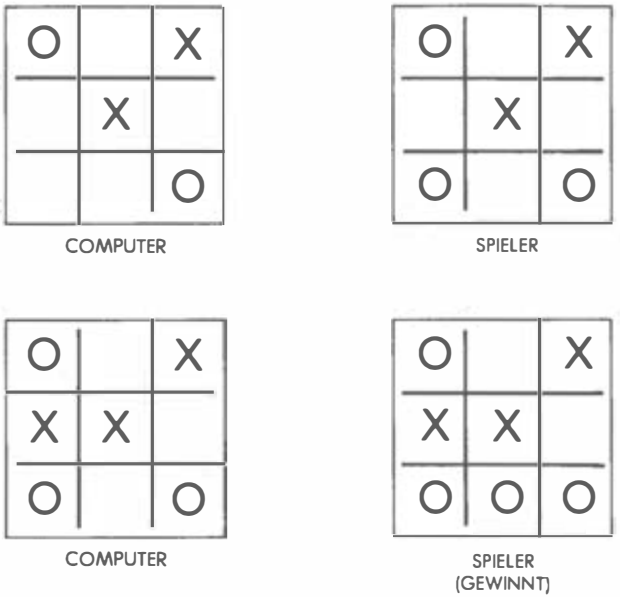



*Abb. 11.42: In die Diagonalfalle gegangen*

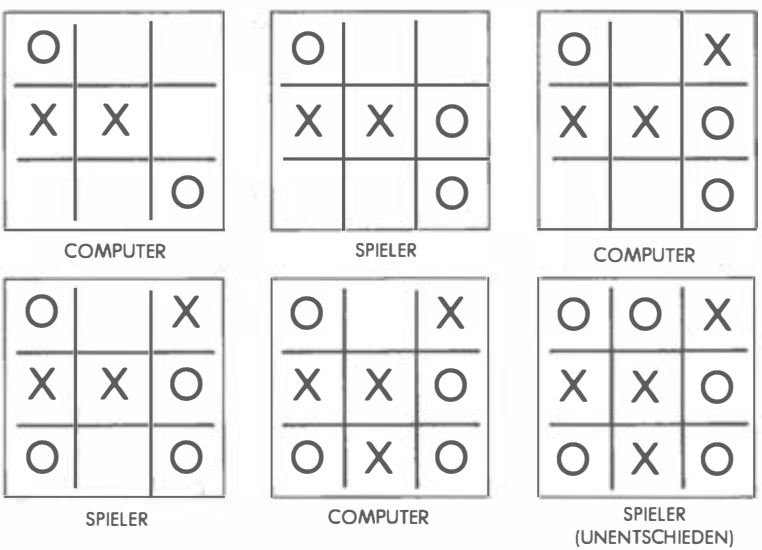



*Abb. 11.43: Spielen am Rand*

| COMPUTER | SPIELER |
|---|---|
| 4 | 5 |
| 7 | 1 |
| 9 | 8 |
| 2 | UNENT-SCHIEDEN |
| 8 | 5 |
| 6 | 3 |
| 7 | 9 |
| 1 | 4 |
| UNENT-SCHIEDEN | |
| 2 | 5 |
| 9 | 1 |
| 7 | 8 |
| 6 | 3 |
| UNENT-SCHIEDEN | |
| 8 | 5 |
| 1 | 7 |
| 3 | 2 |
| 6 | 9 |
| UNENT-SCHIEDEN | |
| 6 | 5 |
| 4 | 8 |
| 2 | 3 |
| 7 | 1 |
| UNENT-SCHIEDEN | |

| COMPUTER | SPIELER |
|---|---|
| | 5 |
| 1 | 6 |
| 4 | 7 |
| 3 | 2 |
| 8 | 9 |
| UNENT-SCHIEDEN | |
| | 5 |
| 3 | 4 |
| 6 | 9 |
| 1 | 2 |
| 8 | 7 |
| UNENT-SCHIEDEN | |
| | 2 |
| 5 | 1 |
| 3 | 7 |
| 4 | 6 |
| 9 | 8 |
| UNENT-SCHIEDEN | |
| | 1 |
| 5 | 3 |
| 2 | 8 |
| 9 | 6 |
| 7 | 4 |
| UNENT-SCHIEDEN | |

| COMPUTER | SPIELER |
|---|---|
| | 6 |
| 5 | 4 |
| 1 | 9 |
| 3 | 7 |
| 2 | NIEDERLAGE |
| | 6 |
| 5 | 4 |
| 8 | 2 |
| 9 | 1 |
| 7 | NIEDERLAGE |
| | 6 |
| 1 | 5 |
| 4 | 7 |
| 3 | 2 |
| 8 | 9 |
| UNENT-SCHIEDEN | |
| 9 | 5 |
| 3 | 6 |
| 4 | 2 |
| 8 | 7 |
| UNENT-SCHIEDEN | |

*Abb. 11.44: Tatsächliche Spielverläufe*

Da dieser Algorithmus nach einem „Ad hoc"-Prinzip aufgebaut ist, ist es schwer zu beweisen, daß er in jedem Fall siegen oder ein Unentschieden erreichen kann. Sie sollten dies auf einem Spielfeld oder mit dem Programm selbst herauszubekommen versuchen. Jedenfalls hat das Programm in allen Testversuchen unentschieden gespielt oder gewonnen. Wenn der Computer jedoch fortlaufend gewinnt, so sinkt sein Intelligenzquozient, wodurch auch der Spieler irgendwann eine Gewinnmöglichkeit erhält. Bild 11.44 zeigt einige Beispiele von tatsächlich auf dem Spielbrett vorgekommenen Zugfolgen.

### *Vorschläge für Modifikationen*

**Übung 11-1:** *Teilen Sie einer bestimmten Taste die Funktion zu, den aktuellen IQ-Wert des Computers auszugeben.*

**Übung 11-2:** *Ändern Sie das Programm so, daß der IQ-Wert zu Beginn jedes Spiels geändert werden kann.*

Wir glauben, daß der beschriebene „Ad hoc"-Algorithmus bisher noch unveröffentlicht ist. Die wesentlichen Ideen stammen von Eric Novikoff. Einige eigene Ideen stammen aber aus „Scientific American" (ausgewählte Ausgaben von 1950-1978) und Dr. Harvard Holmes.

### Andere Strategien

Zu erwägen sind viele andere Strategien. Ein besonders kurzes Programm kann durch die Verwendung von Tabellen entstehen, in denen die Züge zu bestimmten Spielsituationen in Beziehung gesetzt werden. Wenn dabei sämtliche Symmetrien des Spielbrettes berücksichtigt werden, können die Tabellen recht klein werden, denn die Zahl der Möglichkeiten sinkt dabei erheblich. Ein solches Programm würde wohl kürzer werden, es wäre als Entwurf aber auch nicht so interessant.

**Übung 11-3:** *Entwickeln Sie ein solches Tabellen-Programm für Tic-Tac-Toe.*

## DAS PROGRAMM

Die Grobstruktur des Programms ist einfach angelegt und in Bild 11.42 dargestellt. Der komplexeste Teil ist der Algorithmus zu Ermittlung des nächsten Computerzuges. Dieser Algorithmus heißt FINDZUG, er wurde oben ausführlich beschrieben.

Sehen wir uns nun die allgemeine Programmorganisation an, das Flußdiagramm dazu erscheint in Bild 11.45.

1. Der IQ des Computers wird auf 75 Prozent gesetzt.
2. Der Tastendruck des Spielers wird eingelesen.

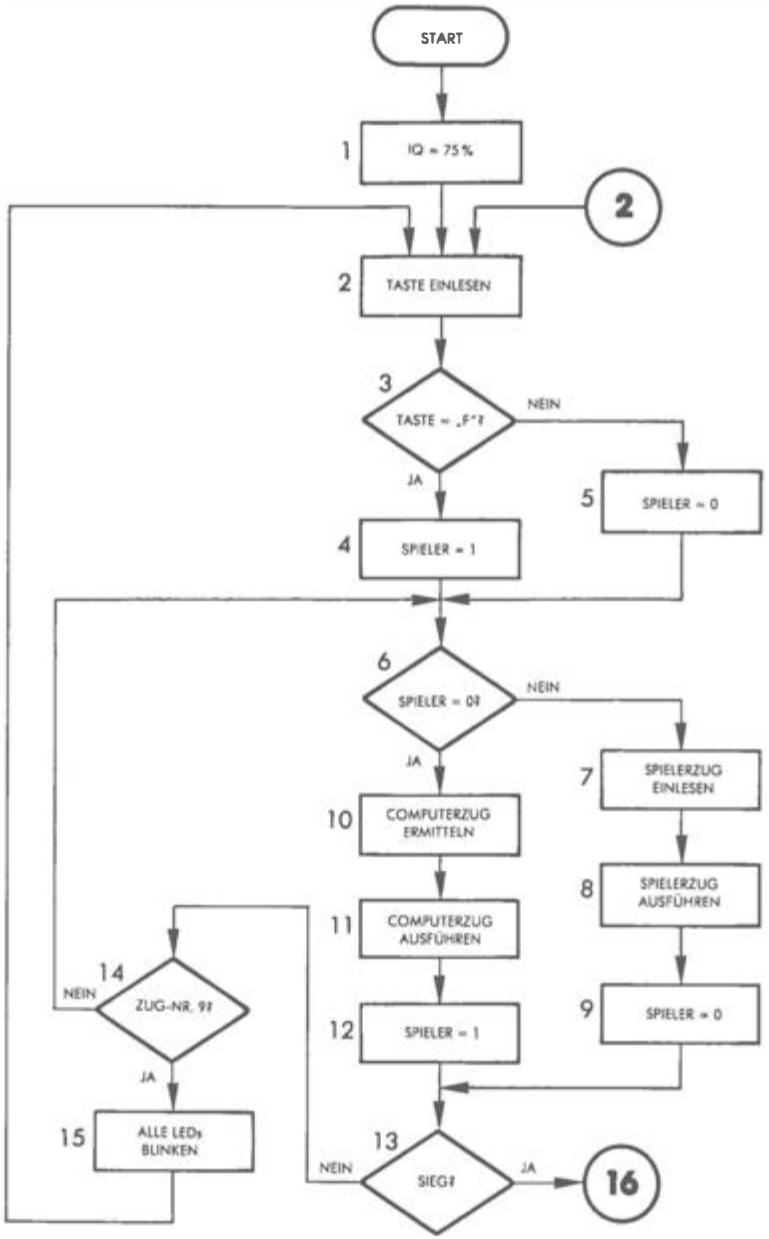



*Abb. 11.45: Flußdiagramm TIC-TAC-TOE*

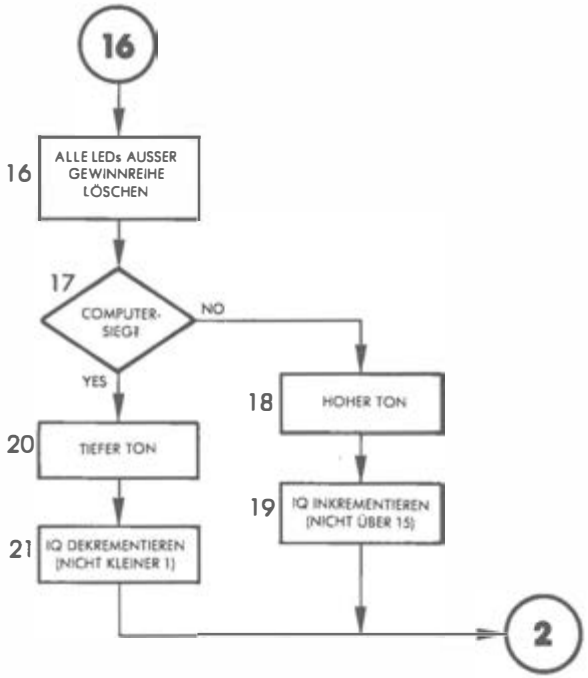



*Abb. 11.45: Flußdiagramm TIC-TAC-TOE (Fortsetzung)*

3. Die Taste wird auf „F" geprüft. Ist F gedrückt worden, so beginnt der Spieler, andernfalls beginnt der Computer. Je nach der gedrückten Taste geht es zu Kasten 4 oder 5 im Flußdiagramm, danach zu Kasten 6.

Wenn der Spieler beginnt (Variable SPIELER ungleich 0), geht es in den rechten Teil des Flußdiagramms.

7. Die vom Spieler gedrückte Taste (der Spielerzug also) wird eingelesen und auf dem Spielbrett ausgegeben.
8. Die entsprechende LED geht an, und der Computer ist am Zug. In Kasten 9 wird SPIELER auf 0 gesetzt.

Wenn der Computer am Zug ist, verlassen wir Kasten 6 in Richtung Kasten 10.

10. Der nächste Computerzug wird ermittelt. Dies tut unser bekannter Algorithmus.
11. Der Zug des Computers wird ausgegeben.
12. SPIELER wird zurück auf 1 gesetzt, um festzuhalten, daß der Spieler jetzt am Zug ist.

Nachdem einer der beiden Kontrahenten gezogen hat, wird in Kasten 13

überprüft, ob eine Gewinnstellung erleuchtet ist. Ist das nicht der Fall, geht es im Flußdiagramm nach links hinüber.

14. Um festzustellen, ob bereits alle Züge gemacht sind, wird die Zugzahl auf den Wert 9 getestet. Leuchten alle neun LEDs, ohne daß eine Gewinnstellung vorliegt, so haben wir ein Unentschieden, und alle LEDs müssen blinken.

15. Alle LEDs werden zum Blinken gebracht. Danach geht es zurück zu Kasten 6, und der nächste Spieler macht seinen Zug.

Liegt beim Verlassen von Kasten 13 eine Gewinnstellung vor, muß dies angezeigt werden:

16. Alle LEDs, ausgenommen die siegreiche Dreiereihe, werden gelöscht, und der Algorithmus muß feststellen, ob der Spieler gewonnen hat oder der Computer.
17. Der Gewinner wird ermittelt. Hat der Spieler gewonnen, geht es nach rechts im Flußdiagramm.
18. Ein hoher Ton erklingt.
19. Der Computer-IQ wird erhöht, aber nicht über das Maximum 15.

Ein Computersieg wird analog in den Kästen 20 und 21 abgehandelt.

Der allgemeine Programmablauf ist geradeheraus. Betrachten wir nun den gesamten Informationsfluß. Das Unterprogramm zur Auswertung einer Spielstellung heißt ANALYSE. Es greift seinerseits auf das Unterprogramm STELLNG zurück, das die Werte der verschiedenen Positionen berechnet.

**Datenstrukturen**

Das wesentliche Datenfeld, das von diesem Programm benutzt wird, ist eine lineare Tabelle mit drei Eingangsstellen, wo die acht möglichen Dreierreihen des Spielfeldes fixiert sind. Bei der Brettauswertung muß das Programm jede dieser möglichen Dreierreihen auswerten. Um diesen Prozeß zu vereinfachen, sind alle acht Reihenkonfigurationen explizit gelistet. Die Speicheraufteilung ist in Bild 11.46 dargestellt.

Die Tabelle ist in drei Sektionen geliedert, die bei RHZG1, RHZG2 und RHZG3 beginnen (RHZG = Reihenzeiger). Wenn die Auswertungsroutine z.B. jeweils die ersten Elemente RHZG1, RHZG2 und RHZG3 anspricht, so sind die zugehörigen Felder 0, 3 und 6 (Pfeile in Bild 11.46). Die nächste Dreierreihe ergibt sich aus den Zweiteintragungen der RHZG-Tabelle: 1, 4 und 7 (das ist die mittlere Senkrechte unserer LED-Matrix).

Die Dreiteilung der Tabelle soll, wie gesagt, den Zugriff erleichtern. Um die Elemente immer richtig ansteuern zu können, muß ein laufender Zeiger verfügbar sein, der beim Zugriff als Index dienen kann. Wenn wir z.B. unsere Dreieranordnungen von 0 bis 7 durchnumerieren, so erhalten wir „Reihe" 3 aus den Adressen RHZG1+3, RHZG2+3 und RHZG3+3. (Wie Bild 11.46 zeigt, sind das die Felder 0, 1 und 2, also die obere Waagerechte.)

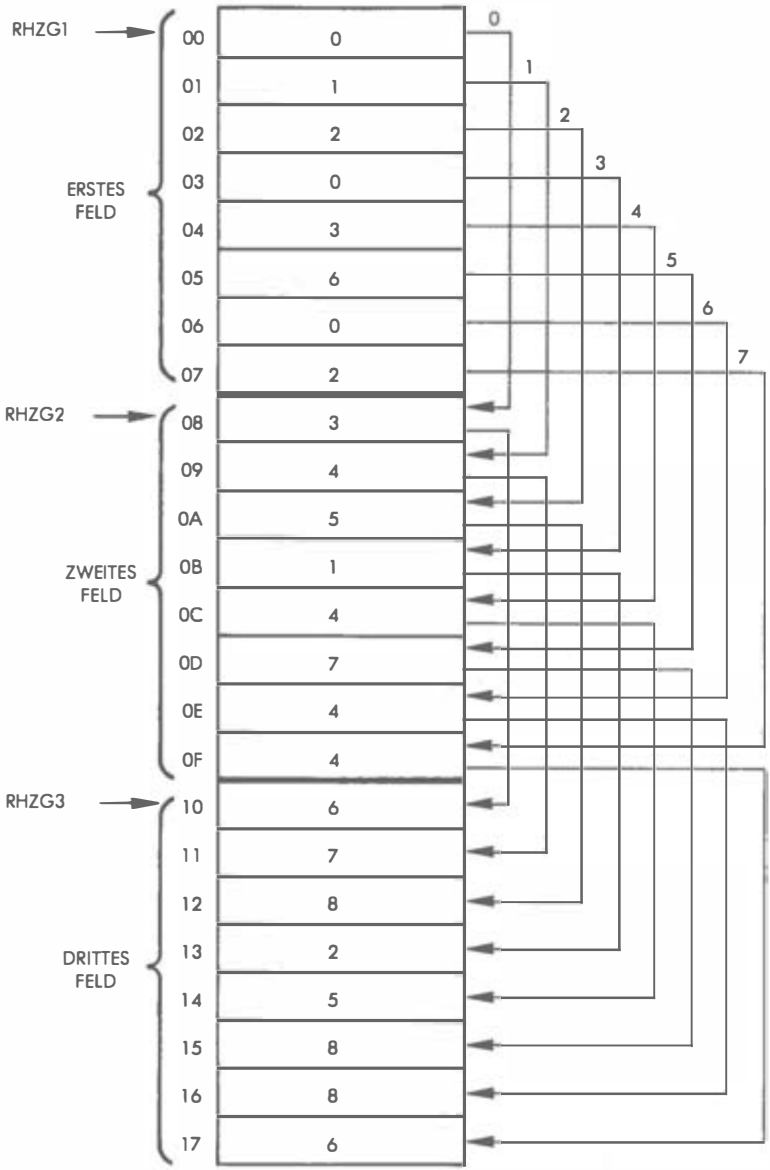



*Abb. 11.46: Speicherung der Dreierreihen*

### Speicherorganisation

Die eben beschriebene RHZG-Tabelle befindet sich auf der 0-Seite (zeropage), ebenso einige andere Tabellen und Variablen (Bild 11.47).

Die BRETT-Tabelle speichert in neun Adressen die aktuelle Spielstellung auf dem Brett. Ein vom Spieler besetztes Feld erhält den Wert 1, ein vom Computer besetztes den Wert 4.

Ebenfalls aus neun Eintragungen besteht die FDSTAT-Tabelle, mit deren Hilfe der taktische Status des Brettes berechnet wird.

Die Werte der Reihensummen der acht möglichen Dreierkombinationen sind in den acht Adressen der RHNSUM-Tabelle enthalten.

Die sechs Positionen der RNDSCR-Tabelle sind der Arbeitsspeicher für den Zufallszahlengenerator.

Die übrigen Adressen werden für Zwischenspeichervariable, Masken und Konstanten benutzt (siehe Bild 11.47), deren Funktionen erklärt werden, während wir die einzelnen Programmabschnitte besprechen.

### Der obere Speicherbereich

Der obere Speicherbereich ist hauptsächlich für die Ein/Ausgabe reserviert, benutzt werden die Tore 1 und 2 und die Unterbrechungen. Die Speicheraufteilung im einzelnen ist in Bild 11.48 dargestellt. Der Unterbrechungsvektor ist in Adressen A67E und A67F untergebracht. Er wird zu Programmanfang so modifiziert, daß der Intervallzeitgeber automatische Interrupts erzeugt, um Blinkeffekte der LEDs zu ermöglichen.

### Das Programm im Detail

Bei Spielbeginn wird der IQ-Wert des Computers auf 75 Prozent gesetzt. Danach erhöht sich dieser Wert um 1, wenn der Spieler gewinnt, und er erniedrigt sich um 1, wenn der Computer gewinnt. Der dezimale Anfangswert des IQ ist 12:

```
START   LDA #12
        STA INTEL       IQ-Wert = 75 %
```

Jetzt wird initialisiert:

```
RESTRT  JSR INIT
```

Das gerade aufgerufene Unterprogramm INIT beginnt bei Adresse 0050 (siehe Listing), sehen wir es uns genauer an. Die erste Aktion dieses Unterprogramms ist das Löschen aller Variablenadressen im unteren Speicherbereich, von CLRST (0018) bis CLREND (0040). Beachten Sie, wie von einer selten

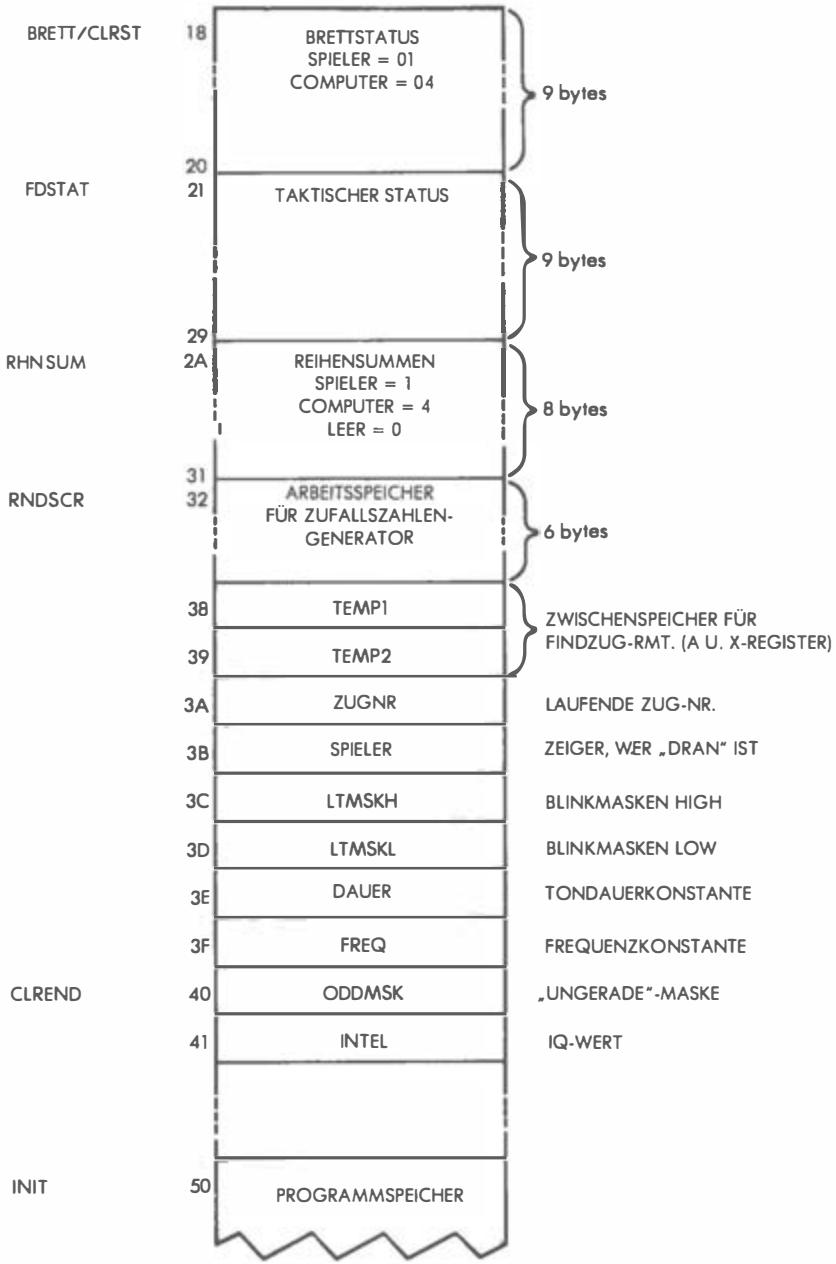



*Abb. 11.47: Unterer Speicherbereich*

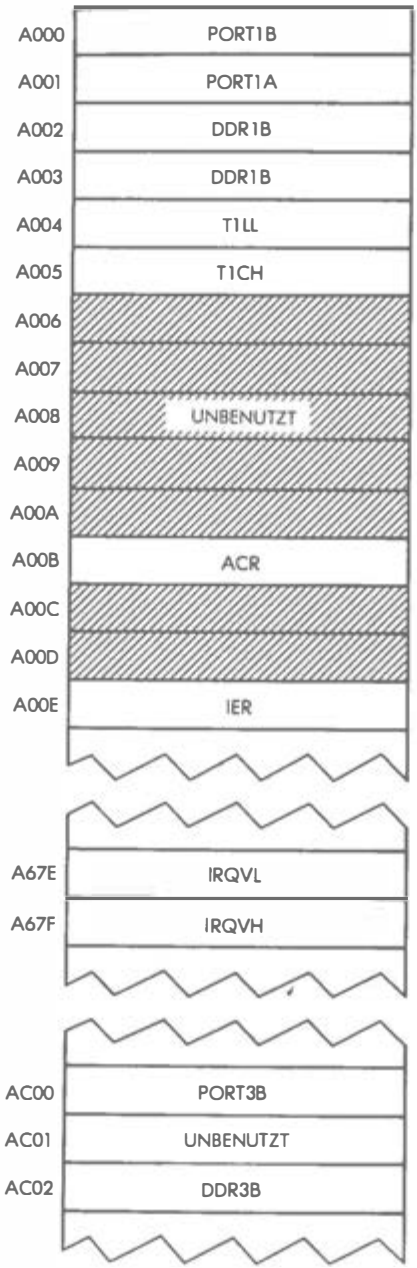



*Abb. 11.48: Oberer Speicherbereich*

benutzten Eigenschaft des Assemblers Gebrauch gemacht wird: Mehrfachmarken für ein- und dieselbe Zeile, um die richtige Anzahl von Speichern zu löschen. Da es während der Programmentwicklung erforderlich werden kann, weitere Variable einzuführen, wurde für die erste zu löschende Adresse die Sondermarkierung CLRST definiert (Adresse 18) und für die letzte Adresse analog CLREND. Adresse 18 korrespondiert z.B. zu CLRST und BRETT. Da der Löschvorgang sich von CLRST an über 40 Adressen bis (CLREND-CLRST) erstrecken soll, laden wir zuerst die Zahl der zu löschenden Adressen ins Indexregister X, und dann löschen wir die entsprechenden Speicher in einer Schleife:

```
INIT    LDA #0

CLRALL  LDX # (CLREND-CLRST)
        STA CLRST,X
        DEX
        BPL CLRALL      Speicher löschen
```

Nach dem Löschen des unteren Speicherbereiches erhalten die zwei „Samenzellen" des Zufallsgenerators ihre Ausgangswerte. Das erledigt wie üblich der untere Zähler des Zeitgebers 1:

```
LDA T1LL
STA RNDSCR+1
STA RNDSCR+4
```

Ausgangstore werden die Tore 1A, 1B und 3A, und die Datenrichtungsregister werden entsprechend geladen:

```
LDA #$FF
STA DDR1A
STA DDR1B
STA DDR3B
```

Alle LEDs auf dem Spielbrett werden gelöscht:

```
LDA #0
STA TOR1A
STA TOR1B
```

Als nächstes wird die Adresse des Unterbrechungsvektors mit einem neuen Zeiger ausgestattet, der auf die Startadresse unserer INTERRUPT-Routine deuten soll, um das Blinken der LEDs zu steuern. Einzelheiten zu diesem Programmteil finden sich in früheren Kapiteln. Die beiden Bytes der Startadresse INTVEC werden in IRQVL und IRQVH geladen, wobei ein spezielles

Assemblersymbol das high Byte (#>INTVEC) und das low Byte (#<INTVEC) identifiziert:

```
JSR ACCESS
LDA #<INTVEC
STA IRQVL        unterer Vektor
LDA #>INTVEC
STA IRQVH        oberer Vektor
```

Der Lösch- und Ladevorgang des Unterbrechungszulassungsregisters ist auch aus früheren Kapiteln bekannt:

```
LDA #$7F
STA IER          Register löschen
LDA #$C0
STA IER          Interrupt zulassen
```

Zeitgeber 1 erhält den Freilaufmodus

```
LDA #$40
STA ACR
```

und das Hilfsregister des Zeitgebers den Maximalwert FFFF:

```
LDA #$FF
STA T1LL
STA T1CH
```

Zum Schluß erfolgt die Unterbrechungszulassung, und die Dezimalmodusflagge wird vorsichtshalber gelöscht:

```
CLI
CLD
RTS
```

**Zurück zum Hauptprogramm**

Wir befinden uns jetzt in Zeile 69 des Programmlistings und lesen als nächstes einen Tastendruck ein:

```
JSR GETKEY
```

Die Eingabe, wer den ersten Zug haben soll, wird entschlüsselt: „F“ heißt, der Spieler beginnt; andernfalls beginnt der Computer:

```
CMP #$F
BNE SPISCHL
```

Der Spieler ist am Zug, und wir halten das mit dem Wert 1 in der Variablen SPIELR (siehe Bild 11.47) fest:

```
LDA #01
STA SPIELR
```

Ein neuer Zug steht bevor, und der entsprechende Zähler wird um 1 erhöht. Er ist in ZUGNR (wieder Bild 11.47) gespeichert:

```
SPISCHL   INC ZUGNR
```

Variable SPIELR zeigt also an, wer am Zug ist: bei 0 der Computer, bei 1 der Spieler. Das muß nun abgefragt werden:

```
LDA SPIELR
BEQ COMZUG
```

Nehmen wir zunächst an, daß der Spieler am Zug ist. Dann wird SPIELR wieder auf 0 gesetzt, damit als nächstes der Computer einen Zug macht:

```
DEC SPIELR
```

Den Spielerzug holt das Unterprogramm SPLZUG ab (Beschreibung weiter unten):

```
JSR SPLZUG
```

Bei der Rückkehr von der SPLZUG-Routine steht der Zug des Spielers im X-Register. Die Tatsache, daß dieser Zug ein Spielerzug ist, wird in der BRETT-Tabelle durch den Wert 1 repräsentiert, und dieser Wert muß im Akkumulator stehen:

```
LDA #01
```

Der Zug erscheint auf dem Brett in Form von Blinkern der entsprechenden LED. Die weiter unten beschriebene STELLNG-Routine leistet nicht nur das, sie bringt auch die Reihensummen auf den neuesten Stand:

```
JSR STELLNG
```

Nach Ausführung des Zuges muß geprüft werden, ob eine Gewinnstellung entstanden ist. Ist das der Fall, so hat der Spieler eine blinkende LED-Dreierreihe, und die entsprechende Reihensumme ist 3. Testen wir also alle acht Dreierkonstellationen auf den RHNSUM-Wert 3:

```
LDA #3
BNE SIEGTST
```

Bei SIEGTST fungiert Y als Index, um die acht Reihensummen von oben nach unten auf 3 abzufragen:

```
SIEGTST   LDY #7
TSTLP     CMP RHNSUM,Y
          BEQ SIEG
          DEY
          BPL TSTLP
```

Fahren wir mit dem Spielerzug fort, und sehen wir uns den Computerzug später im Zusammenhang an (Zeilen 83-88 im Programmlisting, die wir eben übersprungen haben). Das Spiel kann maximal neun Züge haben, prüfen wir nun ZUGNR, um zu sehen, ob das Spiel zu Ende ist:

```
          LDA ZUGNR
          CMP #9
          BNE SPISCHL
```

Damit ist die Hauptschleife abgeschlossen, und es geht zurück nach SPISCHL, wo das Hauptprogramm wieder aufgegriffen wird.

Hätten wir hier das Spielende erreicht, wäre der Ausgang unentschieden, denn ein Sieger wurde nicht ermittelt. Alle LEDs müßten blinken, und ein neues Spiel würde beginnen. Blinken wir also:

```
          LDA #$FF
          STA LTMSKL
          STA LTMSKH
          BNE DLY
```

Die Verzögerung bei DLY sorgt dafür, daß das Blinken etwas andauert. Betrachten wir nun aber das Ende des Spiels.

Liegt eine Gewinnstellung vor, so ist entweder der Spieler oder der Computer der Sieger. Wir erinnern uns, daß ein computerbesetztes Feld den Wert 4 hat (im Gegensatz zu 1 bei Spielerbesetzung). Wenn wir nun die Reihensumme 12 (3 x 4) vorfinden, so haben wir es demnach mit einem Sieg des Computers zu tun. Der IQ-Wert muß in diesem Fall dekrementiert werden:

```
SIEG      CMP #12
          BEQ INTDN
```

Bei INTDN wird zunächst ein tiefer Ton erzeugt, um dem Spieler seine Niederlage akustisch zu untermalen: FREQ erhält den Wert FF:

```
INTDN     LDA #FF
          STA FREQ
```

Der IQ-Wert darf beim Dekrementieren nicht unter 0 fallen:

```
        LDA INTEL
        BEQ GTMSK
        DEC INTEL
```

Die komplette Dreierreihe der Gewinnstellung wird nun kurz erleuchtet, danach ertönt das Signal für das Ende des Spiels. Löschen wir zunächst alle LEDs:

```
GTMSK   LDA #0
        STA TOR1A
        STA TOR1B
```

Die Kennzahl der siegreichen Dreierreihe ist noch im Y-Register. Holen wir uns also aus der RHZG-Tabelle (Bild 11.47) nacheinander die LED-Codes. Erstes Feld:

```
        LDX RHZG1,Y
        JSR LEDLTR
```

Die weiter unten erläuterte LEDLTR-Routine erleuchtet die LED, deren Nummer im X-Register steht. Zweites Feld:

```
        LDX,RHZG2,Y
        JSR LEDLTR
```

und drittes Feld:

```
        LDX RHZG3,Y
        JSR LEDLTR
```

Blinken sollen nur die drei LEDs der „siegreichen" Reihe. Wir müssen also LTMSKL ummaskieren

```
        LDA TOR1A
        AND LTMSKL
        STA LTMSKL
```

und dann auch LTMSKH:

```
        LDA TOR1B
        AND LTMSKH
        STA LTMSKH
```

**Übung 11-4:** *Unmittelbar nachdem LEDRTR (Zeile 125 im Listing) die dritte LED der Gewinnreihe erleuchtet hat, werden die Inhalte von Tor1A und Tor1B*

*eingelesen. Nun besteht theoretisch die Möglichkeit, daß unmittelbar nach LEDLTR eine Unterbrechung vorkommt, was den Inhalt von Tor 1A ändern könnte. Könnte dies Probleme verursachen? Wenn nicht, warum nicht? Wenn ja, ändern Sie das Programm so, daß es auf jeden Fall korrekt arbeitet.*

Die Tore A und B enthalten jetzt also das richtige Muster zur Erleuchtung der siegreichen Dreierreihe. Hat der Spieler gewonnen, ist in LTMSKL und LDMSKH dasselbe Muster, und die Reihe blinkt. Jetzt können wir auch den Sieg- oder Verlustton intonieren. Die Tondauer wird FF:

```
LDA #$FF
STA DAUER
```

Da die Frequenz bereits eingestellt ist, müssen wir den Ton nur noch spielen,

```
LDA FREQ
JSR TON
```

und nach einer kleinen Pause

```
DLY     JSR DELAY
```

kann eine neues Spiel mit neuem Computer-IQ beginnen:

```
JMP RESTRT
```

***Zurück zu SIEG***

Gehen wir nun zurück zu Programmzeile 103, und verfolgen wir den Fall, daß nicht der Computer sondern der Spieler gewonnen hat. Hier muß zunächst eine andere Frequenzkonstante geladen werden:

```
LDA #30
STA FREQ
```

Dann wird, da der Spieler gewonnen hat, der Computer-IQ erhöht, wenn er nicht schon auf dem Höchstwert 15 ist:

```
LDA INTEL
CMP #$0F
BEQ GTMSK
INC INTEL
```

Abgesehen von der anderen Tonfrequenz und der unterschiedlichen IQ-Veränderung ist dieser Programmablauf genau dem analog, in dem der Computer gewonnen hat.

### *Die Computerzüge*

Um zu sehen was passiert, wenn der Computer einen Zug macht, gehen wir zurück zu Zeile 83 im Listing. Die Variable SPIELR wird inkrementiert, und dann täuscht eine Verzögerung vor, daß der Computer „grübelt“:

```
COMZUG    INC SPIELR
          JSR DELAY
```

Wie der Computer zieht, ermittelt die weiter unten beschriebene ANALYSE-Routine,

```
          JSR ANALYSE
```

und die STELLNG-Routine bringt den Computerzug-Code 4 an die richtige Brettposition:

```
          LDA #4
          JSR STELLNG
```

Jetzt werden die acht Reihen auf Gewinnstellung, also den Wert 12 geprüft, und es geht im Hauptprogramm weiter, wie es oben schon dargestellt worden ist:

```
          LDA #12
SIEGTST   LDY #7
          ...
          ...
```

Vergleicht man die beiden Programmabschnitte für den Spielerzug und den Computerzug, so zeigt sich, daß der wesentliche Unterschied der ist, daß der Spielerzug von der Tastatur geholt wird und der Computerzug aus dem ANALYSE-Unterprogramm. Diese Routine ist nun der Schlüssel zum „intelligenten“ Verhalten des Computers. Sehen wir sie uns jetzt im Detail an.

## Die Unterprogramme

### *Unterprogramm ANALYSE*

Diese Routine beginnt bei Programmzeile 143 im Listing, das entsprechende Flußdiagramm ist im Bild 11.40 dargestellt. Zu Beginn wird die „Ungerade“-Maske ODDMSK auf 0 gesetzt:

```
ANALYSE   LDA #0
          STA ODDMSK
```

Wir prüfen nun, ob der Computer die Möglichkeit hat, mit seinem nächsten Zug eine Gewinnstellung zu erzeugen. Ist das der Fall, so muß natürlich dieser Zug ausgeführt werden, das Spiel wäre dann zu Ende. Eine Gewinnmöglichkeit

besteht dann, wenn es eine Reihensumme von 8 gibt. Den Wert 8 laden wir also in den Akkumulator:

```
LDA #8
```

Eine Siegsituation ergibt sich dann, wenn sich die Felder der Reihen 1, 2, 3 alle zur selben Zeit zum Wert 3 summieren. Setzen wir also unsere „Filtervariable" X für die Zahl der in Frage kommenden Reihen auf 3,

```
LDX #3
```

und rufen wir jetzt unsere FINDZUG-Routine auf:

```
JSR FINDZUG
```

Die FINDZUG-Routine, die weiter unten beschrieben wird, muß in A eine spezifische Reihensumme mitbringen und in X eine gewisse Anzahl notwendiger Übereinstimmungen. Findet sie bei der systematischen Überprüfung aller Reihen und Felder ein passendes Feld, so verbleibt dessen Nummer beim Rücksprung in X, und die Z-Flagge ist auf 0 gesetzt. Prüfen wir das also:

```
BNE FERTIG
```

Ist ein Gewinnzug gefunden worden, so wird die ANALYSE-Routine verlassen. Leider ist das meist nicht der Fall, und es muß weiter analysiert werden.

Als nächstes muß die Frage überprüft werden, ob der Spieler einen Gewinnzug hat, damit dieser blockiert werden kann. Charakteristisch für eine solche Situation ist die Reihensumme 2. Mit diesem Wert im Akkumulator wird der Prozeß nun wiederholt:

```
LDA #2
LDX #3
JSR FINDZUG
BNE FERTIG
```

Gibt es einen Gewinnzug für den Spieler, so muß das entsprechende Feld blockiert werden, es erfolgt der Rücksprung bei FERTIG. Andernfalls geht die Analyse weiter.

Prüfen wir als nächstes, ob der Computer eine Falle aufbauen kann. Dazu muß in einer Reihe bereits ein Computerzug gemacht worden sein. Am liebsten würden wir natürlich da spielen, wo sich zwei Reihen mit schon je einem Computerzug schneiden (siehe oben bei der Algorithmusentwicklung). Charakteristisch für eine solche Situation wäre A = 4 und X = 2. Mit diesen Werten suchen wir also die FINDZUG-Routine auf:

```
LDA #4
LDX #2
JSR FINDZUG
BNE FERTIG
```

Finden wir einen Zug, geht es über FERTIG zrück, andernfalls geht es weiter im Flußdiagramm 11.40.

Jetzt kommt der Moment, in dem der Computer intelligent oder fehlerhaft spielen kann, je nachdem, wie sein IQ-Wert gerade lautet. Wir holen uns dazu eine Zufallszahl und stellen sie dem IQ-Wert gegenüber. Ist die Zufallszahl größer als der IQ-Wert, so geht es nach links im Flußdiagramm 11.40, zu einem puren Zufallszug also. Andernfalls geht es nach rechts zu einem „durchdachten" Zug. Holen wir uns also eine Zufallszahl:

```
JSR RANDOM
```

Um eine Zahl kleiner 16 zu bekommen, wird das höherwertige Byte abgeschnitten,

```
AND #$0F
```

und es folgt der Vergleich mit dem IQ-Wert:

```
CMP INTEL
BEQ OK
BCS RNDZUG
```

Ist die Zufallszahl größer als der augenblickliche IQ-Wert, findet ab RNDZUG ein Zufallszug statt. Sehen wir uns aber zuerst an, wie der Computer einen „durchdachten" Zug macht.

Ab Marke OK prüfen wir zunächst, ob es sich um den ersten oder den vierten Zug handelt.

```
OK      LDX ZUGNR
        CPX #1
```

Ist es der erste Zug, so besetzen wir ein beliebiges Feld:

```
BEQ RNDZUG
```

Die Probe auf den vierten Zug:

```
CPX #4
```

Ist es nicht der vierte Zug, so suchen wir bei FALLE nach einer möglichen Fallenstellung durch den Spieler:

```
BNE FALLE
```

Sind wir aber beim vierten Zug, so testen wir die beiden Diagonalen auf die mögliche Sequenz Spieler-Computer-Spieler. Finden wir diese Sequenz, so spielen wir auf einer Seitenreihe, andernfalls kehren wir zur Hauptlinie dieser Routine zurück und überprüfen die Möglichkeit einer Falle durch den Spieler. Das Reihenmuster Spieler-Computer-Spieler ist an der Reihensumme 6 erkennbar. Mit diesem Wert im Akkumulator testen wir die Diagonalen, die erfreulicherweise gerade die beiden letzten Eintragungen (6, 7) unserer RHZG-Tabellen sind (Bild 11.46):

```
LDX #6
TXA
CMP RHNSUM,X
BEQ ODDRND
```

Ist ein kritischer Diagonalwert gefunden worden, so müssen wir bei ODDRND einen Randzug machen (siehe weiter unten), andernfalls muß auch noch die zweite Diagonale überprüft werden:

```
INX
CMP RHNSUM,X
BEQ ODDRND
```

Besteht auch hier keine Gefahr, so testen wir auf eine Fallenstellung durch den Spieler.

### *Überprüfen auf Fallenstellung*

Wie beim Computer vorher ist eine mögliche Falle daran erkennbar, daß in zwei sich schneidenden Reihen schon je ein entsprechender Zug gemacht wurde (siehe Algorithmusbeschreibung), im jetzigen Fall vom Spieler. Der Kennwert für die entsprechende Reihensumme ist 1 (ein Spielerzug und zwei Leerfelder). Mit A = 1 und X = 2 geht es somit in die FINDZUG-Routine:

```
FALLE   LDA #1
        LDX #2
        JSR FINDZUG
        BNE FERTIG
```

Wird ein fallengeeigneter Punkt gefunden, muß der nächste Zug dort plaziert werden. Andernfalls setzt der Computer auf das Zentrumsfeld oder (wenn dies besetzt ist) per Zufall irgendwo anders hin.

```
LDX BRETT+4
BNE RNDZUG
LDX #5
BNE FERTIG
```

### *Zufallszug auf einem Randfeld*

Die vier orthogonalen Seitenfelder sind extern 2, 4, 6 und 8 numeriert und intern 1, 3, 5 und 7. Alle orthogonalen Randfelder sind intern also ungradzahlig, alle diagonalen Randfelder dagegen intern gradzahlig. Je nachdem, ob die Variable ODDMSK den Wert 0 oder 1 hat, können wir mit ihr und dem ORA-Befehl nun eine Zahl gezielt grad- oder ungradzahlig machen. Hier wollen wir eine ungerade Zahl, ODDMSK muß also 1 sein:

```
ODDRND    LDA #1
          STA ODDMSK
```

Da ODDMSK ursprünglich 0 ist, wird die nachfolgend erzeugte Zufallszahl nur über ODDRND ungerade, beim Einsprung direkt bei RNDZUG bleibt ODDMSK ja 0.

```
RNDZUG    JSR RANDOM
```

Das linke Byte wird wieder „abgestreift“:

```
          AND #$0F
```

Je nach dem Wert in ODDMSK bekommen wir nun also ein gerades oder ungerades Zufallsfeld,

```
          ORA ODDMSK
```

das allerdings „real“, also im Bereich 1 bis 9 sein muß:

```
          CMP #9
          BCS RNDZUG
```

Außerdem können wir natürlich kein Feld besetzen, das schon besetzt ist. Wir laden also die Feldnummer ins X-Register und überprüfen den Feldstatus in der BRETT-Tabelle (siehe Bild 11.47):

```
          TAX
          LDA BRETT,X
```

Bei jeder Eintragung, die nicht 0 ist, ist das Feld besetzt, und wir brauchen eine neue Zahl:

```
          BNE RNDZUG
```

Jetzt haben wir einen zulässigen Zug und können ihn ausführen. Dazu muß die externe LED-Nummer beim Rücksprung in X stehen. Da X die interne LED-Nummer bereits enthält, müssen wir diesen Wert nur noch um 1 erhöhen:

```
          INX
FERTIG    RTS
```

### *Unterprogramm FINDZUG*

Diese Routine wertet das Spielbrett dahingehend aus, daß sie ein Feld sucht, das die in A und X fixierten Bedingungen erfüllt: Der Akkumulator bezeichnet dabei eine spezifische Reihensumme, das X-Register legt fest, wievielmal ein spezielles Feld der Reihe mit der in A fixierten Reihensumme angehören muß.

Zu Beginn setzt die FINDZUG-Routine alle Felder auf den Status 0. Jedesmal wenn sie im weiteren Verlauf ein Feld findet, das die festgelegte Reihensummenbedingung erfüllt, erhöht sie den Status dieses Feldes um 1. Am Ende des Auswertungsprozesses kennzeichnet ein Feldstatus 1 also ein Feld, das die Reihensummenbedingung einmal erfüllt, ein Feldstatus 2 ein Feld, das sie zweimal erfüllt usw.

Zum Schluß werden dann alle Felder auf ihre Statuszahl geprüft. Stimmt diese mit der Zahl in X überein, so ist ein Feld gefunden, das die beiden mitgebrachten Anfangsbedingungen erfüllt.

Das vollständige Flußdiagramm der FINDZUG-Routine erscheint in Bild 11.49. Generell unterteilt sie sich in drei Schritte: Schritt 1 ist eine Initialisierungsphase; Schritt 2 wählt die Felder aus, die die Reihensummenbedingung (mitgebracht im Akkumulator) erfüllen, wobei jedes leere Feld eine Statuserhöhung erfährt, das einer Reihe mit der spezifizierten Summe angehört; Schritt 3 schließlich überprüft alle akkumulierten Statuswerte auf den Parameter, der in X fixiert war. Verläuft ein solcher Vergleich positiv, so endet der Prozeß, und das gefundene Feld ist das, wo der Computer spielen muß. Wird kein brauchbares Feld gefunden, so ist X beim Rücksprung auf 0 dekrementiert, und dieser Wert ist dann das Flaggenzeichen dafür, daß die Routine kein brauchbares Feld gefunden hat.

Betrachten wir die Routine nun im einzelnen, sie beginnt in Zeile 204 der Listings.

*Schritt 1: Initialisierung*

Da die Register X und A von der Routine selbst gebraucht werden, müssen die darin mitgebrachten Bedingungswerte zunächst zwischengespeichert werden, das geschieht in den Adressen TEMP1 und TEMP2 (siehe Bild 11.47):

```
FINDZUG     STX TEMP2
            STA TEMP1
```

Den 0-Status für alle Felder stellt eine Schleife her, die neunmal durchlaufen wird (neun Felder hat das Spielbrett):

```
            LDA #0
            LDY #8
CLRLP       STA FDSTAT,Y
            DEY
            BPL CLRLP
```

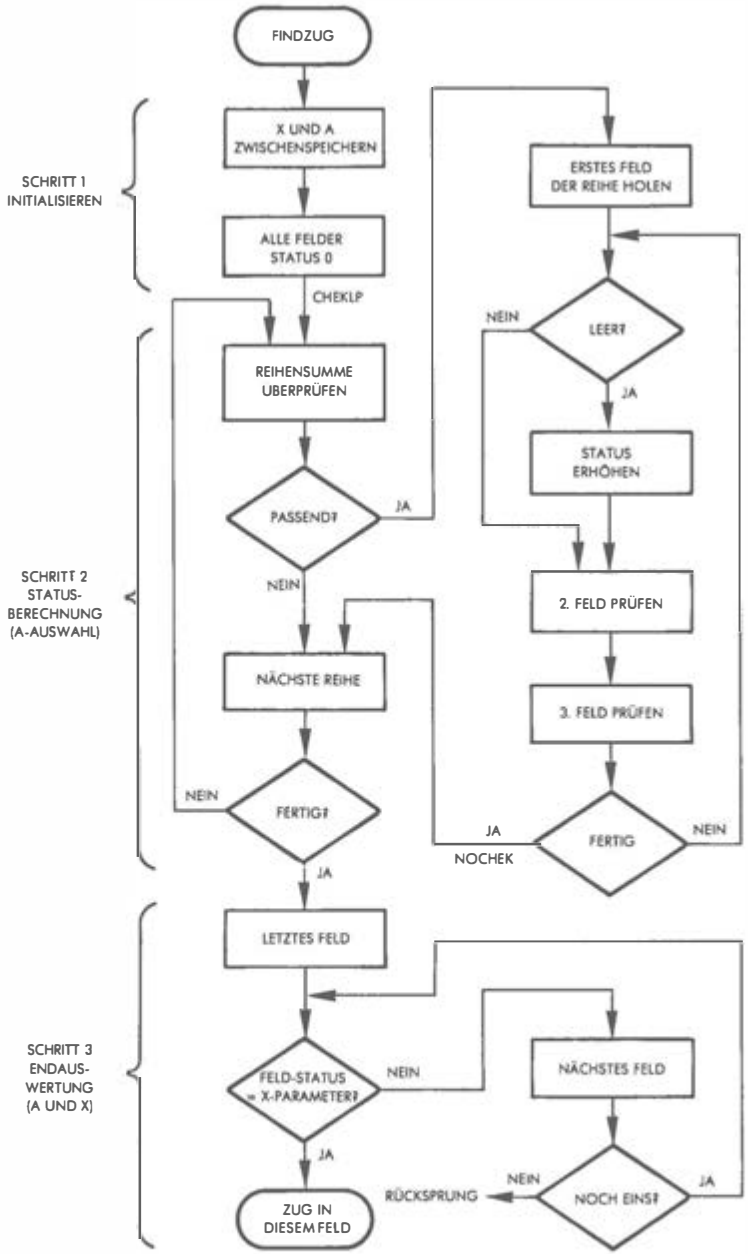



*Abb. 11.49: Flußdiagramm FINDZUG*

*Schritt 2: Statusermittlung für alle Felder*

Jetzt werden die acht Reihensummen nacheinander untersucht. Wird eine Entsprechung zum anfangs mitgebrachten Wert (mittlerweile in TEMP1 zwischengespeichert) gefunden, so wird der Statuswert eines zugehörigen Leerfeldes um 1 erhöht, andernfalls wird der nächste Reihensummenwert untersucht. Als Reihenzeiger fungiert das Y-Register, das nacheinander auf die einzelnen Eintragungen in der RHZG-Tabelle verweist, es wird von 7 auf 0 heruntergezählt:

```
        LDY #7
CHEKLP  LDA TEMP1
        CMP RHNSUM,Y
        BNE NOCHEK
```

Verfolgen wir zunächst den Fall, daß wir eine passende Reihensumme gefunden haben. Wir müssen nun jedes Feld dieser Reihe darauf prüfen, ob es leer ist. Die drei Feldnummern holen wir uns, mit Y als Zeiger, aus RHZG1, RHZG2 und RHZG 3 ins X-Register:

```
        LDX RHZG1,Y
```

Wenn das Feld mit der in X enthaltenen Nummer leer ist, erhöht die weiter unten beschriebene CNTSUB-Routine den Status dieses Feldes:

```
        JSR CNTSUB
```

Mit den beiden restlichen Feldern dieser Reihe wird analog verfahren:

```
        LDX RHZG2,Y
        JSR CNTSUB
        LDX RHZG3,Y
        JSR CNTSUB
```

Damit ist die Reihe durchgetestet, und das geschieht mit allen Reihen:

```
NOCHEK  DEY
        BPL CHEKLP
```

Haben wir alle Reihen überprüft, folgt der dritte Schritt:

*Schritt 3: Schlußauswahl*

Mit dem X-Register als Zeiger werden nun alle Felder nacheinander daraufhin geprüft, ob eines die Bedingung erfüllt, die zu Beginn in X mitgeführt (und in TEMP2 zwischengespeichert) wurde. Anfangswert für X ist 9:

```
        LDX #9
```

Die Statuswerte können nun durchgetestet werden:

```
FNMTCH    LDA TEMP2
          AND FDSTAT-1,X
```

Passen Status und Parameter zusammen, dann haben wir unseren Zug,

```
          BNE GFUNDN
```

andernfalls versuchen wir es weiter:

```
          DEX
          BNE FNMTCH
GFUNDN    RTS
```

**Übung 11-5:** *Warum wurden zum Vergleichen statt der Befehle CMP und BEQ die Befehle AND und BNE benutzt? (Hinweis: Prüfen Sie, wie sich die Strategie des Programms ändern würde.)*

### *Unterprogramm CNTSUB*

Diese Routine, die nur vom FINDZUG-Unterprogramm aufgerufen wird, erhöht den Status eines in X gespeicherten Feldes um 1, wenn dieses Feld leer ist. Ob das Feld leer ist, wird in der BRETT-Tabelle nachgeprüft:

```
CNTSUB    LDA BRETT,X
          BNE NOCNT
```

Ist das Feld besetzt, erfolgt sofort der Rücksprung, andernfalls wird vorher der Status inkrementiert:

```
          INC FDSTAT,X
NOCNT     RTS
```

### *Unterprogramm STELLNG*

Jede Zugausführung muß auf dem Spielbrett dargestellt werden, sie muß aber auch intern in der BRETT-Tabelle eingetragen werden. Dies und die Neuberechnung der Reihensummen erledigt die STELLNG-Routine.

Der Code des Zugausführenden ist im Akkumulator, und die Brettposition ist im X-Register, das für die interne LED-Verarbeitung zunächst dekrementiert werden muß:

```
STELLNG   DEX
```

Danach wird der Zug in die richtige Position der BRETT-Tabelle gebracht:

```
STA BRETT,X
```

Sie bemerken, daß der X-Wert als Zeiger auf die entsprechende Tabellenposition benutzt wird. Der im Akkumulator enthaltene Spielercode muß nun aber auch dafür verwendet werden, die entsprechende LED stetig leuchten oder aber blinken zu lassen. Da der Spielercode 1 und der Computercode 4 ist, testen wir A auf diesen Wert,

```
CMP #4
BEQ NBLINK
```

wonach bei einem Computerzug nach NBLINK verzweigt wird. Verfolgen wir aber zunächst einen Spielerzug:

```
JSR LICHT
```

Nach der Rückkehr von dem Unterprogramm LICHT (Beschreibung weiter unten) enthält der Akkumulator das Bitmuster der zu blinkenden LED, und die Blinkmasken müssen aktualisiert werden:

```
ORA LTMSKL
STA LTMSKL
```

Wenn das Carrybit nach LICHT 0 ist, so ist ein Bit zwischen 0 und 7 gesetzt, und wir sind fertig:

```
BCC NBLINK
```

Andernfalls (Carry = 1) muß Bit 9 gesetzt werden, die höherwertige Maske wird also modifiziert:

```
LDA #1
STA LTMSKH
```

Nachdem die LED-Masken richtig gesetzt sind, können die LEDs angesprochen werden:

```
NBLINK    JSR LEDLTR
```

Die LEDLTR-Routine erleuchtet also die in X fixierte LED, wobei bei einem Computerzug ein Dauerlicht entsteht; bei einem Spielerzug dagegen sorgen Unterbrechungen für einen Blinkeffekt.

Jetzt müssen die Reihensummen noch neu berechnet werden; als Reihenzeiger fungiert das X-Register. Angesichts der bevorstehenden Additionen muß das Carrybit gelöscht werden:

```
                LDX #7
ADDREI          CLC
```

Beginnen wir mit dem ersten Feld von Reihe 8:

```
LDY RHZG1,X
```

Jetzt steht im Y-Register die interne Feldnummer, und wir werden dies gleich für eine weitere Indizierung ausnutzen. Zum Berechnen der neuen Reihensumme lesen wir den Inhalt des Feldes. (Ob eine Reihensumme sich verändert hat oder nicht, wird nicht untersucht, da alle acht Reihen neu berechnet werden.):

```
LDA BRETT,Y
```

Beachten Sie die zweistufige Indizierung bei den beiden letzten Befehlen, die eine besonders effiziente Datenbearbeitungstechnik darstellt. Im Akkumulator steht jetzt also der Wert des ersten Feldes. Dazu werden nun nacheinander die beiden nächsten Feldinhalte addiert:

```
LDY RHZG2,X
ADC BRETT,Y
```

Die zweite Feldnummer wurde von der LDY-Instruktion geholt, und der ADC-Befehl fügte zum Akkumulator den passenden Wert aus der BRETT-Tabelle hinzu. Dasselbe geschieht noch mit dem dritten Feld:

```
LDY RHZG3,X
ADC BRETT,Y
```

Die im Akkumulator nun fixierte Reihensumme wird in der RHNSUM-Tabelle abgelegt, Reihenzeiger ist immer noch X:

```
STA RHNSUM,X
```

Jetzt ist die nächste Reihe dran:

```
DEX
BPL ADDREI
```

Ist X unter 0 dekrementiert, sind wir fertig:

```
RTS
```

***Unterprogramm LEDLTR***

Beim Einsprung in diese Routine wird vorausgesetzt, daß X die interne Nummer der zu erleuchtenden LED enthält. Mit Hilfe der LICHT-Routine, die die LED-Nummer in X in ein Bitmuster umwandelt, das im Akkumulator zurück-

kommt, wird die entsprechende LED erleuchtet:

```
LEDLTR      JSR LICHT
```

Nun muß entweder Tor A oder T B neu gesetzt werden. Wenn es nicht Tor A ist, was an der Carry-Flagge ablesbar ist, so ist der Akkumulatorinhalt 0, und Tor A bleibt unverändert, andernfalls erhält es seinen neuen Wert:

```
            ORA TOR1A
            STA TOR1A
            BCC LTRND
```

War Carry = 1, muß LED 9 erleuchtet werden, eine 1 geht also an Tor B:

```
            LDA #1
            STA TOR1B
LTRDN       RTS
```

***Unterprogramm SPLZUG***

Diese Routine holt einen zulässigen Spielerzug. Ein Tonsignal ist die Aufforderung an den Spieler, seine Taste zu drücken. Ist es eine Taste außerhalb des Bereichs 1 bis 9, so wird die Eingabe ignoriert, und dasselbe geschieht, wenn auf ein bereits besetztes Feld gezogen werden soll. Erzeugen wir zunächst ein Trillersignal,

```
SPLZUG      LDA #$80
            STA DAUER
            LDA #$10
            JSR TON
```

und holen wir dann eine Tasteneingabe:

```
TASTE       JSR GETKEY
```

Zunächst machen wir die Probe auf den Höchstwert 9,

```
            CMP #10
            BCS TASTE
```

dann auf den Minimalwert 0:

```
            TAX
            BEQ TASTE
```

Schließlich prüfen wir, ob das Feld bereits besetzt ist:

```
            LDA BRETT-1,X
            BNE TASTE
            RTS
```

**Übung 11-6:** *Ändern Sie die SPLZUG-Routine so, daß bei jeder unzulässigen Zugeingabe ein Trillersignal ertönt. Um den Spieler auf seinen Fehler aufmerksam zu machen, sollte eine Zweitrillersequenz mit einem anderen Ton generiert werden.*

### *Unterprogramm LICHT*

Diese Routine greift die in X mitgebrachte LED-Nummer auf und wandelt sie ins entsprechende Bitmuster im Akkumulator um. Bei LED 9 (X=8) wird Carry gesetzt. Dieser Programmteil ist aus früheren Kapiteln bekannt:

```
LICHT    STX TEMP1
         LDA #0
         SEC
SHIFT    ROL A
         DEX
         BPL SHIFT
         LDX TEMP1
         RTS
```

### *Unterprogramm DELAY*

Eine klassische Verzögerungsroutine mit Doppelschleife und zusätzlichen „Nichtstun"-Befehlen zum Zeitgewinnen:

```
DELAY    LDY #$FF
DL1      LDX #$FF
DL2      ROL DAUER
         ROR DAUER
         DEX
         BNE DL2
         DEY
         BNE DL1
         RTS
```

### *Die INTERRUPT-Routine*

Bei jeder Unterbrechung werden die jeweiligen LEDs komplementiert (aus „aus" wird „ein", aus „ein" wird „aus"). Die LED-Positionen sind in den LTMSK-Masken fixiert (höher- und niederwertiges Byte).

Der Ein/Aus-Effekt wird durch den EOR-Befehl bewirkt, der ja eine logische Komplementierung darstellt. Da der Akkumulator benutzt wird, muß sein Inhalt auf dem Stapel zwischengespeichert werden.

```
INTVEC   PHA
         LDA TOR1A
```

```
EOR LTMSKL
STA TOR1A
LDA TOR1B
EOR LTMSKH
STA TOR1B
LDA T1LL
PLA
RTI
```

**Übung 11-7:** *Beachten Sie den obigen Befehl LDA T1LL gefolgt von PLA, was den Akkumulatorinhalt mit dem Stapelinhalt überschreibt. Der Akkumulatorinhalt, der gerade aus T1LL eingelesen wurde, wird also gleich darauf zerstört. Ist das ein Programmfehler? Wenn nicht: Was soll diese Befehlsfolge? (Hinweis: Wir sind dieser Situation bereits in früheren Kapiteln begegnet.)*

### *Unterprogramm INIT*

Die Initialisierungs-Routine ist mit dem Hauptprogramm besprochen worden.

### *Unterprogramme RANDOM und TON*

Diese beiden Unterprogramme wurden in früheren Kapiteln erläutert.

## ZUSAMMENFASSUNG

Dieses Programm ist das komplexeste, das wir entwickelt haben. Nach einer ausführlichen Diskussion verschiedener Algorithmusansätze wird ein „Ad hoc"-Algorithmus schließlich implementiert. Lesern, die sich für die Programmierung von Strategiespielen interessieren, sei die Entwicklung eines alternativen Algorithmus empfohlen.

```
;TIC-TAC-TOE
;SPIELPROGRAMM MENSCH GEGEN COMPUTER.
;SPIELBRETT IST 3*3 LED-MATRIX UND TASTATUR.
;WIRD ZU SPIELBEGINN  F -TASTE GEDRUCKT, SO MACHT DER SPIELER
;DEN ERSTEN ZUG, BEI ALLEN ANDEREN TASTEN FANGT DER COMPUTER AN.
;DIE WEITERE ZUGEINGABE ERFOLGT DURCH DRUCKEN DERJENIGEN TASTE,
;DIE DEM GEWUNSCHTEN FELD AUF DER LED-MATRIX ENTSPRICHT.
;
GETKEY    = $100
ACCESS    = $BBB6
TOR1A     = $A001        ;6522 VIA 1
DDR1A     = $A003
TOR1B     = $A000
DDR1B     = $A002
IER       = $A00E        ;UNTERBRECHUNGSZULASSUNGSREGISTER
ACR       = $A00B        ;HILFSKONTROLLREGISTER
T1LL      = $A004        ;ZEITGEBER 1
T1CH      = $A005
TOR3B     = $AC00        ;6522 VIA 3
```

*Abb. 11.50: Programm TIC-TAC-TOE*

```
                DDR3B     = $AC02
                IROVL     = $A67E
                IROVH     = $A67F
                CLRST     = $18          ;BEGINN DES DURCH DAS 'INIT'-UNTERPRO-
                                         ;ZU LöSCHENDEN SPEICHERBEREICHS
                BRETT     = $1B          ;SPIELBRETT (9 ADRESSEN)
                FDSTAT    = $21          ;TAKTISCHER STATUS (9 ADRESSEN)
                RHNSUM    = $2A          ;REIHENSUMMEN (8 ADRESSEN)
                                         ;COMPUTERBESETZTES FELD ZäHLT 4,
                                         ;SPIELERBESETZTES FELD ZäHLT 1,
                                         ;LEERES FELD ZäHLT 0.
                RNDSCR    = $32          ;ZUFALLSZAHLEN (6 ADRESSEN)
                TEMP1     = $38
                TEMP2     = $39
                ZUGNR     = $3A          ;AKTUELLE ZUGNUMMER
                SPIELR    = $3B          ;ZEIGER, WER AM ZUG IST
                LTMSKH    = $3C          ;LED-BLINKMASKE HIGH BYTE
                LTMSKL    = $3D          ;LED-BLINKMASKE LOW BYTE
                DAUER     = $3E          ;TONDAUERKONSTANTE
                FREQ      = $3F          ;FREQUENZKONSTANTE
                CLREND    = $40          ;ENDE DES DURCH 'INIT' ZU LöSCHENDEN
                                         ;SPEICHERBEREICHS
                ODDMSK    = $40          ;MASKE FüR UNGERADE ZUFALLSZAHLEN
                INTEL     = $41          ;INTELLIGENZQUOTIENT (IQ)
                ;
                ;TABELLE DER FELDER FüR DIE ACHT MöGLICHEN DREIERREIHEN
                ;
0000: 00        RHZG1     .BYTE 0,1,2,0,3,6,0,2
0001: 01
0002: 02
0003: 00
0004: 03
0005: 06
0006: 00
0007: 02
0008: 03        RHZG2     .BYTE 3,4,5,1,4,7,4,4
0009: 04
000A: 05
000B: 01
000C: 04
000D: 07
000E: 04
000F: 04
0010: 06        RHZG3     .BYTE 6,7,8,2,5,8,8,6
0011: 07
0012: 08
0013: 02
0014: 05
0015: 08
0016: 08
0017: 06
                ;
                ;HAUPTPROGRAMM
                ;
                          * = $200
0200: A9 0C     START     LDA #12
0202: 85 41               STA INTEL      ;ANFANGS-IQ = 75%
0204: 20 50 00  RESTRT    JSR INIT       ;INITIALISIERUNGSROUTINE
0207: 20 00 01            JSR GETKEY     ;FRAGE: WER ZIEHT ZUERST?
020A: C9 0F               CMP #$F        ;TASTE 'F' GEDRüCKT?
020C: D0 04               BNE SPISCHL
020E: A9 01               LDA #1         ;JA: SPIELER FäNGT AN
0210: 85 3B               STA SPIELR
0212: E6 3A     SPISCHL   INC ZUGNR      ;ZUGE MITZäHLEN
0214: A5 3B               LDA SPIELR     ;WER IST AM ZUG?
0216: F0 0E               BEQ COMZUG     ;BEI 0: COMPUTER AM ZUG
0218: C6 3B               DEC SPIELR     ;SPIELER AM ZUG, DANACH COMPUTER
021A: 20 80 03            JSR SPLZUG     ;SPIELERZUG HOLEN
021D: A9 01               LDA #1         ;SPIELERKODE 1 ABSPEICHERN
021F: 20 40 03            JSR STELLNG    ;AUSFöHREN UND REIHENSUMMEN BERECHNEN
0222: A9 03               LDA #3         ;MUSTER FüR SIEGTEST LADEN
0224: D0 0F               BNE SIEGTST    ;AUF GEWINNSTELLUNG PRüFEN
```

*Abb. 11.50: Programm TIC-TAC-TOE (Fortsetzung)*

```
0226: E6 3B     COM2ZUG     INC SPIELR      ;COMPUTER AM ZUG, DANACH WIEDER SPIELER
0228: 20 A4 03              JSR DELAY       ;'GRUBELZEIT' FUR COMPUTER
022B: 20 9D 02              JSR ANALYSE     ;COMPUTERZUG ERMITTELN
022E: A9 04                 LDA #4          ;COMPUTERKODE 4 LADEN
0230: 20 40 03              JSR STELLNG     ;...UND 'STEIN SETZEN'
0233: A9 0C                 LDA #12         ;MUSTER FUR SIEGSTELLUNGSSUCHE
0235: A0 07     SIEGTST     LDY #7          ;ZAHLER FUR REIHENSUMMENTEST
0237: D9 2A 00  TSTLP       CMP RHNSUM,Y    ;AUF GEWINNSTELLUNG PRUFEN
023A: F0 11                 BEQ SIEG        ;WENN SIEGSTELLUNG GEFUNDEN: VERZWEIGEN
023C: BB                    DEY             ;SONST WEITERSUCHEN
023D: 10 FB                 BPL TSTLP
023F: A5 3A                 LDA ZUGNR       ;ZUGNUMMER HOLEN
0241: C9 09                 CMP #9          ;9. ZUG?
0243: D0 CB                 BNE SPISCHL     ;WENN JA: SPIEL ZUENDE (UNENTSCHIEDEN)
0245: A9 FF                 LDA #$FF        ;ALLE LEDS BLINKEN
0247: 85 3D                 STA LTMSKL
0249: 85 3C                 STA LTMSKH
024B: D0 4A                 BNE DLY         ;EINE WEILE BLINKEN LASSEN
024D: C9 0C     SIEG        CMP #12         ;COMPUTER GEWONNEN?
024F: F0 0E                 BEQ INTDN       ;WENN JA: IQ HERUNTERSETZEN
0251: A9 1E                 LDA #30         ;FREQUENZKONSTANTE FUR SIEG-TON
0253: 85 3F                 STA FREQ
0255: A5 41                 LDA INTEL
0257: C9 0F                 CMP #$0F        ;HOCHSTER IQ-WERT ERREICHT?
0259: F0 0E                 BEQ GTMSK       ;WENN JA: KEINE ANDERUNG
025B: E6 41                 INC INTEL       ;SONST IQ ERHOHEN
025D: D0 0A                 BNE GTMSK       ;REIHE ERLEUCHTEN
025F: A9 FF     INTDN       LDA #$FF        ;FREQUENZKONSTANTE FUR VERLUST-TON
0261: 85 3F                 STA FREQ
0263: A5 41                 LDA INTEL       ;IQ = 0?
0265: F0 02                 BEQ GTMSK       ;WENN JA: NICHT WEITER SENKEN
0267: C6 41                 DEC INTEL       ;SONST IQ HERUNTERSETZEN
0269: A9 00     GTMSK       LDA #0          ;ALLE LEDS LOSCHEN
026B: BD 01 A0              STA TOR1A
026E: BD 00 A0              STA TOR1B
0271: B6 00                 LDX RHZG1,Y     ;BIT FUR ERSTE GEWINNREIHEN-LED IN A LA-
0273: 20 6F 03              JSR LEDLTR      ;DEN UND LED ERLEUCHTEN
0276: B6 08                 LDX RHZG2,Y     ;ZWEITES BIT
0278: 20 6F 03              JSR LEDLTR
027B: B6 10                 LDX RHZG3,Y     ;DRITTES BIT
027D: 20 6F 03              JSR LEDLTR
0280: AD 01 A0              LDA TOR1A       ;UNNOTIGE BITS IN MASKE ELIMINIEREN
0283: 25 3D                 AND LTMSKL
0285: 85 3D                 STA LTMSKL
0287: AD 00 A0              LDA TOR1B
028A: 25 3C                 AND LTMSKH
028C: 85 3C                 STA LTMSKH
028E: A9 FF                 LDA #$FF        ;TONDAUERKONSTANTE
0290: 85 3E                 STA DAUER
0292: A5 3F                 LDA FREQ        ;FREQUENZKONSTANTE
0294: 20 AD 00              JSR TON         ;TON SPIELEN
0297: 20 A4 03  DLY         JSR DELAY       ;ETWAS VERWEILEN
029A: 4C 04 02              JMP RESTRT      ;NEUES SPIEL, IQ UNVERANDERT
                            ;
                            ;UNTERPROGRAMM 'ANALYSE'
                            ;BRETTAUSWERTUNG. RUCKSPRUNG MIT GEFUNDENEM ZU IM X-REGISTER
                            ;
029D: A9 00     ANALYSE     LDA #0          ;MASKE FUR ZUFALLSZUG
029F: 85 40                 STA ODDMSK
02A1: A9 0B                 LDA #0B         ;AUF GEWINNZUG FUR COMPUTER PRUFEN
02A3: A2 03                 LDX #3
02A5: 20 04 03              JSR FINDZUG
02A8: D0 59                 BNE FERTIG      ;WENN ZUG GEFUNDEN: RUCKSPRUNG
02AA: A9 02                 LDA #2          ;AUF GEWINNZUG FUR SPIELER PRUFEN

02AC: A2 03                 LDX #3
02AE: 20 04 03              JSR FINDZUG
02B1: D0 50                 BNE FERTIG      ;WENN ZUG GEFUNDEN: RUCKSPRUNG
02B3: A9 04                 LDA #4          ;KANN COMPUTER FALLE AUFBAUEN?
02B5: A2 02                 LDX #2
02B7: 20 04 03              JSR FINDZUG
02BA: D0 47                 BNE FERTIG      ;WENN JA: ZUG AUSFUHREN
```

*Abb. 11.50: Programm TIC-TAC-TOE (Fortsetzung)*

```
02BC: 20 9A 00            JSR RANDOM     ;ZUFALLSZAHL HOLEN
02BF: 29 0F               AND #$0F       ;AUF WERT ZWISCHEN 0 UND 15 BESCHNEIDEN
02C1: C5 41               CMP INTEL      ;ENTSPRECHEND 'KLUG' ODER 'DUMM' SPIELEN
02C3: F0 02               BEQ OK         ;WENN BEIDE GLEICH: TEST UEBERSPRINGEN
02C5: B0 2B               BCS RNDZUG     ;WENN ZAHL>IQ: 'DUMMEN' ZUG MACHEN
02C7: A6 3A       OK      LDX ZUGNR
02C9: E0 01               CPX #1         ;ERSTER ZUG?
02CB: F0 25               BEQ RNDZUG     ;WENN JA: BELIEBIGES FELD BESETZEN
02CD: E0 04               CPX #4         ;VIERTER ZUG?
02CF: D0 0C               BNE FALLE      ;WENN NICHT: HIER WEITER
02D1: A2 06               LDX #6         ;INDEX FUR 1. DIAGONALE REIHENSUMME
02D3: BA                  TXA            ;REIHENSUMME FUR STELLUNG S-C-S
02D4: D5 2A               CMP RHNSUM,X   ;IST ES DIESE STELLUNG?
02D6: F0 16               BEQ ODDRND     ;WENN JA: SEITENZUG
02D8: E8                  INX            ;ZWEITE DIAGONALE PRUFEN
02D9: D5 2A               CMP RHNSUM,X
02DB: F0 11               BEQ ODDRND
02DD: A9 01       FALLE   LDA #1         ;KANN SPIELER FALLE AUFBAUEN?
02DF: A2 02               LDX #2
02E1: 20 04 03            JSR FINDZUG
02E4: D0 1D               BNE FERTIG     ;WENN JA: BLOCKIEREN
02E6: A6 1C               LDX BRETT+4    ;ZENTRUMSFELD BESETZT?
02E8: D0 08               BNE RNDZUG
02EA: A2 05               LDX #5         ;NEIN: DORTHIN SETZEN
02EC: D0 15               BNE FERTIG
02EE: A9 01       ODDRND  LDA #1         ;'ODDMSK'=1, DAMIT SEITENZUG
02F0: B5 40               STA ODDMSK
02F2: 20 9A 00    RNDZUG  JSR RANDOM     ;ZAHL FUR ZUFALLSZUG HOLEN
02F5: 29 0F               AND #$0F       ;AUF WERT 0-15 BRINGEN
02F7: 05 40               ORA ODDMSK     ;FUR ECKZUG UNGERADE MACHEN
02F9: C9 09               CMP #9         ;ZAHL ZU GROSS?
02FB: B0 F5               BCS RNDZUG     ;WENN JA: NOCH EINMAL
02FD: AA                  TAX
02FE: B5 18               LDA BRETT,X    ;FELD SCHON BESETZT?
0300: D0 F0               BNE RNDZUG     ;WENN JA: NOCH EINMAL
0302: EB                  INX            ;X FUR AUSGABE JUSTIEREN
0303: 60          FERTIG  RTS            ;RUCKSPRUNG MIT ZUG IM X-REGISTER
                  ;
                  ;UNTERPROGRAMM 'FINDZUG'
                  ;FINDET EIN FELD ENTSPRECHEND DEN IN A UND X MITGEBRACHTEN BE-
                  ;DINGUNGEN. DIE REIHENSUMME IN A MUSS EIN FELD X-MAL ENTHALTEN.
                  ;WERT 1 QUALIFIZIERT EIN FELD.
                  ;
0304: B6 39       FINDZUG STX TEMP2      ;REGISTER ZWISCHENSPEICHERN
0306: 85 38               STA TEMP1
0308: A9 00               LDA #0         ;FELDSTATUSREGISTER LOSCHEN
030A: A0 08               LDY #8
030C: 99 21 00    CLRLP   STA FDSTAT,Y
030F: 88                  DEY
0310: 10 FA               BPL CLRLP
0312: A0 07               LDY #7         ;SCHLEIFE SIEBENMAL DURCHLAUFEN
0314: A5 38       CHEKLP  LDA TEMP1      ;REIHENSUMME=PARAMETER?
0316: D9 2A 00            CMP RHNSUM,Y
0319: D0 0F               BNE NOCHEK     ;WENN NICHT: NACHSTER VERSUCH
031B: B6 00               LDX RHZG1,Y    ;1. FELD IN REIHE PRUFEN
031D: 20 39 03            JSR CNTSUB     ;WENN LEERFELD: STATUS ERHOHEN
0320: B6 08               LDX RHZG2,Y    ;2. FELD
0322: 20 39 03            JSR CNTSUB
0325: B6 10               LDX RHZG3,Y    ;3. FELD
0327: 20 39 03            JSR CNTSUB
032A: 88          NOCHEK  DEY            ;NACHSTE REIHE
032B: 10 E7               BPL CHEKLP
032D: A2 09               LDX #9
032F: A5 39       FNMTCH  LDA TEMP2      ;PARAMETER LADEN
0331: 35 20               AND FDSTAT-1,X;(FELDSTATUS)AND(PARAMETER)>0?
0333: D0 03               BNE GFUNDN     ;WENN JA: X IST DER ZUG
0335: CA                  DEX            ;NACHSTER FELDSTATUS
0336: D0 F7               BNE FNMTCH
033B: 60          GFUNDN  RTS
                  ;
                  ;UNTERPROGRAMM 'CNTSUB'
                  ;INKREMENTIERT FELDSTATUS VON LEERFELDERN
```

*Abb. 11.50: Programm TIC-TAC-TOE (Fortsetzung)*

```
                        ;
0339: B5 18     CNTSUB  LDA BRETT,X     ;FELD HOLEN
033B: D0 02             BNE NOCNT       ;WENN BESETZT: ÜBERSPRINGEN
033D: F6 21             INC FDSTAT,X    ;FELDSTATUS ERHÖHEN
033F: 60        NOCNT   RTS             ;FERTIG
                        ;
                        ;UNTERPROGRAMM 'STELLNG'
                        ;FÜHRT ZUG AUS. SPIELERKODE IM AKKUMULATOR, FELD IM X-REGISTER.
                        ;ERLEUCHTET/BLINKT ENTSPRECHENDE LED UND BERECHNET REIHENSUMMEN.
                        ;
0340: CA        STELLNG DEX             ;ZUR KORREKTEN INDIZIERUNG JUSTIEREN
0341: 95 18             STA BRETT,X     ;ZUG AUSFÜHREN
0343: C9 04             CMP #4          ;COMPUTERZUG?
0345: F0 0D             BEQ NBLINK      ;WENN JA: NICHT BLINKEN LASSEN
0347: 20 9B 03          JSR LICHT       ;SPIELERZUG: BIT ZUM LED-BLINKEN HOLEN
034A: 05 3D             ORA LTMSKL      ;BLINKMASKEN SETZEN
034C: 85 3D             STA LTMSKL
034E: 90 04             BCC NBLINK      ;WENN C=0: BIT 9 NICHT SETZEN
0350: A9 01             LDA #1          ;BIT 9 BLINKT
0352: 85 3C             STA LTMSKH
0354: 20 6F 03  NBLINK  JSR LEDLTR      ;LED AN
0357: A2 07             LDX #7          ;REIHENSUMMENBERECHNUNG: X ZÄHLT
0359: 18        ADDREI  CLC             ;FERTIG FÜR ADDITION
035A: B4 00             LDY RHZG1,X     ;ERSTE FELDADRESSE
035C: B9 18 00          LDA BRETT,Y     ;FELDINHALT HOLEN
035F: B4 08             LDY RHZG2,X     ;2. FELD
0361: 79 18 00          ADC BRETT,Y
0364: B4 10             LDY RHZG3,X     ;LETZTES FELD ADDIEREN
0366: 79 18 00          ADC BRETT,Y
0369: 95 2A             STA RHNSUM,X    ;REIHENSUMME ABSPEICHERN
036B: CA                DEX
036C: 10 EB             BPL ADDREI      ;NÄCHSTE REIHENSUMME
036E: 60                RTS
                        ;
                        ;UNTERPROGRAMM 'LEDLTR'
                        ;ERLEUCHTET LED ENTSPRECHEND WERT IM X-REGISTER
                        ;
036F: 20 98 03  LEDLTR  JSR LICHT       ;BIT POSITIONIEREN
0372: 0D 01 A0          ORA TOR1A       ;LED AN
0375: 8D 01 A0          STA TOR1A
0378: 90 05             BCC LTRDN       ;WENN LED 9 NICHT GEMEINT: ÜBERSPRINGEN
037A: A9 01             LDA #1          ;LED 9 AN
037C: 8D 00 A0          STA TOR1B
037F: 60        LTRDN   RTS             ;FERTIG
                        ;
                        ;UNTERPROGRAMM 'SPLZUG'
                        ;HOLT SPIELERZUG UND ÜBERPRÜFT AUF FEHLER
                        ;
0380: A9 80     SPLZUG  LDA #$80        ;KURZER PIEPTON ALS AUFFORDERUNG, EINE
0382: 85 3E             STA DAUER       ;TASTE ZU DRÜCKEN
0384: A9 10             LDA #$10
0386: 20 AD 00          JSR TON
0389: 20 00 10  TASTE   JSR GETKEY      ;ZUGEINGABE HOLEN
038C: C9 0A             CMP #10         ;UNZULÄSSIG?
038E: B0 F9             BCS TASTE       ;WENN JA: WIEDERHOLEN
0390: AA                TAX
0391: F0 F6             BEQ TASTE       ;WENN 0-TASTE: WIEDERHOLEN
0393: B5 17             LDA BRETT-1,X   ;FELD LEER?
0395: D0 F2             BNE TASTE       ;WENN NICHT: EINGABE WIEDERHOLEN
0397: 60                RTS
                        ;
                        ;UNTERPROGRAMM 'LICHT'
                        ;BITWEISES VERSCHIEBEN VON '1' IM AKKUMULATOR ENTSPRECHEND DEM
                        ;DEM IN X MITGEBRACHTEN WERT. WENN X=0, IST CARRY GESETZT.
                        ;
0398: 86 3B     LICHT   STX TEMP1       ;X ZWISCHENSPEICHERN
039A: A9 00             LDA #0          ;AKKUMULATOR FÜR VERSCHIEBEN LEEREN
039C: 38                SEC             ;BIT SETZEN
039D: 2A        SHIFT   ROL A           ;BIT EINE POSITION NACH LINKS
039E: CA                DEX
039F: 10 FC             BPL SHIFT       ;SCHLEIFE DURCHZÄHLEN
03A1: A6 3B             LDX TEMP1       ;X ZURÜCKHOLEN
```

*Abb. 11.50: Programm TIC-TAC-TOE (Fortsetzung)*

```
03A3: 60                RTS
                        ;
                        ;UNTERPROGRAMM 'DELAY'
                        ;
03A4: A0 FF     DELAY   LDY #$FF
03A6: A2 FF     DL1     LDX #$FF
03A8: 26 3E     DL2     ROL DAUER       ;ZEIT 'VERSCHWENDEN'
03AA: 66 3E             ROR DAUER
03AC: CA                DEX
03AD: D0 F9             BNE DL2
03AF: 88                DEY
03B0: D0 F4             BNE DL1
03B2: 60                RTS
                        ;
                        ;UNTERBRECHUNGS-ROUTINE
                        ;BEI JEDEM INTERRUPT GEHEN LEDS ENTSPRECHEND DEN MUSTERN IN DEN
                        ;BLINKMASKEN AN ODER AUS.
                        ;
03B3: 48        INTVEC  PHA
03B4: AD 01 A0          LDA TOR1A
03B7: 45 3D             EOR LTMSKL
03B9: 8D 01 A0          STA TOR1A
03BC: AD 00 A0          LDA TOR1B
03BF: 45 3C             EOR LTMSKH
03C1: 8D 00 A0          STA TOR1B
03C4: AD 04 A0          LDA TILL
03C7: 68                PLA
03C8: 40                RTI
                        ;
                        ;UNTERPROGRAMM 'INIT'
                        ;INITIALISIERT DAS PROGRAMM
                        ;
                                * = $50
                        ;
0050: A9 00     INIT    LDA #0          ;SPEICHER LÖSCHEN
0052: A2 28             LDX #CLREND-CLRST
0054: 95 10     CLRALL  STA CLRST,X
0056: CA                DEX
0057: 10 FB             BPL CLRALL
0059: AD 04 A0          LDA TILL        ;STARTZAHL F. ZUFALLSZAHLENGENERATOR
005C: 85 33             STA RNDSCR+1
005E: 85 36             STA RNDSCR+4
0060: A9 FF             LDA #$FF
0062: 8D 03 A0          STA DDR1A       ;EIN/AUSGABE SETZEN
0065: 8D 02 A0          STA DDR1B
0068: 8D 02 AC          STA DDR3B
006B: A9 00             LDA #0          ;LEDS LÖSCHEN
006D: 8D 01 A0          STA TOR1A
0070: 8D 00 A0          STA TOR1B
                        ;ZEITGEBER FÜR BLINK-LEDS SETZEN
0073: 20 86 8B          JSR ACCESS      ;GESCHÜTZTEN SPEICHERBEREICH ÖFFNEN
0076: A9 B3             LDA #<INTVEC    ;UNTERBRECHUNGSVEKTOR LOW BYTE
0078: 8D 7E A6          STA IRQVL       ;ABSPEICHERN
007B: A9 03             LDA #>INTVEC    ;UNTERBRECHUNGSVEKTOR HIGH BYTE
007D: 8D 7F A6          STA IRQVH       ;ABSPEICHERN
0080: A9 7F             LDA #$7F        ;INTERRUPT ENABLE REGISTER LÖSCHEN
0082: 8D 0E A0          STA IER
0085: A9 C0             LDA #$C0        ;ZEITGEBER 1 INTERRUPT ERMÖGLICHEN
0087: 8D 0E A0          STA IER
008A: A9 40             LDA #$40        ;FREILAUFMODUS F. ZEITGEBER 1
008C: 8D 0B A0          STA ACR
008F: A9 FF             LDA #$FF
0091: 8D 04 A0          STA TILL        ;ZWISCHENSPEICHER LOW BYTE
0094: 8D 05 A0          STA T1CH        ;HIGH BYTE SETZEN UND STARTEN
0097: 58                CLI             ;UNTERBRECHUNGEN ZULASSEN
0098: DB                CLD
0099: 60                RTS
                        ;
                        ;UNTERPROGRAMM 'RANDOM'
                        ;ZUFALLSZAHLENGENERATOR BRINGT ZAHL IM AKKUMULATOR ZURÜCK
                        ;
009A: 38        RANDOM  SEC
```

*Abb. 11.50: Programm TIC-TAC-TOE (Fortsetzung)*

```
009B: A5 33             LDA RNDSCR+1
009D: 65 36             ADC RNDSCR+4
009F: 65 37             ADC RNDSCR+5
00A1: 85 32             STA RNDSCR
00A3: A2 04             LDX #4
00A5: B5 32     RNDLP   LDA RNDSCR,X
00A7: 95 33             STA RNDSCR+1,X
00A9: CA                DEX
00AA: 10 F9             BPL RNDLP
00AC: 60                RTS
                ;
                ;UNTERPROGRAMM 'TON'
                ;ERZEUGT TON MIT HALBZYKLENZAHL IN 'DAUER' UND WELLENLÄNGEN-
                ;KONSTANTE IM AKKUMULATOR
                ;
00AD: 85 3F     TON     STA FREQ
00AF: A9 FF             LDA #$FF
00B1: 8D 00 AC          STA TOR3B
00B4: A9 00             LDA #0
00B6: A6 3E             LDX DAUER
00B8: A4 3F     FL2     LDY FREQ
00BA: 88        FL1     DEY
00BB: 18                CLC
00BC: 90 00             BCC +2
00BE: D0 FA             BNE FL1
00C0: 49 FF             EOR #$FF
00C2: 8D 00 AC          STA TOR3B
00C5: CA                DEX
00C6: D0 F0             BNE FL2
00C8: 60                RTS

SYMBOLTABELLE:
ACCESS  BBB6    ACR     A00B    ADDREI  0359    ANALYSE 029D
CHEKLP  0314    CLRALL  0054    CLREND  0040    CLRLP   030C
CLRST   001B    CNTSUB  0339    COMZUG  0226    DDRIA   A003
DDR1B   A002    DDR3B   AC02    DELAY   03A4    DL1     03A6
DL2     03AB    DLY     0297    FERTIG  0303    DAUER   003E
FINDZUG 0304    FL1     00BA    FL2     00BB    FNMTCH  032F
GFUNDN  033B    FREQ    003F    GETKEY  0100    BRETT   001B
GTMSK   0269    IER     A00E    INIT    0050    INTDN   025F
INTEL   0041    INTVEC  03B3    IRQVH   A67F    IRQVL   A67E
TASTE   03B9    LEDLTR  036F    LICHT   039B    LTMSKH  003C
LTMSKL  003D    LTRDN   037F    ZUGNR   003A    NBLINK  0354
NOCHEK  032A    NOCNT   033F    ODDMSK  0040    ODDRND  02EE
OK      02C7    SPISCHL 0212    SPIELR  003B    SPLZUG  0380
TOR1A   A001    TOR1B   A000    TOR3B   AC00    RANDOM  009A
RESTRT  0204    RNDLP   00A5    RNDZUG  02F2    RNDSCR  0032
RHNSUM  002A    RHZG1   0000    RHZG2   000B    RHZG3   0010
SHIFT   039D    FDSTAT  0021    START   0200    TICH    A005
TILL    A004    TEMP1   003B    TEMP2   0039    TON     00AD
FALLE   02DD    TSTLP   0237    STELLNG 0340    SIEG    024D
SIEGIST 0235
```

*Abb. 11.50: Programm TIC-TAC-TOE (Fortsetzung)*

# Anhang A

## 6502 Befehlssatz – Alphabetisch

| | |
|---|---|
| **ADC** | Addieren mit Übertrag |
| **AND** | Logisches UND |
| **ASL** | Arithmetisches Linksschieben |
| **BCC** | Verzweigen, wenn Übertraggelöscht |
| **BCS** | Verzweigen, wenn Übertrag gesetzt |
| **BEQ** | Verzweigen, wenn Result = 0 |
| **BIT** | Teste Bit |
| **BMI** | Verzweigen, wenn Minus |
| **BNE** | Verzweigen, wenn ungleich 0 |
| **BPL** | Verzweigen, wenn Plus |
| **BRK** | Abbruch |
| **BVC** | Verzweigen, wenn Überlauf gelöscht |
| **BVS** | Verzweigen, wenn Überlauf gesetzt |
| **CLC** | Übertrag löschen |
| **CLD** | Dezimal-Flagge löschen |
| **CLI** | Unterbrechungsabschaltung löschen |
| **CLV** | Überlauf löschen |
| **CMP** | Vergleichen mit Akkumulator |
| **CPX** | Vergleichen mit X |
| **CPY** | Vergleichen mit Y |
| **DEC** | Dekrementieren Speicher |
| **DEX** | Dekrementieren X |
| **DEY** | Dekrementieren Y |
| **EOR** | Exklusives ODER |
| **INC** | Inkrementieren Speicher |
| **INX** | Inkrementieren X |
| **INY** | Inkrementieren Y |
| **JMP** | Verzweigen |
| **JSR** | Verzweigen in Unterprogramm |
| **LDA** | Laden Akkumulator |
| **LDX** | Laden X |
| **LDY** | Laden Y |
| **LSR** | Logisches Rechtsschieben |
| **NOP** | Leerbefehl (keine Operation) |
| **ORA** | Logisches ODER |
| **PHA** | Push A |
| **PHP** | Push P Status |
| **PLA** | Pop A |
| **PLP** | Pop P Status |
| **ROL** | Linksrotieren |
| **ROR** | Rechtsrotieren |
| **RTI** | Rückkehr von Unterbrechung |
| **RTS** | Rückkehr aus Unterprogramm |
| **SBC** | Subtrahieren mit Übertrag |
| **SEC** | Übertrag setzen |
| **SED** | Dezimal setzen |
| **SEI** | Unterbrechungsabschaltung setzen |
| **STA** | Akkumulator speichern |
| **STX** | X speichern |
| **STY** | Y speichern |
| **TAX** | A nach X übertragen |
| **TAY** | A nach Y übertragen |
| **TSX** | SP (Stapelzeiger) nach X übertragen |
| **TXA** | X nach A übertragen |
| **TXS** | X nach SP (Stapelzeiger) übertragen |
| **TYA** | Y nach A übertragen |

# Anhang B

## 6502-Befehlssatz: Hexadezimal mit Zeitangaben

| | | Inplizit | | | Akkumulator | | | Absolut | | | Seite 0 | | | Unmittelbar | | | Abs X | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Mnemonisch | | OP | n | # | OP | n | # | OP | n | # | OP | n | # | OP | n | # | OP | n | # |
| ADC | (1) | | | | | | | 6D | 4 | 3 | 65 | 3 | 2 | 69 | 2 | 2 | 7D | 4 | 3 |
| AND | (1) | | | | | | | 2D | 4 | 3 | 25 | 3 | 2 | 29 | 2 | 2 | 3D | 4 | 3 |
| ASL | | | | | 0A | 2 | 1 | 0E | 6 | 3 | 06 | 5 | 2 | | | | 1E | 7 | 3 |
| BCC | (2) | | | | | | | | | | | | | | | | | | |
| BCS | (2) | | | | | | | | | | | | | | | | | | |
| BEQ | (2) | | | | | | | | | | | | | | | | | | |
| BIT | | | | | | | | 2C | 4 | 3 | 24 | 3 | 2 | | | | | | |
| BMI | (2) | | | | | | | | | | | | | | | | | | |
| BNE | (2) | | | | | | | | | | | | | | | | | | |
| BPL | (2) | | | | | | | | | | | | | | | | | | |
| BRK | | 00 | 7 | 1 | | | | | | | | | | | | | | | |
| BVC | (2) | | | | | | | | | | | | | | | | | | |
| BVS | (2) | | | | | | | | | | | | | | | | | | |
| CLC | | 18 | 2 | 1 | | | | | | | | | | | | | | | |
| CLD | | D8 | 2 | 1 | | | | | | | | | | | | | | | |
| CLI | | 58 | 2 | 1 | | | | | | | | | | | | | | | |
| CLV | | B8 | 2 | 1 | | | | | | | | | | | | | | | |
| CMP | | | | | | | | CD | 4 | 3 | C5 | 3 | 2 | C9 | 2 | 2 | DD | 4 | 3 |
| CPX | | | | | | | | EC | 4 | 3 | E4 | 3 | 2 | E0 | 2 | 2 | | | |
| CPY | | | | | | | | CC | 4 | 3 | C4 | 3 | 2 | C0 | 2 | 2 | | | |
| DEC | | | | | | | | CE | 6 | 3 | C6 | 5 | 2 | | | | DE | 7 | 3 |
| DEX | | CA | 2 | 1 | | | | | | | | | | | | | | | |
| DEY | | 88 | 2 | 1 | | | | | | | | | | | | | | | |
| EOR | (1) | | | | | | | 4D | 4 | 3 | 45 | 3 | 2 | 49 | 2 | 2 | 5D | 4 | 3 |
| INC | | | | | | | | EE | 6 | 3 | E6 | 5 | 2 | | | | FE | 7 | 3 |
| INX | | E8 | 2 | 1 | | | | | | | | | | | | | | | |
| INY | | C8 | 2 | 1 | | | | | | | | | | | | | | | |
| JMP | | | | | | | | 4C | 3 | 3 | | | | | | | | | |
| JSR | | | | | | | | 20 | 6 | 3 | | | | | | | | | |
| LDA | (1) | | | | | | | AD | 4 | 3 | A5 | 3 | 2 | A9 | 2 | 2 | BD | 4 | 3 |
| LDX | (1) | | | | | | | AE | 4 | 3 | A6 | 3 | 2 | A2 | 2 | 2 | | | |
| LDY | (1) | | | | | | | AC | 4 | 3 | A4 | 3 | 2 | A0 | 2 | 2 | BC | 4 | 3 |
| LSR | | | | | 4A | 2 | 1 | 4E | 6 | 3 | 46 | 5 | 2 | | | | 5E | 7 | 3 |
| NOP | | EA | 2 | 1 | | | | | | | | | | | | | | | |
| ORA | | | | | | | | 0D | 4 | 3 | 05 | 3 | 2 | 09 | 2 | 2 | 1D | 4 | 3 |
| PHA | | 48 | 3 | 1 | | | | | | | | | | | | | | | |
| PHP | | 08 | 3 | 1 | | | | | | | | | | | | | | | |
| PLA | | 68 | 4 | 1 | | | | | | | | | | | | | | | |
| PLP | | 28 | 4 | 1 | | | | | | | | | | | | | | | |
| ROL | | | | | 2A | 2 | 1 | 2E | 6 | 3 | 26 | 5 | 2 | | | | 3E | 7 | 3 |
| ROR | | | | | 6A | 2 | 1 | 6E | 6 | 3 | 66 | 5 | 2 | | | | 7E | 7 | 3 |
| RTI | | 40 | 6 | 1 | | | | | | | | | | | | | | | |
| RTS | | 60 | 6 | 1 | | | | | | | | | | | | | | | |
| SBC | (1) | | | | | | | ED | 4 | 3 | E5 | 3 | 2 | E9 | 2 | 2 | FD | 4 | 3 |
| SEC | | 38 | 2 | 1 | | | | | | | | | | | | | | | |
| SED | | F8 | 2 | 1 | | | | | | | | | | | | | | | |
| SEI | | 78 | 2 | 1 | | | | | | | | | | | | | | | |
| STA | | | | | | | | 8D | 4 | 3 | 85 | 2 | | | | | 9D | 5 | 3 |
| STX | | | | | | | | 8E | 4 | 3 | 86 | 2 | | | | | | | |
| STY | | | | | | | | 8C | 4 | 3 | 84 | 2 | | | | | | | |
| TAX | | AA | 2 | 1 | | | | | | | | | | | | | | | |
| TAY | | A8 | 2 | 1 | | | | | | | | | | | | | | | |
| TSX | | BA | 2 | 1 | | | | | | | | | | | | | | | |
| TXA | | 8A | 2 | 1 | | | | | | | | | | | | | | | |
| TXS | | 9A | 2 | 1 | | | | | | | | | | | | | | | |
| TYA | | 98 | 2 | 1 | | | | | | | | | | | | | | | |

(1) 1 zu n dazuzählen, wenn Seitengrenze überschritten wird

| Abs Y | | | (IND. X) | | | (IND)Y | | | Z Seite X | | | Relativ | | | Indirekt | | | Z Seite Y | | | Prozessor Status Kodes | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OP | n | # | OP | n | # | OP | n | # | OP | n | # | OP | n | # | OP | n | # | OP | n | # | N | V | | B | D | I | Z | C |
| 79 | 4 | 3 | 61 | 6 | 2 | 71 | 5 | 2 | 75 | 4 | 2 | | | | | | | | | | ● | ● | | | | | ● | ● |
| 39 | 4 | 3 | 21 | 6 | 2 | 31 | 5 | 2 | 35 | 4 | 2 | | | | | | | | | | ● | | | | | | ● | |
| | | | | | | | | | 16 | 6 | 2 | | | | | | | | | | ● | | | | | | ● | ● |
| | | | | | | | | | | | | 90 | 2 | 2 | | | | | | | | | | | | | | |
| | | | | | | | | | | | | B0 | 2 | 2 | | | | | | | | | | | | | | |
| | | | | | | | | | | | | F0 | 2 | 2 | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | M7 | M6 | | | | | ● | |
| | | | | | | | | | | | | 30 | 2 | 2 | | | | | | | | | | | | | | |
| | | | | | | | | | | | | D0 | 2 | 2 | | | | | | | | | | | | | | |
| | | | | | | | | | | | | 10 | 2 | 2 | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | 1 | | | 1 | | |
| | | | | | | | | | | | | 50 | 2 | 2 | | | | | | | | | | | | | | |
| | | | | | | | | | | | | 70 | 2 | 2 | | | | | | | | | | | | | | 0 |
| | | | | | | | | | | | | | | | | | | | | | | | | 0 | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | | |
| D9 | 4 | 3 | C1 | 6 | 2 | D1 | 5 | 2 | D5 | 4 | 2 | | | | | | | | | | | 0 | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | ● | | | | | | ● | ● |
| | | | | | | | | | | | | | | | | | | | | | ● | | | | | | ● | ● |
| | | | | | | | | | | | | | | | | | | | | | ● | | | | | | ● | ● |
| | | | | | | | | | D6 | 46 | 2 | | | | | | | | | | ● | | | | | | ● | |
| | | | | | | | | | | | | | | | | | | | | | ● | | | | | | ● | |
| 59 | 4 | 3 | 41 | 6 | 2 | 51 | 5 | 2 | 55 | 4 | 2 | | | | | | | | | | ● | | | | | | ● | |
| | | | | | | | | | F6 | 6 | 2 | | | | | | | | | | ● | | | | | | ● | |

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | ● | | | | | | ● | |
| B9 | 4 | 3 | A1 | 6 | 2 | B1 | 5 | 2 | B5 | 4 | 2 | | | | 6C | 5 | 3 | | | | ● | | | | | | ● | |
| BE | 4 | 3 | | | | | | | B4 | 4 | 2 | | | | | | | B6 | 4 | 2 | ● | | | | | | ● | |
| | | | | | | | | | 56 | 6 | 2 | | | | | | | | | | ● | | | | | | ● | |
| 19 | 4 | 3 | 01 | 6 | 2 | 11 | 5 | 2 | 15 | 4 | 2 | | | | | | | | | | 0 | | | | | | ● | ● |
| | | | | | | | | | | | | | | | | | | | | | ● | | | | | | ● | |
| | | | | | | | | | | | | | | | | | | | | | ● | ● | ● | ● | ● | ● | ● | ● |
| | | | | | | | | | 36 | 6 | 2 | | | | | | | | | | ● | | | | | | ● | ● |
| | | | | | | | | | 76 | 6 | 2 | | | | | | | | | | ● | | | | | | ● | |
| F9 | 4 | 3 | E1 | 6 | 2 | F1 | 5 | 2 | F5 | 4 | 2 | | | | | | | | | | ● | ● | ● | ● | ● | ● | ● | ● |
| | | | | | | | | | | | | | | | | | | | | | ● | ● | | | | | ● | ● |
| | | | | | | | | | | | | | | | | | | | | | | | 1 | | | | | 1 |
| | | | | | | | | | | | | | | | | | | | | | | | | | | 1 | | |
| 99 | 5 | 3 | 81 | 6 | 2 | 91 | 6 | 2 | 95 | 4 | 2 | | | | | | | | | | | | | | | | | |
| | | | | | | | | | 94 | 4 | 2 | | | | | | | 96 | 4 | 2 | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | ● | | | | | | ● | |
| | | | | | | | | | | | | | | | | | | | | | ● | | | | | | ● | |
| | | | | | | | | | | | | | | | | | | | | | ● | | | | | | ● | |
| | | | | | | | | | | | | | | | | | | | | | ● | | | | | | ● | |

(2) 2 zu n dazuzählen, wenn in einer Seite gesprungen wird
3 zu n dazuzählen, wenn in eine andere Seite gesprungen wird

# Anhang C

**Bauanleitung zum Anschluß an den Apple II bzw. Commodore 64**

Für den Anbau an diese beiden Rechner muß das Spielbrett erweitert werden, da 2 VIAs 6522 auf dem SYM standardmäßig vorhanden sind, aber beim Apple und Commodore 64 in dieser Form fehlen.

Abb. C.1 zeigt den Gesamtschaltplan. Der umrandete Teil zeigt die zusätzliche Erweiterung. Links im Schaltbild sind die beiden verschiedenen Peripherieanschlüsse zu erkennen, an die das Spielbrett angeschlossen wird. Es handelt sich beim Apple um den Slot 3 und beim Commodore um den Expansion Port. Das Spielbrett kann dann wahlweise an den C64 oder Apple II angeschlossen werden.

**Gesamtstückliste:**

| | | |
|---|---|---|
| 15 | Widerstände | 330 ohm |
| 1 | Widerstand | 1000 ohm |
| 1 | Widerstand | 3300 ohm |
| 1 | Widerstand | 56 ohm |
| 15 | Leuchtdioden | |
| 2 | VIA's | 6522 |
| 3 | IC's | 7416 |
| 1 | IC | 74154 |
| 1 | IC | CD4050 |
| 1 | IC | 7400 |
| 2 | Experimentierplatinen 100*160cm Punktraster | |
| 1 | Lautsprecher | 8 ohm |
| 1 | Tastatur (4*4 Tasten) | |
| 1 | Adapterplatine zum Anschluß an den entsprechenden Peripherieanschluß | |
| 1 | Verbindungskabel vom Spielbrett zur Adapterplatine 20 polig | |

Beim Aufbau sind für die VIAs und ICs entsprechende Sockel zu empfehlen. Die Bauteile sind im Elektronikladen für ca. 120 DM zu haben.

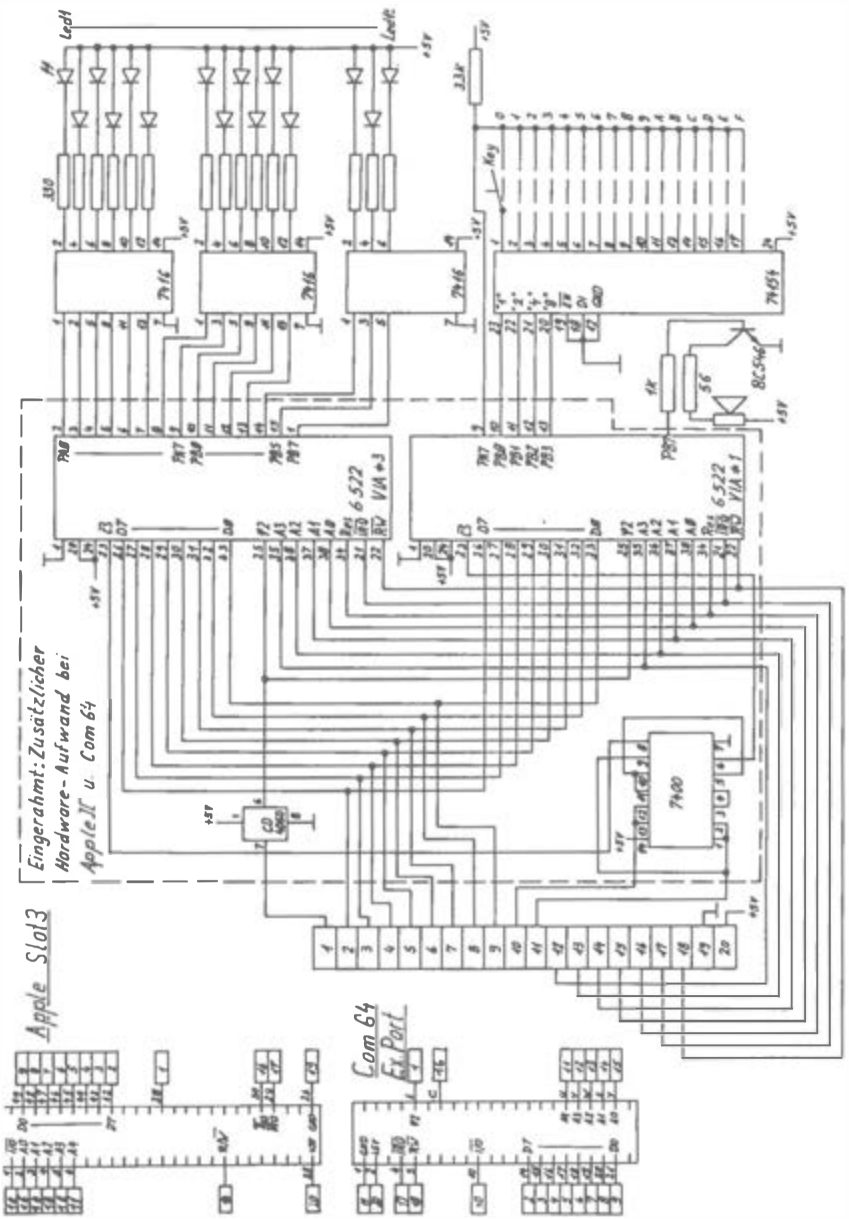



*Abb. C.1: Gesamtschaltplan*

**Softwaremäßige Änderungen**

Da die Adreßbelegungen der verschiedenen Computer nicht völlig übereinstimmen, müssen auch hier Anpassungen vorgenommen werden.

*1. Software-Anpassungen für den Apple II*

Da der Apple den Bereich, für den die Programme geschrieben sind (z.B. $200), für interne Funktionen braucht, müssen die Programme an eine andere Speicherstelle gebracht werden. Hier bietet sich z.B. der Bereich $8000 an, in den auch bekannte Assembler ihre Maschinenprogramme legen (BIG MAC).

Das Unterprogramm GETKEY (Seite 24), das in den meisten Programmen zur Tastaturabfrage benutzt wird, wurde von $100 nach 7FC0 verlegt. Entsprechend ist in allen Programmen GETKEY=$7FC0 zu setzen. Es bleibt dem Anwender überlassen, bei Bedarf andere Adressen im Rechner auszuwählen.

Die Portadressen von VIA#1 und VIA#3 ändern sich gemäß Abb. C.2. Die Interruptvektoren, die für die Programme ab Seite 184 gebraucht werden, ändern sich wie folgt:

| APPLE | SYM | |
|---|---|---|
| 03FE | A67E | IRQVECL |
| 03FF | A67F | IRQVECH |

Ein guter Assembler zur Eingabe der Maschinenprogramme ist der BIG MAC. Damit können die Programme eingegeben und gestartet werden. Wird mit dem APPLE-Monitor gearbeitet, so werden die Maschinenprogramme in bekannter Form:

```
*8000:A9 94 13 54 .. ..
```

eingegeben und anschließend mit

```
*8000G
```

gestartet.

Einige der Programme haben eine andere Startadresse, z. B. $210 beim SYM. Diese Programme werden dann entsprechend mit

```
*8010G
```

gestartet.

Das Unterprogramm ACCESS entfällt beim APPLE.

```
  VIA 3       VIA 1      6522 REGISTER

APPLE SYM   APPLE SYM

C300  AC00  C310  A000   ORB (PB0 TO PB7)
C301  AC01  C311  A001   ORA (PA0 TO PA7)
C302  AC02  C312  A002        DDR B
C303  AC03  C313  A003        DDR A
C304  AC04  C314  A004      T1L-L/T1C-L
C305  AC05  C315  A005        T1C-H
C306  AC06  C316  A006        T1L-L
C307  AC07  C317  A007        T1L-H
C308  AC08  C318  A008      T2L-L/T2C-L
C309  AC09  C319  A009        T2C-H
C30A  AC0A  C31A  A00A         SR
C30B  AC0B  C31B  A00B        ACR
C30C  AC0C  C31C  A00C  PCR(CA1,CA2,CB2,CB1)
C30D  AC0D  C31D  A00D        IFR
C30E  AC0E  C31E  A00E        IER
C30F  AC0F  C31F  A00F        ORA


INTERRUPT VEKTOR

APPLE  SYM

03FE   A67E     IRQVECL
03FF   A67F     IRQVECH
```

*Abb. C.2: Portadressen des Apple II*

```
:ASM
                1            ORG  $8000
                2    * SPIELAUTOMAT (KAP.6)
                3    *
                4    TDR1A   =    $C311
                5    TDR1B   =    $C310
                6    DDR1A   =    $C313
                7    DDR1B   =    $C312
                8    TDR3A   =    $C301
                9    TDR3B   =    $C300
                10   DDR3A   =    $C303
                11   DDR3B   =    $C302
                12   DURAT   =    $0
                13   SCHGRD  =    $1
                14   DNTST   =    $2
8000: 01 02 04
8003: 20 00 80
8006: 40 08     15   LITAB   HEX  01,02,04,20,00,80,40,08
8008: 01 02 03
800B: 06 09 08
800E: 07 04     16   TATAB   HEX  01,02,03,06,09,08,07,04
8010: A9 FF     17   START   LDA  #$FF
8012: 8D 13 C3  18           STA  DDR1A
8015: 8D 12 C3  19           STA  DDR1B
8018: 8D 02 C3  20           STA  DDR3B
8018: A9 08     21           LDA  #8
801D: 85 01     22           STA  SCHGRD
801F: 8D 03 C3  23           STA  DDR3A
8022: A0 00     24   NEUSP   LDY  #0
8024: A9 00     25   SCHLEIFE LDA  #0
```

*Abb. C.3: Beispielprogramm für den Apple II*

```
8026: 85 02    26            STA  DNTST
8028: 8D 10 C3 27            STA  TOR1B
8028: 98       28            TYA
802C: 29 07    29            AND  #7
802E: AA       30            TAX
802F: BD 00 80 31            LDA  LITAB,X
8032: 8D 11 C3 32            STA  TOR1A
8035: D0 05    33            BNE  CHECK
8037: A9 01    34            LDA  #1
8039: 8D 10 C3 35            STA  TOR1B
803C: BD 08 80 36   CHECK     LDA  TATAB,X
803F: 8D 00 C3 37            STA  TOR3B
8042: 2C 01 C3 38            BIT  TOR3A
8045: 30 04    39            BMI  DELAY
8047: A9 01    40   UNGLTG    LDA  #1
8049: 85 02    41            STA  DNTST
804B: A9 80    42   DELAY     LDA  #$80
804D: 85 00    43            STA  DURAT
804F: A5 01    44   DL1       LDA  SCHGRD
8051: 0A       45            ASL
8052: 0A       46            ASL
8053: AA       47            TAX
8054: 26 02    48   DL2       ROL  DNTST
8056: 66 02    49            ROR  DNTST
8058: CA       50            DEX
8059: D0 F9    51            BNE  DL2
805B: A5 02    52            LDA  DNTST
805D: D0 05    53            BNE  NICHT
805F: 2C 01 C3 54            BIT  TOR3A
8062: 10 19    55            BPL  TREFFER
8064: C6 00    56   NICHT     DEC  DURAT
8066: D0 E7    57            BNE  DL1
8068: C8       58            INY
8069: D0 B9    59            BNE  SCHLEIFE
806B: A6 01    60            LDX  SCHGRD
806D: E8       61            INX
806E: 8A       62            TXA
806F: C9 10    63            CMP  #16
8071: D0 02    64            BNE  OK
8073: A9 0F    65            LDA  #15
8075: 85 01    66   OK        STA  SCHGRD
8077: 20 92 80 67            JSR  WARTEN
807A: 4C 22 80 68            JMP  NEUSP
807D: 20 92 80 69   TREFFER   JSR  WARTEN
8080: C6 01    70            DEC  SCHGRD
8082: D0 9E    71            BNE  NEUSP
8084: A9 FF    72            LDA  #$FF
8086: 8D 11 C3 73            STA  TOR1A
8089: 8D 10 C3 74            STA  TOR1B
808C: 20 92 80 75            JSR  WARTEN
808F: 4C 10 80 76            JMP  START
8092: A0 FF    77   WARTEN    LDY  #$FF
8094: A2 FF    78   LP1       LDX  #$FF
8096: 66 00    79   LP2       ROR  DURAT
8098: 26 00    80            ROL  DURAT
809A: 66 00    81            ROR  DURAT
809C: 26 00    82            ROL  DURAT
809E: CA       83            DEX
809F: D0 F5    84            BNE  LP2
80A1: 88       85            DEY
80A2: D0 F0    86            BNE  LP1
80A4: 60       87            RTS

--END ASSEMBLY--

ERRORS: 0

165 BYTES
```

*Abb. C.3: Beispielprogramm für den Apple II (Fortsetzung)*

```
SYMBOL TABLE - ALPHABETICAL ORDER:

   CHECK   =$003C      DDR1A   =$C313      DDR1B   =$C312      DDR3A   =$C303
   DDR3B   =$C302      DELAY   =$004B      DL1     =$004F      DL2     =$0054
   DNTST   =$02        DURAT   =$00        LITAB   =$0000      LP1     =$0094
   LP2     =$0096      NEUSP   =$0022      NICHT   =$0064      OK      =$0075
   SCHGRD  =$01        SCHLEIFE=$0024      START   =$0010      TATAB   =$0008
   TOR1A   =$C311      TOR1B   =$C310      TOR3A   =$C301      TOR3B   =$C300
   TREFFER =$007D   ?  UNGLTG  =$0047      WARTEN  =$0092

SYMBOL TABLE - NUMERICAL ORDER:

   DURAT   =$00        SCHGRD  =$01        DNTST   =$02        LITAB   =$0000
   TATAB   =$0008      START   =$0010      NEUSP   =$0022      SCHLEIFE=$0024
   CHECK   =$003C   ?  UNGLTG  =$0047      DELAY   =$004B      DL1     =$004F
   DL2     =$0054      NICHT   =$0064      OK      =$0075      TREFFER =$007D
   WARTEN  =$0092      LP1     =$0094      LP2     =$0096      TOR3B   =$C300
   TOR3A   =$C301      DDR3B   =$C302      DDR3A   =$C303      TOR1B   =$C310
   TOR1A   =$C311      DDR1B   =$C312      DDR1A   =$C313
```

*Abb. C.3: Beispielprogramm für den Apple II (Fortsetzung)*

### *2. Software-Anpassungen für den Commodore VC 64*

Beim Commodore muß der Speicherbereich, in den die Programme geschrieben sind, ähnlich wie beim APPLE verändert werden. Hier bietet sich für die Maschinenprogramme ebenfalls der Bereich um $8000 an.

Da einige Programme die Adressen der Zeropage (Speicherbereich ($0–$255) benutzen, müssen diese Adressen innerhalb ($0–$255) so gewählt werden, daß sie mit dem Betriebssystem des Rechners nicht kollidieren. Es wurden die Adressen $D9-FF und 39-3B anstelle von $00-$41 gewählt. Zu beachten ist, daß diese geänderten Adressen der Zeropage auch wieder in der Zeropage liegen, um die dafür vorgesehene Adressierungsart zu realisieren.

Die Portadressen von VIA#1 und VIA#3 ändern sich gemäß Abb. C.4. Die Interruptvektoren, die ab Programm auf Seite 184 gebraucht werden, ändern sich wie folgt:

```
COM 64   SYM

0314     A67E    IRQVECL
0315     A67F    IRQVECH
```

Da der Commodore regelmäßig seine eigene Interrupt-Routine ab ($EA31) durchlaufen muß, setzt man am Schluß der eigenen Interrupt-Routine im Programm nicht den Befehl RTI, sondern JMP $EA31.

Das Unterprogramm ACCESS entfällt beim Commodore C64.

```
  VIA 3       VIA 1      6522 REGISTER

COM64 SYM   COM64 SYM

DF00  AC00  DF10  A000   ORB (PB0 TO PB7)
DF01  AC01  DF11  A001   ORA (PA0 TO PA7)
DF02  AC02  DF12  A002        DDR B
DF03  AC03  DF13  A003        DDR A
DF04  AC04  DF14  A004      T1L-L/T1C-L
DF05  AC05  DF15  A005        T1C-H
DF06  AC06  DF16  A006        T1L-L
DF07  AC07  DF17  A007        T1L-H
DF08  AC08  DF18  A008      T2L-L/T2C-L
DF09  AC09  DF19  A009        T2C-H
DF0A  AC0A  DF1A  A00A         SR
DF0B  AC0B  DF1B  A00B        ACR
DF0C  AC0C  DF1C  A00C  PCR(CA1,CA2,CB2,CB1)
DF0D  AC0D  DF1D  A00D        IFR
DF0E  AC0E  DF1E  A00E        IER
DF0F  AC0F  DF1F  A00F        ORA


INTERRUPT VEKTOR

COM64  SYM

0314   A67E     IRQVECL
0315   A67F     IRQVECH
```

*Abb. C.4: Portadressen des Commodore C 64*

```
:ASM
                  1             ORG  $8000
                  2    * SPIELAUTOMAT (KAP.6)
                  3    *
                  4    TDR1A    =    $DF11
                  5    TOR1B    =    $DF10
                  6    DDR1A    =    $DF13
                  7    DDR1B    =    $DF12
                  8    TOR3A    =    $DF01
                  9    TOR3B    =    $DF00
                 10    DDR3A    =    $DF03
                 11    DDR3B    =    $DF02
                 12    DURAT    =    $D9
                 13    SCHGRD   =    $DA
                 14    DNTST    =    $DB
8000: 01 02 04
8003: 20 00 00
8006: 40 08      15   LITAB    HEX  01,02,04,20,00,00,40,00
8008: 01 02 03
800B: 06 09 08
800E: 07 04      16   TATAB    HEX  01,02,03,06,09,08,07,04
8010: A9 FF      17   START    LDA  #$FF
8012: 8D 13 DF 18             STA  DDR1A
8015: 8D 12 DF 19             STA  DDR1B
8018: 8D 02 DF 20             STA  DDR3B
801B: A9 08      21             LDA  #8
801D: 85 DA      22             STA  SCHGRD
801F: 8D 03 DF 23             STA  DDR3A
8022: A0 00      24   NEUSP    LDY  #0
8024: A9 00      25   SCHLEIFE LDA  #0
```

*Abb. C.5: Beispielprogramm für den Commodore C 64*

```
8026: 85 DB    26            STA  DNTST
8028: BD 10 DF 27            STA  TOR1B
8028: 98       28            TYA
802C: 29 07    29            AND  #7
802E: AA       30            TAX
802F: BD 00 B0 31            LDA  LITAB,X
8032: BD 11 DF 32            STA  TOR1A
8035: D0 05    33            BNE  CHECK
8037: A9 01    34            LDA  #1
8039: BD 10 DF 35            STA  TOR1B
803C: BD 08 B0 36   CHECK    LDA  TATAB,X
803F: BD 00 DF 37            STA  TOR3B
8042: 2C 01 DF 38            BIT  TOR3A
8045: 30 04    39            BMI  DELAY
8047: A9 01    40   UNGLTG   LDA  #1
8049: 85 DB    41            STA  DNTST
804B: A9 B0    42   DELAY    LDA  #$B0
804D: 85 D9    43            STA  DURAT
804F: A5 DA    44   DL1      LDA  SCHGRD
8051: 0A       45            ASL
8052: 0A       46            ASL
8053: AA       47            TAX
8054: 26 DB    48   DL2      ROL  DNTST
8056: 66 DB    49            ROR  DNTST
8058: CA       50            DEX
8059: D0 F9    51            BNE  DL2
805B: A5 DB    52            LDA  DNTST
805D: D0 05    53            BNE  NICHT
805F: 2C 01 DF 54            BIT  TOR3A
8062: 10 19    55            BPL  TREFFER
8064: C6 D9    56   NICHT    DEC  DURAT
8066: D0 E7    57            BNE  DL1
8068: CB       58            INY
8069: D0 B9    59            BNE  SCHLEIFE
806B: A6 DA    60            LDX  SCHGRD
806D: EB       61            INX
806E: 8A       62            TXA
806F: C9 10    63            CMP  #16
8071: D0 02    64            BNE  OK
8073: A9 0F    65            LDA  #15
8075: 85 DA    66   OK       STA  SCHGRD
8077: 20 92 B0 67            JSR  WARTEN
807A: 4C 22 B0 68            JMP  NEUSP
807D: 20 92 B0 69   TREFFER  JSR  WARTEN
8080: C6 DA    70            DEC  SCHGRD
8082: D0 9E    71            BNE  NEUSP
8084: A9 FF    72            LDA  #$FF
8086: BD 11 DF 73            STA  TOR1A
8089: BD 10 DF 74            STA  TOR1B
808C: 20 92 B0 75            JSR  WARTEN
808F: 4C 10 B0 76            JMP  START
8092: A0 FF    77   WARTEN   LDY  #$FF
8094: A2 FF    78   LP1      LDX  #$FF
8096: 66 D9    79   LP2      ROR  DURAT
8098: 26 D9    80            ROL  DURAT
809A: 66 D9    81            ROR  DURAT
809C: 26 D9    82            ROL  DURAT
809E: CA       83            DEX
809F: D0 F5    84            BNE  LP2
80A1: 88       85            DEY
80A2: D0 F0    86            BNE  LP1
80A4: 60       87            RTS

--END ASSEMBLY--

ERRORS: 0

165 BYTES
```

*Abb. C.5: Beisspielprogramm für den Commodore C 64 (Fortsetzung)*

```
SYMBOL TABLE - ALPHABETICAL ORDER:

   CHECK   =$803C      DDR1A   =$DF13      DDR1B   =$DF12      DDR3A   =$DF03
   DDR3B   =$DF02      DELAY   =$804B      DL1     =$804F      DL2     =$8054
   DNTST   =$DB        DURAT   =$D9        L1TAB   =$8000      LP1     =$8094
   LP2     =$8096      NEUSP   =$8022      NICHT   =$8064      OK      =$8075
   SCHGRD  =$DA        SCHLEIFE=$8024      START   =$8010      TATAB   =$8008
   TOR1A   =$DF11      TOR1B   =$DF10      TOR3A   =$DF01      TOR3B   =$DF00
   TREFFER =$807D   ?  UNGLTG  =$8047      WARTEN  =$8092

SYMBOL TABLE - NUMERICAL ORDER:

   DURAT   =$D9        SCHGRD  =$DA        DNTST   =$DB        L1TAB   =$8000
   TATAB   =$8008      START   =$8010      NEUSP   =$8022      SCHLEIFE=$8024
   CHECK   =$803C   ?  UNGLTG  =$8047      DELAY   =$804B      DL1     =$804F
   DL2     =$8054      NICHT   =$8064      OK      =$8075      TREFFER =$807D
   WARTEN  =$8092      LP1     =$8094      LP2     =$8096      TOR3B   =$DF00
   TOR3A   =$DF01      DDR3B   =$DF02      DDR3A   =$DF03      TOR1B   =$DF10
   TOR1A   =$DF11      DDR1B   =$DF12      DDR1A   =$DF13
```

*Abb. C.5: Beispielprogramm für den Commodore C 64 (Fortsetzung)*

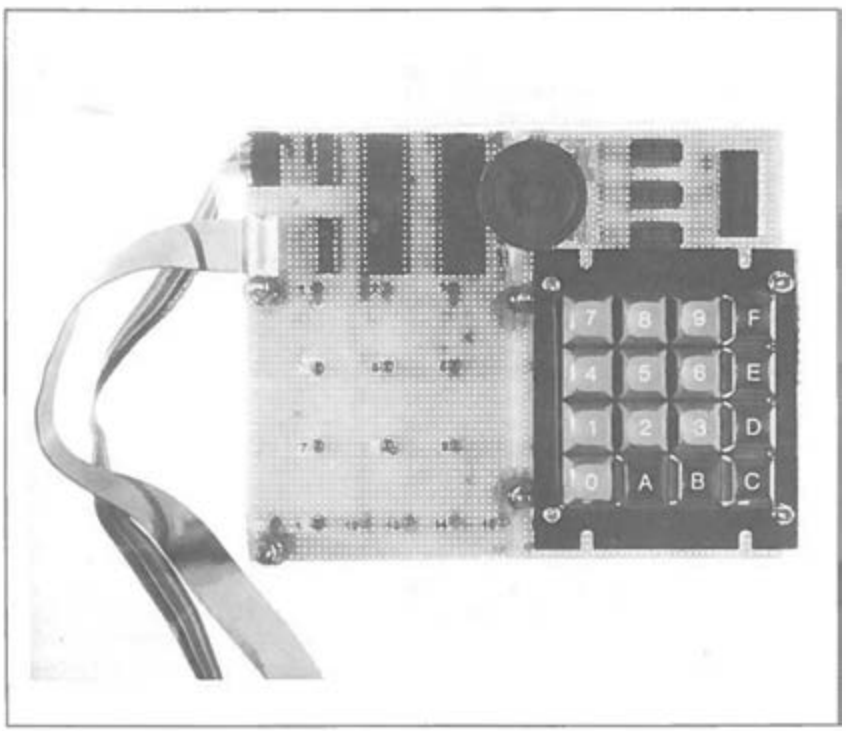



*Abb. C.6: Das erweiterte Spielbrett*

# Index

# Die SYBEX Bibliothek

**MEIN ERSTER COMPUTER** (3. überarbeitete Ausgabe)
**von Rodnay Zaks** – eine Einführung für alle, die den Kauf oder die Nutzung eines Mikrocomputers erwägen. 320 Seiten, 150 Abbildungen, Format DIN A5, Ref.Nr.: **3040** (1984)

**CP/M HANDBUCH MIT MP/M**
**von Rodnay Zaks** – ein umfassendes Lehr- und Nachschlagewerk für CP/M, das Standard-Betriebssystem für Mikrocomputer. 322 Seiten, 100 Abbildungen, Format DIN A5, Ref.Nr. **3002** (1981)

**MIKROPROZESSOR INTERFACE TECHNIKEN** (3. überarbeitete Ausgabe)
**von Rodnay Zaks/Austin Lesea** – Hardware- und Software-Verbindungstechniken samt Digital/Analog-Wandler, Peripheriegeräte, Standard-Busse und Fehlersuchtechniken. 425 Seiten, 400 Abbildungen, Format DIN A5, Ref.Nr.: **3012** (1982)

**PROGRAMMIERUNG DES 6502** (2. überarbeitete Ausgabe)
**von Rodnay Zaks** – Programmierung in Maschinensprache mit dem Mikroprozessor 6502, von den Grundkonzepten bis hin zu fortgeschrittenen Informationsstrukturen. 360 Seiten, 160 Abbildungen, Format DIN A5, Ref.Nr.: **3011** (1982)

**PROGRAMMIERUNG DES Z80**
**von Rodnay Zaks** – ein kompletter Lehrgang in der Programmierung des Z80 Mikroprozessors und eine gründliche Einführung in die Maschinensprache. 606 Seiten, 200 Abbildungen, Format DIN A5, Ref.Nr.: **3006** (1982)

**EINFÜHRUNG IN PASCAL UND UCSD/PASCAL**
**von Rodnay Zaks** – das Buch für jeden, der die Programmiersprache PASCAL lernen möchte. Vorkenntnisse in Computerprogrammierung werden nicht vorausgesetzt. Eine schrittweise Einführung mit vielen Übungen und Beispielen. 535 Seiten, 130 Abbildungen, Ref.Nr.: **3004** (1982)

**DAS PASCAL HANDBUCH**
**von Jacques Tiberghien** – ein Wörterbuch mit jeder Pascal-Anweisung und jedem Symbol, reservierten Wort, Bezeichner und Operator, für beinahe alle bekannten Pascal-Versionen. 476 Seiten, 270 Abbildungen, Format 23 x 18 cm, Ref.Nr.: **3005** (1982)

**PASCAL PROGRAMME – MATHEMATIK, STATISTIK, INFORMATIK**
**von Alan Miller** – eine Sammlung von 60 der wichtigsten wissenschaftlichen Algorithmen samt Programmauflistung und Musterdurchlauf. Ein wichtiges Hilfsmittel für Pascal-Benutzer mit technischen Anwendungen. 398 Seiten, 120 Abbildungen, Format 23 x 18 cm, Ref.Nr.: **3007** (1982)

**POCKET MIKROCOMPUTER LEXIKON**
– die schnelle Informations-Börse! 1300 Definitionen, Kurzformeln, technische Daten, Lieferanten-Adressen, ein englisch-deutsches und französisch-deutsches Wörterbuch. 176 Seiten, Format DIN A6, Ref.Nr. **3008** (1982)

**BASIC COMPUTER SPIELE/Band 1**
**herausgegeben von David H. Ahl** – die besten Mikrocomputerspiele aus der Zeitschrift „Creative Computing" in deutscher Fassung mit Probelauf und Programmlisting. 200 Seiten, 56 Abbildungen, Ref.Nr. **3009**

**BASIC COMPUTER SPIELE/Band 2**
**herausgegeben von David H. Ahl** – 84 weitere Mikrocomputerspiele aus „Creative Computing". Alle in Microsoft-BASIC geschrieben mit Listing und Probelauf. 220 Seiten, 61 Abbildungen, Ref.Nr.: **3010**

**MEIN ERSTES BASIC PROGRAMM**
**von Rodnay Zaks** – das Buch für Einsteiger! Viele farbige Illustrationen und leichtverständliche Diagramme bringen Spaß am Lernen. In wenigen Stunden schreiben Sie Ihr erstes nützliches Programm. 218 Seiten, illustriert, Ref.-Nr.: **3033** (November 1983)

**BASIC PROGRAMME – MATHEMATIK, STATISTIK, INFORMATIK**
**von Alan Miller** – eine Bibliothek von Programmen zu den wichtigsten Problemlösungen mit numerischen Verfahren, alle in BASIC geschrieben, mit Musterlauf und Programmlisting. 352 Seiten, 120 Abbildungen, Ref.Nr.: **3015** (1983)

**PLANEN UND ENTSCHEIDEN MIT BASIC**
**von X. T. Bui** – eine Sammlung von interaktiven, kommerziell-orientierten BASIC-Programmen für Management- und Planungsentscheidungen. 200 Seiten, 53 Abbildungen, Ref.-Nr.: **3025** (1983)

**BASIC FÜR DEN KAUFMANN**
**von D. Hergert** – das BASIC-Buch für Studenten und Praktiker im kaufmännischen Bereich. Enthält Anwendungsbeispiele für Verkaufs- und Finanzberichte, Grafiken, Abschreibungen u.v.m. ca. 240 Seiten, 76 Abbildungen, Ref.-Nr.: **3026** (November 1983)

**PLANEN, KALKULIEREN, KONTROLLIEREN MIT BASIC-TASCHENRECHNERN**
**von P. Ickenroth** – präsentiert eine Reihe von direkt anwendbaren BASIC-Programmen für zahlreiche kaufmännische Berechnungen mit Ihrem BASIC-Taschenrechner. 141 Seiten, 45 Abbildungen, Ref.-Nr.: **3032** (1983)

**EINFÜHRUNG IN DIE TEXTVERARBEITUNG**
**von Hal Glatzer** – woraus eine Textverarbeitungsanlage besteht, wie man sie nutzen kann und wozu sie fähig ist. Beispiele verschiedener Anwendungen und Kriterien für den Kauf eines Systems. 246 Seiten, 67 Abbildungen, Ref.Nr. **3018** (1983)

**EINFÜHRUNG IN WORDSTAR**
**von Arthur Naiman** – eine klar gegliederte Einführung, die aufzeigt, wie das Textbearbeitungsprogramm WORDSTAR funktioniert, was man damit tun kann und wie es eingesetzt wird. 238 Seiten, 30 Abbildungen, Ref.Nr.: **3019** (1983)

**ERFOLG MIT VisiCalc**
**von D. Hergert** – umfassende Einführung in VisiCalc und seine Anwendung. Zeigt Ihnen u. a.: Aufstellung eines Verteilungsbogens, Benutzung von VisiCalc-Formeln, Verwendung der DIF-Datei-Funktion. Ca. 224 Seiten, 58 Abbildungen, Ref.-Nr.: **3030** (Herbst 1983)

**6502 ANWENDUNGEN**
**von Rodnay Zaks** – das Eingabe-/Ausgabe-Buch für Ihren 6502-Microprozessor. Stellt die meistgenutzten Programme und die dafür notwendigen Hardware-Komponenten vor. 282 Seiten, 205 Abbildungen, Ref.Nr.: **3014** (1983)

**BASIC ÜBUNGEN FÜR DEN APPLE**
**von J.-P. Lamoitier** – das Buch für APPLE-Nutzer, die einen schnellen Zugang zur Programmierung in BASIC suchen. Abgestufte Übungen mit zunehmendem Schwierigkeitsgrad. 252 Seiten, 185 Abbildungen, Ref.Nr.: **3016** (1983)

**PROGRAMME FÜR MEINEN APPLE II**
**von S. R. Trost** – enthält eine Reihe von lauffähigen Programmen samt Listing und Beispiellauf. Hilft Ihnen, viele neue Anwendungen für Ihren APPLE II zu entdecken und erfolgreich einzusetzen. 192 Seiten, 192 Abbildungen, Ref.-Nr.: **3029** (1983)

**BASIC ÜBUNGEN FÜR DEN IBM PERSONAL COMPUTER**
**von J.-P. Lamoitier** – vermittelt Ihnen BASIC durch praktische und umfassende Übungen anhand von realistischen Programmen: Datenverarbeitung, Statistik, kommerzielle Programme, Spiele u.v.m. 256 Seiten, 192 Abbildungen, Ref.-Nr.: **3023** (1983)

**PROGRAMMSAMMLUNG ZUM IBM PERSONAL COMPUTER**
**von S. R. Trost** – mehr als 65 getestete, direkt einzugebende Anwenderprogramme, die eine weite Palette von kaufmännischen, persönlichen und schulischen Anwendungen abdecken. 192 Seiten, 158 Abbildungen, Ref.-Nr.: **3024** (1983)

**CHIP UND SYSTEM: Einführung in die Mikroprozessoren-Technik**
**von Rodnay Zaks** – eine sehr gut lesbare Einführung in die faszinierende Welt der Computer, vom Microprozessor bis hin zum vollständigen System. Ca. 560 Seiten, 325 Abbildungen, Ref.Nr.: **3017** (Erscheint Herbst 1983)

**VORSICHT! Computer brauchen Pflege**
**von Rodnay Zaks** – das Buch, das Ihnen die Handhabung eines Computersystems erklärt – vor allem, was Sie damit nicht machen sollten. Allgemeingültige Regeln für die pflegliche Behandlung Ihres Systems. 240 Seiten, 96 Abbildungen, Ref.Nr.: **3013** (1983)

**MEIN SINCLAIR ZX81**
**von D. Hergert** – eine gut lesbare Einführung in diesen Einplatincomputer und dessen Programmierung in BASIC. 173 Seiten, 45 Abbildungen, Ref: **3021** (1983)

**SINCLAIR ZX81 BASIC HANDBUCH**
**von D. Hergert** – vermittelt Ihnen das vollständige BASIC-Vokabular anhand von praktischen Beispielen, macht Sie zum Programmierer Ihres ZX81. 181 Seiten, 120 Abbildungen, Ref.-Nr.: **3028** (1983)

**SINCLAIR ZX SPECTRUM Programme zum Lernen und Spielen**
**von T. Hartnell** – ein Buch zur praktischen Anwendung. Grundzüge des Programmierens aus dem kaufmännischen Bereich sowie Spiele, Lehr- und Lernprogramme in BASIC. 224 Seiten, 120 Abbildungen, Ref. Nr. **3022** (1983)

**SINCLAIR ZX SPECTRUM BASIC HANDBUCH**
**von D. Hergert** – eine wichtige Hilfe für jeden SPECTRUM-Anwender. Gibt eine Übersicht aller BASIC-Begriffe, die auf diesem Rechner verwendet werden können, und erläutert sie ausführlich anhand von Beispielen. Ca. 224 Seiten, ca. 150 Abbildungen, Ref.-Nr.: **3027** (Dezember 1983)

# The SYBEX Library*

**Mikrocomputer-Bücher in englischer Sprache von SYBEX; lieferbar ab-Lager Düsseldorf*

**INTERNATIONAL MICROCOMPUTER DICTIONARY**
140 pp., Ref. 0067
All the definitions and acronyms of microcomputer jargon defined in a handy pocket-size edition. Includes translations of the most popular terms into ten languages.

**INTRODUCTION TO WORD PROCESSING**
**by Hal Glatzer** 216 pp., 140 illustr., Ref 0076
Explains in plain language what a word processor can do, how it improves productivity, how to use a word processor and how to buy one wisely.

**EXECUTIVE PLANNING WITH BASIC**
**by X. T. Bui** 192 pp., 19 illustr., Ref. 0083
An important collection of business management decision models in BASIC, including Inventory Management (EOQ), Critical Path Analysis and PERT, Financial Ratio Analysis, Portfolio Management, and much more.

**FIFTY BASIC EXERCISES**
**by J. P. Lamoitier** 236 pp., 90 illustr., Ref. 0056
Teaches BASIC by actual practice, using graduated exercises drawn from everyday applications. All programs written in Microsoft BASIC.

**BASIC EXERCISES FOR THE APPLE**
**by J. P. Lamoitier** 230 pp., 90 illustr., Ref. 0084
This book is an Apple version of *Fifty BASIC Exercises.*

**BASIC EXERCISES FOR THE IBM PERSONAL COMPUTER**
**by J. P. Lamoitier** 232 pp., 90 illustr., Ref. 0088
This book is an IBM version of *Fifty BASIC exercises.*

**INSIDE BASIC GAMES**
**by Richard Mateosian** 352 pp., 120 illustr., Ref. 0055
Teaches interactive BASIC programming through games. Games are written in Microsoft BASIC and can run on the TRS-80, Apple II and PET/CBM.

**THE PASCAL HANDBOOK**
**by Jacques Tiberghien** 492 pp., 270 illustr., Ref. 0053
A dictionary of the Pascal language, defining every reserved word, operator, procedure and function found in all major versions of Pascal.

**INTRODUCTION TO PASCAL (Including UCSD Pascal)**
**by Rodnay Zaks** 422 pp., 130 illustr., Ref 0066
A step-by-step introduction for anyone wanting to learn the Pascal language. Describes UCSD and Standard Pascals. No technical background is assumed.

**BASIC FOR BUSINESS**
**by Douglas Hergert** 250 pp., 15 illustr., Ref. 0080
A logically organized, no-nonsense introduction to BASIC programming for business applications. Includes many fully-explained accounting programs, and shows you how to write them.

**APPLE PASCAL GAMES**
**by Douglas Hergert and Joseph T. Kalash** 376 pp., 40 illustr., Ref. 0074
A collection of the most popular computer games in Pascal, challenging the reader not only to play but to investigate how games are implemented on the computer.

**CELESTIAL BASIC: Astronomy on Your Computer**
**by Eric Burgess** 228 pp., 65 illustr., Ref. 0087
A collection of BASIC programs that rapidly complete the chores of typical astronomical computations. It's like having a planetarium in your own home! Displays apparent movement of stars, planets and meteor showers.

**PASCAL PROGRAMS FOR SCIENTISTS AND ENGINEERS**
**by Alan R. Miller** 378 pp., 120 illustr., Ref. 0058
A comprehensive collection of frequently used algorithms for scientific and technical applications, programmed in Pascal. Includes such programs as curvefitting, integrals and statistical techniques.

**BASIC PROGRAMS FOR SCIENTISTS AND ENGINEERS**
**by Alan R. Miller** 326 pp., 120 illustr., Ref. 0073
This second book in the "Programs for Scientists and Engineers" series provides a library of problem-solving programs while developing proficiency in BASIC.

**FORTRAN PROGRAMS FOR SCIENTISTS AND ENGINEERS**
**by Alan R. Miller** 320 pp., 120 illustr., Ref. 0082
Third in the "Programs for Scientists and Engineers" series. Specific scientific and engineering application programs written in FORTRAN.

**PROGRAMMING THE 6809**
**by Rodnay Zaks and William Labiak** 520 pp., 150 illustr., Ref. 0078
This book explains how to program the 6809 in assembly language. No prior programming knowledge required.

**PROGRAMMING THE 6502**
**by Rodnay Zaks** 388 pp., 160 illustr., Ref. 0046
Assembly language programming for the 6502, from basic concepts to advanced data structures.

**6502 APPLICATIONS BOOK**
**by Rodnay Zaks** 286 pp., 200 illustr., Ref. 0015
Real-life application techniques: the input/output book for the 6502.

**ADVANCED 6502 PROGRAMMING**
**by Rodnay Zaks** 292 pp., 140 illustr., Ref. 0089
Third in the 6502 series. Teaches more advanced programming techniques, using games as a framework for learning.

**PROGRAMMING THE Z80**
**by Rodnay Zaks** 626 pp., 200 illustr., Ref. 0069
A complete course in programming the Z80 microprocessor and a thorough introduction to assembly laguage.

**PROGRAMMING THE Z8000**
**by Richard Mateosian** 300 pp., 124 illustr., Ref. 0032
How to program the Z8000 16-bit microprocessor. Includes a description of the architecture and function of the Z8000 and its family of support chips.

**THE CP/M® HANDBOOK (with MP/M™)**
**by Rodnay Zaks** 324 pp., 100 illustr., Ref. 0048
An indispensable reference and guide to CP/M – the most widely-used operating system for small computers.

**MASTERING CP/M®**
**by Alan R. Miller** 320 pp., Ref. 0068
For advanced CP/M users or systems programmers who want maximum use of the CP/M operating system ... takes up where our *CP/M Handbook* leaves off.

**INTRODUCTION TO THE UCSD p-SYSTEM™**
**by Charles W. Grant and Jon Butah** 250 pp., 10 illustr., Ref. 0061
A simple, clear introduction to the UCSD Pascal Operating System; for beginners through experienced programmers.

**A MICROPROGRAMMED APL IMPLEMENTATION**
**by Rodnay Zaks** 350 pp., Ref. 0005
An expert-level text presenting the complete conceptual analysis and desing of an APL interpreter, and actual listing of the microcode.

**THE APPLE CONNECTION**
**by James W. Coffron** 228 pp., 120 illustr., Ref. 0085
Teaches elementary interfacing and BASIC programming of the Apple for connection to external devices and household appliances.

**MICROPROCESSOR INTERFACING TECHNIQUES**
**by Rodnay Zaks and Austin Lesea** 458 pp., 400 illustr., Ref. 0029
Complete hardware and software interconnect techniques, including D to A conversion, peripherals, standard buses and troubleshooting.

# FORDERN SIE EIN GESAMTVERZEICHNIS UNSERER VERLAGSPRODUKTION AN:


SYBEX-VERLAG GmbH
Vogelsanger Weg 111
4000 Düsseldorf 30
Tel.: (02 11) 62 64 41
Telex: 8588163

SYBEX
6–8, Impasse du Curé
75018 Paris
Tel.: 1/203-95-95
Telex: 211.801 f

SYBEX INC.
2344 Sixth Street
Berkeley, CA 94710, USA
Tel.: (415) 848-8233
Telex: 336311

# FORTGESCHRITTENE 6502 PROGRAMMIERUNG

**Lernen Sie, wie man schwierige Probleme mit dem 6502 lösen kann!**

FORTGESCHRITTENE 6502-PROGRAMMIERUNG lehrt Sie, wie Sie vollständige Lösungen entwickeln, vom Entwurf des Algorithmus und der Datenstruktur bis hin zur kompletten Programmorganisation. Mit diesem Buch lernen Sie die praktischen Unterschiede alternativer Programmierungstechniken kennen.

Bald können Sie

- den geeigneten Algorithmus auswählen und ihn schrittweise definieren,
- eine leistungsfähige Datenstruktur entwickeln und diese bewerten,
- Echtzeitlösungen realisieren,
- wirksames Eingabe-/Ausgabe-Management samt Interrupts erstellen,
- die Kodierung und Registerzuweisung optimieren, u.v.m.

**Über den Autor:**

Dr. Rodnay Zaks hat weltweit Kurse in Programmierung und über Mikroprozessoren gehalten. Er erhielt seinen Ph.D. in Computer Science von der University of California, Berkeley. Er entwickelte eine mikroprogrammierte APL-Implementation und arbeitete mehrere Jahre in Silicon Valley, wo er Vorreiter in der Entwicklung industrieller Anwendungen von Mikroprozessoren war. Er ist Autor mehrer Bestseller zum Thema Mikrocomputer, und seine Bücher sind international in mehr als zehn Sprachen zu Standardwerken geworden.

ISBN 3-88745-047-7

SYBEX