Kosmos

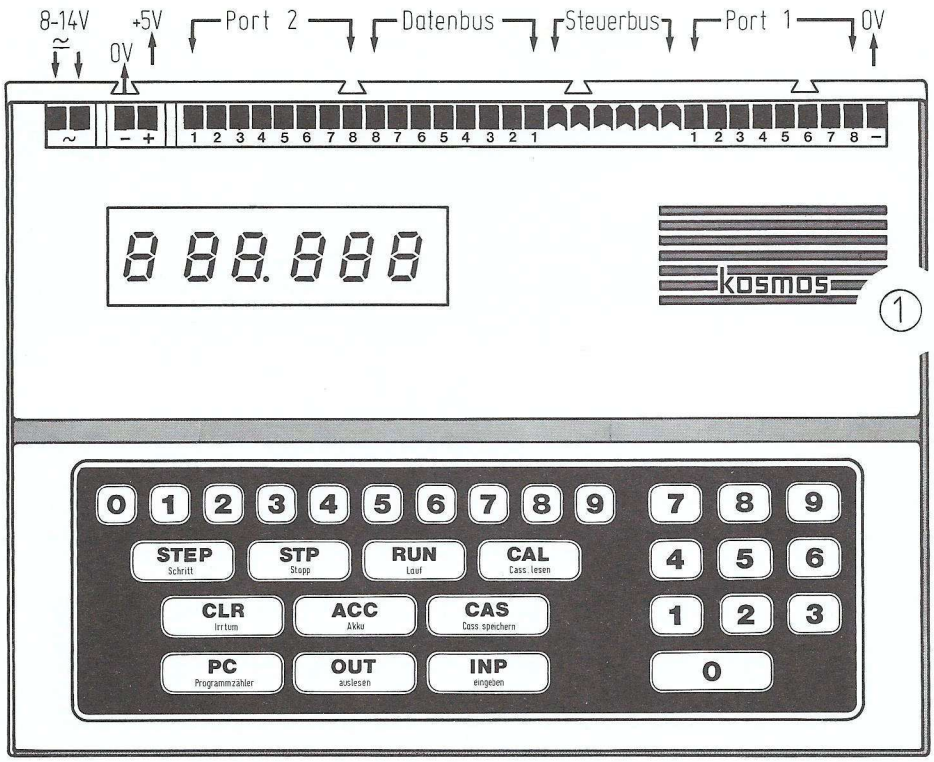
Computer-Praxis

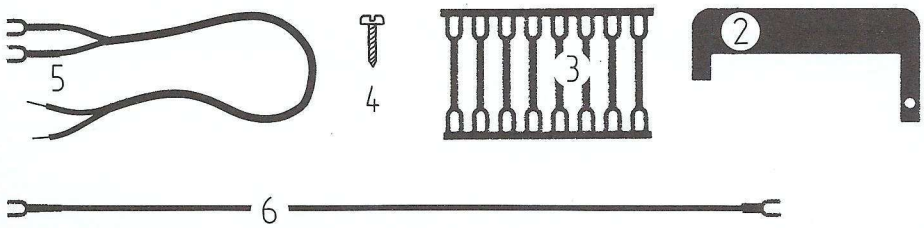
CP1

Der spielerische Einstieg in ein Gebiet, dem die Zukunft gehört. Geeignet für Schule und Fortbildung.

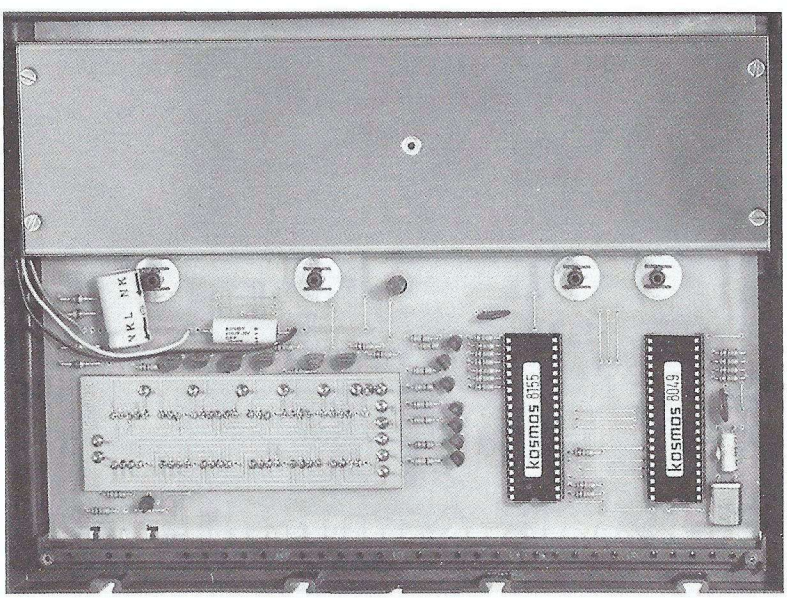
Mit dem Computer auf Du und Du – mit dem KOSMOS-Computer ein faszinierendes, großes Hobby beginnen.

kosmos®





1. Computer 61-2000.6
2. Masseschiene 60-1211.3
3. Acht Kontaktbügel 60-1214.3
4. Zweiundzwanzig Zylinderblechschrauben 60-2012.8
5. Doppellitze } als Bündel
6. Zehn Anschlußlitzen } 60-0060.6



kosmos®

Computer-Praxis

Das universelle Mikroprozessor-System

Programmieren und experimentieren, spielend lernen, wie ein Computer funktioniert

Eine leicht verständliche Einführung in die faszinierende Welt der Computer

Franckh'sche Verlagshandlung Stuttgart

1. Auflage
Franckh'sche Verlagshandlung, W. Keller & Co.,
Stuttgart / 1983

Textillustrationen: J. Nullmeyer, H. Keller, G. Bosch
Technische Bildgestaltung und Konstruktion: KOSMOS Entwicklungslabor
Printed in Germany / Imprimé en Allemagne / L 65 H hc
Gesamtherstellung: Konrad Triltsch, Graphischer Betrieb, Würzburg

KOSMOS Computer-Praxis

Zum Geleit

Unser zukünftiges Zusammenleben und Zusammenarbeiten wird in einem Ausmaß von Computern bestimmt, wie wir es uns vorläufig noch gar nicht vorstellen können:

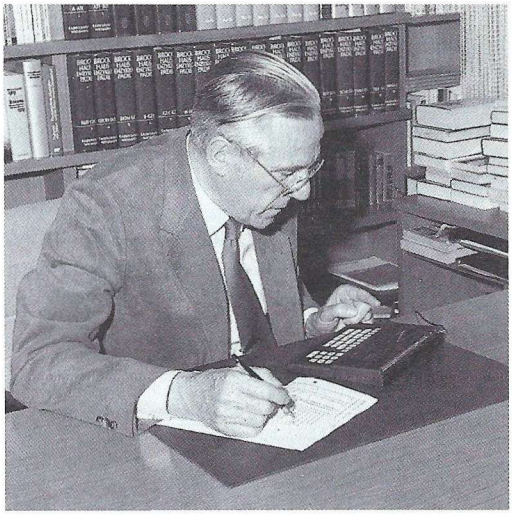
+ In Fabriken arbeiten Roboter mit Mikrocomputersteuerung (CAM), konstruiert und geprüft wird computerunterstützt (CAD und CAC),
+ Banken werden ihre Kundengeschäfte vielfach automatisch über Bildschirmarbeitsplätze und Geldausgabe-Automaten abwickeln,
+ beim elektronischen Markt werden Waren auf dem Bildschirm angeboten und durch elektrische Signale bestellt und evtl. auch bezahlt,
+ bei der telekommunikativen Heimarbeit werden Bürokräfte, Programmierer, Konstrukteure, Lehrkräfte usw. ihre Arbeit in der häuslichen Wohnung, vor einem computergesteuerten Bildschirm erledigen.

Wer in dieser voraussehbaren Welt mit dem Computer nicht geschickt umgehen kann, der ist schlecht daran – vergleichbar einem Analphabeten in unserer Zeit.

Berufliche Erfolge werden wesentlich davon abhängen, ob man den Computer zu nutzen versteht. Dies gilt nicht nur für den Informatiker, der hauptberuflich mit dem Computer arbeitet, sondern eher noch mehr für die vielen anderen Berufe, für welche der Computer ein wichtiges – wenn nicht das wichtigste – Werkzeug sein wird: Vom Ingenieur bis zum Finanzbeamten, vom Mediziner bis zum Soziologen.

In den Gebrauch des Computers müssen sich viele mühsam einarbeiten – an vielen Schulen und Hochschulen werden ja derartige Vorlesungen und Praktika angeboten. Je früher sich einer in dieser zunächst ungewohnten Computer-Welt zurechtfindet, desto leichter gelingt ihm dies meist.

Da ist es eine wertvolle Hilfe, daß die KOSMOS-Verlagsgruppe neuerdings einen KOSMOS-Computer anbietet, mit dessen Hilfe schon Jugendliche frühzeitig und spielerisch die Schwelle in die Denkwelt des Computers überschreiten können.



Prof. Steinbuch bei der Arbeit am KOSMOS-Computer.

Nicht, daß hier ein oberflächlicher Firlefanz geboten würde – nein, hier werden solide Grundlagen gelegt: Systemanalyse, Programm, Betriebssystem, Compiler, Sprungbefehl, Boolesche Operationen usw.

Die Methoden werden zur Lösung anregender Probleme benutzt: Digital-Uhr, Reaktionstester, Hotelreservierung, Heizungssteuerung, Nim-Spiel, Herzschlagmesser, Steuerung für die Modell-Eisenbahn usw. Darüber hinaus werden Anregungen zur selbständigen Weiterentwicklung gegeben.

Ich bin überzeugt davon, daß Jugendliche, welche mit dem KOSMOS-Computer gearbeitet haben, anderen beim Start in das zukünftige Berufsleben ein gutes Stück voraus sind.

Dr.-Ing. Karl Steinbuch
ehem. ord. Professor für
Nachrichtenverarbeitung
an der Universität Karlsruhe (TH)

Erster Teil
Der Computer und seine Arbeitsweise

1.0 Einleitung

Mikroelektronik und Computertechnik haben eine technische Revolution ungeahnten Ausmaßes eingeleitet, die im Begriff ist, alle Bereiche unseres Lebens zu durchdringen und gewaltige Veränderungen herbeizuführen. Einen umfassenden Einblick in jene neuen technischen Gebiete jedermann zu ermöglichen, ist das Ziel, das sich der Verlag mit der Herausgabe der „Computer-Praxis" gesetzt hat.

Bei der Beschäftigung mit dieser Ausrüstung lernt man nicht nur, wie der Computer durch „Programmieren" dazu gebracht wird, genau das zu tun, was wir wollen, sondern man erfährt auch, aus welchen Funktionseinheiten eine Datenverarbeitungsanlage besteht und wie sie arbeitet. Was „hinter den Tasten" vorgeht, bleibt daher kein Geheimnis, und die Bewältigung vieler reizvoller Probleme und Aufgaben schafft die Voraussetzung für den selbstverständlichen Umgang mit der Technik von morgen.

Nicht nur der Nutzen für Beruf und Leben veranlaßt allerdings heute zahlreiche Menschen, sich mit Computern auch privat zu befassen; es ist besonders die große Faszination, selbst die in einem Gerät schlummernde „maschinelle Intelligenz" zum Leben erwecken zu können und sich an der Lösung immer neuer Aufgaben und Probleme zu erfreuen, die das Computer-Hobby so beliebt macht. Der Kreativität sind hierbei keine Grenzen gesetzt.

Diese Computerausrüstung wurde mit der Absicht entwickelt, jedermann den Einstieg in die Computertechnik möglichst leicht zu machen und Barrieren fortzuräumen, die das Verständnis erschweren könnten. Ferner sollten alle Fähigkeiten und Möglichkeiten, die in einem Mikrocomputer stecken, mit Hilfe unseres kleinen Gerätes umfassend dargestellt werden. Man muß erfahren können, wie ein Computer speichert, verwaltet, rechnet, überwacht, sortiert, mißt, spielt und insbesondere, wie er zur Steuerung oder Regelung anderer Geräte, Anlagen oder Schaltungen eingesetzt werden kann. Wegen dieser Zielsetzung (leichte Durchschaubarkeit und Vielseitigkeit) unterscheidet sich der KOSMOS-Computer grundsätzlich von allen Heim-, Personal- und Bürocomputern, die für bestimmte praktische Anwendungszwecke konzipiert sind. Kommt es bei solchen Computern etwa auf Speicherkapazität oder auf Rechengeschwindigkeit an, so spielen dererlei quantitative Eigenschaften bei einem Lern- und Experimentiercomputer keine entscheidende Rolle. Mit ihm muß man die Computertechnik transparent machen können, und seine Stärken liegen daher auf ganz anderem Gebiet. Einer seiner Vorzüge besteht z.B. darin, daß man ihn als Steuergerät für elektrische und elektronische Schaltungen verwenden kann, wie sie sich etwa mit KOSMOS-Elektronikkästen bauen lassen. In Kombination mit diesen Kästen eröffnet sich ein weites Feld für weitere Experimente, die mit der Automation vertraut machen.

Für das Verständnis der Zusammenhänge ist es nützlich, dieses Buch systematisch durchzuarbeiten, ohne einzelne Kapitel zu überspringen. Natürlich lassen sich – sobald man mit der Bedienung vertraut ist – Programme eintippen und starten, auch wenn man noch keine Ahnung von der Bedeutung der verwendeten Befehle oder von den Vorgängen im Computer besitzt. Will man jedoch selbst Programme ausarbeiten, dann erwirbt man das hierzu erforderliche Wissen am besten dadurch, daß man der Reihe nach vorgeht.

Wir wünschen Freude und Erfolg beim Experimentieren!

1.1 Die Stromversorgung

Wir wollen gleich zur Sache kommen: ohne Strom geht gar nichts. Bevor der Computer arbeiten kann, müssen wir ihn an das Stromnetz anschlie-

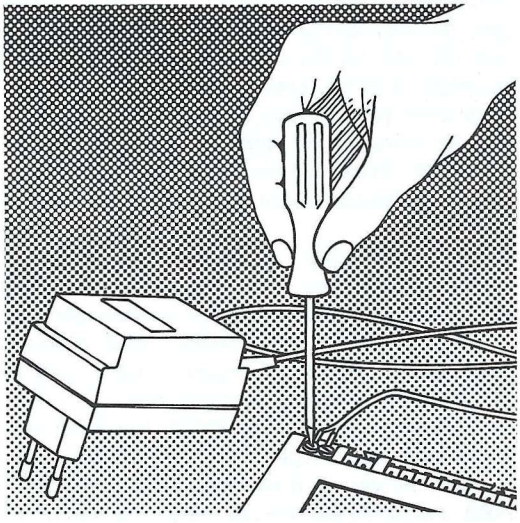


Bild 1

ßen. Aber keinesfalls direkt. Der Computer benötigt nur eine sehr kleine Spannung: mindestens 8 V und höchstens 14 V. Weniger reicht ihm nicht – wesentlich mehr könnte schädlich sein. Zur richtigen Spannung verhilft man ihm, indem man an den zwei linken Klemmen der Anschlußleiste (Bild 1) einen geeigneten Trafo anschließt. Schrauben dazu sind beigegeben. Entweder nimmt man das KOSMOS Computer-Netzanschlußgerät oder einen Modellbahn- bzw. Experimentiertrafo (Gleich- oder Wechselstrom). Bei der Verwendung eines Modellbahntrafos benutzt man zweckmäßigerweise die Anschlüsse für den Fahrstrom und stellt den Fahrt-

Bild 2

regler zunächst einmal auf mittlere Geschwindigkeit. Jetzt wird der Netzstecker in die Steckdose gesteckt, und die Anzeige leuchtet auf. Ist links ein leuchtender Punkt dicht bei dem „P" zu sehen (Bild 2), liegt zuviel Spannung an, und der Fahrtregler des Modellbahntrafos sollte zurückgedreht werden, bis das Pünktchen ausgeht. Bei Verwendung eines Experimentiertrafos ist die nächstkleinere Spannung zu wählen.

1.2 Ein- und ausschalten

Der Netzstecker ist unser Ein- und Ausschalter! Am Gerät selbst gibt es deshalb keinen, und das hat seinen guten Grund. Man kann nicht aus Versehen das Gerät ausschalten. Das wäre nämlich katastrophal, denn in dem Moment, wo die Stromversorgung unterbrochen wird, sind alle in den Computer eingegebenen Informationen gelöscht. Damit dies nicht unbeabsichtigt passiert, wurde kein Schalter vorgesehen.
Hier noch ein Tip: die stromzuführenden Kabel sollten fest angeschraubt werden.
Im Anzeigenfeld erscheint immer, wenn Sie das Gerät einschalten (Netzstecker in die Steckdose) die Anzeige.

Es schadet Ihrem Computer übrigens nicht, wenn er über längere Zeit eingeschaltet bleibt. Die Leistungsaufnahme ist mit 4–6 Watt nur gering.
P ist der Kennbuchstabe für „Programmzähler". Die Ziffern signalisieren, daß der Programmzähler auf Null steht. Das ist ja auch logisch, denn schließlich fangen wir erst an. Der Computer wartet auf Ihre Kommandos.

1.3 Schritt für Schritt in die Computertechnik

Natürlich ist es für Sie verlockend, gleich einmal auf die weißen Tasten zu drücken, um zu sehen, was geschieht. Tun Sie es ruhig, aber erwarten Sie bitte kein sinnvolles Ergebnis. Ein Computer kann nämlich zunächst einmal gar nichts. Man muß ihm erst beibringen, was er überhaupt machen soll. Ob er uns etwa als Spielpartner oder als Stoppuhr, als Steuergerät für die Modelleisenbahn oder als elektronischer Würfel dient, hängt allein von den Befehlen, den unterschiedlichen Programmen ab, die Sie in sein „Gedächtnis" hineinbringen können. Diese Vielseitigkeit ist ein entscheidender Vorzug des Computers. Wie Sie es anfangen müssen, damit er seine erstaunlichen Fähigkeiten entfaltet, erfahren Sie jetzt Schritt für Schritt. Sehen Sie sich zunächst einmal die äußeren Teile des Computers genauer an, damit klar ist, wovon wir reden.

1.4 Betrachtung von außen

An die **Anschlußleiste** haben Sie schon die Stromzufuhr geklemmt. Daneben befinden sich noch weitere Anschlußmöglichkeiten. Über diese kann der Computer Signale von außen empfangen oder nach außen abgeben. Aber darüber werden wir uns später noch ausführlich unterhalten. Im Augenblick hierzu nur soviel: an diese Anschlüsse darf unter keinen Umständen eine höhere Spannung als 5 V angelegt werden, da sonst der Computer beschädigt werden kann.
Das **Anzeigenfeld** macht sich leuchtend bemerkbar. Die Schlitze daneben sind Lüftungsschlitze. Bei längerem „Arbeiten" mit dem Computer wird nicht nur Ihnen warm.
Vorne ist die Tastatur, eine moderne **Flachtastatur,** die schon auf sanften Druck reagiert. Auf dieser gibt es zweierlei Arten von „Tasten": **Zahlentasten** und **Funktionstasten** (Bild 3). Bei den

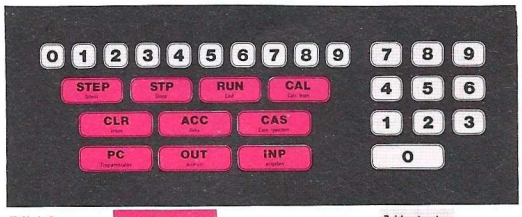


Bild 3

Zahlentasten haben Sie die Auswahl zwischen dem Zahlenblock auf der rechten Seite und der Zahlenreihe oberhalb der Funktionstasten. Es ist völlig egal, welche Zahlentasten Sie benutzen. Wenn Sie oft mit der Schreibmaschine schreiben, wird Ihnen

die Zahlenreihe sympathischer sein; Taschenrechner-Spezialisten werden den Zahlenblock bevorzugen.
Die großen länglichen Tasten auf der linken Seite der Tastatur sind die Funktionstasten. Mit diesen bringt man den Computer dazu, etwas zu tun, zu „funktionieren". Die einzelnen Tasten werden immer dann erklärt, wenn wir sie das erste Mal brauchen.
Allerdings, eine Taste sollten wir schon jetzt kennenlernen. Die CLR (clear)- oder auch Irrtums-Taste (clear ist englisch und heißt frei übersetzt säubern oder auch löschen). Drücken Sie mal auf diese Taste. Was sehen Sie? Nichts mehr? Doch, ein kleines rotes Pünktchen. Dieses Pünktchen zeigt an, daß das Gerät unter Strom steht, also betriebsbereit ist. Das als Anmerkung nebenbei; zurück zur CLR-Taste. Mit ihr können Sie die Anzeige löschen, wie Sie es vom Taschenrechner her kennen. Einzige Ausnahme: sollten Sie versehentlich die Funktionstasten „CAL" oder „CAS" gedrückt haben – auf der Anzeige erscheinen die Symbole

oder

– müssen Sie erst „STP" und dann „CLR" betätigen.

1.5 Das erste Programm – ein Test

Wir haben den Computer eingeschaltet, mit der CLR-Irrtums-Taste haben wir die Anzeige gelöscht. Das kleine rote Pünktchen zeigt uns, daß das Gerät betriebsbereit ist. Nun sollten wir überprüfen, ob der Computer auch funktioniert. Lassen Sie ihn das selbst machen: drücken Sie nacheinander

Auf der Anzeige erscheint in rascher Folge 999999, 888888, 777777 usw. . . . Mit 000000 beendet der Computer das Selbstprüfungsprogramm. Sie können es gleich wiederholen, indem Sie entweder mit CLR die Anzeige löschen und 9-RUN drücken, oder indem Sie auf CLR verzichten und gleich 9-RUN eintippen. Probieren wir's aus. Jetzt darf man sicher sein, daß der Computer funktioniert.

1.6 Das zweite Programm – ein Reaktionsspiel

Lassen Sie uns zur Entspannung ein kleines Spielchen machen. Nicht daß Sie denken, wir wollten uns jetzt einem unnützen Zeitvertreib zuwenden: es ist ein Spiel mit einem ernsten Hintergrund. Überprüfen Sie Ihre Reaktionsfähigkeit. Im Computer ist ein Reaktionstest vorprogrammiert. Die „8" ist hierfür die Programmziffer, mit RUN lösen Sie das Programm aus. Drücken Sie

Im Anzeigenfeld ist zunächst nichts zu sehen; dann aber läßt der Computer Zahlen rasen und zwar in Schritten von 1/1000 Sekunden. Sie sollten nun versuchen, den Zahlenlauf so schnell wie möglich mit Hilfe der STP-Taste zu stoppen. Auf geht's!

Wenn Sie es mehrmals machen, werden Sie feststellen, daß der Zeitpunkt, an dem der Computer zu zählen anfängt, immer unterschiedlich ist. Sonst wäre es ja auch schnell langweilig. Sollten Sie im Eifer die STP-Taste gedrückt haben, *bevor* der Computer angefangen hat zu zählen, erscheint nach kurzer Zeit „000". Schummeln gilt nicht.
Der ernste Hintergrund dieses netten Spielchens: Der Staat billigt einem Autofahrer bei Gefahr eine „Schrecksekunde" von 0,25 Sekunden zu, innerhalb dieses Zeitraumes muß der Autofahrer reagiert haben. Aber Sie lagen bestimmt unter 0,250 Sekunden. Mit weniger als 0,150 (Anzeige .150) sind Sie Spitzenklasse, und bei Zeiten unter 0,100 Sekunden (Anzeige .100) sind Sie reif für das Guinness-Buch der Rekorde.
Bei diesem Spiel haben Sie nun noch eine weitere Funktionstaste kennengelernt. Die STP-Taste.* Mit ihr können Sie Programme, die im Computer ablaufen, unterbrechen und mit der RUN-Taste dann wieder weiterlaufen lassen. Eine Ausnahme: das Selbst-Überprüfungsprogramm kann man nicht stoppen. Wozu auch?!

1.7 Was kann unser Computer?

Man sollte jetzt einmal das Reaktionsprogramm laufen lassen und nicht die STP-Taste drücken.

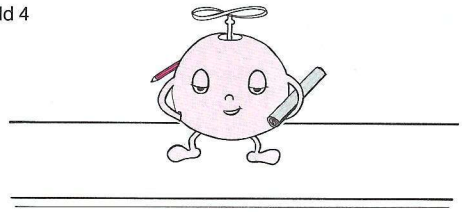
* Eine Zusammenstellung der Tastenfunktionen finden Sie auf dem beigefügten Blatt.

Der Computer zählt und bleibt dann bei 255 stehen. Dies ist sein gesamter Zahlenumfang. Mehr braucht er auch nicht, denn er soll keinen Taschenrechner ersetzen, sondern regeln, steuern, messen, „Entscheidungen treffen", Melodien erzeugen und vieles mehr. Das alles kann er aber nur, wenn man ihm zuvor genau beibringt, was er alles zu tun hat. Er muß – ähnlich wie ein neugeborenes Kind – erst etwas „lernen", bevor er eine Aufgabe richtig zu lösen vermag. Und wir müssen über seine Funktion Bescheid wissen, um uns überhaupt mit ihm verständigen zu können. Sie haben schon erfahren: ohne richtige Anweisungen macht ein Computer gar nichts, da kann man soviel auf die Tasten drükken, wie man will (die fest einprogrammierte „Selbstprüfung" und der „Reaktionstest" stellen hier eine unübliche Ausnahme dar).
Versuchen wir also zunächst der Frage nachzugehen, was ein Computer ist.

1.8 Das Computron stellt sich vor

Was um alles in der Welt ist nun eigentlich ein Computer? Erstens ist „Computer" ein englisches Wort und wird zweitens im Duden mit „elektronische Rechenanlage" übersetzt. Nun ist es zweifelsohne so, daß man den kleinen Kasten, der den KOSMOS-Computer ausmacht, nicht als „Anlage" bezeichnen würde. Aber früher, vor 15 bis 20 Jahren benötigte man für das, was unser kleines Gerät fertigbringt, große Anlagen, die ganze Schränke füllten. Daher dieser „große Begriff". Rechnen ist übrigens längst nicht mehr die Hauptaufgabe von Computern. Aber ihre Funktionsweise beruht auf mathematisch-logischen Prinzipien.
Wichtig für uns ist, daß alles, was im Computer geschieht, **elektronisch** passiert – im Unterschied zu **mechanisch.** Es werden also keine Räder oder Hebel angetrieben, wie z.B. bei einer Schreibmaschine, wo durch Tastendruck ein Hebelarm, Kugelkopf oder Typenrad bewegt wird. Solche mechanischen Bewegungen fallen – bis auf unseren Tastendruck – weg; die Anzeige ist ja auch elektronisch.
Was und wie etwas im Computer elektronisch passiert, entzieht sich unseren Sinnen. Wir können es weder sehen noch hören. Um es dennoch zu verdeutlichen, lassen wir in unseren erklärenden Zeichnungen ein kleines, rundes Wesen herumsausen. Dieses wieselflinke Ding – wir haben es „Computron" getauft – soll uns die Vorgänge veranschaulichen, die im Innern des Computers elektronisch ablaufen (Bild 4).
Vorher allerdings noch eine kurze Zwischenbemerkung, und zwar zur Sprache: die Sprache der Computerleute ist im wesentlichen Englisch. Sie

Bild 4



haben es bei den Funktionstasten schon bemerkt. Das ist selbstverständlich kein Problem für einen Engländer. Wohl aber eventuell für uns. Deshalb haben wir beim Computer unter die englischen Ausdrücke die deutsche Übersetzung geschrieben und werden auch in diesem Buch englische Ausdrücke zunächst einmal übersetzen. Dann aber werden wir die englischen Begriffe, eben weil es internationale Fachbegriffe sind, weiterverwenden.

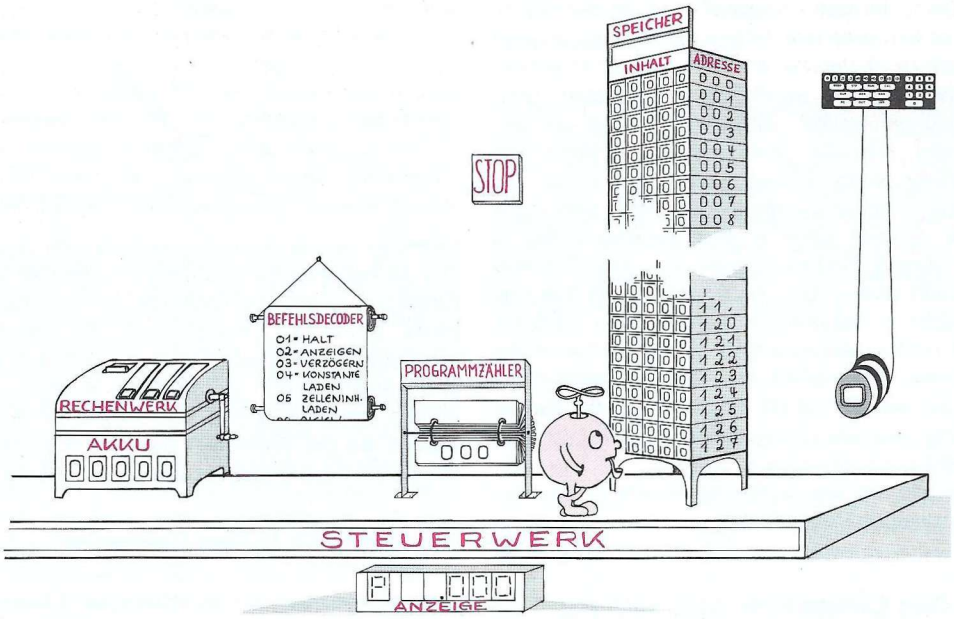
1.9 Ein Blick in den Computer

Um zu verstehen, wie ein Computer funktioniert, bedarf es keiner Kenntnisse seiner elektronischen Funktionsweise oder der Konstruktion seiner Bauteile – genau so wenig, wie man etwas vom Bau oder der Reparatur eines Autos verstehen muß, um es fahren zu können.
Im Bild 5 (Seite 10 oben) sehen wir das Fantasie-Wesen Computron in seinem Reich, das unsere Zeichnerin entsprechend dem elektronischen Innenleben unseres Computers gestaltet hat. Alle elektronischen Funktionseinheiten, die jeder Computer in seinem Innern beherbergt, finden sich auch hier wieder: ein Speicher, ein Akkumulator, ein Programmzähler, ein Rechenwerk, ein Befehlsdecoder, eine Tastatur, eine Anzeige und ein Steuerwerk. Nur, hier ist alles „unelektronisch" und damit auch für Nicht-Elektroniker ohne weiteres verständlich. Wir sehen, daß das Computron gerade Pause macht und gelangweilt den großen Turm mit der Aufschrift „Speicher" betrachtet.
Nehmen wir zur Erläuterung der Speicherfunktion ein Beispiel aus dem täglichen Leben, etwa eine Behörde. Dabei ist es vollkommen egal, ob man sich das Einwohnermeldeamt oder die Kraftfahrzeug-Zulassungsstelle oder was auch immer vorstellt. Eine Behörde eben. Es gibt ja genug davon.

1.10 Der Speicher

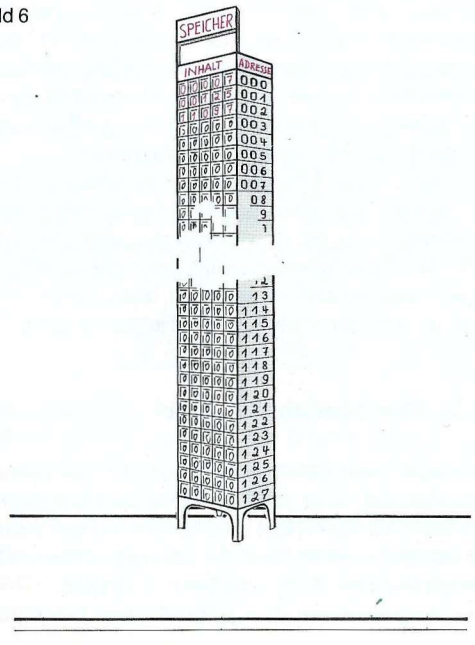
Herzstück einer Behörde ist das Archiv mit seinen Aktenordnern. Und in diesen Aktenordnern befinden sich, der Name läßt es erhoffen, wohlgeordnet die einzelnen Akten, also das Informationsmaterial, das die Beamten für ihre Tätigkeit benötigen.
Ähnlich ist es beim Computer. Dessen Herzstück

Bild 5



ist der Speicher, in dem Informationen „aufbewahrt" werden. Dieser Speicher ist aus sogenannten Speicherplätzen oder **Speicherzellen** aufgebaut. Sie entsprechen den Aktenordnern im Archiv. Und in diesen Aktenordner-Speicherzellen können im Falle unseres Computers bis zu fünf Ziffern „abgelegt" werden. So läßt sich also in eine Speicherzelle des Speichers z:B. die Zahl 7, in eine andere die Zahl 125 und in eine dritte die Zahl 11097 hineintun und auch wieder herausholen. Allerdings muß man, da der Computer keine leeren Stellen kennt, immer fünfstellig schreiben, also: „00007", „00125" und – kein Problem – „11097" (Bild 6).

Bild 6



Der Speicher des KOSMOS-Computers verfügt in der Grundausführung über 128 Speicherplätze, die alle durchnumeriert sind und zwar von 000 bis 127. Sie, und wir, würden zwar von 1 bis 128 numerieren, aber die Mathematiker fangen bei 0 an zu zählen. So sind sie eben.

1.11 Richtig adressiert ist halb gespeichert

Wichtig für uns ist, daß jeder Speicherplatz seine ganz bestimmte Nummer hat – wie in der Behörde der Aktenordner ein ganz bestimmtes Aktenzeichen. Diese Nummer heißt **Adresse.** Wenn wir die Nummer der Speicherzelle, also die Adresse anwählen, können wir über deren Inhalt verfügen oder erst einmal etwas hineingeben. Dabei muß man allerdings aufpassen: wenn man in eine Speicherzelle etwas Neues hineingibt, dann wird der alte Inhalt automatisch gelöscht und ist verloren. Er wird überschrieben durch den neuen Inhalt.

Probieren wir es gleich einmal aus. Sie wollen z.B. in die Speicherzelle 087 Ihres Computers die Zahl „7" eingeben. Soweit, so einfach. Aber wie?

Zunächst wählen Sie die stets dreistellige Adresse der Speicherzelle an. Sie tippen der Reihe nach die Tasten

In Bild 7 sehen wir, daß das schläfrige Computron inzwischen mächtig aktiv geworden ist. Beim

ersten Tastendruck war es zur Stelle, hat eifrig notiert, was an eingegebenen Ziffern durch das Rohr zu sehen war und blitzschnell die notierten Ziffern in die Anzeige eingeschrieben. Das können wir ja auch an unserem Gerät beobachten: bei jedem Tastendruck erscheint die entsprechende Ziffer sofort auf der Anzeige.

Bild 7a

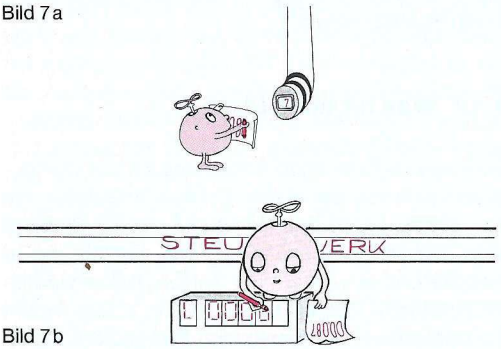


Bild 7b

Um zunächst einmal zu überprüfen, was in Speicherzelle 087 steht, drücken wir die Taste

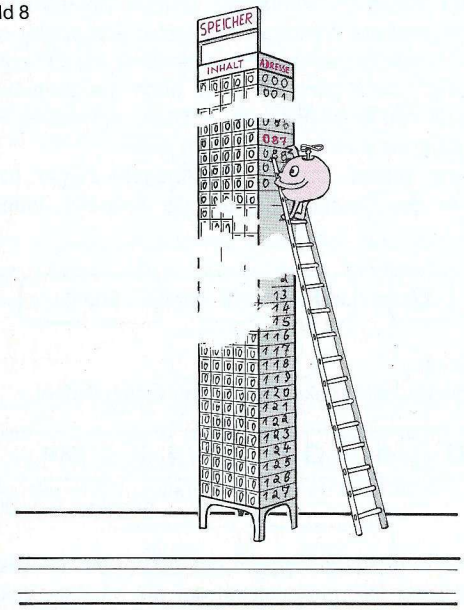
OUT

OUT ist englisch und steht für Ausgabe oder Auslesen. Auf der Anzeige erscheint

C 00.000

Der Kennbuchstabe „C" ganz links bedeutet, daß jetzt der Inhalt einer Speicherzelle angezeigt wird (eigentlich müßte es ja ein „Z" wie Zelle sein, aber auf der Digitalanzeige kann man Z nicht erscheinen lassen). Was ist im Computer passiert?

Bild 8



Das Computron hat sich eine Leiter geschnappt, sie exakt bei Speicherzelle 087 eingehängt, den Inhalt der Speicherzelle abgeschrieben und auf die Anzeige übertragen. Damit haben Sie die Speicherzelle 087 angewählt. Die Leiter, die das Computron entsprechend Ihrer Eingabe an jeder beliebigen Speicherzelle einhängen kann, wird in der Computer-Fachsprache als **Ein-Ausgabe-Zeiger** bezeichnet.

Und jetzt sollen Sie die Zahl „7" in die Speicherzelle 087 eingeben. Das Computron steht schon wieder am Tastenrohr, um weitere Eingaben zu notieren. Sie tippen die Zahl 7 als

0 **0** **0** **0** **7**

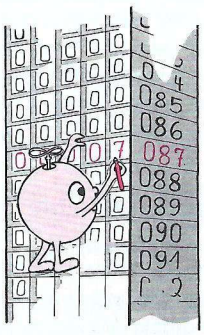
ein und beenden die Eingabe mit

INP

INP ist die Abkürzung des englischen Wortes „Input" = Eingabe.

Behende ist das Computron mit unserem Zahlenwert auf die Leiter gehüpft, hat in Null Komma nichts die Speicherzelle 087 mit 00007 „gefüllt" und in die Anzeige den entsprechenden Wert und ein „E" eingeschrieben. Als Quittung für unsere Eingabe zeigt unser Computer ein „E" an. Fertig. Damit haben Sie in die Zelle 087 etwas hineingegeben. Sehen tut man es natürlich nicht. Könnte man aber – und damit wollen wir das Computron aufs Neue beschäftigen.

Bild 9



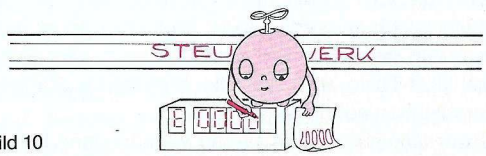
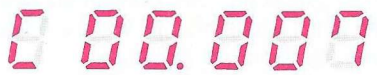


Bild 10

1.12 Der Zelleninhalt wird angezeigt

Wir drücken der Reihe nach die Tasten

Das Computron hat seine Leiter wieder bei Speicherzelle 087 eingehängt, sich den gespeicherten Wert notiert und die Anzeige entsprechend umgeändert. Es erscheint



Jetzt sehen wir das, was wir eben in die Zelle 087 eingegeben haben, nämlich unsere Zahl „7". Das C vor der Zahl gibt uns wieder an, daß dies ein Zellen-Inhalt ist.
Vielleicht probieren Sie das Ganze noch einmal mit Zelle 113. Tippen Sie ein

Auf der Anzeige erscheint

also der Inhalt von Speicherzelle 113. Geben Sie nun eine Zahl zwischen „1" und „255" ein und überprüfen Sie anschließend, ob diese Zahl auch in die Speicherzelle hineingekommen ist, z.B. „102":

und dann

Anzeige jetzt:

1.13 Automatische Fehleranzeige

Vielleicht haben Sie beim Eintippen schon mal einen Fehler gemacht und eine Zahlentaste zuviel oder zuwenig gedrückt. Mit der Anzeige F .001 gibt der Computer Ihnen dann zu verstehen, daß Sie etwas falsch gemacht haben. Probieren Sie es aus! In einem derartigen Fall betätigt man in aller Ruhe die CLR-Taste und gibt die betreffende Zahlenkombination nochmals ein.
Unser Computer kann sieben verschiedene Fehler (F .001 bis F .007) anzeigen und bietet dem Benutzer damit eine wertvolle Hilfe, Irrtümer rechtzeitig und gezielt zu erkennen. Die Bedeutung der verschiedenen Fehleranzeigen haben wir auf dem beiliegenden Informationsblatt in einer Tabelle zusammengefaßt.
Wir wollen das Eingeben in den Speicher nun gleich an einem richtigen Programm etwas üben, einem Würfelprogramm.

1.14 Was ist ein Befehl?

Sie wissen schon, ohne Anweisung tut der Computer von sich aus gar nichts. Er braucht Befehle von uns. Leider kennt der Computer unsere Sprache nicht. Wir müssen also eine Form wählen, die er versteht, und er versteht nur Zahlen. Daher werden wir Befehle in Zahlen verschlüsseln – man nennt das **codieren** – und diese dann mit Hilfe der Tastatur in den Computer eingeben.
Ein Befehl ist also eine in Zahlen verschlüsselte Anweisung für den Computer, etwas Bestimmtes zu tun. Alle Befehle müssen – wohlgeordnet zu einem Programm – zunächst in den Speicher eingegeben werden. Der Computer holt sich, wenn wir ihm durch die RUN-Taste das Startzeichen geben, einen Befehl nach dem anderen heraus und führt ihn aus.

1.15 Computer als Würfelmaschine

Sie werden jetzt zum erstenmal ein kleines Programm, also eine Folge von Befehlen, in den Computer eingeben. Verstehen können Sie es noch nicht, weil die Wirkung der Befehle erst später erklärt wird. Betrachten Sie dies also als „Fingerübung", denn ganz im Ernst: auch das Eingeben hat für den Anfänger seine Tücken, und man sollte es ruhig ein paar Mal regelrecht üben.
Zuerst setzen Sie den Ein-Ausgabe-Zeiger (die Leiter des Computrons) auf die Zelle 001, indem Sie

drücken.
In dieser Zelle speichern Sie den ersten Befehl

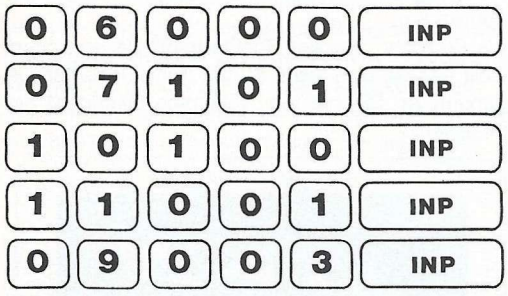
Anzeige: E 04.001

Der Ein-Ausgabe-Zeiger geht nach dem Drücken der Taste INP automatisch auf die nächste Zelle,

d.h. unser Computron legt seine Leiter ganz selbständig auf die folgende Zelle 002 an. In diese geben wir dann den Befehl

Anzeige: E 06.101

Auch jetzt rückt der Ein-Ausgabezeiger automatisch eins weiter; das Computron legt selbständig die Leiter an die Zelle 003. Das Weiterrücken des Ein-Ausgabezeigers erspart uns das jeweilige neue Anwählen der folgenden Zelle. Deshalb können wir nun in rascher Folge die nächsten Befehle eingeben.



Zum Schluß muß bei diesem Programm noch die Zelle 100 angewählt werden

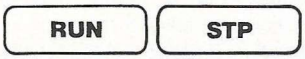
und anschließend

eingegeben werden. Um das Programm zum Laufen zu bringen, teilen wir dem Computer durch Drücken von

zunächst mit, mit welcher Zelle er beginnen soll und drücken dann

Die Anzeige erlischt, der Computer „würfelt", und dabei läßt er sich nicht zuschauen. Stoppen Sie die elektronische Würfelei, indem Sie die Taste

betätigen. Sofort erscheint auf der Anzeige eine Zahl von 1 bis 6 (natürlich computermäßig von 00.001 bis 00.006), eben der gewürfelte Wert. Würfeln Sie weiter, indem Sie immer abwechselnd



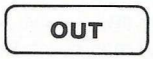
drücken.

1.16 Wenn das Würfelprogramm nicht laufen will . . .

Läuft das Würfelprogramm nicht auf Anhieb? Signalisiert Ihnen die Anzeige einen Fehler (F . . .)? Nun, dann können Sie sicher sein, daß beim Eintippen ein Irrtum passiert ist. Überprüfen Sie also, ob Ihr Programm im Speicher wirklich drin ist. Dazu wählen Sie die Speicherzelle 001 nochmals an:

Es muß

erscheinen. Um den Inhalt der folgenden Speicherzelle zu betrachten, brauchen Sie jetzt nur noch die Taste



zu drücken. Dadurch rückt der Ein-Ausgabe-Zeiger – also die Leiter des Computrons – automatisch um einen Schritt nach unten. Jedesmal wenn Sie die OUT-Taste drücken, erscheint auf der Anzeige der nächste Speicherzelleninhalt, den Sie mit den eingegebenen Ziffern vergleichen können.

Wenn nun auf der Anzeige ein falscher Speicherzellenwert erscheint, können Sie ihn sofort korrigieren, indem Sie ihn einfach (mit dem richtigen Wert) „überschreiben":

Ohne die Speicherzelle neu anzuwählen (der Ein-Ausgabe-Zeiger steht nämlich durch das letzte Drücken von OUT schon richtig), geben Sie den richtigen Wert ein und drücken die INP-Taste. Fertig.

Wenn Sie beim Eingeben noch Probleme hatten, sind die folgenden Hinweise sehr nützlich: Beobachten Sie beim Drücken einer Taste die Anzeige. Wenn Sie Ziffern eintippen, müssen diese immer gleichzeitig auf der Anzeige erscheinen!

Achten Sie darauf, daß Sie jede Eingabe in eine Speicherzelle durch Betätigen der INP-Taste beenden. Als Quittung, daß der Wert wirklich in die Speicherzelle eingeschrieben wurde, zeigt Ihnen der Computer ganz links den Kennbuchstaben E an.

Es kann vorkommen, daß man versehentlich eine Taste zweimal drückt (Sie sehen es auf der Anzeige). Drücken Sie die CLR-Taste und beginnen Sie die Eingabe des betreffenden Speicherzellen-Wertes von vorn, beenden Sie die Eingabe mit INP.

Bemerken Sie erst *nach* dem Drücken von INP, daß Sie einen falschen Wert eingegeben haben, so wählen Sie dieselbe Speicherzelle nochmals an

und überschreiben den fehlerhaften Wert durch eine neue Eingabe: Speicherzellen-Nummer, OUT, dann richtigen Wert und INP drücken (siehe Ausführungen zu Beginn dieses Kapitels).
Zum Schluß noch zwei nützliche Hinweise: Wenn man sich durch fortlaufendes Drücken der OUT-Taste den Inhalt des Speichers anzeigen läßt und vergißt, die Speicherzellen-Nummern (also die Adressen) mitzuzählen, kann man sich vom Computer durch Betätigen von

[9] [OUT]

diejenige Adresse mitteilen lassen, deren Inhalt als letztes angezeigt wurde.
Beispiel: Auf der Ziffernanzeige steht C 07.101. Sie drücken 9-OUT, und der Computer zeigt

C 07.004

an. Das bedeutet, daß der Wert 07.101 in Speicherzelle 004 gespeichert ist.
Beim Eingeben kann man durch Drücken von

[9] [OUT]

feststellen, wo der Ein-Ausgabe-Zeiger steht, in welcher Zelle also der nächste einzugebende Wert gespeichert wird. Der Speicherzelleninhalt oder die Zellenadresse lassen sich natürlich nur anzeigen, wenn kein Programm läuft. Es muß gegebenenfalls mit STP angehalten werden.

1.17 Der Computer zieht Lottozahlen

Wollen Sie mal zur Abwechslung den Computer Lottozahlen ziehen lassen? Kein Problem, es muß lediglich der Wert einer einzigen Speicherzelle geändert werden.
Wählen Sie die Zelle 100 an

[1] [0] [0] [OUT]

und tippen Sie

[0] [0] [0] [5] [0] [INP]

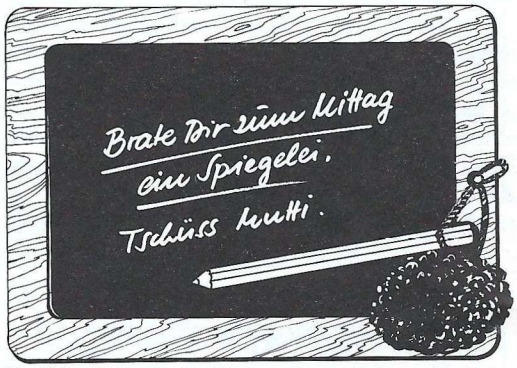
Wie man dieses Programm startet wissen Sie schon:

[0] [0] [1] [PC] [RUN]

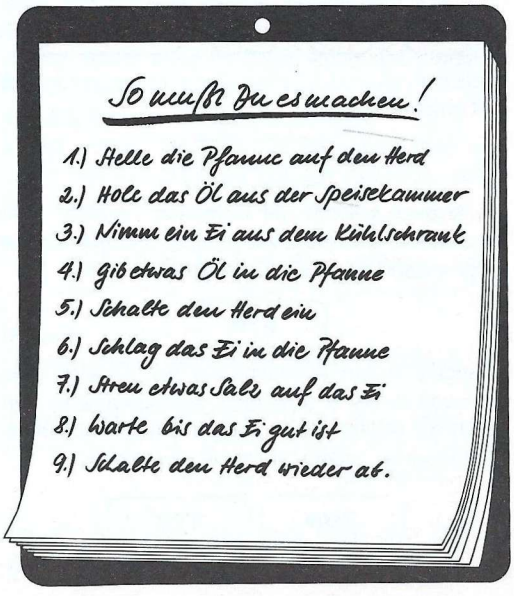
Wenn Sie jetzt auf die Taste STP drücken, zeigt Ihnen der Computer an, welche Zahl Sie für Ihren Lotto-Tip nehmen können. Lassen Sie sich noch weitere Lottozahlen „ziehen", indem Sie den Computer erneut starten und stoppen.

1.18 Was ist ein Programm?

Soviel kann man an dieser Stelle bereits festhalten: Für seine Arbeit benötigt der Computer ein Programm, und dieses Programm muß in seinen Speicher hineingegeben werden. Ohne Programm ist der Computer nichts wert, je besser das Programm, desto wertvoller die Computerarbeit. Kein Computer der Welt kann ideenreicher, scharfsinniger und kreativer sein als der Mensch, der ihn programmiert hat, also der Programmierer.
Vor dem Programmieren steht das Problem, die Aufgabenstellung. Lassen Sie uns das wie gewohnt an einem Beispiel aus dem täglichen Leben verdeutlichen.
Als Fritzchen aus der Schule nach Hause kommt, stellt er fest, daß seine Mutter zum Einkaufen gegangen ist. Sie hat ihm aber eine Nachricht hinterlassen:



Da Fritzchen noch nie in seinem Leben ein Spiegelei gebraten hat, steht er dieser Aufgabe hilflos gegenüber. Das Problem hätte in viele kleine Schritte unterteilt werden müssen, etwa so:

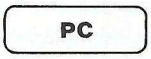


Sie sehen schon, worauf es ankommt: die Ausführung der Tätigkeit muß Schritt für Schritt scharf durchdacht sein. Man nennt dies **Problem-Analyse.** Fehlt in der Liste eine einzige Anweisung, so wird man nicht zum Ziel gelangen.
Die Computerleute arbeiten grundsätzlich nach dem selben Schema:

Aufgabenstellung – Problemanalyse – Programm.

1.19 Der Programmzähler

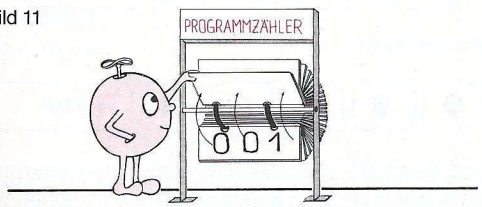
Für die Einhaltung der vorgegebenen Reihenfolge, also für einen ordnungsgemäßen Ablauf eines Programms, sorgt im Computer der **Programmzähler.** Der Programmzähler zeigt die Adresse des Befehls im Speicher an, der als nächstes ausgeführt wird.
Sie können vom Computer jederzeit erfahren, für welchen Befehl er bereit ist, wenn Sie die Taste



drücken. PC bedeutet „Program Counter", auf deutsch Programmzähler.
Sie können ihm umgekehrt aber auch vorschreiben, welchen Befehl er ausführen soll, indem Sie die Adresse des gewünschten Befehls, sagen wir mal die Adresse 001 eingeben und dann die Programmzählertaste drücken:

Lassen Sie uns zum Abschluß dieses Kapitels noch einen Blick auf das (zeichnerisch dargestellte) Computer-Innenleben werfen (Bild 11). Der Programmzähler ist als ein Paket von 128 Klapptafeln dargestellt. Er wird bei Bedarf von unserem Computerwesen betätigt.

Bild 11



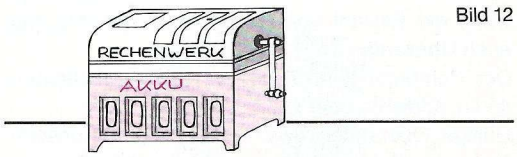
1.20 Zahlen für die Befehle

Kommen wir nun zurück zu den Computer-Befehlen, die – es wurde in Kapitel 1.14 bereits gesagt – durch dezimale Zahlen dargestellt werden, Fachleute sagen: dezimal codiert sind. Was der Computer tun soll, müssen Sie ihm also durch eine Zahlenkombination angeben. Wenn man die verschiedenen Zahlenkombinationen – also die „Befehlscodes" – kennt, kann man ein Programm entwikkeln und dem Computer eingeben.
Der „KOSMOS-Computer" hat 21 verschiedene Befehle, die von 01 bis 21 durchnumeriert sind (für die Ausbauversion gibt es noch drei weitere, aber die sollen uns hier nicht interessieren). Wir werden Ihnen Schritt für Schritt die Befehle vorstellen und die Wirkungen erklären. Und Sie werden zwischendurch immer wieder erfahren, wie man durch Aneinanderreihen von Befehlen zunächst kleinere und einfache, später dann auch längere und kompliziertere Programme entwickelt.
Das erste Programm, das wir gemeinsam mit Ihnen ausarbeiten wollen, soll den Computer dazu bringen, vollautomatisch nacheinander die Inhalte der Speicherzellen Nr. 100 bis Nr. 104 anzuzeigen. Nichts Weltbewegendes, zugegeben, aber für Sie ein Meilenstein auf dem Wege in die faszinierende Computertechnik. Wir werden 4 Befehle benötigen.

1.21 Der Akkumulator

Mit dem ersten Befehl wird der Inhalt der Speicherzelle 100 zunächst in einen Zwischenspeicher transportiert, den man **Akkumulator** oder auch kurz Akku nennt. Der Akku eines Computers ist ein elektronischer Notizzettel, auf dem vorübergehend etwas notiert werden kann. Vielleicht erinnern Sie sich noch an die Aktenordner der Behörde, die wir mit unserem Speicher verglichen haben. Wenn der Sachbearbeiter eine bestimmte Information zu einem Sachverhalt benötigt, wird er den Aktenordner aufschlagen und sich einen kurzen Vermerk auf einem Notizzettel machen. Mit der Information auf dem Zettel kann er dann den Vorgang weiterbearbeiten.

Bild 12



Unser Computer verfährt genauso. Der Inhalt der gewünschten Speicherzelle wird „abgeschrieben" und zwecks Weiterverarbeitung in den Hilfsspeicher, den Akkumulator, eingeschrieben. Was im Akku steht, kann also direkt weiterverarbeitet werden. Der Befehl, den Inhalt einer Speicherzelle in den Akku zu transportieren, heißt allgemein formuliert

„Lade den Inhalt der Speicherzelle xxx in den Akku"

xxx steht dabei für eine xxx-beliebige Speicherzelle, also z. B. Speicherzelle 100. Das Wort „laden" ist sicherlich aus der Sprache der Automobilisten bekannt. Jeder Autobesitzer weiß, daß der (Auto-)Akku, also der (Elektrizitäts-)Speicher, geladen werden muß.

1.22 Operationscode und Operand

Der Zahlencode für „Akku laden" ist 05. Und damit der Computer auch weiß, *welchen* Zelleninhalt er laden soll, wird ihm durch 3 weitere Ziffern die Adresse der betreffenden Speicherzellen angegeben.

05.xxx

ist der komplette Befehl. Er besteht – der Punkt läßt es deutlich erkennen – aus zwei Teilen.

1. Links vom Punkt steht das, **was** gemacht werden soll (Akku laden).
2. Rechts vom Punkt steht, **wo** der Computer sich das holt, das er verarbeiten soll (aus Speicherzelle xxx).

Sie sollten bei dieser Gelegenheit gleich die Fachausdrücke kennenlernen, damit Sie mithalten können, wenn andere Leute Computer-Latein reden.
Der linke Teil des Befehls heißt **Operationscode.** Die Zahlenwerte 01 bis 21 sind für die Grundversion unseres Computers gültige Operationscodes.
Der rechte Teil kann bei den verschiedenen Befehlen unterschiedliche Bedeutung haben.
Bei einer großen Gruppe von Befehlen geben die drei Ziffern rechts vom Punkt eine Adresse an. Unter dieser Adresse findet der Computer den Zahlenwert, mit dem er etwas tun soll. Diesen Zahlenwert nennt man in der Fachsprache **Datum** (jawohl, die Einzahl von Daten heißt Datum!) oder auch **Operand.**
Der Computer kann Daten vergleichen, transportieren, addieren oder subtrahieren.
Gültige Operanden bzw. Daten sind für unseren Computer Zahlenwerte von 000 bis 255, die computermäßig als 00.000 bis 00.255 geschrieben werden müssen.
Bei einer zweiten Gruppe von Befehlen steht im rechten Teil eine Adresse, die jedoch angibt, wo der Computer im Programm *fortfahren* soll.
Bei anderen Befehlen wiederum ist der Operand direkt im Befehl enthalten. Die drei rechten Ziffern sind also bereits der Operand, der Computer muß ihn nicht erst aus dem Speicher holen. Solche Operanden heißen auch **Konstanten.**

Eine letzte, etwas abseits stehende Gruppe soll der Vollständigkeit halber noch erwähnt werden. Bei diesen Befehlen enthält der rechte Teil die Adresse einer Speicherzelle, in welcher der Computer dann erst die Adresse für den Operanden findet.
Operanden haben auch eine Schlüsselzahl, einen Code, damit man sie von den Befehlen eindeutig unterscheiden kann. Ein Speicherzellen-Inhalt, der mit 00 beginnt, ist kein Befehl, sondern ein Operand bzw. Datum.
Diese wichtige Erkenntnis wollen wir kurz noch einmal zusammenfassen:

Im Speicher eines Computers können sowohl Befehle (erkennbar an den Operationscodes 01 bis 21) als auch Daten (erkennbar an der Zahlenkombination 00 links vom Punkt) stehen.

Da Sie noch nicht alle Befehle Ihres Computers kennen, werden Sie vielleicht an dieser Stelle Verständnisschwierigkeiten gehabt haben. Das macht aber nichts, denn bei der späteren Erklärung der einzelnen Befehle erfahren Sie am praktischen Beispiel nochmals ganz genau, was die verschiedenen Bestandteile der Befehle für eine Bedeutung haben.

1.23 LDA – Speicherzellen-Inhalte laden

Wir werden jetzt den Befehl

„Lade den Akku mit dem Inhalt der Speicherzelle 100"

in den Computer eingeben, sagen wir mal in Speicherzelle 001. Sie wählen Speicherzelle 001 an:

und geben ein:

Und damit Sie anschließend auch sofort überprüfen können, ob „der Befehl funktioniert", geben wir als Inhalt in Speicherzelle 100 den Zahlenwert „11", computermäßig 00.011, ein. Wie das geht, wissen Sie ja bereits: Wählen Sie die Zelle 100 an

und geben Sie ein

Wie bringen Sie den Computer dazu, einen Befehl auszuführen, der in Speicherzelle 001 steht? Das Stichwort heißt Programmzähler (Kapitel 1.19). Stellen Sie den Programmzähler auf 001, indem Sie

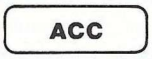
drücken.
Auf der Anzeige erscheint

als Zeichen, daß der Computer bereit ist, den Befehl der Zelle 001 auszuführen.

1.24 STEP – mit einem Schritt voran

Sie werden nun eine Möglichkeit kennenlernen, den Computer nur einen einzigen Befehl ausführen zu lassen. Er macht also nur einen Schritt und dann nichts mehr. Man erreicht dies durch Drükken der Taste

Was passiert? Die Anzeige wechselt von P .001 auf P .002. Das ist in Ordnung, denn der Computer hat ja den Befehl von Speicherzelle 001 ausgeführt und signalisiert, daß er bereit ist, den Befehl von Speicherzelle 002 zu erledigen (falls dort einer steht).
Aber was ist mit unserem Befehl?
Nun, schauen wir noch einmal an, ob der Akku tatsächlich geladen wurde. Dafür ist die Taste



vorgesehen. Wenn wir sie drücken, erscheint ganz links der Kennbuchstabe A und rechts davon der Akku-Inhalt.
Anzeige also

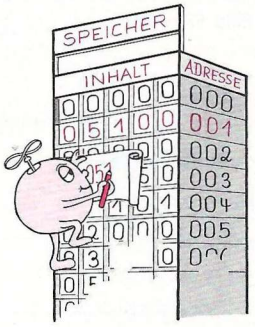
Wenn Sie ein mißtrauischer Mensch sind, können Sie zusätzlich noch überprüfen, ob der Inhalt der Speicherzelle 100 auch wirklich erhalten geblieben, also nur kopiert worden ist.

1.25 Was der Befehl alles in Gang setzt

Bevor wir unser Programm weiterentwickeln, sollten wir jetzt einen Blick in das Innere des Computers tun und unser Computerwesen Computron bei der Ausführung des Lade-Befehls beobachten. Erstaunlicherweise zerstückelt der Computer einen Befehl in eine Reihe von kleinen Unteraufgaben, die nacheinander erledigt werden:

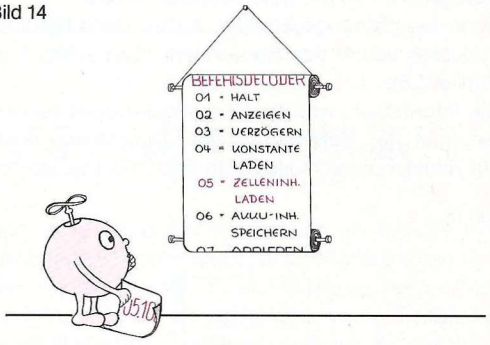
1. Der Inhalt der Speicherzelle, die den Befehl enthält, wird kopiert (Computron schreibt den Inhalt ab, Bild 13).

Bild 13



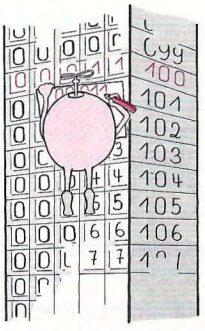
2. Der Befehl wird „entschlüsselt" (Computron schaut nach, was zu tun ist, Bild 14).

Bild 14



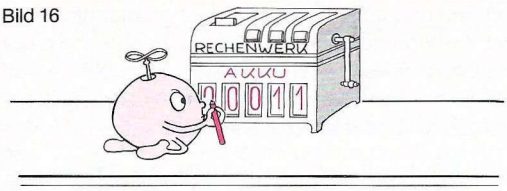
3. Der Inhalt von Speicherzelle 100 wird gelesen (Computron kopiert den Inhalt von Speicherzelle 100, Bild 15).

Bild 15



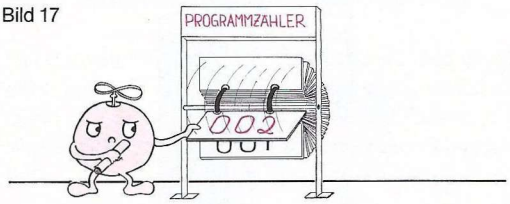
4. Der Speicherzellen-Inhalt wird zum Akku gebracht (Computron schreibt ihn in den Akku ein, Bild 16).

Bild 16



5. Der Programmzähler wird um eins erhöht (Computron klappt die nächste Tafel herunter, Bild 17).

Bild 17



Diese fünf Unterschritte nennt man einen **Befehlszyklus.** Bei anderen Befehlen sind mehr oder auch weniger Schritte pro Befehlszyklus möglich.
Über den ordnungsgemäßen Ablauf eines Befehlszyklusses wacht das **Steuerwerk** (Siehe Bild 5 in Kapitel 1.8).
Die Information, was die Operations-Codes bedeuten, gibt der **Befehlsdecoder.** Decodieren heißt auf deutsch entschlüsseln. In Bild 18 ist er als Ar-

Bild 18

chivrolle dargestellt, auf der die Bedeutung aller (Befehls-) Schlüsselzahlen vermerkt ist. Mit einer Kopie des Befehls in der Hand erfährt unser Computron stets am Befehlsdecoder, was zu tun ist.

1.26 Anzeigeprobleme

Lassen Sie uns überlegen, welche weiteren Anweisungen wir dem Computer geben müssen, damit er automatisch der Reihe nach die Inhalte der Speicherzellen 100 bis 104 anzeigt.
Durch den Lade-Befehl wurde der Inhalt der Speicherzelle 100 kopiert und im Akku zwischengespeichert. Jetzt muß er noch angezeigt werden. Von allein tut der Computer dies nicht – wir müssen es ihm per Befehl sagen. Sie werden also in Speicherzelle 002 den Befehl eingeben

„Zeige den Akku-Inhalt an"

(Der Zahlencode dafür lautet: 02.000)
Aber ehe wir zur Tat schreiten, schauen wir uns rasch den Rest des Programmes an.

1.27 Erst durch Verzögerung sichtbar

Es gibt noch ein Problem zu lösen. Die hohe Computergeschwindigkeit wird uns zu schaffen machen. Sie müssen dem Computer noch mitteilen, *wie lange* Sie den Akku-Inhalt auf der Anzeige zu sehen wünschen. Tun Sie's nicht, wird er das gesamte Programm in 6½ Tausendstel Sekunden erledigt haben – und Sie werden buchstäblich gar nichts sehen.
Sie können den Computer durch einen entsprechenden Befehl zwingen, seine Geschwindigkeit der Trägheit des menschlichen Auges anzupassen. Dieser Befehl heißt

„Warte xxx Millisekunden"

Wobei xxx Werte zwischen 000 und 255 annehmen kann (wir werden den Befehl als 03.250 eingeben). 250 Millisekunden sind ¼ Sekunde. Das ist auch für träge Zeitgenossen ein ganz brauchbarer Wert.

1.28 Kein Schluß ohne „Halt"

Und noch ein Letztes müssen Sie beachten. Ein ordentliches Programm muß zum Schluß einen Befehl enthalten, der den Computer anhalten läßt. Er kann ja nicht wissen, wann Ihr Programm zu Ende ist und würde – je nachdem, was sich in den folgenden Speicherzellen befindet – weiterlaufen oder eine Fehleranzeige bringen. Dieser Befehl heißt Halt-Befehl (codiert: 01.000).
Wir haben das komplette Programm nachfolgend aufgelistet. Die Computer-Leute bevorzugen natürlich englische Ausdrücke und nennen so etwas ein **Listing.**
Das Programm besteht aus Wiederholungen der Befehle Laden, Anzeigen und Verzögern und endet mit einem Halt-Befehl. Die Speicherzellen 100 bis 104 enthalten Operanden, also die Werte, die angezeigt werden sollen.
Sie sehen ganz links die Nummern der Speicherzellen, in die Befehle und Operanden eingegeben werden sollen (Spalte „Adresse").

Listing 1: **Speicherzellen-Inhalte automatisch anzeigen**

Adresse	Mnemonics	Code	Kommentar
001	LDA 100	**05.100**	Inhalt von Zelle 100 in den Akku laden
002	ANZ	**02.000**	Akku-Inhalt anzeigen
003	VZG 250	**03.250**	250 ms verzögern
004	LDA 101	**05.101**	Inhalt von Zelle 101 in den Akku laden
005	ANZ	**02.000**	Akku-Inhalt anzeigen
006	VZG 250	**03.250**	250 ms verzögern
007	LDA 102	**05.102**	Inhalt von Zelle 102 in den Akku laden
008	ANZ	**02.000**	Akku-Inhalt anzeigen
009	VZG 250	**03.250**	250 ms verzögern
010	LDA 103	**05.103**	Inhalt von Zelle 103 in den Akku laden
011	ANZ	**02.000**	Akku-Inhalt anzeigen
012	VZG 250	**03.250**	250 ms verzögern
013	LDA 104	**05.104**	Inhalt von Zelle 104 in den Akku laden
014	ANZ	**02.000**	Akku-Inhalt anzeigen
015	VZG 250	**03.250**	250 ms verzögern
016	HLT	**01.000**	Anhalten
100		**00.011**	Zahlenwerte, die der Reihe nach angezeigt werden sollen
101		**00.022**	
102		**00.033**	
103		**00.044**	
104		**00.055**	

1.29 Mnemonics als Gedächtnisstütze

Direkt daneben erscheint eine Spalte „Mnemonics". Mneme ist griechisch und heißt Gedächtnis. Bei den Befehlen sind in dieser Spalte statt des Zahlenschlüssels Buchstaben-Kombinationen eingetragen, also LDA (= lade den Akku), ANZ (= anzeigen), VZG (= verzögern, warten) usw. Diese Buchstaben-Kürzel sind als Hilfe für den Programmierer gedacht, als Gedächtnisunterstützung sozusagen. Wenn man ein längeres Programm schreibt, kann man anhand dieser Buchstaben viel leichter überschauen, was das Programm tut, als wenn man *nur* Zahlenwerte aufschreiben würde, die in unserem Listing in der Spalte „Code" vermerkt sind. Unser Computer wird selbstverständlich nur mit Zahlen „gefüttert". Buchstaben versteht er nicht. Ganz rechts haben wir zusätzlich zu jedem Befehl einen Kommentar gegeben.

1.30 Wir probieren das Programm aus

Wenn auf Speicherzelle 001 noch der erste Befehl 05.100 (= lade in den Akku den Inhalt der Speicherzelle 100) steht, beginnen Sie jetzt mit der Eingabe des Programmes beim zweiten Befehl. Sie wählen die Speicherzelle 002 an

und geben dann der Reihe nach die Befehle ein. Nach jeder Eingabe muß INP gedrückt werden und der Kennbuchstabe E als Quittung erscheinen. Aber Vorsicht: da wir nach Adresse 016 die fortlaufende Reihenfolge verlassen, müssen Sie die Speicherzelle 100 neu anwählen

und dann nacheinander die letzten fünf Werte eingeben. INP nicht vergessen! Wenn in Zelle 100 der Wert von vorhin noch stand, dann beginnen Sie mit Speicherzelle 101.

Wenn Sie sich beim Eingeben vertippen und Sie bemerken es, *bevor* das Quittungs-„E" ganz links erscheint, können Sie den bis dahin eingegebenen Wert mit der CLR-Taste einfach löschen und neu eintippen. War das „E" schon sichtbar, so müssen Sie die betreffende Speicherzelle zunächst neu anwählen (Speicherzellen-Nummer und dann OUT) und anschließend den richtigen Wert eingeben.

Da sich unser erster Befehl in Speicherzelle 001 befindet, stellen wir den Programmzähler auf

Auf der Anzeige signalisiert der Computer uns, daß er mit dem Befehl auf Adresse 001 beginnen wird (P .001).
Jetzt fehlt noch das Startzeichen

RUN

und los geht's.

Die Werte, die wir in den Speicherzellen 100 bis 104 hinterlegt hatten, erscheinen nun fein säuberlich der Reihe nach auf der Anzeige. Dann wird

P 88.017

angezeigt, und der Computer bleibt stehen. Die Anzeige P .017 gibt Auskunft darüber, welchen Befehl der Computer als nächstes ausgeführt hätte, wenn wir ihm nicht mit dem Befehl in Zelle 016 einen HALT befohlen hätten. Sie können das kleine Programm durch Drücken von 001 – PC – RUN beliebig oft wiederholen. Es steht ja abrufbereit im Speicher und ginge nur durch Ziehen des Netzsteckers verloren.
Wie wär's, wenn Sie sich zur Abwechslung auch einmal andere Zahlenwerte anzeigen ließen? Kein Problem, Sie brauchen nur die Speicherzellen 100 bis 104 zu verändern.

1.31 Computer im Einzelschritt-Betrieb

Übrigens: Sie können das Programm auch schrittweise ablaufen lassen und sich nach jeder Befehlsausführung den Akku anschauen, indem Sie zunächst

0 0 1 PC

und dann fortlaufend

STEP ACC STEP ACC . . .

drücken.
Wir werden die STEP-Funktion noch oft benötigen, um den Ablauf eines Programmes in aller Ruhe verfolgen und eventuelle Fehler finden zu können. (Siehe auch Kapitel 1.71.)

1.32 ANZ – per Befehl anzeigen

Der zweite Befehl, den wir bei dem vorigen Programm benutzt haben, sorgte dafür, daß die Speicherzellen-Werte nach dem „Transport" zum Akku tatsächlich auf der Anzeige erschienen. Denn: Wenn unser Computer arbeitet, schaltet er die Anzeige einfach ab. Er läßt sich eben nicht gern in die Karten sehen, aber das hat durchaus einen vernünftigen Grund. Wegen der hohen Computer-Geschwindigkeit wechselt bei den meisten Programmen der Akku-Inhalt so rasch, daß es nutzlos wäre, das entstehende Geflimmer anzuschauen. Also spart man Strom und läßt die Anzeige dunkel.
Der Befehl, mit dem man gezielt den Akku-Wert anzeigen kann, heißt Anzeige-Befehl mit der Kurzbezeichnung ANZ und der Schlüsselzahl 02. Wenn man ihn in ein Programm einfügt, wird der Wert, der gerade im Akku steht, während des Programmlaufes sichtbar gemacht. Dieser Wert bleibt nun auf der Anzeige – unabhängig von allem, was im Akku zwischenzeitlich passiert – solange stehen, bis im Programm ein weiterer ANZ-Befehl auftritt.
Etwas sonderbar verhält es sich mit dem rechten Teil dieses Befehls. Man braucht ihn nämlich gar nicht. Denn Akku-Anzeige ist eindeutig und bedarf keiner weiteren Angabe. Da alle Befehle jedoch aus fünfstelligen Dezimalzahlen bestehen müssen, fügen wir irgendeine Zahl von 000 bis 255 an die Schlüsselzahl 02 an, damit „der Befehl komplett wird". Am einfachsten ist es, wenn man sich angewöhnt, den Anzeige-Befehl immer als

02.000

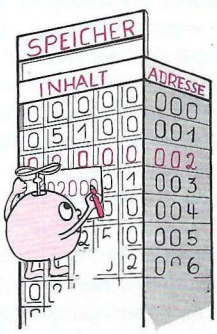
oder in der Buchstaben-Kurzform nur als

ANZ

zu schreiben. Dann kann es nie Probleme geben.
Der Computer verarbeitet den Anzeige-Befehl folgendermaßen:

1. Der Befehl wird aus dem Speicher geholt (Bild 19)

Bild 19



2. Der Befehl wird decodiert (Bild 20)
3. Der Akku-Inhalt wird abgeschrieben (Bild 21)
4. Der Akku-Wert wird in die Anzeige eingeschrieben (Bild 22)
5. Der Programmzähler wird um eins weitergeschaltet (Bild 23).

Bild 20

Bild 21

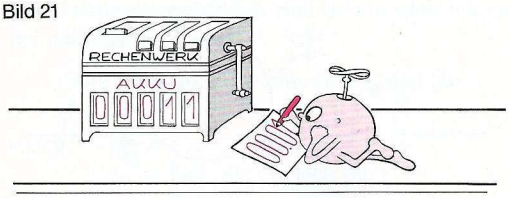


Bild 22

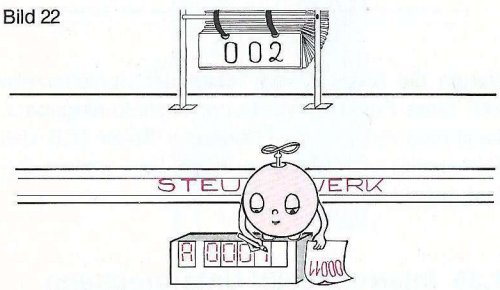
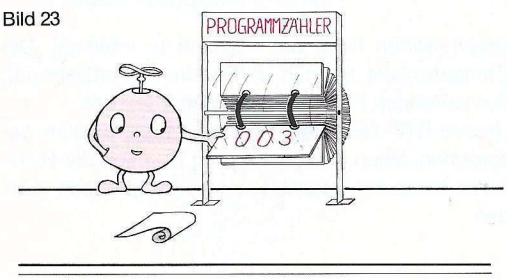


Bild 23



1.33 VZG – die programmierte Verzögerung

Sind sie flott im Rechnen? Wie lange brauchen Sie, um – sagen wir mal – 124 und 77 zusammenzuzählen? Wenn Sie's in 3 Sekunden schaffen, sind sie Spitzenklasse.

Unser Computer benötigt dafür nur 315 Millionstel Sekunden. Und das ist für einen Elektronenrechner nicht einmal besonders schnell. Das Ausarbeiten eines Programmes braucht seine Zeit, und das Eingeben eines Programmes in den Speicher kann ebenfalls eine gute Weile dauern. Aber dann – wenn das Programm einmal eingegeben wurde – erweist sich der Computer wegen seiner Wahnsinnsgeschwindigkeit dem Menschen gegenüber als unschlagbar.

Die hohe Verarbeitungsgeschwindigkeit eines Computers kann allerdings auch unerwünscht sein. Erinnern Sie sich bitte an das letzte Programm: hätte man den Computer gewähren lassen, so hätte er in wenigen Tausendstel Sekunden nacheinander alle gewünschten Speicherzellen-Inhalte angezeigt – und Sie hätten nicht einmal gemerkt, daß etwas geschehen ist.

In unser Programm sind daher, wie schon in Kapitel 1.27 erläutert, „Brems"-Befehle eingebaut. Sie stehen in den Speicherzellen 003, 006, 009, 012 und 015. Wir sagten bereits, daß wir sie Verzögerungsbefehle nennen und durch die Buchstaben VZG abkürzen.

VZG xxx

bedeutet, daß der Computer xxx Millisekunden Pause macht. Sie können jede Pausenzeit von 000 bis 255 Millisekunden wählen. Die drei Zahlen rechts vom Punkt geben also (in Millisekunden) an, wie lange der Computer verzögern soll. Die Schlüsselzahl für den VZG-Befehl ist 03. Computermäßig notiert lautet der Befehl:

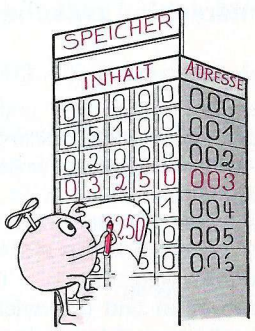
03.xxx

Natürlich sind 255 Millisekunden (also etwas mehr als ¼ Sekunde) für den trägen menschlichen Geist immer noch eine kurze Zeit. Das ist jedoch kein Problem: wir können ja dem Computer befehlen, mehrmals Pause zu machen. Durch Programmiertricks, die wir später kennenlernen werden, können Verzögerungszeiten von Minuten, ja sogar von Stunden erreicht werden.

Der Befehlszyklus für den VZG-Befehl läuft folgendermaßen ab:

1. Der Befehl wird aus dem Speicher geholt (Bild 24)

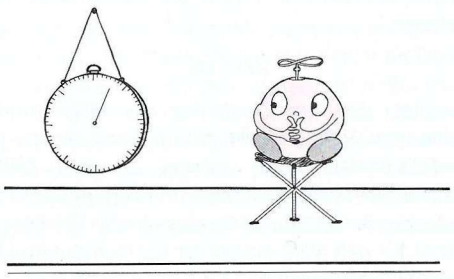
Bild 24



2. Der Befehl wird decodiert (Bild 25)
3. Die Verzögerungszeit wird ausgeführt (Bild 26)

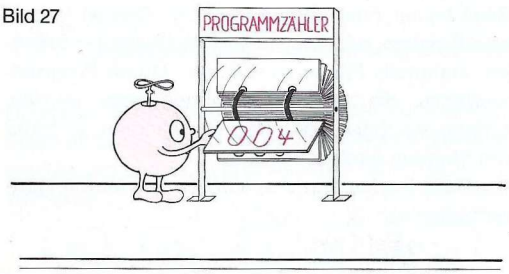
Bild 25

Bild 26



4. Der Programmzähler wird um eins erhöht (Bild 27).

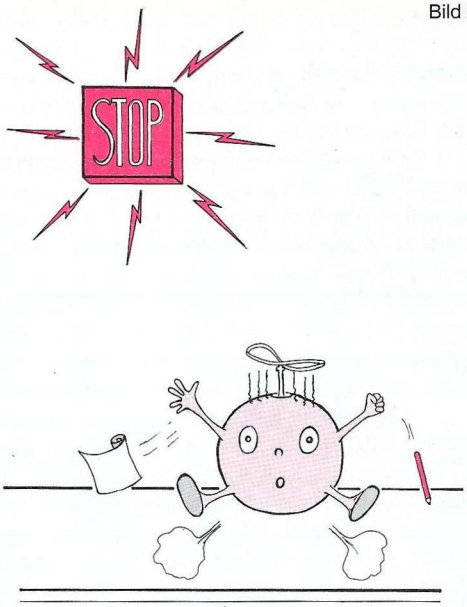
Bild 27



1.34 Der Unterschied zwischen Stop und Halt

Natürlich werden Sie sagen, „Stop" und „Halt" sei doch ein und dasselbe. Und das stimmt selbstverständlich auch, wenn Sie erreichen wollen, daß der Computer anhält.
Warum hat unser Computer aber eine Stop-*Taste* (Funktionstaste) und einen Halt-*Befehl*? Wenn die Stop-Taste gedrückt wird, läßt das Computron buchstäblich alles fallen und unterbricht auf der Stelle die Arbeit, mit der es sich gerade beschäftigt. Es schwingt sich lediglich noch schnell wie der Blitz zur Speicherzelle 000 hinauf, schreibt den Zellen-Inhalt dort ab, notiert ihn auf der Anzeige und . . . setzt sich zur Ruhe.

Bild 28



Warum die Anzeige vom Inhalt der Speicherzelle 000? Diese Funktion wurde nur deshalb eingebaut, damit man verschiedene Computer-Spiele (z. B. das Würfelspiel) leichter spielen kann. Eine weitere Bedeutung hat das nicht.

1.35 Interrupt heißt Unterbrechung

Unterbrechen heißt auf englisch to interrupt. Die Computerleute nennen daher eine Unterbrechung des laufenden Programmes einen **Interrupt**.
Unsere STP-Taste bewirkt also, fachmännisch gesprochen, einen Interrupt. Durch Drücken der RUN-Taste kann der Interrupt wieder aufgehoben werden.

1.36 HLT – das programmierte Ende

Während der Computer auf den Interrupt mit der STP-Taste sofort reagiert, muß ein Halt-Befehl warten, bis er „an der Reihe" ist.
Im Klartext: Wenn z. B. in einem Programm an 16. Stelle ein Halt-Befehl steht, so werden zunächst die 15 vorangehenden Befehle vom Computer abgearbeitet, und er wird erst dann durch den Halt-Befehl angewiesen, anzuhalten.
Ihnen signalisiert der Computer, daß er den Halt-Befehl ausgeführt hat, indem er den Programmzählerstand automatisch auf der Anzeige anzeigt. Und da Sie wissen, daß jeder Befehlszyklus (also auch der des Halt-Befehls) mit dem Weiterschalten des Programmzählers endet, ist auch klar, warum bei

dem Programm aus Kapitel 1.30 „P .017" angezeigt wurde.
In Mnemonics wird der Halt-Befehl als

HLT

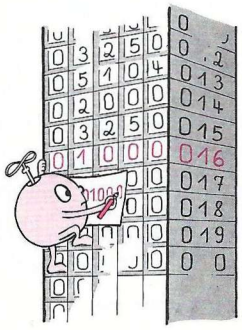
notiert, wobei wie beim ANZ-Befehl der rechte Teil des Befehls ohne Bedeutung ist, denn Halt ist eben Halt. Wir empfehlen Ihnen aber, den Befehl computermäßig stets als

01.000

einzugeben.
Der Befehlsablauf für den Halt-Befehl sieht folgendermaßen aus:

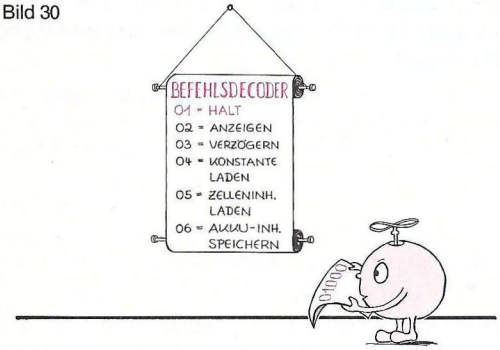
1. Befehl aus dem Speicher holen (Bild 29)

Bild 29



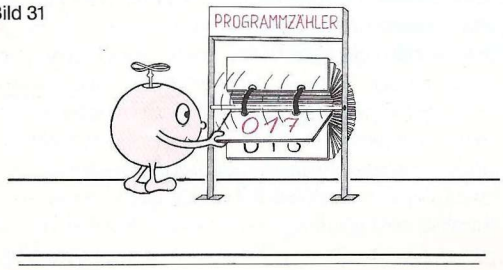
2. Befehl decodieren (Bild 30)

Bild 30



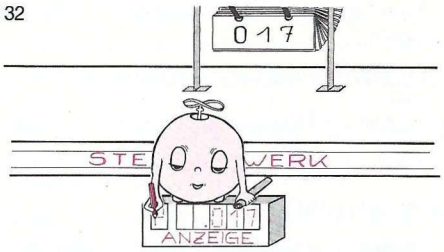
3. Programmzähler weiterschalten (Bild 31)

Bild 31



4. Programmzählerstand auf der Anzeige notieren (Bild 32).

Bild 32



1.37 ABS – vom Akku zur Speicherzelle

Unser Computer kann nicht nur den Inhalt einer beliebigen Speicherzelle in den Akku transportieren (LDA-Befehl), es gibt auch einen Befehl für den umgekehrten Transportweg vom Akku zu einer Speicherzelle. Selbstverständlich muß man dem Computer mitteilen, in welche Speicherzelle (zu welcher Adresse) man den Akku-Inhalt zu bringen wünscht. Der Befehl

06.xxx

bedeutet:

„Speichere den Akku-Inhalt in Speicherzelle xxx"

und mit der Buchstabenkombination

ABS xxx

sollen Sie daran erinnert werden, daß dies der Abspeicher- (oder sprachlich wohl korrekter: der Speicher-) Befehl ist.
Der Befehlszyklus ist dem des Lade-Befehls sehr ähnlich, nur wird hier der Akku-Inhalt abgeschrieben und in die Speicherzelle eingeschrieben.

1.38 AKO – Konstante laden

Und damit Sie mit diesem Befehl auch gleich ein ganz kleines Programm absolvieren können, stellen wir Ihnen sofort noch einen weiteren Befehl vor, der Ihnen später beim Programmieren die Arbeit erleichtern wird. In der Computer-Anweisung

04.xxx

ist xxx keine Adresse, unter der der Computer einen Operanden findet, sondern es ist der Operand selbst, ein beliebiger Zahlenwert zwischen 000 und 255. Wenn Sie also beispielsweise die Zahl 178 in den Akku laden wollen, so müssen Sie diese

Zahl nicht unbedingt vorher in eine Speicherzelle eingegeben haben. Sie schreiben einfach den Befehl „04.178“, lassen ihn ausführen, und schon ist der Akku-Inhalt 00.178. Wir können uns den Befehl so merken

„Lade in den Akku die Konstante xxx“

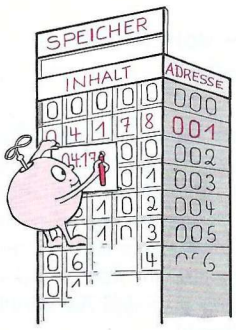
und für **A**kku mit **Ko**nstante laden kürzen wir ab

AKO xxx

Befehlszyklus AKO:

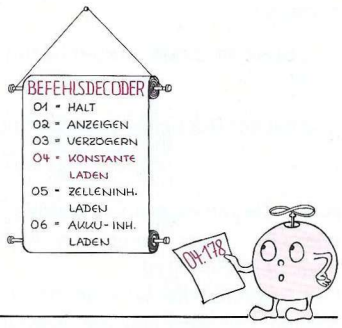
1. Befehl aus dem Speicher holen (Bild 33)

Bild 33



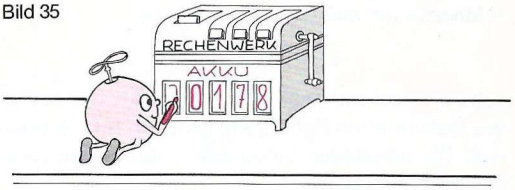
2. Befehl decodieren (Bild 34)

Bild 34



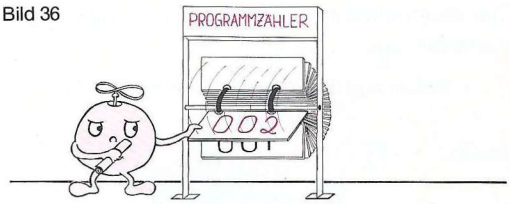
3. Konstante in den Akku schreiben (Bild 35)

Bild 35



4. Programmzähler eins weiterschalten (Bild 36).

Bild 36



Jetzt also das kleine Programm: laden Sie in den Akku eine Konstante, z.B. die Zahl 178, und speichern Sie sie auf den Speicherzellen 100, 101, 102, 103 und 104 ab (Listing 2).

Starten Sie das Programm mit

0	0	1	PC	RUN

Ehe Sie auch nur mit den Augen zwinkern, ist der Computer fertig und zeigt Ihnen seinen Programmzählerstand

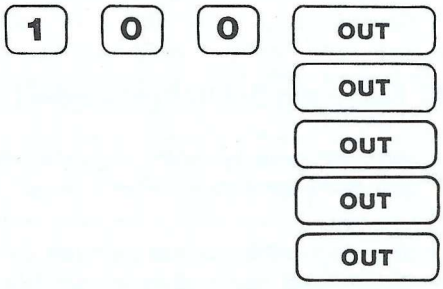
P 00.008

an.

Listing 2: **Akku-Inhalt automatisch speichern**

Adresse	Mnemonics	**Code**	Kommentar
001	AKO 178	**04.178**	In den Akku „178“ laden
002	ABS 100	**06.100**	Akku-Inhalt in Speicherzelle 100 speichern
003	ABS 101	**06.101**	Akku-Inhalt in Speicherzelle 101 speichern
004	ABS 102	**06.102**	Akku-Inhalt in Speicherzelle 102 speichern
005	ABS 103	**06.103**	Akku-Inhalt in Speicherzelle 103 speichern
006	ABS 104	**06.104**	Akku-Inhalt in Speicherzelle 104 speichern
007	HLT	**01.000**	anhalten
100		**00.xxx**	Speicherzellen, in denen der Akku-Inhalt gespeichert wird
101		**00.xxx**	
102		**00.xxx**	
103		**00.xxx**	
104		**00.xxx**	

Überprüfen Sie nun, ob die Speicherzellen 100, 101, 102, 103 und 104 den Wert 00.178 enthalten. Wie man das macht, wissen Sie ja von Kapitel 1.16.



Zum Schluß dieses Kapitels sollten wir der guten Ordnung halber noch auf den entscheidenden Unterschied zwischen dem LDA-Befehl und dem AKO-Befehl hinweisen:
Mit dem LDA-Befehl wird grundsätzlich das, was in einer Speicherzelle steht, in den Akku geladen – gleichgültig, ob sich in der Zelle ein Befehl oder ein Datum befindet. Mit dem AKO-Befehl kann ausschließlich ein Zahlenwert, niemals also ein Befehl, in den Akku geladen werden.

1.39 Rechnen und Daten verarbeiten

Wir werden mit Ihnen jetzt ein Programm vorbereiten, bei dem der Computer Rechenaufgaben lösen muß. Das mag Sie verwundern, denn wir hatten ja jede Konkurrenz mit dem Taschenrechner strikt abgelehnt. Aber kann man dann eigentlich einen Computer, der nicht mindestens die fünfte Wurzel aus der dritten Potenz einer siebenstelligen Zahl berechnen kann, überhaupt als richtigen Computer bezeichnen?
Selbstverständlich kann man das. Sicherlich haben Sie schon einmal den Begriff „elektronische Datenverarbeitung" gehört (abgekürzt EDV). Eine EDV-Anlage enthält natürlich auch einen Computer. Aber einen, der vorwiegend Daten verarbeitet.
Als Daten bezeichnet man Informationen aller Art. Also z.B. Zahlenwerte. Oder aber auch Informationen darüber, ob eine Lichtschranke unterbrochen ist, ob ein Taster gedrückt oder ein Schalter betätigt wurde. Ob es warm oder kalt, Tag oder Nacht, trocken oder feucht ist, ob sich ein Gegenstand schnell oder langsam bewegt, viel oder wenig Licht reflektiert, magnetisch oder unmagnetisch ist usw. usw. Alle diese Daten können elektrisch so aufbereitet werden, daß sie in den Computer eingegeben werden können. Sie werden das an einem Beispiel gleich erfahren.

1.40 Das Parkhausproblem

Die Firma Autopause GmbH & Co. KG betreibt ein Parkhaus mit 100 Stellplätzen, das sie mit Hilfe eines Computers automatisiert hat. Am Eingang steht eine Ampel, die computergesteuert grünes Licht zeigt, solange noch Plätze frei sind. Fährt ein Auto hinein, so wird zu der Menge der geparkten Autos eine „1" addiert. Verläßt ein Auto das Parkhaus, so wird „1" subtrahiert.
Damit die Firma abends weiß, was sie eingenommen hat, wird außerdem separat die Anzahl der an einem Tag durch den Eingang gefahrenen Autos durch fortlaufende Additionen registriert.
So ganz nebenbei kann der Computer auch die Parkdauer aus Parkbeginn und Parkende durch eine Subtraktion ausrechnen und nach Multiplikation mit dem Stundensatz dem Kunden anzeigen, was er zu zahlen hat.
Der Computer steuert außerdem die Schranke am Ein- und Ausgang.
Eine solche Parkhaus-Abwicklung bezeichnet man als einen technischen **Prozeß.** Dieser Prozeß wird durch den Computer vollautomatisch gesteuert. Man nennt diesen Computer daher auch **Prozeßrechner.** Der KOSMOS-Computer wäre dank seiner vielen Anschlußklemmen an der Rückseite durchaus (in bescheidenem Rahmen natürlich) in der Lage, einen solchen Prozeß zu steuern. **Er ist ein kleiner Prozeßrechner.**
Was wir Ihnen mit diesem Beispiel zeigen wollten: unser Computer braucht selbstverständlich auch arithmetische (Rechen-) Befehle. Aber seine Stärke liegt in der Prozeß-Steuerung, nicht im Abspulen von mathematischen Funktionen.
Nach dieser grundsätzlichen Klärung sollen nun rasch der Additions- und der Subtraktions-Befehl besprochen werden.

1.41 ADD – der Additionsbefehl

Das einzig Überraschende an diesem ersten arithmetischen Befehl mag sein, daß das, was addiert werden soll, also die beiden Summanden, im Befehl gar nicht enthalten sind. Der Computer muß sie sich erst zusammensuchen.
Einen Summanden findet er im Akku. Genauer gesagt: der Programmierer muß dafür sorgen, daß der erste Summand im Akku ist. Der *zweite* Summand ist in einer beliebigen Speicherzelle gespeichert. Es gilt also, beim Programmieren darauf zu achten, daß Summanden, die im Zuge eines Programmes benötigt werden, im Speicher „hinterlegt" werden.
Dem Computer wird dann im Additions-Befehl nur noch die Adresse mitgeteilt, unter der er den zwei-

ten Summanden findet. Mit der Buchstaben-Abkürzung ADD lautet der Befehl

ADD xxx

Seine Wirkung kann man so beschreiben:

„Addiere *zum* Akku-Inhalt den Wert, der in Speicherzelle xxx gespeichert ist. Das Additionsergebnis steht im Akku."

Der Zahlenschlüssel für die Addition ist 07. Computergerecht heißt der Befehl also

07.xxx

Wenn Sie in einem Programm den Additions-Befehl einsetzen, müssen Sie genau überprüfen, ob das Ergebnis der Addition immer kleiner als 256 bleibt, der Zahlenbereich des Computers also nicht überschritten wird. Der Computer ist unbestechlich: entsteht bei einem Programmlauf durch Addition eine Zahl, die größer als 255 ist, so bricht der Computer das Programm ab und zeigt F .006 an. In einem solchen Fall können Sie durch Drücken der PC-Taste feststellen, bei welchem Programmschritt der Fehler aufgetreten ist und gegebenenfalls Ihr Programm korrigieren.
Ablauf des Befehlszyklus ADD:

5. Ersten Summanden aus dem Akku holen und Addition durchführen (Ergebnis erscheint im Akku).
6. Programmzähler weiterschalten (Bild 37).

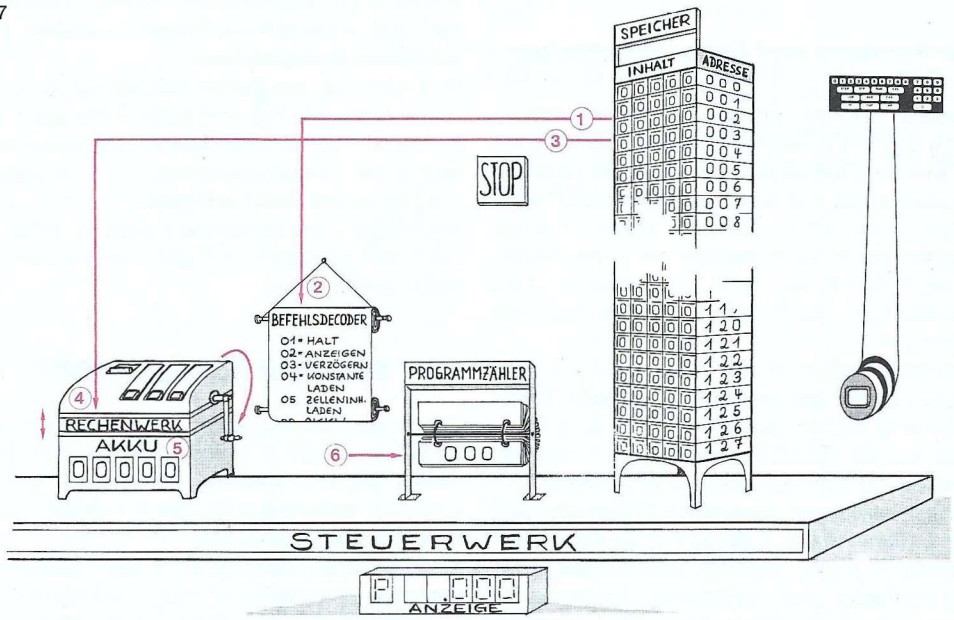
1.42 SUB – der Subtraktionsbefehl

Der zweite arithmetische Befehl, die Subtraktion, läuft nach exakt demselben Schema ab wie die Addition:

„Subtrahiere *vom* Akku-Inhalt den Wert, der in Speicherzelle xxx gespeichert ist. Das Subtraktions-Ergebnis steht dann im Akku".

Auch hier müssen Sie als Programmierer dafür sorgen, daß die Zahl, von der etwas abgezogen werden soll (der Minuend), zuvor in den Akku transportiert wird und daß die Zahl, die abgezogen wird (der Subtrahend), in einer Speicherzelle „hinterlegt" wird.
Sollten Sie bei der Entwicklung Ihres Programmes übersehen haben, daß man eine große Zahl nicht von einer kleineren abziehen kann, so wird der Computer das Programm abbrechen und Sie durch die Fehleranzeige F .006 dezent auf Ihre Eselei aufmerksam machen. Drücken Sie die Taste PC, und Sie werden sofort wissen, an welcher Stelle der Fehler aufgetreten ist.

Bild 37



1. Befehl aus dem Speicher holen.
2. Befehl decodieren.
3. Zweiten Summanden von Speicherzelle xxx abschreiben.
4. Zweiten Summanden ins Rechenwerk eingeben.

1.43 Ein Automatik-Zähler

Erinnern Sie sich an unser Gedankenspiel „Computer-gesteuertes Parkhaus?" Dort werden durch Addition und Subtraktion Autos abgezählt. Wir haben jetzt für Sie ein Programm bereit, das selbsttätig von 000 bis 255 zählt. Was dabei abgezählt werden soll, überlassen wir Ihrer Fantasie, es spielt auch gar keine Rolle. Wir nennen das Programm schlicht „automatischer Zähler mit Digitalanzeige".
Wie würden Sie vorgehen, wenn Sie das Programm entwerfen müßten? Sie würden sicher zuerst in den Akku eine „1" laden und für eine gewisse Dauer anzeigen lassen (Befehle LDA, ANZ und VZG). Dann sollte man eine „1" addieren, wieder den Akku-Inhalt (der nach der Addition „2" ist) anzeigen und verzögern. Das sähe dann so aus:

001 Lade in den Akku eine „1"
002 Zeige den Akku-Inhalt an
003 Verzögern
004 Addiere zum Akku-Inhalt eine „1"
005 Zeige den neuen Akku-Inhalt an
006 Verzögern
007 Addiere ein weiteres Mal eine „1"
(1. Wiederholung)
008 Zeigen den neuen Akku-Inhalt an
009 Verzögern
010 Addiere ein weiteres Mal eine „1"
(2. Wiederholung)
usw.

Wenn man diese Prozedur 253mal wiederholen würde, könnte man bis 255 zählen. Aber – der Speicherplatz würde bei weitem nicht ausreichen. Wir haben ja nur 128 Speicherzellen zur Verfügung.
Was tun?

1.44 SPU – der Sprung hilft weiter

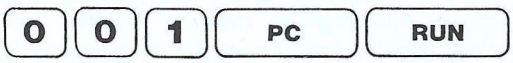
Wir befehlen dem Computer (mit einem Befehl, den wir gleich kennenlernen werden), die normale Programmreihenfolge zu verlassen und nach dem Verzögerungsbefehl in Zelle 006 zum Additionsbefehl in Zelle 004 *zurückzukehren,* Addition, Anzeige und Verzögerung auszuführen, wieder zurückzukehren usw.
Sie erkennen natürlich sofort den Vorteil: Programmteile, die sich ständig wiederholen, müssen nur *einmal* in den Speicher eingegeben werden. Die Befehlsfolge ADD – ANZ – VZG wiederholt der Computer immer wieder bis . . . nun, bis durch die fortlaufenden Additionen im Akku der Wert 255 steht, und der Computer ein weiteres Mal eine „1" addieren soll. Er wird jetzt das Programm abbrechen und Ihnen durch die Fehleranzeige F .006 signalisieren, daß nunmehr sein Zahlenbereich überschritten wird.
Der Befehl, mit dem wir den Computer zum Verlassen der normalen Befehlsreihenfolge bringen können, heißt **Sprungbefehl.** Der Operationscode ist 09, das Buchstaben-Kürzel SPU (**Sp**ringe **u**nbedingt).

09.xxx bzw. SPU xxx

heißt, daß der Computer das Programm mit dem Befehl fortsetzen soll, der auf Adresse xxx steht.
SPU steht als Abkürzung für „Unbedingter Sprung", der Befehl wird also ohne jede Einschränkung durchgeführt. Sie ahnen schon: es gibt noch einen „Bedingten Sprung", bei dem der Computer nur unter bestimmten Bedingungen einen Sprung ausführt. Aber davon später.
Bevor das Zählprogramm eingegeben wird, schauen wir uns rasch den Befehlszyklus an:

1. Befehl aus dem Speicher holen
2. Befehl decodieren
3. Programmzähler auf die im Befehl angegebener Adresse einstellen (Bild 38).

Starten Sie das Programm (Listing 3) durch

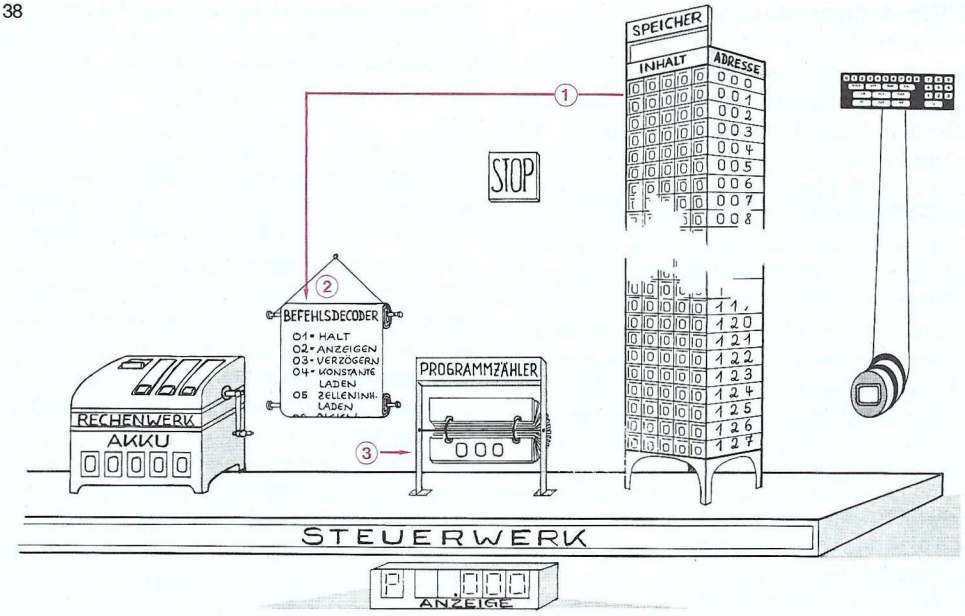


Es hält nach der Anzeige A 00.255 mit der Fehlermeldung F .006 an, da dann der Zahlenbereich überschritten wird.

Listing 3: **Automatischer Zähler mit Digitalanzeige**

Adresse	Mnemonics	**Code**	Kommentar
001	LDA 100	**05.100**	Inhalt von Zelle 100 in den Akku laden
002	ANZ	**02.000**	Akku-Inhalt anzeigen
003	VZG 250	**03.250**	250 ms verzögern
004	ADD 100	**07.100**	Zum Akku-Inhalt den Inhalt von Zelle 100 addieren
005	ANZ	**02.000**	Akku-Inhalt anzeigen
006	VZG 250	**03.250**	250 ms verzögern
007	SPU 004	**09.004**	Zum Addieren auf Adresse 004 springen
100		**00.001**	Schrittweite

Bild 38



1.45 Die richtige Speichereinteilung

Vielleicht ist es Ihnen beim letzten Programmlisting bereits aufgefallen: Wir verwenden einen Teil des Speichers für Befehle und einen anderen für Daten. Der Speicherbereich mit den Daten – wir werden in Zukunft vom Datenbereich sprechen – ist in einem gesonderten Kasten unter dem Listing angeordnet. Die Einteilung des Speichers in einen **Befehlsbereich** und einen **Datenbereich** ist Sache des Programmierers. Wir empfehlen zunächst einmal, die *Befehle* in die Adressen 001 bis 099 und *Daten* ab Adresse 100 einzugeben. Andere Einteilungen wären durchaus möglich und manchmal auch notwendig, dem Speicher ist es vollkommen egal, wo Befehle und wo Daten stehen. Man sollte jedoch Befehle und Daten niemals „mischen". Sonst kann es nämlich passieren, daß bei der Abarbeitung eines Programmes ein Datum zum Befehlsdecoder gebracht wird. Sie erinnern sich: ein Datum beginnt mit 00, und das ist kein gültiger Operations-Code. Der Computer würde prompt reagieren und Ihnen die Fehleranzeige F .002 präsentieren.

Natürlich kann ein automatischer Zähler nicht nur 1, 2, 3, 4, 5 . . . 255 zählen, sondern z.B. auch 2, 4, 6 . . . 254 oder 3, 6, 9, 12 . . . 255. Die „Zählsprünge" nennt man in der Computer-Fachsprache auch **Schrittweite.** Bei unserem Zählprogramm hatten wir eine Schrittweite von „1" vorgesehen.

Aufgaben:
Wie muß das Programm für die Schrittweite „2" verändert werden?
Sie können aber nicht nur die Schrittweite, sondern auch die Zählgeschwindigkeit variieren.
Welche Befehle müssen verändert werden, damit der Zähler schneller zählt? (Lösungen auf Seite 30.)

1.46 Computer-Logik

Pessimistische Zeitgenossen neigen dazu, warnend den Finger zu erheben und ein düsteres Bild von einer Zukunft zu entwerfen, in der der Computer Entscheidungen trifft, denen sich der Mensch unterzuordnen hat.

Tatsache ist, daß es oft den Anschein hat, als könne ein Computer wie ein Mensch denken. Man stellt ihm eine Frage (man gibt ihm bestimmte Daten ein), und er gibt eine vollkommen logische Antwort (die er anzeigen oder ausdrucken kann). Dies ist für jeden, der das einmal mitgemacht hat, zunächst außerordentlich erstaunlich. Sie sollten sich jedoch nicht allzu sehr verblüffen lassen, sondern versuchen, die Dinge zu durchschauen. Wir geben Ihnen deshalb ein sehr einfaches Beispiel von „Computer-Logik".

Zwei Wanderer gelangen an eine Weggabelung, von der ein Weg nach A-Dorf und der zweite nach B-Dorf führt. Da sie einander überdrüssig sind, trennen sie sich, und der eine marschiert in Richtung A-Dorf weiter.

Fragen wir den Computer, welche Entscheidung er für den anderen Wanderer treffen würde. Da er „logisch" denkt, wird er antworten: „Der andere geht nach B-Dorf".
Sie meinen, das habe mit „Denken" gar nichts zu tun? Richtig, es geht hier nur um die Auswahl zwischen zwei Möglichkeiten. Wenn die eine Möglichkeit ausgeschlossen wird, bleibt nur die zweite. Das zum Beispiel ist Computer-Logik.
Lassen Sie uns noch ein zweites Beispiel erfinden. In der Firma Gaumenfreuden OHG soll der Lehrling Pralinen abpacken. Immer 25 Stück in einen Karton.
Nach jeder Praline, die er in einen Karton eingelegt hat, muß er eine „Entscheidung" treffen: entweder eine weitere Praline in denselben Karton oder mit einem neuen Karton weitermachen? Als Entscheidungsgrundlage dient ihm die Anweisung des Chefs, daß jeder Karton 25 Pralinen zu enthalten habe. Ist also die Zahl 25 erreicht, muß ein neuer Karton genommen werden.
Wir wollen jetzt statt des Lehrlings unseren Computer Pralinen zählen und ihn genau wie den Lehrling „Entscheidungen" fällen lassen.

1.47 VGL – vergleichen und Entscheidungen treffen

Um es klipp und klar zu sagen: das Prinzip der Computer-Entscheidung beruht darauf, zwei Zahlen (davon eine im Akku und die andere in einer Speicherzelle) miteinander zu vergleichen und dann in Abhängigkeit vom Vergleichsergebnis in der normalen Programmreihenfolge fortzufahren oder zu einem anderen Programmteil zu springen.
Der erste Vergleichsbefehl, den Sie kennenlernen sollen, läßt den Computer prüfen, ob zwei Zahlen *gleichgroß* sind:

> „Prüfe, ob der Akku-Inhalt gleichgroß ist wie der Inhalt der Speicherzelle xxx. Wenn ja, merke Dir dies".

Wie kann ein Computer sich ein Prüfergebnis merken?
Er hat dafür eine winzig kleine Merk-Speicherzelle, die von Computerleuten als **Flag** bezeichnet wird (englisch flag = Fahne, Wimpel). In dieser Minizelle kann entweder eine „1" (Vergleich ergab Gleichheit der Zahlen) oder „0" (Zahlen waren ungleich) eingeschrieben werden. Fachleute sagen auch: das Flag ist gesetzt (1) oder das Flag ist nicht gesetzt (0).
Das Flag, den Merker, kann man von außen nicht sehen. Man braucht das auch gar nicht, denn es kommt einzig und allein darauf an, daß der Computer beim nächsten Befehl (den Sie gleich kennenlernen werden), aus dem Inhalt des Flags ablesen kann, wie er im Programm fortfahren soll.
Übrigens, der Vergleich wird im *Rechenwerk* durchgeführt: es kann also nicht nur „rechnen" sondern auch logische Operationen ausführen. Wie Sie im Bild 39 erkennen können, fährt auf dem Rechenwerk ein Schieber mit der Aufschrift „ja" heraus, wenn der Vergleich Übereinstimmung ergab. Dieser Schieber symbolisiert also unser Flag.

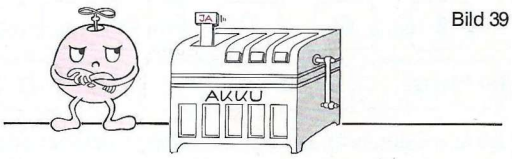


Bild 39

Der Prüfbefehl auf Gleichheit wird computermäßig notiert als

10.xxx

Dabei ist unter der Adresse xxx die Zahl zu finden, die mit dem Akku-Inhalt verglichen werden soll.
Mnemonisch schreiben wir den Befehl als

VGL xxx (**V**ergleiche, ob **gl**eich)

Der Befehlszyklus läuft folgendermaßen ab:

1. Befehl aus dem Speicher holen
2. Befehl decodieren
3. Inhalt der Speicherzelle xxx kopieren
4. Kopie in das Rechenwerk eingeben
5. Vergleich mit Akkuinhalt im Rechenwerk durchführen, ggf. Flag setzen
6. Programmzähler um eins weiterschalten (Bild 40).

1.48 SPB – Sprung unter Vorbehalt

Der Vergleichsbefehl allein ist genauso wenig wert wie die Frage, ob es draußen regnet, wenn man keinen Regenschirm besitzt.
Das Ergebnis des Vergleichs muß den weiteren Programmlauf beeinflussen, also Konsequenzen haben. Man benutzt dazu den **Bedingten Sprungbefehl.** Wir haben in Kapitel 1.44 den Unterschied zum Unbedingten Sprung schon angedeutet.
Der Bedingte Sprungbefehl bewirkt im Computer folgendes:

> „Wenn der Vergleich Übereinstimmung ergab (wenn das Flag gesetzt ist), setze das Programm mit dem Befehl auf Speicherzelle xxx fort (springe auf Speicherzelle xxx)

und lösche das Flag. War keine Übereinstimmung (Flag nicht gesetzt), so fahre mit dem Programm in der normalen Reihenfolge fort."

Lösungen (zu S. 28)

Soll der Zähler 2, 4, 6, 8, ... 254 zählen (Schrittweite 2), so ist in Speicherzelle 100 der Wert 00.002 einzugeben
also:

Soll der Zähler schneller zählen, so sind die Verzögerungsbefehle in den Speicherzellen 003 und 006 zu verändern (z.B. VZG 100 für Zählgeschwindigkeit von 1/10 Sekunden).
Soll der Zähler z.B. im Einsekundentakt zählen, so müssen jeweils vier Verzögerungsbefehle VZG 250 aufeinander folgen. Das gesamte Programm verschiebt sich dann nach unten, und die Sprungadresse im Sprungbefehl ändert sich (bei vier VZG-Befehlen auf den Adressen 003, 004, 005 und 006 müßte der Sprungbefehl SPU 007 heißen).

Bild 40

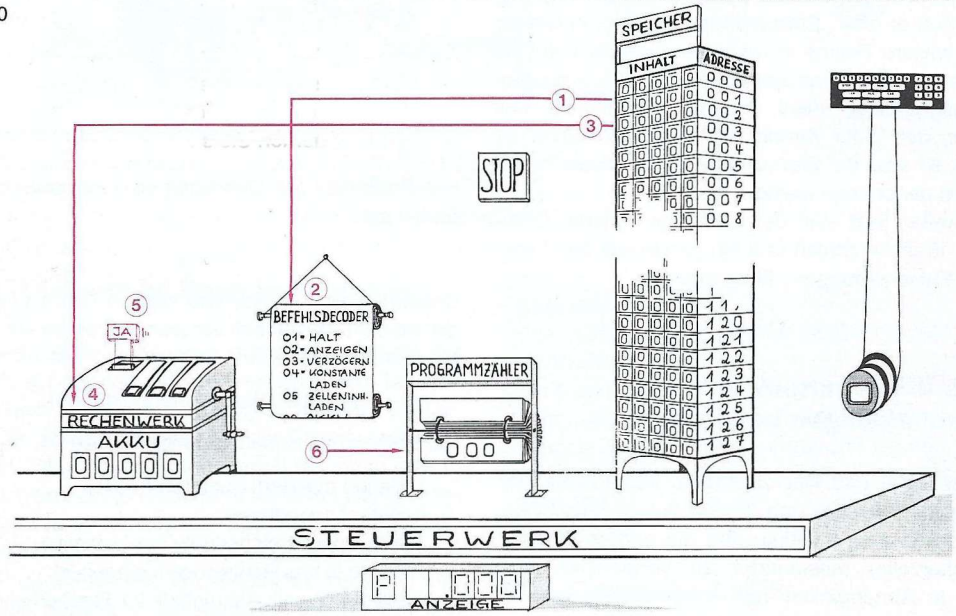
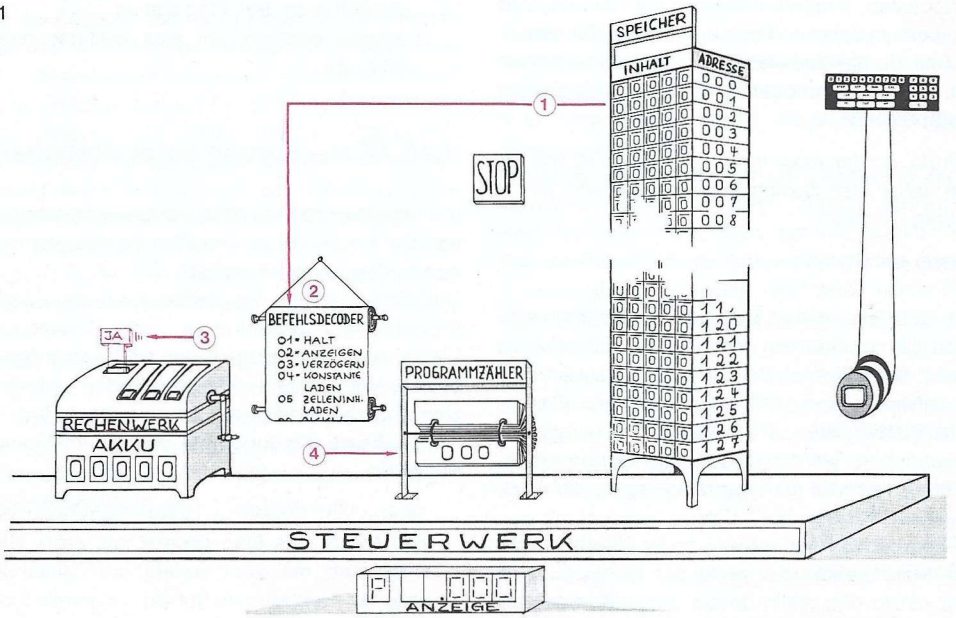


Bild 41



Das ist nun das ganze Geheimnis einer „Computer-Entscheidung": Vergleich zweier Zahlenwerte und anschließende „Verzweigung" des Programms. Das Thema Programmverzweigung wird uns auch in Kapitel 1.50 noch weiter beschäftigen.
Der Bedingte Sprungbefehl lautet

11.xxx bzw. SPB xxx (**Sprung** **b**edingt)

Der Befehlszyklus für diesen Befehl besteht aus vier Schritten:

1. Befehl aus dem Speicher holen
2. Befehl decodieren
3. Flag (Merker, Schieber) prüfen
4. Gegebenenfalls Programmzähler auf den Wert xxx einstellen, sonst um eins erhöhen (Bild 41).

1.49 Der unendliche Zähler

Das angekündigte, entscheidungsträchtige Programm kann jetzt leicht entwickelt und erklärt werden. Wir lassen wiederum den Computer zählen, indem wir ständig eine „1" addieren. Nach jeder Addition soll der Computer prüfen, ob die Zahl „25" schon erreicht ist. Die Vergleichszahl, in unserem Falle also 25, muß zu diesem Zweck natürlich in einer Speicherzelle des Datenbereiches hinterlegt sein.
Der Bedingte Sprungbefehl bringt es anschließend an den Tag. Waren Akku-Inhalt und Speicherzellen-Inhalt ungleich (Flag nicht gesetzt) – das wird 24mal der Fall sein – so muß das Programm mit einer Addition fortgesetzt werden. Sind die Inhalte gleich (Flag gesetzt), so muß der Computer wieder bei 1 anfangen zu zählen (Sprung auf den Anfang des Programms).

Wenn Sie Zweifel haben, ob Ihr Programm richtig im Speicher steht, empfiehlt es sich stets, alle Befehle mit dem Listing zu vergleichen. Wählen Sie dazu zunächst die Speicherzelle 001 durch Drükken von 001-OUT an und betätigen Sie dann fortlaufend OUT. Entdecken Sie einen fehlerhaften Befehl, so tippen Sie den richtigen Wert ein und drükken die INP-Taste. Wenn Sie einen Befehl vergessen oder zweimal eingegeben haben, geben Sie am besten das ganze Programm neu ein (Weitere Hinweise zur Fehlersuche siehe Kapitel 1.71).

Sie starten das Programm wie gewohnt mit

und werden feststellen, daß der Computer nur bis 25 zählt und dann wieder bei 0 beginnt. Aber – vielleicht bemerken Sie's mit Schrecken – er hört nie wieder auf (es sei denn, Sie würden auf die STP-Taste drücken). Wenn wir in Gedanken noch einmal zu unserem Lehrling zurückkehren, der Pralinen 25stückweise in Kartons verpackt, so wäre er durch unser Programm bis an sein Lebensende zu dieser Arbeit verdammt.

Aufgabe:
Helfen Sie ihm und versuchen Sie, das Programm so abzuändern, daß er nach 50 Kartons aufhören darf. Schreiben Sie noch ein paar Befehle dazu, so daß der Computer 50mal bis 25 zählt und dann durch einen Halt-Befehl anhält. Die Lösung haben wir für alle Fälle für Sie auf Seite 32 abgedruckt.

Listing 4: **Unendlicher Zähler**

Adr.	Mnemonics	**Code**	Kommentar
001	LDA 100	**05.100**	Inhalt von Zelle 100 in den Akku laden
002	ANZ	**02.000**	Akku-Inhalt anzeigen
003	VZG 250	**03.250**	250 ms verzögern
004	ADD 100	**07.100**	Zum Akku-Inhalt Inhalt von Zeile 100 addieren
005	ANZ	**02.000**	Akku-Inhalt anzeigen
006	VZG 250	**03.250**	250 ms verzögern
007	VGL 101	**10.101**	Akku-Inhalt mit Inhalt von Zelle 101 vergleichen
008	SPB 001	**11.001**	Wenn Gleichheit dann auf 001 springen
009	SPU 004	**09.004**	Wenn nicht Gleichheit, auf 004 springen
100		**00.001**	Schrittweite
101		**00.025**	Vergleichszahl

Listing 5: **Zähler, der 50 mal bis 25 zählt**

Adresse	Mnemonics	**Code**	Kommentar
001	LDA 100	**05.100**	Inhalt von Zelle 100 in den Akku laden
002	ANZ	**02.000**	Akku-Inhalt anzeigen
003	VZG 250	**03.250**	250 ms verzögern
004	ADD 100	**07.100**	Zum Akku-Inhalt den Inhalt von Zelle 100 addieren
005	ANZ	**02.000**	Akku-Inhalt anzeigen
006	VZG 250	**03.250**	250 ms verzögern
007	VGL 101	**10.101**	Akku-Inhalt mit Inhalt von Zelle 101 vergleichen
008	SPB 010	**11.010**	Wenn Gleichheit, dann auf 010 springen
009	SPU 004	**09.004**	Wenn nicht Gleichheit, dann auf 004 springen
010	LDA 103	**05.103**	Inhalt von Zelle 103 in den Akku laden
011	ADD 100	**07.100**	Zum Akku-Inhalt den Inhalt von Zelle 100 addieren
012	ABS 103	**06.103**	Akku-Inhalt in Zelle 103 speichern
013	VGL 102	**10.102**	Akku-Inhalt mit dem Inhalt von Zelle 102 vergleichen
014	SPB 016	**11.016**	Wenn Gleichheit, dann auf 016 springen
015	SPU 001	**09.001**	Wenn nicht Gleichheit, dann auf 001 springen
016	HLT	**01.000**	Anhalten
100		**00.001**	Schrittweite
101		**00.025**	1. Vergleichszahl
102		**00.050**	2. Vergleichszahl
103		**00.xxx**	Speicherzelle für Zwischenergebnisse

Sie müssen beachten, daß Sie im Datenbereich des Speichers die Zahlen „1" (Schrittweite), „25" (1. Vergleichszahl) und „50" (2. Vergleichszahl) hinterlegen. Außerdem muß eine Zelle als „Merkzelle" freigehalten und *vor jedem Programmstart* mit „0" gefüllt werden. In diese „Merkzelle" schreibt der Computer während des Programmlaufs hinein, wie oft die Zahl „25" erreicht wurde.

1.50 Zwischen uns und dem Computer: das Flußdiagramm

Wenn Sie die Lösung für den Pralinenlehrling selbst gefunden haben, dürfen Sie sich (und wir Sie) beglückwünschen. Sie ist nämlich in der Tat ziemlich verzwickt. Wenn nicht, so ist das kein Beinbruch, dann wird die nachfolgende Betrachtung über grafische Darstellungen von Programmabläufen sicherlich besonders hilfreich sein.

Wie kommt man von einer Aufgabenstellung zu einem Computer-Programm? Der einfachste Weg führt – wie bei allen Problemen auf dem Gebiet der Technik – über eine Zeichnung. Am einfachsten ist die Sache natürlich bei einem Programm, das ohne Sprünge Befehl für Befehl der Reihe nach abarbeitet und auf einem Halt-Befehl stehen bleibt. Ein solches Programm heißt **linear.**

Stellen Sie sich bitte vor, Sie müssen Ihren Wagen zur Inspektion geben. Wenn Sie sich vor unliebsamen Überraschungen schützen wollen, folgen Sie dem Rat der Automobilclubs und geben der Werkstatt ganz klare Anweisungen, was am Auto gemacht werden soll.

In Form einer Zeichnung könnte die Anweisung z. B. so aussehen:

Bild 42

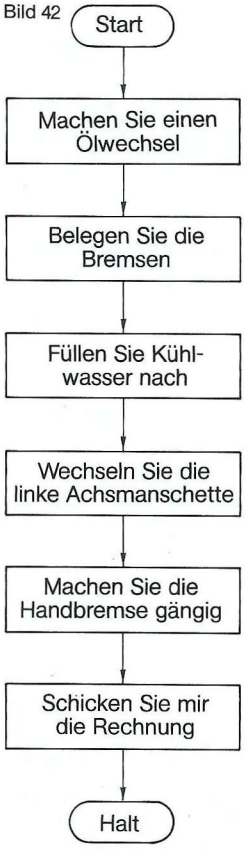
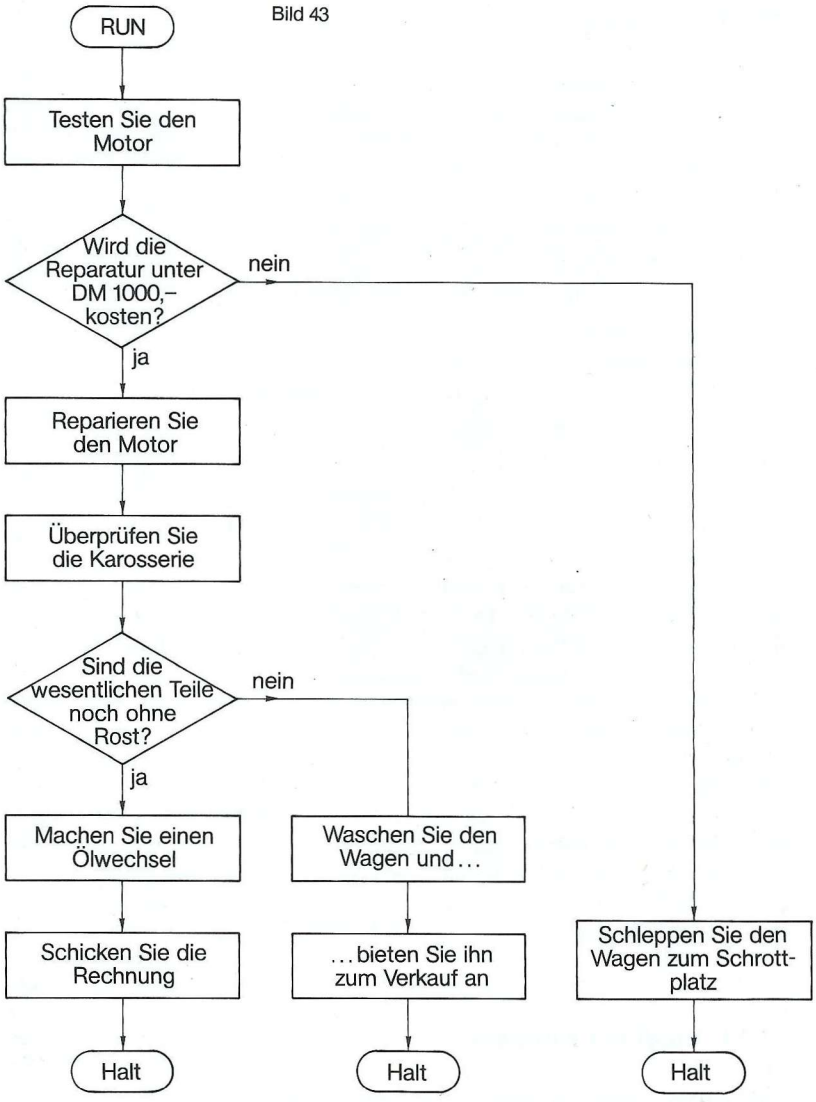


Bild 43

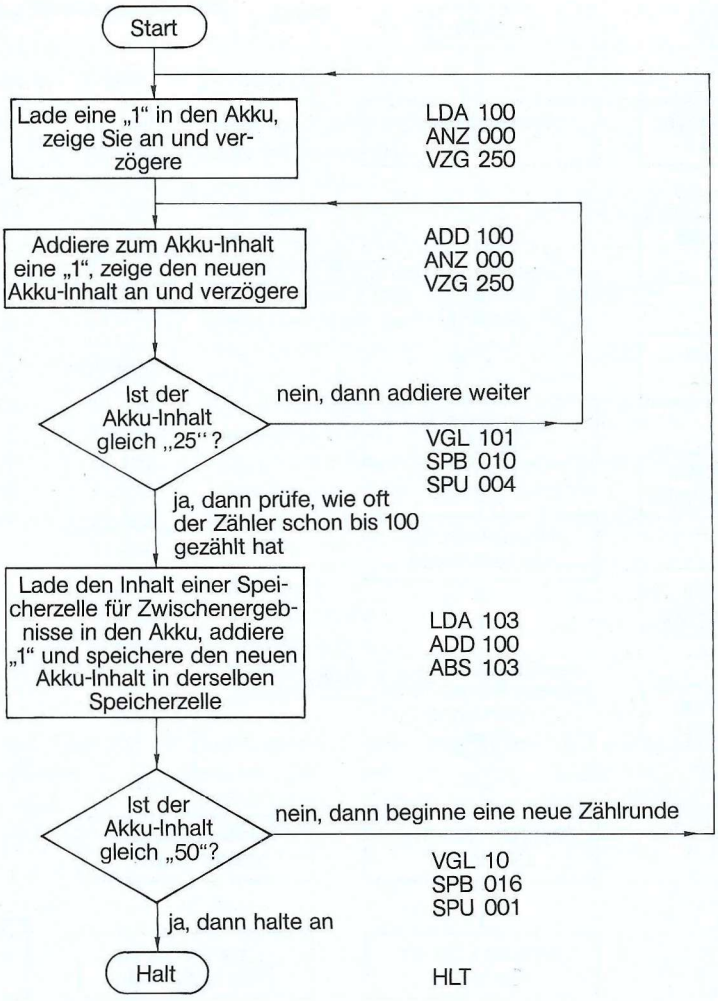


Schwieriger wird die Sache schon, wenn sie feststellen, daß der Motor seinen Geist aufgegeben hat und der Allgemeinzustand des Autos ohnehin zu wünschen übrig läßt. Ihr Auftrag an die Werkstatt könnte dann so aussehen wie in Bild 43 oben rechts.

Sie sehen, daß es an den Stellen, wo die Rauten eingezeichnet sind, zwei Möglichkeiten gibt, im (Werkstatt-) Programm fortzufahren. Die Rauten sind **Verzweigungspunkte** des Programmes. Man bezeichnet es daher auch als **verzweigtes Programm.** Bei einem Computer können Vergleichsbefehle eine Programmverzweigung zur Folge haben.

Die zeichnerische Darstellung eines Computerprogramms nennt man **Flußdiagramm.** Das Flußdiagramm zeigt den logischen Ablauf eines Programmes. Hat man eine Problemstellung in ein Flußdiagramm umgesetzt, ist es ein leichtes, die einzelnen Funktionsblöcke des Diagramms durch einen oder mehrere Computerbefehle zu ersetzen. Wie hilfreich ein Flußdiagramm sein kann, werden Sie erkennen, wenn Sie sich die Aufgabe des Zählers bis 25, der den Zählvorgang 50mal durchführen soll, noch einmal vergegenwärtigen. Schauen Sie sich das nachfolgende Flußdiagramm an, und Sie werden das Programm nun ohne Mühe verstehen können.

Bild 44



1.51 Spaß am Knobeln

Wir können Ihnen an dieser Stelle feierlich versichern, daß der Spaß am Computer mit dem Kennenlernen von verzweigten Programmen jetzt erst so richtig beginnt. Das Austüfteln von Programmlösungen mit Verzweigungen, die sich wiederum verzweigen, dann noch einmal verzweigen usw. ist spannender als ein Krimi, anregender als eine Patience, lehrreicher als ein Mengenlehrebuch und amüsanter als ein Gesellschaftsspiel. Darin ist wohl auch der Grund zu sehen, warum sich immer mehr Menschen dem faszinierenden Computer-Hobby zuwenden, Computer-Clubs gründen, untereinander Programme und Erfahrungen austauschen und einfach Freude daran haben, daß ein Programm, das sie entwickelt haben, vom Computer gehorsam abgearbeitet wird.
Zwei weitere Vergleichsbefehle sollen Sie daher gleich kennenlernen, die wiederum eine Fülle von neuen Programmiermöglichkeiten erschließen.

„Prüfe, ob der Akku-Inhalt *größer* als der Inhalt der Speicherzelle xxx ist. Wenn ja, merke Dir dies."

Die Logik dieses Befehls ist für Sie kein Problem mehr, sie ist identisch mit der des VGL-Befehls. Daher beschränken wir uns darauf, Ihnen Code und Mnemonic anzugeben:

12.xxx bzw. VGR xxx (**V**ergleiche, ob **gr**ößer)

Der letzte Vergleichsbefehl lautet:

„Prüfe, ob der Akku-Inhalt *kleiner* ist als der Inhalt der Speicherzelle xxx. Wenn ja, merke Dir dies."

Dieser Befehl wird so notiert:

13.xxx bzw. VKL xxx (**V**ergleiche ob **kl**einer)

Beide Befehle bleiben völlig wirkungslos, wenn Sie nicht unmittelbar oder später einen Bedingten Sprung SPB anfügen.

Aufgabe:
Haben Sie Lust, ein bißchen zu knobeln? Schreiben Sie das Würfelprogramm von Kapitel 1.15 um. Benutzen Sie statt des VGL-Befehls einmal den VGR- und einmal den VKL-Befehl. Ein kleiner Tip: es gibt eine ganze Reihe richtiger Lösungen. Eventuell müssen Sie die Vergleichszahl in Speicherzelle 100 verändern. Lösungsvorschläge finden Sie auf Seite 36.

1.52 Mensch und Computer

Nicht zu Unrecht wird der Computer immer wieder mit dem Menschen verglichen, und in vielen Punkten ist der Vergleich auch gar nicht falsch. Auch der Mensch kann Daten verarbeiten. Für die Dateneingabe verfügt er über seine fünf Sinne, die Datenverarbeitung besorgt das Gehirn, und zur Ausgabe der Ergebnisse benutzt er die Stimme, die Arme, die Beine usw.
Eine EDV-Anlage ist grundsätzlich nach demselben Schema aufgebaut: sie besitzt eine oder mehrere Eingabe-Einheiten (Kartenleser, Tastatur, Magnetbandgerät), eine Verarbeitungseinheit (die sogenannte Zentral-Einheit mit Speicher, Rechenwerk und Steuerwerk) und ein oder mehrere Ausgabe-Einheiten (Drucker, Stanzer, Bildschirm, Magnetband oder -platte. Aber wir sollten uns nicht täuschen. Die Unterschiede zwischen Mensch und Computer sind beträchtlich. Die wichtigsten haben wir in einer kleinen Gegenüberstellung zusammengefaßt.

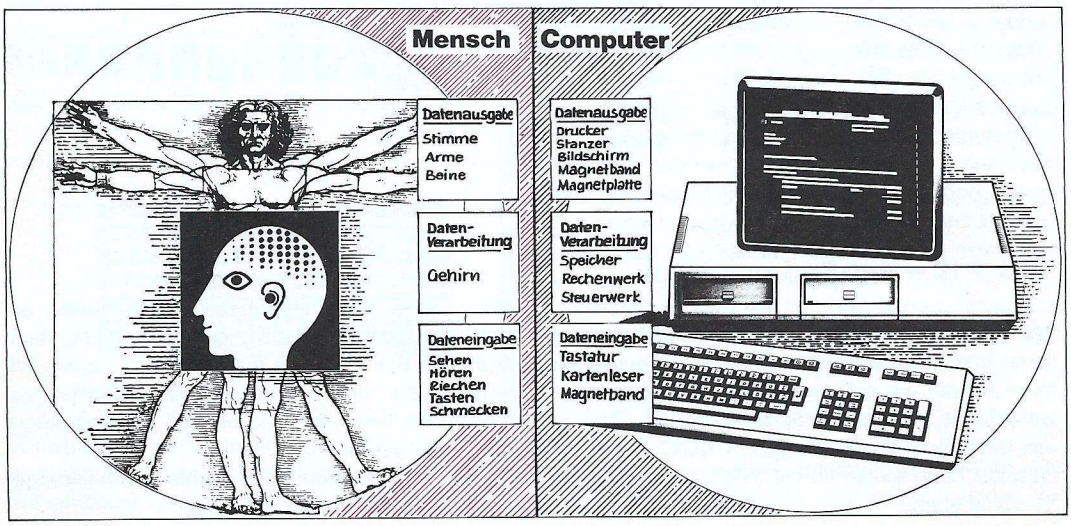
Mensch	Computer
arbeitet langsam	arbeitet schnell
kann Fehler machen	macht keine Fehler
ist vergeßlich	vergißt nie (solange er eingeschaltet ist)
hat weniger Ausdauer	hat Ausdauer
denkt selbst	kann nicht denken
ist kreativ	ist nicht kreativ

Sie haben alle wesentliche Merkmale einer EDV-Anlage an Ihrem KOSMOS-Computer bereits kennengelernt:

Er hat eine Eingabe-Einheit (Tastatur)
Er hat eine Zentraleinheit (Speicher, Steuerwerk, Rechenwerk)
Er hat eine Ausgabe-Einheit (Leuchtanzeige)

Auch die typischen Computer-Eigenschaften – Schnelligkeit, Fehlerfreiheit, Gedächtnis, Ausdauer und Unvermögen, selbst zu denken – konnten Sie bereits an ihm feststellen.
Das allerdings, was ihn gegenüber vielen Computern seiner Klasse auszeichnet, ist bislang noch nicht zur Sprache gekommen. Ganz zu Beginn dieses Buches wiesen wir Sie schon auf die vielen Anschlußklemmen an der Rückseite des Computers hin. Davon soll jetzt erneut die Rede sein. Und von den fast unbegrenzten Möglichkeiten, zwischen dem Computer und seiner Umwelt im wahrsten Sinne des Wortes eine gute Verbindung herzustellen.

Bild 45



Listing 6: **Würfelprogramm mit VGL**

Adresse	Mnemonics	**Code**	Kommentar
001	AKO 001	**04.001**	In den Akku eine „1" laden
002	ABS 101	**06.101**	Akku-Inhalt in Speicherzelle 101 speichern
003	ABS 000	**06.000**	Akku-Inhalt in Speicherzelle 000 speichern
004	ADD 101	**07.101**	Zum Akku-Inhalt „1" addieren
005	VGL 100	**10.100**	Prüfen, ob Akku-Inhalt gleich „7" ist
006	SPB 001	**11.001**	Wenn ja, bei 001 neu beginnen
007	SPU 003	**09.003**	Wenn nein, zum Speichern springen
100		**00.007**	Vergleichszahl
101		**00.001**	Schrittweite

Listing 7: **Würfelprogramm mit VGR**

Adresse	Mnemonics	**Code**	Kommentar
001	AKO 001	**04.001**	In den Akku eine „1" laden
002	ABS 101	**06.101**	Akku-Inhalt in Speicherzelle 101 speichern
003	ABS 000	**06.000**	Akku-Inhalt in Speicherzelle 000 speichern
004	ADD 101	**07.101**	Zum Akku-Inhalt eine „1" addieren
005	VGR 100	**12.100**	Prüfen, ob Akku-Inhalt größer als „6" ist
006	SPB 001	**11.001**	Wenn ja, bei 001 neu beginnen
007	SPU 003	**09.003**	Wenn nein, zum Speichern springen
100		**00.006**	Vergleichszahl
101		**00.001**	Schrittweite

Listing 8: **Würfelprogramm mit VKL**

Adresse	Mnemonics	**Code**	Kommentar
001	AKO 001	**04.001**	In den Akku eine „1" laden
002	ABS 101	**06.101**	Akku-Inhalt in Speicherzelle 101 speichern
003	ABS 000	**06.000**	Akku-Inhalt in Speicherzelle 000 speichern
004	ADD 101	**07.101**	Zum Akku-Inhalt eine „1" addieren
005	VKL 100	**13.100**	Prüfen, ob Akku-Inhalt kleiner als „7" ist
006	SPB 003	**11.003**	Wenn ja, zum Speichern springen
007	SPU 001	**09.001**	Wenn nein, bei 001 neu beginnen
100		**00.007**	Vergleichszahl
101		**00.001**	Schrittweite

1.53 P1E – vom Port zum Akku

Wir wollen wie gewohnt eine kleine Aufgabe formulieren und dann überlegen, wie der Computer sie lösen könnte. Es soll eine Stoppuhr programmiert werden, die auf Tastendruck losläuft und – das ist das Besondere an der Aufgabe – durch Druck auf dieselbe Taste wieder anhält. Wie eine lebensechte Stoppuhr eben.

Also, um es gleich zu sagen: mit den Tasten auf unserem Computer selbst geht das nicht, denn RUN und STP sind nun mal getrennte Tasten. Wir wollen dafür eine Eingabe-Einheit *außerhalb* des Computers benutzen, und die soll jetzt mal angeschlossen werden.

Folgende vorbereitende Handgriffe sind bei **ausgeschaltetem Computer** zu erledigen.

1. Die **Masseschiene** wird an der ganz rechten Klemme der Anschlußleiste angeschraubt (Bild 46)

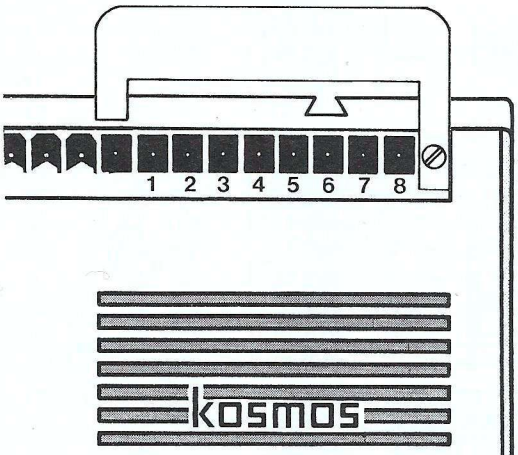


Bild 46

Die ganz rechte Klemme und zusätzlich die dritte Klemme von links sind mit dem Minuspol der internen Computerstromversorgung verbunden. Elektroniker pflegen den Minuspol auch als Masse- oder Null-Volt-Klemme zu bezeichnen, und so wollen wir es im folgenden auch halten: wenn wir von null Volt (0V) sprechen, so wird damit stets der Minuspol gemeint sein.

2. Ein **Kontaktbügel** wird nun an die vorletzte rechte Klemme mit der Aufschrift 8 geschraubt (Bild 47)

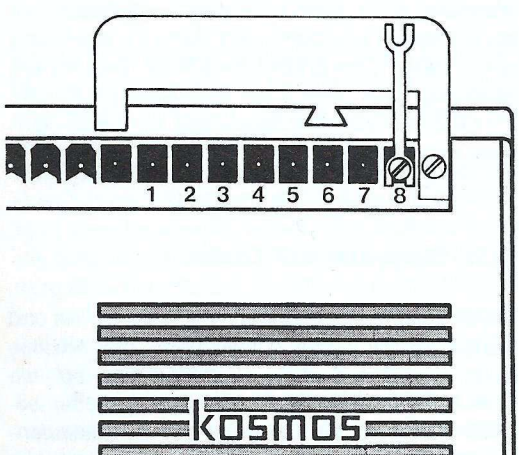


Bild 47

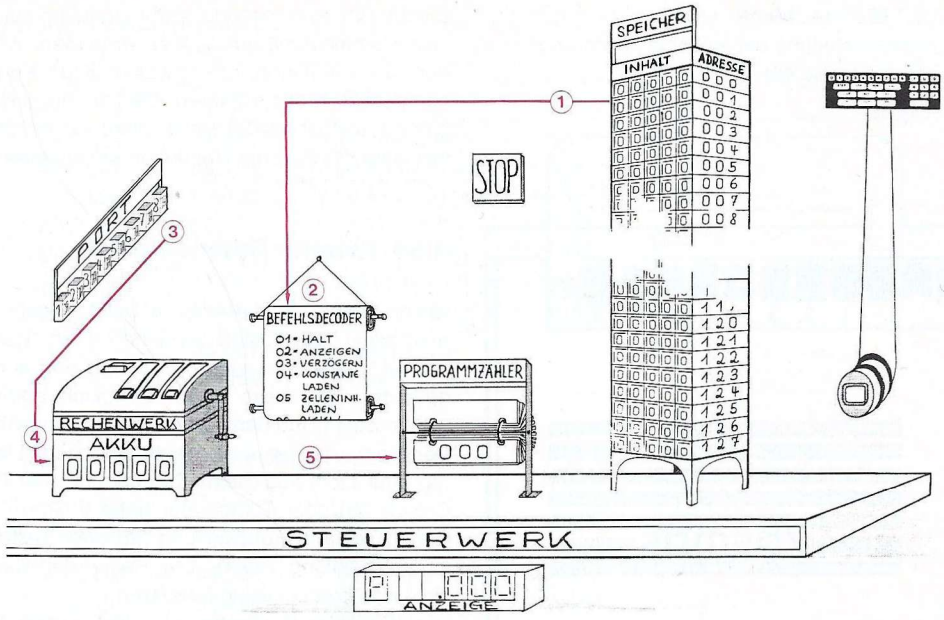
Masseschiene und Kontaktbügel bilden einen sehr komfortablen „Taster": ein leichtes Tippen auf den Kontaktbügel genügt, um zwischen den beiden Blechteilen eine leitende Verbindung herzustellen, also Klemme 8 mit null Volt zu verbinden. Wir werden diese einfache, aber zweckmäßige Konstruktion sehr bald ausprobieren. Sie können den Computer zunächst wieder einschalten und mit uns gemeinsam die weiteren Überlegungen anstellen.

1.54 Externe Dateneingabe

Unser Thema ist die externe Dateneingabe. Was man alles unter Daten verstehen kann, wurde in Kapitel 1.39 schon gesagt. Es wurde dort auch angedeutet, daß diese Daten elektrisch aufbereitet sein müssen. Und damit ist folgendes gemeint:
Der Computer braucht exakte Angaben: Licht an (ja) oder Licht aus (nein). Temperatur über 0 Grad Celsius (ja) oder Temperatur unter 0 Grad Celsius (nein), Taste in Ruhestellung (ja) oder Taste nicht in Ruhestellung (nein). Die Reihe der Beispiele ließe sich noch beliebig fortsetzen.
Der KOSMOS-Computer wird Ihre Daten bereitwillig verarbeiten, wenn Sie dafür sorgen, daß an seinen Klemmen die Aussage „ja" einer Spannung von **fünf Volt** (5 V) und die Aussage „nein" einer Spannung von **null Volt** (0 V) entspricht. Mit Zwischenwerten kann er nichts anfangen.
Alles, was Sie wissen müssen, ist nun folgendes: Der Computer ist (mit einem Befehl, den wir Ihnen gleich vorstellen werden) in der Lage, die Information, also die elektrischen Werte 0 V oder 5 V, von den Anschlußklemmen als „Ja-oder-Nein-Angabe" in den Akku zu übernehmen. Und wenn man schon ein gewisses Gespür für die Computer-Logik hat, erscheint es einsichtig, daß eine Ja-Information (also 5 V) im Akku als eine „1" (computermäßig 00.001) und eine Nein-Information (also 0 V) als eine „0" (computermäßig 00.000) erscheint.
Die Elektronik-Konstrukteure Ihres Computers haben sich natürlich die Frage gestellt, wie er reagieren soll, wenn an den Eingangsklemmen nichts angeschlossen ist. Und weil der Computer mit „nichts" nichts anfangen kann, haben sie durch eine elektrische Schaltung innen im Computer vorgesehen, daß bei „offenen Klemmen" in den Akku eine Ja-Information, also eine „1" eingegeben wird. Sie hätten's auch andersherum machen können, aber diese Lösung hat, wie Sie später feststellen werden, eine Reihe von Vorteilen.
Beginnen wir also mit der praktischen Arbeit und geben dem Computer auf Speicherzelle 001 den Befehl

zunächst ohne weitere Erklärung ein, stellen den Programmzähler auf 001 und drücken einmal die

Bild 48



Taste STEP. Damit ist der Befehl ausgeführt, und wir können uns das Ergebnis im Akku betrachten (Taste ACC). Sie werden feststellen, daß der Akku-Inhalt erwartungsgemäß 00.001 ist.

Machen Sie nun gewissermaßen die „Gegenprobe". Stellen Sie den Programmzähler nochmals auf 001, drücken Sie den Taster (so daß Klemme 8 mit 0 V verbunden ist) und betätigen Sie *gleichzeitig* die STEP-Taste. Ganz klar, die Information 0 V ist in den Akku als 00.000 übernommen worden.

Mit diesem Befehl, den wir einfach schon einmal ausprobiert haben, kann man jede der rechten 8 Leitungen „abfragen". Man braucht nur die letzte Ziffer des Adreßcodes zu verändern, also 16.001, 16.002 ... 16.008. Allgemein formuliert lautet der Befehl

16.00x

und mnemonisch

P1E 00x

P steht hier für den Computer-Fachausdruck **Port** (englisch port = Tür, Pforte). Eine Gruppe von 8 Computer-Eingangs- und Ausgangsleitungen werden als Port bezeichnet. Die rechte Achter-Gruppe heißt P1 und die linke Achter-Gruppe heißt P2. Daß in diesem Fall „E" Dateneingabe bedeutet, bedarf sicher keiner weiteren Erläuterung. Wir wollen die Wirkung des Befehls noch einmal kurz zusammenfassen.

„Gib die Ja- oder Nein-Information der Klemme 00x von Port 1 in den Akku"

Der Befehlszyklus läuft folgendermaßen ab

1. Befehl aus dem Speicher holen
2. Befehl decodieren
3. Den Zustand der Klemme 00x abfragen
4. Entsprechend der Ja/Nein-Information an der Klemme in den Akku eine „0" oder eine „1" schreiben
5. Programmzähler eins weiterschalten (Bild 48).

Wir wollen an dieser Stelle die eingangs gegebene **Warnung** noch einmal deutlich wiederholen: An keine Klemme des Computers darf eine Spannung gelegt werden, die **größer als 5 V ist.** Eine höhere Spannung oder aber auch eine Spannung unter 0 V (also eine negative Spannung) kann den Computer beschädigen.

1.55 Stoppuhr mit Taste

Geben Sie jetzt das Stoppuhren-Programm ein und starten Sie es durch 001-PC-RUN. Die Anzeige bleibt zunächst dunkel. Wenn Sie jetzt auf die „Taste" an Klemme 8 tippen und sofort wieder loslassen, beginnt der Computer mit 1/10-Sekunden-Geschwindigkeit zu zählen. Stoppen Sie ihn durch erneutes Drücken auf die Taste. Die gestoppte Zeit erscheint auf der Anzeige direkt in Zehntelsekunden. Aber Vorsicht, auch wenn Sie es nicht sehen können: der Computer läuft weiter (und zwar in einer Schleife). Wenn Sie probehalber versuchen, et-

Listing 9: **Stoppuhr mit externer Start/Stop-Taste**

Adresse	Mnemonics	Code	Kommentar
001	AKO 000	**04.000**	Zahlenwerte im Datenbereich speichern
002	ABS 100	**06.100**	
003	ABS 102	**06.102**	
004	AKO 001	**04.001**	
005	ABS 101	**06.101**	
006	P1E 008	**16.008**	Bringe die Information von Port 1/8 in den Akku
007	VGL 100	**10.100**	Prüfe, ob Akku-Inhalt „0" ist
008	SPB 010	**11.010**	Wenn ja, springe nach 010
009	SPU 006	**09.006**	Wenn nein, springe zurück nach 006
010	P1E 008	**16.008**	Bringe die Information von Port 1/8 in den Akku
011	VGL 101	**10.101**	Prüfe, ob Akku-Inhalt „1" ist
012	SPB 014	**11.014**	Wenn ja, springe zum Laden nach 014
013	SPU 010	**09.010**	Wenn nein, springe zurück nach 010
014	LDA 102	**05.102**	Lade den Inhalt der Zählzelle in den Akku
015	ADD 101	**07.101**	Addiere „1" dazu
016	ABS 102	**06.102**	Speichere den neuen Akku-Inhalt in Zelle 102
017	ANZ	**02.000**	Zeige ihn an
018	VZG 087	**03.087**	Verzögere 87 ms
019	P1E 008	**16.008**	Bringe die Information von Port 1/8 in den Akku
020	VGL 100	**10.100**	Prüfe, ob Akku-Inhalt „0" ist
021	LDA 102	**05.102**	Lade den Inhalt der Zählzelle in den Akku
022	SPB 024	**11.024**	Wenn Akku-Inhalt „0" war, springe zum Anzeigen
023	SPU 015	**09.015**	Wenn nicht, springe zum Addieren
024	ANZ	**02.000**	Zeige den Akku-Inhalt an
025	VZG 250	**03.250**	Verzögere 250 ms
026	VZG 250	**03.250**	Verzögere nochmal 250 ms
027	SPU 001	**09.001**	Springe zum Anfang zurück
100		**00.000**	Vergleichszahl
101		**00.001**	Vergleichszahl und Schrittweite
102		**00.xxx**	Zählzelle

was einzugeben, wird er nicht reagieren. Erst wenn Sie die STP-Taste drücken, würde er anhalten. Abermaliges Drücken auf die Taste startet die Stoppuhr wieder bei Null.

Wenn sich beim Eintippen ein Fehler eingeschlichen haben sollte, kann das Programm natürlich nicht ordnungsgemäß laufen. Überprüfen Sie, ob alle Befehle richtig im Speicher stehen. Wählen Sie dazu Speicherzelle 001 an (001-OUT) und drücken Sie fortlaufend OUT. Vergleichen Sie die angezeigten Werte (Kennbuchstabe „C" für Speicherzellen-Inhalt) mit dem Listing.

Das Programm ist mit Hilfe der Kommentare leicht zu verstehen. Ein Hinweis vielleicht: auf Adresse 010 steht ein zweiter Eingabebefehl, dem ein Vergleich auf „1" (Taste nicht mehr gedrückt?) folgt. Damit wird sichergestellt, daß man das Programm nur stoppen kann, wenn die Taste nach dem Start erst einmal losgelassen wurde (Dauerdrücken ignoriert der Computer).

1.56 P1A – vom Akku zum Port

Glücklicherweise ist die Verbindung zwischen Port 1 und dem Akku keine Einbahnstraße. Das heißt: man kann sowohl Daten von Port 1 zum Akku befördern als auch umgekehrt vom Akku zum Port. In der Praxis sieht das so aus: Sie laden z.B. in den Akku eine „1" aus einer Speicherzelle und befehlen dem Computer, diese Ja-Information auf eine Klemme von Port 1 auszugeben. An der im Ausgabe-Befehl angegebenen Klemme wird eine Spannung von 5 V erscheinen. Wir werden den Befehl kurz erläutern und Ihnen dann anhand eines kleinen Beispiel-Programmes sagen, was Sie praktisch damit anfangen können.

P1A 00x

bedeutet

„Gib die Ja-oder-Nein-Information aus dem Akku auf die Klemme 00x von Port 1"

Für x kann man je nachdem, welche Klemme man anwählen will, die Zahlen 1 bis 8 einsetzen. Der Computer-Code lautet

17.00x

Der Befehlszyklus läuft wie beim Eingabe-Befehl ab, nur daß Unterschritte 3 und 4 sinngemäß gegeneinander vertauscht sind, also eine „0" oder eine „1" vom Akku an die Port-Klemme gegeben wird.

1.57 Blinkschaltung per Computer

Wenn Sie Elektronik-Bastler sind oder irgendeinen KOSMOS-Elektronik-Experimentierkasten besitzen, so sind Ihnen Blinkschaltungen aller Art geläufig. Sie wissen dann auch, daß bei einer solchen Schaltung Widerstände und Kondensatoren für die Blinkgeschwindigkeit (Blinkfrequenz) verantwortlich sind. Bei einem computergesteuerten Blinklicht können Sie die Blinkfrequenz natürlich elegant per Befehl (fast) beliebig verändern.

In Bild 49 haben wir ein Computer-Blinklicht mit den Teilen aus dem KOSMOS-Kasten „Spiele mit Elektronik" dargestellt. (Es läßt sich natürlich auch das Material anderer KOSMOS-Kästen verwenden, und man kann ohne weiteres das Lämpchen durch eine Reihenschaltung aus einer Leuchtdiode und einem 120 Ω-Widerstand ersetzen. Übrigens: sämtliche Elektronikteile können Sie beim örtlichen Fachhandel beziehen oder bei uns bestellen. Ein Bestellschein liegt bei.) Da man den Klemmen des Computers nur einen sehr geringen Strom entnehmen kann, müssen für den Betrieb von Lämpchen oder Leuchtdioden (LEDs) Verstärker-Transistoren verwendet werden. Eine Ja-Information (also 5 V) an der gewählten Klemme bewirkt dann, daß das Lämpchen brennt, eine Nein-Information (0 V) hat zur Folge, daß es dunkel ist.
Das nachfolgende Programm ist sehr einfach zu verstehen. Es besteht aus den Befehls-Folgen

In den Akku eine „0" laden und auf Klemme 1 ausgeben (Lämpchen ausschalten)

Verzögern (Festlegen der Ausschaltdauer)

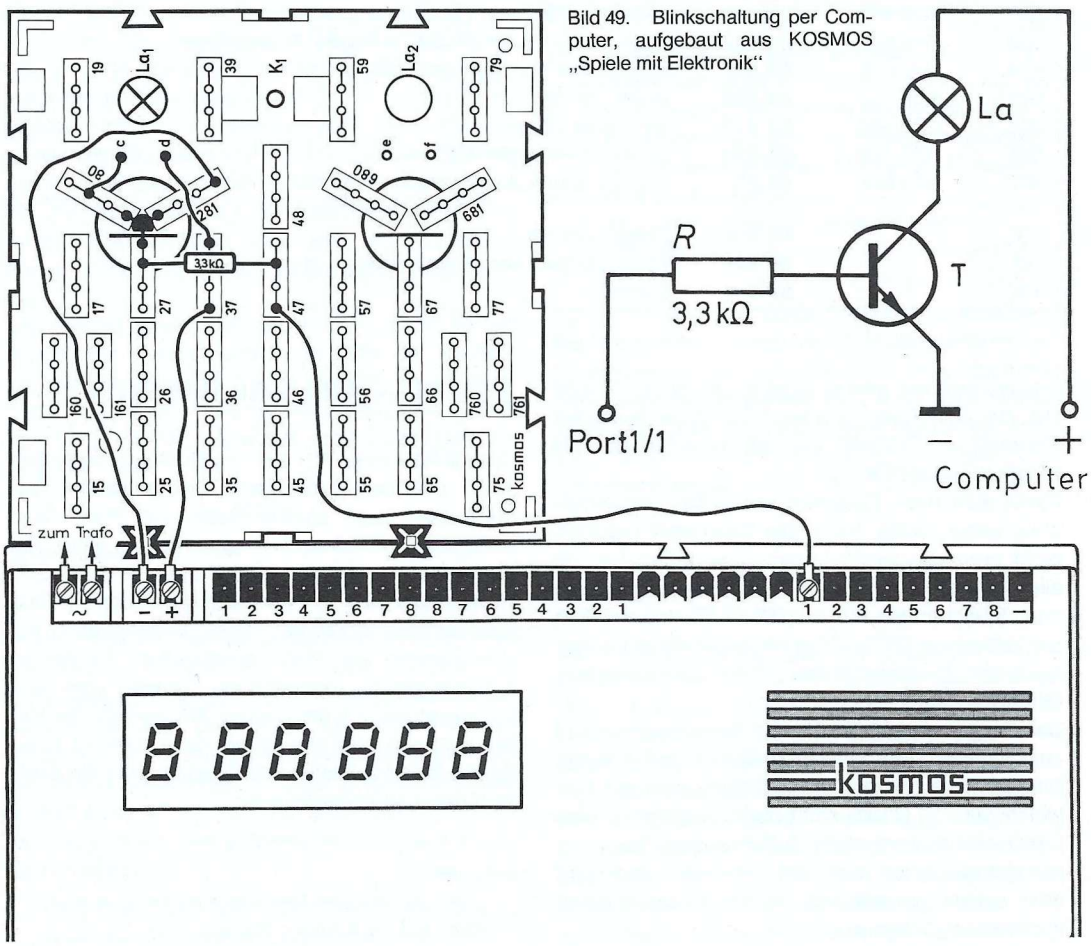


Bild 49. Blinkschaltung per Computer, aufgebaut aus KOSMOS „Spiele mit Elektronik"

Listing 10: **Einfacher Blinker**

Adresse	Mnemonics	**Code**	Kommentar
001	AKO 000	**04.000**	In den Akku eine „0" laden
002	P1A 001	**17.001**	Akku-Inhalt auf Klemme 1 des Port 1 bringen
003	VZG 100	**03.100**	100 ms verzögern
004	AKO 001	**04.001**	In den Akku „1" laden
005	P1A 001	**17.001**	Akku-Inhalt auf Klemme 1 des Port 1 bringen
006	VZG 100	**03.100**	100 ms verzögern
007	SPU 001	**09.001**	Zurück nach 001 springen

In den Akku eine „1" laden und auf Klemme 1 ausgeben (Lämpchen einschalten)

Verzögerung (Festlegen der Einschaltdauer)

Rücksprung

Stellen Sie bitte alle Verbindungen zum Elektronikteil her, **bevor** Sie den Computer einschalten. Wenn Sie nun den Netzstecker einstecken, werden Sie feststellen, daß das Lämpchen brennt. Das ist in Ordnung, denn der Computer gibt beim Einschalten automatisch und einheitlich auf alle Port-Klemmen eine Ja-Information, also 5 V aus. Sie können jetzt auch verstehen, warum wir in unserem Blinkprogramm das Lämpchen zuerst ausschalten – eben weil es nach dem Einschalten schon brannte.

Aufgaben:
1. Ändern Sie die Blinkfrequenz
2. Ändern Sie das Programm so, daß unterschiedliche Ein- und Ausschaltzeiten entstehen
3. Schalten Sie ein zweites Lämpchen dazu und lassen Sie die Lämpchen abwechselnd blinken
4. Ergänzen Sie das Programm so, daß ein „Tastendruck" den Blinkvorgang startet und ein weiterer ihn stoppt (Lösungen auf Seite 42)

1.58 P2A – ein zweiter Port zum Ausgeben

Ein kurzes Wort zu den Klemmen 1 bis 8 auf der *linken* Seite, zu Port 2. Durch einen weiteren Ausgabe-Befehl kann der Akku per Befehl mit diesen Klemmen „verbunden" werden. Allerdings funktioniert die Sache nur in einer Richtung: Port 2 dient ausschließlich der Ausgabe von Informationen aus dem Akku.

Port 1 ist ein sogenannter **bidirektionaler** Port (von bi = 2); es können Daten über dieses Port sowohl ein- als auch ausgegeben werden.
Port 2 ist ein **reiner Ausgabeport.**

Der Ausgabe-Befehl für Port 2 lautet

P2A 00x bzw. 18.00x

Der Befehlszyklus ist identisch mit dem für den Ausgabe-Befehl für Port 1.

Aufgabe (ohne Lösung):
Entwickle ein Programm für eine Blinkschaltung mit Port 2 als Ausgabe für die Lämpchen und mit Port 1 als Eingabe für 2 Taster. Nur wenn beide Taster gleichzeitig gedrückt sind, soll der Start- bzw. Stop-Vorgang ausgelöst werden.

1.59 Logeleien à la Boole

Die Ja/Nein-Aussagen und die zugehörigen 1- und 0-Werte, die Sie im letzten Kapitel kennengelernt haben, nennt man **logische Werte.** Nicht etwa, daß alles andere unlogisch wäre. Die Bezeichnung weist vielmehr auf eine besondere Art der Zahlenzuordnung hin, auf die nach dem irischen Mathematiker Lincoln Boole benannte **Boolesche Algebra.**
In der Computertechnik leistet diese Art Algebra hervorragende Dienste, so daß wir Ihnen ein paar kleine Kostproben davon servieren wollen.
Herr Boole könnte z. B. behaupten:

„Wenn es regnet, **und** wenn außerdem die Sonne scheint, gibt es einen Regenbogen."

Das scheint uns logisch. Fehlt eine der beiden Bedingungen (Regen oder Sonnenschein), so wird man vergeblich auf den Regenbogen warten.
Eine solche Verknüpfung von zwei Aussagen nennt man eine **Konjunktion** oder **UND-Verknüpfung.** (Sie werden auch oft die englische Bezeichnung AND finden.) Man kann die Möglichkeiten

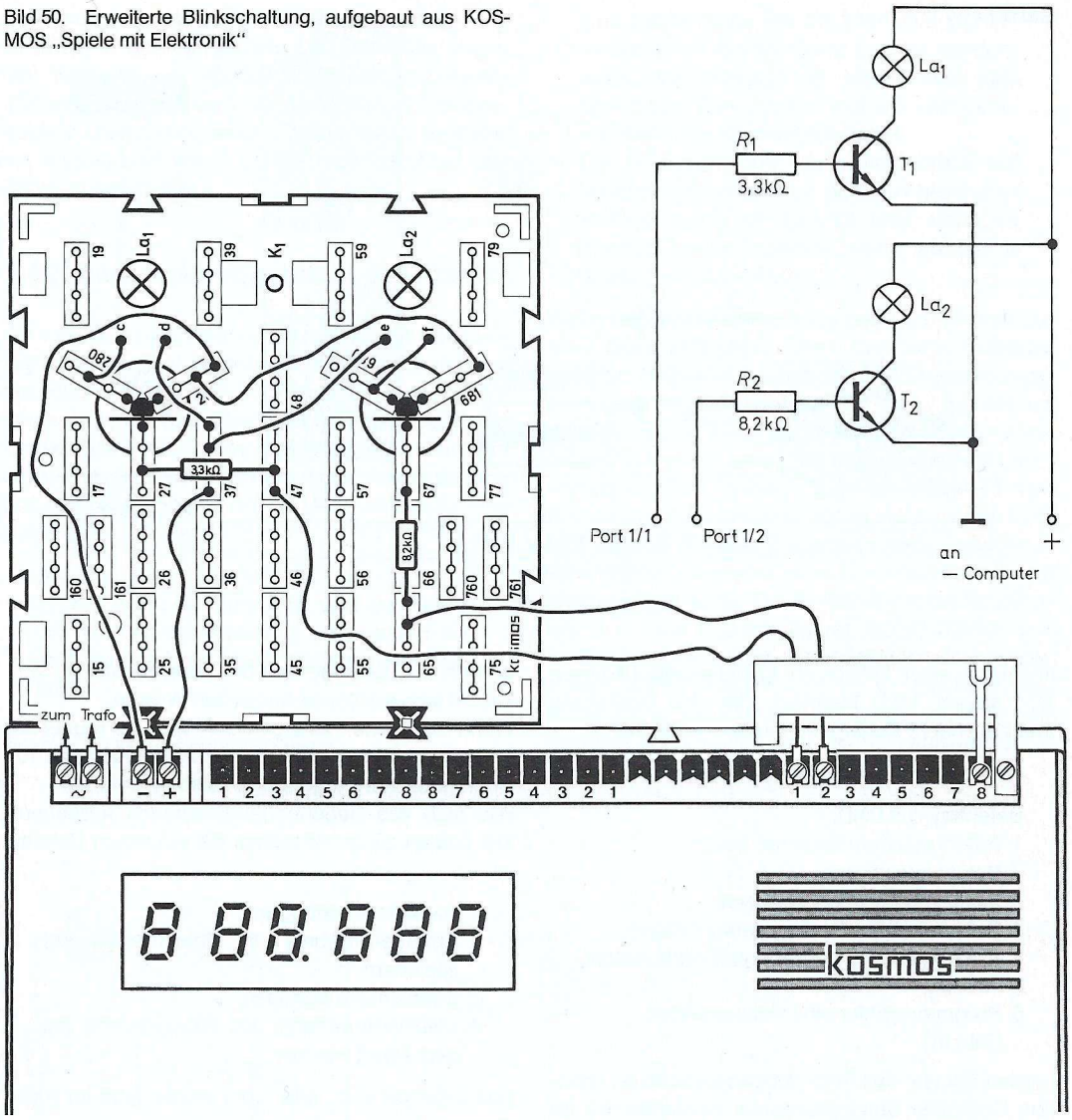
Listing 11: **Wechselblinker**

Adresse	Mnemonics	**Code**	Kommentar
001	AKO 000	**04.000**	In den Akku eine „0" laden
002	P1A 001	**17.001**	Akku-Inhalt auf Klemme 1 des Port 1 bringen
003	AKO 001	**04.001**	In den Akku eine „1" laden
004	P1A 002	**17.002**	Akku-Inhalt auf Klemme 2 des Port 1 bringen
005	VZG 250	**03.250**	250 ms verzögern
006	AKO 001	**04.001**	In den Akku eine „1" laden
007	P1A 001	**17.001**	Akku-Inhalt auf Klemme 1 des Port 1 bringen
008	AKO 000	**04.000**	In den Akku eine „0" laden
009	P1A 002	**17.002**	Akku-Inhalt auf Klemme 2 des Port 1 bringen
010	VZG 250	**03.250**	250 ms verzögern
011	SPU 001	**09.001**	Zurück nach 001 springen

Listing 12: **Wechselblinker mit externer Start/Stop-Taste**

Adresse	Mnemonics	**Code**	Kommentar
001	AKO 000	**04.000**	In den Akku eine „0" laden
002	P1A 001	**17.001**	Akku-Inhalt auf Klemme 1 des Port 1 ausgeben
003	P1A 002	**17.002**	Akku-Inhalt auf Klemme 2 des Port 1 ausgeben
004	P1E 008	**16.008**	Information von Klemme 8 des Port 1 in den Akku bringen
005	VGL 100	**10.100**	Akku-Inhalt mit Inhalt v. Zelle 100 vergleichen
006	SPB 008	**11.008**	Wenn Gleichheit, dann nach 008 springen
007	SPU 004	**09.004**	Wenn nicht Gleichheit, nach 004 springen
008	P1E 008	**16.008**	Information v. Klemme 8 des Port 1 in den Akku bringen
009	VGL 101	**10.101**	Akku-Inhalt mit Inhalt von Zelle 101 vergleichen
010	SPB 012	**11.012**	Wenn Gleichheit, dann auf 012 springen
011	SPU 008	**09.008**	Wenn nicht Gleichheit, auf 008 springen
012	AKO 001	**04.001**	In den Akku eine „1" laden
013	P1A 001	**17.001**	Akku-Inhalt auf Klemme 1 des Port 1 ausgeben
014	AKO 000	**04.000**	In den Akku eine „0" laden
015	P1A 002	**17.002**	Akku-Inhalt auf Klemme 2 des Port 1 ausgeben
016	VZG 250	**03.250**	250 ms verzögern
017	AKO 000	**04.000**	In den Akku eine „0" laden
018	P1A 001	**17.001**	Akku-Inhalt auf Klemme 1 des Port 1 ausgeben
019	AKO 001	**04.001**	In den Akku eine „1" laden
020	P1A 002	**17.002**	Akku-Inhalt auf Klemme 2 des Port 1 ausgeben
021	VZG 250	**03.250**	250 ms verzögern
022	P1E 008	**16.008**	Information v. Klemme 8 des Port 1 in den Akku bringen
023	VGL 100	**10.100**	Akku-Inhalt mit Inhalt von Zelle 100 vergleichen
024	SPB 026	**11.026**	Wenn Gleichheit, dann auf 026 springen
025	SPU 012	**09.012**	Wenn nicht Gleichheit, auf 012 springen
026	VZG 250	**03.250**	Verzögere 250 ms
027	VZG 250	**03.250**	Verzögere nochmals 250 ms
028	SPU 004	**09.004**	Springe zum Anfang (nach 004) zurück
100		**00.000**	1. Vergleichszahl
101		**00.001**	2. Vergleichszahl

Bild 50. Erweiterte Blinkschaltung, aufgebaut aus KOSMOS „Spiele mit Elektronik“



einer solchen Verknüpfung sehr übersichtlich in einer Tabelle zusammenfassen, etwa so:

Regen	Sonnenschein	Regenbogen
nein	nein	nein
nein	ja	nein
ja	nein	nein
ja	ja	ja

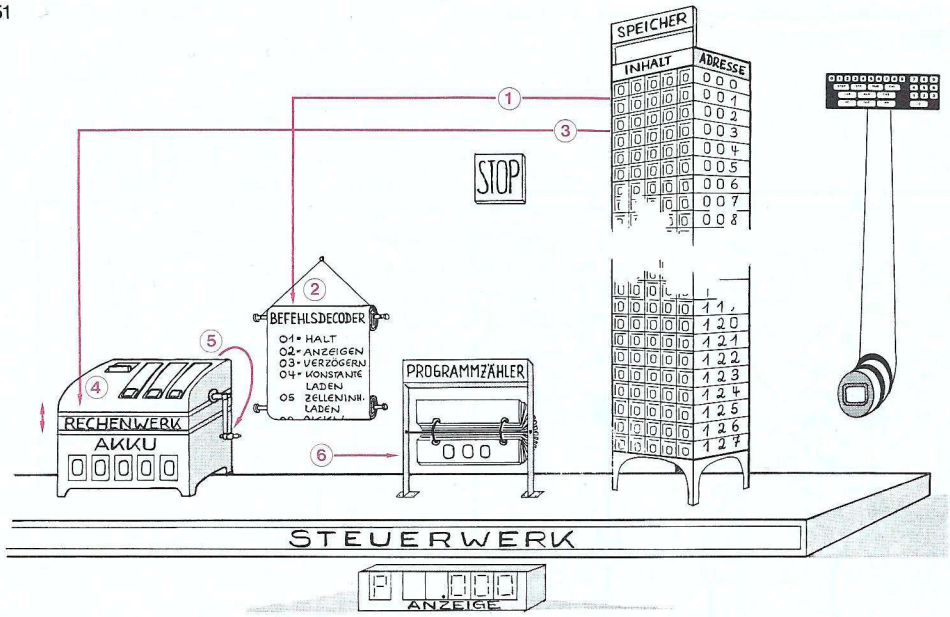
Und wenn man, was wir bereits in Kapitel 1.54 getan haben, statt ja eine „1“ und statt nein eine „0“ setzt, sieht die Tabelle folgendermaßen aus:

Regen	Sonnenschein	Regenbogen
0	0	0
0	1	0
1	0	0
1	1	1

Eine solche Tabelle heißt **Wahrheitstabelle,** wohl deshalb, weil man das „ja“ als wahr, das „nein“ dagegen als falsch einordnen würde.

Unser Computer kann zwei Boolesche Werte (also „0“ oder „1“, ja oder nein) logisch mit dem sog. Konjunktions- oder UND-Befehl miteinander verknüpfen. Der eine Wert muß dazu im Akku, der

Bild 51



andere in einer beliebigen Speicherzelle (Adresse xxx) stehen. UND bedeutet: die eine Bedingung **und** die andere Bedingung müssen erfüllt sein.

UND xxx bzw. 15. xxx

Befehlszyklus UND:

1. Befehl aus dem Speicher holen
2. Befehl decodieren
3. Inhalt von Zelle xxx kopieren
4. Zelleninhalt ins Rechenwerk eingeben
5. UND-Verknüpfung ausführen, Ergebnis steht im Akku
6. Programmzähler eins weiterschalten (Bild 51).

Lassen Sie uns das Regenbogen-Beispiel an unserem Computer durchexerzieren. Schließen wir an Klemme 1 von Port 1 die „Regentaste" und an Klemme 2 die „Sonnentaste" an. Für die Tasten soll folgendes gelten:
Wird die Regentaste gedrückt, so hört der Regen endlich auf. Betätigt man die Sonnentaste, so verkriecht sich die Sonne hinter den Wolken.
Wenn also keine Taste gedrückt wird, so regnet es, und die Sonne scheint. Alle Voraussetzungen für einen hübschen Regenbogen sind dann erfüllt.
Wie muß das Regenbogen-Programm aussehen? Wir deuten zunächst einmal die einzelnen Schritte an:

1. Regentaste abfragen
2. Abfrage-Ergebnis in Speicherzelle xxx speichern
3. Sonnentaste abfragen
4. UND-Verknüpfung des Akku-Inhaltes mit dem Inhalt von xxx

Das Ergebnis („1" oder „0") würde jetzt im Akku stehen.
Geben wir das kurze Programm ein.

Listing 13: **„Regenbogen"-Programm**

Adresse	Mnemonics	**Code**	Kommentar
001	P1E 001	**16.001**	Information von Klemme 1 des Port 1 in den Akku bringen
002	ABS 100	**06.100**	Akku-Inhalt in Zelle 100 speichern
003	P1E 002	**16.002**	Information von Klemme 2 des Port 1 in den Akku bringen
004	UND 100	**15.100**	Akku-Inhalt und Inhalt v. Speicherz. 100 UND-verknüpfen
005	ANZ	**02.000**	Akku-Inhalt anzeigen
006	SPU 001	**09.001**	Zurück nach 001 springen
100		**00.00x**	Speicherzelle zum Zwischenspeichern

Der Start erfolgt mit 001-PC-RUN. Nun betätigen wir die Tasten. Sie können alle Möglichkeiten, die in der „Regenbogen-Tabelle" aufgeführt sind, durchprobieren. Im Akku wird nur dann eine „1" (Regenbogen) erscheinen, wenn Taste 1 nicht gedrückt (es regnet) **und** wenn Taste 2 nicht gedrückt (die Sonne scheint) sind.

1.60 Boolesche Algebra in der Praxis

Sie werden einwenden wollen, daß unser meteorologisches Beispiel bestenfalls akademischen Wert hat, denn Regen und Sonnenschein per Tastendruck gehören zum Glück noch in das Reich der Utopie. Damit haben Sie natürlich völlig recht, und deshalb wollen wir noch ein paar Anwendungen praktisch brauchbarer Art für UND-Verknüpfungen aufzählen.

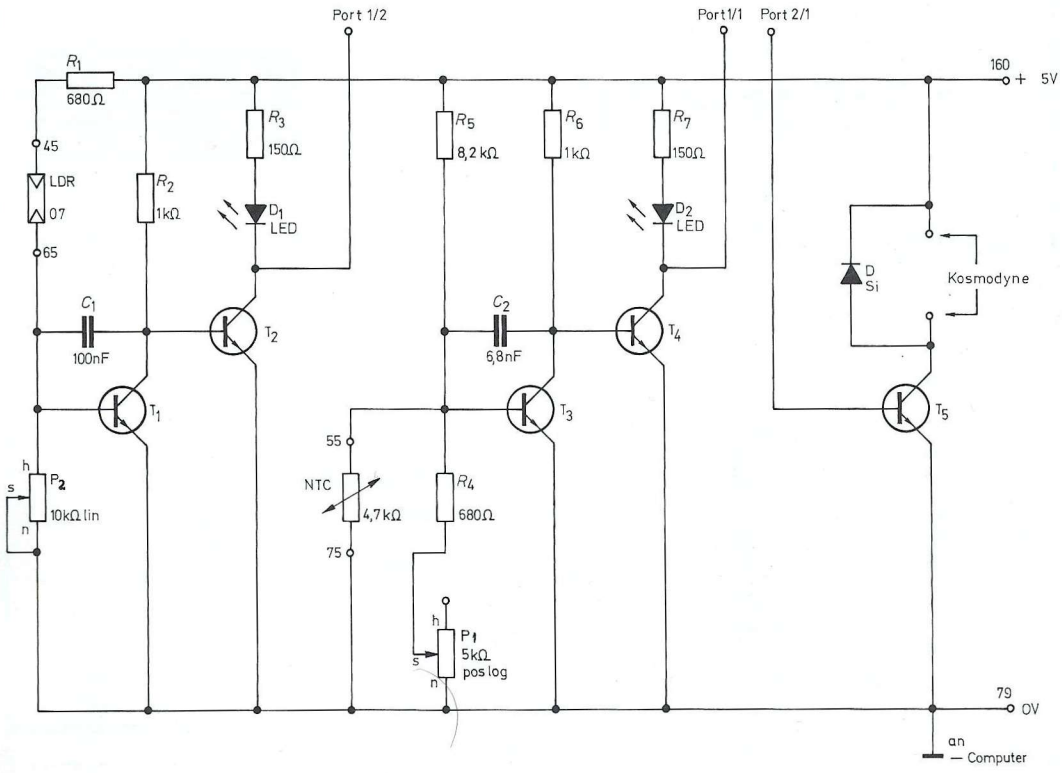
Die Tür einer Straßenbahn läßt sich nur öffnen, wenn die Bahn hält **und** der Fahrer die Türöffnungssicherung freigegeben hat **und** ein Fahrgast einen Türöffnerknopf betätigt.

Eine hydraulische Presse kann aus Sicherheitsgründen nur in Gang gesetzt werden, wenn der Arbeiter mit einer Hand den Sicherheitshebel drückt **und** mit der anderen Hand den Startknopf betätigt.

Die Heizung eines Einfamilienhauses soll computergesteuert nur dann eingeschaltet werden, wenn es Tag ist **und** wenn es draußen kalt ist (natürlich, sonst wäre es ja reinste Verschwendung).

Wenn Sie ein Einfamilienhaus besitzen, können Sie ohne Schwierigkeiten Ihren Computer nutzbringend für eine energiesparende Heizungssteuerung einsetzen. Was Sie elektronisch dazu vorbereiten müssen, zeigen Ihnen die Bilder 52 u. 53. Wenn die Temperatur *unter* einen am Potentiometer P1 vorwählbaren Wert absinkt, liegt an Klemme 1 von Port 1 eine Ja-Information. Wenn Licht auf den LDR fällt, liegt an Klemme 2 ebenfalls eine Ja-Information. Die Heizung wird über Klemme 1 von Port 2 eingeschaltet. In der Praxis muß dafür ein Schaltrelais (z.B. das Netzschaltgerät KOSMODYNE von KOSMOS) über einen Verstärkertransistor angesteuert werden. Das Programm für die Heizungssteuerung ist denkbar einfach. (Programme für raffiniertere Steuerungen sind natürlich aufwendiger.)

Bild 52 Schaltbild Heizungsregelung



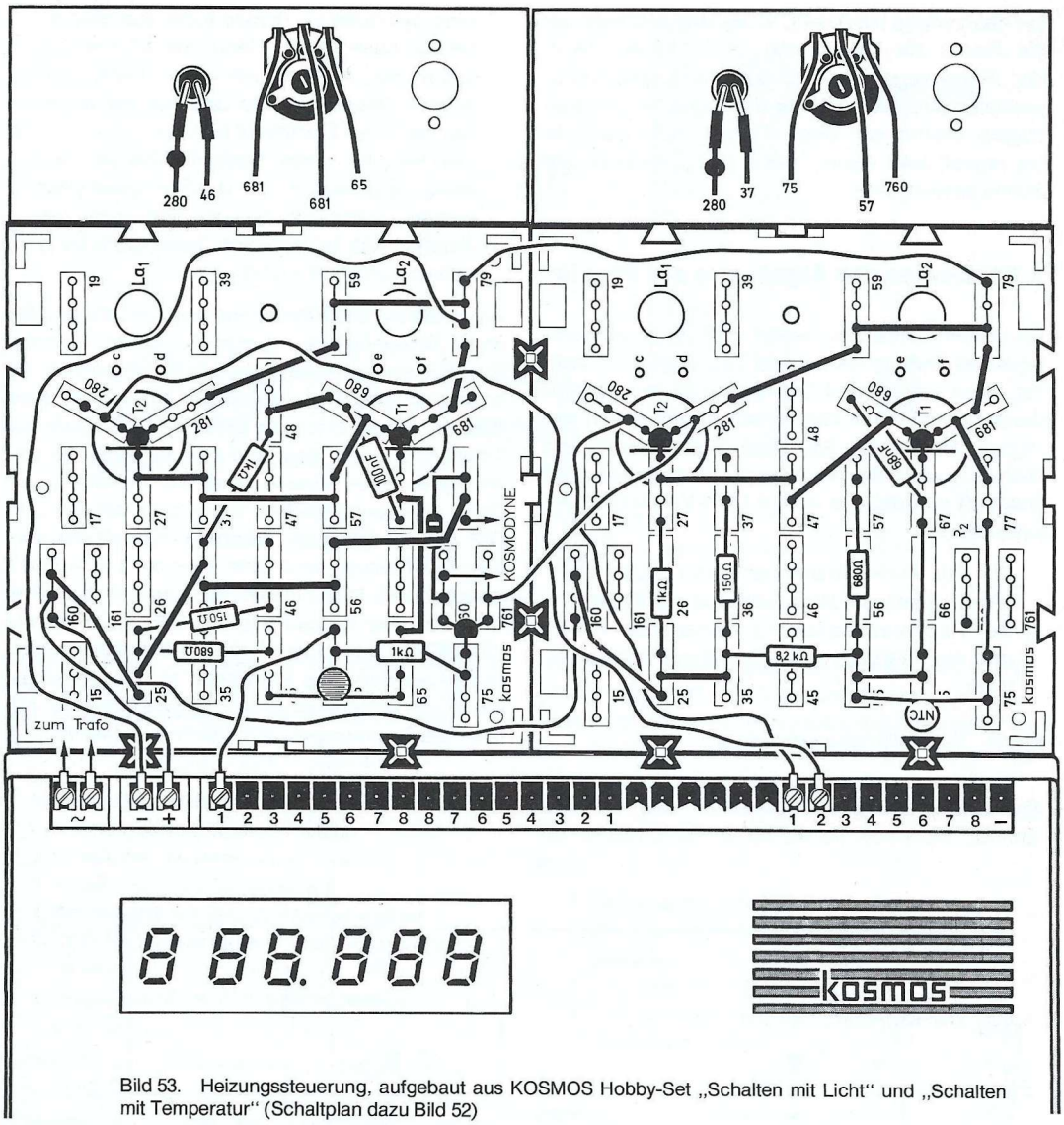


Bild 53. Heizungssteuerung, aufgebaut aus KOSMOS Hobby-Set „Schalten mit Licht" und „Schalten mit Temperatur" (Schaltplan dazu Bild 52)

Listing 14: **Heizungssteuerung**

Adresse	Menmonics	**Code**	Kommentar
001	P1E 001	**16.001**	Information von Port 1/1 in den Akku bringen
002	ABS 100	**06.100**	Akku-Inhalt in 100 speichern
003	P1E 002	**16.002**	Information von Port 1/2 in den Akku bringen
004	UND 100	**15.100**	UND-Verknüpfung mit Zelle 100
005	ANZ	**02.000**	Anzeigen
006	P2A 001	**18.001**	Akku-Inhalt an Port 2/1 ausgeben
007	VZG 250	**03.250**	1 Sekunde verzögern
008	VZG 250	**03.250**	
009	VZG 250	**03.250**	
010	VZG 250	**03.250**	
011	SPU 001	**09.001**	Sprung Zurück an den Anfang
100		**00.00x**	Zwischenspeicher

1.61 NEG – logische Umkehr

Wenn man sich mit dem Entwurf von Programmen für selbst ausgedachte Steuerungsaufgaben beschäftigt, stößt man garantiert auf das Problem, daß man als Ergebnis einer UND-Verknüpfung eine Ja-Information (also im Akku eine „1") erhält, während man für die Weiterverarbeitung oder für die Ansteuerung eines Gerätes eine Nein-Information benötigt. Klassisches Beispiel dafür ist die Alarmanlage beim Safe einer Bank die gerade dann *nicht* ausgelöst werden soll (nein), wenn es Tag ist (ja) und man versucht, den Safe mit dem richtigen Schlüssel (ja) zu öffnen. Schauen Sie sich die „Safe-Tabelle" an. Man kann sofort ablesen, wie die Alarmanlage zu arbeiten hat:

Ist es Tag?	Richtiger Schlüssel?	Alarm auslösen?
ja	ja	nein
ja	nein	ja
nein	ja	ja
nein	nein	ja

Der Computer-Befehl, der eine im Akku stehende Ja/Nein-Information in ihr jeweiliges Gegenteil verkehrt, heißt Negations-Befehl. Ein UND-Befehl, gefolgt von einem Negationsbefehl, ergibt eine UND-NICHT-Verknüpfung (englisch NOT-AND = NAND). Der Negationsbefehl lautet:

„Verkehre einen logischen Akku-Inhalt in sein Gegenteil (aus „0" wird „1" und aus „1" wird „0")".

und wird notiert als

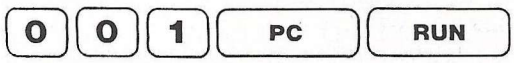
14.000 bzw. NEG

Die drei Ziffern rechts vom Punkt haben wie beim ANZ- und HLT-Befehl keinerlei Bedeutung, aber Sie wissen schon: wir empfehlen Ihnen, sich den Befehl der Einfachheit halber so zu merken, wie wir ihn oben notiert haben.
Der Befehlszyklus läuft so ab:

1. Befehl aus dem Speicher holen
2. Befehl decodieren
3. Befehl ausführen
4. Programmzähler um eins weiterschalten (Bild 54)

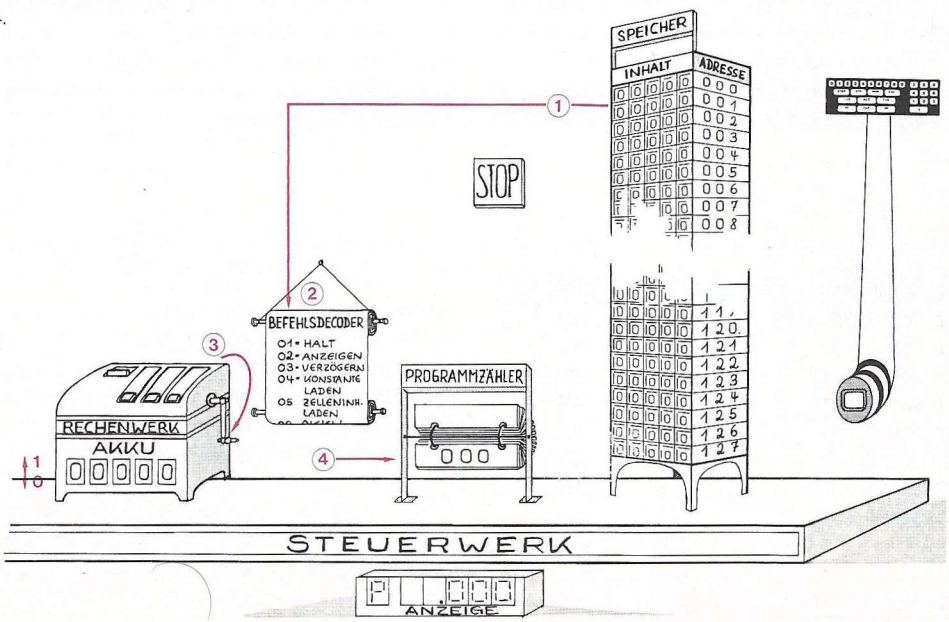
Anhand eines ganz kleinen Programmes können wir die Wirkungsweise des NEG-Befehls verfolgen. Wir laden in den Akku eine „1", zeigen sie an, verzögern, negieren den Akku-Inhalt und springen zurück zum Anzeigen.

Das Programm wird wie gewohnt durch



gestartet.
Zum Abschluß unserer Ausführungen über „Computerlogik" geben wir Ihnen noch einen wichtigen Hinweis.
Eigentlich versteht es sich von selbst, aber es könnte im Eifer des Gefechtes doch einmal übersehen werden: Die Befehle

Bild 54.



Listing 15: **Prüfprogramm für Negation**

Adresse	Mnemonics	**Code**	Kommentar
001	AKO 001	**04.001**	„1" in den Akku laden
002	ANZ	**02.000**	Akku-Inhalt anzeigen
003	VZG 250	**03.250**	250 ms verzögern
004	NEG	**14.000**	Akku-Inhalt negieren
005	SPU 002	**09.002**	Zurück zum Anzeigen springen

P1A 00x, P2A 00x, UND xxx und NEG

sind nur auf die logischen Werte „0" und „1" anwendbar. Bieten Sie dem Computer zur Ausführung dieser Befehle andere Werte an, so wird er Sie durch die Fehleranzeige F. 005 auf Ihren Irrtum aufmerksam machen.

1.62 Zahlenbetrachtungen

Hand aufs Herz: ist es Ihnen bis hierhin irgendwie bemerkenswert vorgekommen, daß Ihr Computer mit dezimalen Zahlen, also mit den Ziffern 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, arbeitet?
Natürlich nicht, jeder gewöhnliche Taschenrechner tut das ja. Das ist doch die selbstverständlichste Sache der Welt!
Sicher, aber was wir als normal empfinden, ist in den allermeisten Fällen lediglich eine Frage der Gewohnheit. Die „magische" Zahl 10, die unser Rechnen bestimmt, ist uns deshalb lieb und teuer geworden, weil wir eben gerade (zufällig) 10 Finger haben. Wenn wir über 9 hinauszählen, landen wir wieder bei 0 und müssen also zur Kennzeichnung, daß wir über 9 hinaus sind, eine zweite Stelle, eine 1 (die Zehnerstelle) hinzufügen. Das machen wir tagtäglich wie im Schlaf, ohne auch nur im geringsten darüber nachzudenken.
Denken Sie beispielsweise an den mechanischen Kilometerzähler im Auto. Wenn seine letzte Stelle wieder bei 0 angekommen ist, springt die vorletzte Stelle um eins weiter, und die letzte Stelle zählt wieder von 1 bis 0. Sie kann eben nur 10 Werte, nämlich die Ziffern von 0 bis 9 anzeigen und heißt daher Dezimalstelle (dezi = 10).

1.63 Das Dual-System

Grundsätzlich anders ist es bei einem elektronischen Gerät, also auch bei Ihrem Computer. Seine Transistoren können leitend sein oder nicht. Durch seine Bauteile kann Strom fließen oder nicht. An seinen Schaltkreisen kann Spannung liegen (5 V) oder nicht (0 V). Zwei Möglichkeiten kann der Computer nur unterscheiden, so als ob er zwei Finger hätte. 0 V oder 5 V, ja oder nein; oder in Ziffern ausgedrückt: 0 oder 1. Sie haben das ja eigentlich bei der Betrachtung der Ein-Ausgabe-Ports in Kapitel 1.54 schon kennengelernt.
In seinem Innern arbeitet ein Computer entsprechend der Anzahl seiner „Finger", also dem Ziffernvorrat 0 und 1, in einem Zweiersystem, dem sogenannten Dual-System (duo = 2). Dabei muß er, wie wir es bei dem dezimalen Auto-Kilometerzähler beobachten konnten, eine neue Stelle hinzunehmen, wenn sein Ziffernvorrat 0 und 1 erschöpft ist. Das würde bei einem dualen Kilometerzähler nach dem zweiten, vierten, achten, sechzehnten, usw. Zählschritt der Fall sein:

gefahrene Kilometer	dezimaler Kilometerzähler	dualer Kilometerzähler
Auto fabrikneu:	000	00000000
nach dem 1. km	001	00000001
nach dem 2. km	002	00000010
nach dem 3. km	003	00000011
nach dem 4. km	004	00000100
nach dem 5. km	005	00000101
nach dem 6. km	006	00000110
nach dem 7. km	007	00000111
nach dem 8. km	008	00001000
•	•	•
•	•	•
nach dem 255. km	255	11111111

Wozu wir Ihnen das hier vorführen? Nun, erstens sollte man darüber ein wenig Bescheid wissen, wenn man sich mit Computern befaßt. Aber zweitens können wir Ihnen die Umwandlung vom dualen ins dezimale Zahlensystem (und umgekehrt), die der Computer automatisch zu ihrer Erleichterung ausführt, an den Ein-Ausgabe-Ports sichtbar machen.

Sie werden es geahnt haben: es war kein Zufall, daß wir den dezimalen Kilometerzähler bis 255 zählen ließen und daß der duale Kilometerzähler gerade 8 Stellen hat. Natürlich, der Zahlenumfang des KOSMOS-Computers beträgt genau 255, und jeder Port hat genau 8 Klemmen. Und damit bieten wir Ihnen folgende Möglichkeiten: wenn Sie bei den Ein-Ausgabe-Befehlen P1E und P1A für das „x" der Klemmen-Nummer eine 0 setzen (Klemme 0 gibt es ja nicht!!), so können Sie jede (dezimale) Zahl, die im Akku steht, als duale Zahl auf einen beliebigen Port geben. Die Tabelle auf Seite 50 zeigt, daß Sie mit den Zahlenwerten 000 bis 255 im Akku sage und schreibe 256 verschiedene Möglichkeiten haben, die Klemmen an den Ports unterschiedliche Zustände annehmen zu lassen. Und natürlich gibt es 256 Eingabe-Kombinationen (an Port 1), die Sie in den Akku übernehmen lassen können. Diese enorme Vielfalt der Daten-Ein- und Ausgabe-Möglichkeiten ist es, die den KOSMOS-Computer für Prozeß-Steuerungen so geeignet macht. Wir deuteten es in Kapitel 1.40 schon an.

1.64 Umwandlung von dual in dezimal und umgekehrt

Probieren Sie doch jetzt einmal aus, was im Akku erscheint, wenn Sie die Information an den Klemmen von Port 1 einlesen, aber an Port 1 nichts anschließen.

Also: Befehl 16.000 auf Speicherzelle 001 eingeben, dann 001-PC und einmal STEP drücken. Die ACC-Taste bringt die Anzeige

A 00.255

Wir erinnern uns: „Nichts angeschlossen" bedeutet (da „nichts" für den Computer keine gültige Aussage ist) eine Ja-Information, also eine logische „1" (siehe Kapitel 1.54). Logische Einsen an allen Klemmen ergeben die Dual-Information 11111111. Ein Blick in die Tabelle auf Seite 50 zeigt Ihnen, daß dies der dezimalen Zahl 255 entspricht.

Sie können jetzt jede beliebige Klemme von Port 1 auf 0 V legen (stellen Sie die Verbindung mit den beigelegten Drähten her) und den Eingabe-Befehl 16.000 wiederholen. Anhand der Tabelle können Sie dann mühelos die Dual-Dezimal-Wandlung des Computers verfolgen.

Natürlich kann Ihr Computer die Wandlung von einem Zahlensystem ins andere auch in umgekehrter Richtung erledigen: Jede Dezimalzahl von 000 bis 255, die im Akku steht, kann in eine Dualzahl umgewandelt und zu Port 1 oder Port 2 transportiert werden. Die entsprechenden Befehle lauten folgerichtig

17.000	bzw.	P1A 000	für Port 1
18.000	bzw.	P2A 000	für Port 2

Für alle Arten von Prozeßsteuerungen liegt der Vorteil dieser Befehle auf der Hand: Diverse an den Computer angeschlossene Geräte (Lampen, Relais für Modellbahnweichen und -signale, tonerzeugende Elektronik-Schaltungen usw.) können sozusagen „auf einen Schlag", also gleichzeitig ein- oder ausgeschaltet werden. Wenn Ihnen die mathematischen Hintergründe der Dezimal-Dualwandlung nicht geläufig sind, so suchen Sie einfach aus der Tabelle (Seite 50) die gewünschte 0-1-Kombination heraus, lesen die zugehörige Dezimalzahl ab, laden sie in den Akku und geben sie mit einem der obigen Befehle auf den entsprechenden Port aus.

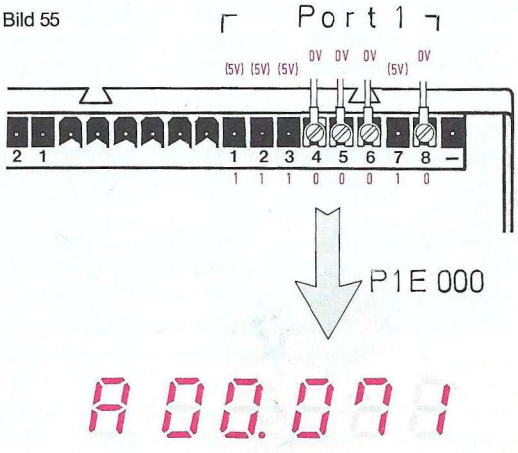
Zur Vertiefung des Verständnisses geben wir Ihnen nachfolgend drei Beispiele für Ein- und Ausgabe mit Umwandlung von einem Zahlensystem ins andere, und wenig später werden wir ein paar Aufgaben formulieren, so daß Sie mit der Wirkungsweise der Befehle vertraut werden.

Erstes Beispiel:

Ja/Nein-Kombination an Port 1: 01000111

Nach Umwandlung im Akku erscheinende dezimale Zahl: 071 (*Hinweis:* bei beiden Ports ist aus technischen Gründen die niedrigstwertige Stelle links, die höchstwertige Stelle rechts angeordnet. Dies mag zu Beginn etwas verwirrend sein, aber betrachten Sie Ihren Computer einmal von hinten: da stimmt es dann!)

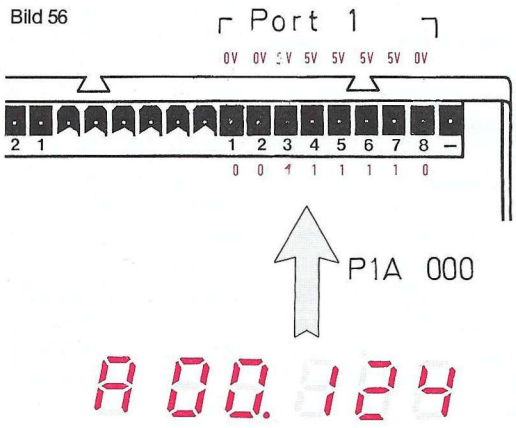
Bild 55



dezimal	dual	dezimal	dual	dezimal	dual	dezimal	dual
000	00000000	064	01000000	128	10000000	192	11000000
001	00000001	065	01000001	129	10000001	193	11000001
002	00000010	066	01000010	130	10000010	194	11000010
003	00000011	067	01000011	131	10000011	195	11000011
004	00000100	068	1000100	132	10000100	196	11000100
005	00000101	069	01000101	133	10000101	197	11000101
006	00000110	070	01000110	134	10000110	198	11000110
007	00000111	071	01000111	135	10000111	199	11000111
008	00001000	072	01001000	136	10001000	200	11001000
009	00001001	073	01001001	137	10001001	201	11001001
010	00001010	074	01001010	138	10001010	202	11001010
011	00001011	075	01001011	139	10001011	203	11001011
012	00001100	076	01001100	140	10001100	204	11001100
013	00001101	077	01001101	141	10001101	205	11001101
014	00001110	078	01001110	142	10001110	206	11001110
015	00001111	079	01001111	143	10001111	207	11001111
016	00010000	080	01010000	144	10010000	208	11010000
017	00010001	081	01010001	145	10010001	209	11010001
018	00010010	082	01010010	146	10010010	210	11010010
019	00010011	083	01010011	147	10010011	211	11010011
020	00010100	084	01010100	148	10010100	212	11010100
021	00010101	085	01010101	149	10010101	213	11010101
022	00010110	086	01010110	150	10010110	214	11010110
023	00010111	087	01010111	151	10010111	215	11010111
024	00011000	088	01011000	152	10011000	216	11011000
025	00011001	089	01011001	153	10011001	217	11011001
026	00011010	090	01011010	154	10011010	218	11011010
027	00011011	091	01011011	155	10011011	219	11011011
028	00011100	092	01011100	156	10011100	220	11011100
029	00011101	093	01011101	157	10011101	221	11011101
030	00011110	094	01011110	158	10011110	222	11011110
031	00011111	095	01011111	159	10011111	223	11011111
032	00100000	096	01100000	160	10100000	224	11100000
033	00100001	097	01100001	161	10100001	225	11100001
034	00100010	098	01100010	162	10100010	226	11100010
035	00100011	099	01100011	163	10100011	227	11100011
036	00100100	100	01100100	164	10100100	228	11100100
037	00100101	101	01100101	165	10100101	229	11100101
038	00100110	102	01100110	166	10100110	230	11100110
039	00100111	103	01100111	167	10100111	231	11100111
040	00101000	104	01101000	168	10101000	232	11101000
041	00101001	105	01101001	169	10101001	233	11101001
042	00101010	106	01101010	170	10101010	234	11101010
043	00101011	107	01101011	171	10101011	235	11101011
044	00101100	108	01101100	172	10101100	236	11101100
045	00101101	109	01101101	173	10101101	237	11101101
046	00101110	110	01101110	174	10101110	238	11101110
047	00101111	111	01101111	175	10101111	239	11101111
048	00110000	112	01110000	176	10110000	240	11110000
049	00110001	113	01110001	177	10110001	241	11110001
050	00110010	114	01110010	178	10110010	242	11110010
051	00110011	115	01110011	179	10110011	243	11110011
052	00110100	116	01110100	180	10110100	244	11110100
053	00110101	117	01110101	181	10110101	245	11110101
054	00110110	118	01110110	182	10110110	246	11110110
055	00110111	119	01110111	183	10110111	247	11110111
056	00111000	120	01111000	184	10111000	248	11111000
057	00111001	121	01111001	185	10111001	249	11111001
058	00111010	122	01111010	186	10111010	250	11111010
059	00111011	123	01111011	187	10111011	251	11111011
060	00111100	124	01111100	188	10111100	252	11111100
061	00111101	125	01111101	189	10111101	253	11111101
062	00111110	126	01111110	190	10111110	254	11111110
063	00111111	127	01111111	191	10111111	255	11111111

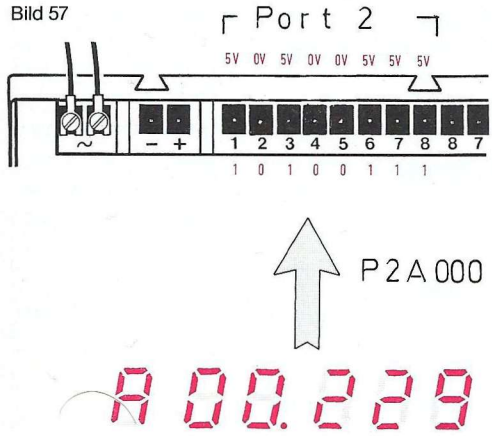
Zweites Beispiel:
Dezimale Zahl im Akku: 124
Ja/Nein-Kombination an Port 1: 01111100

Bild 56



Drittes Beispiel:
Dezimale Zahl im Akku: 229
Ja/Nein-Kombination an Port 2: 11100101

Bild 57



Aufgaben:
Ändern Sie das Blinkprogramm von Kapitel 1.57 so ab, daß Sie ausschließlich P1A 000-Befehle benutzen.

Ändern Sie das Blinkprogramm so ab, daß die Lämpchen zyklisch nacheinander die vier Zustände

a) beide Lämpchen dunkel
b) linkes Lämpchen brennt
c) rechtes Lämpchen brennt
d) beide Lämpchen brennen

einnehmen. (Lassen Sie dieses Programm zur Kontrolle doch einmal schrittweise ablaufen, also 001-PC-STEP-STEP-STEP . . .)
Ändern Sie das Blinkprogramm so ab, daß beide Lämpchen *gleichzeitig* ein- und ausgeschaltet werden.

(Lösungen auf Seite 52)

1.65 Von Bits und Bytes und anderen Fachausdrücken

Bevor Sie die neuen Befehle anwenden, wollen wir Ihnen noch ein paar Fachausdrücke erklären, mit denen die Computer-Spezialisten nur so um sich werfen.
Da wäre zunächst einmal das Wort **Bit.** Ein Bit ist eine Stelle einer Dualzahl, also die kleinstmögliche Speichereinheit, die „0" oder „1" enthalten kann. Unser Computer kann gleichzeitig 8 Bits verarbeiten. Daher hat jedes Port auch 8 Klemmen. Eine beliebige Ja/Nein-Kombination an einem Port nennt man auch **Bitmuster.**
Eine Gruppe von 8 Bits nennt man **Byte.** Jeder Port des KOSMOS-Computers ist also 1 Byte breit.
Bit kann auch als Maßeinheit verwendet werden, dann schreibt man es aber klein und sagt z. B.: der KOSMOS-Computer ist ein 8 bit-Computer. Als Maßeinheit kleingeschrieben darf es dann allerdings nicht in die Mehrzahl gesetzt werden.
Sie werden mit Sicherheit auch auf den Begriff **binär** stoßen (bi = 2) und fragen, wieso es zwei Zweiersysteme gibt. Die Antwort ist einfach.
Während die Bezeichnung „binär" lediglich darüber Auskunft gibt, daß zwei Zeichen verwendet werden (z. B. „0" und „1"), sind bei einer Dualzahl die 0- und 1-Symbole nach Wertigkeiten geordnet: ganz rechts ist die niedrigstwertige und ganz links die höchstwertige Stelle (so wie ja z. B. auch bei einer vierstelligen Dezimalzahl rechts die Einer und links die Tausender stehen). Bei einem Binärwert können auch andere Verabredungen zur Zahlendarstellung getroffen werden, so daß ein Binärwert nicht unbedingt einer Dualzahl entspricht.
In Fachkreisen wird das Dualsystem übrigens auch als „reines Binärsystem" bezeichnet.

Als **Interface** bezeichnet man eine elektronische Schaltung zur Anpassung des Computers an seine „Außenwelt". Zum Betrieb einer computergesteuerten Modellbahn-Anlage benötigt man z. B. ein sog. Relais-Interface (siehe Kap. 2.24).

Schließlich sollten Sie noch den Unterschied zwischen **Hardware** und **Software** kennenlernen.

Listing 16: **Wechselblinker mit dem P1A 000-Befehl**

Adresse	Mnemonics	**Code**	Kommentar
001	AKO 001	**04.001**	In den Akku „1" laden
002	P1A 000	**17.000**	Akku-Inhalt als Dualzahl auf Port 1 bringen
003	VZG 250	**03.250**	250 ms verzögern
004	AKO 002	**04.002**	In den Akku „2" laden
005	P1A 000	**17.000**	Akku-Inhalt als Dualzahl auf Port 1 bringen
006	VZG 250	**03.250**	250 ms verzögern
007	SPU 001	**09.001**	Zurück nach 001 springen

Listing 17: **Blinkprogramm mit 4 Zuständen**

Adresse	Mnemonics	**Code**	Kommentar
001	AKO 000	**04.000**	In den Akku „0" laden
002	P1A 000	**17.000**	Akku-Inhalt als Dualzahl auf Port 1 bringen
003	VZG 250	**03.250**	250 ms verzögern
004	AKO 001	**04.001**	In den Akku „1" laden
005	P1A 000	**17.000**	Akku-Inhalt als Dualzahl auf Port 1 bringen
006	VZG 250	**03.250**	250 ms verzögern
007	AKO 002	**04.002**	In den Akku eine „2" laden
008	P1A 000	**17.000**	Akku-Inhalt als Dualzahl auf Port 1 bringen
009	VZG 250	**03.250**	250 ms verzögern
010	AKO 003	**04.003**	In den Akku eine „3" laden
011	P1A 000	**17.000**	Akku-Inhalt als Dualzahl auf Port 1 bringen
012	VZG 250	**03.250**	250 ms verzögern
013	SPU 001	**09.001**	Zurück nach 001 springen

Listing 18: **Blinkprogramm mit zwei Lämpchen im Parallelbetrieb**

Adresse	Mnemonics	**Code**	Kommentar
001	AKO 000	**04.000**	In den Akku eine „0" laden
002	P1A 000	**17.000**	Akku-Inhalt als Dualzahl auf Port 1 bringen
003	VZG 250	**03.250**	250 ms verzögern
004	AKO 003	**04.003**	In den Akku eine „3" laden
005	P1A 000	**17.000**	Akku-Inhalt als Dualzahl auf Port 1 bringen
006	VZG 250	**03.250**	250 ms verzögern
007	SPU 001	**09.001**	Zurück nach 001 springen

Hardware ist die Elektronik, die Geräte, also alles, was körperlich vorhanden ist. Software hingegen ist „nur" Papier, also die Programme für einen Computer. Ein Computer-Elektroniker wird als Hardware-Mann und ein Programmierer als Software-Mann bezeichnet.

1.66 AIS – indirekt speichern

Die höhere Kunst des Programmierens beginnt mit einem Trick, den man als „Adreß-Modifikation" bezeichnet. **Modifizieren** heißt verändern, und in diesem Fall wird eine Adresse verändert. Zur Erklärung eines „modifizierten" Speicherbefehls greifen wir noch einmal auf das Beispiel des Lottozahlengenerators von Kapitel 1.17 zurück, stellen jetzt aber die Forderung, daß alle „gezogenen" Lottozahlen fein säuberlich der Reihe nach in den Speicherzellen 030 bis 035 gespeichert werden sollen, damit man sie später noch einmal anschauen kann.

Natürlich könnte man „normale" Speicherbefehle (ABS) benutzen. Das wäre aber umständlich. Man müßte dann nämlich den Computer mitzählen lassen, die wievielte Lottozahl gerade gezogen wurde

und für jede „Ziehung" Vergleichs-, Sprung- und Speicherbefehle vorsehen. Das ergäbe rund 30 Befehle nur für das geordnete Speichern, das eigentliche Lottozahlen-Programm käme noch hinzu. Speicherplatz ist kostbar, und so lassen wir den Computer selbst dafür sorgen, daß er für jede neue Lottozahl eine andere Speicherzelle parat hat. Er kann dies mit dem „Indirekten Speicherbefehl".

Indirekt speichern heißt für unser Computerwesen: Gehe mit dem Akku-Inhalt zu der im Befehl angegebenen Adresse. Du findest dort in der Speicherzelle einen Zahlenwert, den Du notieren sollst. Betrachte diesen Zahlenwert als neue Adresse, bei der Du den Akku-Inhalt endgültig abliefern sollst.

Das klingt kompliziert, ist aber nicht anders als bei einem Nachsende-Antrag, den man seiner Zimmerwirtin erteilt, wenn man auf Reisen geht. Bei eingehender Post streicht die Wirtin die auf dem Kuvert vermerkte Adresse durch und notiert stattdessen die neue, unter der der Empfänger z. Zt. zu erreichen ist.

Kommen wir zurück zu unserem Problem. Nehmen wir an, das Computron findet nach dem Ziehen der ersten Lottozahl unter der Adresse, die im Indirekten Speicherbefehl angegeben ist, den Zahlenwert „30" (computermäßig 00.030). Es wird den Akku-Inhalt (in unserem Fall die Lottozahl) also zur Speicherzelle 030 bringen und dort hineinschreiben. Wenn wir den Computer jetzt zu dem Zahlenwert „30" eine „1" hinzuaddieren lassen, wird das Computron die nächste Lottozahl bei Speicherzelle 031, die nächste dann bei Speicherzelle 032 abliefern usw., und . . . unser Problem ist gelöst!

Sie sehen jetzt den enormen Vorteil einer **Indirekten Adressierung:** der Zahlenwert, der die letztendliche Adresse darstellt (man nennt sie Zieladresse) ist änderbar. Man kann ihn durch eine Addition größer oder durch eine Subtraktion kleiner machen. Entsprechend ändert sich auch die Zieladresse. Mit ein und demselben Befehl (dem Indirekten Speicherbefehl) kann man also Akku-Inhalte in beliebigen Speicherzellen speichern. Der Befehl lautet

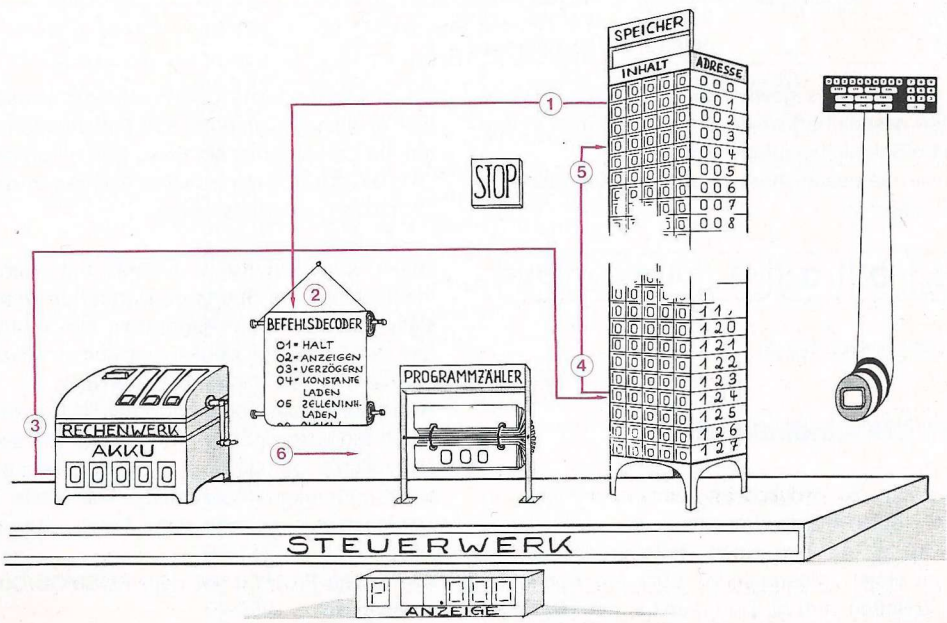
20.xxx bzw. AIS xxx
(**A**kku-Inhalt **i**ndirekt **s**peichern)

und wir wollen den Befehlszyklus noch einmal zusammenstellen:

1. Befehl aus dem Speicher holen
2. Befehl decodieren
3. Akku-Inhalt abschreiben
4. Inhalt der Speicherzelle xxx abschreiben (neue Adreßangabe)
5. Akku-Inhalt in die Speicherzelle der neuen Adresse einschreiben
6. Programmzähler um eins weiterschalten (Bild 58)

Das Lottozahlen-Programm ist gegenüber dem von Kapitel 1.17 etwas verändert worden, da wir jetzt eine externe Start- und Stopp-Taste benutzen müssen (Start des Programms wie gewohnt durch 001-PC-RUN, dann Stop des Zahlenlaufes mit Taste 1 am Port 1, erneuter Start mit Taste 1). Es ist auch etwas länger geworden, aber mit den vier Befehlen auf den Speicherzellen 017, 018, 019 und

Bild 58



Listing 19: **Automatisch speichernder Lottozahlen-Generator**

Adresse	Mnemonics	**Code**	Kommentar
001	AKO 001	**04.001**	Zahlenwerte im Datenbereich speichern
002	ABS 100	**06.100**	
003	AKO 030	**04.030**	
004	ABS 104	**06.104**	
005	AKO 049	**04.049**	
006	ABS 101	**06.101**	
007	AKO 000	**04.000**	
008	ABS 103	**06.103**	
009	ABS 102	**06.102**	
010	ADD 100	**07.100**	Addiere „1" zum Akku-Inhalt
011	ANZ	**02.000**	Zeige den Akku-Inhalt an
012	ABS 102	**06.102**	Speichere ihn in der Zählzelle
013	P1E 001	**16.001**	Bringe die Information von Klemme 1 des Port 1 in den Akku
014	VGR 103	**12.103**	Prüfe, ob Taste 1 *nicht* gedrückt ist
015	LDA 102	**05.102**	Lade den Inhalt der Zählzelle in den Akku
016	SPB 025	**11.025**	Wenn Taste *nicht* gedrückt, springe zum Vergleich nach 025
017	AIS 104	**20.104**	Speichere den Akku-Inhalt in der Speicherzelle, deren Adresse in 104 steht
018	LDA 104	**05.104**	Lade den Inhalt von 104 in den Akku
019	ADD 100	**07.100**	Addiere „1" dazu
020	ABS 104	**06.104**	Speichere den neuen Akku-Inhalt in 104
021	P1E 001	**16.001**	Bringe die Information von Klemme 1 des Port 1 in den Akku
022	VGL 103	**10.103**	Prüfe, ob Taste 1 gedrückt ist
023	SPB 021	**11.021**	Wenn Taste gedrückt, springe zur Abfrage zurück
024	SPU 007	**09.007**	Sonst springe zurück zu einem neuen Durchlauf
025	VGL 101	**10.101**	Prüfe, ob die Zählgrenze erreicht ist
026	SPB 007	**11.007**	Wenn ja, springe zum Anfang zurück
027	SPU 010	**09.010**	Sonst springe zum Addieren
100		**00.001**	Schrittweite und Vergleichszahl
101		**00.049**	Vergleichszahl
102		**00.xxx**	Zählzelle
103		**00.000**	Vergleichszahl
104		**00.030**	Startadresse

020 können (falls gewünscht) bis zu 70 (!) Lottozahlen gespeichert werden, ohne daß man irgendwelche Befehle hinzufügen muß.
Wie wir die gespeicherten Zahlen anschauen können, indem wir

drücken, wissen Sie ja.

1.67 LIA – indirekt laden

Um auch die Möglichkeit kennenzulernen, „indirekt" Speicherzellen-Inhalte in den Akku zu laden, stellen wir uns einen Reiseveranstalter vor, der im Hotel „Meeresperle" über ein Kontingent von 90 Betten verfügt. Und damit in der Hektik der Hauptreisezeit keine Überbuchungen vorkommen, läßt er seinen Computer die Ferienbetten verwalten. Im Computer ist ein freies Bett durch den Wert „0" (00.000) und ein belegtes Bett durch den Wert „1" (00.001) gekennzeichnet.

Immer wenn ein Kunde kommt und nach einem momentan freien Bett fragt, startet der Reiseveranstalter sein Computer-Programm mit 001-PC-RUN, und der Computer zeigt (nach kurzem „Nachdenken") die Anzahl der freien Betten an.
Wenn der Kunde bucht, muß der Reiseveranstalter durch Eingabe einer „1" in eine der „Betten-Speicherzellen" von 030 bis 119 die Buchung im Computer vermerken (auch diese Handgriffe könnte man noch vereinfachen, siehe Anregung später).

Wie dieses Problem mit dem KOSMOS-Computer zu lösen ist?

Ganz einfach. Mit dem Indirekten Lade-Befehl (den Sie sicher an dieser Stelle erwartet haben) werden nacheinander die Inhalte der Speicherzellen 030 bis 119 in den Akku geladen und mit „0" verglichen.
Bei Gleichheit (freies Bett), wird der Inhalt der „Betten-Zählzelle" in den Akku geladen, eine „1" addiert und der neue Akku-Inhalt in derselben Zelle wieder gespeichert. Durch einen weiteren Vergleich stellt der Computer fest, wann er die Zelle 119 abgefragt hat.
Dann lädt er den Inhalt der „Betten-Zählzelle" in den Akku und zeigt (in einer Dauerschleife) den Akku-Inhalt an.
Der Indirekte Lade-Befehl lautet

„Lade den Inhalt derjenigen Speicherzelle in den Akku, deren Adresse sich in der Zelle xxx befindet"

Code und Kürzel sind

19.xxx bzw. LIA xxx
(**L**ade Speicherzelleninhalt **i**ndirekt in den **A**kku)

Der Befehlszyklus ist dem des Indirekten Speicherbefehls sehr ähnlich und wird daher nicht noch einmal erläutert.

Anregung (ohne Lösung):
Ändern Sie das Programm so ab, daß der Reiseveranstalter, wenn ein Kunde ein Bett gebucht hat, nur eine Taste (z.B. Port 1/Klemme 8) zu drücken braucht, um den neuen Stand an freien Betten angezeigt zu bekommen.
Hinweis:
Für diese Programm-Erweiterung müssen einige der Speicherzellen geopfert werden, die das Betten-Kontingent darstellen.

Listing 20: **Reisebüro-Computer**

Adresse	Mnemonics	**Code**	Kommentar
001	AKO 001	**04.001**	Zahlenwerte im Datenbereich abspeichern
002	ABS 123	**06.123**	
003	AKO 000	**04.000**	
004	ABS 124	**06.124**	
005	ABS 126	**06.126**	
006	AKO 119	**04.119**	
007	ABS 125	**06.125**	
008	AKO 030	**04.030**	
009	ABS 127	**06.127**	
010	LIA 127	**19.127**	Akku indirekt laden. Die Lade-Adresse steht in Zelle 127
011	VGL 124	**10.124**	Ist Akku-Inhalt gleich „0"?
012	SPB 019	**11.019**	Wenn ja, nach 019 springen
013	LDA 127	**05.127**	Wenn nein, den Inhalt von 127 in den Akku laden
014	ADD 123	**07.123**	Zum Akku-Inhalt „1" addieren
015	ABS 127	**06.127**	Neuen Akku-Inhalt in 127 speichern
016	VGR 125	**12.125**	Ist Akku-Inhalt größer als „119"?
017	SPB 023	**11.023**	Wenn ja, nach 023 springen
018	SPU 010	**09.010**	Wenn nein, nach 010 springen
019	LDA 126	**05.126**	Inhalt von 126 in den Akku laden
020	ADD 123	**07.123**	Zum Akku-Inhalt eine „1" addieren
021	ABS 126	**06.126**	Neuen Akku-Inhalt in 126 speichern
022	SPU 013	**09.013**	Nach 013 springen
023	LDA 126	**05.126**	Inhalt von 126 in den Akku laden
024	ANZ	**02.000**	Akku-Inhalt anzeigen
025	SPU 024	**09.024**	Nach 024 springen (Dauerschleife, Akku-Inhalt soll angezeigt bleiben)
123		**00.001**	Schrittweite
124		**00.000**	Vergleichszahl
125		**00.119**	119 = Schlußadresse
126		**00.000**	„Betten-Zählzelle"
127		**00.030**	030 = Startadresse

1.68 SIU – indirekt springen

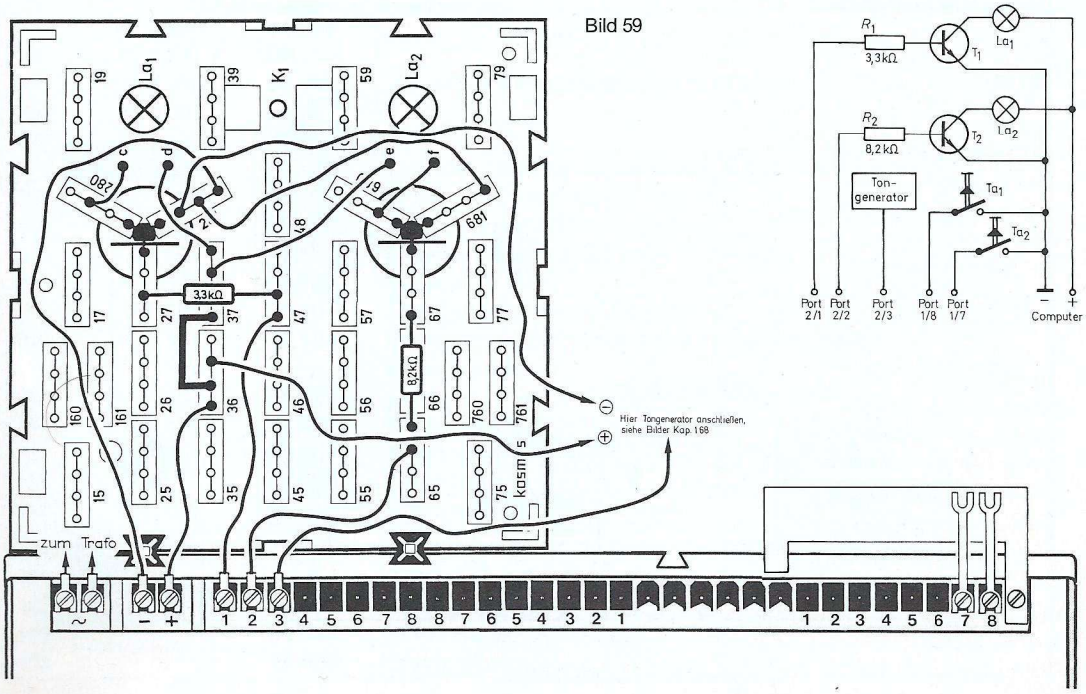
Sie kennen jetzt alle Befehle Ihres Computers – bis auf einen. Und mit diesem letzten Befehl begeben wir uns in Gedanken nach Cape Canaveral, um unseren Computer den Countdown eines Raketenstarts steuern zu lassen. Sie wissen um die Bedeutung eines Countdowns. Ab einem festgesetzten Zeitpunkt läuft die Uhr rückwärts, und bei der Sekunde Null wird die Rakete gezündet. Wenn jedoch – und oft genug ist dies ja der Fall – eine technische Schwierigkeit auftritt, wird die Uhr angehalten, Unterbrechungsalarm gegeben, der Fehler repariert, und dann werden der Uhrenlauf und sämtliche Vorbereitungssteuerungen exakt an der Stelle fortgesetzt, an der sie unterbrochen worden waren. Und das ist genau das, was wir zeigen wollen: wir lassen den Computer bei 240 beginnen, im Sekundentakt rückwärts zählen und außerdem zwei getrennte Steuerimpulse ausgeben. Bei uns werden durch die Steuerimpulse zwei Lämpchen zum abwechselnden Blinken gebracht, für den „Ernstfall" können Sie sich Impulse für wichtige Triebwerkfunktionen vorstellen.

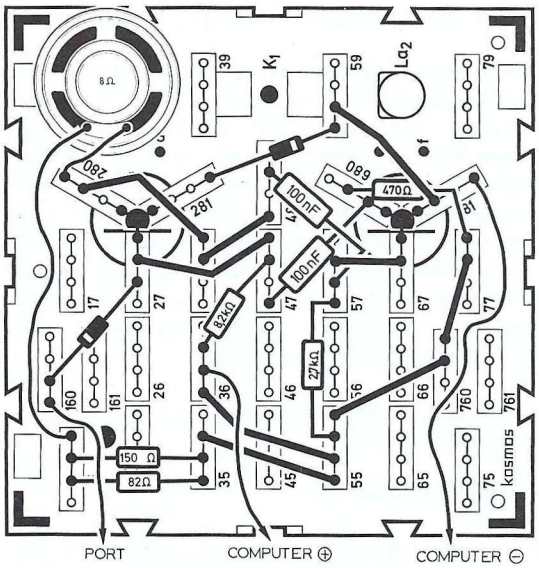
Listing 21: **Countdown**

Adresse	Mnemonics	**Code**	Kommentar
001	AKO 000	**04.000**	Zahlenwerte im Datenbereich speichern
002	ABS 100	**06.100**	
003	AKO 001	**04.001**	
004	ABS 101	**06.101**	
005	AKO 240	**04.240**	
006	ABS 102	**06.102**	
007	ANZ	**02.000**	Zeige zu Beginn „240" an
008	VZG 250	**03.250**	Verzögere 250 ms
009	VZG 250	**03.250**	Verzögere nochmals 250 ms
010	AKO 015	**04.015**	Lade die Adresse, zu der aus dem Unterprogramm zurückgesprungen werden soll
011	ABS 103	**06.103**	Speichere sie in 103
012	P1E 008	**16.008**	Frage die Alarmtaste ab
013	VGL 100	**10.100**	Ist sie gedrückt?
014	SPB 050	**11.050**	Wenn ja, springe ins Unterprogramm
015	VZG 250	**03.250**	Wenn nein, fahre mit 250 ms Verzögerung fort
016	VZG 250	**03.250**	Verzögere nochmals 250 ms
017	LDA 102	**05.102**	Lade den Inhalt der Zählzelle in den Akku
018	SUB 101	**08.101**	Subtrahiere „1"
019	ANZ	**02.000**	Zeige den neuen Akku-Inhalt an
020	VGL 100	**10.100**	Prüfe, ob „0" bereits erreicht ist
021	SPB 070	**11.070**	Wenn ja, springe zum Raketenstart nach 070
022	ABS 102	**06.102**	Wenn nein, speichere Akku-Inhalt in Zählzelle
023	AKO 000	**04.001**	Lade „0" in den Akku, um das Blinkprogramm zu beginnen
024	P2A 001	**18.001**	Gib die Information an Klemme 1 des Port 2
025	VZG 250	**03.250**	Verzögere 250 ms
026	NEG	**14.000**	Ändere die „0" im Akku durch Negieren in eine „1"
027	P2A 001	**18.001**	Gib diese Information an Klemme 1 des Port 2
028	VZG 250	**03.250**	Verzögere 250 ms
029	AKO 034	**04.034**	Lade die Adresse, zu der aus dem Unterprogramm zurückgesprungen werden soll
030	ABS 103	**06.103**	Speichere sie in 103
031	P1E 008	**16.008**	Frage die Alarmtaste ab
032	VGL 100	**10.100**	Ist sie gedrückt?
033	SPB 050	**11.050**	Wenn ja, springe ins Unterprogramm
034	AKO 001	**04.001**	Wenn nein, beginne ein weiteres Blinkprogramm
035	P2A 002	**18.002**	Gib die Akku-Information an Klemme 2 des Port 2
036	VZG 250	**03.250**	Verzögere 250 ms
037	NEG	**14.000**	Ändere die „0" im Akku durch Negieren in eine „1"

038	P2A 002	**18.002**	Gib diese Information an Klemme 2 des Port 1
039	VZG 250	**03.250**	Verzögere 250 ms
040	AKO 045	**04.045**	Lade die Adresse, zu der aus dem Unterprogramm zurückgesprungen werden soll
041	ABS 103	**06.103**	Speichere sie auf 103
042	P1E 008	**16.008**	Frage die Alarmtaste ab
043	VGL 100	**10.100**	Ist sie gedrückt?
044	SPB 050	**11.050**	Wenn ja, springe ins Unterprogramm
045	SPU 017	**09.017**	Wenn nein, fahre mit dem Countdown auf Zeile 017 fort
			Ab hier: Unterprogramm
050	AKO 001	**04.001**	Lade „1" in den Akku
051	P2A 002	**18.003**	Gib die Akku-Information auf Klemme 3 des Port 2
052	VZG 250	**03.250**	Verzögere 250 ms
053	NEG	**14.000**	Ändere die „1" im Akku durch Negieren in eine „0"
054	P2A 003	**18.003**	Gib diese Information auf Klemme 3 des Port 2
055	VZG 250	**03.250**	Verzögere 250 ms
056	P1E 007	**16.007**	Frage die Taste „Alarm aufheben" ab
057	VGR 100	**12.100**	Ist sie nicht gedrückt?
058	SPB 050	**11.050**	Wenn nicht gedrückt, dann springe zum Alarmanfang zurück
059	SIU 103	**21.103**	Wenn gedrückt, zu der Stelle des Programms zurückkehren, an der unterbrochen wurde
			Ab hier: Programmteil „Daueralarm"
070	AKO 001	**04.001**	Lade „1" in den Akku für den Raketenstart
071	P2A 003	**18.003**	Gib die Akku-Information auf Klemme 3 des Port 2 (Daueralarm = Raketen-Start)
072	HLT	**01.000**	Halte an
100		**00.000**	Vergleichszahl
101		**00.001**	Schrittweite
102		**00.0xx**	Speicherzelle für aktuellen Zählerstand
103		**00.xxx**	Rücksprungadresse

Bild 59





Bilder 59–61. Raketen-Countdown, aufgebaut aus KOSMOS „Radio + Elektronik 100", „Spiele mit Elektronik" und 2 zus. Si-Dioden

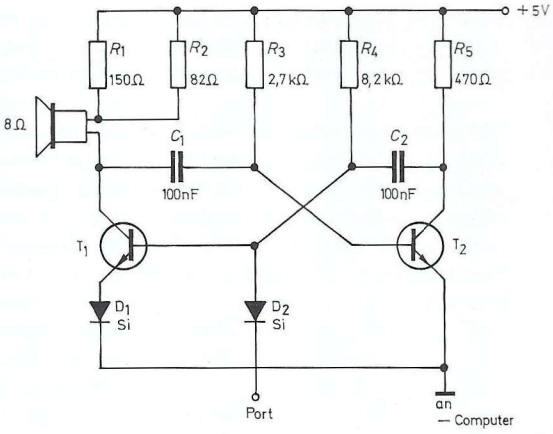
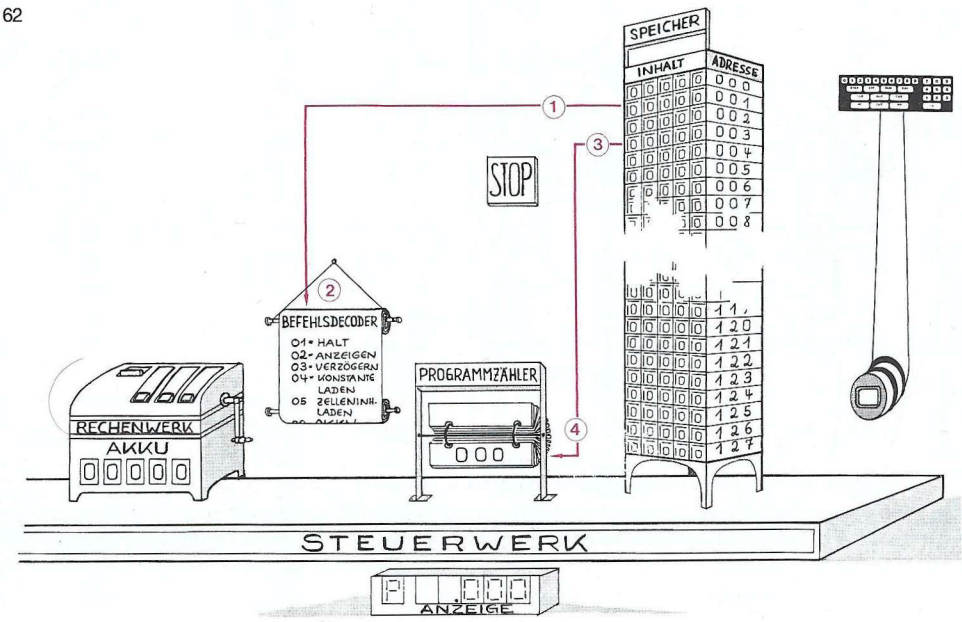


Bild 61. Tongenerator

Wir simulieren eine technische Panne und drücken Taste 8. Augenblicklich stoppt der Countdown, die Steuersignale bleiben aus, und stattdessen ertönt ein Alarmsignal. Das Verblüffende ist nun – und das erreichen wir mit dem „Indirekten Unbedingten Sprung" – daß wir durch Drücken der Taste 7 (Freigabe des Countdown) das Programm an genau der Adresse fortsetzen können, an der es unterbrochen wurde. Ein kleines Modell also für den großen Raketenbahnhof.

Sie erinnern sich sicher an den Begriff der „Adreßmodifikation" (s. Kapitel 1.67), also die Änderung der in einem Befehl angegebenen Adresse.
Die **Rücksprung-Adresse** vom Alarmprogramm (**Unterprogramm**) in das Countdown-Programm (**Hauptprogramm**) ist änderbar. Bevor der Computer aus dem Hauptprogramm springt, hat er sich die Rücksprung-Adresse „gemerkt", d.h. in einer Speicherzelle (dafür kann eine beliebige Speicherzelle des Speichers gewählt werden) gespeichert. Wird nun (per Tastendruck) das Alarmprogramm beendet, so wird durch den Indirekten Unbedingten Sprung die Adreß-Speicherzelle abgefragt, die die Rücksprungadresse enthält.

Bild 62



Der Indirekte Unbedingte Sprungbefehl lautet:

„Springe auf die in der Speicherzelle xxx angegebene Adresse"

Code und Kürzel sind

21.xxx bzw. SIU xxx (**S**prung **i**ndirekt **u**nbedingt)

Befehlszyklus des Indirekten Sprungbefehls SIU:

1. Befehl aus dem Speicher holen
2. Befehl decodieren
3. Inhalt der Speicherzelle xxx abschreiben (neue Adreßangabe)
4. Programmzähler auf die neue Adresse einstellen (Bild 62).

1.69 Multiplizieren durch Addieren

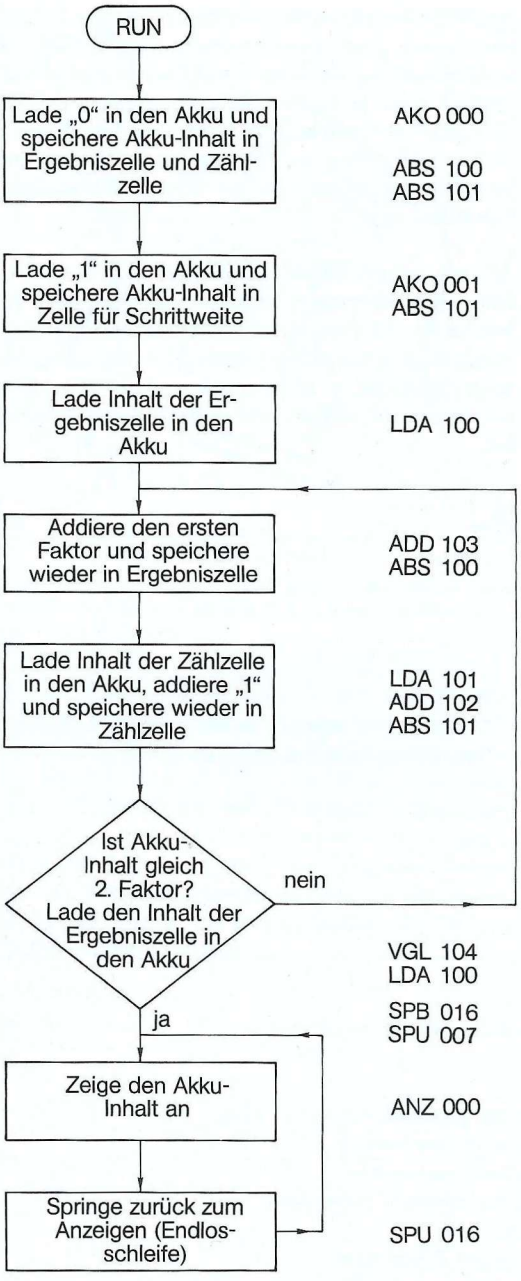
Eine Multiplikation ist – hoffen wir, daß es Ihnen in der Grundschule so vermittelt wurde – nichts weiter als eine abgezählte Folge von Additionen. Bei einem Taschenrechner läuft beim Druck auf die Multiplikationstaste eine solche Additionsroutine ab. Unserem Computer müssen wir diese Routine erst einprogrammieren. Für ihn bedeutet z.B. die Aufgabe 25 × 9, daß er die Zahl Fünfundzwanzig neunmal addieren soll. Man läßt ihn also in einer Additionsschleife laufen und fragt bei jedem Durchlauf durch einen Vergleichsbefehl ab, ob es die neunte Addition war. Wenn nicht, wird noch einmal addiert, wenn ja, ist die Aufgabe beendet, und man läßt das Programm in einer Anzeigeschleife laufen. Eine Befehlsfolge zur Lösung einer bestimmten Aufgabe nennt man in der Fachsprache **Algorithmus.**

Mit den angegebenen Kommentaren sind Sie sicher ohne weiteres in der Lage, das Programm zu verstehen. Als Programmierer müssen Sie darüber wachen, daß das Endergebnis einer Multiplikation unter 256 bleibt . . . sofern man nicht den Programmiertrick anwendet, der in Kapitel 2.15 beschrieben ist.

Aufgabe:
Zeichnen Sie zu dem Multiplikationsprogramm das Flußdiagramm (Lösung Seite 60).

Listing 22: **Multiplikation 25 × 9**

Adresse	Mnemonics	**Code**	Kommentar
001	AKO 000	**04.000**	Zahlenwerte im Datenbereich speichern
002	ABS 100	**06.100**	
003	ABS 101	**06.101**	
004	AKO 001	**04.001**	
005	ABS 102	**06.102**	
006	LDA 100	**05.100**	Lade den Inhalt der Ergebniszelle in den Akku
007	ADD 103	**07.103**	Addiere „25" dazu (erster Faktor)
008	ABS 100	**06.100**	Speichere das Additionsergebnis
009	LDA 101	**05.101**	Lade den Inhalt der Zählzelle in den Akku
010	ADD 102	**07.102**	Addiere „1" dazu
011	ABS 101	**06.101**	Speichere den neuen Zählerstand
012	VGL 104	**10.104**	Prüfe (vergleiche), ob bereits 9 mal (zweiter Faktor) addiert wurde
013	LDA 100	**05.100**	Lade den Inhalt der Ergebniszelle in den Akku
014	SPB 016	**11.016**	Wenn Gleichheit bei 012, dann springe zur Anzeige
015	SPU 007	**09.007**	Wenn nicht, fahre auf 007 mit dem Addieren fort
016	ANZ	**02.000**	Zeige das Ergebnis an
017	SPU 016	**09.016**	Springe zurück zum Anzeigen
100			Ergebniszelle (wird zu Beginn auf 00.000 gesetzt)
101			Zählzelle (wird zu Beginn auf 00.000 gesetzt)
102		**00.001**	Schrittweite
103		**00.025**	Erster Faktor
104		**00.009**	Zweiter Faktor



1.70 Dividieren durch Subtrahieren

Bekanntlich ist die Division die Umkehrung der Multiplikation: Es wird abgezählt, wie oft man eine Zahl von einer anderen subtrahieren kann. Das Zählergebnis ist gleichzeitig das Ergebnis der Division, und außerdem gibt es in den meisten Fällen noch das, „was übrig bleibt", den Rest. Unser Computer muß also am Ende einer Division zwei Zahlen vorweisen können: das Ergebnis und den Rest.

Bei unserem Beispielprogramm wird nach Eingabe von 001-PC-RUN das Divisionsergebnis sofort angezeigt. Drückt man die STP-Taste, so wird automatisch der Rest der Division angezeigt (der Rest wird im Programm auf Zelle 000 gespeichert, Bedeutung der Zelle 000 im Zusammenhang mit der STP-Taste siehe Kapitel 1.34). Der Dividend (also die Zahl, die geteilt werden soll) wird in Zelle 100, der Divisor (also die Zahl, die teilt) in Zelle 101 eingegeben.

Beispiel:

213 : 15
Dividend Divisor

213 (00.213) wird in Zelle 100, 15 (00.015) wird in Zelle 101 eingegeben.

Wie man mit dem KOSMOS-Computer ein Divisionsergebnis mit beliebig vielen Stellen hinter dem Komma erzielen kann, erfahren Sie in Kapitel 2.16.

1.71 Fehler finden – erfolgreich programmieren

Es ist geradezu eine Seltenheit, daß ein Programm auf Anhieb funktioniert. In der Regel treten beim Eingeben Tipp-Fehler auf oder man läßt versehentlich eine Zeile aus oder speichert einen Befehl im Eifer des Gefechts zweimal ein oder man vertut sich mit dem Befehlscode – kurz: als guter Programmierer muß man in der Lage sein, Fehler möglichst rasch zu finden und zu beseitigen. Der KOSMOS-Computer gibt Ihnen bei der Fehlersuche einige sehr wertvolle Hilfestellungen. Die wichtigste ist die Fehleranzeige „F.001" bis „F.007".

Lassen Sie uns ein einfaches Beispiel wählen. Beim Divisionsprogramm von Kapitel 1.70 haben Sie statt 02.000 versehentlich 00.200 eingetippt. Sie starten das Programm, und der Computer bringt Ihnen die Fehleranzeige „F .002" (Bedeutung der verschiedenen Fehleranzeigen siehe beiliegende Tabelle). Der Computer hat also einen ungültigen Operationscode erkannt. Prüfen Sie nun, *an welcher Stelle* des Programms dieser Fehler auftritt und drücken Sie die PC-Taste. Der Computer wird Ihnen „P .017" anzeigen. Sie brauchen sich nunmehr nur noch den Inhalt von Speicherzelle 017 anzeigen zu lassen (017-OUT), und Ihr Irrtum wird sofort offenbar werden. Achtzig Prozent aller Fehler lassen sich nach dieser Methode finden. Was aber machen Sie in einem Fall, der in die restlichen zwanzig Prozent fällt?

Listing 23: **Division**

Adresse	Mnemonics	**Code**	Kommentar
001	AKO 000	**04.000**	Zahlenwerte im Datenbereich
002	ABS 102	**06.102**	
003	ABS 101	**04.001**	
004	AKO 103	**06.103**	
005	LDA 100	**05.100**	Dividenden in den Akku laden . . .
006	ABS 000	**06.000**	und in der Restzelle speichern
007	LDA 000	**05.000**	Restzelle in den Akku laden
008	VKL 101	**13.101**	Kleiner als Divisor?
009	SPB 016	**11.016**	Wenn ja, keine Division möglich, zum Anzeigen springen
010	SUB 101	**08.101**	Wenn nein, Divisor subtrahieren
011	ABS 000	**06.000**	Zwischenergebnis in „Restzelle" speichern
012	LDA 102	**05.102**	Inhalt der „Ergebniszelle" in den Akku laden
013	ADD 103	**07.103**	Eine „1" addieren . . .
014	ABS 102	**06.102**	. . . und wieder in „Ergebniszelle" speichern
015	SPU 007	**09.007**	Einen neuen Durchlauf beginnen
016	LDA 102	**05.102**	Inhalt der „Ergebniszelle" in den Akku laden
017	ANZ	**02.000**	Ergebnis anzeigen
018	SPU 017	**09.017**	Zurückspringen zum Anzeigen
100		**00.xxx**	Dividend
101		**00.xxx**	Divisor
102			„Ergebniszelle"
000			„Restzelle"

Prüfen Sie zunächst einmal Schritt für Schritt nach, ob Ihr Programm wirklich ordnungsgemäß im Speicher ist.

Wählen Sie die Speicherzelle an, die Ihren ersten Befehl enthält (Adreß-Nummer und OUT-Taste), und drücken Sie dann fortlaufend OUT. Vergleichen Sie Ihr Listing mit den angezeigten Zelleninhalten. Wenn Sie zwischendurch die Übersicht verlieren, welcher Befehl Ihnen gerade angezeigt wird (wo der Ein-Ausgabezeiger steht, siehe Kapitel 1.11), so tippen Sie 9-OUT, und die Adresse der Speicherzelle, deren Inhalt Sie als letztes gesehen haben, wird angezeigt (mit dem Kennbuchstaben C).

Und wenn Sie bei dieser Prozedur nichts Verdächtiges entdecken können, sollten Sie als nächstes anhand unserer beiliegenden Befehlsliste überprüfen, ob in Ihrem Listing die Mnemonics und Codes übereinstimmen.

Viele Fehler treten dadurch auf, daß man sich bei der Adresse eines Lade- bzw. Speicherbefehls irrt, also falsche Zellen-Inhalte lädt oder den Akku-Inhalt zu einer nicht sinnvollen Adresse transportiert.

Wenn Sie ein Programm mit Sprungbefehlen geschrieben und dieses Programm im Laufe der Entwicklung mehrmals verbessert haben, könnte sich eine falsche Sprungadresse eingeschlichen haben. Überprüfen Sie also, ob Ihre Sprungadressen einen Sinn ergeben.

Wenn alle diese Maßnahmen nicht zum Erfolg führen, müssen Sie das Programm Schritt für Schritt mit der STEP-Taste abarbeiten. Stellen Sie den Programmzähler auf die Adresse des ersten Befehls (Adreß-Nummer und PC-Taste) und betätigen Sie die STEP-Taste. Auf der Anzeige erscheint nun der Programmzählerstand. War der abgearbeitete Befehl kein Sprung, so wurde der Programmzähler um eins erhöht. Im Falle eines Sprunges erscheint die Sprungadresse als neuer Programmzählerstand. Überprüfen Sie nach jeder STEP-Betätigung den Akku-Inhalt mit der ACC-Taste. In sehr vielen Fällen gibt die Akku-Anzeige Aufschluß, was im Programm passiert ist. Bei den ABS-(Speicher-)Befehlen können Sie in der betreffenden Zelle nachschauen, ob der Akku-Inhalt richtig gespeichert, bei den LDA-(Lade-)Befehlen überprüfen, ob der Inhalt der richtigen Zelle geladen wurde.

Überlegungsfehler kann Ihnen der Computer natürlich nicht signalisieren. Er ist eine Maschine und führt nur gehorsam Ihre Befehle aus – auch solche, die nicht zum Ziel führen können. Aber mit den vielfältigen Anzeigemöglichkeiten hilft er Ihnen, Ihr Programm nochmals auf seine Logik und Folgerichtigkeit zu überprüfen.

Computerfachleute schwören auf das bissige Zitat „Alle auftretenden Fehler liegen **vor** der Maschine". Da ist natürlich durchaus was dran. Deshalb verdächtigen Sie bitte nicht zuallererst Ihren Compu-

ter, wenn etwas nicht funktioniert. Auf 99,3% aller Fälle trifft das obige Zitat zu.
Zum Abschluß dieser Betrachtungen noch ein Ausdruck der Computerfachsprache: das Aufspüren von Fehlern am Computer heißt „debugging" (englisch bug eigentlich Wanze, aber in der amerikanischen Umgangssprache auch „Macke", Fehler).

1.72 Tips und Tricks für den Programmierer

Listing 24: **1-Minuten-Verzögerung**

Adresse	Mnemonics	**Code**	Kommentar
•	•	•	Irgendein Programmteil
•	•	•	
•	•	•	
068	VZG 250	**03.250**	Verzögere 250 ms
069	LDA 100	**05.100**	Lade den Inhalt von Zelle 100 in den Akku
070	SUB 101	**08.101**	Subtrahiere „1"
071	ABS 100	**06.100**	Speichere neuen Akku-Inhalt in 100
072	VGR 102	**12.102**	Ist der Akku-Inhalt größer als „0"
073	SPB 068	**11.068**	Wenn ja, springe zurück zum Verzögern
100		**00.240**	Zählerzelle
101		**00.001**	Schrittweite
102		**00.000**	Vergleichszahl

Zu den besonderen Künsten des Programmierers gehört eine optimale Ausnutzung des vorhandenen Speicherplatzes. Mit der Angabe einiger Tricks wollen wir Sie nachfolgend in die Geheimnisse der Programm-Optimierung einweihen.
Zuerst eine goldene Regel. Machen Sie sich bei der Entwicklung eines Programmes ein exaktes Listing und versuchen Sie nicht, einen Teil des Programms „aus dem Kopf" dem Computer einzugeben. Wir können Ihnen versichern, daß dies meistens schiefgeht. Keinesfalls schafft man es jedoch dergestalt „freihändig", die günstigste Möglichkeit zu überblicken (zum Thema „eigene Programmentwicklung" siehe Kapitel 1.78).
Benutzen Sie für verschiedene Programmteile wenn möglich denselben Datenbereich (Datenbereich siehe Kapitel 1.45). Wenn Sie also beispielsweise im Zuge eines Programms mehrmals die gleiche Schrittweite (Schrittweite siehe Kapitel 1.45) benötigen, so reicht es aus, wenn sie im Datenbereich einmal hinterlegt ist. Sinngemäß gilt das natürlich für alle Zahlenwerte, die in einem Programm auftauchen können.
Für Routinearbeiten – wenn z.B. mehrmals, aber an verschiedenen Stellen durch die gleiche Zahl dividiert werden soll – sehen Sie ein Unterprogramm mit dem Indirekten Unbedingten Sprung vor (siehe Kapitel 1.68). Durch die Unterprogrammtechnik kann enorm viel Platz eingespart werden.
Scheuen Sie sich nicht, für lange Verzögerungszeiten ein kleines Schleifenprogramm zu schreiben. Für eine Verzögerungszeit von beispielsweise einer Minute würden Sie mit einem linearen Programm zweihundertvierzig VZG 250-Befehle benötigen, also mehr Platz brauchen, als überhaupt zur Verfügung steht. Elegant löst man dieses Problem gemäß Listing 24.
Wenn Sie die benötigten Datenspeicherzellen mitzählen, brauchen Sie für die einminütige Verzögerung nur noch 9 Speicherzellen! Übrigens: Würden Sie das obige Programm testen, könnten Sie feststellen, daß die erzielte Verzögerungszeit in Wirklichkeit eine Minute und acht Sekunden beträgt. Des Rätsels Lösung: Der Computer braucht zur Ausführung der Befehle LDA, SUB, ABS, VGR und SPB natürlich auch eine – wenn auch äußerst geringe – Zeit. Da die Verzögerungsschleife aber 240mal durchlaufen wird, schlägt diese Zeit mit vollen 8 Sekunden zu Buche (siehe auch Kapitel 1.77). Abhilfe: machen Sie die Verzögerungszeit kürzer (z.B.: VZG 217).
Vermeiden Sie unnötige „Transportbefehle" wie LDA, ABS, LIA, AIS und AKO. Wenn Sie also beispielsweise eine Information von einem Port in den Akku geholt haben, lassen Sie diese auch *sofort* verarbeiten, anstatt sie erst einmal in einer Speicherzelle des Datenbereiches „aufzubewahren". Meistens gewinnen Sie mit derartigen Überlegungen höchstens ein, zwei Speicherzellen. Aber ein einfallsreiches Programm, das Sie entwickelt ha-

ben, könnte u.U. genau an zwei Speicherzellen scheitern. In diesem Sinne noch ein weiterer Trick. Wenn bei einer Programmverzweigung in beiden Programmzweigen als nächstes der gleiche Befehl gebraucht wird, so legen Sie diesen Befehl *zwischen* den Vergleich und den Bedingten Sprung. Wir haben dies schon einmal gemacht, ohne daß Sie es vielleicht bewußt registriert haben. Schauen Sie sich das Flußdiagramm Kapitel 1.69 an. In die Raute der Programmverzweigung ist außer dem Vergleichsbefehl auch noch ein Lade-(LDA)-Befehl eingetragen. Sowohl vor dem Addieren als vor dem Anzeigen wird dort nämlich der Inhalt der „Ergebniszelle" im Akku benötigt.

1.73 Die Rucksack-Methode

Anfängern passiert es recht häufig, daß sie beim Entwickeln eines Programms einen Befehl vergessen, der später, wenn das Programm bereits eingetippt ist und ausgetestet wird, noch eingefügt werden müßte.
Zwei Methoden bieten sich für einen solchen Fall an. Man kann einen (oder mehrere) Befehle „hinten anhängen", d.h. an das Programmende anfügen. Ein solches Anhängsel nennt man einen **Rucksack.**
Beispiel: Beim Würfelprogramm von Kapitel 1.15 haben Sie den Additionsbefehl vergessen. Auf Adresse 004 steht also statt des Additionsbefehls bereits der Vergleichsbefehl. Machen Sie nun folgendes: Überschreiben Sie den Vergleichsbefehl durch einen Sprung nach 007 und bringen Sie den fehlenden Additionsbefehl nun in Zelle 007, den Vergleichsbefehl in Zelle 008 und in Zelle 009 schließlich einen Rücksprungbefehl nach Adresse 005 des Hauptprogramms.

1.74 NOP – tue nichts

Bei der zweiten Methode plant man von vornherein ein, daß es Probleme geben könnte und fügt in gewissen Abständen im Programm Befehle ein, die buchstäblich „nichts" bewirken. Ein solcher Befehl ist z.B.

VZG 000 („Warte nicht")

Er stört den Programmablauf überhaupt nicht, kann jedoch gegebenenfalls durch einen anderen Befehl ersetzt werden, der vergessen wurde. Ein Befehl ohne Wirkung heißt in der Computer-Fachsprache **NOP-Befehl** (NOP = No Operation).

1.75 Mehrere Programme im Speicher

Eine (an sich selbstverständliche) Möglichkeit, die der Computer bietet, soll nicht unerwähnt bleiben. Man kann im Speicher *ein langes,* jedoch auch *mehrere kurze* Programme gleichzeitig unterbringen. Sie können z.B. Programm I von Adresse 001 bis 027, Programm II von 028 bis 074 und Programm III von 075 bis 099 in den Speicher eingeben. Möchten Sie, daß der Computer Programm II abarbeitet, so stellen Sie den Programmzähler auf 028 (028-PC) und starten es durch Drücken der RUN-Taste.

1.76 Programm-Schieber

Besonders trickreich können Sie vorgehen, wenn z.B. gerade am Programmanfang ein oder mehrere Befehle vergessen wurden. Ein solcher Fehler ist besonders ärgerlich, weil man eigentlich das gesamte Programm neu eingeben müßte. Hier hilft

Listing 25: **Rucksackprogramm**

Adresse	Mnemonics	**Code**	Kommentar	
001	AKO 001	**04.001**	Lade „1" in den Akku	
002	ABS 101	**06.101**	Speichere den Akku-Inhalt in 101	
003	ABS 000	**06.000**	Speichere den Akku-Inhalt in 000	
004	SPU 007	**09.007**	Springe zum „Rucksack"	
005	SPB 001	**11.001**	Wenn ja, beginne wieder von vorn	
006	SPU 003	**09.003**	Wenn nein, mache bei 003 weiter	
007	ADD 101	**07.101**	Addiere eine „1"	Rucksack
008	VGL 100	**10.100**	Ist Akku-Inhalt schon „7"?	
009	SPU 005	**09.005**	Springe zurück nach 005	

die Methode des Programmschiebens: das gesamte Programm wird einfach um ein paar Zeilen nach unten verschoben, so daß am Programmanfang einige Speicherplätze frei werden.
Natürlich braucht man ein kleines Zusatz-Programm, das dieses Schieben bewerkstelligt. Es kann irgendwo in den Speicher eingegeben werden. In unserem unten angeführten Beispiel beginnt es auf Adresse 088.
Zum Ausprobieren lassen wir das Zählprogramm von Kapitel 1.43 um sechs Zeilen verschieben. Es beginnt ursprünglich bei Speicherzelle 001 und endet bei Zelle 007. Wir wollen es das **Urprogramm** und das um sechs Zeilen verschobene das **Zielprogramm** nennen. Das Zielprogramm soll also bei Zelle 007 beginnen und bei 013 enden (in der Praxis lohnt sich das Schieben natürlich nur bei wesentlich längeren Programmen!).
Geben Sie also zunächst das Zählprogramm ab Adresse 001 und dann das Schiebeprogramm ab Adresse 088 in den Computer ein.

Starten Sie das Schiebeprogramm mit

Mit der Anzeige

bleibt der Computer stehen – das Verschieben ist bereits geschehen! Schauen Sie sich den Inhalt der Zelle 007 bis 013 an, das Zählprogramm wird sich jetzt dort befinden. Natürlich sind die Zellen 001 bis 006 unverändert geblieben, aber sie stehen jetzt für andere Dinge zur Verfügung.

1.77 Computer-Geschwindigkeit

Wie schnell ist denn unser Computer nun wirklich? Bei verschiedenen Programmen konnten Sie bereits beobachten, daß der Computer Zeit zum „Nachdenken" braucht (siehe Bemerkung in Kapitel 1.67). Natürlich, die Ausführung eines Befehls dauert eine kurze, aber doch meßbare Zeit (zwischen 0,153 und 1,025 Millisekunden), und wenn der Computer sehr viele Programmschritte durchläuft (bei einer Programmschleife können es Tausende sein), dann ist diese Zeit auch vom menschlichen Auge registrierbar.
Wir haben für Sie ein Zählprogramm entwickelt, das nicht einen einzigen Verzögerungsbefehl enthält und dennoch die einzelnen Zählschritte gut ablesbar anzeigt. Allerdings: wir haben es so gestaltet, daß es im Laufe des Zählvorganges automatisch immer schneller wird und schließlich auf einem Halt-Befehl stehenbleibt.
Das Prinzip ist eigentlich sehr einfach: wir lassen

Listing 26: **Schiebeprogramm**

Adresse	Mnemonics	**Code**	Kommentar
•	•		
•	•		
•	•		
088	LIA 100	**19.100**	Lade in den Akku den Inhalt der Zelle, deren Adresse in Zelle 100 steht
089	AIS 101	**20.101**	Speichere den Akku-Inhalt in der Zelle, deren Adresse in Zelle 101 steht
090	LDA 101	**05.101**	Lade den Inhalt von Zelle 101 in den Akku
091	SUB 103	**08.103**	Subtrahiere „1" vom Akku-Inhalt
092	ABS 101	**06.101**	Speichere den neuen Akku-Inhalt in Zelle 101
093	LDA 100	**05.100**	Lade den Inhalt der Zelle 100 in den Akku
094	SUB 103	**08.103**	Subtrahiere „1" vom Akku-Inhalt
095	ABS 100	**06.100**	Speichere den neuen Akku-Inhalt in Zelle 100
096	VKL 102	**13.102**	Ist der Akku-Inhalt kleiner als die Anfangsadresse des Urprogramms?
097	SPB 099	**11.099**	Wenn ja, springe zum Halt
098	SPU 088	**09.088**	Wenn nein, beginne wieder von vorn
099	HLT	**01.000**	Halte an
100		**00.007**	Endadresse des Urprogramms
101		**00.013**	Endadresse des Zielprogramms
102		**00.001**	Anfangsadresse des Urprogramms
103		**00.001**	Schrittweite

Listing 27: **Automatisch schneller werdender Zähler**

Adresse	Mnemonic	Code	Kommentar
001	AKO 000	**04.000**	Zahlenwerte im Datenbereich speichern
002	ABS 100	**06.100**	
003	AKO 120	**04.120**	
004	ABS 101	**06.101**	
005	AKO 001	**04.001**	
006	ABS 102	**06.102**	
007	AKO 030	**04.030**	
008	ABS 103	**06.103**	
009	ABS 104	**06.104**	
010	LIA 104	**19.104**	Lade in den Akku den Inhalt der Zelle, deren Adresse in Zelle 127 steht
011	LDA 104	**05.104**	Lade in den Akku den Inhalt der Zelle 104 (Anfangsadresse)
012	ADD 102	**07.102**	Addiere „1" dazu . . .
013	ABS 104	**06.104**	. . . und speichere den neuen Akku-Inhalt in Zelle 104
014	VKL 101	**13.101**	Ist Akku-Inhalt noch kleiner als „120"? (Endadresse)
015	SPB 010	**11.010**	Wenn ja, lade weitere Speicherzellen-Inhalte in den Akku
016	LDA 100	**05.100**	Wenn nein, so ist der Durchlauf beendet, daher lade Inhalt von Zählerzelle
017	ADD 102	**07.102**	Addiere „1" dazu . . .
018	ABS 100	**06.100**	. . . und speichere den neuen Akku-Inhalt in Zelle 100
019	ANZ	**02.000**	Zeige ihn an
020	LDA 103	**05.103**	Lade die Anfangsadresse in den Akku
021	ADD 102	**07.102**	Addiere „1" dazu . . .
022	ABS 103	**06.103**	. . . und speichere die neue Anfangsadresse in Zelle 103
023	ABS 104	**06.104**	Speichere auch hier die neue Anfangsadresse
024	VKL 101	**13.101**	Ist Akku-Inhalt noch kleiner als Endadresse?
025	SPB 010	**11.010**	Wenn ja, beginne den nächsten Durchlauf mit der neuen Startadresse
026	HLT	**01.000**	Wenn nein, halte an
100		**00.000**	Speicherzelle für Zählerstand (Anfangswert 00.000)
101		**00.120**	Vergleichszahl für Endadresse
102		**00.001**	Schrittweite
103		**00.030**	Vergleichszahlen für Anfangsadressen
104		**00.030**	

der Reihe nach (mit Hilfe des Indirekten Ladebefehls) die Inhalte der Speicherzellen 030 bis 119 in den Akku laden und sodann einen Zählschritt ausführen und anzeigen. In einem zweiten Durchlauf werden die Inhalte aller Speicherzellen von 031 bis 119 in den Akku geladen, sodann von 032 bis 119, dann von 033 bis 119, bis im letzten Durchlauf schließlich nur noch der Inhalt von 119 geladen wird und der Computer anhält.
Sie können sofort erkennen, daß die Durchlaufzeiten immer kleiner werden, die letzten Zählschritte gehen in einem Flimmern unter.

Anregung (ohne Lösung):
Ändern Sie das Programm so ab, daß der Zähler auch wieder rückwärts zählt, schnell beginnend und langsam endend, dann wieder von vorn beginnt usw.

1.78 Kochrezepte für die eigene Programmierarbeit

Sicherlich werden Sie nun auch eigene Ideen verwirklichen und selbst Programme entwickeln wollen. Deshalb wollen wir Ihnen eine Art „Kochrezept" für die Umsetzung menschlicher Ideen in die formale Sprache unseres Computers an die Hand geben. Wir gehen dabei *vom Menschen* aus und setzen die Gedanken Schritt für Schritt um, bis ein Programm mit den dezimal codierten Befehlen entstanden ist, das wir in den Computer eingeben können. Nur so stellt man auch sicher, daß das Programm das tut, was man mal als Idee und Zielvorstellung hatte!

1. Schritt: Das Flußdiagramm
Der Zweck des Flußdiagramms ist es, sich den logischen Ablauf eines Programmes klarzumachen. Auch hier geht man vom Groben zum Feinen, zuerst stellt man also die Frage, welche *Funktionen* und *logischen Blöcke* eine Idee eigentlich enthält und wie diese miteinander verknüpft sind. Dann geht man daran, die einzelnen Blöcke genauer in ihrem Ablauf zu untersuchen: welche Daten werden benötigt und wie werden sie beeinflußt, wann muß z.B. Port 1 eingelesen werden usw.?
So füllen sich die logischen Blöcke nach und nach mit detaillierten Einzelanweisungen.

2. Schritt: Namensgebung
Im Flußdiagramm wird das Problem noch in Worten geschildert und auch die Operationen mit „Namen" benannt, zum Beispiel: „lade das Zwischenergebnis – addiere den ersten Faktor dazu – speichere das Zwischenergebnis wieder – ist es gleich der Obergrenze?" usw.
Das ist auch sehr vernünftig, weil es menschenorientiert ist. Wir können das Flußdiagramm in klaren Worten lesen und sofort verstehen. Mit dem Begriff „Zähler, Grenze" können wir sofort bestimmte Funktionen verbinden, was bei Adressenangaben wie 100 oder 117 kaum mehr möglich ist. Deshalb wird auch der Text des Flußdiagramms den wichtigsten Teil Ihres späteren Programmkommentars ausmachen. Aber soweit sind wir noch nicht; zunächst müssen Sie die Blöcke und Funktionen in den Mnemonic-Code übersetzen. Dabei halten Sie aber an den Namen für die Datenzellen und Sprungadressen fest, damit auch diese Stufe der Programmentwicklung noch leicht lesbar bleibt.
Bei dieser Umsetzung dürfen Sie mit dem Papier nicht geizen; beginnen Sie mit dem ersten logischen Block irgendwo mitten auf dem Papier. Das sollte dann etwa so aussehen:

```
          ...
weiter:   LDA  Zähler
          ADD  Summand
          ABS  Zähler
          VGL  Grenze
          SPB  heraus
          SPU  weiter
heraus:   LDA  Ergebnis
          ANZ
          ...
```

Hier sind also noch alle Daten und Sprünge mit Namen versehen, und wenn Sie es aufmerksam lesen, können Sie es sofort verstehen. Werden Operationen mit festen Werten durchgeführt (zum Beispiel „ADD Summand"), so können Sie diese auch direkt angeben, also etwa ADD „1". Das ist Geschmackssache.
So entwickeln Sie nach und nach alle Bruchstücke des Flußdiagramms im leicht verständlichen Mnemonic-Code und schreiben sie alle hintereinander auf das Papier.

3. Schritt: Adreßzuweisung
Legen Sie nun eine Liste aller Namen an, die Sie im Programm für Daten verwendet haben; bei unserem Minibeispiel sähe das so aus:

```
Zähler
Summand „1"
Grenze
Ergebnis (am Anfang „0")
```

Wenn es sich dabei um feste Werte handelt, sollten Sie sie jetzt gleich dazuschreiben, so wie wir es bei „Summand" getan haben. Bei anderen Daten ist es oft notwendig, sie am Programmanfang auf einen

bestimmten Wert zu setzen (man nennt das „Vorbelegen"); auch das sollten Sie bei der Datenliste wie gezeigt tun.
Das ist nämlich wichtig für den nächsten Schritt: Sie fügen vor den ersten logischen Programmblock noch die Vorbelegung des Datenbereichs. Etwa so:

```
AKO 001
ABS Summand
AKO 000
ABS Ergebnis
...
```

Dann haben Sie alles, was für das Programm nötig ist und können mit der Durchnumerierung beginnen. Fangen Sie bei der Vorbelegungsroutine mit 001 an und schreiben Sie vor jede Programmzeile die Adresse, wo der Befehl später gespeichert werden wird. Natürlich können Sie zu diesem Zeitpunkt auch ein paar der früher erwähnten Leerbefehle VZG 000 mit einbauen, um später noch Luft für Programmergänzungen zu haben.
Auch den Datenteil numerieren Sie durch; wo Sie damit anfangen, bleibt Ihnen überlassen. Wir haben ja in der Regel bei 100 damit begonnen.

4. Schritt: Ersetzen der Namen
Schreiben Sie nun das Programm nochmal ab und ersetzen Sie dabei die Namen der angesprochenen Datenzellen durch ihre Adressen und lösen Sie die Sprunganweisungen auf, indem Sie die jetzt feststehenden Adressen angeben. Aber ein Punkt ist an dieser Stelle ganz wichtig: der Kommentar! Wenn jetzt nur noch Zahlenkolonnen hinter den Mnemonics auftauchen, wird das Programm zunehmend unverständlicher. Deshalb ist eine ausführliche Kommentierung unerläßlich. Wie erwähnt, können Sie ja dabei auf den Text des Flußdiagramms zurückgreifen. Unser Beispiel sieht dann so aus:

```
001 AKO 001          Lade „1"
002 ABS 101          speichere als Schrittweite
003 AKO 000          Lade „0"
004 ABS 103          speichere als Ergebnis-Startwert
... ...              ...
010 LDA 100          lade den Zähler
011 ADD 101          addiere „1" dazu
012 ABS 100          speichere den Zähler wieder
013 VGL 102          ist er gleich der Grenze?
014 SPB 016          falls ja, springe zur Ergebnisanzeige
015 SPU 010          sonst springe zum Weiterzählen
016 LDA 103          lade das Ergebnis
017 ANZ              und zeige es an
... ...              ...
... ...              ...
100          –       Zähler
101          00.001  „1" als Schrittweite (Summand)
102          –       Grenze des Zählers
103          –       Ergebnis
```

5. Schritt: Der Operationscode
Was nun noch folgt, ist die Übersetzung der Mnemonics in den Operationscode, den der Computer versteht. Zu diesem Zweck haben wir oben im Beispiel zwischen der Mnemonics- und der Kommentarspalte noch viel Platz gelassen, in den wir jetzt den Operationscode noch Zeile für Zeile einfügen.
Damit sind Sie fertig. Das Programm braucht nur noch eingetippt zu werden und funktioniert (hoffentlich)!

1.79 Computersprachen

In unserer Zeit weltweiter Touristik ist schon manch einer auf Verständnislosigkeit gestoßen, weil es mit seinen Sprachkenntnissen haperte.
Auch Computer haben ihre eigene Sprache, die sogenannte Maschinensprache. Und die Computer-Maschinensprachen unterscheiden sich durchaus genauso voneinander wie Französisch und Chinesisch. Pfiffige Computerleute haben daher allgemein verständliche Computersprachen erdacht, die man höhere Programmiersprachen nennt. Die bekanntesten sind **BASIC, FORTRAN** und **COBOL** – um nur einige zu nennen. Die Übersetzungsarbeit in die Maschinensprache überläßt man dabei getrost dem Computer selbst, der Benutzer weiß also gar nicht mehr, in welcher Maschinensprache sein Computer arbeitet. Der Computer benötigt lediglich ein Übersetzungsprogramm, den sog. **Compiler** (engl. to compile = zusammentragen), das einmal für den betreffenden Computertyp erstellt werden muß.
Auch unser Computer hat in seinem ROM (ROM s. Kapitel 1.80) ein Übersetzungsprogramm für die Umwandlung der Ihnen bekannten Befehle mit dezimal codierten Operationscodes, so wie sie für uns Menschen ohne große Mühe benutzbar sind, in die Maschinenbefehle des von uns verwendeten Mikrocomputers 8049.
Dem Ziel, den Computer dem Menschen anzupassen, dient auch unser Mnemonic-Code, der der Gruppe der sog. Assembler-Sprachen zuzuordnen ist. Als Assembler- oder Assembliersprache bezeichnet man eine maschinenorientierte, symbolische Programmiersprache.

1.80 Vom Mikroprozessor zum Computer

Das Herz des KOSMOS-Computers ist ein sogenannter **Mikrocomputer** des Typs 8049. Mikrocomputer sind komplette miniaturisierte Computer auf einem Siliziumplättchen von Daumennagelgröße.

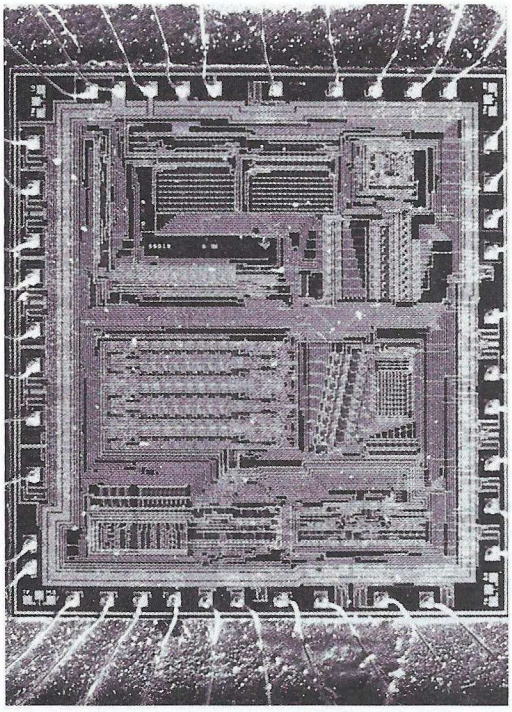


Bild 64a

Ein solches Siliziumplättchen, auf dem nach einem außerordentlich komplizierten technologischen Verfahren zehntausende von Transistorfunktionen integriert sind, heißt **Chip.** Bei dem **Integrierten Schaltkreis** (engl. Integrated Circuit = IC), den Sie in Bild 64b sehen, ist der winzige Chip von einer schwarzen Plastikmasse umpreßt und seine 40 (!) Anschlüsse seitlich in zwei Reihen herausgeführt, damit man ihn bequem in eine gedruckte Schaltung (Leiterplatine) einlöten kann.

Ein Mikrocomputer besteht aus den Funktionseinheiten **Zentralprozessor** (engl. Central Processing Unit = CPU) mit dem Steuerwerk und dem Rechenwerk, sowie dem **Zentralspeicher.** Bausteine, bei denen der Zentralspeicher fehlt, heißen **Mikroprozessoren.**

Es ist Ihnen inzwischen geläufig, daß ein Computer nichts ohne Programm tut. Was Sie allerdings er-

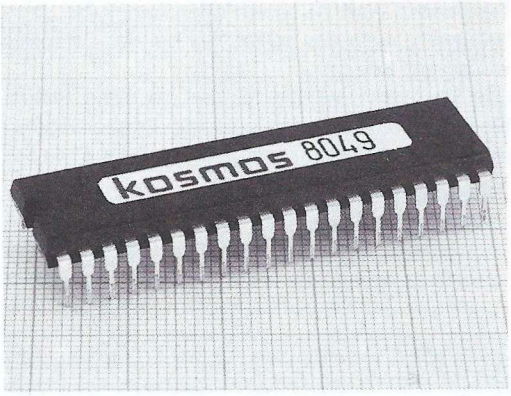
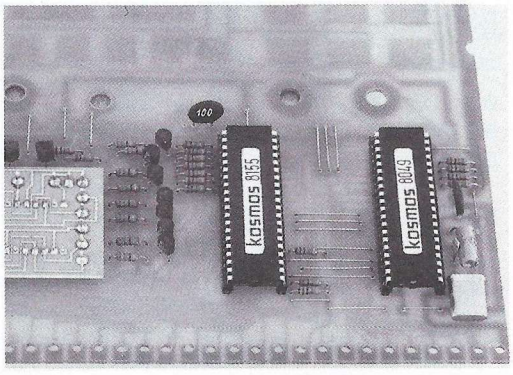


Bild 64b

staunen wird: es würde nicht einmal die Anzeige leuchten, und kein Tastendruck würde ohne Programm funktionieren. Dieses Programm, das sog. Monitor-Programm oder Betriebssystem, haben Sie nicht kennengelernt, weil es fest und unveränderlich in einen speziellen Speicher im Werk hineingebracht wurde, den man **ROM** (engl. Read-Only-Memory = Nur-Lese-Speicher) nennt. In den 2048 Speicherzellen des ROM befinden sich u.a. Programmteile zur Ansteuerung der Anzeige, zur Abfrage der Tastatur, zur Umwandlung von Dual- in Dezimalzahlen, zur Erkennung von Fehlern und zu deren Anzeige, zur Vereinfachung der fast 100 Befehle, die der Prozessor intern ausführt, in die Ihnen bekannten 21 KOSMOS-Befehle usw. ... und natürlich: das Selbstprüfprogramm (9-RUN) und der Reaktionstest (8-RUN) sind ebenfalls im ROM fest gespeichert.

Damit Sie den Computer nach Ihren Wünschen und Vorstellungen programmieren können, gibt es

Bild 64c



noch eine zweite Speicherart, das **RAM** (engl. Random Access Memory = Speicher mit wahlfreiem Zugriff) oder auch Arbeitsspeicher genannt wird. Wie der Name es schon sagt, können Sie mit diesem Speicher arbeiten, d.h. Ihre Daten und Befehle hineinbringen.

Unser Mikrocomputer selbst enthält 128 RAM-Speicherzellen zu je 8 bit, die auch **Register** genannt werden. Diese Register werden allerdings für die interne „Verwaltungsarbeit" des Prozessors benötigt, so daß sie bis auf die Akkumulator-Register für den Benutzer nicht zugänglich sind. Der KOSMOS-Computer enthält daher noch einen zusätzlichen RAM-Baustein – Sie sehen ihn in Bild 64c links neben dem Microcomputer.

An der Rückseite des KOSMOS-Computers befinden sich zwischen den beiden Ports noch 14 Anschlußklemmen, an die die Speichererweiterung (s. S. 69) angeschlossen werden kann. Sie sollten an diese Klemmen unter keinen Umständen irgendetwas anderes anschließen und die Klemmen möglichst auch nicht mit den Fingern berühren.

Die 8 Klemmen rechts neben Port 2 dienen dem Austausch von Befehlen und Daten mit der Speichererweiterung und heißen **Datenbus,** während über die 6 folgenden Klemmen Steuersignale von und zur Speichererweiterung laufen. Diese Klemmen heißen **Steuerbus.**

Ein Bus ist in der Computertechnik eine Sammelleitung, auf der elektrische Signale zwischen verschiedenen Computereinheiten (Speicher, Akkumulator, Rechenwerk, Anzeige) hin- und hertransportiert werden können. Über den Steuerbus werden die Computereinheiten „aufgefordert", Daten aufzunehmen oder abzusenden. Der Vorteil eines solchen Bussystems liegt in der enormen Leitungsersparnis. Verschiedene Computereinheiten sind direkt über ein und dieselben Leitungen (eben über den Bus) untereinander verbunden. Trotzdem beeinflussen oder stören sie sich gegenseitig nicht, da sie erst über den Steuerbus „aktiviert", d.h. zu- oder abgeschaltet werden.

1.81 Wie geht es weiter?

Sie haben Ihren Computer jetzt gründlich kennengelernt, wissen, was die einzelnen Befehle bewirken und haben verschiedene Programme eingegeben und ausprobiert. Der zweite Teil dieses Buches hält noch eine Fülle von weiteren, reizvollen Spiel- und Anwendungsprogrammen bereit. Aber Ihr neues Hobby ist damit keinesfalls beendet. Im Gegenteil: Sie können es durch Erweiterungs- und Zusatzmodule noch ausbauen und vertiefen.

KOSMOS-Kassetten-Interface:
Mit Hilfe des KOSMOS-Kassetten-Interfaces (Best.-Nr. 612211) können Programme, die über die Tastatur in den Speicher eingegeben wurden, auf einer handelsüblichen Tonbandcassette gespeichert und bei Bedarf von der Tonband-Cassette wieder in den Speicher geladen werden. Man erspart sich dadurch das neuerliche Eingeben über die Tastatur, wenn der Computer abgeschaltet war. Die Funktionstasten CAS und CAL sind für diese Zusatzleistung bereits im Computer vorgesehen.

KOSMOS-Speicher-Erweiterung:
Für längere, noch interessantere Programme bietet KOSMOS die Möglichkeit, mit einem Erweiterungsmodul (Best.-Nr. 612111) die Anzahl der Speicherzellen zu verdoppeln und die Anzahl der Ein-Ausgabe-Leitungen auf insgesamt 38 (!) zu erhöhen. Eine Anleitung mit genauer Anschlußbeschreibung und weiteren Programmbeispielen liegt dem Erweiterungsmodul bei.

Weitere Zusätze in Vorbereitung:
Für „lebensechte" Prozeßsteuerungen, wie Modellbahnanlagen und dergleichen, ist ein Relais-Interface mit 8 Treiberstufen, 8 Schwachstromrelais und 8 entstörten Eingängen in Vorbereitung. Ein Ein-Ausgabe-Interface ist ebenfalls geplant.

1.82 Technische Daten

Der KOSMOS-Computer ist ein kleiner Prozeßrechner zum Lernen, Spielen und Experimentieren.

Zentraleinheit:	Ein-Chip 8 bit-Mikrocomputer 8049
Speicher:	Im Chip integriertes 2048 Byte ROM sowie 128 Byte RAM. Zusätzlicher Speicherbaustein 8155 mit 256 Byte RAM und E/A-Erweiterung.
Taktfrequenz:	6 MHz.
Befehlssatz:	21 allgemeingültige, leistungsfähige Befehle (3 zusätzliche Befehle in der Ausbauversion).
Ein/Ausgabe:	8 Eingabe- u. Ausgabe- *oder* 16 Ausgabe-Leitungen (programmabhängig), TTL-kompatibel.
Anzeige:	Sechs 0,5 Zoll 7-Segment-Leuchtziffern.
Tastatur:	Moderne Flachtastatur mit Ziffern- und Funktionstasten.
Stromversorgung:	8 . . . 14 V Wechsel- oder Gleichspannung, ca. 0,5 Ampère. Zur Stromversorgung geeignet ist jeder beliebige Modell- oder Experimentiertransformator oder das KOSMOS-Computernetzteil. Gleichrichtung und elektronische Stabilisierung auf 5 V im Computer integriert.
Integrierte Festprogramme:	Selbstprüfungsprogramm, Reaktionstestspiel, Prüfprogramm für die Speichererweiterung, automatisches Laden von Nullen in den Speicher beim Einschalten, Fehleranzeige-Programm, Speicherinhalt auf Tonbandcassette überspielen und von einer Tonbandcassette in den Speicher laden.
Erweiterungen:	Die Anzahl der Speicherplätze kann durch die KOSMOS-Speichererweiterung (Bestell-Nr. 612111) auf 256 verdoppelt werden. Verschiedene Interface-Module (Kassetten-Interface, Bestell-Nr. 612211) Relais-Interface, Ein-Ausgabe-Interface in Vorbereitung.
Prozeß-Steuerungen:	Der KOSMOS-Computer ist mit allem KOSMOS-Experimentierkästen kombinierbar. Dadurch ergibt sich eine Fülle von reizvollen Steuerungsmöglichkeiten (u. a. auch durch Verwendung des KOSMOS-Schaltrelais KOSMODYNE B, Best.-Nr. 662511)

Zweiter Teil
Programmvorschläge zum Spielen, Knobeln, Steuern, Messen und Lernen

2.0 Vorbemerkung

Nachdem Sie jetzt Ihren Computer kennengelernt haben, können Sie natürlich selbst Programme schreiben. Aber erfahrungsgemäß bereiten komplizierte Probleme dem Anfänger immer noch beträchtliche Schwierigkeiten. Das beginnt mit der Erstellung des logischen Ablaufplans, dem Flußdiagramm und der anschließenden Fehlersuche. Im ersten Teil des Buches waren ja schon einige kleinere Aufgaben enthalten, die zum Selbstprogrammieren anleiten sollten, doch waren sie zum Zwecke der Übersichtlichkeit immer recht kurz gehalten. In diesem Abschnitt wollen wir nun eine Reihe zum Teil sehr umfangreicher Programme vorstellen, die zwei Bedürfnisse erfüllen sollen: Zum einen sind sie einfach dazu da, um mit dem Computer spielen zu können, ihn also zu benutzen. Das heißt, daß Sie nicht unbedingt das Programm durchschauen müssen, um es zu verwenden. Dabei handelt es sich vorwiegend um Computerspiele (vom Nim-Spiel bis zur Mondlandung). Zum anderen soll aber auch höheren Ansprüchen genügt werden, die von den angehenden „Profis" an den Computer gestellt werden. Zu diesem Zweck haben wir die Programme stets ausführlich kommentiert und beschrieben. Auch die Elektronik-Spezialisten werden dabei nicht zu kurz kommen, da wir typische Anschlußbeispiele mit Schaltung und Aufbaubild für KOSMOS-Elektronikmaterial beigefügt haben.

Das Prinzip, den Befehlsbereich in der Regel bei 001 und den Datenbereich bei 100 anfangen zu lassen, haben wir zumeist beibehalten. Mitunter war es aber gar nicht möglich, die Trennung exakt bei 100 zu machen, so daß die Daten manchmal auch woanders gespeichert werden. Sie haben jedoch im ersten Teil des Buches gelernt, daß es im Prinzip egal ist, wo was gespeichert wird, und wenn man das im Hinterkopf behält, fällt das Verstehen der Programme sicher nicht schwer. Noch etwas zu den Datenbereichen: Sie sind stets am Ende des Programmlistings angegeben; dazu wird dann jede benötigte Datenzelle mit Adresse, Inhalt und Name aufgeführt.

Da sich einige Inhalte während des Programmablaufs ändern – zum Teil werden sie nur als Zwischenspeicher (Hilfszelle) verwendet – sind nicht bei allen Speicherzellen bestimmte Werte als Inhalt angegeben. In der Liste steht dann dort ein „–". Andere Zellen *müssen* dagegen vor dem Programmstart mit Werten gefüllt werden. Sie sind mit einem „*" hinter der Adresse versehen. Manchmal handelt es sich dabei um feste Werte (wie 00.000), die das Programm selbst nicht mehr ändert, zum Teil bleibt es aber auch Ihnen überlassen, was Sie eingeben wollen; dann steht als Inhaltsangabe ein „xxx". Ein Beispiel dazu:

100*	00.000	Vergleichszahl „0" (muß eingegeben werden)
101*	00.001	Schrittweite „1" (muß eingegeben werden)
102*	00.xxx	Startwert für Zufallszahlerzeugung (muß eingegeben werden)
103*	00.xxx	Multiplikationsfaktor (muß eingegeben werden)
104	–	Zähler (vom Programm belegt)
105	–	Hilfszelle (vom Programm belegt)
106	–	Ergebniszelle (vom Programm belegt)

In diesem Beispiel müssen also in die Zellen 100 bis 103 vor dem Programmstart vom Benutzer Zahlenwerte eingegeben werden, die Zellen 104 bis 106 dagegen füllt das Programm selbst. Wenn das Programm aber am Anfang die Zellen 100 und 101 mit den festen Werten „0", „1" u.a. belegt, wäre dort der „*" in der Liste weggelassen worden.

Die Programmlistings sind so kommentiert, daß ein Zugriff auf eine Datenzelle immer mit ihrem Namen erfolgt („05.104 Lade den Zähler"); außerdem sind Sprünge in der Mnemonicspalte zusätzlich mit einem **Sprungnamen** versehen.

Beispiel:

Adresse		Mnemonics	**Code**	Kommentar
...		...	...	...
040	A2:	LDA 104	**05.104**	Zählerschleife – Lade den Zähler
...		...	...	...
055		SPU 040 (A2)	**09.040**	Springe in die Zählerschleife
...		...	...	...

Das erleichtert das Lesen des Programms und das Verständnis einzelner logischer Blöcke sehr. Die Sprungnamen bestehen dabei immer aus einer Buchstabe-Zahl-Kombination, also zum Beispiel „A2", „Z1" usw. Manchmal taucht auch eine Kombination der Art „RX" auf; dies ist dann die Bezeichnung für einen Indirekten Sprung, dessen Zieladresse ja nicht direkt beim Befehl, sondern in einer Datenzelle steht. Das Füllen der Sprungadreßzelle wurde in der Regel mit einem AKO- und ABS-Befehl gemacht, wo dann die entsprechende Kombination, beispielsweise „R3", dahinter angegeben ist.

Noch etwas zu den Spielprogrammen: Einige von ihnen benötigen Zufallszahlen, um den Benutzer zu überraschen. Nun ist ja der Computer an sich ein völlig logischer und schlüssiger Zeitgenosse und überhaupt nicht in der Lage, etwas „aus Zufall" zu machen (sonst wäre er ja kreativ). Deshalb müssen wir einen Trick anwenden, um „Pseudozufallszahlen" zu erzeugen. Wir erinnern uns an die Darstellung der Zahlen im Dualsystem, wie wir es im ersten Teil des Buches bei der Port-Ein- und -Ausgabe kennengelernt haben. Jede Dezimalzahl hat also eine eindeutige Zuordnung zu einer achtstelligen Dualzahl, wie sie in der Tabelle Seite 50 nachgesehen werden kann. Nehmen wir nun zwei der acht Bits und vertauschen sie, erhalten wir unter Umständen eine ganz andere Zahl. Und wenn wir gar alle acht miteinander vermischen und neu sortieren, kommt in den allermeisten Fällen eine neue Kombination und damit auch eine neue Dezimalzahl heraus. So weit, so gut. Wie aber können wir die computerinterne Dualdarstellung mischen? Ganz einfach: wir geben die Dezimalzahl als Dualzahl an Port 2 aus (P2A 000), schließen dort acht Drähte an und führen diese dann zu Port 1, wobei wir einfach die Reihenfolge vertauschen. Die so „vermischten" Bits lassen wir von Port 1 in den Akku übernehmen (P1E 000) und haben eine neue Zahl!

Nun sind nicht alle Vertauschungen gleich gut, um viele verschiedene Zahlen zu erzeugen. Eine der besten ist die folgende, und sie wird bei zahlreichen Programmen dann Verwendung finden:

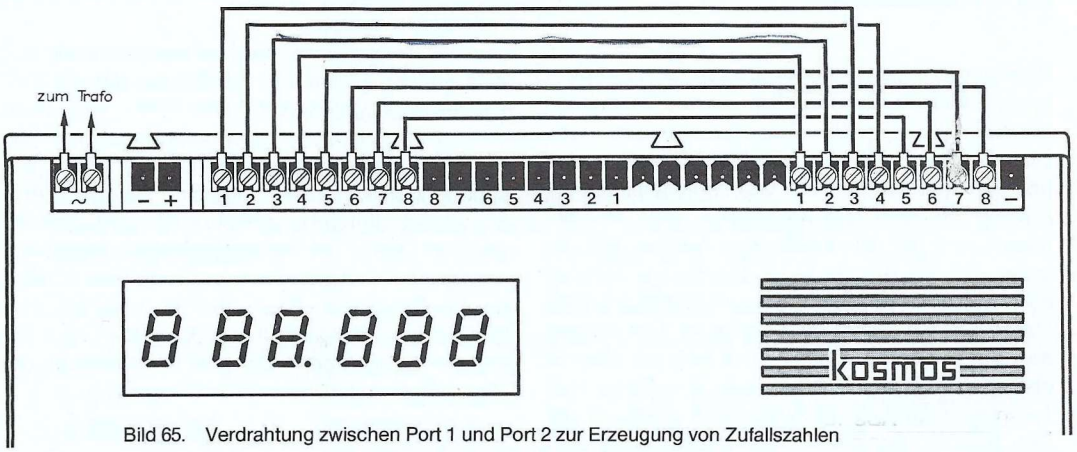


Bild 65. Verdrahtung zwischen Port 1 und Port 2 zur Erzeugung von Zufallszahlen

Einige Programme sind ziemlich kurz und deshalb schnell zu durchschauen, andere dagegen sehr komplex und benötigen einige Geduld; aber wenn man sie verstanden hat, sind sicher zusätzlich ein paar gute Ideen für eigene Programme herausgekommen.

2.1 Uhr mit wechselnder Stunden-/Minutenanzeige

Um es gleich vorweg zu sagen: kritische Geister könnten bei dieser Uhr die Art der Anzeige bemängeln. Sie wechselt im Sekundenrhythmus zwischen der Stunden- und der Minutenanzeige. Das Programm schiebt die Stundenanzeige *vor* den Punkt, also dorthin, wo normalerweise der Operationscode steht. Wir machen uns dabei die Tatsache zunutze, daß wir auch *Befehle* in den Akku laden können (siehe Kapitel 1.38, letzter Abschnitt). Im Programm ist deshalb eine Vergleichstabelle eingebaut, die dafür sorgt, daß immer der „richtige" Befehl, den wir als Stundenanzeige mißbrauchen, geladen wird. Und wenn wir dann z.B. die Anzeigenfolge 00.012–07.000 usw. sehen, ist die Uhrzeit klar: 7.12 Uhr!

Vor dem Programmstart muß in Zelle 003 die aktuelle (also Start-) Minute und in Zelle 005 die aktuelle Stunde als Operand der AKO-Befehle eingegeben werden. Das Programm und die Daten müssen bis einschließlich Zelle 116 eingegeben werden.

Wem das Programm, so wie es geschrieben wurde, zu lang ist, kann ja seinen Grips mal anstrengen: die lange Latte von Vergleichs- und Sprungbefehlen zum Stundenladen könnte man nämlich sehr elegant mit den Indirekten Befehlen LIA und AIS kürzen; so etwa mit folgender Überlegung: momentane Stunde als Adresse für die Anzeigetabelle speichern und damit indirekt den richtigen „Befehl" laden. Nur Mut, es ist gar nicht so schwer!

Listing 28: **Uhr mit wechselnder Stunden- und Minutenanzeige**

Adresse		Mnemonics	**Code**	Kommentar
000		AKO 000	**04.000**	lade „0"
001		ABS 118	**06.118**	speichere als Viertelsekundenzähler . . .
002		ABS 121	**06.121**	und als Sekundentakt
003		AKO XXX	**04.XXX**	lade die aktuelle Minute (Uhr!)
004		ABS 119	**06.119**	speichere in 119
005		AKO YYY	**04.YYY**	lade die aktuelle Stunde (Uhr!)
006		ABS 120	**06.120**	speichere in 120
007		SPU 044 (B1)	**09.044**	das Programm fängt so richtig erst bei 044 an
008	A1:	AKO 001	**04.001**	lade „1"
009		ADD 121	**07.121**	addiere den Sekundentakt zu dieser „1". . .
010		ABS 121	**06.121**	und speichere den Sekundentakt wieder
011		VGR 092	**12.092**	ist er größer als „4", also eine Sekunde vorbei?
012		SPB 015 (A2)	**11.015**	falls ja, springe zu 015 (Stunde laden)
013		LDA 119	**05.119**	sonst lade die Minute in den Akku . . .
014		SPU 020 (A4)	**09.020**	und springe zum Anzeigen
015	A2:	VKL 096	**13.096**	ist der Takt noch kleiner als „8"?
016		SPB 019 (A3)	**11.019**	falls ja, springe direkt zum Stunde-Laden
017		AKO 000	**04.000**	sonst lade „0". . .
018		ABS 121	**06.121**	und speichere als Sekundentakt: Neubeginn
019	A3:	LDA 117	**05.117**	lade den Inhalt der Anzeigezelle in den Akku . . .
020	A4:	ANZ	**02.000**	und zeige den Wert an (Minute und Stunde im Sekundentakt)
021	A5:	VZG 217	**03.217**	warte kurz
022		AKO 001	**04.001**	lade „1"
023		ADD 118	**07.118**	addiere den Viertelsekundenzähler dazu . . .
024		ABS 118	**06.118**	und speichere den erhöhten Wert
025		VKL 116	**13.116**	ist er kleiner als „240"?
026		SPB 008 (A1)	**11.008**	dann ist die Minute noch nicht rum – weiterzählen
027		AKO 000	**04.000**	sonst lade „0"
028		ABS 118	**06.118**	speichere als Viertelsekundenzähler: Neubeginn
029		AKO 001	**04.001**	lade „1". . .
030		ADD 119	**07.119**	und addiere dazu den bisherigen Minutenwert
031		ABS 119	**06.119**	speichere die neue Minute wieder
032		VZG 000	**03.000**	warte – (Zeitkorrektur möglich)
033		VKL 115	**13.115**	ist die Minute kleiner als „60"?
034		SPB 021 (A5)	**11.021**	falls ja, weiter bei 021 (neuer Start der Viertelsekunden)
035		AKO 000	**04.000**	sonst lade „0"
036		ABS 119	**06.119**	speichere als Minutenwert: eine Stunde ist rum
037		AKO 001	**04.001**	deshalb lade „1". . .
038		ADD 120	**07.120**	und addiere dazu den bisherigen Stundenwert
039		ABS 120	**06.120**	speichere die Stunde wieder
040		VKL 114	**13.114**	ist die Stunde „13" noch nicht erreicht?
041		SPB 044 (B1)	**11.044**	falls ja, springe zum Anzeigevorbereiten
042		AKO 001	**04.001**	sonst lade „1"

Fortsetzung siehe nächste Seite

Fortsetzung Listing 28:

Adresse		Mnemonics	Code	Kommentar
043		ABS 120	**06.120**	speichere als Stundenwert
044	B1:	VKL 113	**13.113**	Ist die Stunde (immer noch im Akku!) noch kleiner als „2“?
045		SPB 069 (C1)	**11.069**	falls ja, springe mal zu 069: Stundenaufbereitung
046		VKL 091	**13.091**	ist sie kleiner als „3“?
047		SPB 071 (C2)	**11.071**	falls ja, zu 071 springen
048		VKL 092	**13.092**	kleiner als „4“?
049		SPB 073 (C3)	**11.073**	dann weiter bei 073
050		VKL 093	**13.093**	kleiner als „5“?
051		SPB 075 (C4)	**11.075**	dann zu 075
052		VKL 094	**13.094**	kleiner als „6“?
053		SPB 077 (C5)	**11.077**	dann zu 077
054		VKL 095	**13.095**	kleiner als „7“?
055		SPB 079 (C6)	**11.079**	dann zu 079
056		VKL 096	**13.096**	kleiner als „8“?
057		SPB 081 (C7)	**11.081**	dann zu 081
058		VKL 097	**13.097**	kleiner als „9“?
059		SPB 083 (C8)	**11.083**	dann zu 083
060		VKL 098	**13.098**	kleiner als „10“?
061		SPB 085 (C9)	**11.085**	dann zu 085
062		VKL 099	**13.099**	kleiner als „11“?
063		SPB 087 (D1)	**11.087**	dann zu 087
064		VKL 100	**13.100**	kleiner als „12“?
065		SPB 089 (D2)	**11.089**	dann geht es bei 089 weiter
066		LDA 112	**05.112**	sonst ist „12“ drin – lade deshalb Zelle 112 (12.000) . . .
067	B2:	ABS 117	**06.117**	und speichere in 117 (Anzeigezelle)
068		SPU 008 (A1)	**09.008**	Dann fange wieder bei 008 an
069	C1:	LDA 101	**05.101**	lade Zelle 101 (1. Stunde, also 01.000) in den Akku
070		SPU 067 (B2)	**09.067**	springe zum Umspeichern in die Anzeigezelle bei 067
071	C2:	LDA 102	**05.102**	lade Zelle 102 (2. Stunde, also 02.000) in den Akku
072		SPU 067 (B2)	**09.067**	springe zum Umspeichern in die Anzeigezelle bei 067
073	C3:	LDA 103	**05.103**	lade Zelle 103 (3. Stunde, also 03.000) in den Akku
074		SPU 067 (B2)	**09.067**	springe zum Umspeichern in die Anzeigezelle bei 067
075	C4:	LDA 104	**05.104**	lade Zelle 104 (4. Stunde, also 04.000) in den Akku
076		SPU 067 (B2)	**09.067**	springe zum Umspeichern in die Anzeigezelle bei 067
077	C5:	LDA 105	**05.105**	lade Zelle 105 (5. Stunde, also 05.000) in den Akku
078		SPU 067 (B2)	**09.067**	springe zum Umspeichern in die Anzeigezelle bei 067
079	C6:	LDA 106	**05.106**	lade Zelle 106 (6. Stunde, also 06.000) in den Akku
080		SPU 067 (B2)	**09.067**	springe zum Umspeichern in die Anzeigezelle bei 067
081	C7:	LDA 107	**05.107**	lade Zelle 107 (7. Stunde, also 07.000) in den Akku
082		SPU 067 (B2)	**09.067**	springe zum Umspeichern in die Anzeigezelle bei 067
083	C8:	LDA 108	**05.108**	lade Zelle 108 (8. Stunde, also 08.000) in den Akku
084		SPU 067 (B2)	**09.067**	springe zum Umspeichern in die Anzeigezelle bei 067
085	C9:	LDA 109	**05.109**	lade Zelle 109 (9. Stunde, also 09.000) in den Akku
086		SPU 067 (B2)	**09.067**	springe zum Umspeichern in die Anzeigezelle bei 067
087	D1:	LDA 110	**05.110**	lade Zelle 110 (10. Stunde, also 10.000) in den Akku
088		SPU 067 (B2)	**09.067**	springe zum Umspeichern in die Anzeigezelle bei 067
089	D2:	LDA 111	**05.111**	lade Zelle 111 (11. Stunde, also 11.000) in den Akku
090		SPU 067 (B2)	**09.067**	springe zum Umspeichern in die Anzeigezelle bei 067
091			**00.003**	Vergleichszahlen für den Stundenwert: Stunde 3
092			**00.004**	Stunde 4

Fortsetzung siehe nächste Seite

Fortsetzung Listing 28:

Adresse	Mnemonics	Code	Kommentar
093		**00.005**	Stunde 5
094		**00.006**	Stunde 6
095		**00.007**	Stunde 7
096		**00.008**	Stunde 8
097		**00.009**	Stunde 9
098		**00.010**	Stunde 10
099		**00.011**	Stunde 11
100		**00.012**	Stunde 12
101		**01.000**	Anzeigezahlen für den Stundenwert: Stunde 1
102		**02.000**	Stunde 2
103		**03.000**	Stunde 3
104		**04.000**	Stunde 4
105		**05.000**	Stunde 5
106		**06.000**	Stunde 6
107		**07.000**	Stunde 7
108		**08.000**	Stunde 8
109		**09.000**	Stunde 9
110		**10.000**	Stunde 10
111		**11.000**	Stunde 11
112		**12.000**	Stunde 12
113	*	**00.002**	noch ein Stundenvergleich: „2“
114	*	**00.013**	„13“ als Obergrenze der Stundenanzeige
115	*	**00.060**	„60“ als Obergrenze für Minutenzähler
116	*	**00.240**	„240“ als Grenze für Viertelsekundenzähler
117		–	Anzeigezelle (Stundenwert als „Befehl“)
118		–	Viertelsekundenzähler
119		–	momentane Minute
120		–	momentane Stunde
121		–	Sekundentakt zum Anzeigewechsel

2.2 Reaktionstester mit variabler Vorlaufzeit

Das folgende Programm ist ähnlich aufgebaut wie der im ROM gespeicherte, durch 8-RUN abrufbare Reaktionstester. Die variable Vorlaufzeit wird dadurch erzeugt, daß jeweils das vorangegangene Testergebnis durch zehn dividiert und der Rest mit einer Verzögerungszeit von 200 ms multipliziert wird. Also: war z. B. die letzte Reaktionszeit 158, so wird 8 × 200 ms gewartet, bei 121 nur 1 × 200 ms und bei 110 gar nicht (Rest ist 0).
Nach der Verzögerungszeit leuchtet die Anzeige auf, und der Zählvorgang wird angezeigt, bis die STP-Taste gedrückt wird. Bei einem erneuten Start durch 001-PC-RUN wird das in Zelle 000 gespeicherte Testergebnis zur Berechnung der neuen Verzögerungszeit benutzt.
Statt des Zehnerwerts kann natürlich (durch Ändern der Zelle 005) auch jeder andere Rest genommen werden. Die 200 ms-Verzögerung in Zelle 014 kann ebenfalls leicht geändert werden.

2.3 Telefonzeittakt-Gebührenanzeiger

Seit einiger Zeit hat die Deutsche Bundespost einen Zeittakt für Orts- und Nahgespräche eingeführt. Für eine Gebühreneinheit (zur Zeit 23 Pfennig) kann man tagsüber 8, abends und am Wochenende sogar 12 Minuten telefonieren. Um zu wissen, wieviel Geld man schon wieder „verquasselt" hat, kann man ein Computerprogramm verwenden: Man startet es bei Gesprächsbeginn, der Computer zählt die anfallenden Gebühreneinheiten und zeigt sie an.
Dabei enthält das Programm sogar schon ein paar Feinheiten, die man bei seiner Kürze gar nicht erwartet: zum einen kann es ohne weiteres vom 8-Minuten-Takt auf den 12-Minuten-Takt umgestellt werden (Zelle 008), des weiteren geht die Berechnung genau wie bei der Post auch: jeder angebrochene Takt muß bezahlt werden! Die Anzeige beginnt also sofort nach dem Start des Programms bei **001** mit „A 00.023" und erhöht sich bei Beginn der neunten Minute auf „A 00.046" usw.

Listing 29: **Reaktionstester mit variabler Vorlaufzeit**

Adresse		Mnemonics	**Code**	Kommentar
001		AKO 001	**04.001**	lade „1" in den Akku . . .
002		ABS 102	**06.102**	und speichere sie in 102 (Zählerschrittweite)
003		AKO 000	**04.000**	lade „0" in den Akku . . .
004		ABS 101	**06.101**	und speichere sie in 101 (Vergleichszahl)
005		AKO 010	**04.010**	lade „10". . .
006		ABS 100	**06.100**	und speichere sie in 100 (zum Zehnerabtrennen)
007		LDA 000	**05.000**	lade Inhalt von 000: letzter Zählerstand
008	A0:	VGR 100	**12.100**	ist er größer als „10"?
009		SPB 011 (A1)	**11.011**	falls ja, Sprung zu 011 (Abtrennen des Zehnerteils)
010		SPU 013 (A2)	**09.013**	sonst Sprung zu 013
011	A1:	SUB 100	**08.100**	subtrahiere „10" vom Akkuinhalt . . .
012		SPU 008 (A0)	**09.008**	und mache weiter beim Vergleich
013	A2:	VGL 101	**10.101**	Akku ist kleiner als „10"! Ist er schon „0"?
014		VZG 200	**03.200**	warte ein bißchen
015		SPB 018 (A3)	**11.018**	falls ja, springe zu 018 (Ende der Verzögerung)
016		SUB 102	**08.102**	sonst subtrahiere „1" vom Akku . . .
017		SPU 013 (A2)	**09.013**	und springe zurück zum Vergleich mit „0"
018	A3:	AKO 000	**04.000**	variable Verzögerung zuende – „0" als Zeitanfang der Reaktionszeit laden
019		ANZ	**02.000**	Anzeige des Akkuinhalts
020	A4:	ADD 102	**07.102**	addiere „1". . .
021		ANZ	**02.000**	und zeige erneut an
022		ABS 000	**06.000**	speichere den Zählerstand in Zelle 000
023		VZG 005	**03.005**	warte kurz . . .
024		SPU 020 (A4)	**09.020**	und springe dann zum Weiterzählen
100			**00.010**	„10" zum Zehnerabtrennen
101			**00.000**	Vergleichszahl „0"
102			**00.001**	Zählerschrittweite „1"
000			**–**	Ergebniszelle (Zeitzähler)

Listing 30: **Telefonzeittakt-Gebührenanzeiger**

Adresse		Mnemonics	Code	Kommentar
001		AKO 000	**04.000**	lade „0"
002		ABS 100	**06.100**	speichere als Vergleichszahl
003		AKO 001	**04.001**	lade „1"
004		ABS 101	**06.101**	speichere als Schrittweite
005		AKO 023	**04.023**	lade „23"
006		ABS 102	**06.102**	speichere als Gebühreneinheit. . .
007		ABS 105	**06.105**	und als Anfangswert des Endpreises
008	A1:	AKO 008	**04.008**	lade „8"
009		ABS 103	**06.103**	speichere als Taktlänge (8-Minuten-Takt)
010	A2:	AKO 240	**04.240**	lade „240"
011		ABS 104	**06.104**	speichere als Viertelsekundenzähler
012		LDA 105	**05.105**	lade den Endpreis
013		ANZ	**02.000**	zeige ihn an . . .
014		ABS 000	**06.000**	und speichere ihn in 000 zum Anzeigen
015	A3:	LDA 104	**05.104**	lade den Viertelsekundenzähler
016		VZG 219	**03.219**	warte 219 ms (korrigierte Viertelsekunde)
017		SUB 101	**08.101**	verringere den Zähler um „1"
018		ABS 104	**06.104**	speichere ihn wieder
019		VGL 100	**10.100**	ist er schon bei „0", also Minute vorbei?
020		SPB 022 (A4)	**11.022**	falls ja, springe zur Minutenzählung
021		SPU 015 (A3)	**09.015**	sonst springe zurück (während der Minute)
022	A4:	LDA 103	**05.103**	lade den Minutenstand des Taktes
023		SUB 101	**08.101**	verringere ihn um „1". . .
024		ABS 103	**06.103**	und speichere ihn wieder
025		VGL 100	**10.100**	ist die Minute bei „0", also Taktende angelangt?
026		SPB 028 (A5)	**11.028**	falls ja, springe zum neuen Taktbeginn
027		SPU 010 (A2)	**09.010**	sonst springe zurück (während des Taktes)
028	A5:	LDA 105	**05.105**	ein neuer Takt beginnt! Lade den Gesamtpreis
029		ADD 102	**07.102**	erhöhe ihn um die Gebühreneinheit
030		ABS 105	**06.105**	und speichere ihn wieder
031		SPU 008 (A1)	**09.008**	springe zurück zur Taktmessung
000			–	Endpreis für die Anzeige nach STP
100			**00.000**	„0" als Vergleichszahl
101			**00.001**	„1" als Schrittweite
102			**00.023**	„23" als Gebühreneinheit
103			**00.008**	„8" als Taktlänge in Minuten
104			–	Viertelsekundenzähler (240 pro Minute)
105			–	Endpreis bzw. Zwischenstand

Beim Gesprächsende wird die STP-Taste gedrückt; dann wird ja bekanntlich der Inhalt von Zelle 000 angezeigt. Da aber gerade in dieser Zelle die Gesprächskosten gespeichert werden, sieht man den Endpreis unmittelbar.

2.4 Digitalvoltmeter

Mit wenig zusätzlichem Elektronikaufwand kann unser Computer auch als Meßgerät eingesetzt werden. Was kann gemessen werden? Nun, alle physikalischen Größen, die in eine ihrem Wert entsprechende Frequenz umgesetzt und in dieser Form dem Computer eingegeben werden. Der Computer kann Frequenzen bis etwa 100 Hz messen.

Den Besitzern des KOSMOS Elektronik-Labors ist die Schaltung des Spannungs-Frequenz-Wandlers in Bild 67 wohlbekannt, es werden für die Anpassung an den Computer lediglich einige Bauteile ausgetauscht.

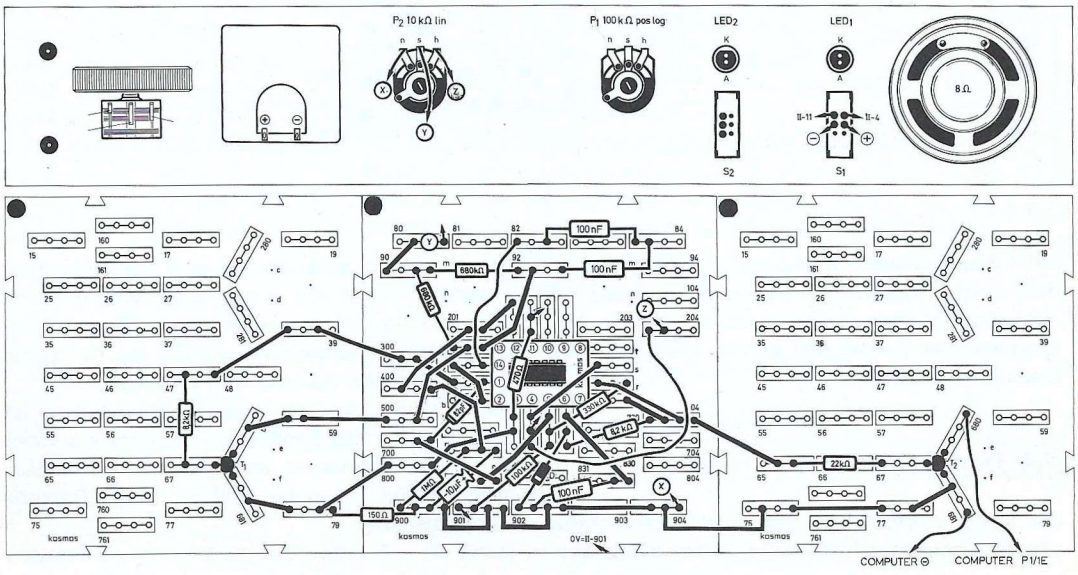
Damit die Computer-Anzeige mit dem zu messen-

Listing 31: **Digitalvoltmeter**

Adresse	Mnemonics	Code	Kommentar
001	AKO 189	**04.189**	lade „189" (oder einen anderen passenden Wert) . . .
002	ABS 100	**06.100**	als Zeitgrenze der Schleife speichern
003	A1: AKO 000	**04.000**	lade „0". . .
004	ABS 101	**06.101**	als Start-Zustand der Leitung,
005	ABS 102	**06.102**	als Flankenzähleranfang . . .
006	ABS 103	**06.103**	und als Zeitnullpunkt speichern
007	A2: P1E 001	**16.001**	lies Port 1 Klemme 1: Pulseingabe
008	VGL 101	**10.101**	Zustand noch wie bei der letzten Eingabe?
009	ABS 101	**06.101**	speichere den jetzigen Zustand ab
010	SPB 014 (A3)	**11.014**	falls ja, ist keine Flanke aufgetaucht: Nur Zeit zählen
011	AKO 001	**04.001**	sonst lade „1". . .
012	ADD 102	**07.102**	und erhöhe so den Flankenzähler
013	ABS 102	**06.102**	speichere ihn wieder
014	A3: AKO 001	**04.001**	Zeitmessung: lade „1"
015	ADD 103	**07.103**	erhöhe so die Zeit-Zelle . . .
016	ABS 103	**06.103**	und speichere sie wieder
017	VKL 100	**13.100**	ist die Zeit noch kleiner als die gesetzte Grenze?
018	SPB 007 (A2)	**11.007**	falls ja, weiter einlesen und zählen
019	LDA 102	**05.102**	sonst lade den Flankenzähler
020	ANZ	**02.000**	zeige ihn an . . .
021	SPU 003 (A1)	**09.003**	und starte einen neuen Zyklus
100		–	Zeitschleifenlänge (Zeitbasis)
101		–	letzter Zustand von Port 1 Klemme 1
102		–	Flankenzähler (Frequenzmessung)
103		–	Zeitzähler

den Spannungswert übereinstimmt, kann die Konstante im AKO-Befehl auf Adresse 001 und damit die Dauer der Abfrageschleife verändert werden. Die Anzeige „A 00.152" entspricht dann beispielsweise einem Meßwert von 1,52 V.

Bild 66. Digitalvoltmeter, aufgebaut aus KOSMOS „Elektronik-Labor E 200" (Schaltplan dazu Bild 67)



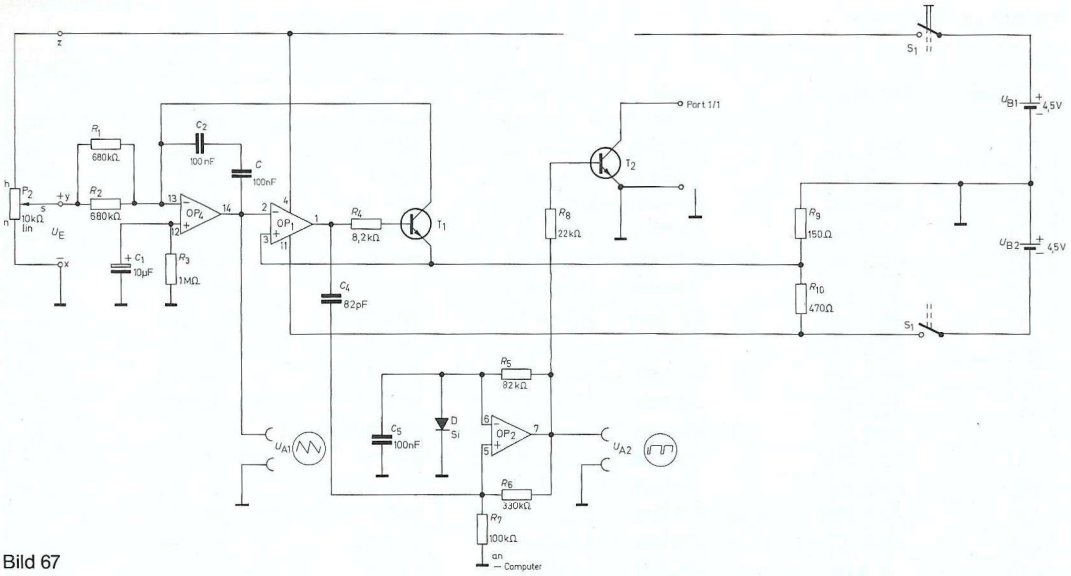
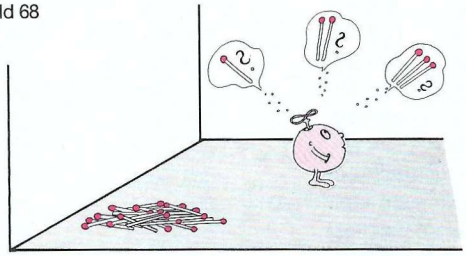


Bild 67

2.5 Nim-Spiel

Bild 68



Es handelt sich um ein uraltes Spiel, bei dem es darum geht, aus einem Haufen von 15 Streichhölzern abwechselnd 1, 2 oder 3 wegzunehmen. Es gewinnt, wer das letzte Hölzchen nicht nehmen muß (oder umgekehrt: wer das letzte nimmt, verliert!).

Wir spielen hier gegen den Computer und beginnen mit dem „Wegnehmen". Als nächstes macht der Computer einen Zug, dann wieder wir usw. Die 15 Hölzchen sind natürlich nur eine Zahl im Speicher, und das Wegnehmen ist nichts anderes als eine entsprechende Subtraktion. Wie nehmen *wir* weg? Ganz einfach, bei Port 1 müssen an den Klemmen 1, 2 und 3 Taster angeschlossen werden (siehe Kapitel 1.53). Wird Taste 1 gedrückt, signalisieren wir, daß wir ein Hölzchen wegnehmen wollen. Bei Taster 2 dann zwei und bei Taster 3 drei Hölzchen. Haben wir „gezogen", ist der Computer an der Reihe: Auf der Anzeige erscheint kurz, was er nimmt, dann die Anzahl der restlichen Hölzchen. Ist schließlich das Ende erreicht, zeigt er entweder „A 00000" (Spieler hat gewonnen) oder „A 11111" (Computer hat gewonnen).

In der Eingabeschleife, wo der Computer auf unseren Zug wartet, ist übrigens eine Verzögerung eingebaut, die den Kontakt „entprellt", das heißt, es macht nichts, wenn wir versehentlich hintereinander mehrmals den Kontakt berühren. Erst nach etwa einer halben Sekunde ist der Port wieder „scharf".

Listing 32: **Nim-Spiel**

Adresse	Mnemonics	**Code**	Kommentar
001	AKO 001	**04.001**	Vorbelegen des Datenbereichs
002	ABS 111	**06.111**	
003	AKO 002	**04.002**	
004	ABS 112	**06.112**	
005	AKO 003	**04.003**	
006	ABS 113	**06.113**	
007	AKO 015	**04.015**	

Fortsetzung siehe nächste Seite

Fortsetzung Listing 32:

Adresse	Mnemonics	Code	Kommentar
008	ABS 100	**06.100**	Vorbelegen des Datenbereichs
009	AKO 014	**04.014**	
010	ABS 101	**06.101**	
011	AKO 013	**04.013**	
012	ABS 102	**06.102**	
013	AKO 012	**04.012**	
014	ABS 103	**06.103**	
015	AKO 011	**04.011**	
016	ABS 104	**06.104**	
017	AKO 010	**04.010**	
018	ABS 105	**06.105**	
019	AKO 009	**04.009**	
020	ABS 106	**06.106**	
021	AKO 008	**04.008**	
022	ABS 107	**06.107**	
023	AKO 007	**04.007**	
024	ABS 108	**06.108**	
025	AKO 006	**04.006**	
026	ABS 109	**06.109**	
027	AKO 005	**04.005**	
028	ABS 110	**06.110**	
029	AKO 000	**04.000**	
030	ABS 117	**06.117**	
031	LDA 100	**05.100**	lade den Hölzchenrest („15“)
032	A0: ANZ	**02.000**	zeige ihn an
033	VZG 255	**03.255**	warte eine Viertelsekunde
034	A1: P1E 001	**16.001**	lies Port 1 Klemme 1 in den Akku
035	VGL 117	**10.117**	ist sie „0“, also betätigt?
036	SPB 044 (A2)	**11.044**	falls ja, springe zu 044
037	P1E 002	**16.002**	sonst lies Port 1 Klemme 2
038	VGL 117	**10.117**	ist diese „0“, also betätigt?
039	SPB 046 (A3)	**11.046**	falls ja, springe zu 046
040	P1E 003	**16.003**	sonst lies Port 1 Klemme 3
041	VGL 117	**10.117**	ist diese „0“, also betätigt?
042	SPB 048 (A4)	**11.048**	falls ja, springe zu 048
043	SPU 034 (A1)	**09.034**	sonst ist gar keine betätigt – springe zu 034
044	A2: AKO 001	**04.001**	lade „1“ (ein Hölzchen genommen)
045	SPU 049 (A5)	**09.049**	springe zu 049
046	A3: AKO 002	**04.002**	lade „2“ (zwei Hölzchen genommen)
047	SPU 049 (A5)	**09.049**	springe zu 049
048	A4: AKO 003	**04.003**	lade „3“ (drei Hölzchen genommen)
049	A5: ABS 114	**06.114**	speichere die Wegnahme des Spielers in 114
050	VZG 255	**03.255**	warte kurz
051	LDA 100	**05.100**	lade den alten Hölzchenrest aus Zelle 100 . . .
052	SUB 114	**08.114**	und subtrahiere, was der Spieler wegnimmt
053	ABS 100	**06.100**	speichere den neuen Hölzchenrest wieder
054	VGL 101	**10.101**	ist der Hölzchenrest jetzt „14“?
055	SPB 076 (B1)	**11.076**	falls ja, springe zu 076
056	VGL 102	**10.102**	ist der Hölzchenrest jetzt „13“?
057	SPB 076 (B1)	**11.076**	falls ja, springe zu 076
058	VGL 103	**10.103**	ist der Hölzchenrest jetzt „12“?

Fortsetzung siehe nächste Seite

Fortsetzung Listing 32:

Adresse		Mnemonics	Code	Kommentar
059		SPB 080 (B3)	**11.080**	falls ja, springe zu 080
060		VGL 104	**10.104**	ist der Hölzchenrest jetzt „11“?
061		SPB 078 (B2)	**11.078**	falls ja, springe zu 078
062		VGL 105	**10.105**	ist der Hölzchenrest jetzt „10“?
063		SPB 076 (B1)	**11.076**	falls ja, springe zu 076
064		VGL 106	**10.106**	ist der Hölzchenrest jetzt „9“?
065		SPB 076 (B1)	**11.076**	falls ja, springe zu 076
066		VGL 107	**10.107**	ist der Hölzchenrest jetzt „8“?
067		SPB 080 (B3)	**11.080**	falls ja, springe zu 080
068		VGL 108	**10.108**	ist der Hölzchenrest jetzt „7“?
069		SPB 078 (B2)	**11.078**	falls ja, springe zu 078
070		VGL 109	**10.109**	ist der Hölzenchrest jetzt „6“?
071		SPB 076 (B1)	**11.076**	falls ja, springe zu 076
072		VGL 110	**10.110**	ist der Hölzchenrest jetzt „5“?
073		SPB 090 (E1)	**11.090**	falls ja, springe zu 090 (Spieler gewinnt)
074		VKL 110	**13.110**	ist der Hölzchenrest kleiner als „5“?
075		SPB 093 (E3)	**11.093**	falls ja, springe zu 093 (Computer gewinnt)
076	B1:	LDA 111	**05.111**	lade „1“. . .
077		SPU 081 (B4)	**09.081**	und springe zum Speichern
078	B2:	LDA 112	**05.112**	lade „2“. . .
079		SPU 081 (B4)	**09.081**	und springe zum Speichern
080	B3:	LDA 113	**05.113**	lade „3“
081	B4:	ABS 115	**06.115**	speichere als Computer-Wegnahme
082		ANZ	**02.000**	zeige sie an . . .
083		VZG 255	**03.255**	} für eine Dreiviertelsekunde
084		VZG 255	**03.255**	
085		VZG 255	**03.255**	
086		LDA 100	**05.100**	lade den alten Hölzchenrest
087		SUB 115	**08.115**	subtrahiere davon die Computer-Wegnahme
088		ABS 100	**06.100**	speichere als neuen Hölzchenrest wieder
089		SPU 032 (A0)	**09.032**	springe zurück zu 032 und warte dort auf die nächste Eingabe am Port 1
090	E1:	AKO 000	**04.000**	lade „0“ als Zeichen des Spieler-Sieges
091	E2:	ANZ	**02.000**	zeige den Akku-Inhalt an . . .
092		SPU 091 (E2)	**09.091**	in einer Anzeigeschleife
093	E3:	LDA 116	**05.116**	lade „11.111“ als Zeichen des Computer-Sieges . . .
094		SPU 091 (E2)	**09.091**	und springe zum Anzeigen
100			**00.015**	Start-Hölzchenzahl und Zwischenstand
101			**00.014**	} Vergleichszahlen
102			**00.013**	
103			**00.012**	
104			**00.011**	
105			**00.010**	
106			**00.009**	
107			**00.008**	
108			**00.007**	
109			**00.006**	
110			**00.005**	
111			**00.001**	„1“ Hölzchen zum Wegnehmen
112			**00.002**	„2“ Hölzchen zum Wegnehmen
113			**00.003**	„3“ Hölzchen zum Wegnehmen
114			**–**	Wegnahme des Spielers (über Port 1)
115			**–**	Wegnahme des Computers (berechnet)
116		*	**11.111**	„11.111“ als Zeichen des Computer-Sieges
117			**00.000**	„0“ als Zeichen des Spieler-Sieges

2.6 Code-Knacker

Hier wird unser Grips vom Computer gefordert, denn er denkt sich drei Ziffern aus, die wir herausfinden müssen. Die drei Ziffern merkt er sich in den Zellen 104, 105 und 106. Wenn wir das Programm bei **001** starten, zeigt er uns zunächst „A 00.111" an als Zeichen dafür, daß wir raten dürfen. Wir tun das, indem wir das Programm durch Drücken von STP zunächst einmal anhalten, sodann unsere

Listing 33: **Code-Knacker**

Adresse		Mnemonics	**Code**	Kommentar
001		LDA 108	**05.108**	lade die aktuelle Rundenzahl in den Akku . . .
002		ADD 111	**07.111**	und addiere den Zufallszahlenwert dazu
003	A1:	SUB 110	**08.110**	subtrahiere „10" davon
004		VGR 110	**12.110**	ist jetzt noch mehr als „10" im Akku?
005		SPB 003 (A1)	**11.003**	falls ja, weiterhin subtrahieren bei 003
006		ABS 104	**06.104**	sonst ist eine Ziffer zwischen 0 und 9 übrig; speichere sie als 1. Computerziffer
007		AKO 014	**04.014**	lade „14"
008		ADD 104	**07.104**	addiere die 1. Computerziffer dazu
009		ADD 108	**07.108**	addiere die Rundenzahl dazu
010	A2:	SUB 110	**08.110**	subtrahiere „10" vom Akku-Inhalt
011	A5:	VGR 110	**12.110**	ist der Wert jetzt noch größer als „10"?
012		SPB 010 (A2)	**11.010**	falls ja, weiterhin subtrahieren bei 010
013		VGL 104	**10.104**	sonst vergleiche es mit der 1. Computerziffer
014		SPB 032 (B2)	**11.032**	falls Gleichheit, springe zu 032 (Korrektur)
015		ABS 105	**06.105**	sonst speichere Akku als 2. Computerziffer
016		AKO 017	**04.017**	lade „17"
017		ADD 104	**07.104**	addiere die 1. Computerziffer . . .
018		ADD 105	**07.105**	und die 2. Computerziffer
019	A3:	SUB 110	**08.110**	subtrahiere „10" vom Akku-Inhalt
020	A6:	VGR 110	**12.110**	ist der Wert jetzt noch größer als „10"?
021		SPB 019 (A3)	**11.019**	falls ja, subtrahiere weiter bei 019
022		VGL 104	**10.104**	sonst – ist der Wert gleich der 1. Computerziffer?
023		SPB 034 (B3)	**11.034**	falls ja, springe zu 034 (Korrektur)
024		VGL 105	**10.105**	ist der Wert gleich der 2. Computerziffer?
025		SPB 034 (B3)	**11.034**	falls ja, springe zu 034 (Korrektur)
026		ABS 106	**06.106**	sonst speichere als 3. Computerziffer
027		AKO 000	**04.000**	lade „0"
028		ABS 108	**06.108**	speichere als Rundenzahl fürs Raten
029		AKO 111	**04.111**	lade „111"
030	A4:	ANZ	**02.000**	zeige dies an als Ende der Zufallszahlenerzeugung
031		SPU 030 (A4)	**09.030**	springe zum Anzeigen zurück (Schleife)
032	B2:	ADD 109	**07.109**	Addiere „1" zum Akku-Inhalt (2. Computerziffer) . . .
033		SPU 011 (A5)	**09.011**	und springe zurück zum Test des Wertes
034	B3:	ADD 109	**07.109**	Addiere „1" zum Akku-Inhalt (3. Computerziffer). . .
035		SPU 020 (A6)	**09.020**	und springe zurück zum Test des Wertes
				Auswertung der Eingabe:
036	C1:	AKO 000	**04.000**	lade „0" und . . .
037		ABS 107	**06.107**	speichere als Ergebnis (Ausgangswert)
038		AKO 001	**04.001**	lade „1". . .
039		ADD 108	**07.108**	und erhöhe so den Rundenzähler
040		ABS 108	**06.108**	speichere ihn wieder
041		LDA 101	**05.101**	lade die 1. Spielerziffer
042		VGL 104	**10.104**	ist sie gleich der 1. Computerziffer?
043		SPB 065 (X1)	**11.065**	falls ja, springe zu 065 (addiere „10")
044		VGL 105	**10.105**	sonst: ist sie gleich der 2. Computerziffer?

Fortsetzung siehe nächste Seite

Fortsetzung Listing 33:

Adresse		Mnemonics	Code	Kommentar
045		SPB 077 (Y1)	**11.077**	falls ja, springe zu 077 (addiere „1“)
046		VGL 106	**10.106**	sonst: ist sie gleich der 3. Computerziffer?
047		SPB 077 (Y1)	**11.077**	falls ja, springe zu 077
048	C2:	LDA 102	**05.102**	lade die 2. Spielerziffer
049		VGL 105	**10.105**	ist sie gleich der 2. Computerziffer?
050		SPB 069 (X2)	**11.069**	falls ja, springe zu 069 (addiere „10“)
051		VGL 104	**10.104**	sonst: ist sie gleich der 1. Computerziffer?
052		SPB 081 (Y2)	**11.081**	falls ja, springe zu 081 (addiere „1“)
053		VGL 106	**10.106**	sonst: ist die gleich der 3. Computerziffer?
054		SPB 081 (Y2)	**11.081**	falls ja, springe zu 081
055	C3:	LDA 103	**05.103**	lade die 3. Spielerziffer
056		VGL 106	**10.106**	ist sie gleich der 3. Computerziffer?
057		SPB 073 (X3)	**11.073**	falls ja, springe zu 073 (addiere „10“)
058		VGL 104	**10.104**	sonst: ist sie gleich der 1. Computerziffer?
059		SPB 085 (Y3)	**11.085**	falls ja, springe zu 085 (addiere „1“)
060		VGL 105	**10.105**	sonst: ist sie gleich der 2. Computerziffer?
061		SPB 085 (Y3)	**11.085**	falls ja, springe zu 085
062		LDA 107	**05.107**	lade das Ergebnis der Auswertung
063	E1:	ANZ	**02.000**	zeige es an
064		SPU 063 (E1)	**09.063**	springe zurück zum Anzeigen (Schleife)
065	X1:	LDA 107	**05.107**	lade die Ergebniszelle . . .
066		ADD 110	**07.110**	und erhöhe sie um „10“: Platz stimmt!
067		ABS 107	**06.107**	speichere das Ergebnis wieder . . .
068		SPU 048 (C2)	**09.048**	und komme zum Test der 2. Ziffer
069	X2:	LDA 107	**05.107**	lade die Ergebniszelle . . .
070		ADD 110	**07.110**	und erhöhe sie um „10“: Platz stimmt!
071		ABS 107	**06.107**	speichere das Ergebnis wieder . . .
072		SPU 055 (C3)	**09.055**	und komme zum Test der 3. Ziffer
073	X3:	LDA 107	**05.107**	lade die Ergebniszelle . . .
074		ADD 110	**07.110**	und erhöhe sie um „10“: Platz stimmt!
075		ABS 107	**06.107**	speichere das Ergebnis wieder . . .
076		SPU 063 (E1)	**09.063**	und springe zum Anzeigen des Ergebnisses
077	Y1:	LDA 107	**05.107**	lade die Ergebniszelle . . .
078		ADD 109	**07.109**	und erhöhe sie um „1“: Ziffer stimmt!
079		ABS 107	**06.107**	speichere das Ergebnis wieder . . .
080		SPU 048 (C2)	**09.048**	und komme zum Test der 2. Ziffer
081	Y2:	LDA 107	**05.107**	lade die Ergebniszelle . . .
082		ADD 109	**07.109**	und erhöhe sie um „1“: Ziffer stimmt!
083		ABS 107	**06.107**	speichere das Ergebnis wieder . . .
084		SPU 055 (C3)	**09.055**	und komme zum Test der 3. Ziffer
085	Y3:	LDA 107	**05.107**	lade die Ergebniszelle . . .
086		ADD 109	**07.109**	und erhöhe sie um „1“: Ziffer stimmt!
087		ABS 107	**06.107**	speichere das Ergebnis wieder . . .
088		SPU 063 (E1)	**09.063**	und springe zum Anzeigen des Ergebnisses
101		*	**00.xxx**	1. Spielerziffer
102		*	**00.xxx**	2. Spielerziffer
103		*	**00.xxx**	3. Spielerziffer
104			–	1. Computerziffer
105			–	2. Computerziffer
106			–	3. Computerziffer
107			–	Ergebnis der Auswertung
108			–	Rundenzähler
109		*	**00.001**	„1“ zum Weiterzählen
110		*	**00.010**	„10“ zum Weiterzählen
111		*	**00.xxx**	Zufallszahlenstartwert (min. „10“, max. „220“)

Zahlen in die Zellen 101, 102 und 103 eingeben und das Programm bei **036** starten. Der Computer vergleicht unsere Eingabe mit dem, was er sich „ausgedacht" hat und zeigt uns das Ergebnis an. (Die Zahlen bewegen sich übrigens im Bereich 0–9, damit es nicht so schwer ist.) Die Ergebnisanzeige ist wie folgt: Die *Zehnerstelle* zeigt uns, *wieviele* Ziffern richtig sind *und* an der *richtigen* Stelle stehen – aber natürlich nicht welche.
Die *Einerstelle* zeigt uns, *wieviele* Ziffern *richtig* geraten wurden, aber noch *falsch* stehen.

1. Beispiel: Computer denkt sich: 9 und 0 und 7
   Wir versuchen: 1 und 2 und 7
   Anzeige: A 00.010
   (eine richtige Ziffer am richtigen Platz)
2. Beispiel: Computer denkt sich: 3 und 4 und 8
   Wir versuchen: 4 und 5 und 3
   Anzeige: A 00.002
   (zwei richtige Ziffern am falschen Platz)

Zeigt der Computer als Ergebnis A 00.030 an, sind wir fertig – alle drei Ziffern stehen an der richtigen Stelle! Erscheint aber etwas Anderes, so müssen wir den Computer stoppen und durch neue Eingaben in 101 bis 103 weiterraten, bis wir die richtige Lösung gefunden haben; also drei neue Ziffern eingeben und wieder bei **036** starten.
Noch ein Hinweis: Für gutes Arbeiten des Zufallszahlengenerators (Programmzeilen 001–035) sollte vor dem ersten Start in Zelle 111 irgendetwas eingegeben werden. Das Programm ist außerdem so geschrieben, daß sich die drei Ziffern *stets* unterscheiden; doppelte tauchen nicht auf.

2.7 Computer-Schaltuhr

Mechanische und elektronische Schaltuhren, die zu einer bestimmten Zeit ein Gerät ein- und zu einer anderen Zeit ausschalten, können ohne Schwierigkeiten durch unseren vielseitigen Computer ersetzt werden. Unser Programm arbeitet wie folgt: Vor dem Start wird die aktuelle Uhrzeit in die Zellen 125–127, die Einschaltzeit (nur Minute und Stunde) in 123 und 124, sowie die Ausschaltzeit in 121 und 122 eingegeben. Bei den Schaltminuten ist übrigens *immer eine weniger als tatsächlich ge-*

Bild 69

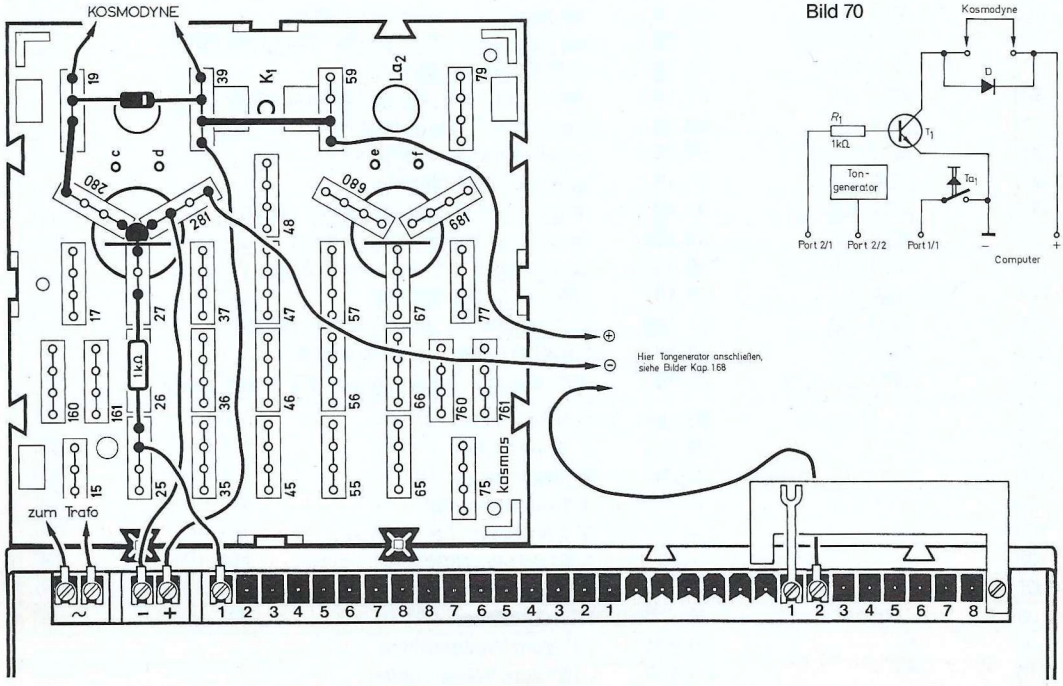


Bild 70

wünscht einzugeben; das liegt am VGR-Befehl in Zelle 068. Und noch eine Bemerkung zu den vielleicht etwas merkwürdigen Werten bei den VZG-Befehlen: da der Computer zur Ausführung der Befehle eine gewisse Zeit benötigt, muß diese für eine korrekte Zeitzählung berücksichtigt werden. So haben wir bei der Erstellung des Programms einfach ausprobiert, mit welchen Verzögerungen die richtige Zeit am besten erreicht wird.

Was wir schalten wollen, müssen wir natürlich auch anschließen: An Port 2 Klemme 1 beispielsweise über einen Treibertransistor ein Relais, das dann den Strom zum Gerät schaltet. Ist das zum Beispiel ein Radio, so wird es durch den Computer zum Radiowecker! Wir starten das Programm bei **001.** Es zählt die aktuelle Uhrzeit fort, und sobald die Einschaltzeit erreicht ist, wird an Port 2 Klemme 1 eine „1" ausgegeben. Das Relais zieht an, und das Gerät wird eingeschaltet. Sobald die Ausschaltzeit erreicht ist, gibt das Programm eine „0" an Port 2 Klemme 1 und schaltet das Relais wieder aus. An dieser Stelle stoppt das Programm! Es ist aber nicht allzu schwer, es so umzuschreiben, daß die Zeit danach noch weiterlaufen kann. Nur Vorsicht: nach dem Einschaltvorgang wird diese Zeit mit der Ausschaltzeit überschrieben! Wenn der Ein- und Ausschaltvorgang täglich wiederholt werden soll, muß dafür gesorgt werden, daß die Einschaltzeit nicht verlorengeht. Das heißt, in Zeile 076 ff. muß ein echter Austausch zwischen Ein- und Ausschaltzeit programmiert werden!

Mit zusätzlichen elektronischen Bauteilen können noch zwei Besonderheiten des Programms verwendet werden. Legt man zwischen 0 V und Port 1 Klemme 1 einen Taster und betätigt diesen, so wird die aktuelle Zeit (erst Stunde, dann Minute) angezeigt.

Weiterhin kann an Port 1 Klemme 2 ein Tongenerator angeschlossen werden, der bei jedem Stundenschlag einen Ton abgibt. Bei der Verwendung als Radiowecker ist das aber nicht unbedingt zu empfehlen; wer will schon jede Stunde der Nacht einen Pieps hören?

Listing 34: **Computer-Schaltuhr**

Adresse	Mnemonics	**Code**	Kommentar
001	AKO 024	**04.024**	lade „24"
002	ABS 120	**06.120**	speichere als Stundenvergleichszahl
003	AKO 060	**04.060**	lade „60"
004	ABS 119	**06.119**	speichere als Minuten-/Sekundenvergleich
005	AKO 000	**04.000**	lade „0"
006	ABS 118	**06.118**	speichere als Vergleichszahl . . .
007	ABS 117	**06.117**	und als AUS-Zustand des Gerätes
008	P1A 000	**17.000**	gib die „0" aus an Port 1 . . .
009	P2A 000	**18.000**	und an Port 2: alles ausschalten
010	LDA 120	**05.120**	lade „24"
011	VKL 127	**13.127**	ist dies kleiner als die aktuelle Stunde?
012	SPB 027 (E0)	**11.027**	falls ja, springe zu 027 (Fehleingabe-Halt)
013	VKL 124	**13.124**	ist dies kleiner als die Einschalt-Stunde?
014	SPB 027 (E0)	**11.027**	falls ja, springe zu 027 (Fehleingabe-Halt)
015	VKL 122	**13.122**	ist dies kleiner als die Ausschalt-Stunde?
016	SPB 027 (E0)	**11.027**	falls ja, springe zu 027 (Fehleingabe-Halt)
017	LDA 119	**05.119**	lade „60"
018	VKL 126	**13.126**	ist dies kleiner als die aktuelle Minute?
019	SPB 027 (E0)	**11.027**	falls ja, springe zu 027 (Fehleingabe-Halt)
020	VKL 125	**13.125**	ist dies kleiner als die aktuelle Sekunde?
021	SPB 027 (E0)	**11.027**	falls ja, springe zu 027 (Fehleingabe-Halt)
022	VKL 123	**13.123**	ist dies kleiner als die Einschalt-Minute?
023	SPB 027 (E0)	**11.027**	falls ja, springe zu 027 (Fehleingabe-Halt)
024	VKL 121	**13.121**	ist dies kleiner als die Ausschalt-Minute?
025	SPB 027 (E0)	**11.027**	falls ja, springe zu 027 (Fehleingabe-Halt)
026	SPU 028 (C1)	**09.028**	sonst springe zu 028 (Zeit läuft)
027	E0: HLT	**01.000**	anhalten – fehlerhafte Eingabe einer Zeit

Fortsetzung siehe nächste Seite

Fortsetzung Listing 34:

Adresse		Mnemonics	Code	Kommentar
028	C1:	VZG 200	**03.200**	warte 1 Sekunde
029	C2:	VZG 187	**03.187**	
030		VZG 250	**03.250**	
031		VZG 250	**03.250**	
032		P1E 001	**16.001**	Lies Port 1 Klemme 1: Tasteninformation
033		VGL 118	**10.118**	ist sie „0", also Taste gedrückt?
034		SPB 082 (Z1)	**11.082**	falls ja, springe zum Anzeigen bei 082
035		AKO 001	**04.001**	sonst lade „1"
036		ADD 125	**07.125**	und erhöhe damit die Sekunden
037		ANZ	**02.000**	und zeige sie an . . .
038		ABS 125	**06.125**	und speichere sie wieder
039		VKL 119	**13.119**	sind sie noch kleiner als „60"?
040		SPB 028 (C1)	**11.028**	falls ja, Rücksprung zum Weiterzählen
041	C3:	AKO 000	**04.000**	sonst lade „0"
042		ABS 125	**06.125**	als Sekundenstand speichern
043		AKO 001	**04.001**	lade „1"
044		ADD 126	**07.126**	addiere dazu den Minutenstand
045		ABS 126	**06.126**	und speichere die neue Minute wieder
046		VKL 119	**13.119**	ist die Minute noch kleiner als „60"?
047		SPB 062 (S1)	**11.062**	falls ja, zur nächsten Sekunde springen
048		VZG 180	**03.180**	Zeitkorrektur – siehe Text
049		AKO 000	**04.000**	eine neue Stunde bricht an: lade „0"
050		ABS 126	**06.126**	speichere sie als Minutenstand
051		AKO 001	**04.001**	lade „1"
052		P1A 002	**17.002**	Ausgabe an Port 1 Klemme 2: Tongenerator an
053		VZG 200	**03.200**	für 200 ms
054		ADD 127	**07.127**	addiere zur „1" im Akku die bisherige Stunde . . .
055		ABS 127	**06.127**	und speichere sie wieder
056		VKL 120	**13.120**	ist die Stunde noch kleiner als „24"?
057		AKO 000	**04.000**	lade mal schon „0"
058		P1A 002	**17.002**	Ausgabe an Port 1 Klemme 2: Tongenerator aus
059		SPB 063 (S2)	**11.063**	falls Vergleich in 056 richtig, springe zu 063
060		ABS 127	**06.127**	sonst speichere die „0" als neue Stunde
061		SPU 063 (S2)	**09.063**	fertig mit dem Ton: weiter in der Zeitschleife
062	S1:	VZG 160	**03.160**	Zeitkorrektur – siehe Text
063	S2:	LDA 124	**05.124**	lade die Schaltstunde
064		VGL 127	**10.127**	ist sie gleich der aktuellen Stunde?
065		SPB 067 (S3)	**11.067**	falls ja, springe zur Minutenkontrolle
066		SPU 029 (C2)	**09.029**	sonst Rücksprung in die Sekundenroutine, da Schaltzeit noch nicht erreicht
067	S3:	LDA 123	**05.123**	lade die Schaltminute
068		VGR 126	**12.126**	ist sie noch größer als die aktuelle Minute?
069		SPB 029	**11.029**	falls ja, wird noch nicht geschaltet: weiter in der Schleife
070		LDA 117	**05.117**	sonst lade den Gerätezustand („1" oder „0")
071		NEG	**14.000**	invertiere ihn: AUS wird EIN und umgekehrt
072		P2A 001	**18.001**	Ausgabe an Port 2 Klemme 1: Relaissteuerung
073		ABS 117	**06.117**	dann speichere den Zustand wieder
074		VGL 118	**10.118**	ist er „0", das Gerät also aus?
075		SPB 027 (E0)	**11.027**	falls ja, springe zum HALT
076		LDA 122	**05.122**	sonst lade die Ausschaltstunde . . .

Fortsetzung siehe nächste Seite

Fortsetzung Listing 34:

Adresse	Mnemonics	Code	Kommentar
077	ABS 124	**06.124**	und speichere sie in der Einschalt-Zelle!
078	LDA 121	**05.121**	lade auch die Ausschaltminute . . .
079	ABS 123	**06.123**	und speichere sie: Einschaltzeit ist jetzt weg!
080	AKO 001	**04.001**	lade „1" zur Zeitkorrektur
081	SPU 093 (K0)	**09.093**	dann zur Korrekturroutine springen
082	Z1: LDA 127	**05.127**	Unterprogramm zur Zeitanzeige: Stunde laden
083	ANZ	**02.00**	anzeigen
084	VZG 250	**03.250**	750 ms warten
085	VZG 250	**03.250**	
086	VZG 250	**03.250**	
087	LDA 126	**05.126**	Minute laden . . .
088	ANZ	**02.000**	und anzeigen
089	VZG 250	**03.250**	750 ms warten
090	VZG 250	**03.250**	
091	VZG 250	**03.250**	
092	AKO 002	**04.002**	Zeitkorrektur: 2 Sekunden mehr sind vergangen
093	KO: ADD 125	**07.125**	diese zur aktuellen Sekunde addieren . . .
094	ABS 125	**06.125**	und wieder speichern
095	VKL 119	**13.119**	ist sie noch kleiner als „60"?
096	SPB 029 (C2)	**11.029**	falls ja, weiterzählen in der normalen Schleife
097	SUB 119	**08.119**	sonst um „60" verringern
098	ABS 125	**06.125**	speichern der Sekunde . . .
099	SPU 041 (C3)	**09.041**	und zur Minutenkontrolle springen
117		**00.000**	Vergleichszahl
118		**00.000**	Vergleichszahl
119		**00.060**	Vergleichszahl (Minuten u. Sekunden)
120		**00.024**	Vergleichszahl (Stunden)
121	*	**00.0xx**	Minuten — Ausschaltzeit
122	*	**00.0xx**	Stunden — Ausschaltzeit
123	*	**00.0xx**	Minuten — Einschaltzeit
124	*	**00.0xx**	Stunden — Einschaltzeit
125	*	**00.0xx**	Sekunden — aktuelle Uhrzeit
126	*	**00.0xx**	Minuten — aktuelle Uhrzeit
127	*	**00.0xx**	Stunden — aktuelle Uhrzeit

2.8 Gedächtnistraining

Mit diesem Programm können Sie Ihre Auffassungsgabe und Ihr Kurzzeitgedächtnis testen. Dazu zeigt Ihnen der Computer eine Reihe von Zahlen, die Sie sich merken sollen. Dann stoppt das Programm, und Sie müssen die Zahlen, die Sie sich merken konnten, in den Speicher eingeben (Zellen 111–120). Nach einem erneuten Programmstart an der Stelle, wo er automatisch anhielt, wird die Trefferquote berechnet und angezeigt. Aber Achtung: Zahlen werden nur dann als Treffer anerkannt, wenn sie auch an der richtigen Stelle in der Reihe stehen; tauchte beispielsweise die Zahl „73" an der vierten Stelle auf, wird aber als zweite eingegeben (also in Zelle 112), so wird dies nicht als richtig ausgewertet.

Um Zufallszahlen zu erzeugen, müssen wir vor dem Programmstart die beiden Ports miteinander verbinden, wie dies in Kapitel 2.0 beschrieben wurde. So erreichen wir eine gute, „wilde" Zahlenverteilung. Außerdem müssen wir in Zelle 124 einen Startwert für die Zufallszahlenerzeugung eingeben. In Zelle 123 können wir dann angeben, wieviele Zahlen uns nacheinander vorgeführt werden sollen. Es sind maximal „10" möglich – aber die merkt sich zu Anfang wohl niemand! Wird in 123 etwas Größeres als „10" eingegeben, so gibt das Programm „A 00.111" als Fehlermeldung aus.
Die Zufallszahlenerzeugung und -vorstellung beginnt bei **001,** die Auswertung bei **047.**
(Siehe Listing S. 88 u. 89.)

Listing 35: **Gedächtnistraining**

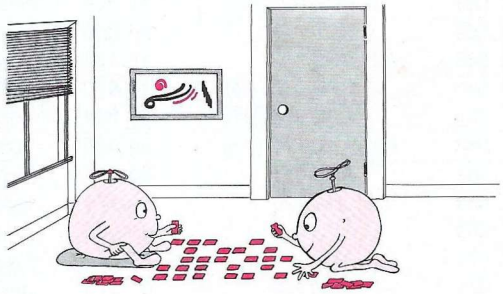
Adresse	Mnemonics	Code	Kommentar
001	AKO 000	**04.000**	lade „0“
002	ABS 100	**06.100**	speichere als Trefferquote,
003	ABS 125	**06.125**	als Zähler . . .
004	ABS 122	**06.122**	und als Vergleichszahl
005	AKO 001	**04.001**	lade „1“
006	ABS 121	**06.121**	speichere als Schrittweite
007	AKO 010	**04.010**	lade „10“
008	VKL 123	**13.123**	ist das kleiner als die gewünschte Anzahl?
009	SPB 072 (E1)	**11.072**	falls ja, springe zur Fehlermeldung
010	A1: LDA 125	**05.125**	lade den Zähler
011	ADD 121	**07.121**	erhöhe um „1“. . .
012	ABS 125	**06.125**	und speichere wieder
013	VGR 123	**12.123**	Zähler größer als Anzahl, also genug erzeugt?
014	SPB 031 (B1)	**11.031**	falls ja, springe zum Anzeigen
015	AKO 100	**04.100**	lade „100“ . . .
016	ADD 125	**07.125**	und addiere den Zähler dazu
017	ABS 126	**06.126**	speichere dies als Adresse (Zeile 029)
018	LDA 124	**05.124**	lade die Zufallszahl
019	VGR 122	**12.122**	ist sie größer als „0“?
020	SPB 022 (A2)	**11.022**	falls ja, springe zu 022
021	ADD 126	**07.126**	sonst ist sie „0“! Erhöhe sie um die Adresse
022	A2: P2A 000	**18.000**	Ausgabe der Zufallszahl an Port 2
023	VZG 010	**03.010**	warte kurz
024	P1E 000	**16.000**	lies Port 1: bitvertauschte Zufallszahl
025	VKL 123	**13.123**	ist sie kleiner als die Anzahl?
026	SPB 028 (A3)	**11.028**	falls ja, springe direkt zum Speichern
027	SUB 121	**08.121**	sonst verringere sie um „1“
028	A3: ABS 124	**06.124**	speichere als Zufallszahl . . .
029	AIS 126	**20.126**	und als Computerzahl zum Vorzeigen
030	SPU 010 (A1)	**09.010**	dann erzeuge die nächste in der Schleife
031	B1: AKO 001	**04.001**	Anzeigeroutine – lade „1“
032	B2: ABS 125	**06.125**	speichere als Zähler fürs Anzeigen
033	VGR 123	**12.123**	ist er größer als die Anzahl?
034	SPB 046 (B3)	**11.046**	falls ja, sind alle angezeigt und wir stoppen
035	AKO 100	**04.100**	sonst lade „100“ . . .
036	ADD 125	**07.125**	und addiere den Zähler dazu
037	ABS 126	**06.126**	speichere als Adresse
038	LIA 126	**19.126**	lade damit indirekt die Computerzahl . . .
039	ANZ	**02.000**	und zeige sie an
040	VZG 200	**03.200**	drei Verzögerungen, die insgesamt eine
041	VZG 100	**03.100**	halbe Sekunde ausmachen; sie können beliebig
042	VZG 200	**03.200**	geändert werden (Anzeigedauer)
043	LDA 125	**05.125**	lade den Zähler
044	ADD 121	**07.121**	erhöhe ihn um „1“. . .
045	SPU 032 (B2)	**09.032**	und mache weiter in der Anzeigeroutine
046	B3: HLT	**01.000**	hier stoppt die Anzeigeroutine
047	AKO 001	**04.001**	Vergleichsroutine – lade „1“
048	C1: ABS 125	**06.125**	speichere als Zähler fürs Vergleichen
049	VGR 123	**12.123**	schon mehr als die gewünschte Anzahl?
050	SPB 069 (D1)	**11.069**	falls ja, zum Endresultat springen
051	AKO 100	**04.100**	sonst lade „100“
052	ADD 125	**07.125**	addiere dazu den Zähler

Fortsetzung siehe nächste Seite

Fortsetzung Listing 35:

Adresse	Mnemonics	Code	Kommentar
053	ABS 126	**06.126**	speichere als Adresse der Computerzahl
054	LIA 126	**19.126**	lade damit indirekt die Computerzahl . . .
055	ABS 127	**06.127**	und speichere sie in 127
056	AKO 110	**04.110**	dann lade „110"
057	ADD 125	**07.125**	addiere dazu den Zähler
058	ABS 126	**06.126**	speichere als Adresse der Benutzerzahl
059	LIA 126	**19.126**	lade damit die vom Spieler geratene Zahl
060	VGL 127	**10.127**	ist sie gleich der Computer-Zahl?
061	SPB 065 (C3)	**11.065**	falls ja, springe zum Verrechnen
062	C2: LDA 125	**05.125**	sonst lade den Zähler
063	ADD 121	**07.121**	erhöhe um „1". . .
064	SPU 048 (C1)	**09.048**	und mache weiter in der Auswertung
065	C3: LDA 100	**05.100**	Verrechnungsroutine – lade die Trefferquote
066	ADD 121	**07.121**	erhöhe sie um „1". . .
067	ABS 100	**06.100**	und speichere sie wieder
068	SPU 062 (C2)	**09.062**	springe zum Vergleich der nächsten Zahl
069	D1: LDA 100	**05.100**	Anzeigeroutine – lade die Trefferquote . . .
070	D2: ANZ	**02.000**	und zeige sie an
071	SPU 070 (D2)	**09.070**	in einer Anzeigeschleife
072	E1: AKO 111	**04.111**	Fehlerroutine – lade „111"
073	E2: ANZ	**02.000**	zeige sie an: In 123 steht mehr als „10"!!
074	SPU 073 (E2)	**09.073**	in einer Anzeigeschleife
100		**–**	Trefferquote
101		**–**	maximal zehn Computerzahlen
. . .		**. . .**	
110		**–**	
111	*	**00.xxx**	die maximal zehn (vom Spieler geratenen) Benutzerzahlen
. . .	*	**. . .**	
120	*	**00.xxx**	
121		**00.001**	„1" als Schrittweite
122		**00.000**	„0" als Vergleichszahl
123	*	**00.0xx**	Anzahl vorzustellender Zahlen
124	*	**00.xxx**	Startwert für Zufallszahlenerzeugung
125		**–**	Zähler
126		**–**	Adresszelle für indirekte Adressierung
127		**–**	Hilfszelle beim Vergleichen

2.9 Pärchen-Suche



Spielen Sie zuhause auch „Memory"? Dann wissen Sie, wie das normalerweise geht: Es gibt einen Kartensatz mit Bildern, wobei jedes Bild zweimal vorkommt. Diese Karten werden verdeckt (also mit dem Bild nach unten) auf dem Tisch verteilt, und das Ziel ist es nun, die Paare zu entdecken. Dazu dürfen alle Mitspieler reihum immer zwei Karten aufdecken, und wenn die Bilder übereinstimmen, darf sie der „Entdecker" behalten. Außerdem kann er noch ein weiteres Paar suchen, also nochmals aufdecken. Weisen sie jedoch zwei unterschiedliche Bilder auf, müssen sie an der alten Stelle wieder verdeckt abgelegt werden. Dann ist der nächste Spieler dran. Es ist natürlich am Anfang des Spiels ziemlich unergiebig, weil andauernd nur unterschiedliche Bilder aufgedeckt werden, aber mit der Zeit merken sich die Spieler, wo einzelne Bilder schon einmal aufgetaucht sind und können

dann beim Erscheinen des anderen gezielt das erste aufdecken.

Und das spielen wir jetzt mit dem Computer; natürlich nicht gegen ihn, er hat ja bekanntlich ein phänomenales Gedächtnis. Nein, er soll nur Spielgerät und Schiedsrichter zwischen zwei Menschen sein, die gegeneinander spielen. Der Computer mischt also nur die Karten, legt sie aus und zählt dann, wieviele Paare jeder Spieler gefunden hat. Die „Karten" sind natürlich nur Zahlen, und „ausgelegt" werden sie nur im Speicher, nicht auf dem Tisch. Aber jetzt im Einzelnen, was alles passiert:

1. Beim Programmstart bei **037** werden 36 Felder, die man sich in einem 6 × 6-Quadrat vorstellen kann, mit Zahlenpaaren von 0 bis 17 belegt; das Spielfeld sieht also so aus:

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

Listing 36: **Pärchen-Suche**

Adresse		Mnemonics	**Code**	Kommentar
037		LDA 024	**05.024**	lade den Inhalt von Feld 024 als Startwert
038		ABS 120	**06.120**	für die Zufallsbelegung in 120 speichern
039		AKO 001	**04.001**	mit Feld 1 soll angefangen werden
040		ABS 116	**06.116**	deshalb die „1" in der Feldadreßzelle speichern
041		AKO 043 (I1)	**04.043**	lade „043" als Rücksprungadresse
042		SPU 090 (Z1)	**09.090**	und springe in die Zufallsbelegung
043	I1:	LDA 011	**05.011**	lade den Inhalt von Feld 011 als Startwert
044		ABS 120	**06.120**	für die Belegung der 2. Hälfte speichern
045		AKO 047 (I2)	**04.047**	„047" ist die Rücksprungadresse
046		SPU 090 (Z1)	**09.090**	aus der hier aufgerufenen Belegungsroutine
047	I2:	AKO 000	**04.000**	lade „0" für folgende Speicherzellen:
048		ABS 125	**06.125**	Spielernummer
049		ABS 118	**06.118**	Punktekonto 0. Spieler
050		ABS 119	**06.119**	Punktekonto 1. Spieler
051		HLT	**01.000**	HALT – Feldbelegung ist fertig
052		AKO 066 (A3)	**04.066**	lade „066" als Rücksprungadresse . . .
053		ABS 117	**06.117**	und speichere für den indirekten Sprung
054		LIA 127	**19.127**	lade das 2. Geratene über die Adreßzelle 127
055		VKL 123	**13.123**	ist es noch kleiner als „100", also nicht gefunden?
056		SPB 058 (A1)	**11.058**	falls ja, weiter auswerten
057		HLT	**01.000**	sonst ist Fehleingabe erfolgt: Feld ist schon weg!
058	A1:	ABS 120	**06.120**	speichere das 2. Geratene zwischendurch für den Vergleich
059		LIA 126	**19.126**	lade über die andere Adreßzelle 126 das 1. Geratene
060		VKL 123	**13.123**	ist es kleiner als „100", also noch nicht gefunden?
061		SPB 063 (A2)	**11.063**	falls ja, zum direkten Vergleich der beiden springen
062		HLT	**01.000**	sonst HALT – Fehleingabe wie oben! Nochmal versuchen
063	A2:	VGL 120	**10.120**	vergleiche den 1. mit dem zwischengespeicherten 2.
064		SPB 070 (A4)	**11.070**	falls gleich, zur Paaranzeige springen
065		SPU 084 (B1)	**09.084**	sonst zum Anzeigen des 1. Wertes springen
066	A3:	AKO 080 (A5)	**04.080**	„080" als Rücksprungadresse laden
067		ABS 117	**06.117**	und speichern in 117
068		LDA 120	**05.120**	lade den 2. Wert . . .
069		SPU 084 (B1)	**09.084**	und zeige auch ihn an in der Anzeigeroutine
070	A4:	ANZ	**02.000**	Paar gefunden! Den Wert anzeigen
071		ADD 123	**07.123**	um „100" erhöhen
072		AIS 126	**20.126**	und so den 1. wie den . . .
073		AIS 127	**20.127**	2. Wert als „gefunden" markieren
074		AKO 118	**04.118**	lade „118" als Punktekontenanfangsadresse
075		ADD 125	**07.125**	erhöhe diese um die Spielernummer (0 oder 1) . . .

Fortsetzung siehe nächste Seite

Fortsetzung Listing 36:

Adresse		Mnemonics	Code	Kommentar
076		ABS 116	**06.116**	und speichere in der Adreßzelle
077		LIA 116	**19.116**	lade so indirekt das richtige Punktekonto (118 oder 119) . .
078		ADD 121	**07.121**	und erhöhe es um „1“
079		AIS 116	**20.116**	dann speichere es wieder
080	A5:	LDA 125	**05.125**	jetzt ist der andere dran – lade die Spielernummer
081		NEG	**14.000**	negiere sie („1“ wird „0“ und umgekehrt) . . .
082		ABS 125	**06.125**	und speichere sie wieder
083		HLT	**01.000**	HALT – jetzt Eingabe machen durch den anderen Spieler
084	B1:	ANZ	**02.000**	Anzeige des 1. bzw. 2. Wertes
085		VZG 255	**03.255**	} für 1 Sekunde
086		VZG 255	**03.255**	
087		VZG 255	**03.255**	
088		VZG 255	**03.255**	
089		SIU 117 (AX)	**21.117**	dann Rücksprung, wie vor dem Aufruf in 117 gespeichert
090	Z1:	ABS 117	**06.117**	Zufallsbelegungsroutine – speichere den Akkuinhalt als Rücksprungadresse in 117
091		AKO 000	**04.000**	lade „0“. . .
092		ABS 000	**06.000**	und speichere als Rundenzahl für Belegung
093	Z2:	LDA 120	**05.120**	lade den Ausgangswert
094		P2A 000	**18.000**	gib ihn an Port 2 aus . . .
095		P1E 000	**16.000**	und an Port 1 wieder hinein – bitvertauscht
096		VGR 122	**12.122**	größer als „17“, also nicht erlaubt?
097		SPB 102 (Z3)	**11.102**	falls ja, spare Dir das Speichern
098		AIS 116	**20.116**	sonst speichere den Wert indirekt über die Spielfeldadreßzelle
099		AKO 001	**04.001**	lade „1“
100		ADD 116	**07.116**	erhöhe so die Spielfeldadresse um „1“. . .
101		ABS 116	**06.116**	und speichere sie wieder
102	Z3:	LDA 000	**05.000**	lade die Rundenzahl
103		VGL 124	**10.124**	ist sie schon bei „255“ angelangt?
104		SPB 115 (Z6)	**11.115**	falls ja, heraussprungen aus der Schleife
105		ADD 121	**07.121**	sonst um „1“ erhöhen . . .
106		ABS 000	**06.000**	und wieder speichern
107		LDA 120	**05.120**	lade den Ausgangswert der Zufallszahlerzeugung
108		VGL 124	**10.124**	ist er „255“?
109		SPB 112 (Z4)	**11.112**	falls ja, zurücksetzen bei 112 auf „0“
110		ADD 121	**07.121**	sonst um „1“ erhöhen . . .
111		SPU 113 (Z5)	**09.113**	und zum Speichern springen
112	Z4:	AKO 000	**04.000**	setze den Ausgangswert auf „0“
113	Z5:	ABS 120	**06.120**	speichere ihn . . .
114		SPU 093 (Z2)	**09.093**	und weiter in der Zufallsschleife
115	Z6:	SIU 117 (IX)	**21.117**	fertig – Rücksprung wie gewünscht indirekt
116			–	Adreßzelle für Feldadressierung
117			–	Rücksprungadresse für indirekte Sprünge
118			–	Punktekonto Spieler 0
119			–	Punktekonto Spieler 1
120			–	Hilfszelle (Zwischenspeicher)
121		*	**00.001**	Schrittweite
122		*	**00.017**	Maximalwert eines Feldes
123		*	**00.100**	„100“ zum Paaremarkieren
124		*	**00.255**	Vergleichszahl – Obergrenze des Zahlbereichs
125			–	Spieler, der gerade dran ist (0 oder 1)
126		*	**00.xxx**	1. geratenes Feld
127		*	**00.xxx**	2. geratenes Feld
000			–	Zählerzelle für Zufallsbelegung

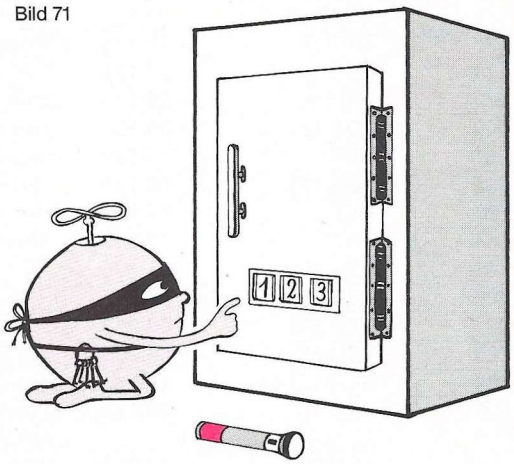
(Das Spielfeld benutzt übrigens gerade die Speicherzellen 001 bis 036 – aber nicht mogeln und anschauen!)
Wenn jede Zahl zwischen 0 und 17 zweimal im Speicher ist, stoppt das Programm mit der Anzeige P 052.
(Noch eine Bemerkung zur Feldbelegung: Als Startwert für die Zufallszahlenerzeugung wird der Inhalt von Zelle 024 verwendet. Um also eine unterschiedliche Belegung des Feldes zu erhalten, sollte nach dem Eintippen des Programms dort eine beliebige Zahl eingegeben werden.)

2. Jetzt kann der erste Spieler raten, wo sich ein Paar versteckt hält. Die beiden Feldnummern gibt er dazu in die Speicherzellen 126 und 127 ein. Ein Start des Programmes bei **052** zeigt dann für jeweils eine Sekunde den Inhalt der beiden Felder an. Wahrscheinlich waren sie nicht gleich, und das Programm stoppt mit der Meldung P 084. Die Spieler müssen sich natürlich merken, welche Zahlen in den beiden Feldern stehen – aber bitte nicht notieren!
3. Wenn aber ein Paar gefunden wurde, blitzt die Anzeige nur kurz auf und stoppt bei P 084. In diesem Fall dürfen die beiden Felder als gefunden markiert werden (vielleicht mit einem Bleistiftkreuz), damit sie nicht noch einmal aufgedeckt werden. Auch das Programm notiert die beiden gefundenen Felder und erhöht das Punktekonto des Spielers um 1.
4. Dann ist *immer* der andere Spieler dran. Das geht genauso wie unter 2. beschrieben. Der Computer weiß schon, daß der andere dran ist. (Die Spieler haben übrigens programmintern die Nummern „0" und „1", der aktive Spieler ist in Zelle 125 notiert.)
5. Sind alle Felder aufgedeckt, können beide Punktekonten in den Zellen 118 und 119 angesehen werden.
6. Wird versehentlich ein Feld eingegeben, dessen Inhalt schon erraten wurde, taucht automatisch ein HALT auf; auf der Anzeige steht dann P 058, wenn das erste Feld schon aufgedeckt ist bzw. P 063, wenn das zweite Feld schon aufgedeckt ist. Dann kann die Eingabe vom *selben* Spieler wiederholt werden.

Vor dem Start müssen aber noch die beiden Ports, wie in Kapitel 2.0 beschrieben, miteinander verbunden werden. Dann klappt auch die gute Zufallsbelegung des Feldes.
Außerdem müssen (natürlich) das Programm sowie die Daten in die Zellen 121–124 eingegeben werden. Und los geht's bei Punkt 1!

2.10 Codeschloß mit Alarmanlage

Bild 71



Der Computer kann hervorragend als „intelligente Alarmanlage" benutzt werden, um einen Safe, die Haustür oder ein Auto zu sichern. Aber auch dazu braucht er natürlich ein Programm. Das hier vorgestellte arbeitet wie folgt: Vor dem Start werden drei Taster an Port 1, Klemmen 1 bis 3 angeschlossen, außerdem ein Tongenerator an Port 2, Klemme 1 und über einen Treibertransistor ein Relais an Port 2, Klemme 6 (z. B. KOSMOS-Schaltrelais KOSMODYNE B).
Mit dem Relais kann z. B. ein handelsüblicher Türöffner geschaltet werden; er ist in unserem Bild 73 nicht eingezeichnet.
Dann werden drei Zahlen in die Speicherzellen 101, 102 und 103 eingegeben, als „Codierung", wie oft die Tasten gedrückt werden müssen, damit das Relais schaltet und so das Schloß geöffnet wird.
Gibt man in die Speicherzellen z. B. folgende Werte ein

00004 in Zelle 101
00001 in Zelle 102
00002 in Zelle 103

so muß zum Öffnen des Schlosses Taste 1 viermal, dann Taste 2 einmal, Taste 3 zweimal und zum Schluß (gleichgültig wie die Tasten „codiert" sind) stets noch einmal Taste 1 betätigt werden.
Bei einem fehlerhaften Tastendruck gibt der Tongenerator einen kurzen, bei mehr als insgesamt sieben Fehlern einen langen Ton ab (Alarm).
Diese computergesteuerte Alarmanlage hat gegenüber „klassischen" Systemen den enormen Vorteil, daß die Codierung absolut mühelos geändert werden kann. Da außerdem die Anzahl der zulässigen Tastendrücke (bis 255 je Taste!) beliebig auf die

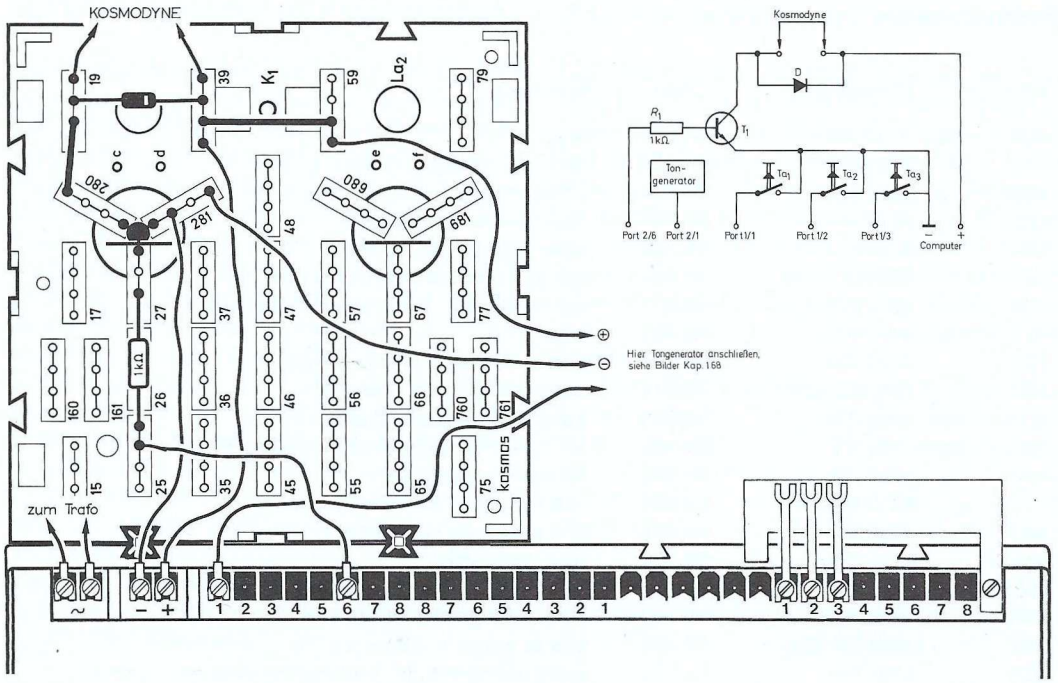
Listing 37: **Codeschloß mit Alarmanlage**

Adresse	Mnemonics	Code	Kommentar
001	AKO 255	**04.255**	lade „255"
002	ABS 105	**06.105**	speichere als Vergleichszahl
003	AKO 254	**04.254**	lade „254"
004	ABS 106	**06.106**	speichere als Vergleichszahl für T1
005	AKO 253	**04.253**	lade „253"
006	ABS 107	**06.107**	speichere als Vergleichszahl für T2
007	AKO 251	**04.251**	lade „251"
008	ABS 108	**06.108**	speichere als Vergleichszahl für T3
009	AKO 007	**04.007**	lade „7"
010	ABS 109	**06.109**	speichere als Maximalfehlerzahl
011	S1: AKO 000	**04.000**	lade „0"
012	ABS 110	**06.110**	speichere als Fehlerzahl (aktuell)
013	S2: ABS 111	**06.111**	als Zähler für T1
014	ABS 112	**06.112**	als Zähler für T2
015	ABS 113	**06.113**	als Zähler für T3
016	P2A 000	**18.000**	gib die „0" an Port 2 aus: alles abschalten
017	AKO 027	**04.027**	lade „027" als Rücksprungadresse
018	U1: ABS 114	**06.114**	speichere Akku-Inhalt in Adreßzelle
019	R1: P1E 000	**16.000**	hier Port 1 (Tasteninformation)
020	VGL 105	**10.105**	ist der Wert „255", also nichts gedrückt?
021	SPB 019 (R1)	**11.019**	falls ja, springe zurück zum Lesen
022	ABS 104	**06.104**	sonst speichere den Wert in 104
023	R2: P1E 000	**16.000**	lies wieder Port 1
024	VKL 105	**13.105**	ist der Wert noch kleiner als „255", also Taste noch gedrückt?
025	SPB 023 (R2)	**11.023**	falls ja, weiterhin einlesen
026	SIU 114	**21.114**	sonst springe zurück wie in 144 gespeichert
027	LDA 104	**05.104**	die Taste ist wieder losgelassen – lade den Wert wieder
028	VKL 106	**13.106**	ist der Wert kleiner als „254"?
029	SPB 034 (A1)	**11.034**	falls ja, ist nicht T1 gedrückt
030	AKO 001	**04.001**	sonst lade „1"
031	ADD 111	**07.111**	addiere dazu den T1-Zähler
032	ABS 111	**06.111**	und speichere ihn wieder
033	SPU 019 (R1)	**09.019**	Rücksprung zum Lesen des Ports
034	A1: LDA 111	**05.111**	lade den T1-Zähler
035	VGL 101	**10.101**	ist er gleich dem Sollwert?
036	SPB 038 (A2)	**11.038**	falls ja, springe zur nächsten Stufe
037	SPU 069 (F1)	**09.069**	sonst springe in die Fehlerroutine
038	A2: LDA 104	**05.104**	lade die Tasteninformation wieder
039	VKL 107	**13.107**	ist der Wert kleiner als „253", also T2 nicht gedrückt?
040	SPB 045 (B1)	**11.045**	falls ja, springe zum Sollwert-Vergleich
041	AKO 001	**04.001**	sonst lade „1"
042	ADD 112	**07.112**	addiere dazu den T2-Zähler
043	ABS 112	**06.112**	und speichere ihn wieder
044	SPU 019 (R1)	**09.019**	springe zurück zum Lesen des Ports
045	B1: LDA 112	**05.112**	lade den T2-Zähler
046	VGL 102	**10.102**	ist er gleich dem Sollwert?
047	SPB 049 (B2)	**11.049**	falls ja, springe zur dritten Stufe
048	SPU 069 (F1)	**09.069**	sonst springe in die Fehlerroutine
049	B2: LDA 104	**05.104**	lade die Tastenformation wieder
050	VGL 108	**10.108**	ist der Wert „251", also T3 gedrückt?
051	SPB 053 (C1)	**11.053**	falls ja, springe zum Zähler

Fortsetzung siehe nächste Seite

Fortsetzung Listing 37:

Adresse		Mnemonics	Code	Kommentar
052		SPU 058 (E1)	**09.058**	sonst springe zum Sollwert-Vergleich
053	C1:	AKO 001	**04.001**	lade „1"
054		ADD 113	**07.113**	addiere dazu den T3-Zähler
055		ABS 113	**06.113**	speichere ihn wieder
056		AKO 049	**04.049**	lade Rücksprungadresse „049" in den Akku
057		SPU 018 (R1)	**09.018**	springe zurück zum Lesen des Ports
058	E1:	LDA 113	**05.113**	lade den T3-Zähler
059		VGL 103	**10.103**	ist der Wert gleich dem Sollwert?
060		SPB 062 (E2)	**11.062**	falls ja, springe zur Endstufe
061		SPU 069 (F1)	**09.069**	sonst springe in die Fehlerroutine
062	E2:	LDA 104	**05.104**	lade die Tasteninformation wieder
063		VKL 106	**13.106**	ist der Wert kleiner als „254", also T1 nicht gedrückt?
064		SPB 069 (F1)	**11.069**	falls ja, springe in die Fehlerroutine
065		AKO 001	**04.001**	sonst lade „1"
066		P2A 006	**18.006**	gib sie an Port 2 Klemme 6 (Relais) aus
067		VZG 250	**03.250**	warte eine Viertelsekunde
068		SPU 011 (S1)	**09.011**	Rücksprung zum Neuanfang
069	F1:	AKO 001	**04.001**	Fehlerroutine – lade „1"
070		ABS 100	**06.100**	speichere sie in 100
071		P2A 001	**18.001**	gib sie am Summer (Port 2 Klemme 1) aus
072		VZG 250	**03.250**	warte insgesamt eine halbe Sekunde
073		VZG 250	**03.250**	
074		ADD 110	**07.110**	addiere die bisherige Fehlerzahl zu dieser „1"
075		ABS 110	**06.110**	speichere als neue Fehlerzahl
076		ANZ	**02.000**	zeige sie auch an
077		VKL 109	**13.109**	ist der Wert noch kleiner als die Maximalzahl?
078		AKO 000	**04.000**	lade „0"
079		SPB 013 (S2)	**11.013**	falls ja, ein neuer Versuch gestattet
080	F2:	ADD 100	**07.100**	sonst addiere „1" zum Akkuinhalt
081		VZG 040	**03.040**	warte 40 msec.
082		VKL 106	**13.106**	ist „254" noch nicht erreicht?
083		SPB 080 (F2)	**11.080**	falls ja, weiterhin warten und tönen
084		SPU 011 (S1)	**09.011**	nach insgesamt 10 Sekunden: Neuanfang
100			**00.001**	Schrittweite
101		*	**00.XXX**	Sollzahl für T1
102		*	**00.XXX**	Sollzahl für T2
103		*	**00.XXX**	Sollzahl für T3
104			**–**	Tasteninformation an Port 1
105			**00.255**	Vergleichszahlen für Tastenbetätigung
106			**00.254**	
107			**00.253**	
108			**00.251**	
109			**00.007**	Maximalfehlerzahl
110			**–**	aktuelle Fehlerzahl
111			**–**	Zähler für T1
112			**–**	Zähler für T2
113			**–**	Zähler für T3
114			**–**	Rücksprungadreßzelle



drei Tasten verteilt werden, ergibt sich eine unübersehbare Zahl von Kombinationsmöglichkeiten.

Aufbau und Schaltung der gesamten Anlage sind den Bildern 72 und 73 zu entnehmen.

Bild 72 und 73. Codeschloß mit Alarmanlage, aufgebaut aus KOSMOS „Radio + Elektronik 100" und Hobby-Set „Alarmanlage"

Listing 38: **Doppelwürfel mit Paschanzeige** (siehe dazu Text S. 97)

Adresse		Mnemonics	**Code**	Kommentar
001	A0:	AKO 006	**04.006**	lade „6" . . .
002		ABS 102	**06.102**	als Grenze für jeden Würfel
003		AKO 001	**04.001**	lade „1" und speichere sie . . .
004		ABS 101	**06.101**	als Schrittweite,
005		ABS 103	**06.103**	als Startwert des 1. Würfels und . . .
006		ABS 104	**06.104**	als Startwert des 2. Würfels
007		AKO 000	**04.000**	lade „0". . .
008		ABS 100	**06.100**	als Vergleichszahl . . .
009		ABS 105	**06.105**	und in der Anzeigezelle speichern
010		ANZ	**02.000**	zeige, daß das Programm läuft
011	A1:	P1E 001	**16.001**	lies Port 1 Klemme 1: Tasteninformation
012		VGL 100	**10.100**	ist sie „0", also gedrückt?
013		SPB 030 (B1)	**11.030**	falls ja, bereite die Anzeige vor bei 030
014		LDA 103	**05.103**	sonst lade den ersten Würfel
015		VGL 102	**10.102**	ist er schon bei „6"?
016		SPB 020 (A2)	**11.020**	falls ja, springe zu 020
017		ADD 101	**07.101**	sonst addiere noch „1" dazu
018		ABS 103	**06.103**	speichere wieder als neuen Würfelstand
019		SPU 011 (A1)	**09.011**	dann zähle weiter bei 011

Fortsetzung siehe nächste Seite

Fortsetzung Listing 38:

Adresse		Mnemonics	Code	Kommentar
020	A2:	AKO 001	**04.001**	lade „1“. . .
021		ABS 103	**06.103**	und setze so den 1. Würfel zurück
022		ADD 104	**07.104**	addiere den 2. Zähler zu dieser „1“. . .
023		ABS 104	**06.104**	und speichere ihn wieder
024		VGR 102	**12.102**	ist er jetzt größer als „6“?
025		SPB 027 (A3)	**11.027**	falls ja, setze ihn auch zurück bei 027
026		SPU 011 (A1)	**09.011**	sonst „würfele“ den ersten weiter bei 011
027	A3:	AKO 001	**04.001**	lade „1“. . .
028		ABS 104	**06.104**	und setze den 2. Würfel zurück
029		SPU 011 (A1)	**09.011**	weiterzählen beim 1. Würfel
030	B1:	AKO 010	**04.010**	Taste war gedrückt! Lade „10“
031	B2:	ABS 106	**06.106**	als Zähler für Multiplikation speichern
032		LDA 105	**05.105**	lade die Anzeigezelle
033		ADD 104	**07.104**	addiere den 2. Würfelwert dazu . . .
034		ABS 105	**06.105**	und speichere ihn wieder
035		LDA 106	**05.106**	lade den Zähler
036		SUB 101	**08.101**	subtrahiere „1“ davon
037		VGR 100	**12.100**	ist er noch größer als „0“?
038		SPB 031 (B2)	**11.031**	falls ja, weiter addieren bei 031
039		LDA 105	**05.105**	sonst wurde mit „10“ multipliziert; lade Anzeigezelle . . .
040		ADD 103	**07.103**	und zähle den 1. Würfel hinzu
041		ABS 105	**06.105**	speichere den anzeigbaren Wert in 105 . . .
042		ANZ	**02.000**	und zeige ihn an
043		LDA 103	**05.103**	lade den 1. Zähler (1. Würfel)
044		VGL 104	**10.104**	ist er gleich dem 2. Zähler (2. Würfel)?
045		SPB 052 (C2)	**11.052**	falls ja, springe zum Blinken
046		AKO 000	**04.000**	sonst lade „0“ als Verzögerungszählerstart
047	C1:	VZG 250	**03.250**	Verzögerungsschleife: 6 × 250 ms
048		ADD 101	**07.101**	erhöhe um „1“
049		VKL 102	**13.102**	noch kleiner als „6“?
050		SPB 047 (C1)	**11.047**	falls ja, weiter verzögern
051		SPU 001 (A0)	**09.001**	fertig – neuer Würfelstart am Programmanfang
052	C2:	AKO 012	**04.012**	Blinken erwünscht! Lade „12“
053	C3:	ABS 106	**06.106**	in der Zählerzelle speichern
054		VZG 250	**03.250**	warte 1/4 Sekunde
055		AKO 100	**04.100**	lade „100“
056		ANZ	**02.000**	zeige sie an
057		VZG 025	**03.025**	aber nur kurz
058		LDA 105	**05.105**	lade den erwürfelten Wert
059		ANZ	**02.000**	zeige ihn an
060		LDA 106	**05.106**	lade den Zähler
061		SUB 101	**08.101**	verringere ihn um „1“
062		VGR 100	**12.100**	ist er noch größer als „0“?
063		SPB 053 (C3)	**11.053**	falls ja, weiterblinken
064		SPU 001 (A0)	**09.001**	sonst fertig – neuer Start am Programmanfang
100			**00.000**	„0“ als Vergleichszahl
101			**00.001**	„1“ als Schrittweite
102			**00.006**	„6“ als Grenze des Zählers
103			–	1. Zähler (1. Würfel)
104			–	2. Zähler (2. Würfel)
105			–	Doppelwürfelanzeige
106			–	Hilfszelle

2.11 Doppelwürfel mit Paschanzeige

Das Programm simuliert *zwei* Würfel. Das Ergebnis wird so angezeigt, daß der eine in der Zehnerstelle der Anzeige auftaucht, der andere in der Einerstelle. Außerdem blinkt die Anzeige, wenn ein Pasch, also beispielsweise „11", „erwürfelt" wurde. Und wie würfeln wir? Diesmal nicht mit der STP-Taste, sondern über unsere externe Taste zwischen Port 1, Klemme 1 und 0 V. Ganz einfach ist auch das Programm: Es läuft wie beim simplen Würfel ein Zähler ständig durch, der beim Erreichen der „7" eben nicht nur auf „1" zurücksetzt, sondern gleichzeitig auch einen zweiten Zähler (den zweiten Würfel) um „1" erhöht, sozusagen einen Übertrag auf den zweiten Würfel durchführt. Und wenn dann der zweite Würfel ebenfalls die „7" erreicht, wird auch er auf „1" zurückgesetzt. Sobald der Kontakt geschlossen wird, springt das Programm in eine Anzeigeroutine, so daß die momentanen Würfelwerte für die Anzeige aufbereitet und sichtbar gemacht werden; dann startet der Zähler automatisch wieder.

2.12 Computersimulierte Mondlandung

Das nachfolgende Programm simuliert die Landung einer Rakete auf dem Mond. Es berechnet zu jedem Zeitpunkt die Höhe, die Geschwindigkeit und den Treibstoffvorrat des Raumschiffs. Und der Mensch vor dem Computer ist dann der Astronaut, der darüber zu entscheiden hat, wieviel Treibstoff er in jeder Sekunde durch das Triebwerk pfeifen lassen will – je mehr, desto stärker wird gebremst; ist es zuviel, kehrt sich sogar das Sinken in ein Steigen um, und die Rakete fliegt wieder vom Mond weg!

Die Wahl der Treibstoffmenge erfolgt über einen Kontakt zwischen 0 V und einer Klemme des Ports 1. Wird Klemme 1 mit 0 V verbunden, so sieht das Programm dieses als Verbrauch von 0 Treibstoffeinheiten in der nächsten Sekunde an, bei Klemme 2 nimmt es 1 Treibstoffeinheit, bei Klemme 3 dann 2 usw. bis zu Klemme 8, wo 7 Treibstoffeinheiten in der nächsten Sekunde verbrannt werden sollen. Dort ist natürlich auch die Bremskraft am größten. Man geht bei der Berechnung davon

Listing 39: **Computersimulierte Mondlandung**

Adresse		Mnemonics	**Code**	Kommentar
001		AKO 100	**04.100**	Treibstoffmenge am Anfang: 100 Einheiten . . .
002		ABS 105	**06.105**	in der dafür vorgesehenen Speicherzelle speichern
003		AKO 127	**04.127**	Starthöhe: 127 m . . .
004		ABS 106	**06.106**	speichern
005		AKO 101	**04.101**	Startgeschwindigkeit: 1 m/s fallend . . .
006		ABS 108	**06.108**	speichern
007		AKO 001	**04.001**	lade „1" als erste Vergleichszahl
008		ABS 120	**06.120**	speichere sie in der Hilfszelle
009		AKO 111	**04.111**	lade die Startadresse „111" für die Vergleichszahlen . . .
010		ABS 119	**06.119**	und speichere sie
011	I1:	ABS 110	**06.110**	speichere Akkuinhalt als aktuelle Adresse
012		LDA 120	**05.120**	lade die Hilfszelle . . .
013		AIS 110	**20.110**	und speichere sie unter der Adresse in 110
014		VGL 121	**10.121**	ist „128" schon erreicht?
015		SPB 082 (H3)	**11.082**	falls ja, sind alle Zellen 111 – 118 richtig belegt
016		ADD 120	**07.120**	sonst verdopple den Akkuinhalt . . .
017		ABS 120	**06.120**	und speichere ihn in der Hilfszelle
018		LDA 110	**05.110**	dann lade die Adreßzelle
019		ADD 101	**07.101**	erhöhe die aktuelle Adresse um „1" . . .
020		SPU 011 (I1)	**09.011**	und gehe weiter in der Schleife
021	R0:	LDA 105	**05.105**	lade die Treibstoffmenge . . .
022		ANZ	**02.000**	und zeige sie an
023		AKO 025 (R1)	**04.025**	lade „25" als 1. Rücksprungadresse . . .

Fortsetzung siehe nächste Seite

Fortsetzung Listing 39:

Adresse		Mnemonics	Code	Kommentar
024		SPU 032 (U1)	**09.032**	und springe in die Einleseroutine
025	R1:	LDA 106	**05.106**	dann lade die Höhe
026		ANZ	**02.000**	zeige sie an
027		AKO 029 (R2)	**04.029**	lade „29" als 2. Rücksprungadresse . . .
028		SPU 032 (U1)	**09.032**	für die Einleseroutine
029	R2:	LDA 107	**05.107**	dann die Geschwindigkeit . . .
030		ANZ	**02.000**	anzeigen . . .
031		AKO 021 (R0)	**04.021**	und den Anzeigeschleifen-Anfang als Rücksprungadresse
032	U1:	ABS 122	**06.122**	Beginn der Einleseroutine: Rücksprung speichern
033		VZG 250	**03.250**	warte eine dreiviertel Sekunde
034		VZG 250	**03.250**	
035		VZG 250	**03.250**	
036		P1E 000	**16.000**	lies Information an Port 1 in den Akku
037		VKL 104	**13.104**	ist eine Leitung gesetzt, also Eingabe erfolgt?
038		SPB 041 (F1)	**11.041**	falls ja, mache weiter bei der Berechnung
039		VZG 250	**03.250**	warte anderenfalls
040		SIU 122 (RX)	**21.122**	springe zurück zu derjenigen Adresse, die bei 032 in 122 gespeichert wurde
041	F1:	ABS 109	**06.109**	speichere das Eingelesene
042		LDA 104	**05.104**	lade stattdessen „255". . .
043		SUB 109	**08.109**	und subtrahiere das Eingelesene davon
044		ABS 109	**06.109**	jetzt liegt das Gelesene als Rest zu „255" vor
045		LDA 119	**05.119**	lade Startadresse für Vergleichsserie
046	F2:	ABS 110	**06.110**	speichere die aktuelle Vergleichsadresse
047		LIA 110	**19.110**	lade den Vergleichswert, der über 110 zu finden ist
048		VGL 109	**10.109**	ist er so groß wie das Eingelesene?
049		LDA 110	**05.110**	lade die Adresse des Vergleichs
050		SPB 053 (F3)	**11.053**	falls beim vorletzten Befehl Gleichheit vorlag, ist die richtige Adresse gefunden
051		ADD 101	**07.101**	erhöhe die aktuelle Adresse um „1" . . .
052		SPU 046 (F2)	**09.046**	und mache weiter in der Schleife
053	F3:	SUB 119	**08.119**	gefunden: subtrahiere die Startadresse von der aktuellen . . .
054		ABS 109	**06.109**	und nimm dies als effektiven Treibstoffverbrauch (0 – 7)
055		LDA 105	**05.105**	lade die Treibstoffmenge
056		VKL 109	**13.109**	ist sie kleiner als der gewünschte Verbrauch?
057		SPB 094 (E1)	**11.094**	falls er nicht reicht, springe zur Ende-Routine
058		SUB 109	**08.109**	sonst subtrahiere den gewünschten Verbrauch vom Vorrat
059		ABS 105	**06.105**	speichere den neuen Rest Treibstoff
060	H1:	AKO 104	**04.104**	lade „104" als Grundhöhenänderung durch Mond-anziehung
061		ADD 108	**07.108**	addiere die alte Geschwindigkeit dazu
062		SUB 103	**08.103**	subtrahiere „100"
063		ADD 108	**07.108**	das Ganze . . .
064		SUB 103	**08.103**	nochmal . . .
065		SUB 109	**08.109**	subtrahiere den Treibstoffverbrauch und . . .
066		SUB 109	**08.109**	nochmal
067		VKL 103	**13.103**	ist der Akkuinhalt kleiner als „100", die Rakete also im Steigen begriffen?
068		SPB 077 (H2)	**11.077**	falls ja, springe zur Höhenaddition
069		SUB 103	**08.103**	sonst subtrahiere „100", um auf reales Sinken zu kommen

Fortsetzung siehe nächste Seite

Fortsetzung Listing 39:

070		VGR 106	**12.106**	ist der Sinkbetrag größer als die letzte Höhe?
071		SPB 097 (E2)	**11.097**	falls ja, ist der Boden erreicht; springe zum Ende
072		ABS 120	**06.120**	sonst in der Hilfszelle zwischenspeichern
073		LDA 106	**05.106**	lade stattdessen die Höhe . . .
074		SUB 120	**08.120**	und subtrahiere den Sinkbetrag
075		ABS 106	**06.106**	dann speichere die Höhe wieder . . .
076		SPU 082 (H3)	**09.082**	und springe zur neuen Geschwindigkeitsberechnung
077	H2:	ABS 120	**06.120**	speichere das Ansteigen mal in der Hilfszelle
078		AKO 100	**04.100**	lade „100“ . . .
079		SUB 120	**08.120**	und subtrahiere die Hilfszelle davon: reiner Steigbetrag . . .
080		ADD 106	**07.106**	zu dem die alte Flughöhe addiert wird
081		ABS 106	**06.106**	dies ist die neue Höhe
082	H3:	LDA 108	**05.108**	lade die Geschwindigkeit (interne Darstellung)
083		ADD 102	**07.102**	addiere „2“ als Steigerung durch die Mondanziehung . . .
084		SUB 109	**08.109**	und subtrahiere den durch das Triebwerk verursachten Betrag
085		ABS 108	**06.108**	speichere die neue Geschwindigkeit
086		ABS 107	**06.107**	auch für die Anzeige
087		VGR 103	**12.103**	ist der Wert größer als „100“, also sinken?
088		SPB 021 (R0)	**11.021**	falls ja, springe zur Anzeige und neuen Eingabe
089		AKO 200	**04.200**	sonst steigt die Rakete – Aufbereitung der Geschwindigkeitsanzeige:
090		SUB 107	**08.107**	subtrahiere die interne Darstellung von „200“ . . .
091		ADD 103	**07.103**	und addiere „100“ dazu: Steigen hat jetzt . . .
092		ABS 107	**06.107**	„2“ als Hunderterstelle, Sinken „1“
093		SPU 021 (R0)	**09.021**	dann springe zur Anzeige und Eingabe
094	E1:	AKO 000	**04.000**	wir haben keinen Treibstoff mehr!
095		ABS 109	**06.109**	der Verbrauch ist deshalb auch null
096		SPU 060 (H1)	**09.060**	Weiter zur Berechnung
097	E2:	LDA 107	**05.107**	wir sind unten! Lade die Geschwindigkeit . . .
098	E3:	ANZ	**02.000**	und zeige sie an
099		SPU 098 (E3)	**09.098**	immer weiter so!
100		*	**00.000**	„0“
101		*	**00.001**	„1“
102		*	**00.002**	„2“ } Vergleichszahlen
103		*	**00.100**	„100“
104		*	**00.255**	„255“
105			–	Treibstoffvorrat
106			–	Flughöhe
107			–	Geschwindigkeit (für Anzeige aufbereitet)
108			–	Geschwindigkeit (intern)
109		*	**00.000**	Treibstoffverbrauch wie an Port 1 gelesen
110			–	Adresszelle zur Einlesedecodierung
111			**00.001**	„1“
112			**00.002**	„2“
113			**00.004**	„4“
114			**00.008**	„8“ } Vergleichszahlen zur Eingabedecodierung
115			**00.016**	„16“
116			**00.032**	„32“
117			**00.064**	„64“
118			**00.128**	„128“
119			**00.111**	„111“ als Startadresse dieses Vergleichszahlenbereichs
120			–	Hilfszelle während der Rechnung
121		*	**00.128**	Vergleichszahl „128“
122			–	Rücksprungadresse für Einleseroutine

aus, daß die Bremskraft dem Treibstoffverbrauch gerade entspricht. Wird also nichts verbraucht, gibt es nur den freien Fall zur Mondoberfläche hin!
Nach der Eingabe über eine der Klemmen zeigt uns das Programm den verbliebenen Treibstoff, die momentane Höhe und Geschwindigkeit an; dies geht immer so weiter, bis die nächste Eingabe gemacht wird. Der Treibstoffvorrat ist aber begrenzt!
Noch eine Bemerkung zur Anzeige der Geschwindigkeit: Ist sie auf den Mond **zu**gerichtet, so wird in der Hunderterstelle eine 1 gezeigt, weist sie vom Mond **weg,** dagegen eine 2.
Die Berechnungsweise ist wie folgt:
Ist die Höhe vor der nächsten Zeiteinheit H und danach H′, so besteht zwischen diesen der Zusammenhang:

$$H' = H - v \cdot t - \tfrac{1}{2} \cdot a \cdot t^2 + \tfrac{1}{2} \cdot b \cdot t^2$$

v ist hierin die momentane Geschwindigkeit, a die Mondanziehung (im Programm mit $2\,m/s^2$ angesetzt, in Wirklichkeit etwas geringer), und b die Beschleunigung vom Mond weg durch das Triebwerk (also der Treibstoffverbrauch in der Sekunde). Die Geschwindigkeit selbst ändert sich natürlich auch von v (vorher) zu v′ (nachher):

$$v' = v + a \cdot t - b \cdot t$$

Im Programm wird jetzt nach jeder Eingabe von b über Port 1 der Zustand (H′ und v′) 2 Sekunden später (also für t = 2) berechnet und angezeigt. Am Programmende schließlich – wenn die Mondoberfläche erreicht ist – bleibt die Anzeige mit der Landegeschwindigkeit stehen. Ist der Wert größer als 2 (Anzeige also 102), ging das Raumschiff dabei wohl zu Bruch!

2.13 Computer als Melodiengenerator

Wir können in unseren Computer ein Programm eingeben, das mit Hilfe von ein paar zusätzlichen elektronischen Bauteilen verschiedene Melodien erzeugt. Das Programm ist sehr kurz, so daß im Speicher noch viel Platz für verschlüsselte Melodien bleibt. Es läßt einen Zähler laufen und benutzt diesen dazu, sich nacheinander die Inhalte von Datenzellen in den Akku zu laden und dann an Port 2 auszugeben. Entsprechend dem ausgegebenen Wert werden unterschiedliche Töne angesteuert. Die Klemmen von Port 2 sind also bestimmten Tonhöhen zugeordnet. Das geht mit folgender Tabelle:

Tonhöhe	c	d	e	f	g	a	h	c′
Klemme	1	2	3	4	5	6	7	8
Code	1	2	4	8	16	32	64	128

Den Code zur Ansteuerung der verschiedenen Klemmen müssen wir im Datenbereich („Musikspeicher") eingeben; um z.B. den Ton „a" zu erzeugen, muß also 00.032 gespeichert werden.
Das Programm weiß natürlich nicht von selbst, wie lange jeder einzelne Ton dauern soll. Deshalb speichern wir die Tonlänge immer unmittelbar nach der Tonhöhe und zwar in Einheiten von Achtelnoten. Für eine Viertelnote ist also 00.002, für drei Achtel 00.003 usw. zu speichern. Als Beispiel sei hier das Motiv „Stille Nacht, Heilige Nacht" verschlüsselt:

Adresse	Datum	Bedeutung
033	00.016	Tonhöhe: g
034	00.006	Länge: 6/8
035	00.032	Tonhöhe: a
036	00.002	Länge: 2/8
037	00.016	Tonhöhe: g
038	00.004	Länge: 4/8 = 1/2
039	00.004	Tonhöhe: e
040	00.012	Länge: 12/8 = 3/2
041	00.000	Tonhöhe: gar kein Ton (Pause)
042	00.008	Dauer der Pause: 8/8 = 1/1
043	00.000	Tonhöhe: gar kein Ton (Musik aus)
044	00.000	Länge: 0 – Ende der Melodie

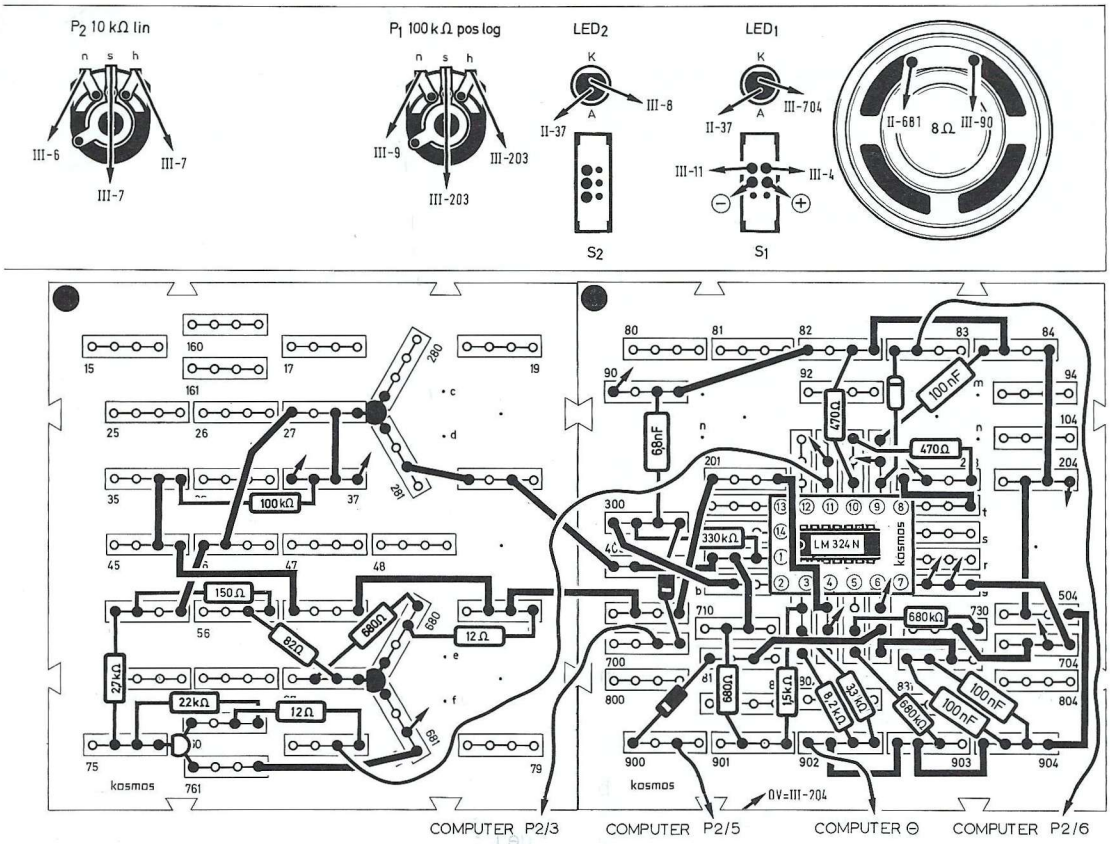


Bild 74. Computer als Melodiengenerator, aufgebaut aus KOSMOS „Elektronik-Labor E 200" (Schaltplan dazu Bild 75)

Die Pausen zwischen den Tönen werden als „Tonhöhe 0" (= kein Ton) codiert, die Länge der Pause entspricht dann der Dauer dieses „nicht vorhandenen" Tones. Für das Ende der Melodie wird 00.000 sowohl für die Tonhöhe als auch für die Tonlänge eingegeben.

Eine „1" in Zelle 031 bewirkt, daß die Melodie ständig wiederholt wird, bei einer „0" läuft sie dagegen nur einmal ab; sie wird aber nicht gelöscht, so daß sie noch mehrmals neu gestartet werden kann. Die automatische Wiederholung ist natürlich besonders beim Tonhöhenabgleich nützlich, um mit den Potentiometern die Tonoszillatoren exakt einzustellen. Bild 75 zeigt eine Schaltung mit drei Oszillatoren; das genügt auch zum Abspielen der beiden angegebenen Melodien, die mit den Tönen e, g und a auskommen. Mit mehr Oszillatoren kann man selbstverständlich die ganze weiter oben codierte Oktave hörbar machen.

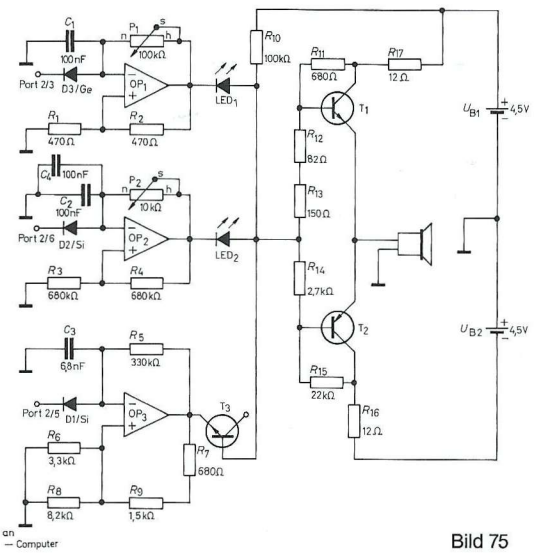


Bild 75

Hier sei noch eine längere Melodie mit den Tönen e, g und a angegeben. Es handelt sich um „If I had a hammer" von Lee Hays und Pete Seeger. Sie paßt auch noch gut in den Speicher hinter der Testmelodie:

Adresse	Datum	Adresse	Datum
045	00.004	055	00.016
046	00.001	056	00.004
047	00.016	057	00.000
048	00.002	058	00.002
049	00.016	059	00.032
050	00.001	060	00.002
051	00.032	061	00.016
052	00.001	062	00.001
053	00.016	063	00.004
054	00.004	064	00.001
065	00.016	082	00.001
066	00.001	083	00.016
067	00.016	084	00.004
068	00.001	085	00.016
069	00.004	086	00.002
070	00.004	087	00.032
071	00.004	088	00.002
072	00.002	089	00.032
073	00.000	090	00.001
074	00.002	091	00.032
075	00.004	092	00.001
076	00.002	093	00.032
077	00.016	094	00.002
078	00.001	095	00.016
079	00.016	096	00.006
080	00.001	097	00.000
081	00.032	098	00.000

Listing 40: **Computer als Melodiengenerator**

Adresse		Mnemonics	**Code**	Kommentar
001		AKO 001	**04.001**	lade „1" . . .
002		ABS 029	**06.029**	und speichere sie als Schrittweite und Vergleichszahl
003	A1:	AKO 000	**04.000**	lade „0" . . .
004		ABS 028	**06.028**	und speichere sie als Vergleichszahl
005		LDA 032	**05.032**	lade die Startadresse der Melodie . . .
006	A2:	ABS 030	**06.030**	und speichere sie in der Zählerzelle
007		LIA 030	**19.030**	lade indirekt die Tonhöhe . . .
008		P2A 000	**18.000**	und gib ihn aus: Ton wird eingeschaltet!
009		LDA 030	**05.030**	dann lade den Zähler
010		ADD 029	**07.029**	erhöhe ihn um „1" . . .
011		ABS 030	**06.030**	und speichere ihn wieder
012		LIA 030	**19.030**	lade so indirekt die Tonlänge
013		VGL 028	**10.028**	ist sie „0"?
014		SPB 024 (A4)	**11.024**	falls ja, ist die Melodie am Ende
015	A3:	VZG 100	**03.100**	sonst warte 100 ms
016		SUB 029	**08.029**	erniedrige die verbliebene Dauer um „1"
017		VGR 028	**12.028**	ist sie noch größer als „0"?
018		SPB 015 (A3)	**11.015**	falls ja, dehne den Ton noch weiter
019		P2A 000	**18.000**	sonst gib die „0" aus
020		VZG 020	**03.020**	kurze Pause zum Töne-Trennen
021		LDA 030	**05.030**	lade den Zähler
022		ADD 029	**07.029**	erhöhe ihn um „1" . . .
023		SPU 006 (A2)	**09.006**	und mache beim nächsten Ton weiter
024	A4:	LDA 031	**05.031**	Ende – lade die Wiederholungszelle
025		VGL 029	**10.029**	ist sie „1"?
026		SPB 003 (A1)	**11.003**	falls ja, fange wieder von vorne an
027		HLT	**01.000**	sonst HALT
028			**00.000**	„0" als Vergleichszahl
029			**00.001**	„1" als Schrittweite
030			–	Zähler für Melodie
031		*	**00.00x**	Wiederholung: ja = „1", nein = „0"
032		*	**00.xxx**	Startadresse der Melodie

2.14 Strategie am Schachbrett

Dieses reizvolle Spiel beruht auf den Bewegungen der Dame eines Schachspiels. Hier ist es aber der Dame nur gestattet, sich nach rechts, nach unten oder diagonal nach rechts zu bewegen. Im Spiel gegen den Computer müssen Sie versuchen, als erster die Dame in das Feld 158 (in der Ecke rechts unten) zu bringen. Die Züge wechseln zwischen Ihnen und dem Computer ab. Sie beginnen, indem Sie die Dame auf irgendein Feld der obersten Reihe oder der linken Spalte setzen (Startzone). Das ist Ihr erster Zug. Den nächsten Zug macht dann der Computer usw.

Das Spielbrett ist wie folgt numeriert:

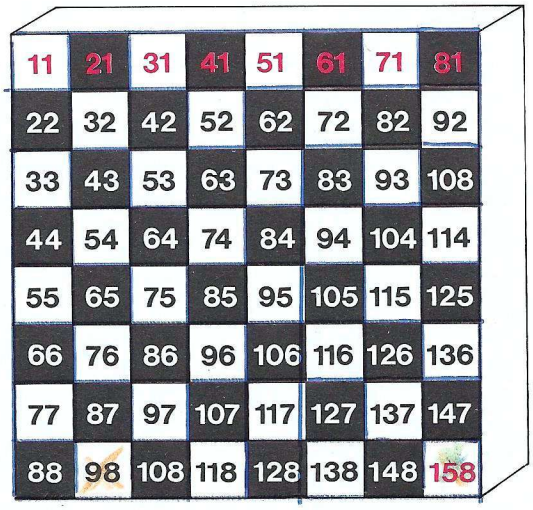


Bild 76

Geben Sie für Ihren ersten Zug in Zelle 100 die Nummer desjenigen Feldes aus der Startzone ein, auf dem Sie starten wollen. Setzen Sie den Programmzähler auf 001 und drücken Sie die RUN-Taste. In der Anzeige erscheint die Feldnummer, auf die der Rechner zieht. Stoppen Sie die Anzeigeschleife und geben Sie die neue Position Ihrer Wahl in Zelle 100 ein und starten Sie erneut bei 001. Gelingt es Ihnen, als erste(r) das Zielfeld zu erreichen, so können Sie beobachten, wie sich der Rechner „ärgert"; andernfalls zeigt er zum Zeichen seines Sieges das Gewinnerfeld, also 158, als Ziel seines Zuges an. Das Programm überprüft nicht, ob auch nach den Spielregeln gezogen wurde! Nur soviel ist sicher: Der Computer spielt ehrlich!

Listing 41: **Strategie am Schachbrett**

Adresse		Mnemonics	Code	Kommentar
001		AKO 005 (R1)	**04.005**	1. Rücksprungadresse laden . . .
002		ABS 116	**06.116**	und in 116 speichern
003		LDA 100	**05.100**	Spielerposition laden . . .
004		SPU 061 (U1)	**09.061**	und im Unterprogramm testen
005	R1:	VGL 105	**10.105**	ist die Position gut?
006		SPB 045 (A1)	**11.045**	falls ja, mache einen Verlegenheitszug
007		AKO 070	**04.070**	lade „70" als Zählergrundwert (Spalten) . . .
008		ABS 103	**06.103**	und speichere in 103
009		AKO 007	**04.007**	lade „7" als Grundwert (Reihen)
010		ABS 102	**06.102**	in 102 speichern
011	L0:	AKO 017 (R2)	**04.017**	2. Rücksprungadresse laden . . .
012		ABS 116	**06.116**	und in 116 speichern
013		LDA 103	**05.103**	lade den Zählergrundwert . . .
014		ADD 100	**07.100**	und addiere ihn zur Spielerposition
015		ABS 101	**06.101**	dies ist die neue Computerposition (gleiche Reihe)
016		SPU 061 (U1)	**09.061**	teste sie im Unterprogramm
017	R2:	VGL 105	**10.105**	ist sie gut?
018		SPB 058 (A4)	**11.058**	falls ja, zeige sie an
019		AKO 025 (R3)	**04.025**	sonst lade 3. Rücksprungadresse . . .
020		ABS 116	**06.116**	und in 116 speichern
021		LDA 102	**05.102**	lade Reihengrundwert . . .
022		ADD 101	**07.101**	und addiere die Computerposition (gleiche Spalte)
023		ABS 101	**06.101**	wieder speichern . . .
024		SPU 061 (U1)	**09.061**	und testen der neuen Position

Fortsetzung siehe nächste Seite

Fortsetzung Listing 41:

Adresse		Mnemonics	Code	Kommentar
025	R3:	VGL 105	**10.105**	ist sie gut?
026		SPB 058 (A4)	**11.058**	falls ja, anzeigen lassen . . .
027		AKO 035 (R4)	**04.035**	4. Rücksprungadresse laden . . .
028		ABS 116	**06.116**	und in 116 speichern
029		LDA 101	**05.101**	Computerposition laden . . .
030		VGR 114	**12.114**	und nachschauen, ob sie größer als „158" ist
031		SPB 037 (L1)	**11.037**	falls ja, muß sie verringert werden
032		ADD 103	**07.103**	sonst addiere den Spaltenzähler hinzu . . .
033		ABS 101	**06.101**	und speichere als neue Position
034		SPU 061 (U1)	**09.061**	teste sie
035	R4:	VGL 105	**10.105**	ist sie gut?
036		SPB 058 (A4)	**11.058**	falls ja, anzeigen
037	L1:	LDA 103	**05.103**	sonst lade den Zähler für Spalten
038		SUB 106	**08.106**	und verringere ihn um „10"
039		ABS 103	**06.103**	wieder abspeichern
040		LDA 102	**05.102**	ebenso den Reihenzähler laden
041		SUB 105	**08.105**	um „1" verringern . . .
042		ABS 102	**06.102**	und wieder speichern
043		VGR 104	**12.104**	ist er noch nicht bei „0" angelangt?
044		SPB 011 (L0)	**11.011**	falls ja, eine neue Testrunde starten
045	A1:	LDA 100	**05.100**	sonst lade die Spielerposition
046		VGL 114	**10.114**	ist sie das Gewinnfeld?
047		SPB 051 (A3)	**11.051**	falls ja, zum „Ärgern" springen
048		ADD 106	**07.106**	sonst addiere „10" (eine Reihe weiterrücken)
049	A2:	ANZ	**02.000**	zeige diesen Verlegenheitszug an
050		SPU 049 (A2)	**09.049**	dauernd
051	A3:	LDA 104	**05.104**	lade „0" . . .
052		ANZ	**02.000**	und zeige sie an
053		VZG 050	**03.050**	kurze Zeit
054		LDA 115	**05.115**	dann „11.111" laden und . . .
055		ANZ	**02.000**	anzeigen
056		VZG 080	**03.080**	etwas länger
057		SPU 051 (A3)	**09.051**	weiter „ärgern" – der Spieler hat gewonnen
058	A4:	LDA 101	**05.101**	lade die Rechnerposition . . .
059	A5:	ANZ	**02.000**	und zeige sie . . .
060		SPU 059 (5)	**09.059**	dauernd an
				ab hier: Unterprogramm zum Testen der Postition
061	U1:	VGL 114	**10.114**	ist sie gerade „158"?
062		SPB 077 (E1)	**11.077**	falls ja, springe zu 077
063		VGL 113	**10.113**	ist sie „127"?
064		SPB 077 (E1)	**11.077**	falls ja, springe zu 077
065		VGL 112	**10.112**	ist sie „126"?
066		SPB 077 (E1)	**11.077**	falls ja, springe zu 077
067		VGL 111	**10.111**	ist sie „75"?
068		SPB 077 (E1)	**11.077**	falls ja, springe zu 077
069		VGL 110	**10.110**	ist sie „73"?
070		SPB 077 (E1)	**11.077**	falls ja, springe zu 077
071		VGL 109	**10.109**	ist sie „44"?
072		SPB 077 (E1)	**11.077**	falls ja, springe zu 077
073		VGL 108	**10.108**	ist sie „41"?
074		SPB 077 (E1)	**11.077**	falls ja, springe zu 077

Fortsetzung siehe nächste Seite

Fortsetzung Listing 41:

Adresse		Mnemonics	Code	Kommentar
075		AKO 000	**04.000**	Position ist nicht gut: „0" zurückbringen
076		SIU 116 (RX)	**21.116**	Rücksprung gemäß Adreßzelle
077	E1:	AKO 001	**04.001**	Position ist gut: „1" zurückbringen
078		SIU 116 (RX)	**21.116**	wie oben!
100		*	**00.xxx**	eingegebenes Feld
101			–	vom Programm ermitteltes Feld
102			–	Zähler für Reihen
103			–	Zähler für Spalten
104		*	**00.000**	„0"
105		*	**00.001**	„1"
106		*	**00.010**	„10"
107		*	**00.029**	„29"
108		*	**00.041**	„41"
109		*	**00.044**	„44" } Vergleichszahlen
110		*	**00.073**	„73"
111		*	**00.075**	„75"
112		*	**00.126**	„126"
113		*	**00.127**	„127"
114		*	**00.158**	„158"
115		*	**11.111**	für blinkende Anzeige
116			–	Rücksprungadreßzelle

2.15 Multiplikation–die aufwendige Art

Wir haben Ihnen im ersten Teil des Buches ein Multiplikationsprogramm vorgestellt, das leider einen Nachteil hatte: einfache Rechnungen, die wir im Kopf meist schneller machen können, waren damit möglich, während „richtig schwierige" an der 255er Grenze für das Produkt scheiterten! Unser neues Programm vermeidet dies nun, es kann bis maximal 156 × 255 rechnen (den Experten unter Ihnen wird es aber sicher nicht schwerfallen, es für noch größere Faktoren umzuschreiben). Nun ist aber 156 × 255 = 39 780 – wie sollen wir diese Zahl anzeigen lassen?

Ganz einfach: Wir zerpflücken sie in Zweiergruppen, also 03, 97, 80 und speichern jede Gruppe in einer eigenen Datenzelle. Das geht, weil ja bei zwei Ziffern maximal „99" auftreten kann, und das läßt unser Zahlenbereich noch zu. Und was das Programm nun macht, ist sofort klar: es addiert den ersten Faktor zur Zehner + Einer-Zelle (Z + E). Wird dieses Zwischenergebnis größer als „99", also eine dritte Stelle benötigt, trennt er diese einfach ab und addiert dafür „1" in der Tausender + Hunderter-Zelle (T + H). Und wenn diese Zelle dann überläuft, also „99" überschreitet, wird der Übertrag zur Zehntausender-Zelle (ZT) gemacht.

Dies ist übrigens genau das gleiche Vorgehen, wie wir es üblicherweise bei der Addition von Zahlenkolonnen machen. Dabei addieren wir aber immer nur *eine* Stelle und machen einen Übertrag auf die nächste. Das Programm hätte natürlich genauso geschrieben werden können, wäre dann aber länger und langsamer geworden; deshalb also in Zweiergruppen.

Am Schluß wird übrigens der Inhalt der drei benötigten Datenzellen hintereinander angezeigt, beginnend mit dem Zehntausender.

Hinweis: Will man dieselbe Aufgabe mehrmals lösen lassen, müssen die Faktoren in die Zellen 102 und 103 jeweils neu eingegeben werden.

Listing 42: **Multiplikation – die aufwendige Art**

Adresse	Mnemonics	Code	Kommentar
001	AKO 100	**04.100**	lade „100" . . .
002	ABS 104	**06.104**	und speichere sie als Vergleichszahl
003	AKO 001	**04.001**	lade „1" . . .

Fortsetzung siehe nächste Seite

Fortsetzung Listing 42:

Adresse		Mnemonics	Code	Kommentar
004		ABS 101	**06.101**	und speichere sie als Vergleichszahl
005		AKO 000	**04.000**	lade „0" . . .
006		ABS 100	**06.100**	und speichere sie als Vergleichszahl . . .
007		ABS 105	**06.105**	ebenso in den Produktzellen: (Z + E) . . .
008		ABS 106	**06.106**	(T + H) . . .
009		ABS 107	**06.107**	(ZT)
010	A0:	ADD 102	**07.102**	addiere 1. Faktor zum Akku
011	V0:	VKL 104	**13.104**	ist Akku jetzt noch kleiner als „100"?
012		SPB 014 (A1)	**11.014**	falls ja, nichts abtrennen
013		SPU 022 (A2)	**09.022**	sonst springe zur Hunderterabtrennung
014	A1:	ABS 105	**06.105**	speichere die (Z + E) wieder
015		LDA 103	**05.103**	lade den 2. Faktor . . .
016		SUB 101	**08.101**	und verringere ihn um „1": einmal mehr wurde addiert
017		VGL 100	**10.100**	ist „0" und damit das Ende schon erreicht?
018		SPB 036 (A4)	**11.036**	wenn ja, springe zur Anzeige
019		ABS 103	**06.103**	sonst speichere den 2. Faktor wieder . . .
020		LDA 105	**05.105**	lade die Zwischensumme (Z + E) . . .
021		SPU 010 (A0)	**09.010**	und springe zur nächsten Addition
022	A2:	SUB 104	**08.104**	subtrahiere „100" von (Z + E) . . .
023		ABS 105	**06.105**	und speichere (Z + E). Dann . . .
024		LDA 106	**05.106**	lade dafür (T + H) . . .
025		ADD 101	**07.101**	erhöhe sie um „1" . . .
026		ABS 106	**06.106**	und speichere sie wieder
027		VKL 104	**13.104**	ist (T + H) noch kleiner als „100"?
028		SPB 034 (A3)	**11.034**	falls ja, springe zur nächsten Addition
029		SUB 104	**08.104**	sonst reduziere sie um „100" . . .
030		ABS 106	**06.106**	und speichere (T + H) wieder
031		LDA 107	**05.107**	dann lade die (ZT) . . .
032		ADD 101	**07.101**	erhöhe sie um „1" . . .
033		ABS 107	**06.107**	und speichere sie wieder
034	A3:	LDA 105	**05.105**	lade die Zwischensumme (Z + E) . . .
035		SPU 011 (V0)	**09.011**	und mache weiter beim Vergleich
036	A4:	LDA 107	**05.107**	lade die (ZT) . . .
037		ANZ	**02.000**	und zeige sie an
038		VZG 250	**03.250**	} für insgesamt eine halbe Sekunde
039		VZG 250	**03.250**	
040		LDA 106	**05.106**	lade ebenso die (T + H)
041		ANZ	**02.000**	zeige sie an
042		VZG 250	**03.250**	} für eine halbe Sekunde . . .
043		VZG 250	**03.250**	
044		LDA 105	**05.105**	und lade die (Z + E)
045		ANZ	**02.000**	zeige sie an
046		SPU 045	**09.045**	weiterhin (Anzeigeschleife)
100			**00.000**	Vergleichszahl „0"
101			**00.001**	Schrittweite „1"
102		*	**00.xxx**	1. Faktor (maximal „156")
103		*	**00.xxx**	2. Faktor (maximal „255")
104			**00.100**	Vergleichszahl zum Abspalten
105			–	Z + E des Produkts
106			–	T + H des Produkts
107			–	ZT des Produkts

Listing 43: **Das endlose Divisionsprogramm** (siehe dazu Text S. 108)

Adresse		Mnemonics	Code	Kommentar
001	S0:	AKO 010	**04.010**	lade „10" . . .
002		ABS 105	**06.105**	und speichere sie als Multiplikationsfaktor
003		AKO 001	**04.001**	lade „1" . . .
004		ABS 103	**06.103**	und speichere sie als Schrittweite
005		AKO 000	**04.000**	lade „0"
006		ABS 102	**06.102**	setze das Ergebnis zurück . . .
007		ABS 104	**06.104**	ebenso den Divisionsrest . . .
008		ABS 106	**06.106**	und den Multiplikationszähler
009	A0:	LDA 100	**05.100**	lade den Dividenden
010		VKL 101	**13.101**	ist er kleiner als der Divisor?
011		SPB 018 (A1)	**11.018**	falls ja, springe zur Anzeige des Ergebnisses
012		SUB 101	**08.101**	sonst subtrahiere den Divisor . . .
013		ABS 100	**06.100**	und speichere den Rest der Dividenden wieder
014		LDA 102	**05.102**	dann lade das Ergebnis (Zähler der Subtraktionen)
015		ADD 103	**07.103**	erhöhe um „1" . . .
016		ABS 102	**06.102**	und speichere wieder
017		SPU 009 (A0)	**09.009**	dann springe zurück zur nächsten Subtraktion
018	A1:	ABS 104	**06.104**	genug geteilt – speichere den Dividenden als *Rest*
019		LDA 102	**05.102**	lade dafür das Ergebnis . . .
020		ANZ	**02.000**	und zeige es an
021		VZG 250	**03.250**	1 Sekunde lang
022		VZG 250	**03.250**	
023		VZG 250	**03.250**	
024		VZG 250	**03.250**	
025		LDA 104	**05.104**	alten Rest in den Akku laden
026		VGL 106	**10.106**	ist er bei „0" angelangt?
027		SPB 038 (E0)	**11.038**	falls ja, springe zum Programmende
028		AKO 000	**04.000**	sonst fange mit der Multiplikation des Rests mit „10" an
029	A2:	ADD 104	**07.104**	addiere den Rest zum Akku . . .
030		ABS 100	**06.100**	und speichere als Dividenden
031		LDA 106	**05.106**	lade den Multiplikationszähler
032		ADD 103	**07.103**	erhöhe ihn um „1" . . .
033		VGL 105	**10.105**	und vergleiche mit „10". Schon erreicht?
034		SPB 001 (S0)	**11.001**	falls ja, ist der neue Dividend fertig; springe zum neuen Dividieren
035		ABS 106	**06.106**	sonst speichere den Zähler wieder ab . . .
036		LDA 100	**05.100**	und lade den (noch nicht fertigen) Dividenden
037		SPU 029 (A2)	**09.029**	springe zurück in der Multiplikationsschleife
038	E0:	AKO 000	**04.000**	lade „0" – wir sind am Ende
039		ANZ	**02.000**	zeige sie . . .
040		VZG 250	**03.250**	kurz an . . .
041		HLT	**01.000**	und halte an.
100		*	**00.xxx**	Dividend
101		*	**00.xxx**	Divisor (≦ „26")
102			–	Ergebnis
103			**00.001**	Schrittweite
104			–	Rest bei Division
105			**00.010**	Multiplikationsfaktor
106			–	Zähler für Multiplikation

2.16 Das endlose Divisionsprogramm

Im ersten Teil des Buches haben wir ein Divisionsprogramm kennengelernt. Es arbeitete nach dem Prinzip der Reihen-Subtraktion, bis vom Dividenden weniger übrigblieb als der Divisor betrug. Rest und Ergebnis wurden in zwei Zellen gespeichert.
Jetzt gehen wir noch einen Schritt weiter: Wir multiplizieren den Rest mit „10", teilen diesen daraufhin wie oben bis zum nächsten Rest usw.
Das ist genau dasselbe, was wir bei der schriftlichen Division machen: Null anhängen, weiterteilen. Ein Beispiel:

```
30 : 7 = 4,285 . . .
28
 20
 14
  60
  56
   40
   35
    5
    .
    .
    .
```

Unser Programm tut also dasselbe, und zwar genauso lange, wie wir es auch machen würden: nämlich teilen, bis der Rest null wird; gibt es aber kein Ende (wie in obigem Beispiel), dann teilt das Programm ewig weiter!

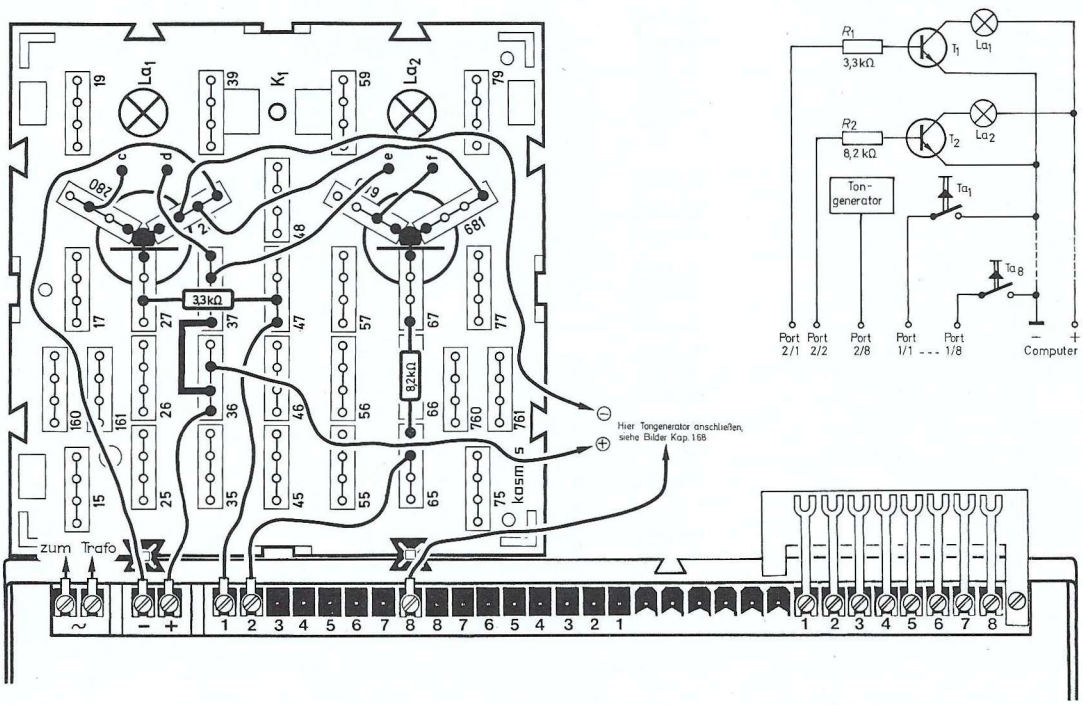
Wie zeigt uns das Programm das Ergebnis an?
Die **erste Anzeige** gibt uns den **ganzzahligen Teil,** die zweite, dritte usw. dann hintereinander die erste, zweite usw. **Stelle hinter dem Komma.** Um „unendlich viele" Stellen zu speichern, reicht unser Speicherplatz natürlich nicht aus.

Hinweis: Will man dieselbe Aufgabe mehrmals lösen lassen, müssen der Dividend und der Divisor in die Zellen 100 und 101 jeweils neu eingegeben werden. Der Divisor muß kleiner oder gleich „25" gewählt werden.

2.17 Computergesteuerte Personenrufanlage

In vielen Betrieben, Instituten usw. findet man optisch-akustische Personenrufanlagen, mit denen die Mitarbeiter darauf aufmerksam gemacht werden können, daß sie am Telefon verlangt werden. Das Prinzip ist sehr einfach: durch ein akustisches Signal (in unserem Fall ein Tongenerator) wird angekündigt, daß jemand gesucht wird, und durch ein „optisches Muster" von zwei Lampen (oder zwei LEDs), die ein- oder ausgeschaltet sein können oder blinken, wird die betreffende Person „angesprochen". Jedem Mitarbeiter ist also ein bestimmtes „Muster" zugeordnet, z.B. nach folgendem Schema:

Bild 77. Personenrufanlage, aufgebaut aus KOSMOS „Spiele mit Elektronik". Tongenerator, falls gewünscht, wie Bild 61



Personen-Nummer (Klemme von Port 1)	Name	Lampe 1 bzw. LED 1 (Klemme 1, Port 2)	Lampe 2 bzw. LED 2 (Klemme 2, Port 2)
1	Herr Meyer	an	aus
2	Frau Schultz	aus	an
3	Frau Müller	an	an
4	Frl. Zeisig	blinkt	aus
5	Herr Hansen	aus	blinkt
6	Herr Wagner	an	blinkt
7	Frl. Schmitz	blinkt	an
8	alle zusammen	blinkt	blinkt

Der Tongenerator wird an Klemme 8 von Port 2 angeschlossen.
Wenn niemand gerufen wird (*keine* der Klemmen von Port 1 auf 0 V), sind alle Lampen dunkel. Die verschiedenen Anzeigemuster müssen in Form einer Tabelle in die Zellen 112 bis 127 eingegeben werden. Jeweils zwei hintereinander liegende Zellen, also beispielsweise 112 und 113, enthalten dann die erste und die zweite Hälfte des Musters. Sie werden nacheinander in den Akku geladen und für jeweils 200 ms an die Lampen über Port 2 ausgegeben. Dies wird 25mal wiederholt und dauert also insgesamt 10 Sekunden. Dann ist der Eingabeport wieder „scharf", und die Klemmen von Port 1 werden vom Programm zyklisch abgefragt, ob die nächste Person gerufen wird, indem der Kontakt der entsprechenden Klemme zu 0 V geschlossen wird.

Listing 44: **Computergesteuerte Personenrufanlage**

Adresse		Mnemonics	**Code**	Kommentar
001		AKO 025	**04.025**	lade „25" . . .
002		ABS 100	**06.100**	als Dauer der Anzeige speichern
003		AKO 001	**04.001**	lade „1" . . .
004		ABS 101	**06.101**	als Schrittweite speichern
005		AKO 128	**04.128**	lade „128" . . .
006		ABS 102	**06.102**	als Zählergrenze für Port-Abfrage speichern
007	S1:	AKO 000	**04.000**	lade „0" . . .
008		P2A 000	**18.000**	und gib sie aus: Tongenerator und Lampen aus
009		ABS 104	**06.104**	setze den Anzeigeschleifenzähler zurück . . .
010		ABS 105	**06.105**	und die Eingabedecodierung
011		AKO 001	**04.001**	Eingaberoutine: lade „1" . . .
012		ABS 103	**06.103**	und speichere in der Musterzelle
013	S2:	P1E 000	**16.000**	lies Information an Port 1 und . . .
014		ABS 108	**06.108**	speichere sie in 108
015		AKO 255	**04.255**	lade „255" . . .
016		SUB 108	**08.108**	und subtrahiere das Eingelesene davon: jetzt als Rest bis 255 vorliegend
017		VGL 103	**10.103**	vergleiche mit der Musterzelle: Werte gleich?
018		SPB 027 (F1)	**11.027**	falls ja, wurde eine Eingabe gemacht; springe zum Anzeigen
019		AKO 002	**04.002**	sonst lade „2" . . .
020		ADD 105	**07.105**	und erhöhe so die Startadresse des Anzeigebereichs
021		ABS 105	**06.105**	speichere sie wieder
022		LDA 103	**05.103**	lade die Musterzelle
023		VGL 102	**10.102**	ist die 8. Klemme schon erreicht, also 128 gespeichert?
024		SPB 007 (S1)	**11.007**	falls ja, fange wieder bei der 1. Klemme an
025		ADD 103	**07.103**	sonst verdopple den Wert: Nimm nächste Klemme . . .
026		SPU 012 (S2)	**09.012**	und springe zum Port-Einlesen
027	F1:	AKO 112	**04.112**	richtige Leitung gefunden! Erzeuge die richtige Adresse zum . . .

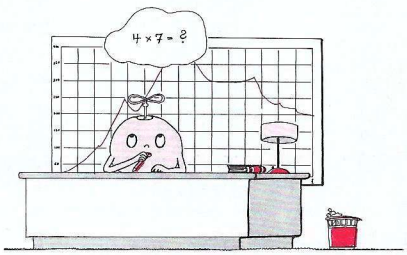
Fortsetzung siehe nächste Seite

Fortsetzung Listing 44:

Adresse	Mnemonics	Code	Kommentar
028	ADD 105	**07.105**	Anzeigen, indem zur Startadresse der Wert „112" addiert wird
029	ABS 106	**06.106**	dies ist die 1. Anzeigeadresse
030	ADD 101	**07.101**	„1" mehr wird dann als Adresse der zeitlich . . .
031	ABS 107	**06.107**	zweiten Hälfte des Musters gespeichert
032	F2: LIA 106	**19.106**	lade den Inhalt der ersten Anzeigeadresse . . .
033	P2A 000	**18.000**	und gib ihn aus
034	VZG 200	**03.200**	für 200 ms
035	LIA 107	**19.107**	auch den Inhalt der zweiten Musteradresse laden . . .
036	P2A 000	**18.000**	und ausgeben
037	VZG 200	**03.200**	für 200 ms
038	LDA 104	**05.104**	lade den Anzeigeschleifenzähler . . .
039	ADD 101	**07.101**	und erhöhe ihn um „1"
040	ABS 104	**06.104**	wieder speichern
041	VKL 100	**13.100**	ist er noch kleiner als „25"
042	SPB 032 (F2)	**11.032**	falls ja, weiterhin anzeigen
043	SPU 007 (S1)	**09.007**	sonst warte auf die nächste Eingabe
100		**00.025**	Anzeigeschleifengrenze
101		**00.001**	Schrittweite
102		**00.128**	Grenze des Musters beim Port-Vergleich
103		–	Vergleichszahl für Port-Information
104		–	Anzeigeschleifenzähler
105		–	Startadressenverschiebung für Muster
106		–	Adresse der ersten Musterhälfte
107		–	Adresse der zweiten Musterhälfte
108		–	Hilfszelle für die Port-Information
112	*	**00.129**	Codierte Anzeigemuster
113	*	**00.129**	
114	*	**00.130**	
115	*	**00.130**	
116	*	**00.131**	
117	*	**00.131**	
118	*	**00.129**	
119	*	**00.128**	
120	*	**00.130**	
121	*	**00.128**	
122	*	**00.129**	
123	*	**00.131**	
124	*	**00.130**	
125	*	**00.131**	
126	*	**00.129**	
127	*	**00.130**	

2.18 Arithmetik-Übungen

Bild 78



Wer kann richtig addieren/subtrahieren/multiplizieren/dividieren? Der Computer natürlich! Aber ob wir es auch können? Lassen wir uns doch einfach vom Computer testen. Er soll uns Aufgaben stellen, und wir müssen die Antwort geben. Stimmt sie, wird eine neue Aufgabe gestellt, sonst die alte nochmal.

Unser Programm ist in fünf Blöcke gegliedert: der erste Teil (Zelle 001–028) ist für Addition/Subtraktion, also Strichrechnung, der zweite (030–061) für Punktrechnung (Multiplikation/Division), der dritte (061–076) zeigt uns die Aufgabenstellung an, der vierte (077–087) erzeugt Zufallszahlen für die Aufgaben. Der letzte schließlich (088–107) testet, ob unsere Eingabe mit dem von ihm selbst ermittelten richtigen Ergebnis übereinstimmt. Ist dies der Fall, blinkt die Anzeige für eine gewisse Dauer, bevor sich der Computer eine neue Aufgabe „ausdenkt".
Die Zufallszahlen werden übrigens wieder mit der „Bitvertauschung" erzeugt, wie sie in Kapitel 2.0 beschrieben wurde. Port 1 und Port 2 sind also entsprechend zu verdrahten, bevor das Programm eingegeben wird!
Vor dem Programmstart müssen folgende Eingaben gemacht werden:

1. Die gewünschte Operation, die geübt werden soll, muß in Zelle 113 eingegeben werden. „1" steht dabei für „+", „2" für „–", „3" für „×" und „4" für „:".
2. Die oberen Grenzen für die beiden Rechenarten *müssen* im Speicher stehen und zwar der Zahlenwert „128" in Zelle 111 für Strich- und „16" in Zelle 112 für Punktrechnung.
3. Außerdem sollte eine Start-Zufallszahl in 118 eingegeben werden, und die Zellen 120–122 belegt werden.

Dann kann das Programm gestartet werden – übrigens bei **104!**
Der Computer denkt sich zwei Zahlen aus, mit denen die gewünschte Operation durchgeführt werden soll. Diese Zahlen nennen wir allgemein „Operanden", weil es sich ja – je nach Operation – um Faktoren, Minuend, Divisor usw. handeln kann. Dann zeigt der Computer den ersten Operanden für eine Sekunde an, anschließend die codierte Operation (also „1", „2", „3" oder „4") für eine Viertelsekunde und schließlich den zweiten Operanden in einer Anzeigeschleife. Nach Drücken auf STP kann das (vermutete? geratene? berechnete?) Ergebnis in 110 eingegeben und der Test auf Richtigkeit bei **088** gestartet werden.

Listing 45: **Arithmetik-Übungen**

Adresse	Mnemonics	Code	Kommentar
			Addition:
001	A1: AKO 003 (R1)	**04.003**	Erste Rücksprungadresse laden . . .
002	SPU 077 (U1)	**09.077**	und Zufallszahl erzeugen im Unterprogramm
003	R1: VGR 111	**12.111**	Rückkehr: ist sie größer als erlaubt?
004	SPB 001 (A1)	**11.001**	falls ja, eine neue erzeugen
005	ABS 114	**06.114**	sonst als 1. Operand speichern
006	A2: AKO 008 (R2)	**04.008**	zweite Rücksprungadresse laden . . .
007	SPU 077 (U1)	**09.077**	und zweite Zufallszahl erzeugen
008	R2: VGR 111	**12.111**	ist diese größer als erlaubt?
009	SPB 006 (A2)	**11.006**	falls ja, eine neue erzeugen
010	ABS 115	**06.115**	sonst als 2. Operand speichern
011	AKO 001	**04.001**	lade „1" als Operation, also „+"
012	VKL 113	**13.113**	ist dies kleiner als die gewünschte in 113?
013	SPB 018 (A4)	**11.018**	falls ja, zu 018 springen (Subtraktion)
014	A3: LDA 114	**05.114**	sonst lade den 1. Operanden
015	ADD 115	**07.115**	addiere den 2. Operanden dazu . . .
016	ABS 117	**06.117**	und speichere das Ergebnis
017	SPU 062 (C1)	**09.062**	dann springe zum Anzeigen der Aufgabe
			Subtraktion:
018	A4: LDA 115	**05.115**	lade den 2. Operanden (jetzt Subtrahend)
019	VKL 114	**13.114**	ist er kleiner als der Minuend?
020	SPB 026 (A5)	**11.026**	falls ja, ist keine Vertauschung nötig
021	ABS 116	**06.116**	sonst speichere den 2. Operanden in 116
022	LDA 114	**05.114**	lade den 1. Operanden
023	ABS 115	**06.115**	speichere ihn als neuen 2. Operanden
024	LDA 116	**05.116**	dann lade den alten 2. Operanden aus 116 . . .
025	ABS 114	**06.114**	und speichere ihn als neuen 1. Operanden
026	A5: LDA 114	**05.114**	lade den Minuenden . . .

Fortsetzung siehe nächste Seite

Fortsetzung Listing 45:

Adresse		Mnemonics	Code	Kommentar
027		SUB 115	**08.115**	und subtrahiere den Subtrahenden davon
028		ABS 117	**06.117**	speichere das Ergebnis in 117 . . .
029		SPU 062 (C1)	**09.062**	und springe zur Aufgabenstellung
				Multiplikation:
030	B1:	AKO 032 (R3)	**04.032**	dritte Rücksprungadresse laden . . .
031		SPU 077 (U1)	**09.077**	und Zufallszahl erzeugen im Unterprogramm
032	R3:	VGR 112	**12.112**	ist diese größer als erlaubt?
033		SPB 030 (B1)	**11.030**	falls ja, eine neue erzeugen
034		ABS 114	**06.114**	sonst als 1. Operanden speichern
035	B2:	AKO 037 (R4)	**04.037**	vierte Rücksprungadresse laden . . .
036		SPU 077 (U1)	**09.077**	und eine Zufallszahl erzeugen
037	R4:	VGR 112	**12.112**	ist diese größer als erlaubt?
038		SPB 035 (B2)	**11.035**	falls ja, nochmal eine erzeugen
039		ABS 115	**06.115**	sonst als 2. Operanden speichern
040		AKO 000	**04.000**	lade „0“ . . .
041		ABS 117	**06.117**	und setze so die Ergebniszelle zurück
042		LDA 115	**05.115**	lade den 2. Operanden
043		ABS 116	**06.116**	speichere ihn als Zähler in der Hilfszelle
044	B3:	LDA 116	**05.116**	lade den Zähler der Serienaddition
045		VGL 120	**10.120**	ist er schon „0“, also fertig?
046		SPB 053 (B4)	**11.053**	falls ja, springe heraus aus der Schleife
047		SUB 121	**08.121**	sonst verringere ihn um „1“ . . .
048		ABS 116	**06.116**	und speichere ihn wieder
049		LDA 117	**05.117**	lade das (Zwischen-)Ergebnis . . .
050		ADD 114	**07.114**	und addiere einmal mehr den 1. Operanden
051		ABS 117	**06.117**	speichere es wieder
052		SPU 044 (B3)	**09.044**	dann weiter in der Additionsschleife
053	B4:	AKO 003	**04.003**	lade „3“: Operation ist „x“
054		VGL 113	**10.113**	ist dies gleich der gewünschten Operation?
055		SPB 062 (C1)	**11.062**	falls ja, springe zur Aufgabenstellung
056		LDA 114	**05.114**	sonst: **Division!** Lade 1. Operanden . . .
057		ABS 116	**06.116**	und speichere ihn in 116
058		LDA 117	**05.117**	lade das Multiplikationsergebnis
059		ABS 114	**06.114**	speichere es als 1. Operanden (Dividend)
060		LDA 116	**05.116**	dann lade den alten 1. Operanden aus 116 . . .
061		ABS 117	**06.117**	und speichere ihn als neues Ergebnis
				Aufgabenstellung:
062	C1:	LDA 114	**05.114**	lade den 1. Operanden . . .
063		ANZ	**02.000**	und zeige ihn an
064		AKO 100	**04.100**	lade „100“ als Startwert der Zeitschleife
065	C2:	SUB 121	**08.121**	subtrahiere „1“ vom Zeitzähler
066		VZG 010	**03.010**	warte 10 Millisekunden
067		VGR 120	**12.120**	ist der Zeitzähler noch größer als „0“?
068		SPB 065 (C2)	**11.065**	falls ja, weiter warten und zählen
069		LDA 113	**05.113**	sonst lade die Rechenoperation . . .
070		ANZ	**02.000**	und zeige sie an . . .
071		VZG 255	**03.255**	für 3 × 250 Millisekunden . . .
072		VZG 255	**03.255**	also insgesamt . . .
073		VZG 255	**03.255**	eine Dreiviertelsekunde
074		LDA 115	**05.115**	lade den 2. Operanden
075		ANZ	**02.000**	zeige ihn an

Fortsetzung siehe nächste Seite

Fortsetzung Listing 45:

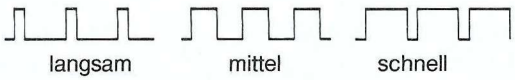
076	C3: SPU 076 (C3)	**09.076**	in einer Endlosschleife
			Unterprogramm Zufallszahlenerzeugung:
077	U1: ABS 119	**06.119**	speichere die Rücksprungadresse
078	LDA 118	**05.118**	lade den Unterprogrammzähler
079	VGL 121	**10.121**	ist er bei „1" angelangt, also fertig?
080	SPB 083 (U2)	**11.083**	falls ja, bei „255" neu starten
081	SUB 121	**08.121**	sonst um „1" verringern . . .
082	SPU 084 (U3)	**09.084**	und zum Speichern springen
083	U2: AKO 255	**04.255**	„255" laden als neuen Anfang
084	U3: ABS 118	**06.118**	speichern für Zufallszahlenzähler und . . .
085	P2A 000	**18.000**	außerdem an Port 2 ausgeben
086	P1E 000	**16.000**	an Port 1 vertauschte Information lesen . . .
087	SIU 119 (RX)	**21.119**	und mit dieser Zufallszahl Rücksprung gemäß gespeicherter Adresse in 119
			Test des geratenen Wertes:
088	E1: LDA 110	**05.110**	eingegebenes Ergebnis laden
089	VGL 117	**10.117**	ist es gleich dem gespeicherten, richtigen?
090	SPB 092 (E2)	**11.092**	falls ja, zum Anzeigen springen
091	SPU 062 (C1)	**09.062**	sonst die Aufgabe nochmal stellen
092	E2: AKO 010	**04.010**	anzeigen – lade „10"
093	E3: ABS 116	**06.116**	speichere in der Hilfszelle (10 mal blinken)
094	LDA 117	**05.117**	lade das richtige Ergebnis
095	ANZ	**02.000**	zeige es an . . .
096	VZG 250	**03.250**	für eine Viertelsekunde
097	LDA 122	**05.122**	dann lade „22.222" aus 122 . . .
098	ANZ	**02.000**	und zeige dies an . . .
099	VZG 030	**03.030**	aber nur für 30 Millisekunden
100	LDA 116	**05.116**	lade den Blinkzähler
101	SUB 121	**08.121**	verringere ihn um „1"
102	VGR 120	**12.120**	ist er noch größer als „0"?
103	SPB 093 (E3)	**11.093**	falls ja, weiter in der Blinkroutine
			Stellen einer Aufgabe:
104	AKO 003	**04.003**	lade „3" als Code für „x"
105	VGR 113	**12.113**	ist dies größer als das gewünschte?
106	SPB 001 (A1)	**11.001**	falls ja, springe zur Strichrechnung
107	SPU 030 (B1)	**09.030**	sonst springe zur Punktrechnung
110	*	**00.xxx**	geratenes Ergebnis
111	*	**00.128**	„128": obere Grenze Strichrechnung
112	*	**00.016**	„16": obere Grenze Punktrechnung
113	*	**00.00x**	Rechenoperation – siehe Text!
114		**–**	1. Operand
115		**–**	2. Operand
116		**–**	Hilfszelle zum Zählen
117		**–**	richtiges Ergebnis
118	*	**00.xxx**	Zufallszahl
119		**–**	Rücksprungadreßzelle (indirekt)
120	*	**00.000**	„0" als Vergleichszahl
121	*	**00.001**	„1" als Schrittweite und Vergleichszahl
122	*	**22.222**	„22.222" für blinkende Anzeige

2.19 Computergesteuerter Gleichstrommotor

Im Anleitungsbuch zum KOSMOS-Elektronik-Experimentierkasten E 203 wird die Arbeitsweise einer Motorsteuerung beschrieben. Die dort mit Elektronikbauteilen produzierten Rechteck-Steuerimpulse kann man selbstverständlich auch vom Computer erzeugen lassen. Die Überlegung für die Steuerung ist folgende:

Ein Gleichstrommotor läuft langsam, wenn die Spannung am Motor klein ist und schnell, wenn sie groß ist. Die Motorgeschwindigkeit kann jedoch auch dadurch verändert werden, daß man den Maximalwert der Betriebsspannung in rascher Folge ein- und ausschaltet und das Verhältnis von Einschalt- zu Ausschaltzeit variiert. Als Spannungs-Zeitdiagramm sieht das so aus:

Bild 79



Unser Programm macht eine solche Pulssteuerung, die auch ein realistisches Anfahren und Abbremsen ermöglicht. Es wurde für eine Eisenbahnanlage (Gleichstrom!) mit Signal und Kontaktgleisstück geschrieben, kann aber auch anderen Fällen angepaßt werden. Die nötigen Beschaltungen zwischen Computer und Eisenbahn können folgenden Bildern entnommen werden:

Bild 80. Computergesteuerte Gleichstrom-Modellbahn, aufgebaut mit Teilen des Fleischmann-Systems und KOSMOS Elektronik-Material (E 203 u. a.) (Schaltplan dazu Bild 81)

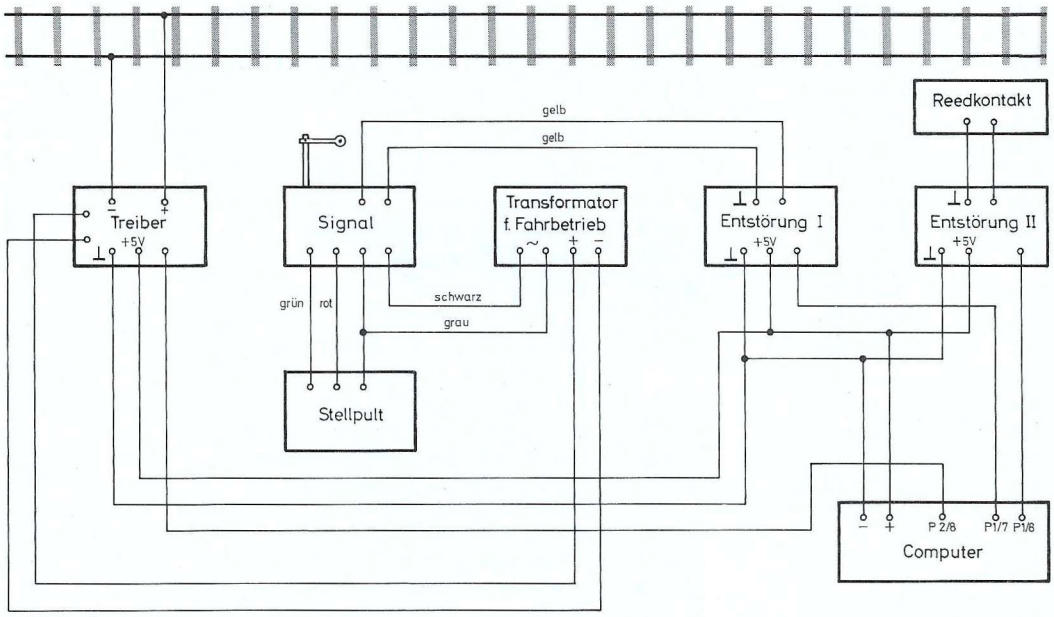
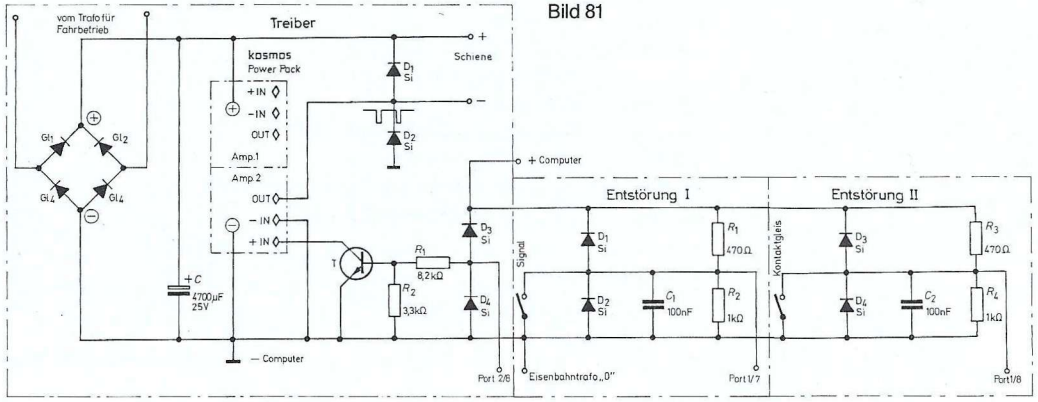


Bild 81



(Da dieses Programm sicherlich nur für die „Profis" unter Ihnen interessant ist, haben wir hier auf ein Aufbaubild für die Elektronik verzichtet.)
Was passiert während des Programmablaufs?
Ein Zug, der vor dem geschlossenen Signal steht, fährt langsam an, sobald sich das Signal öffnet (dann wird eine „0" an Port 1/Klemme 7 anliegen). Das Programm erzeugt nun eine Serie von Pulsen, deren Abstand immer kürzer wird, der Mittelwert der Spannung also steigt.
Beim Bremsvorgang geht das dann umgekehrt: Die Pulsfolge wird immer langsamer, demzufolge sinkt auch der Spannungsmittelwert ab. Aber natürlich wird dieses Bremsen nur eingeleitet, wenn das Signal geschlossen ist („1" an Port 1/Klemme 7) **und** das davor liegende Kontaktgleisstück überfahren wird („0" an Port 1/Klemme 8). Der Zug wird also nur langsamer, wenn er sich dem geschlossenen Signal nähert.
Die Länge der Beschleunigungs- bzw. Verzögerungsstrecke kann durch Ändern des Inhalts von Zelle 101 beeinflußt werden.
Um ein möglichst sauberes Anfahren und Abbremsen der Lok zu erreichen, sollte man in den Zeilen 010, 024, 045 und 056 verschiedene Verzögerungszeiten ausprobieren, sinnvolle Werte liegen zwischen „5" und „10". Die im Listing angegebenen VZG 000 müssen sicherlich in „echte" VZG-Befehle abgeändert werden. Es hängt natürlich von der jeweiligen Eisenbahnanlage ab, welche am besten geeignet sind. Es darf damit „gespielt" werden!

Listing 46: **Computergesteuerter Gleichstrommotor**

Adresse	Mnemonics	**Code**	Kommentar
001	AKO 000	**04.000**	lade „0"
002	ABS 102	**06.102**	als Anfangsgeschwindigkeit
003	A1: P1E 007	**16.007**	lies Signal-Zustand an Port 1
004	VGL 104	**10.104**	Signal auf Halt, also „1" gelesen?
005	SPB 003 (A1)	**11.003**	falls ja, weiterhin abfragen
006	LDA 101	**05.101**	sonst lade den Beschleunigungswert . . .
007	A2: ABS 100	**06.100**	und speichere ihn in der Zählerzelle
008	A3: AKO 001	**04.001**	Start des Zählers mit „1" im Akku
009	P2A 008	**18.008**	Ausgabe an den Motor
010	VZG 000	**03.000**	kurz warten . . .
011	P1E 008	**16.008**	lies Kontaktgleis-Zustand an Port 1
012	VGL 104	**10.104**	ist kein Zug dort (also „1" eingelesen)?
013	SPB 017 (A4)	**11.017**	falls richtig, weiterhin Pulse erzeugen
014	P1E 007	**16.007**	sonst lies noch den Signalzustand ein
015	VGL 104	**10.104**	ist das Signal auf Halt, also „1"?
016	SPB 043 (B2)	**11.043**	falls ja, springe zum Abbremsen
017	A4: LDA 102	**05.102**	sonst lade die Geschwindigkeit
018	A5: VGL 103	**10.103**	ist sie „0"?
019	SPB 022 (A6)	**11.022**	falls ja, kann der Motor ausgeschaltet werden
020	SUB 104	**08.104**	sonst warte noch ein bißchen – verringere den Akku um „1" . . .
021	SPU 018 (A5)	**09.018**	und vergleiche wieder
022	A6: AKO 000	**04.000**	„0" laden . . .
023	P2A 008	**18.008**	und dem Motor mitteilen: Halt!
024	VZG 000	**03.000**	Dauer des „Nullpulses"
025	LDA 102	**05.102**	Geschwindigkeit laden
026	A7: VGL 105	**10.105**	ist sie gleich „10", also maximal?
027	SPB 030 (A8)	**11.030**	dann springe zur Zählschleife
028	ADD 104	**07.104**	sonst erhöhe um „1" . . .
029	SPU 026 (A7)	**09.026**	und vergleiche erneut
030	A8: LDA 100	**05.100**	lade den Zähler . . .
031	SUB 104	**08.104**	und verringere ihn um „1"
032	VGR 103	**12.103**	ist er noch größer als „0"?
033	SPB 007 (A2)	**11.007**	dann läuft der Motor noch – weiter bei 007

Fortsetzung siehe nächste Seite

Fortsetzung Listing 46:

Adresse	Mnemonics	Code	Kommentar
034	LDA 101	**05.101**	sonst lade die Beschleunigung . . .
035	ABS 100	**06.100**	als neuen Zähleranfang speichern
036	LDA 102	**05.102**	lade die Geschwindigkeit
037	VGL 105	**10.105**	ist sie maximal?
038	SPB 008 (A3)	**11.008**	falls ja, sofort den nächsten Puls erzeugen
039	ADD 104	**07.104**	sonst um „1" erhöhen . . .
040	ABS 102	**06.102**	und wieder speichern . . .
041	SPU 008 (A3)	**09.008**	und den nächsten Puls erzeugen
042	B1: ABS 100	**06.100**	Zähler speichern
043	B2: AKO 001	**04.001**	Puls an Motor senden . . .
044	P2A 008	**18.008**	über Port 2
045	VZG 000	**03.000**	Dauer des Pulses
046	P1E 007	**16.007**	Signalzustand einlesen
047	VGL 103	**10.103**	Ist es geöffnet, also „0" eingelesen?
048	SPB 008 (A3)	**11.008**	falls ja, springe zum Beschleunigen
049	LDA 102	**05.102**	sonst lade die momentane Geschwindigkeit
050	B3: VGL 103	**10.103**	ist sie „0"?
051	SPB 054 (B4)	**11.054**	falls ja, springe zum Null-Senden
052	SUB 104	**08.104**	sonst reduziere die Geschwindigkeit um „1" . . .
053	SPU 050 (B3)	**09.050**	und vergleiche wieder
054	B4: AKO 000	**04.000**	„0" laden für den Motor . . .
055	P2A 008	**18.008**	und ihn so ausschalten
056	VZG 000	**03.000**	Dauer des Ausschaltens
057	LDA 102	**05.102**	lade die Geschwindigkeit
058	B5: VGL 105	**10.105**	ist sie „10", also maximal?
059	SPB 062 (B6)	**11.062**	falls ja, muß sie verringert werden
060	ADD 104	**07.104**	sonst wird sie erhöht um „1" . . .
061	SPU 058 (B5)	**09.058**	und wieder verglichen
062	B6: LDA 100	**05.100**	lade den Zähler für die Verlangsamung
063	SUB 104	**08.104**	und verringere ihn um „1" . . .
064	VGR 103	**12.103**	ist er noch nicht bei „0" angelangt (Verzögerung beendet)?
065	SPB 042 (B1)	**11.042**	falls ja, müssen noch Pulse gesendet werden
066	LDA 102	**05.102**	sonst lade die Geschwindigkeit
067	VGL 103	**10.103**	ist sie „0"?
068	SPB 003 (A1)	**11.003**	falls ja, steht der Zug: jetzt kontrolliere das Signal
069	SUB 104	**08.104**	sonst reduziere die Geschwindigkeit . . .
070	ABS 102	**06.102**	und speichere sie wieder
071	LDA 101	**05.101**	lade den Verzögerungswert für die Zählerschleife . . .
072	SPU 042 (B1)	**09.042**	und springe zur Verlangsamung
100		–	Zähler für Beschleunigung/Verlangsamung
101	*	**00.050**	Größe der Beschleunigung/Verlangsamung
102		–	momentane Geschwindigkeit
103	*	**00.000**	Vergleichszahl „0"
104	*	**00.001**	Schrittweite „1"
105	*	**00.010**	Vergleichszahl „10"

2.20 Digitalanzeigender Herzschlagmesser

Bild 82

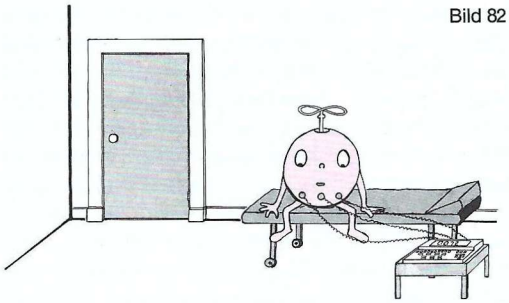


Bild 83. Digitalanzeigender Herzschlagmesser, aufgebaut aus KOSMOS „Elektronik-Labor E 200" und „Laborausbau E 204" (Schaltplan dazu Bild 84)

Das folgende Programm kann als (langsamer) Frequenzmesser für elektronische Pulse verwendet werden. Es ist so konzipiert, daß Frequenzen zwischen 50 und 150 Pulse je Minute registriert und angezeigt werden können. Damit ist es zur Messung der Herzfrequenz gut geeignet: in Ruhe hat der Mensch einen Puls von etwa 70 je Minute, während und kurz nach einer Anstrengung aber bis über 130. Mit Bauteilen des KOSMOS-Elektroniklabors E 200 und der Erweiterung E 204 „Bio-Elektronik" können wir uns selbst an den Computer anschließen und die Herzfrequenz messen.

Um eine vollständige elektrische Trennung zwischen der batterieversorgten Meßanordnung und dem netzbetriebenen Computer zu erreichen, wurden hier eine Infrarot-Leuchtdiode und ein Fototransistor (zum Beispiel aus dem Ergänzungskasten KOSMOS E 202 „Infrarot-Praxis") verwendet.

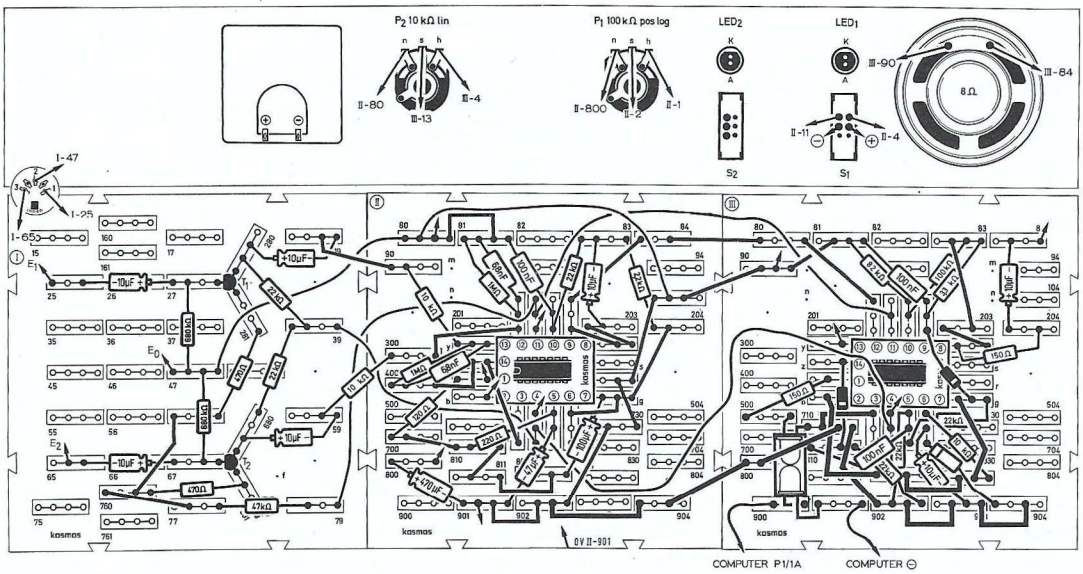
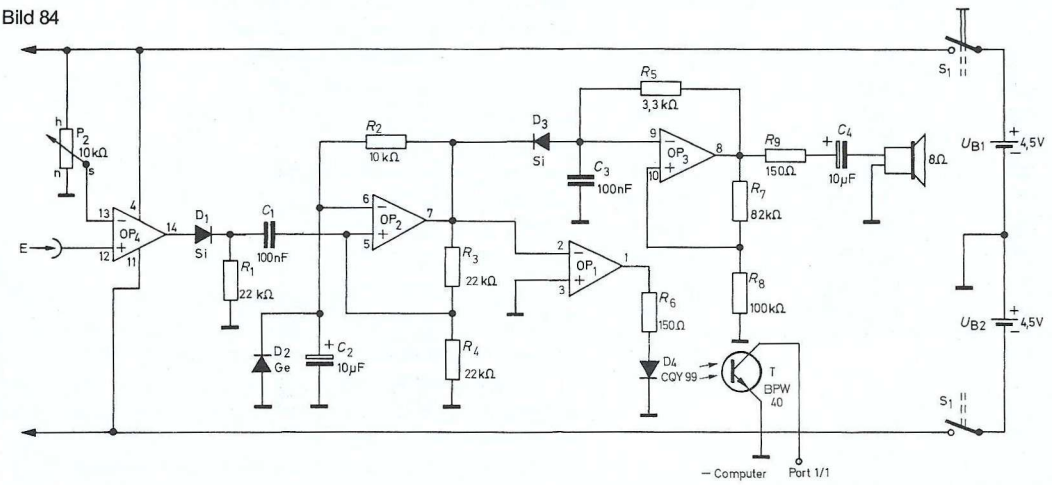


Bild 84



Der Puls, den die Meßschaltung verstärkt, wird also zuerst in infrarotes Licht und dann wieder in elektrischen Strom zurückverwandelt; diese Anordnung nennt man „Opto-Koppler", sie gewährleistet eine sichere Trennung.
Wir können die Pulse, die wir messen wollen, natürlich auch durch eine andere Schaltung, etwa einem langsamen Multivibrator, simulieren lassen. Das Programm verarbeitet alle Pulse von mindestens 10 Millisekunden Länge und zwischen 300 und 1000 Millisekunden langen Pausen dazwischen.
Auf eine Besonderheit des Programms soll noch hingewiesen werden: Es kann nur *Zeiten* messen, die zugehörigen Frequenzen müssen dann über eine Tabelle (Zelle 036–070) ermittelt werden. Da die Zeitmessung zwischen zwei Pulsen meistens leicht unterschiedliche Werte ergibt, würde auch die Frequenz ständig variieren und bei jedem Puls ein anderer Wert angezeigt werden. Deshalb wurde

Listing 47: **Digitalanzeigender Herzschlagmesser**

Adresse		Mnemonics	**Code**	Kommentar
001		AKO 025	**04.025**	lade „25" . . .
002		ABS 102	**06.102**	als Startwerte für Pulsmessung: Frequenz 1 . . .
003		ABS 103	**06.103**	und Frequenz 2 speichern
004		AKO 001	**04.001**	lade „1" . . .
005		ABS 101	**06.101**	und speichere als Zählerschrittweite
006		AKO 000	**04.000**	lade „0" . . .
007		ABS 100	**06.100**	und speichere als Vergleichszahl
008	Z1:	AKO 035	**04.035**	lade „35" als Startwert für die Frequenztabelle . . .
009		ABS 104	**06.104**	und speichere als Zeit-Startwert
010	Z2:	LDA 104	**05.104**	lade den Zeitzähler
011		ADD 101	**07.101**	erhöhe um „1" . . .
012		ABS 104	**06.104**	und speichere ihn wieder
013		VZG 007	**03.007**	warte kurz
014		P1E 001	**16.001**	lies Port 1 Klemme 1 (Pulsleitung)
015		VGL 101	**10.101**	ist der Wert „1", also Puls da?
016		SPB 025 (F1)	**11.025**	falls ja, springe zur Umrechnung
017		VZG 007	**03.007**	sonst warte 7 Millisekunden
018		P1E 001	**16.001**	lies Port 1 Klemme 1 (Pulsleitung)
019		VGL 101	**10.101**	ist der Wert „1", also Puls da?
020		SPB 025 (F1)	**11.025**	falls ja, springe zur Umrechnung
021		VZG 007	**03.007**	sonst warte 7 Millisekunden
022		P1E 001	**16.001**	lies Port 1 Klemme 1 (Pulsleitung)
023		VGL 100	**10.100**	ist der Wert „0", also kein Puls da?
024		SPB 010 (Z2)	**11.010**	falls ja, springe zurück zum Zeitzählen
025	F1:	LIA 104	**19.104**	Puls erfaßt! Lade die aktuelle Frequenz, die der Zeit in 104 entspricht (Tabelle)
026		ADD 102	**07.102**	addiere die vorletzte Frequenz dazu . . .
027		ADD 103	**07.103**	addiere die letzte Frequenz dazu
028		ANZ	**02.000**	und zeige dies als gemittelte Frequenz an
029		LDA 103	**05.103**	lade die letzte Frequenz
030		ABS 102	**06.102**	speichere sie als neue vorletzte
031		LIA 104	**19.104**	lade die aktuelle Frequenz über 104
032		ABS 103	**06.103**	speichere sie als neue letzte
033		VZG 255	**03.255**	dann warte rund 0,3 Sekunden, bis der . . .
034		VZG 030	**03.030**	Puls garantiert beendet ist.
035		SPU 008 (Z1)	**09.008**	springe zur nächsten Zeitmessung
100			**00.000**	„0" als Vergleichszahl
101			**00.001**	„1" als Schrittweite
102			–	vorletzte Frequenz
103			–	letzte Frequenz
104			–	Zeitzähler; gleichzeitig Adresse für die neue Frequenz

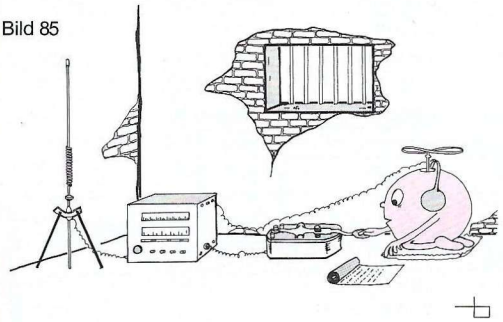
eine Mittelungsroutine eingefügt, die den momentan gemessenen Wert zu den beiden letzten gemessenen Werten addiert und über drei Pulse hinweg mittelt. Für die nächste Messung wird dann immer der „älteste" Wert gelöscht und nur die beiden „jüngsten" aufbewahrt (Zelle 102 und 103), um dann wieder gemittelt zu werden. Die dreifache Wiederholung der P1E-Folge (Zelle 013–024) dient übrigens nur dazu, selbst kürzeste Pulslängen (ca. 10 Millisekunden) noch exakt zu erfassen. Hätten wir statt der drei VZG 007 einen VZG 021 eingebaut, wäre die Mindestdauer für einen Puls auf etwa 25 Millisekunden angewachsen. So wird der Port öfter gelesen und in den Akku transportiert und die Chance, einen Puls zu übersehen, ist dadurch verringert.

Frequenztabelle:

Adresse	Datum	Adresse	Datum
036	00.056	054	00.023
037	00.053	055	00.023
038	00.050	056	00.022
039	00.047	057	00.021
040	00.044	058	00.021
041	00.042	059	00.020
042	00.040	060	00.020
043	00.038	061	00.019
044	00.036	062	00.019
045	00.034	063	00.018
046	00.032	064	00.018
047	00.031	065	00.018
048	00.029	066	00.017
049	00.028	067	00.017
050	00.027	068	00.017
051	00.026	069	00.017
052	00.025	070	00.016
053	00.024		

2.21 Morsecomputer

Bild 85



Morsen ist auch im Zeitalter des Telefons, des Funksprechgerätes und der Satelliten kein alter Hut – viele Kurzwellenfreunde funken auch heute noch Morsezeichen, und das altbewährte Seenotrettungszeichen SOS (dreimal kurz, dreimal lang, dreimal kurz = . . . – – – . . .) ist wohl auch jedem Laien geläufig.

Wenn es mit dem fehlerfreien Tasten der Morsezeichen noch hapert, sollte man den Computer mit unserem Morseprogramm füttern und ihn die Morsezeichen automatisch senden lassen. Da unser Computer nur Zahlen verarbeiten kann, müssen die Buchstaben des Morsealphabets in Form von Zahlen codiert werden.

Der Morse-Code besteht bekanntlich aus kurzen Tönen (Punkten) und langen Tönen (Strichen) in einer bestimmten Reihenfolge. Der Computer muß also wissen, aus *wievielen* Tönen ein Morsezeichen besteht und *auf welche Weise* kurze und lange Töne aufeinanderfolgen. Er braucht zwei Informationen für jedes Zeichen.

Für die Tonart gibt es zwei Möglichkeiten: lang oder kurz. Das legt natürlich nahe, die Tonfolge binär zu codieren und für kurz = „0" und für lang = „1" zu setzen. Das Morse-Q (– – . –) lautet dann binär 1101. Es besteht aus vier Bits. Die Zahl Vier verschlüsseln wir als Dualzahl (100) und setzen sie *vor* den Binärwert für die Tonfolge. Das würde dann 100/1101 ergeben. Da unser Computer ein 8 bit-Computer ist, wird an diesen Binärwert ganz rechts eine 0 „angehängt", und das Ganze, aus zwei Teilcodierungen zusammengesetzte Byte ist fertig. Man kann ihm natürlich jetzt auch eine Dezimalzahl (nach der Tabelle auf Seite 50 ergibt sich 154) zuordnen; aber Vorsicht: denken Sie stets daran, daß das Bitmuster eigentlich keine Dualzahl ist, die Dual-Dezimalwandlung hier also nur eine Hilfsfunktion erfüllt.

100	1101	0
Anzahl der Töne (dual: 100 = dezimal 4)	Tonfolge (1 = lang, 0 = kurz)	zur Vervollständigung des Byte angehängte 0

In der nachfolgenden Tabelle ist zusammengestellt, wie die Morsezeichen codiert und in unseren Computer als Dezimalzahlen eingegeben werden müssen.

Zeichen	Morsezeichen	Binärwert	Dezim.-wert
A	· –	01001000	72
B	– · · ·	10010000	144
C	– · – ·	10010100	148
D	– · ·	01110000	112
E	·	00100000	32
F	· · – ·	10000100	132
G	– – ·	01111000	120
H	· · · ·	10000000	128
I	· ·	01000000	64
J	· – – –	10001110	142
K	– · –	01110100	116
L	· – · ·	10001000	136
M	– –	01011000	88
N	– ·	01010000	80
O	– – –	01111100	124
P	· – – ·	10001100	140
Q	– – · –	10011010	154
R	· – ·	01101000	104
S	· · ·	01100000	96
T	–	00110000	48
U	· · –	01100100	100
V	· · · –	10000010	130
W	· – –	01101100	108
X	– · · –	10010010	146
Y	– · – –	10010110	150
Z	– – · ·	10011000	152
0	– – – – –	10111111	191
1	· – – – –	10101111	175

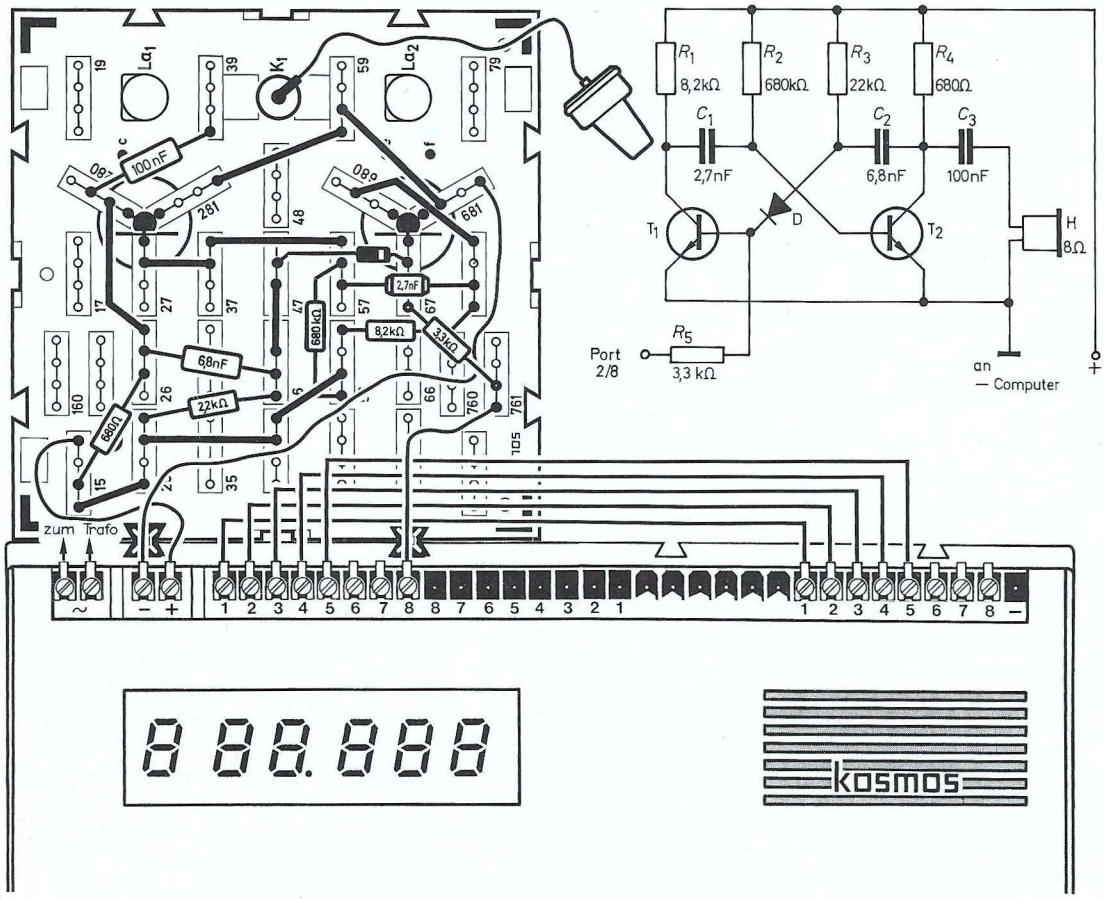
Zeichen	Morsezeichen	Binärwert	Dezim.-wert
2	· · – – –	10100111	167
3	· · · – –	10100011	163
4	· · · · –	10100001	161
5	· · · · ·	10100000	160
6	– · · · ·	10110000	176
7	– – · · ·	10111000	184
8	– – – · ·	10111100	188
9	– – – – ·	10111110	190
Ende		00000000	0

Was wir also zu tun haben, ist, die Dezimalcodierung in den Speicher (nennen wir ihn „Textspeicher", ab Zelle 087) einzugeben. Etwa so:

Text:	Hallo Kurt (Ende)									
Code:	128	72	136	136	124	116	100	104	48	0
Zelle:	087	088	089	090	091	092	093	094	095	096

Dann verdrahten wir die Ein-/Ausgabeports wie folgt mit dem Tongenerator:

Bild 86. Morsecomputer, aufgebaut aus KOSMOS „Elektronik-Junior"



und starten das Programm. In Windeseile wird der Text herausgepiepst! Im Prinzip könnten wir diese Morsekette auch über eine Amateurfunkanlage in den Äther schießen, und wenn wir uns bei der Codierung nicht vertan haben, sogar fehlerfrei!
Im Datenbereich müssen übrigens die Zellen 078–085 vom Benutzer selbst belegt werden, ab 087 fängt dann der Textspeicher an. Wer die Speichererweiterung besitzt, kann sogar 169 Zeichen lange Texte morsen lassen! Wem die Morserei zu langsam geht, kann sie auch noch beschleunigen: in Zelle 072 ist die Pausendauer zwischen einzelnen Tönen vermerkt, in Zelle 005 diejenige zwischen den Zeichen. Und die Tondauer selbst in den Zellen 065 (lang) und 066 (kurz).

Listing 48: **Morsecomputer**

Adresse	Mnemonics	**Code**	Kommentar
001	AKO 087	**04.087**	lade „87"
002	ABS 086	**06.086**	speichere als Startadresse des Textbereichs
003	A1: AKO 000	**04.000**	lade „0"
004	ABS 075	**06.075**	speichere als Startwert Tonzähler
005	VZG 150	**03.150**	warte 150 ms zwischen den Zeichen
006	LIA 086	**19.086**	lade das nächste Zeichen (Adresse in 086) . . .
007	ABS 076	**06.076**	und speichere es zwischendurch in 076
008	VGR 078	**12.078**	ist es größer als „0"?
009	SPB 012 (A3)	**11.012**	falls ja, ist es ein gültiger Code
010	A2: ANZ	**02.000**	sonst zeige „0" als Textende an . . .
011	SPU 010 (A2)	**09.010**	in einer Anzeigeschleife (Programmende)
012	A3: LDA 076	**05.076**	lade das zwischengespeicherte Zeichen
013	VKL 080	**13.080**	ist es kleiner als „32"?
014	SPB 021 (A4)	**11.021**	falls ja, sind die ersten drei Bits abgetrennt für die Tonzahl: nur noch Tonfolge allein!
015	SUB 080	**08.080**	sonst subtrahiere „32" . . .
016	ABS 076	**06.076**	und speichere den Wert wieder
017	LDA 075	**05.075**	lade den Tonzähler
018	ADD 079	**07.079**	und addiere „1" dazu: ein Ton mehr
019	ABS 075	**06.075**	speichere ihn wieder
020	SPU 012 (A3)	**09.012**	Rücksprung zum nächsten Test
021	A4: ABS 077	**06.077**	Tonanzahl wurde gezählt. Speichere Rest als Tonfolge in 077
022	P2A 000	**18.000**	gib die Tonfolge an Port 2 aus
023	AKO 028 (R1)	**04.028**	1. Rücksprungadresse laden . . .
024	ABS 074	**06.074**	und in der Rücksprungadreßzelle speichern
025	LDA 081	**05.081**	Befehl „P1E 005" laden (Stelle des 1. Tons) . . .
026	ABS 062	**06.062**	und ins Unterprogramm schreiben!!
027	SPU 060 (U1)	**09.060**	springe ins Unterprogramm zur Tonerzeugung
028	R1: VGL 078	**10.078**	ist der Rest jetzt „0" (alle Töne raus)?
029	SPB 056 (R5)	**11.056**	falls ja, Ende dieses Zeichens: Heraussspringen
030	AKO 035 (R2)	**04.035**	2. Rücksprungadresse laden . . .
031	ABS 074	**06.074**	und in der Rücksprungadreßzelle speichern
032	LDA 082	**05.082**	Befehl „P1E 004" laden (Stelle des 2. Tons) . . .
033	ABS 062	**06.062**	und ins Unterprogramm schreiben!!
034	SPU 060 (U1)	**09.060**	springe ins Unterprogramm zur Tonerzeugung
035	R2: VGL 078	**10.078**	ist der Rest jetzt „0" (alle Töne raus)?
036	SPB 056 (R5)	**11.056**	falls ja, Ende dieses Zeichens: Heraussspringen
037	AKO 042 (R3)	**04.042**	3. Rücksprungadresse laden . . .
038	ABS 074	**06.074**	und in der Rücksprungadreßzelle speichern
039	LDA 083	**05.083**	Befehl „P1E 003" laden (Stelle des 3. Tons) . . .
040	ABS 062	**06.062**	und ins Unterprogramm schreiben!!

Fortsetzung siehe nächste Seite

Fortsetzung Listing 48:

Adresse		Mnemonics	Code	Kommentar
041		SPU 060 (U1)	**09.060**	springe ins Unterprogramm zur Tonerzeugung
042	R3:	VGL 078	**10.078**	ist der Rest jetzt „0" (alle Töne raus)?
043		SPB 056 (R5)	**11.056**	falls ja, Ende dieses Zeichens: Heraussspringen
044		AKO 049 (R4)	**04.049**	4. Rücksprungadresse laden . . .
045		ABS 074	**06.074**	und in der Rücksprungadreßzelle speichern
046		LDA 084	**05.084**	Befehl „P1E 002" laden (Stelle des 4. Tons) . . .
047		ABS 062	**06.062**	und ins Unterprogramm schreiben!!
048		SPU 060 (U1)	**09.060**	springe ins Unterprogramm zur Tonerzeugung
049	R4:	VGL 078	**10.078**	ist der Rest jetzt „0" (alle Töne raus)?
050		SPB 056 (R5)	**11.056**	falls ja, Ende dieses Zeichens: Heraussspringen
051		AKO 056 (R5)	**04.056**	5. Rücksprungadresse laden . . .
052		ABS 074	**06.074**	und in der Rücksprungadreßzelle speichern
053		LDA 085	**05.085**	Befehl „P1E 001" laden (Stelle des 5. Tons) . . .
054		ABS 062	**06.062**	und ins Unterprogramm schreiben!!
055		SPU 060 (U1)	**09.060**	springe ins Unterprogramm zur Tonerzeugung
056	R5:	LDA 086	**05.086**	lade die Adresse des gemorsten Zeichens
057		ADD 079	**07.079**	addiere „1" dazu . . .
058		ABS 086	**06.086**	und speichere sie wieder
059		SPU 003 (A1)	**09.003**	springe zurück zum Decodieren des nächsten Zeichens
				Unterprogramm zur Tonausgabe:
060	U1:	AKO 001	**04.001**	lade „1"
061		P2A 008	**18.008**	Ausgabe an Tongenerator: Ton an
062		P1E 001	**16.001**	lies die Länge des Tones an Port 1 – wird vom Hauptprogramm überschrieben!
063		VGL 078	**10.078**	ist die Länge „0", also „kurz"?
064		SPB 066 (U2)	**11.066**	falls ja, springe zu 066 (kurzer Ton)
065		VZG 200	**03.200**	sonst warte 200 ms (langer Ton)
066	U2:	VZG 050	**03.050**	warte 50 ms zur Tonausgabe
067		AKO 000	**04.000**	lade „0"
068		P2A 008	**18.008**	Ausgabe an Tongenerator: Ton aus
069		LDA 075	**05.075**	lade den Tonzähler
070		SUB 079	**08.079**	verringere ihn um „1" . . .
071		ABS 075	**06.075**	und speichere ihn wieder
072		VZG 200	**03.200**	200 ms Pause zwischen den Tönen
073		SIU 074 (RX)	**21.074**	indirekter Rücksprung ins Hauptprogramm
074			–	Rücksprungadresse aus Unterprogramm
075			–	Tonzähler (Anzahl der Töne des Zeichens)
076			–	Hilfszelle
077			–	Tonfolge des Zeichens allein
078		*	**00.000**	„0" als Vergleichszahl
079		*	**00.001**	„1" als Schrittweite
080		*	**00.032**	„32" als Vergleichszahl
081		*	**16.005**	Befehle zum Lesen des richtigen Zeichencodes; werden ins Unterprogramm hineinkopiert
082		*	**16.004**	
083		*	**16.003**	
084		*	**16.002**	
085		*	**16.001**	
086			–	Adresse der Zeichen im Textbereich

Listing 49: **Programmierbares Monoflop** (siehe dazu Text S. 124)

Adresse		Mnemonics	**Code**	Kommentar
001	S1:	LDA 107	**05.107**	lade den Ausgangszustand
002		NEG	**14.000**	invertiere ihn: dies ist der Ruhezustand
003		P2A 001	**18.001**	gib ihn an Port 2/1 aus
004		LDA 108	**05.108**	lade die Haltezeit . . .
005		ABS 102	**06.102**	und speichere sie in der Rückwärtszählerzelle
006	S2:	P1E 001	**16.001**	lies die Eingangsklemme Port 1/1
007		VGL 105	**10.105**	ist sie wie in 105 vorgesehen, also geschaltet
008		SPB 010 (A1)	**11.010**	falls ja, starte das Monoflop
009		SPU 006 (S2)	**09.006**	sonst lies weiterhin Port 1/1
010	A1:	LDA 107	**05.107**	lade den Ausgangszustand . . .
011		P2A 001	**18.001**	und gib ihn aus
012	A2:	AKO 100	**04.100**	lade „100“ . . .
013	A3:	ABS 103	**06.103**	und speichere als Hundertstelsekundenzähler
014		P1E 001	**16.001**	lies die Eingangsklemme Port 1/1 wieder ein
015		VGL 105	**10.105**	ist sie schon wieder betätigt?
016		SPB 032 (R1)	**11.032**	falls ja, gehe in die Retriggerroutine
017	A4:	P1E 008	**16.008**	sonst lies den Reset-Eingang Port 1/8 in den Akku
018		VGL 100	**10.100**	ist er auf „0“, also betätigt?
019		SPB 001 (S1)	**11.001**	falls ja, gibt es einen neuen Anfang
020		VZG 005	**03.005**	sonst warte ein bißchen
021		LDA 103	**05.103**	lade den Hundertstelsekundenzähler
022		SUB 101	**08.101**	verringere um „1“: 1/100 Sekunde weniger
023		VGR 100	**12.100**	ist die Sekunde noch nicht rum?
024		SPB 013 (A3)	**11.013**	falls ja, mache weiter in der Schleife
025		LDA 102	**05.102**	lade den Sekunden-Rückwärtszähler
026		SUB 101	**08.101**	verringere ihn um „1“ . . .
027		ANZ	**02.000**	und zeige die verbleibenden Sekunden an
028		ABS 102	**06.102**	dann speichere die Restsekunden wieder
029		VGR 100	**12.100**	ist die Haltezeit noch nicht durchlaufen, also Zähler größer als „0“?
030		SPB 012 (A2)	**11.012**	falls ja, eine neue 1/100-Runde starten
031		SPU 001 (S1)	**09.001**	sonst sind wir fertig – Ausgangzustand wiederherstellen
032	R1:	LDA 106	**05.106**	Retrigger-Zelle laden
033		VGL 100	**10.100**	ist das Retriggern abgeschaltet, also „0“?
034		SPB 017 (A4)	**11.017**	falls ja, mache einfach in der 1/100-Schleife weiter sonst starte die Sekundenzählerei von vorne:
035		LDA 108	**05.108**	lade die Haltezeit aus 108
036		ABS 102	**06.102**	speichere sie in der Rückwärtszählerzelle . . .
037		SPU 012 (A2)	**09.012**	und starte eine neue Haltezeit
100		*	**00.000**	Vergleichszahl „0“
101		*	**00.001**	Schrittweite „1“
102			–	Zeitschleifenzähler
103			–	Hilfszelle
104			–	Hilfszelle
105		*	**00.00x**	Flankenansprechung
106		*	**00.00x**	Wiederauslösbarkeit
107		*	**00.00x**	Ausgangszustand
108		*	**00.xxx**	Haltezeit des Monoflops in Sekunden

2.22 Programmierbares Monoflop (Timer)

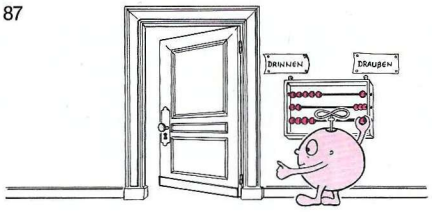
Das ist ein Programm für „Elektronik-Experten", denen bewiesen werden soll, daß man Zeitverzögerungen (und um solche handelt es sich ja bei Monoflops) auch mittels Programm erzeugen kann. Es simuliert alle gängigen Monoflops: auf positive oder negative Flanken reagierend (also auf 0 → 1 und 1 → 0 Übergänge), wieder auslösbar während der Haltezeit oder auch nicht, Ausgang aktiv auf „1" oder auf „0". Die Betriebsweise dieses programmierbaren Timers wird in den Zellen 105, 106 und 107 eingestellt, bevor man das Programm startet:

105	Flankenansprechung	„0" = negative „1" = positive
106	Wiederauslösbarkeit	„0" = nein „1" = ja
107	Ausgangszustand	„0" = aktiv „0" „1" = aktiv „1"

In Zelle 108 wird eingegeben, wie lange die Haltezeit des Monoflop in Sekunden sein soll.
Der Eingangspuls wird an Port 1/Klemme 1 gelegt, der Ausgang ist Port 2/Klemme 1, wo eine Leuchtdiode angeschlossen werden kann. Während der Haltezeit wird der verbliebene Rest in Sekunden auf der Anzeige sichtbar gemacht.
Noch eine Spezialität: Sogar ein Lösch- oder Reset-Eingang ist im Mikroprozessor-Monoflop enthalten: eine „0" an Port 1/Klemme 8 stellt sofort den Ruhezustand wieder her.

2.23 Personenzählanlage mit Automatikschalter

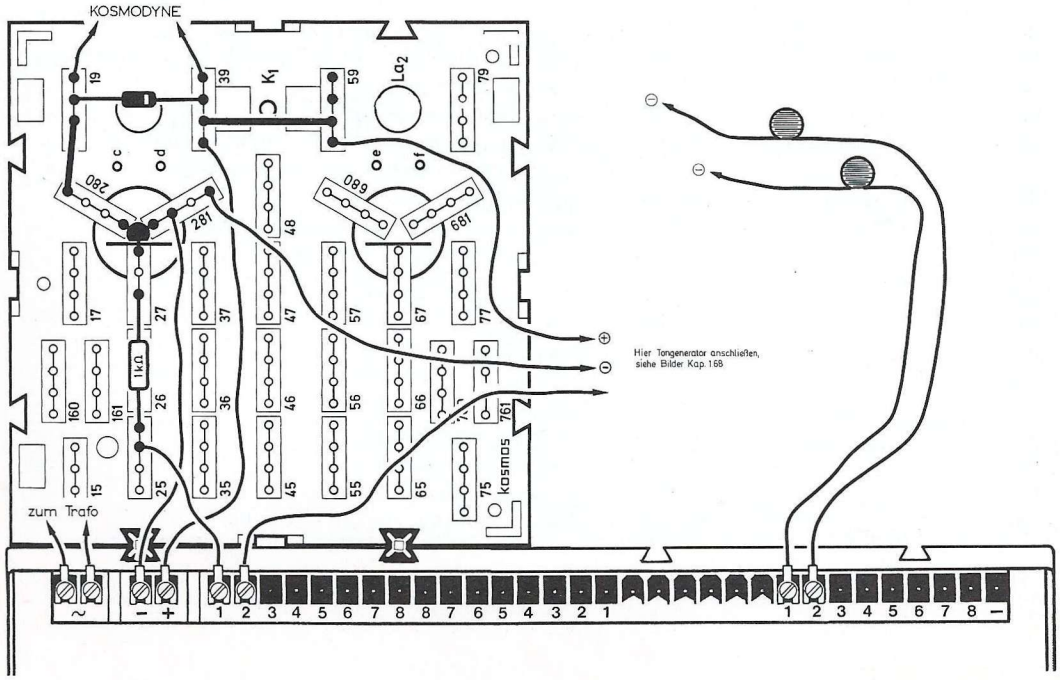
Bild 87



Das Programm zählt die einen Raum betretenden Personen. Bei der ersten Person wird das Licht eingeschaltet; sobald die letzte Person den Raum verläßt, wird das Licht wieder ausgeschaltet. In Zelle 106 kann dem Rechner die bei Programmstart anwesende Personenzahl mitgeteilt werden.
An elektronischen Bauteilen außerhalb des Computers brauchen wir:

2 Lichtschranken. Eine „1" am Ausgang der Lichtschranken bedeutet „Lichtschranke nicht unterbrochen", eine „0" bedeutet „Lichtschranke unterbrochen". Der Abstand der Lichtschranken darf nur so groß sein, daß die 2. Lichtschranke unterbrochen wird, solange die 1. noch unterbrochen ist.

Bild 88. Personenzählanlage, aufgebaut aus KOSMOS Elektronik-Material und Schaltrelais KOSMODYNE B (Schaltplan dazu Bild 89). Tongenerator, falls gewünscht, wie Bild 61



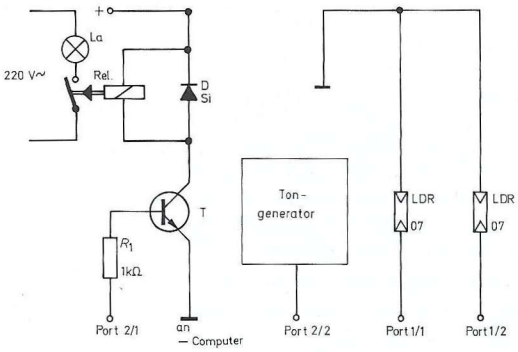


Bild 89

1 Tongenerator. Eine „0" am Eingang des Tongenerators bedeutet „ausgeschaltet", eine „1" bedeutet „eingeschaltet".
1 Relais mit Treibertransistor, um den Lichtstromkreis zu schalten. Eine „0" bedeutet „Licht aus", eine „1" bedeutet „Licht ein".

Es werden die Klemmen 1 und 2 des Ports 1 benutzt. (Klemme 1: äußere Lichtschranke, Klemme 2: innere Lichtschranke.)
An den Klemmen 1 und 2 des Ports 2 werden der Tongenerator (Klemme 2) und das Lichtrelais (Klemme 1) angeschlossen.

Listing 50: **Personenzählanlage mit Automatikschalter**

Adresse	Mnemonics	**Code**	Kommentar
001	AKO 000	**04.000**	lade „0" . . .
002	ABS 100	**06.100**	und speichere sie als Vergleichszahl
003	AKO 001	**04.001**	lade „1" . . .
004	ABS 101	**06.101**	und speichere sie als Schrittweite
005	AKO 255	**04.255**	lade „255" . . .
006	ABS 102	**06.102**	und speichere sie als Lichtschrankenvergleich
007	AKO 254	**04.254**	lade „254"
008	ABS 103	**06.103**	und speichere sie als Lichtschrankenvergleich
009	AKO 252	**04.252**	lade „252" . . .
010	ABS 104	**06.104**	und speichere sie als Lichtschrankenvergleich
011	E0: LDA 106	**05.106**	lade Personenzahl in den Akku . . .
012	ANZ	**02.000**	und zeige sie an
013	VGL 100	**10.100**	ist die Zahl „0"?
014	AKO 000	**04.000**	nimm an, das Licht sei aus: lade „0"
015	SPB 017 (E2)	**11.017**	wenn niemand da, direkt zur Relaisansteuerung
016	NEG	**14.000**	sonst mache aus der „0" eine „1" (Licht an!)
017	E2: P2A 001	**18.001**	Ausgabe des Akkuinhalts am Lichtrelais
018	E1: P1E 000	**16.000**	lies Information von Port 1 in den Akku
019	VGL 102	**10.102**	ist keine Lichtschranke unterbrochen, also „255" gelesen?
020	SPB 018 (E1)	**11.018**	falls ja, weiter warten
021	VGL 103	**10.103**	ist Port 1 auf „254" (also äußere Schranke unterbrochen)?
022	AKO 001	**04.001**	lade die „1" mal schon als Schalter
023	SPB 025 (E3)	**11.025**	ergab der vorletzte Befehl Gleichheit, springe zum Speichern
024	NEG	**14.000**	ist es aber die innere (d.h. „253"), merke Dir eine „0"
025	E3: ABS 105	**06.105**	speichere die unterbrochene Lichtschranke: 1 = außen 0 = innen
026	E4: P1E 000	**16.000**	lies Information von Port 1 in den Akku
027	VGL 102	**10.102**	sind beide Lichtschranken frei (also falscher Alarm = „255")?
028	SPB 018 (E1)	**11.018**	falls ja, war alles eine Ente. Weiter bei 018
029	VGR 104	**12.104**	ist die Information größer als „252", also nur eine Lichtschranke unterbrochen?
030	SPB 026 (E4)	**11.026**	dann warte auf die Unterbrechung beider oder die Freigabe der einen unterbrochenen – bei 026.
031	B1: P1E 000	**16.000**	lies die Information von Port 1
032	VGL 104	**10.104**	sind noch beide unterbrochen, also „252" gelesen?

Fortsetzung siehe nächste Seite

Fortsetzung Listing 50:

Adresse		Mnemonics	Code	Kommentar
033		SPB 031 (B1)	**11.031**	falls ja, warte noch, bis *eine* frei wird
034		VGL 103	**10.103**	sonst: ist die *äußere* noch unterbrochen?
035		AKO 000	**04.000**	lade „0" – die *äußere* wird später freigegeben
036		SPB 038 (B2)	**11.038**	falls ja, springe zu 038 – vergleiche die Null
037		NEG	**14.000**	sonst mache aus der „0" eine „1": die innere wird später frei
038	B2:	VGL 105	**10.105**	ist die Ende-Information gleich der Beginn-Information?
039		SPB 041 (B3)	**11.041**	falls ja, weiter beim Addieren/Subtrahieren
040		SPU 018 (E1)	**09.018**	sonst mache bei 018 weiter – alles falscher Alarm!
041	B3:	VGL 101	**10.101**	enthält der Akku eine „1", also Eintritt einer Person?
042		LDA 105	**05.106**	lade die Personenzahl
043		SPB 048 (A1)	**11.048**	falls ja, springe zum Addieren
044		VGL 100	**10.100**	sind null Personen im Raum?
045		SPB 049 (A2)	**11.049**	dann spare Dir die Subtraktion
046		SUB 101	**08.101**	sonst verringere die Zahl um „1"
047		SPU 049 (A2)	**09.049**	springe nach 049
048	A1:	ADD 101	**07.101**	hier kommt nun eine Person dazu
049	A2:	ABS 106	**06.106**	speichere die Personenzahl wieder
050		AKO 001	**04.001**	lade „1" (zum Einschalten des Tongenerators)
051		P2A 002	**18.002**	ausgeben am Port 2/Klemme 2
052		VZG 250	**03.250**	warte 1/4 Sekunde
053		NEG	**14.000**	dann lade „0" durch Invertierung der „1" im Akku
054		P2A 002	**18.002**	gib sie am Tongenerator aus: Abschalten des Tons
055		SPU 011 (E0)	**09.011**	dann warte auf die nächste Person
100			**00.000**	„0" als Vergleichszahl
101			**00.001**	„1" als Schrittweite
102			**00.255**	Vergleichszahlen für die Lichtschranken
103			**00.254**	
104			**00.252**	
105			**–**	Zustand der Lichtschranke
106		*	**00.xxx**	Personenzahl im Raum

2.24 Computergesteuerter Modellbahnhof

Da der KOSMOS-Computer ein kleiner Prozeßrechner ist, kann er auch eine Modellbahnanlage automatisieren. Dies ist aber nur mit einer Zusatzelektronik – einem „Interface" – möglich, um auch Wechselstromanlagen zu steuern (KOSMOS-Relais-Interface, in Vorbereitung). Als Beispielprogramm haben wir eine Bahnhofsanlage mit vier Gleisen, die vom Computer freigegeben bzw. gesperrt werden, entworfen (Bild 90). Dazu brauchen wir fünf Kontaktgleisstücke K0 bis K4 und können drei Weichen W1 bis W3 sowie das Einfahrtssignal S0 damit steuern. Die Weichen und Signale müssen übrigens allesamt Magnetartikel sein, die mittels elektrischer Impulse umgeschaltet werden können, welche über die Relais vom Computer gesteuert werden (Bild 91). Die Bahnhofssignale S1 bis S4 werden von Hand (also am Stellpult) gestellt; das Programm merkt, ob ein Zug im Gleis steht oder nicht!

Anschlußtabelle:

Kontaktgleis	K0	Port 1/1
Kontaktgleis	K1	Port 1/2
Kontaktgleis	K2	Port 1/3
Kontaktgleis	K3	Port 1/4
Kontaktgleis	K4	Port 1/5
Weiche	W1	Port 2/1 + 2 (gerade + krumm)
Weiche	W2	Port 2/3 + 4 (krumm + gerade)
Weiche	W3	Port 2/5 + 6 (krumm + gerade)
Signal	S0	Port 2/7 + 8 (auf + zu)

Was passiert im Programm?
Ein Zug, der über K0 fährt, wird in ein freies Gleis des Bahnhofs dirigiert. Sind alle Gleise belegt, so

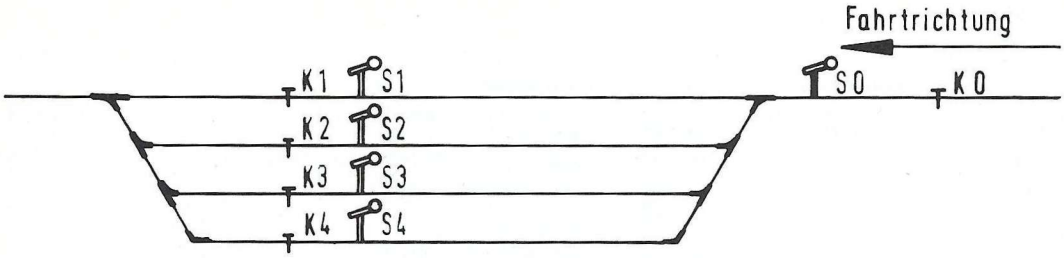


Bild 90. Gleisbild für computergesteuerten Modellbahnhof

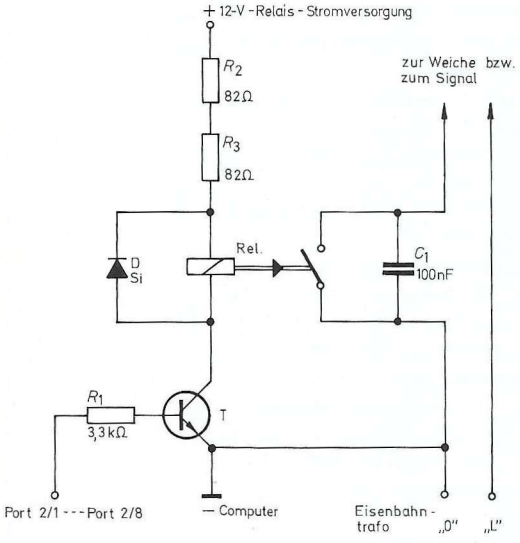


Bild 91. Eine Schaltstufe des KOSMOS Relais-Interface (für den hier vorgeschlagenen Aufbau ist das komplette KOSMOS Relais-Interface ausreichend)

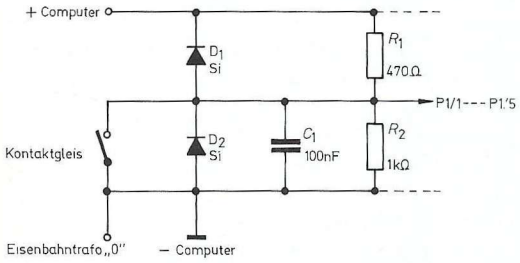


Bild 92. Eine Entstörstufe des KOSMOS Relais-Interface (siehe Text „Wichtige Hinweise")

wird er am Einfahrtssignal angehalten. Sobald ein Gleis frei wird, kann der blockierte Zug nachrücken. Die Anzeige zeigt immer die Anzahl der Züge im Bahnhof an; ist er jedoch blockiert, ist A 00.099 zu sehen.

Vor dem Programmstart muß in die Zellen 106–109 eingegeben werden, welche Gleise belegt sind (siehe Datenbereich, wie das zu tun ist). Außerdem müssen die Zellen 111–114 und 116–117 mit den angegebenen Werten gefüllt werden, um für eine ordnungsgemäße Weichenansteuerung zu sorgen. Die Werte und die Anschlußtabelle stimmen natürlich nur für die gezeichnete Weichenanordnung! Wer den Bahnhof etwas anders aufbaut, muß sich die Anschlüsse selbst überlegen.

Wichtige Hinweise:

1. Aus Kapitel 1.57 wissen Sie, daß der Computer beim Einschalten alle Port-Klemmen automatisch auf „1" setzt. Um die Weichen und das Signal vor Beschädigungen zu schützen, müssen alle Ausgänge vor dem Anschluß des Modellbahntrafos an das Stromnetz auf „0" gesetzt werden:

Adresse	Mnemonics	Code
001	AKO 000	04.000
002	P2A 000	18.000
003	HLT	01.000

Starten Sie dieses „Schutzprogramm" durch 001-PC-RUN, schließen Sie jetzt den Modellbahntrafo an das Netz an und geben Sie nun das Modellbahnprogramm ein.

2. Da fahrende Lokomotiven erhebliche Störungen (z.B. durch einen mäßig guten Kontakt zwischen Rädern und Schiene) produzieren, müssen alle Leitungen, die von der Modellbahnanlage in den Computer führen, **unbedingt** entstört sein. In Bild 92 sind die entsprechenden Entstörglieder so eingezeichnet, wie sie das KOSMOS-Relais-Interface (s. Kapitel 1.81) enthält.

Listing 51: **Computergesteuerter Modellbahnhof**

Adresse	Mnemonics	Code	Kommentar
001	AKO 000	**04.000**	lade „0"
002	ABS 127	**06.127**	speichere als Signalzustand (offen) . . .
003	ABS 100	**06.100**	und als Vergleichszahl
004	ADD 106	**07.106**	berechne die Zugzahl durch Addition aller „1" in den Speicherzellen für die Einzelgleise 106 – 109
005	ADD 107	**07.107**	
006	ADD 108	**07.108**	
007	ADD 109	**07.109**	
008	ABS 105	**06.105**	speichere die Gesamtzahl in 105 . . .
009	ANZ	**02.000**	und zeige sie an
010	AKO 001	**04.001**	lade „1" . . .
011	ABS 101	**06.101**	und speichere als Schrittweite und Vergleichszahl . . .
012	ABS 120	**06.120**	auch für das Kontaktgleis K0 (Einfahrtsgleis)
013	ADD 120	**07.120**	verdopple auf „2" . . .
014	ABS 121	**06.121**	als Vergleichszahl für Kontaktgleis K1
015	ADD 121	**07.121**	verdopple auf „4" . . .
016	ABS 122	**06.122**	als Vergleichszahl für Kontaktgleis K2
017	ADD 122	**07.122**	verdopple auf „8" . . .
018	ABS 123	**06.123**	als Vergleichszahl für Kontaktgleis K3
019	ADD 123	**07.123**	verdopple auf „16" . . .
020	ABS 124	**06.124**	als Vergleichszahl für Kontaktgleis K4
021	AKO 004	**04.004**	lade „4" als maximale Zugzahl in Bahnhof
022	VGL 105	**10.105**	ist die berechnete Zugzahl gleich der „4"?
023	SPB 077 (A6)	**11.077**	falls ja, springe zur Blockierung
			Steuerungsanfang:
024	A1: P1E 000	**16.000**	lies Port 1 (Kontaktgleiszustände) in den Akku
025	VGL 116	**10.116**	ist er „255", also kein Kontakt geschlossen?
026	SPB 024 (A1)	**11.024**	falls ja, weiter warten bei 024
027	ABS 102	**06.102**	sonst speichere den Wert kurz
028	LDA 116	**05.116**	lade „255" . . .
029	SUB 102	**08.102**	und subtrahiere das Gespeicherte davon
030	ABS 103	**06.103**	speichere es als Kontaktgleis-Eingabe
031	VZG 250	**03.250**	warte eine halbe Sekunde
032	VZG 250	**03.250**	zur Entprellung der Kontakte
033	AKO 000	**04.000**	lade „0" für das Einfahrtsgleis K0
034	A2: ABS 104	**06.104**	speichere sie als Kontaktnummer
035	AKO 120	**04.120**	lade „120" als den Beginn des Vergleichsdatenbereichs der Kontakte
036	ADD 104	**07.104**	addiere die zu testende Gleisnummer dazu . . .
037	ABS 126	**06.126**	und speichere dies als indirekte Adresse
038	LIA 126	**19.126**	lade so die Vergleichszahl für das Gleis
039	VGL 103	**10.103**	ist die Kontaktgleis-Eingabe gleich?
040	SPB 044 (A3)	**11.044**	falls ja, ist der richtige Kontakt gefunden
041	AKO 001	**04.001**	sonst lade „1" . . .
042	ADD 104	**07.104**	und erhöhe so die Kontaktnummer in 104
043	SPU 034 (A2)	**09.034**	springe zurück zum Speichern und Testen
			Auswertung des Kontaktes:
044	A3: LDA 104	**05.104**	gefunden! Lade die Kontaktnummer, wo der Kontakt stattfand
045	VGL 100	**10.100**	ist es „0", also K0, wo der Kontakt erfolgte?
046	SPB 068 (A4)	**11.068**	falls ja, suche ein freies Gleis (bei 068)
047	ADD 117	**07.117**	sonst addiere „105" zur Kontaktnummer

Fortsetzung siehe nächste Seite

Fortsetzung Listing 51:

048	ABS 126	**06.126**	speichere als Adresse für das Gleis
049	AKO 000	**04.000**	lade „0" (Gleis ist jetzt frei)
050	AIS 126	**20.126**	indirekt speichern im adressierten Gleis
051	LDA 105	**05.105**	lade die bisherige Zuganzahl
052	SUB 101	**08.101**	reduziere sie um „1" . . .
053	ABS 105	**06.105**	speichere sie wieder . . .
054	ANZ	**02.000**	und zeige sie an
055	VZG 250	**03.250**	warte kurz, bis die Lok über den Kontakt ist
056	LDA 127	**05.127**	lade den Signalzustand von S0
057	VGL 100	**10.100**	ist das Signal offen, also „0"?
058	SPB 024 (A1)	**11.024**	falls ja, warte auf eine neue Eingabe
059	VZG 250	**03.250**	sonst ist eine „1" im Akku; dann warte bis zur Zugausfahrt aus dem Bahnhof eine Dreiviertelsekunde
060	VZG 250	**03.250**	
061	VZG 250	**03.250**	
062	P2A 007	**18.007**	gib die „1" ans Signal aus: Öffnen! Der bisher blockierte Zug kann jetzt einfahren.
063	VZG 050	**03.050**	warte 50 ms als Schaltdauer des Signals
064	NEG	**14.000**	mache aus der „1" eine „0" durch Negation . . .
065	P2A 007	**18.007**	und schalte das Signalrelais wieder aus
066	ABS 127	**06.127**	speichere die „0" als Signalzustand: offen!
067	SPU 086 (A7)	**11.086**	Sprung zur Zugeinfahrt in den Bahnhof
			Suche nach freiem Gleis:
068	A4: AKO 110	**04.110**	lade „110" als Startadresse für die Gleissuche
069	A5: SUB 101	**08.101**	verringere die Adresse im Akku um „1" . . .
070	ABS 126	**06.126**	und speichere die Adresse des Gleises
071	LIA 126	**19.126**	lade indirekt den Zustand dieses Gleises
072	VGL 100	**10.100**	ist er „0", das Gleis also frei?
073	SPB 086 (A7)	**11.086**	falls ja, soll der Zug hier hinein!
074	LDA 126	**05.126**	sonst lade die Adresse wieder
075	VGR 117	**12.117**	ist sie noch größer als „105"?
076	SPB 069 (A5)	**11.069**	falls ja, springe zurück zur „Leer-Abfrage"
			Blockierung des Bahnhofs:
077	A6: AKO 001	**04.001**	sonst sind alle voll! Lade „1"
078	ABS 127	**06.127**	speichere es als Signalzustand: zu! . . .
079	P2A 008	**18.008**	und gib es am Signalrelais aus: Schließen!
080	VZG 050	**03.050**	warte kurz zum Umschalten
081	NEG	**14.000**	negiere die „1" zur „0"
082	P2A 008	**18.008**	gib sie aus: Ende des Impulses für das Relais
083	AKO 099	**04.099**	lade „99" . . .
084	ANZ	**02.000**	und zeige sie an: Einfahrtsgleis ist blockiert!
085	SPU 024 (A1)	**09.024**	springe zu 024 und warte auf einen neuen Kontakt
			Zugeinfahrt:
086	A7: AKO 001	**04.001**	lade „1" (Gleis ist belegt)
087	AIS 126	**20.126**	und speichere dies im adressierten Gleis
088	ADD 105	**07.105**	addiere zur „1" die Zugzahl im Bahnhof . . .
089	ABS 105	**06.105**	und speichere die neue Zugzahl wieder
090	ANZ	**02.000**	zeige sie außerdem an
091	AKO 005	**04.005**	lade „5"
092	ADD 126	**07.126**	addiere dazu die Adresse des Gleises
093	ABS 126	**06.126**	speichere als Adresse für die Weichensteuerung
094	LIA 126	**19.126**	lade damit den richtigen Code für die Weichen . . .
095	P2A 000	**18.000**	und gib ihn an Port 2 (Weichenrelais) aus
096	VZG 050	**03.050**	für 50 Millisekunden

Fortsetzung siehe nächste Seite

Fortsetzung Listing 51:

Adresse	Mnemonics	Code	Kommentar
097	AKO 000	**04.000**	dann lade „0" . . .
098	P2A 000	**18.000**	und gib sie aus: alle Relais werden abgeschaltet
099	SPU 024 (A1)	**09.024**	springe in die Warteschleife für den nächsten Kontakt
100		**00.000**	„0" als Vergleichszahl
101		**00.001**	„1" als Vergleichszahl und Schrittweite
102		–	Hilfszelle
103		–	Port 1 – Kontaktgleis-Eingabe
104		–	Gleisnummer, wo Kontakt erfolgte
105		–	Anzahl der Züge im Bahnhof
106	*	**00.00x**	Zustand Gleis 1: „0" = leer, „1" = voll
107	*	**00.00x**	Zustand Gleis 2: „0" = leer, „1" = voll
108	*	**00.00x**	Zustand Gleis 3: „0" = leer, „1" = voll
109	*	**00.00x**	Zustand Gleis 4: „0" = leer, „1" = voll
110			frei
111	*	**00.001**	Weichen-Code für Einfahrt in Gleis 1
112	*	**00.006**	Weichen-Code für Einfahrt in Gleis 2
113	*	**00.026**	Weichen-Code für Einfahrt in Gleis 3
114	*	**00.106**	Weichen-Code für Einfahrt in Gleis 4
115			frei
116	*	**00.255**	Vergleichszahl für Kontakt-Eingabe
117	*	**00.105**	Summand für Adressverschiebung
118			frei
119			frei
120		**00.001**	Vergleichszahl für Kontaktgleis K0
121		**00.002**	Vergleichszahl für Kontaktgleis K1
122		**00.004**	Vergleichszahl für Kontaktgleis K2
123		**00.008**	Vergleichszahl für Kontaktgleis K3
124		**00.016**	Vergleichszahl für Kontaktgleis K4
125			frei
126		–	Zelle für indirekte Adressierung
127		–	Signalzustand S0: „0" = offen/„1" = zu

2.25 Roulette

Bei diesem Programm „rollt" statt der Kugel ein Lichtpunkt über eine Reihe von 8 Leuchtdioden. Wie wir die LEDs über Treiber-Transistoren ansteuern müssen, zeigt das Schaltbild.

Auf dem Bild ist auch zu sehen, was wir sonst noch brauchen: einen Taster an Port 1/1, über den wir „die Kugel ausrollen" lassen können. Der Lichtpunkt wird immer langsamer, bis er schließlich auf einer LED liegenbleibt. Zusätzlich können wir an Port 1/2 (über einen Treibertransistor) einen Lautsprecher anschließen, so daß beim Rollen Geräusche erzeugt werden; dies ist jedoch für die Funktion des Programms nicht unbedingt erforderlich.

Gestartet wird es bei **001.** Es läßt den Punkt so lange über alle 8 LEDs flitzen, bis die Taste gedrückt wird; dann folgen noch 40 weitere Schritte, die immer langsamer werden, bis der Lichtpunkt ruht. Ein neuer Start ist dann einfach durch RUN möglich.

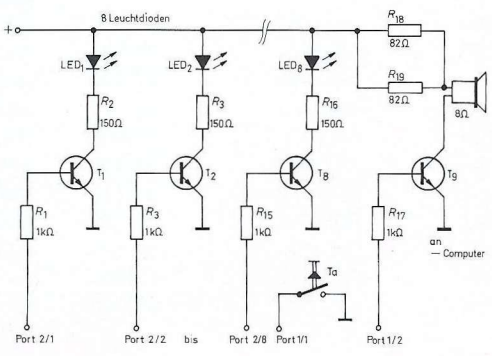


Bild 93

Listing 52: **Roulette**

Adresse		Mnemonics	Code	Kommentar
001	C0:	AKO 040	**04.040**	lade „40" . . .
002		ABS 124	**06.124**	als Startwert für Verlangsamung speichern
003		AKO 001	**04.001**	lade „1" . . .
004		ABS 125	**06.125**	als erstes LED-Muster speichern (dual: 00000001) . . .
005		ABS 126	**06.126**	als ersten Verlangsamungswert speichern . . .
006		ABS 127	**06.127**	als Schrittweite speichern . . .
007		ABS 123	**06.123**	und als Tastenzustand: „1" = frei (nicht gedrückt)
008	C1:	LDA 125	**05.125**	lade das LED-Muster
009		P2A 000	**18.000**	gib es aus an Port 2 (LED-Reihe)
010		AKO 129	**04.129**	Test, ob LED Nr. 8 erreicht: lade „129"
011		SUB 125	**08.125**	subtrahiere das Muster davon, also maximal 10000000 (dual) = 128 (dezimal)
012		VGR 127	**12.127**	ist der Rest größer als „1", also „128" als Muster noch nicht erreicht?
013		SPB 016 (C2)	**11.016**	falls ja, weiter in der Bit-Shift-Schleife
014		AKO 001	**04.001**	sonst neuer Musterstart: lade „1"
015		SPU 018 (C3)	**09.018**	springe zum Speichern als LED-Muster
016	C2:	LDA 125	**05.125**	Multiplikation mit „2": lade das Muster
017		ADD 125	**07.125**	addiere es noch einmal dazu
018	C3:	ABS 125	**06.125**	speichere es als neues Muster
019		AKO 001	**04.001**	lade „1"
020		P1A 002	**17.002**	gib sie aus an Port 1 Klemme 2 (Lautsprecher)
021		VZG 015	**03.015**	warte kurz (Lautsprecher ist jetzt an)
022		AKO 000	**04.000**	lade „0"
023		P1A 002	**17.002**	gib sie aus an Port 1 Klemme 2 (Lautsprecher aus)
024		LDA 126	**05.126**	lade den aktuellen Verlangsamungswert aus dem Speicher . . .
025	W1:	VZG 003	**03.003**	und verzögere kurz
026		SUB 127	**08.127**	verringere die Verlangsamung um „1"
027		VGR 127	**12.127**	ist sie noch größer als „1", also noch nicht lange genug gewartet?
028		SPB 025 (W1)	**11.025**	falls ja, warte weiter in der Schleife
029		P1E 001	**16.001**	sonst lies Port 1 Klemme 1 (Taster)
030		VGL 127	**10.127**	ist der Wert „1", also nicht gedrückt?
031		SPB 034 (W2)	**11.034**	falls ja, springe zum „Weiterrollen"
032		ABS 123	**06.123**	sonst speichere die „0" als Tastenzustand und Zeichen für das Ausrollenlassen.
033		SPU 037 (W3)	**09.037**	springe zu 037 (starte die Verlangsamung)
034	W2:	AKO 001	**04.001**	lade „1"
035		VGL 123	**10.123**	ist der Tastenzustand gleich?
036		SPB 008 (C1)	**11.008**	falls ja, springe zur Muster-Ausgabe
037	W3:	LDA 126	**05.126**	sonst verlangsame die „Rollbewegung": lade den aktuellen Verlangsamungswert
038		ADD 127	**07.127**	erhöhe ihn um „1"
039		ABS 126	**06.126**	speichere ihn wieder: Die Schleife 025 – 028 benötigt jetzt mehr Zeit („rollt langsamer")
040		ANZ	**02.000**	Anzeigen des Verlangsamungswertes
041		VKL 124	**13.124**	ist er noch kleiner als der Startwert „40"?
042		SPB 008 (C1)	**11.008**	falls ja, springe zur Muster-Ausgabe
043		HLT	**01.000**	sonst HALT – die Kugel liegt fest
044		SPU 001 (C0)	**09.001**	neuer Start – ein neues Glück bei 001
123			**–**	Tastenzustand: „1" = frei „0" = gedrückt gleichzeitig Zeichen für das „Ausrollen"
124			**00.040**	„40" als Schrittzahl bis zum Stillstand
125			**–**	LED-Muster (Startwert: 00000001 = „1")
126			**–**	aktueller Verlangsamungswert
127			**00.001**	Zählerschrittweite

Dritter Teil
Einige Grundprinzipien der elektronischen Datenverarbeitung

3.1 Was geht in den Schaltkreisen des Computers vor?

Sie haben bisher erfahren, daß der Computer Zahlenwerte speichert, in den Akku lädt, anzeigt, addiert und subtrahiert, daß Befehle decodiert und ausgeführt, Steuersignale ein- und ausgegeben werden. Was Sie als Computer-Experte jedoch nicht unbedingt zu wissen brauchen: wie kann man eigentlich elektronisch eine Addition durchführen, aus welchen elektronischen Grundelementen besteht ein Computer und auf welche Weise können hochkomplizierte Aufgaben, die ein Computer zu lösen instande ist, mit den bereits aus Kap. 1.59 bekannten logischen Zuständen „0" und „1" bewerkstelligt werden?
Ein paar Gedanken zum Thema „Digital-Elektronik" (lat. digitus = Finger; bei der Digitaltechnik erfolgt alles schrittweise, wie beim Abzählen der Finger. Unser Computer ist ein Digitalcomputer) möchten wir Ihnen aber dennoch nahebringen, als Anregung sozusagen, sich auch einmal mit der Hardware-Seite der Computertechnik zu befassen.

3.2 Alles im Zweiersystem

Folgendes ist Ihnen bereits bekannt: Digitalcomputer arbeiten binär, kommen also mit zwei Schaltzuständen aus.
1. elektrische Spannung vorhanden
2. keine Spannung vorhanden.

Bei Gebrauch unserer Ein- und Ausgänge haben Sie diese Arbeitsweise erfahren. Die beiden Zustände wurden mit „1" und „0" bezeichnet.
Wie aber läßt sich mit nur zwei verschiedenen Zuständen eine schwierige Aufgabe überhaupt bewältigen?
Welche Möglichkeiten im Zweiersystem stecken, zeigt unter anderem das Morsealphabet. Nur mit Punkt und Strich, also mit zwei Zeichen, lassen sich damit Texte jeder Art schreiben. Übrigens kommt auch unser Gehirn – davon kann man heute ausgehen – mit nur zwei digitalen Zuständen bei seiner Arbeit aus: Seine Gehirnzellen sind entweder erregt oder nicht erregt. Allerdings verfügt unser Gehirn über die unvorstellbar große Menge von 10 Milliarden Gehirnzellen, die in vielfältiger Weise miteinander verknüpft werden können. Dadurch wird seine enorme Leistung ermöglicht.
Auch die Digitalcomputer bestehen aus vielen Tausenden von Grundbausteinen, die sich mit Gehirnzellen vergleichen lassen und die jeweils zwei verschiedene Zustände annehmen können. Sie arbeiten also im Prinzip wie ein Schalter, mit dem wir zum Beispiel eine Lampe ein- und ausschalten. Ein solcher Schalter ist gleichzeitig ein Speicher für eine Informationseinheit, für 1 bit. Er hält seinen jeweiligen Zustand „ein" oder „aus" solange fest, bis er wieder betätigt wird.

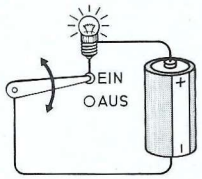
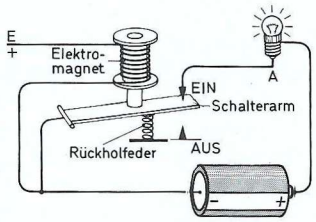


Bild 94

3.3 Das Grundelement des Computers: ein einfacher Schalter

Handbediente Schalter kommen natürlich für einen Computer nicht in Frage. Sie müssen vielmehr elektrisch funktionieren. Dies läßt sich z. B. mit einem Elektromagnet erreichen, wie abgebildet.

Bild 95



Wird durch diesen Elektromagnet Strom geschickt, dann zieht er den Schalterarm an, der Kontakt wird geschlossen, und das Lämpchen brennt. Man kann auch sagen: legt man an den **Eingang** E, also an die Spule eine elektrische Spannung, dann ist auch am **Ausgang** A, also am Lämpchen eine Spannung vorhanden, so daß es leuchtet.
Ein solcher elektromechanischer Schalter, wie wir ihn hier beschrieben haben, heißt **Relais.**
Bei den Vorgängern unseres Computers hat

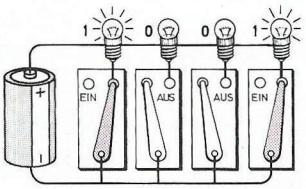
man vor vierzig Jahren tatsächlich Relais eingesetzt, die wegen ihres großen Platzbedarfs, hohen Stromverbrauchs und langsamen Arbeitstempos für die heutige Datenverarbeitung natürlich völlig ungeeignet wären. Deswegen werden seit etwa 25 Jahren Transistoren als Schalter verwendet, die ebenfalls zwei Ausgangszustände besitzen: Sie lassen entweder Strom durchfließen oder sperren den Durchfluß. Transistoren können heute so mikroskopisch klein hergestellt werden, daß mehrere Zehntausend davon auf der Fläche eines Quadratzentimeters Platz finden.

Wer sich in der Elektronik etwas auskennt (z.B. die Besitzer von KOSMOS Elektronik-Experimentierkästen), weiß, daß man mit zwei Transistoren ein Speicherelement für ein Bit bauen kann. Man nennt dieses Element **Flip-Flop** oder **bistabilen Multivibrator.** Wer kein Elektronikexperte ist, braucht sich nur merken, daß bei solchen Flip-Flops ein geringer Stromstoß, ein Stromsignal genügt, um die Schaltfunktion zu bewirken. Dieses Signal kann z.B. durch Druck auf eine Taste oder aber im Computer selbst erzeugt werden. Das Flip-Flop reagiert sofort und ändert wie ein einfacher Schalter den Zustand an seinem Ausgang von „Spannung vorhanden" zu „keine Spannung" oder umgekehrt.

3.4 Vom Schalter zum Register

Mit einem Speicherelement allein ist natürlich nicht viel anzufangen. Es reicht gerade aus, um ein Lämpchen einzuschalten. Aber schon mit 2 Schaltelementen kann man 4, mit 3 Schaltern 8 und mit 4 Schaltern 16 verschiedene Ausgabemöglichkeiten erzeugen. Bei 8 Schaltern, also bei 8 bit kommen wir, wie Sie sich erinnern werden (Kapitel 1.63), bereits auf 256. Faßt man also mehrere Schalter zu einem sogenannten **Register** zusammen, dann kann man in solch einer Speicherzelle schon allerhand unterbringen.

Bild 96



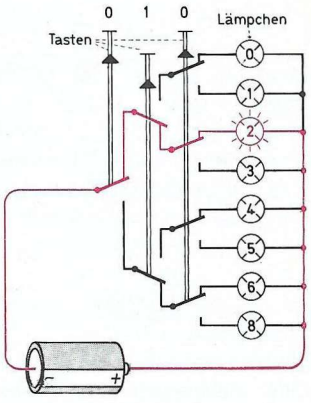
Wir haben hier auf der Abbildung vier Schalter zusammengefaßt, mit denen 16 verschiedene Ein-Aus-Kombinationen der vier Lämpchen möglich sind.

Eingestellt ist gerade die Dualzahl 1001 (dezimal = 9). Auch in unserem Computer müssen für jede der Ziffern 0 bis 9, überall wo sie gespeichert werden, mehrere solcher Flip-Flop-Schalter zur Verfügung stehen.

3.5 Codieren und Decodieren

Mit einfachen Schaltern und Lämpchen können wir also eine Dualzahl anzeigen. Dualzahlen sind aber für einen Ungeübten schwer zu lesen. Wer weiß z.B. auf Anhieb, daß dual 1001 der Dezimalzahl 9 entspricht? Deswegen wandelt unser Computer die Dualzahlen, mit denen er intern arbeitet, in Dezimalzahlen um, wie wir sie auf der Ziffernanzeige sehen. Wie läßt sich solch eine Umwandlung oder **Decodierung** bewerkstelligen? Auch das kann mit einem Schaltermodell gezeigt werden, von dem wir der Einfachheit halber eines für eine nur 3-stellige Dualzahl gezeichnet haben.

Bild 97



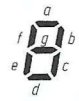
Drückt man von oben auf eine der abgebildeten Tasten, dann bedeutet dies „1". Nicht gedrückt ist „0". Das Drücken könnte auch je ein Elektromagnet (siehe 3.3) besorgen, dem dann Strom zugeführt werden muß. (Dann wäre 1 = Strom, 0 = kein Strom.)

Auf dem Bild ist die zweite der drei Tasten mit ihren zwei Schaltelementen gedrückt. Die Eingabe ist also **0 1 0,** und diese Dualzahl entspricht dem Wert „2" im Dezimalsystem. Das entsprechende Lämpchen in unserem Modell leuchtet daher.

In unserem Computer verwenden wir allerdings keine einzelnen Lämpchen für die Anzeige der Zahlen, sondern sogenannte Sieben-Segment-Leuchtanzeigen. Jede Anzeige besitzt sieben Teile (Segmente) – auf unserem Bild mit a bis g

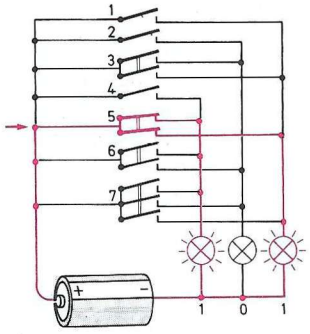
gekennzeichnet – mit denen, je nachdem man sie aufleuchten läßt, alle Ziffern von 0 bis 9 gebildet werden können.

Bild 98



Will man z.B. „1" anzeigen, dann müssen die Segmente b und c leuchten. Für „8" braucht man alle Segmente.
Wir haben gesehen, wie man von Dualzahlen zu Dezimalzahlen gelangen kann. Nun wollen wir uns überlegen, wie man Dezimalzahlen in Dualzahlen wandeln kann. Auf dem nächsten Bild erkennen wir sieben Schalter, z.T. mit Mehrfachfunktion, die mit drei Lämpchen für die Dualzahlen-Anzeige verbunden sind. Gedrückt ist der Schalter „5", was zur Anzeige **1 0 1,** führt. Sie können in Gedanken auch einmal andere Schalter drücken und feststellen, daß stets die richtige zugehörige Dualzahl erscheint.

Bild 99



Auch in unserem Computer muß jede von uns eingetippte Ziffer zunächst in Dualsignale umgewandelt werden, denn nur sie kann der Computer „verstehen". Diese Wandlung von einem Zahlensystem ins andere, auch Codieren und Decodieren genannt, läßt sich natürlich elektronisch wesentlich eleganter machen, als mit elektrischen Schaltern.

3.6 Die Verarbeitung der Daten

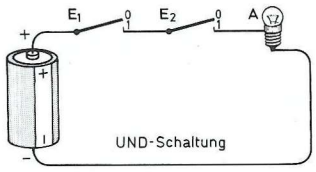
Wie man Daten speichert und codiert, können wir uns jetzt in etwa vorstellen. Wie aber werden Dualzahlen elektronisch zusammengezählt oder miteinander verglichen? Da der Computer nur die Größe 0 oder 1 kennt, besteht der Gipfel seiner Rechenkunst in der Formel $1 \times 1 = 1$. Wie die Tabelle zeigt, ergeben die übrigen Multiplikationsmöglichkeiten – wer will das bezweifeln – durchweg 0.

$$0 \times 0 = 0$$
$$0 \times 1 = 0$$
$$1 \times 0 = 0$$
$$1 \times 1 = 1$$

Erinnert Sie dies nicht an die Wahrheitstafel in Kapitel 1.59, wo die UND-Funktion erläutert wird? Wir können beim Vergleichen feststellen: die UND-Schaltung beherrscht das duale Einmaleins.

Zur Veranschaulichung dieser wichtigen logischen Grundbeziehung dient uns wieder ein Modell mit einfachen Schaltern: Nur dann, wenn **beide** Schalter geschlossen sind, geht die Lampe an.

Bild 100



Ein offener Schalter bedeutet jeweils 0, ein geschlossener 1. Werden die Schalter nach dem Beispiel in Kapitel 3.3. elektrisch betrieben, dann schließen sie, wenn durch ihre jeweiligen Elektromagnetspulen Strom geschickt wird. Es gilt dann folgendes: Liegt am Eingang (Schalter) E_1 **und** am Eingang (Schalter) E_2 Strom, dann ist auch am Ausgang A Strom vorhanden, der durch das brennende Lämpchen angezeigt wird.

Das läßt sich in einer Tabelle wie folgt darstellen:

Eingang E_1	Eingang E_2	Ausgang A
0	0	0
0	1	0
1	0	0
1	1	1

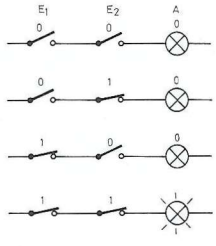
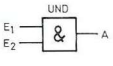


Bild 101

Die UND-Funktion kommt in Logik-Schaltungen häufig vor. Ihr Schaltsymbol sieht folgendermaßen aus:

Bild 102



Es können übrigens noch mehr als zwei Eingänge vorhanden sein. Auch dann gilt aber, daß an allen Eingängen Spannung anliegen muß, damit auch am Ausgang eine Spannung erscheint.

3.7 Duale Addition

Die soeben beschriebenen bescheidenen Multiplizierfähigkeiten reichen zur Lösung größerer Aufgaben nicht aus. Zum Glück kann man aber jede Multiplikation duch fortlaufendes Addieren ersetzen, wie wir es bei unserem Multiplikationsprogramm (Kapitel 1.69) getan haben. Auch alle Rechenmaschinen multiplizieren auf diese Weise. Subtrahiert und dividiert wird übrigens ebenfalls mit Hilfe der Additionsmethode, mit der wir uns daher näher befassen wollen.
Bei der dualen Addition sind die Rechenregeln ebenfalls sehr einfach. Sie sehen folgendermaßen aus:

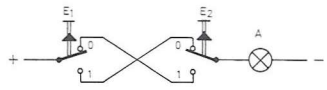
$$0+0=0$$
$$0+1=1$$
$$1+0=1$$
$$1+1=10$$

Wie wir sehen, wird das Ergebnis in der letzten Zeile größer als die größtmögliche Zahl des Dualsystems und muß daher als 10 geschrieben werden. Es gibt ja keine höhere Dualziffer als 1, so daß – wie Ihnen bereits bekannt – eine weitere Stelle benötigt wird, um den nächsthöheren Wert auszudrücken. Auch bei unserem Zehnersystem ist es nicht anders, wenn wir die größte Ziffer 9 erreicht haben und weiterzählen wollen.
Um zu verstehen, wie man elektrisch bzw. elektronisch eine Addition durchführen kann, gehen wir schrittweise vor und lassen zunächst einmal die bei der Addition von 1+1 zusätzlich entstandene linke Stelle, die auch **Übertrag** (englisch „carry") genannt wird, außer acht. Die zugehörige Wahrheitstabelle für die Addition sieht dann folgendermaßen aus:

Eingang E_1	Eingang E_2	Ausgang A
0	0	0
0	1	1
1	0	1
1	1	0

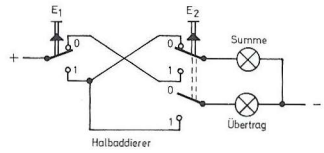
Ein Schaltermodell, das entsprechend dieser Tabelle funktioniert, ist einfach herzustellen, wie unsere Abbildung zeigt.

Bild 103



Wir sehen, daß immer dann am Ausgang eine „1" (Lampe an) erscheint, wenn die Eingänge ungleich sind, also **entweder** am einen Eingang **oder** am anderen eine „1" (Schalter gedrückt) liegt. Das Lämpchen leuchtet also nur dann, wenn eine der beiden Tasten E_1 oder E_2 gedrückt ist. Eine solche Verknüpfung wird **Antivalenz-** oder **XOR-Schaltung** genannt. XOR bedeutet „exclusiv ODER" (engl.: exclusiv OR).
Als nächstes müssen wir nun erreichen, daß auch der Übertrag berücksichtigt wird. Wie läßt sich das machen? Erweitern wir einfach den Schalter E_2 durch ein zusätzliches Schaltelement, wie auf der Abbildung zu sehen:

Bild 104

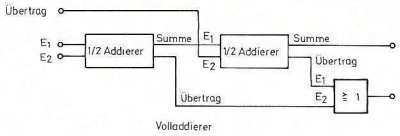


Drücken wir jetzt einmal in Gedanken beide Schalter E_1 und E_2, dann leuchtet das „Übertraglämpchen", und das „Summelämpchen" bleibt dunkel. Mit dieser Schaltung läßt sich also das Ergebnis 10 anzeigen. In der zugehörigen Wahrheitstabelle wird der logische Zusammenhang nochmals dargestellt:

Eingang E_1	Eingang E_2	Übertrag	Summe
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

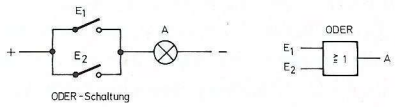
Eine solche Schaltung, wie wir sie soeben kennengelernt haben, wird als **Halbaddierer** bezeichnet. „Halb" deswegen, weil nur zwei Eingangsgrößen E_1 und E_2 verarbeitet werden können, sich aber ein aus einer davorliegenden Addierstufe eventuell vorhandener Übertrag nicht berücksichtigen läßt. Um auch dies zu erreichen, schalten wir einfach zwei Halbaddierer zusammen, wie hier dargestellt. Um die Zeichnung übersichtlicher zu machen, haben wir die Halbaddierer, deren Funktion Sie nun kennen, vereinfacht als Kästchen mit ihren Eingängen E_1 und E_2 sowie ihren beiden Ausgängen abgebildet. Das Ganze heißt **Volladdierer.**

Bild 105



Rechts unten in der Zeichnung erscheint noch ein Kästchen, das sich von dem uns bekannten UND-Symbol etwas unterscheidet. Es handelt sich um das Zeichen für eine **ODER-Schaltung.**

Bild 106



Ihr Ausgang wird stets 1, wenn an einem oder an beiden Eingängen ein „1"-Signal liegt. Das machen wir uns wieder mit einem Schalterbeispiel und der zugehörigen Wahrheitstabelle klar.

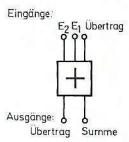
E_1	E_2	A
0	0	0
0	1	1
1	0	1
1	1	1

Sie können nun wieder in Gedanken verfolgen, was beim Anlegen einer elektrischen Spannung an die Eingänge E_1 und E_2 und an den Übertragseingang geschieht. Liegt z.B. sowohl am Übertragseingang als auch an den beiden Eingängen E_1 und E_2 eine 1, dann erscheint an beiden Ausgängen ebenfalls eine 1, also 11, was einer 3 im Dezimalsystem entspricht.
Falls Sie sich nicht intensiver mit Digitalelektronik befassen wollen, brauchen Sie sich die Funktionsweise des Volladdierers und des nachfolgend beschriebenen Rechenwerks nicht unbedingt zu merken. Sie sollen lediglich eine ungefähre Vorstellung davon erhalten, wie im Computer Aufgaben durch eine Kombination sinnvoll miteinander verbundener einfacher Schalter gelöst werden können.

3.8 Das vollständige Rechenwerk

Nun haben wir zwar einen Volladdierer, können aber trotzdem erst zwei einstellige Dualzahlen zusammenzählen. Für jede weitere Stelle braucht man zusätzlich einen Volladdierer, wie die nächste Abbildung zeigt. Damit auch hier die Sache nicht zu unübersichtlich wird, haben wir für den Volladdierer ebenfalls eine vereinfachte Darstellung gewählt. Er ist als Kästchen mit fünf Anschlüssen gezeichnet.

Bild 107



Das abgebildete **Rechenwerk** kann zwei dreistellige Dualzahlen zusammenzählen und unter Berücksichtigung des Übertrags ein vierstelliges Ergebnis anzeigen. Die beiden Zahlen, die zusammengezählt werden sollen, also die Summanden, sind je in einem Register untergebracht und dort – wie Sie in den vorhergehenden Kapiteln erfahren haben – in Flip-Flops gespeichert, von denen für jede Stelle eines vorhanden ist. Wenn Sie jetzt den Verarbeitungsweg verfolgen, werden Sie feststellen, daß sich mit solch einem Rechenwerk dreistellige Dualzahlen addieren lassen. Sollen Zahlen mit mehr Stellen zusammengezählt werden, dann müssen die Zahlenspeicher erweitert und zusätzliche Addierer für jede weitere Stelle eingebaut werden. Auf unserem Bild findet eine Addition der Zahlen 5 (dual 101) und 3 (dual 011) statt.

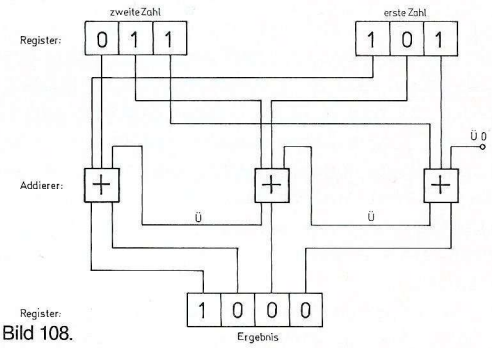


Bild 108.

3.9 Schlußbetrachtung

Mit diesen wenigen Bemerkungen über die elektronische Datenverarbeitung hoffen wir, Ihnen den Computer etwas besser durchschaubar gemacht zu haben. Noch näher auf die Schaltungsdetails einzugehen, würde den Rahmen dieses Buches sprengen, zumal die Darstellung der gesamten Schaltung mit der umfangreichen Verdrahtung ihrer riesigen Mengen von Schaltelementen sogar das Auffassungsvermögen von Fachleuten übersteigt und daher in langen Computerprogrammen enthalten ist.
Wer sich noch intensiver mit Digitalelektronik befassen möchte, kann sich mit entsprechenden Fachbüchern weiterbilden.

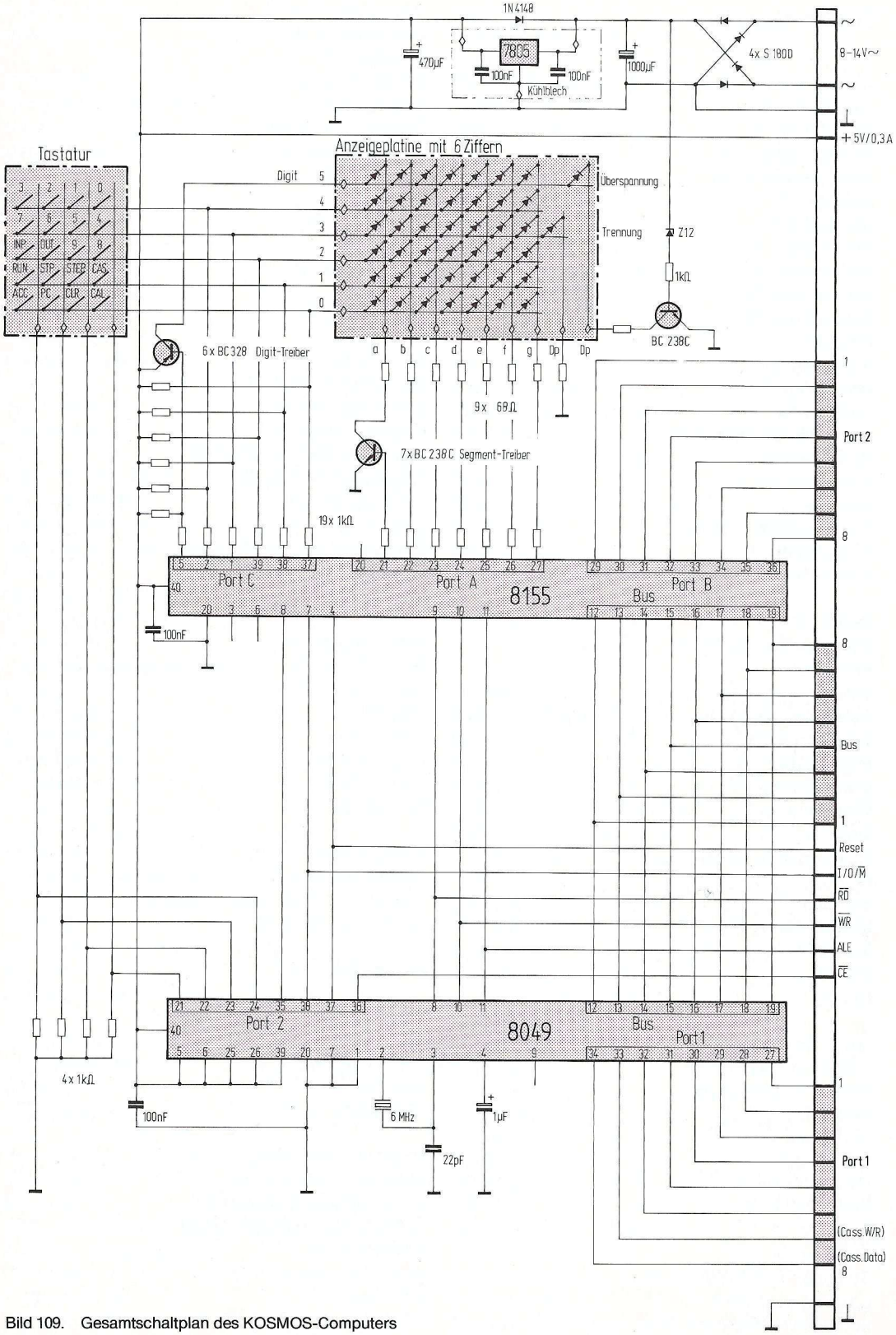


Bild 109. Gesamtschaltplan des KOSMOS-Computers

Sachregister

Kosmos Elektronik-Labor E 200 Super Labor

Eine komplette, unübertroffene Elektronik-Ausrüstung für Anfänger und Fortgeschrittene. In über 200 Versuchen und Schaltungen lernt man die faszinierende Welt der Elektronik kennen, z. B. Transistor- und Operationsverstärker-Technik, NF-Technik, Radiotechnik, Opto-Elektronik, Meßtechnik, spezielle HF-Technik. Die komplette Labor-Ausrüstung besteht aus mehr als 135 Einzelteilen und einem ausführlichen leichtverständlichen Anleitungsbuch. Ein Experimentalkurs, der nicht nur Spaß macht, sondern auch ein breites Fachwissen vermittelt.

Und in Kombination mit der „Kosmos Computer-Praxis" ist eine Fülle von hochinteressanten computergesteuerten Anlagen für Regelung und Automation, Spielgeräte usw. möglich.

Bestell-Nr. 611811

Kosmos Labor-Ausbau E 201
Digital-Praxis
Praktische Digital-Schaltungen mit Ziffernanzeigen und Zählermodulen: Stoppuhr, elektronischer Würfel, Digital-Voltmeter, Buchstabenspiel, Digital-Thermometer, Lottozahlengenerator, Mini-Synthesizer und vieles mehr. Ein ausführliches Begleitbuch gibt praktikable Anleitungen.
Bestell.-Nr. 615211

Kosmos Labor-Ausbau E 202
Infrarot-Praxis
Dieser Kosmos Labor-Ausbau führt in die Geheimnisse der unsichtbaren Infrarot-Strahlen ein und demonstriert verblüffende Kombinationen von Elektronik und Optik. Fernsteuerung, Infrarot-Strahlentelefon, Entfernungsmeßgerät, unsichtbare Lichtschranke, Alarmanlage, Differenzlichtschalter sind nur ein Teil des faszinierenden Versuchsprogramms. Mit ausführlichem Anleitungsbuch.
Bestell-Nr. 615111

Kosmos Labor-Ausbau E 203
HiFi-Praxis
Der Aufbau eines 2 × 10 W Stereo-Verstärkers oder eines 1 × 20 W Monoverstärkers bringt Disco-Stimmung ins Haus. Eigenbau ist Trumpf für die Besitzer des E 200-Labors und dieser Ausbaustufe. Zusätzlich ist der Bau eines Mischpultes, eines Rhythmus-Gerätes, eines Leistungsindikators für Lautsprecher möglich. Und noch viele weitere Versuche bietet das ausführliche Anleitungsbuch.
Bestell-Nr. 615311

Kosmos Labor-Ausbau E 204
Bio-Elektronik
Winzige Spannungsimpulse, die man an der Hautoberfläche abnehmen kann, geben Auskunft über die Funktion unserer Organe. Diese Kleinstspannungen können nach gewaltiger Verstärkung hörbar und sichtbar gemacht werden. Die Elektronik im Dienste der Medizin führt hier bereits in den Grundlagen-Experimenten zu einer Fülle aufregender Entdekkungen. Ein Auszug aus dem Versuchsprogramm gibt Aufschluß über die vielfältigen Möglichkeiten: Herzaktionssignale (EKG), Muskelspannungen (EMG), Gehirnaktivitäten (EEG), Hörgrenze, Hauttemperatur, Reaktionszeit, Pulsfrequenz u. a.
Bestell-Nr. 615411

Erhältlich im Hobby- und Spielwarenhandel

kosmos®

Druckfehlerberichtigung-Computer Praxis

Seite 82
Listing 33:

Adresse	Mnemonics	**Code**
.	.	.
.	.	.
.	.	.
004	VGR 112	**12.112**
.	.	.
.	.	.
.	.	.
011	VGR 112	**12.112**
.	.	.
.	.	.
.	.	.
020	VGR 112	**12.112**
.	.	.
.	.	.
.	.	.
027	VZG 000	**03.000**
028	VZG 000	**03.000**
.	.	.
.	.	.
.	.	.
112	*	**00.009**

Vor jedem neuen Spielturnier muß in Zelle 108 der Wert 00.000 eingegeben werden.

Seite 121
Listing 48:

Adresse	Mnemonics	**Code**
.	.	.
.	.	.
.	.	.
060	AKO 000	**04.000**
.	.	.
.	.	.
.	.	.
067	AKO 001	**04.001**
.	.	.
.	.	.
.	.	.

Technischer Hinweis

Ein Teil unserer Computer-Produktionsserie enthält wegen Lieferengpässen auf dem Bauteilemarkt statt des Mikrocomputers 8049 (siehe Experimentierbuch Seite 70) einen äquivalenten IC-Satz, bestehend aus 8039, 2716, 74LS126 und 2 x 74LS75. Auf die Funktion des Computers hat dies jedoch keinerlei Einfluß.

61-2061.9/100/VII. 83/Re

150