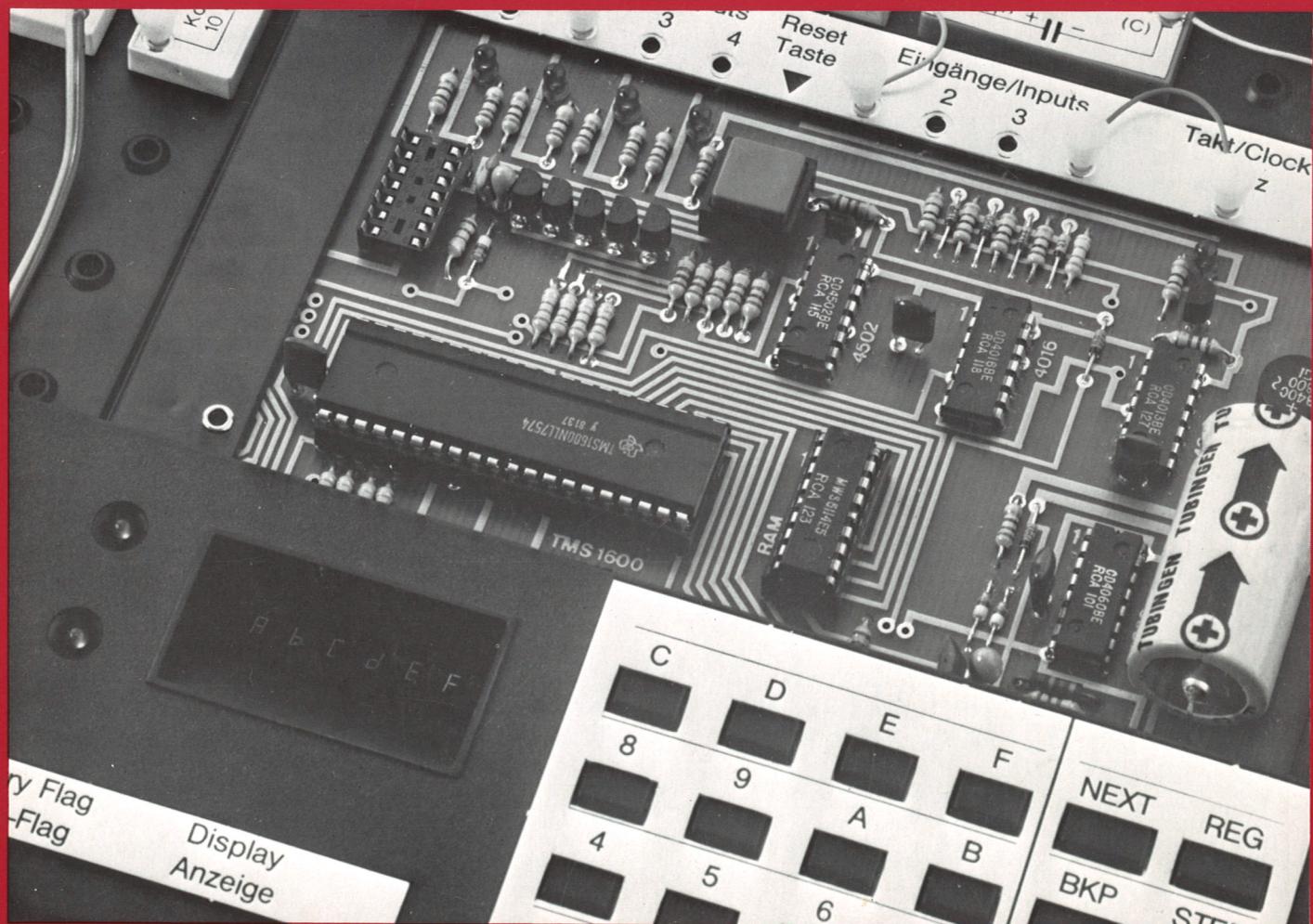




# 2090

## Anleitungsbuch 1. Teil

**Programmieren – Experimentieren – spielend lernen  
wie ein Computer funktioniert.**



In Zusammenarbeit mit  
dem Elektronik-Magazin





## Anleitungsbuch 1. Teil

Einführung in die  
Mikroprozessor- Technik.  
Programmieren und  
Experimentieren  
mit dem Microcomputer.

Von Jörg Vallen



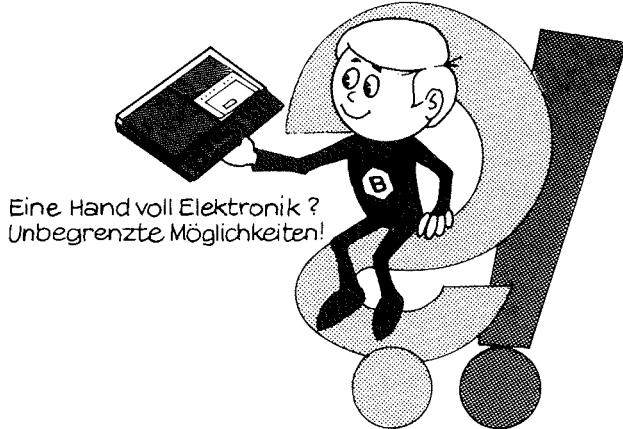
Bei einigen Experimenten  
zusätzlich erforderlich:  
Eine Batterie 9 V (IEC 6 F 22)  
oder Netzteil 2059

Produktion und Vertrieb:  
**BUSCH-Modellspielwaren**  
**Postfach 1360**  
**D 6806 Viernheim**

In Zusammenarbeit mit dem  
**ELO-Magazin Franzis-Verlag**  
**Postfach 370120**  
**8000 München 37**

Copyright 1981 by  
BUSCH GmbH. + Co. KG, Vierneheim  
Alle Rechte vorbehalten.  
Grafiken  
Atelier Wuthe, Weinheim  
Printed in W.-Germany  
10/81

## Der Computer – das unbekannte Wesen



Man sieht es ihm nicht an – und dennoch steht vor uns ein technisches Wunder unserer Zeit. Ein richtig funktionierender Mikro-Computer, dem wir über seine Tastatur Befehle eingeben, damit er auf Knopfdruck diese „programmierten“ Befehle bearbeitet.

Wenn wir ihn richtig bedienen, wird er uns die tollsten Kunststücke vorführen. Seine Möglichkeiten, uns in Erstaunen zu versetzen, sind faszinierend.

Wir werden Zeuge enormer Leistungen sein, die unser Mikro-Computer vor unseren Augen vollbringen wird.

Vielleicht wundern wir uns über das „bißchen Elektronik“, was da vor uns steht?

Vor wenigen Jahren hätte man mit diesem „bißchen Elektronik“ noch einen ganzen Schrank gefüllt. Heute ist die Elektronik auf mikroskopische Größenordnungen geschrumpft. Wo bei ihre Leistungsfähigkeit astronomische Möglichkeiten erreicht hat.

Können wir uns vorstellen, daß alleine in dem ca. 5 cm langen schwarzen IC-Block mit den 40 silbernen Anschlußbeinchen (auf der Platine unter der rauchglasfarbigen Abdeckhaube) ca. 35.000 Transistor-Funktionen enthalten sind?

Ein gigantisches elektronisches Schaltwerk.

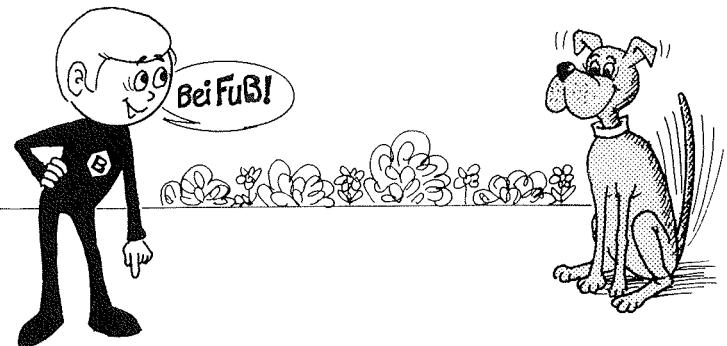
Nun wollen wir dieses unbekannte Wesen näher kennenlernen.

Hätten wir anstelle des Computers z. B. einen jungen Hund angeschafft, ergebe sich folgende Situation: Hund und Mensch müssen lernen, sich gegenseitig zu verstehen. Jeder hat eine eigene Sprache, die der anderen nicht verstehen kann. Also wird Herrchen vermutlich versuchen herauszufinden, was sein neuer Freund versteht, auf welche „Befehle“ er reagiert – oder nicht reagiert, oder vielleicht auch falsch reagieren wird.

Auch wir müssen herausfinden, auf welche Befehle unser neuer Freund, der Mikro-Computer, gehorchen wird, um all das auszuführen, was wir von ihm erwarten.

Wenn wir in seine tiefsten Geheimnisse eindringen möchten, müssen wir mit ihm arbeiten, uns intensiv mit ihm beschäftigen.

Dafür werden wir ein technisch raffiniertes Abenteuer erleben, wie dies vor wenigen Jahren, vor allem für den „Hausgebrauch“ noch undenkbar war.



Übrigens, das ist „BUSCHI“, das BUSCH-Maskottchen. Er demonstriert auf seine Weise, wie er sich die Funktion eines Computer vorstellt.

## Start frei zum ersten Probelauf!

Wir stecken zunächst einmal das Netzgerät in die Steckdose. Sofort erscheinen auf der Leuchtanzeige des Armaturenboards zwei Nullen, eine Dunkelstelle und dann drei Nullen. Eine Leuchtdiode auf der Computer-Platine (vor der Bezeichnung „Takt/Clock“) beginnt zu blinken. Eine weitere Reaktion werden wir zunächst nicht feststellen. Unser Computer ist bereit „Befehle“ von uns entgegenzunehmen. Er erwartet von uns eine Mitteilung, was er tun soll.

Unser Mikro-Computer ist darauf vorbereitet, sich selbst zu kontrollieren und zu prüfen, ob alle seine Funktionen einwandfrei arbeiten. Dieser Computer-Selbsttest gibt uns die Sicherheit, daß beim Transport nichts beschädigt wurde.

Wir sollten nun keineswegs damit beginnen, wahllos die Tasten zu betätigen, weil unser Computer keine sinnlosen, sondern logische Befehle von uns erwartet. Sollten wir jedoch bis zum Lesen dieser Zeilen bereits „Unsinn“ produziert haben, wird sich der Computer diesem sinnlosen Vorhaben sperren, d. h. er wird auf keinen weiteren Tastendruck reagieren.

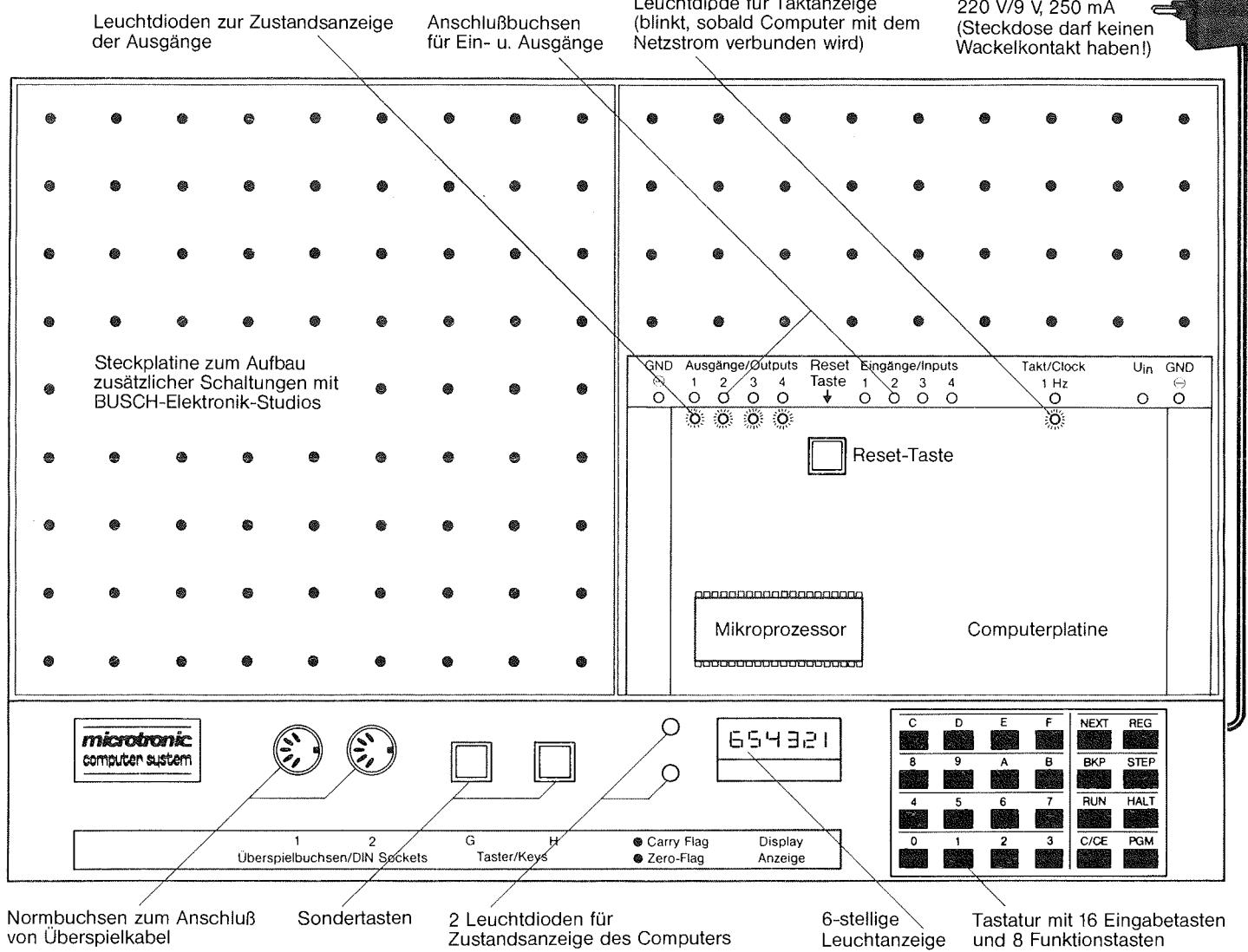
Es bleibt uns nichts anderes übrig, als dem Computer mitzuteilen, daß wir unsere sinnlose Betätigung soeben beendet haben. Zu diesem Zweck drücken wir kurz auf die **grüne Reset-Taste** auf der Computer-Platine. Der Computer wird uns über seine Leuchtanzeige sofort bestätigen, daß er wieder mit uns zusammenarbeiten will (Leuchtanzeige: 00 000).

## Der Computer testet sich selbst – das Prüfprogramm

Für den Selbsttest müssen einige Kabelverbindungen hergestellt werden. Hierzu werden die Kabel gemäß Abbildung mit den blanken (abisolierten) Drahtenden in die Buchsen an der hinteren Platinen-Seite eingesteckt und durch einen aufgedrückten gelben Plastikstecker gehalten. Wir achten darauf, daß die Drähte nicht zu weit in die Buchse eingesteckt werden, weil nur das blaue Drahtende den erwünschten Kontakt in der Buchse ergibt. Die vier notwendigen Kabel führen jeweils



Netzteil zur Stromversorgung  
220 V/9 V, 250 mA  
(Steckdose darf keinen Winkelkontakt haben!)



von den numerierten Ausgangsbuchsen zu den numerierten Eingangsbuchsen, wobei Ausgang Nr. 1 mit Eingang Nr. 1, Nr. 2 mit Nr. 2 usw. zu verbinden ist. Vertauschte Leitungen führen zu einer Fehlermeldung des Computers. (Siehe Abbildung unten). Wir müssen noch wissen, daß unser Computer eine gleichmäßige Stromversorgung benötigt, d. h., daß die zum Einstekken des Netzgerätes benutzte Schuko-Steckdose keinen Winkelkontakt haben darf, weil sonst ein Programmablauf nicht möglich ist.

Nun geben wir dem Computer den Befehl, sein Prüfprogramm zu starten. Hierzu betätigen wir (durch kurzes einmaliges leichtes Tippen) die Taste „HALT“, anschließend die Taste „PGM“. Drücken wir jetzt noch auf die Taste „0“, beginnt der erste Teil des automatischen Prüfprogramms. Auf der Leuchtanzeige („Display“ genannt), erscheinen nacheinander:

```
0000000
1111111
2222222
3333333
4444444
5555555
6666666
7777777
8888888
9999999
A A A A A A A
B B B B B B B
C C C C C C C
D D D D D D D
E E E E E E E
F F F F F F F
```

Der Computer demonstriert uns, daß er zählen kann, wobei er neben den Ziffern 0 bis 9 auch die Buchstaben A bis F vorführt. Es fällt uns auf, daß er die Buchstaben b und d klein schreibt, während die Buchstaben A – C – E und F groß geschrieben werden.

Während der Computer zählt, können wir kontrollieren, ob alle „Leucht-Stäbchen“, aus denen die einzelnen Ziffern und Buchstaben gebildet werden, richtig leuchten.

Hat der Computer bis „FFFFF“ gezählt, leuchten die beiden Leuchtdioden im Armaturenbrett links neben dem Display auf und das Display meldet:

00 100

Die Meldung ist für uns das Zeichen, daß der Computer den ersten Teil des Prüfprogramms richtig absolviert hat. Wenn wir nochmals den gleichen Prozesslauf kontrollieren möchten, betätigen wir erneut die Tasten HALT – PGM – 0 – und warten, bis der Durchlauf beendet ist.

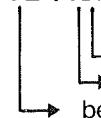
Nun möchte der Computer seine Tastatur überprüfen, um zu kontrollieren, ob alle Tasten bei leichtem Druck einwandfrei funktionieren. Auf der Leuchtanzeige steht ganz rechts eine „0“, wodurch der Computer uns mitteilt, daß wir die Taste „0“ betätigen sollen. Wurde die Taste „0“ richtig betätigt, erscheint in der Anzeige:

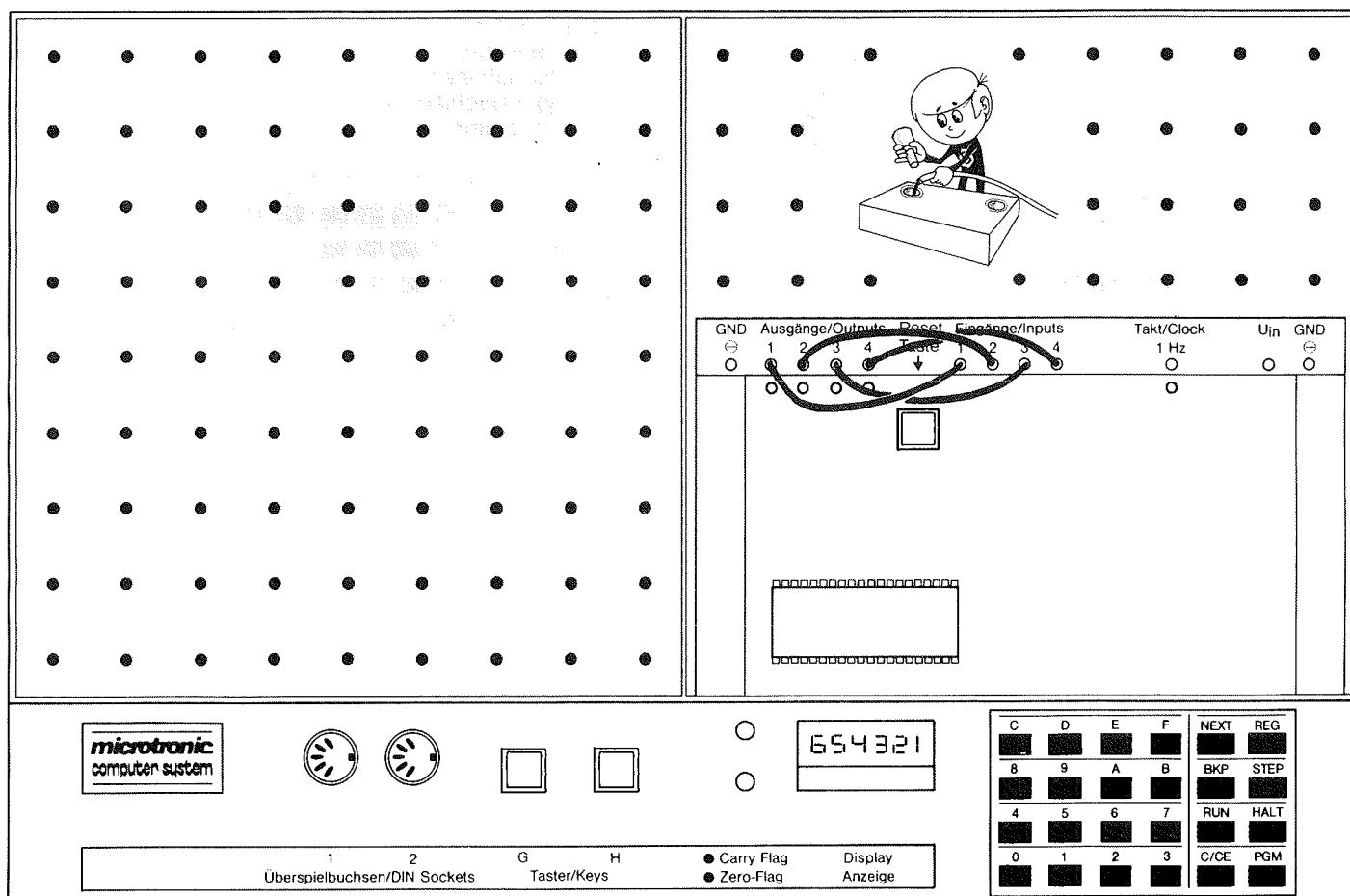
00 101

Für uns heißt dies, die Taste „1“ betätigen. Der Computer meldet 00 102, also Taste „2“ betätigen. So werden alle Tasten bis zu „F“ durchgeprüft.

Sollten wir versehentlich eine andere, als die vom Computer gewünschte Taste betätigt haben, wird er diese Fehlbedienung sofort erkennen, und er meldet auf der Anzeige:

FE 1 XX

  
 → Anzeige der Taste die betätigt werden sollte  
 → Anzeige der falsch betätigten Taste  
 bedeutet: FEHLER!



Kabelverbindungen für Prüf-Programm

Falls wir einen Fehler gemacht haben, drücken wir noch einmal die vom Computer gewünschte Taste (rechte Anzeigenstelle). Sollte trotz richtiger Tasteneingabe (genau kontrollieren) wieder eine Fehlermeldung erscheinen, werden wir das Programm zur Sicherheit noch einmal ganz von vorne starten (Taste „Halt“, „PGM“, „0“). Vielleicht prüfen Sie vorsichtig, ob auch alle Kabelverbindungen von den Ausgängen zu den Eingängen richtig hergestellt wurden. Ergibt sich dennoch eine erneute Fehlermeldung, senden Sie bitte das Gerät unter Nennung des festgestellten Fehlers in der Original-Styropor-Verpackung direkt an unsere Anschrift. Füllen Sie bitte die dem Gerät beigelegte Registrierungskarte genau aus, Sie erhalten kurzfristig Ersatz.



Wurden alle Tasten bis einschließlich „F“ durchgeprüft, setzt der Computer sein Prüfprogramm automatisch alleine fort. Die beiden Leuchtdioden links neben der Anzeige verlöschen und die Anzeige (Display) meldet nacheinander

00 200  
00 211  
00 222  
00 233  
00 244  
usw. bis  
00 2FF

Der Computer zählt in den beiden rechten Stellen von 0 bis F. Er kontrolliert die vier Eingänge und die vier Ausgänge. Sollte der Computer nicht bis zur Stelle „00 2FF“ durchzählen, ergibt sich die Fehlermeldung „FE 2XX“ (der Computer zeigt an der Stelle „X“ eine Ziffer oder einen Buchstaben). Die Fehlermeldung zeigt uns, daß mit Sicherheit an den Kabelverbindungen zwischen Eingänge und Ausgänge etwas nicht in Ordnung ist. Wir sollten nochmals kontrollieren und auch prüfen, ob alle Plastiksteckerchen fest aufgedrückt sind, damit sich kein unerwünschter Wackelkontakt ergibt. Alsdann ist das Prüfprogramm nochmals von Anfang an zu starten (Taste „HALT“, „PGM“, „0“).

Wurde auch dieser Test erfolgreich beendet, leuchtet die untere Leuchtdiode neben dem Display und die Leuchtanzeige schaltet sich für ca. 20-30 Sekunden ab. Intern läuft in dieser Zeit ein weiteres Testprogramm ab, denn der Computer prüft, ob alle Speicher usw. in Ordnung sind. Daß der Computer noch arbeitet, erkennen wir am wechselweisen Aufblitzen der Leuchtdioden an den Ausgängen der Computer-Platine. Sobald auch dieser Test erfolgreich durchlaufen wurde, erscheint in der Anzeige:

00 000

Alle vier Leuchtdioden auf der Computer-Platine vor den Ausgängen leuchten.

Unser Computer ist zu neuen Taten bereit.

Sollte der Computer die letzte Meldung „00 000“ nicht in dieser Weise anzeigen, wäre ein Bauelement auf der Computer-Platine nicht in Ordnung. Das Gerät müßte zu uns eingeschickt werden.

Auch bei einwandfreiem Testlauf sollten Sie uns die Registrierungskarte direkt zusenden. Sie erhalten von uns dann weitere Informationen wie Ihr Microtronic-Computer-System durch neu hinzukommende Ausbaustufen weiter ergänzt werden kann.

Sollten wir bei späteren Experimenten einmal feststellen, daß der Computer nicht so arbeitet wie es im Anleitungsbuch beschrieben wird, können wir durch das Prüfprogramm jederzeit die richtige Funktion des Computers kontrollieren.

## Ein guter Vergleich:

### Computer – Mensch?

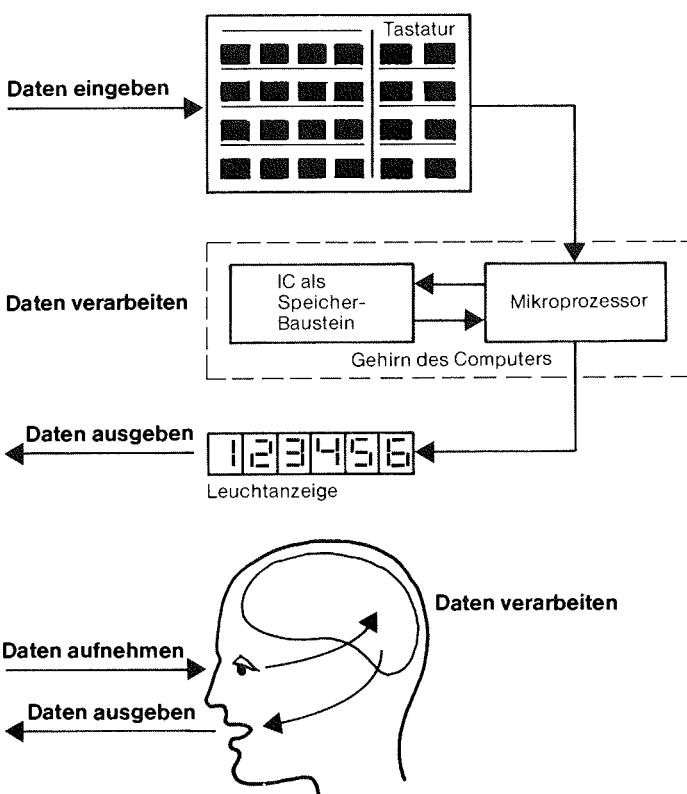
Computer sind elektronische Maschinen, die nicht selbstdäig denken können. Ein Computer besitzt keine Intelligenz – er setzt diese voraus!

Wir müssen dem Computer erst beibringen, wie und wann er was tun soll. Auch der Mensch muß in seinen ersten Lebensjahren lernen zu verstehen und das Verstandene auszuführen. Am Anfang lernt der Mensch nur die wichtigsten Dinge, z. B. „Mama“ und „Papa“. Weitere Worte kommen hinzu – es werden Sätze gebildet – der Mensch lernt denken, sehen, spüren, fühlen, empfinden, rechnen und schreiben. Mit unseren Sinnesorganen nehmen wir die Informationen unserer Umwelt auf. Sie werden im Gehirn verarbeitet und im Gedächtnis gespeichert.

Anstelle des Begriffs „Informationen“ können wir auch das Wort „Daten“ setzen. Jetzt leuchtet uns das Prinzip der „Datenverarbeitung“ schon recht gut ein: Daten aufnehmen (sehen, hören, fühlen) – Daten verarbeiten (im Gehirn) – Daten speichern (im Gedächtnis behalten) – Daten ausgeben (zeigen, sprechen, schreiben).

Die „menschliche Datenverarbeitung“ und die „elektronische Datenverarbeitung“ haben fast das gleiche Prinzip:  
Daten eingeben. Daten verarbeiten. Daten ausgeben.

Ein Kind lernt nicht nur von seinen Eltern – die Eltern müssen berücksichtigen, was das Kind verstehen kann. Ähnliches gilt für unseren Computer: In langer Entwicklungszeit wurde ihm beigebracht („er wurde vorprogrammiert“) eine spezielle Sprache, bestehend aus vielen Worten („Befehle“) zu verstehen. Der Computer hat seine eigene Sprache („Befehlssatz“). Nachdem der Computer keine Intelligenz besitzt, müssen wir seine „Sprache“ erlernen, damit der Computer in der Lage ist, unsere „Befehle“ auszuführen. Das technische Wesen Computer hat etwas ähnliches wie ein Gehirn: Einen Mikroprozessor und einen Speicher-Baustein. Seine Sinnesorgane (Hören, Sehen, Fühlen) werden durch eine Tastatur und die Dateneingänge wahrgenommen. Die Weitergabe seiner gespeicherten Informationen (Daten) werden durch die Leuchtanzeige (Display), verschiedene Leuchtdioden (LED) und die Ausgänge vorgenommen. Während beim Menschen die einzelnen



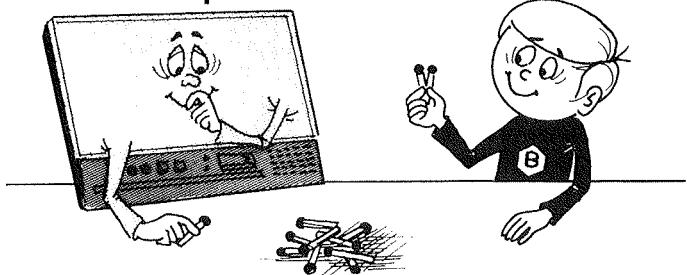
Organe durch Nervenstränge miteinander verbunden sind, verwendet der Computer hierfür die „Leiterbahnen“ auf der Computer-Platine und die winzigen Leiterbahnen-Stränge innerhalb des Mikroprozessors und der verschiedenen IC-Bausteine.

Die Abbildung demonstriert uns, daß auch bei der „menschlichen Datenverarbeitung“ keine direkte Verbindung zwischen den Dateneingabestellen (Augen, Ohren) und zwischen den Datenausgabestellen (z. B. Mund) bestehen. Alle Informationen werden durch die Verarbeitung im Gehirn weitergegeben. Auch beim Computer können die Daten nicht direkt von der Tastatur zur Leuchtanzeige gelangen, sondern sie müssen über den Mikroprozessor entsprechend verarbeitet werden. Der Mikroprozessor ist der „denkende Teil“ des Computer-Gehirns, während sein „Wissen“ in den Speicher-Bausteinen gespeichert wird.

Der Intelligenter gibt nach – also müssen wir die Sprache unseres Computers erlernen. Sobald wir jedoch seine Sprache beherrschen um ihm befehlen (programmieren) zu können, wird er all das tun, was wir von ihm verlangen. Wir erkennen jetzt: Der Computer ist ein Sklave, ohne eigenen Willen. Er tut nur das, was ihm der intelligente Mensch befiehlt.

Unser Microtronic-Computer hat in jahrelangem Training bei seinem Entwickler vieles gelernt. Man hat ihm z. B. ein kleines Spiel in seinem technischen Gehirn fest verankert (fest programmiert), welches er auch während seiner Lagerung und dem Transport nicht vergessen hat. Mit dem folgenden Experiment wollen wir einmal gegen den Computer spielen. Wer wird Sieger?

## Der Computer als Spiele-Partner: Das Nimm-Spiel



Das sogenannte Nimm-Spiel (es ist auch unter anderen Namen bekannt) ist ein kleines Denkspiel für 2 Personen. Es wird normalerweise mit Streichhölzern gespielt.

### Spielregel:

Die Spieler einigen sich, daß z. B. 15 Streichhölzchen auf dem Tisch liegen. Abwechselnd darf jeder Spieler 1-2 oder maximal 3 Hölzchen wegnehmen. Wer das letzte Hölzchen vom Tisch nehmen muß, hat das Spiel verloren. Die Überlegungen gehen also dahin, gegen Ende des Spieles die Zahl der noch vorhandenen Hölzchen so einzurichten, daß man selbst in der Lage ist, 1-3 Hölzchen wegzunehmen, damit dem Spielpartner das letzte Hölzchen verbleibt, und er das Spiel verliert.

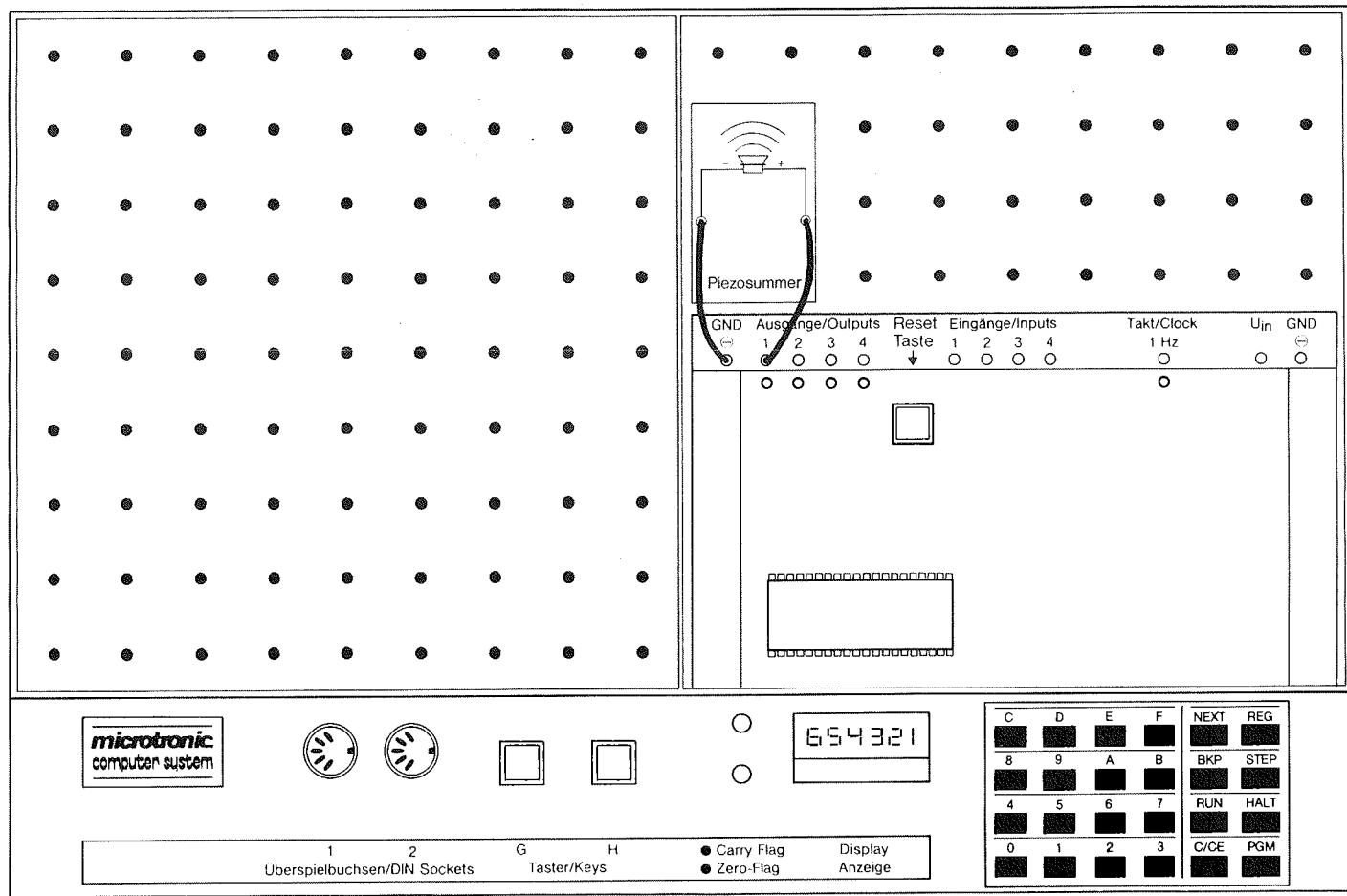
Unser Computer freut sich schon darauf, nun als Spiele-Partner tätig zu werden und zu zeigen, was er bereits gelernt hat. Wir spielen nicht mit Hölzchen, sondern der Computer zeigt uns auf dem Display den Spielverlauf an.

Das Spiel ist im Computer fest programmiert. Wir müssen ihm nun helfen in seinem Gedächtnis (Speicher) dieses Spiel-Programm zu finden. Hierfür betätigen wir nacheinander die Tasten: „HALT“, „PGM“, „7“. Sobald der Computer das Spiel in seinem Speicher gefunden hat, meldet er auf dem Display: **44 F07**. Wir bestätigen ihm, daß wir das Spiel mit ihm spielen möchten, durch Betätigung der Tasten: „HALT“, „NEXT“, „0“, „0“. Auf dem Display erscheint: **00 F08** als Bestätigung, daß der Computer für den Spielbeginn bereit ist. Wir betätigen die Taste „RUN“ – das Spiel beginnt: Die vier Leuchtodiode an den Ausgängen sind dunkel, der Signaltongeber (falls wir den Piezo-Summer gemäß Abbildung angeschlossen haben) ist verstummt. Das Display zeigt: **0000**.

### Anschluß des Piezosummers:

Der Piezosummer muß richtig gepolt angeschlossen werden. Kabel nicht vertauschen! Bitte Abbildung unten beachten!

Falls der Summer (z. B. in den Pausen des folgenden Nimm-Spiels) stört, eine Verbindungsleitung abklemmen.



Der Computer möchte jetzt von uns wissen, wieviel Hölzchen zum Spielbeginn vorhanden sein sollen (z. B. 15 Stück) und wieviel Hölzchen maximal weggenommen werden dürfen (z. B. 3 Stück). Wir betätigen die Zifferntasten „1“, „5“ (Menge der Hölzchen) sowie die Taste „3“ (maximale Wegnahmemenge), im Display erscheint:

**3015**

- Anzeige der vorhandenen Hölzchen
- Anzeige wieviel Hölzchen maximal weggenommen werden dürfen.

Wir wollen zum Spielbeginn von den 15 Hölzchen 2 Stück wegnehmen – betätigen also die Taste „2“. Jetzt sind also noch 13 Hölzchen vorhanden. Der Computer reagiert sofort – nimmt 1 Hölzchen weg – es bleiben 12 Hölzchen übrig. Was der Computer gemacht hat und wieviel Hölzchen übrigbleiben, erkennen wir aus der Display-Meldung:

**1012**

- es bleiben 12 übrig
- Computer nimmt 1

Nun nehmen wir ebenfalls 1 Hölzchen (Taste „1“) – es bleiben 11 übrig – der Computer nimmt 2 Hölzchen und meldet: **2009** (2 genommen, 9 bleiben übrig). Wir nehmen „3“ – der Computer meldet **1005** (er hat 1 genommen, 5 Hölzchen bleiben übrig). Wir nehmen „1“ (somit bleiben 4 übrig) – der Computer reagiert blitzschnell und meldet: **3EE1**. Gleichzeitig blinken die vier Leuchtdioden und der evtl. angeschlossene Piezo-Summer bringt einen wechselnden Pfeifton. Aus der Display-Meldung: **3EE1** sehen wir, daß der Computer 3 Hölzchen weggenommen hat, 1 Hölzchen bleibt für uns übrig (wir haben leider verloren) und mit EE gibt der Computer das Spielende bekannt.

Der Computer bricht das Spiel mit der Meldung **19F00** auch dann ab, wenn wir ihn bemogeln. Würden wir z. B. während des Spielverlaufs mit „0“ kein Hölzchen wegnehmen, oder wir würden statt der maximal zulässigen Zahl 3 z. B. 4 Hölzchen wegnehmen, wird dies vom Computer sofort bemerkt und mit Spielabbruch quittiert. Wir können jedoch das Spiel immer wieder mit den bekannten Tasten neu starten.

Nachdem wir nun wissen, wie wir den Computer überlisten können, werden wir unser Nimm-Spiel in einer neuen Variation fortsetzen. Wir können nämlich nicht nur mit 15, sondern bis zu 99 Hölzchen spielen. Wir können auch festlegen, daß nicht nur maximal 3, sondern bis maximal 9 Hölzchen weggenommen werden dürfen. Wir müssen lediglich beim Spielanfang (HALT-NEXT-0-0-RUN) z. B. die Werte 99 und 9 eingeben. Der Computer wird sich den geänderten Gegebenheiten sofort anpassen und er wird wieder versuchen, die Spiele zu gewinnen. Wenn wir das Spiel beenden, werden wir zweckmäßigerweise den Piezo-Summer (durch Entfernen einer Zuleitung) abschalten, weil er bei den folgenden Experimenten immer wieder dazwischen „piepsen“ würde.

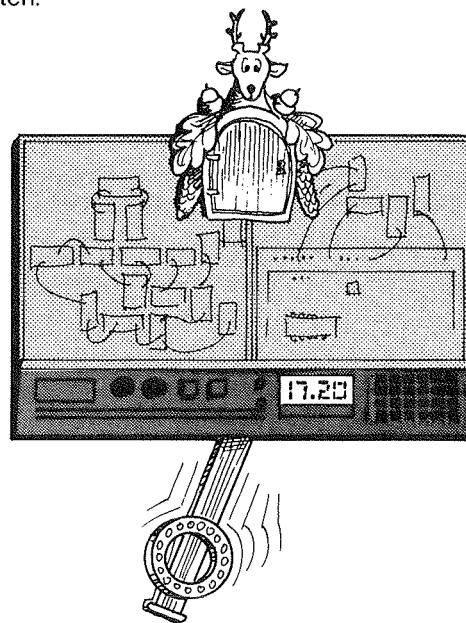
## Der Computer wird zur Digital-Leucht-Uhr

Unser Computer kann noch mehr als „nur“ sich selbst zu testen und ein Spielchen machen. Er weiß auch z. B. was er zu tun hat, um als sekundengenau gehende Digital-Leucht-Uhr zu arbeiten.



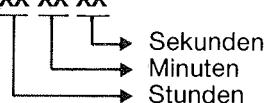
Wir können das Spiel wiederholen, indem wir erneut die Tasten betätigen: **HALT, NEXT, 0, 0, RUN**. Und den Startwert „15“ und „3“ eingeben. Vermutlich wird der Computer auch dieses Spiel gewinnen, es sei denn, daß wir zufällig den kritischen Punkt dieses Spiels herausgefunden haben: Derjenige, welcher beim Hölzchen wegnehmen die Zahl 9 erreicht hat und von dort aus richtig weiterrechnet, kann das Spiel gewinnen. Wir wollen dies wie folgt demonstrieren:

Neuer Spielstart durch Betätigung der Tasten: **HALT, NEXT, 00, RUN**. Wir geben wieder den Startwert mit 15 und 3 ein. Zum Spielbeginn nehmen wir 2 weg. Das Display meldet: 1012 (der Computer hat ein Hölzchen genommen – es sind noch 12 übrig. Wir nehmen 3 (es bleiben 9 übrig). Der Computer nimmt 1 und meldet 1008. Wir nehmen 3 (es bleiben 5 übrig). Der Computer nimmt 1 und meldet 1004. Von den 4 übrigbleibenden nehmen wir 3 – für den Computer bleibt das letzte Hölzchen übrig. Der Computer ärgert sich, daß er das Spiel verloren hat – er bricht das Spiel mit der Meldung **19F00** ab. Alle vier Leuchtdioden an den Ausgängen leuchten und der Piezo-Summer gibt einen Dauerton von sich.



Zunächst müssen wir an der Computer-Platine bei Eingänge/Inputs die Buchse Nr. 4 mit der Anschlußbuchse Takt-/Clock verbinden (siehe Abbildung). Nun helfen wir dem Computer, das in seinem Gedächtnis (Speicher) gespeicherte Uhren-Programm zu finden. Hierfür betätigen wir die Tasten **HALT, PGM, 3**. Da unser Computer nicht weiß, wieviel Uhr es ist, können auf dem Display zunächst verschiedene Zahlen stehen. Wir sagen dem Computer die richtige Zeit z. B. 17:20 Uhr. Hierfür betätigen wir nacheinander wieder die Zifferntasten „1“, „7“, „2“, „0“. Wir sehen, wie die eingegebene Zeit am Display von rechts nach links wandert, wobei die beiden letzten Stellen (für die Sekundenanzeige) auf Null stehen. Die beiden ersten Stellen zeigen die Stunden, die mittleren Stellen die Minuten.

**XX XX XX**



Intern ist die Uhr mit Eingabe der letzten Minutenziffer bereits angelaufen – auf dem Display ist jedoch keine Veränderung zu bemerken. Um die laufende Uhrzeit anzuzeigen, müssen wir die Tasten: „HALT“, „PGM“, „4“ betätigen. Jetzt sehen wir, wie sich die Zeit im Sekundentakt ändert. Nach der 59. Sekunde erfolgt der Minutenprung und nach Ende der 59. Minute der Stundensprung.

Wir können die Uhr unbesorgt im Dauerbetrieb laufen lassen. Auch nach längerer Zeit wird die Uhr sekundengenau weiterarbeiten.

Wir können jedoch ohne weiteres aus dem Uhren-Programm „herausgehen“, um z. B. wieder das Nimm-Spiel anzuwählen mit den Tasten: „HALT“, „PGM“, „7“ dann „HALT“, „NEXT“, „0“, „0“. Obwohl wir jetzt das Nimm-Spiel spielen, läuft die Uhr intern weiter. Wir können dies nach Beendigung des Nimm-Spiels kontrollieren, indem wir uns durch die Tasten „HALT“, „PGM“, „4“, die sich jetzt ergebende Uhrzeit anzeigen lassen. Voraussetzung hierfür ist, daß das Verbindungsleitung zwischen der Eingangsbuchse Nr. 4 und Takt/Clock nicht entfernt wurde.

Der Ordnung halber soll noch erwähnt werden, daß durch den Programmwechsel geringfügige Zeitfehler im Sekundenbereich auftreten können. Auf diese Zeitfehler werden wir später noch eingehen. Wir werden auch erfahren, wie unser Uhren-Programm zu einem Wecker-Radio oder zu einer Schalt-Uhr ausgebaut wird, um z. B. mehrere Geräte zu bestimmten Zeiten ein- und auszuschalten.

#### **Wichtiger Hinweis für die Stromversorgung!**

Der Computer zeigt uns die genaue Zeit solange an, bis die Stromzufuhr (Netzgerät an der Steckdose) unterbrochen wird. Nach wieder Inbetriebnahme muß die Uhrzeit erneut eingegeben werden. Wir können den Computer jedoch unbesorgt im Dauerbetrieb an der Netzsteckdose angeschlossen halten. Der Stromverbrauch ist äußerst gering (ca. 4 Watt) d. h., daß unser Computer in 24 Stunden weniger Strom benötigt, als eine 100 Watt Haushalts-Glühbirne pro Stunde.

## **Unsere ersten Computer-Kenntnisse**

Wir haben die ersten Funktionen und Bedienungselemente kennengelernt:

Die Tastatur für Dateneingabe und Programmaufruf.

Die Leuchtanzeige (Display) für Daten- und Ergebnis-Ausgabe. Eingangsbuchsen und Ausgangsbuchsen zur Herstellung spezieller Anschlüsse.

Den Piezo-Summer als Signaltongeber.

Die PGM-Taste (Programm-Wahlweise): Wir haben festgestellt, daß wir mit der PGM-Taste und der Programm-Nummer die fest im Computer eingespeicherten Programme abrufen können (wobei wir vor jedem Programmwechsel die Taste „HALT“ betätigen):

PGM 0 = Prüfprogramm

PGM 3 = Stellen der Uhrzeit

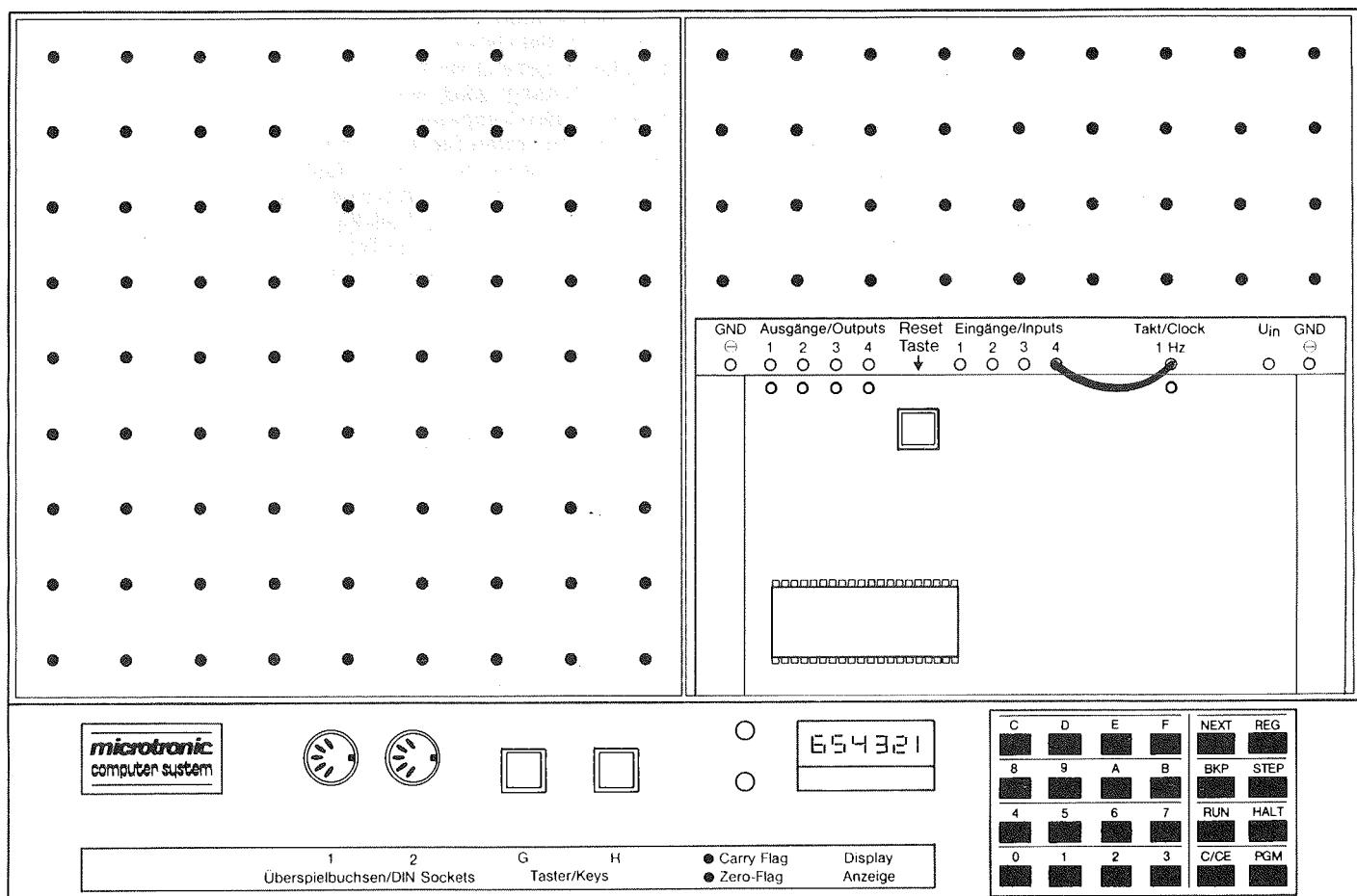
PGM 4 = Anzeigen der Uhrzeit

PGM 7 = Aufruf des Nimm-Spiels (Spiel-Start: HALT-NEXT-0-RUN).

#### **Wenn etwas nicht richtig funktioniert?**

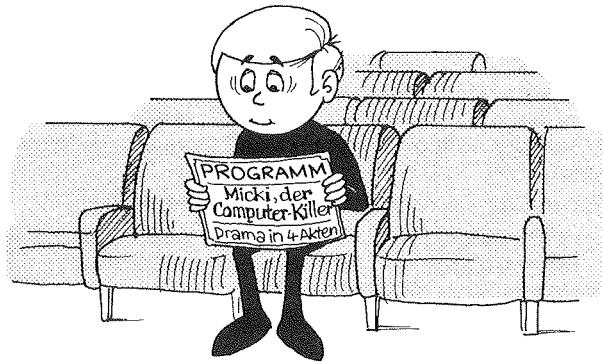
Zum Starten eines Programms müssen die verschiedenen Tasten und die Reihenfolge der Betätigung wie beschrieben eingehalten werden. Wenn wir wahllos und ohne Sinn die Tasten betätigen, wird dies dem Computer zu dumm – er reagiert auf weitere Tastatureingabe nicht mehr. Dies passiert auch, wenn wir mit der Programm-Wahlweise PGM z. B. versehentlich die Programme 1 oder 2 aufgerufen haben.

Wenn der Computer nicht mehr reagiert, betätigen wir die grüne RESET-Taste auf der Computer-Platine – der Computer kommt wieder in seine Start-Position (Display-Anzeige: 00 000). Als nächstes folgt die HALT-Taste und dann die Reihenfolge der übrigen Tasten wie beschrieben. Durch die RESET-Taste werden die eingegebenen „Register-Werte“ z. B. Uhrzeit gelöscht. Die Uhrzeit muß neu eingegeben werden.



Für Uhrenprogramm TAKT/CLOCK mit EINGÄNGE/INPUTS verbinden.

# Computer Programm – was ist das eigentlich?



“Das Programm”

Vieles im täglichen Leben läuft nach einem festen Schema ab. Die meisten Menschen sind in irgendeiner Form für den Tagesablauf „vor-programmiert“. Wenn z. B. morgens der Wecker klingelt, könnte sich folgender Programm-Ablauf ergeben:

1. Ist heute Feiertag? – ja! – dann weiterschlafen
2. Kein Feiertag – also aufstehen
3. Zähne putzen, waschen, anziehen
4. Frühstücke
5. Noch hungrig? – ja – dann zurück zu 4.
6. Nicht mehr hungrig? – Frühstück beenden
7. Wohnung verlassen

Wir sehen, daß schon am frühen Morgen ein festes Programm abläuft, welches sich jeden Tag wiederholt. Wir erkennen die Bestandteile des Programms:

Die einzelnen Programm-Schritte, welche nacheinander auszuführen sind.

Die fortlaufende Numerierung der Programm-Schritte, welche Befehls-Nummer oder **Adresse** genannt wird.

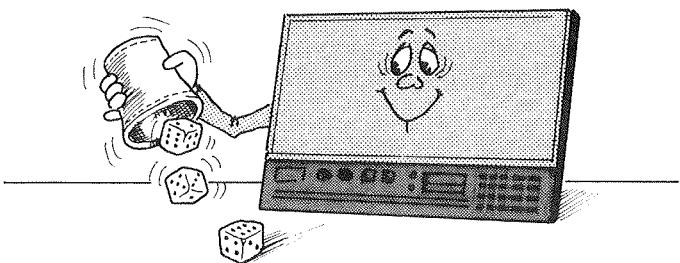
Wir sehen auch, daß die aufgezeigten Programm-Schritte nicht unbedingt nacheinander abgearbeitet werden müssen. Zwischen der Adresse (Befehls-Nummer) 1 und 2 kann eine zeitliche Verzögerung entstehen (weil Feiertag). Es können auch Sprünge ausgeführt werden wie z. B. ein Rücksprung von 5 nach 4 (falls noch hungrig).

Damit ein Computer weiß, was er tun soll, benötigt er ein Programm. Die einzelnen Programm-Schritte bestehen aus kurzen „Befehlen“ – einem Befehls-Code, den der Computer erkennen kann.

Wenn wir einen Computer programmieren möchten, muß der notwendige Programm-Ablauf durchdacht und in kleinere Programm-Schritte unterteilt werden. Die einzelnen Programm-Schritte werden in den für den Computer verständlichen Befehls-Code übersetzt. Die Eingabe solcher Befehle nennt man „programmieren“.

Bevor wir selbst Programme erstellen können, müssen wir die Computer-Befehle lernen, damit wir uns dem Computer gegenüber verständlich machen können.

## Unser erstes Programm: Ein elektronischer Würfel



Wir werden unseren Computer nun erstmals programmieren. Wir möchten gerne, daß er als elektronischer Würfel arbeitet, d. h. er soll zufällige Zahlen zwischen 1 und 6 ermitteln.

Damit der Computer weiß, daß wir ihm nun ein Programm (also mehrere Befehle) eingeben wollen, betätigen wir folgende Tasten:

Zuerst **HALT**  
dann **NEXT**  
dann **0**  
nochmals **0**

Wir sollten uns diese Tasten-Folge einprägen, weil hiermit jede Programmierung beginnt.

Der Computer zeigt jetzt auf dem Display 2 Nullen, eine Dunkelstelle und 3 beliebige Zahlen oder Buchstaben. Sollte dies nicht der Fall sein, müssen nochmals die genannten Tasten betätigt werden.

Bevor wir unseren elektronischen Würfel programmieren, beachten wir folgende Hinweise:

In der folgenden Tabelle finden wir unter **Eingabe** die Hinweise, welche Einzeltasten (Ziffern oder Buchstaben) bzw. welche Funktionstasten zu bedienen sind. Unter der Rubrik **Anzeige** finden wir die Hinweise wie der eingegebene Befehl auf dem Display angezeigt wird. Jedesmal, wenn die Funktions-Taste „NEXT“ betätigt wird, schieben wir den vorher eingegebenen Befehl in den Computer-Speicher. Die Leuchtanzeige zeigt auf den beiden ersten Stellen eine fortlaufende Numerierung und jedesmal, wenn die „NEXT“-Taste betätigt wurde, erscheinen auf den letzten 3 Anzeigestellen beliebige Ziffern oder Buchstaben. Weil diese beliebige Ziffernfolge nicht vorhersehbar ist, werden diese in der Tabelle durch „xxx“ dargestellt. Der Computer zeigt uns einen zufälligen Befehl an, der in seinem Speicher vorhanden ist, welcher jedoch durch unsere neue Befehlseingabe gelöscht wird.

Wenn uns beim Eingeben der einzelnen Befehle ein Fehler unterläuft, schieben wir diesen falschen Befehl nicht mit der „NEXT“-Taste in den Computer-Speicher, sondern wir nehmen eine Korrektur vor. Hierfür verwenden wir die Korrektur-Taste **C/CE**. Wenn wir zweimal auf die Taste „C/CE“ drücken, wird der falsch eingegebene Befehl gelöscht (das Display zeigt auf den 3 letzten Stellen jeweils eine Null) und wir können den neuen Befehl eingeben. Wenn wir einen falsch eingegebenen Befehl zu spät bemerken, müssen wir noch einmal von vorne beginnen, indem wir die Tasten betätigen: „**HALT**“, „**NEXT**“, „**0**“, „**0**“.

Nachdem wir nun schon fast perfekte Programmierer sind, beginnen wir mit der Programm-Eingabe nach folgender Tabelle:

<b>Eingabe</b> (Befehls-Code)	<b>Anzeige</b> (Display)
Einzeltasten: <b>F 0 5</b>	00 F05
Funktionstaste: <b>NEXT</b>	01 xxx
Einzeltasten: <b>9 0 D</b>	01 90d
Funktionstaste: <b>NEXT</b>	02 xxx
Einzeltasten: <b>E 0 0</b>	02 R00
Funktionstaste: <b>NEXT</b>	03 xxx
Einzeltasten: <b>9 6 D</b>	03 96d
Funktionstaste: <b>NEXT</b>	04 xxx
Einzeltasten: <b>D 0 0</b>	04 d00
Funktionstaste: <b>NEXT</b>	05 xxx
Einzeltasten: <b>F 1 D</b>	05 F1d
Funktionstaste: <b>NEXT</b>	06 xxx
Einzeltasten: <b>F F 0</b>	06 FF0
Funktionstaste: <b>NEXT</b>	07 xxx
Einzeltasten: <b>C 0 0</b>	07 C00
Funktionstaste: <b>NEXT</b>	08 xxx

Die Programm-Eingabe ist jetzt abgeschlossen. Unser Computer soll nun das Programm abarbeiten, d. h. er soll elektronisch würfeln.

Hierzu ist folgende Eingabe notwendig:

Zuerst Taste „HALT“  
dann Taste „NEXT“  
dann Taste „0“  
nochmals „0“  
dann Taste „RUN“



Auf dem Display wird jetzt nur noch eine Ziffer angezeigt.

Jedesmal, wenn wir die Taste „0“ betätigen, würfelt der Computer eine neue Zahl zwischen 1 und 6, die im Display angezeigt wird. Die „gewürfelte“ Zahl wird im Computer durch einen Zufallsgenerator erzeugt. Wir haben keine Möglichkeit, das Würfelergebnis zu beeinflussen, wobei es wie beim richtigen Würfel passieren kann, daß auch die gleiche Zahl mehrmals hintereinander erscheint.

Wie haben wir unseren Computer dazu gebracht, daß er bei jedem Tastendruck eine neue Zahl würfelt?

Um dies besser zu verstehen, wollen wir das vorher eingegebene Programm etwas übersichtlicher darstellen. Mit der NEXT-Taste haben wir dem Computer-Speicher nacheinander folgende sieben Programm-Schritte eingegeben:

<b>Befehls-Nummer</b> (Adresse)	<b>Programm-Schritt</b> (Befehls-Code)
00	F05
01	90D
02	E00
03	96D
04	D00
05	F1D
06	FF0
07	C00

Vorstehende Tabelle haben wir im Speicher des Computers abgestellt. Unter der Rubrik Befehls-Nummer (Adresse) ergibt sich eine fortlaufende Numerierung, und wir haben schon beim Eingeben des Programmes bemerkt, daß der Computer diese Numerierung automatisch durchgeführt hat. Jeder Programm-Schritt (Befehls-Code) besteht aus einer Gruppe von 3 Buchstaben, bzw. Ziffern. Durch unsere Programmierung haben wir dem Computer die in nachstehender Tabelle erklärten Befehle erteilt:

<b>Adresse</b>	<b>Befehls-Code</b>	<b>Kurzerklärung</b>
00	F05	Eine Zufallszahl soll ermittelt werden
01	90D	Ist die Zufallszahl eine Null,
02	E00	neue Zufallszahl ermitteln
03	96D	Falls die Zufallszahl größer als 6,
04	D00	neue Zufallszahl ermitteln
05	F1D	Die ermittelte Zufallszahl anzeigen
06	FF0	Warten bis eine Zahlentaste bestätigt wird
07	C00	Auf Adresse 00 am Programm-Anfang zurückspringen und neue Zufallszahl ermitteln

Mit sieben logisch aufgebauten Programm-Schritten haben wir unseren Computer zum elektronischen Würfautomaten gemacht. Eine gewisse Häufigkeit einzelner Zufallszahlen wird bei später folgenden Experimenten durch ein etwas aufwendigeres Programm beseitigt.

## Der Computer soll automatisch zählen

Um das Seelenleben unseres Computers weiter kennenzulernen, wollen wir ihn mit einem neuen Programm füttern.

Aufgabe: Der Computer soll automatisch von 0 bis 99 zählen, um alsdann wieder von vorne zu starten – solange, bis wir das Programm abbrechen.

Aus unserem ersten Programmierungsversuch wissen wir, wie der Computer für die Programm-Eingabe vorbereitet wird:

Funktionstaste „HALT“

Funktionstaste „NEXT“

Zifferntaste „0“

Zifferntaste „0“

Nachfolgende Tabelle zeigt uns, welchen Befehls-Code wir eingeben müssen, wobei wir nach jeder Befehls-Eingabe die Funktionstaste „NEXT“ betätigen. In der vordersten Spalte der Tabelle ist die Adresse (Befehls-Nummer) angegeben, und wir wissen, daß der Computer die Adressen-Numerierung automatisch vornimmt.

Adresse (macht Computer automatisch)	Befehls-Code (über Tastatur eingeben)	Funktionstaste (nach jeder Befehls- Eingabe betätigen)
00	F08	NEXT
01	01D	NEXT
02	02E	NEXT
03	F03	NEXT
04	F2D	NEXT
05	1FF	NEXT
06	71F	NEXT
07	E09	NEXT
08	C06	NEXT
09	511	NEXT
0A	FB2	NEXT
0B	962	NEXT
0C	E0E	NEXT
0D	C01	NEXT
0E	941	NEXT
0F	E00	NEXT
10	C01	NEXT

Damit ist die Programmierung beendet. Wir dürfen niemals vergessen, nach Eingabe eines Befehl-Codes die Funktionstaste NEXT zu betätigen. Die NEXT-Taste sagt dem Computer, daß er den soeben eingegebenen Befehl speichern muß.

Eine falsche Eingabe können wir durch zweimaliges Betätigen der Funktionstaste C/CE löschen (korrigieren), um dann den richtigen Befehl einzugeben. Haben wir zuviel falsch gemacht, brechen wir die Programmierung ab („HALT“, „NEXT“, „0“, „0“) und beginnen nochmals neu.

Der Programm-Lauf soll nun endlich beginnen. **Wir wissen**, daß wir die Programmierung beendet haben – **der Computer weiß dies nicht**, weil er ja unsere Gedanken nicht erraten kann. Wir müssen ihm sagen, daß die Programmierung beendet ist:

Funktionstaste: **HALT**

Funktionstaste: **NEXT**

Zahlentaste: **0**

Zahlentaste: **0**

Nun müssen wir dem Computer noch sagen, daß er sein Programm starten soll:

Funktionstaste: **RUN**

Auf dem Display werden zwei Stellen angezeigt, der Computer zählt automatisch beginnend bei 00 bis 99, um dann wieder bei 00 zu beginnen.

Durch Betätigung der Taste HALT können wir den Programm-Lauf abbrechen. Der Computer zeigt dann die Adresse (Befehls-Nummer) und den Befehls-Code, bei welchem das Programm unterbrochen wurde. Betätigen wir die Taste RUN, wird das Programm an der Stelle fortgesetzt, an welcher wir es unterbrochen wurde.

Sollte die Programmierung unseres Computers nicht einwandfrei funktionieren, können wir kontrollieren, ob wir alle Befehle richtig eingegeben haben. Dies ergibt sich wie folgt:

Funktionstaste **HALT**

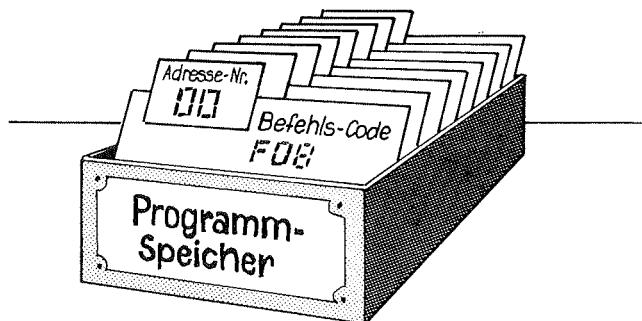
Funktionstaste **NEXT**

Zifferntasten **00**

Auf dem Display erscheint der zuerst eingegebene Befehl: 00 F08. Bei jeder weiteren Betätigung der Taste NEXT, erscheint der nächste Befehl z. B.: 01 01D.

Auf diese Weise können wir das gesamte Programm mit der NEXT-Taste kontrollieren. Zeigt das Display einen falsch eingegebenen Befehl, schieben wir ganz einfach durch Eintippen der richtigen Ziffern- oder Buchstaben-Tasten den neuen Befehl dazwischen, wobei der falsche Befehl durch die Neueingabe gelöscht wird. Mit der Taste RUN kann der automatische Programm-Lauf wieder gestartet werden.

Übrigens, wenn wir die NEXT-Taste immer weiter betätigen, fällt uns auf, daß auch Befehle angezeigt werden, die wir gar nicht eingegeben haben. Die Erklärung hierfür erfahren wir später.



## Die Funktionstasten des Computers

Neben den 16 kleineren Ziffern-/Buchstabentasten, hat unser Computer noch 8 etwas größere Funktionstasten (bzw. Kontrolltasten). Vier Funktionstasten haben wir bereits kennengelernt:

**HALT-Taste:** Sie gibt uns die Möglichkeit, jederzeit die momentan vom Computer ausgeführte Tätigkeit zu unterbrechen. Der Computer geht in Wartestellung, bis eine andere Funktionstaste betätigt wird.

**NEXT-Taste:** Mit ihr kann eine bestimmte Adresse (Befehls-Nr.) angewählt werden (Beispiel: „NEXT 00“ bringt uns an den Programm-Anfang). Beim Programmier-Vorgang wird durch Betätigung der NEXT-Taste der programmierte Befehls-Code zum Speicher weitergegeben und gleichzeitig wird die Adresse automatisch um eine Ziffer erhöht.

**RUN-Taste:** Dient zum Starten eines im Computer vorhandenen Programmes. Der Computer arbeitet.

**PGM-Taste:** Abruftaste für die im Computer fest-gespeicherten Programme (z. B. PGM 0 = Prüfprogramm, PGM 7 = Nimm-Spiel usw.).

## Etwas über Speicher und Adressen

Wir haben bei unseren ersten Programmier-Versuchen, gesehen, daß wir dem Computer Befehle eingeben (Befehls-Code), die er verstehen kann. Sie werden in seinem „Speicher“ abgestellt.



Damit der Computer die im Speicher abgestellten Befehle wiederfindet, hat jeder Befehls-Code (bestehend aus 3 Ziffern oder Buchstaben) eine fortlaufende Numerierung, die wir als Adresse bezeichnen. Der Speicher ist vergleichbar mit einem Karteikasten, in welchem die einzelnen Karteikarten mit dem Befehls-Code abgestellt sind. Damit die einzelnen Befehlskarten leicht auffindbar sind, wurden sie fortlaufend numeriert (mit der Adressen-Nr.).

Mit der Eingabe: HALT – NEXT – 00 sagen wir dem Computer, daß er den ersten eingegebenen Befehl suchen soll, der beim Programmieren automatisch die Adresse 00 erhalten hat. Wird jetzt noch die Taste RUN gedrückt, weiß der Computer, daß er bei der ersten Adresse beginnend das eingegebene Programm abarbeiten muß. Für den Begriff Befehls-Nr. werden wir zukünftig nur noch das Wort Adresse verwenden.

Wir wollen den Computer einmal programmieren, daß er wartet, bis wir eine Zifferntaste betätigen, um dann den Wert der betätigten Zifferntaste anzuzeigen.

Jede Programm-Eingabe wird durch folgende Tasten eingeleitet: **HALT** **NEXT** **0 0**. Die beiden zuletzt eingegebenen Nullen sagen dem Computer, daß der erste Befehls-Code unter der Adresse 00 abgespeichert wird, wobei er bei jedem folgenden Befehl die Adressen-Nr. automatisch erhöht. Damit er dies tut, muß **nach jedem** eingegebenen Befehls-Code die Taste **NEXT** betätigt werden. Wir sehen dies, wenn wir die nachfolgenden drei Befehls-Codes eingeben:

Adresse	Befehls-Code
00	F10
01	FF0
02	C00

Unser Mini-Programm besteht also lediglich aus den drei Adressen 00 bis 02, wobei wir auch nach Eingabe des letzten Befehls die NEXT-Taste nicht vergessen dürfen. Der Computer muß nun wissen, daß die Programmierung beendet ist, also: **HALT** **NEXT** **0 0**. Der Computer hat durch HALT die Programmierung abgebrochen und ist durch NEXT 00 auf den Programm-Anfang zurückgegangen. Er wartet auf unsere weitere Entscheidung. Diese geben wir ihm mit der Taste **RUN**. Der Computer überprüft die Befehle – es tut sich jedoch nichts. Das ist korrekt, weil wir dem Computer ja sagten: Warten bis eine Ziffern- oder Buchstaben-Taste betätigt wird – dann diesen Wert anzeigen. Also drücken wir auf die Zifferntaste 5 – im Display erscheint die 5. Drücken wir auf die Taste F, erscheint auch im Display ein F usw.

### Was bewirken die drei eingegebenen Befehle?

Erster Befehl: F10 = „Anzeige“-Befehl: Du sollst an einer bestimmten Stelle einen bestimmten Wert anzeigen.

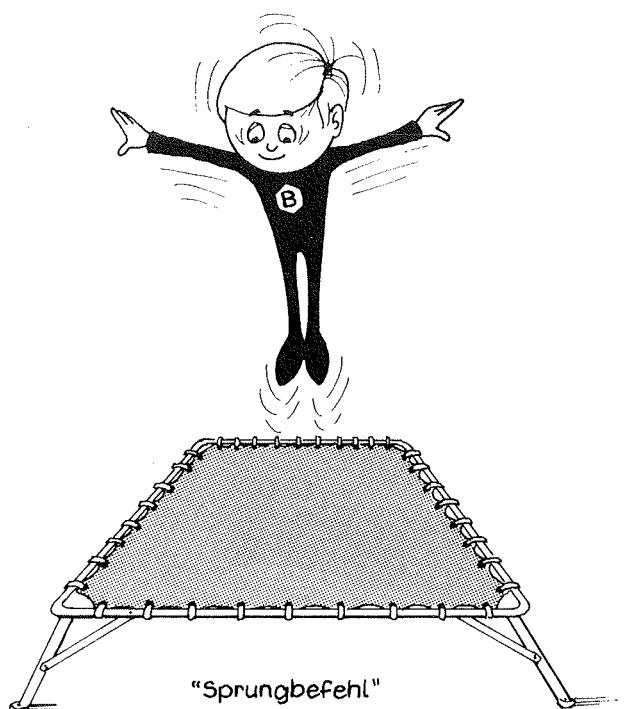
Zweiter Befehl: FF0 = „Warte- und Eingabe“-Befehl: Warten bis eine Ziffern- oder Buchstaben-Taste betätigt wird, den Wert dieser Taste (z. B. Ziffer 1) in einem „Register“ speichern, damit dieser Wert auch nach loslassen der Taste erhalten bleibt.

Dritter Befehl: C00 = „Sprung“-Befehl: Du sollst zum Programm-Anfang (00) zurückspringen.

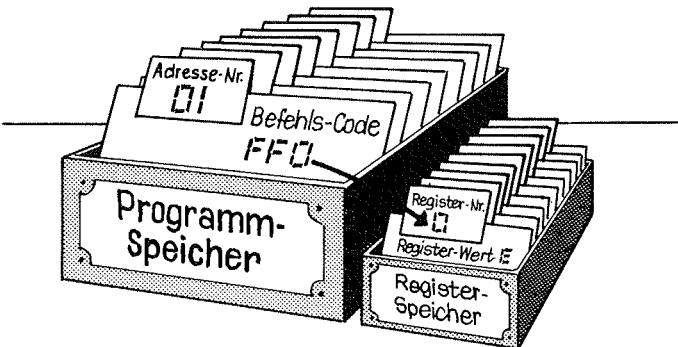
Alles ganz einfach und logisch. Wir müssen uns angewöhnen, logisch zu denken.

## Wir lernen die ersten Befehle

Ein Computer macht nichts von alleine. Alles was er tun soll, muß ihm durch Befehls-Eingabe beigebracht werden. Da er nicht denken kann, müssen wir für ihn denken und dürfen nicht vergessen, ihm mitunter auch die einfachsten Dinge beizubringen. Wenn wir ihm z. B. über die Tastatur eine Zahl eingeben, dann weiß er deshalb noch lange nicht, daß er diese Zahl auf dem Display anzeigen soll.



## Was sind „Register“?



Unter einem Register verstehen wir ebenfalls einen Speicher. Unser Computer hat 2 verschiedene Speichermöglichkeiten: Einen großen Programm-Speicher, in welchem die eingegebenen Programme nach Adressen numeriert gespeichert werden. Er hat außerdem 16 kleine Register-Speicher. Diese Register-Speicher sind so klein, daß nur eine einzige Zahl oder ein einziger Buchstabe pro Register gespeichert werden kann. Das Register ist also ein Miniatur-Speicher.

### Denksport-Aufgabe:

Die 16 verschiedenen Register sollen wie die Programm-Schritte (Adressen) fortlaufend nummeriert werden. Hierfür kann jedoch nur ein 1-stelliger Wert verwendet werden. Mit den 1-stelligen Ziffern 0-9 haben wir keine Probleme, denn wenn wir für jedes Register eine andere Zahl verwenden, haben wir bereits 10 Möglichkeiten der unterschiedlichen Register-Bezeichnung. Für die 11. Möglichkeit bräuchten wir zwei Stellen – wir können jedoch nur eine Stelle pro Register verwenden?

### Lösung des Problems:

Für die 11. Möglichkeit verwenden wir den Buchstaben A, für 12 = B, für 13 = C, für 14 = D, für 15 = E, für 16 = F.  
16 verschiedene Möglichkeiten jeweils mit einer einzigen Stelle dargestellt. Das hätten wir also geschafft!  
Nun wollen wir uns nochmals die Befehle, die wir im vorigen Kapitel gelernt haben, genauer ansehen. Hierbei werden wir feststellen, daß wir mit den wenigen Buchstaben und Zahlen eines Befehls-Codes dem Computer eine ganze Menge Aufgaben erteilt haben:

#### Der Anzeige-Befehl: F10

F = auf dem Display anzeigen  
1 = nur eine einzige Stelle anzeigen  
0 = ab Register 0 anzeigen

#### Eingabe-/Warte-Befehl FF0

FF = warten auf Tasten-Eingabe  
0 = Tastenwert in Register 0 speichern

#### Sprung-Befehl: C00

C = zurückspringen  
00 = bis zur Adresse 00 (also zum Programm-Anfang).

## Wie kann der Computer mehrere Stellen anzeigen?

Mit den drei vorangegangenen Befehlen konnten wir den Computer dazu bringen, uns eine Stelle anzuzeigen. Wenn wir logisch mitgedacht haben, ist es kein Problem, den Computer z. B. zur Anzeige von drei Stellen zu bringen. Wir starten die Programmierung wieder wie bekannt: HALT NEXT 00. Wir sind jetzt wieder am Programm-Anfang bei der Adresse 00, und wenn wir nun einen neuen Befehl eingeben, wird der bisher unter 00 gespeicherte Befehl gelöscht und durch die neue Befehlseingabe ersetzt. Wir programmieren:

#### Anzeige-Befehl: F30

F = Anzeige  
3 = drei Stellen anzeigen (vorher hatten wir nur eine Stelle befohlen)  
0 = ab Register 0 anzeigen  
(NEXT-Taste nicht vergessen!)

#### Eingabe-/Warte-Befehl: FF0

FF = warten auf Tastenbetätigung  
0 = betätigten Tastenwert in Register 0 speichern (gleicher Befehl wie vorher gehabt)

#### Eingabe-/Warte-Befehl: FF1

FF = warten (wie oben)  
1 = Tastenwert in Register 1 speichern (damit sich der Computer auch den zweiten eingegebenen Wert merken kann)

#### Eingabe-/Warte-Befehl: FF2

FF = warten (wie oben)  
2 = Tastenwert in Register 2 speichern (hier merkt sich der Computer die dritte Stelle).

#### Sprung-Befehl: C00

C = zurückspringen  
00 = zur Adresse 00 (Programm-Anfang).  
(Auch nach der letzten Eingabe NEXT nicht vergessen)

Programmierung beendet, also:

#### HALT NEXT 00

Halt = Programmierung beendet  
NEXT = eine Adresse anwählen  
00 = auf Adresse 00 zurückgehen

#### RUN = Programm-Lauf starten.

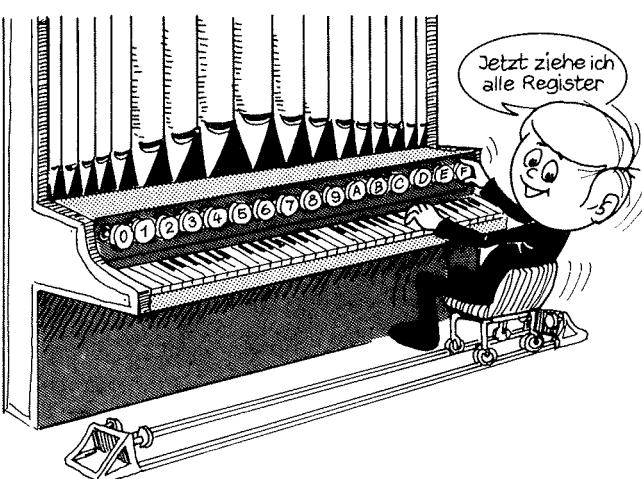
Nacheinander drei Tasten betätigen – die drei Tastenwerte werden auf dem Display angezeigt. Wir können auch ABC betätigen – die drei Buchstaben stehen auf dem Display.

Das war doch eine ganz einfache Sache! Wir wollten zwei Stellen mehr als bei unserem ersten Versuch anzeigen und haben dem Computer beim Anzeige-Befehl lediglich gesagt, daß er uns anstelle einer nun drei Stellen anzeigen soll (statt F10 geändert auf F30) und wir haben für die zwei zusätzlich anzulegenden Stellen zwei weitere Eingabe-Warte-Befehle (FF1 und FF2) eingefügt.

Kein Problem, den Computer auf eine 6-stellige Anzeige zu programmieren. Wir brechen das bisherige Programm ab: HALT NEXT 00 und beginnen mit der neuen Programmierung:

#### Anzeige-Befehl: F60 (es sollen 6 Stellen angezeigt werden)

Erster Eingabe-/Warte-Befehl: FF0 (wie gehabt)  
Zweiter Eingabe-/Warte-Befehl: FF1 (wie gehabt)  
Dritter Eingabe-/Warte-Befehl: FF2 (wie gehabt)  
Vierter Eingabe-/Warte-Befehl: FF3 (für die vierte Anzeige-Stelle)  
Fünfter Eingabe-/Warte-Befehl: FF4 (für die fünfte Anzeige-Stelle)  
Sechster Eingabe-/Warte-Befehl: FF5 (für die sechste Anzeige-Stelle)  
Sprung-Befehl:  
Programmierung beendet:  
Programm starten:



Mal sehen, ob alles geklappt hat? 6 Ziffern- oder Buchstaben-Tasten betätigen – das Display zeigt die betätigten Stellenwerte an. Respekt vor uns – wir wissen jetzt, wie wir den Computer programmieren können!

Ganz nebenbei sollten wir in unserem Gedächtnis registrieren, daß wir durch die Variationen des zuerst eingegebenen Anzeige-Befehls (F10 bis F60) in Verbindung mit den verschiedenen Eingabe-/Warte-Befehlen zwischen 1 und 6 Stellen auf dem Display anzeigen konnten. Unser Computer weiß, daß er nur 6 Stellen anzeigen kann. Würden wir ihm den völlig unlogischen Befehl F70 geben, würde auch der Computer unlogisch reagieren. Vermeiden wir deshalb bei unseren Experimenten unlogische Befehls-Eingaben.

Machen wir statt dessen einmal einen logischen Unsinn: Wir haben den Anzeige-Befehl bisher variiert zwischen F10 und F60. Die letzte Ziffer 0 blieb unverändert, denn wir wollten ja, daß er alle Register anzeigt, die mit den Eingabe-/Warte-Befehlen (FF0 bis FF5) angesprochen wurden. Wenn wir den Anzeige-Befehl ändern in F63 so befehlen wir dem Computer, daß er zwar 6 Stellen anzeigen soll, er bringt uns jedoch nur noch die Stellen, die ab Register 3 angesprochen werden (die durch Tastendruck eingespeicherten Werte in Register 3, 4 und 5).

Probieren wir es doch mal aus. Programm-Lauf stoppen: **HALT NEXT 00**. Eingabe wie gehabt, lediglich die Änderung im Anzeige-Befehl:

**F63**  
**FF0**  
**FF1**  
**FF2**  
**FF3**  
**FF4**  
**FF5**  
**C00**

Programm-Eingabe beenden: **HALT NEXT 00**

Programm starten: **RUN**

Wir betätigen jetzt 6 verschiedene Ziffern- oder Buchstaben-Tasten, und wir sehen, daß sich auf dem Display nur 3 Stellen ändern.



## Mnemonics – und anderes Computer-chinesisch

Bei unseren Programmier-Arbeiten haben wir schon eine ganze Reihe von Befehlen kennengelernt. Schritt für Schritt werden weitere hinzukommen. Wir haben aber auch festgestellt, daß die Befehle für unterschiedliche Funktionen sehr ähnlich sind (Anzeige-Befehl = F10, Eingabe-/Warte-Befehl = FF0). Nachdem sich innerhalb eines Befehls-Codes auch noch die Register-Bezeichnungen ändern können, wäre es umständlich, bei allen Erklärungen, immer wieder z. B. von Anzeige-Befehlen oder Eingabe-/Warte-Befehlen zu sprechen.

Nachdem die Computer- und Mikroprozessor-Entwicklungen ihren Ursprung überwiegend in den USA haben, wurden aus englischen Befehls-Namen englische Kurzformen gebildet, die sich auf der ganzen Welt verbreitet haben. Derartige Kurzformen nennt man „Mnemonics“ (Hilfe – was für ein Wort!).

Vielleicht haben wir bei Computer oder Mikroprozessor begeisterten Freunden schon festgestellt, daß derartige Mnemonics mit wachsender Begeisterung verwendet werden. Mit diesem umweltverblüffenden Computer-chinesisch wird gerne dokumentiert, daß man „Spezialist“ auf diesem Gebiet ist.

Neben dem Verblüffungs-Effekt haben die Mnemonics jedoch den Vorteil, daß wir uns derartige Kurzformen für einen Befehls-Code leichter merken können. Wenn wir dann später Fachzeitschriften oder ausführliche Bücher über die Mikroprozessor-Technik lesen, werden wir immer wieder über diese international gebrauchten Kurzformen stolpern.

### Also lernen wir Computer-chinesisch:

Befehls-Kurzform	Befehls-Code	Erklärungen (Mnemonics)
<b>KIN</b>	<b>FFd</b>	<b>Eingabe-Befehl</b> durch die Tastatur (Keyboard-input). Der eigentliche Befehls-Code = <b>FF</b> . Für <b>d</b> (destination-register*) muß ein Register-Wert (Ziffer oder Buchstabe) zur Speicherung des eingegebenen Wertes eingesetzt werden. Beispiel: <b>FF1</b> , <b>FF2</b> usw.
<b>GOTO</b>	<b>Caa</b>	<b>Sprung-Befehl.</b> GOTO ist die englische Abkürzung von „go to“ also „gehe nach“. Der eigentliche Befehls-Code = <b>C</b> . Für <b>aa</b> wird die entsprechende Programm-Adresse eingesetzt. Kommt der Computer beim Arbeiten eines Programmes auf „Caa“ erfolgt der Programm-Sprung zu der für „aa“ eingesetzten Adresse und wird von dort aus fortgesetzt. Beispiel: <b>C00</b> für den Programm Anfang.
<b>DISP</b>	<b>Fns</b>	<b>Anzeige-Befehl/Ausgabe-Befehl</b> auf das Display (Leuchtanzeige). Der eigentliche Befehls-Code ist <b>F</b> . Für <b>n</b> wird eine Zahl zwischen 1 und 6 eingesetzt. <b>n</b> gibt an, wieviele Stellen angezeigt werden sollen. <b>s</b> (source-register*) gibt an, ab welcher Register-Adresse angezeigt werden soll. Beispiel: <b>F10</b> , <b>F63</b> usw.
<b>DISOUT</b>	<b>F02</b>	<b>Anzeige ausschalten</b> (Display out). Durch diesen Befehl werden alle Anzeigen-Stellen dunkel geschaltet. Beim DISOUT-Befehl gibt es keine Variationen für Register-Bezeichnungen, der Befehls-Code lautet immer <b>F02</b> .

Nachdem wir den ersten Schock überwunden haben, schon wieder neue Bezeichnungen lernen zu müssen, werden wir gerne registrieren, daß wir uns das Kürzel DISP viel leichter als einen Befehl für das Display merken können, gegenüber dem Befehls-Code „Fns“.

Dieses Computer-chinesisch auf Anhieb im Gedächtnis zu behalten, ist etwas schwierig. Deshalb ist dem Anleitungsbuch ein Buchlesezeichen beigegeben, welches uns helfen wird, die versch. Spezialbegriffe, Befehlsbezeichnungen usw. leicht zu finden. Wir benutzen das Buchlesezeichen als unseren privaten Computer-Speicher.

#### \*destination-register

Beim KIN-Befehl ergibt sich der Befehls-Code „**FFd**“, wobei wir für „**d**“ einen Register-Wert einsetzen können. „**d**“ ist die Abkürzung für die englische Bezeichnung „destination regi-

ster" und bedeutet „Ziel-Register“ – also **das Ziel wohin** (zu welcher Register-Bezeichnung) ein Wert abgespeichert werden soll.

### \*source-register

Beim DISP-Befehl ergibt sich der Befehls-Code „Fns“. „n“ gibt an, wieviele Stellen angezeigt werden sollen. „s“ ist die Abkürzung des englischen Begriffs „source-register“, welches wir mit Quelle-Register übersetzen können – also „**die Quelle woher** der angezeigte Wert kommt.“

Kurzgefaßt haben wir uns also lediglich zu merken:

**d = Ziel – wohin?**

**n = wieviel?**

**s = Quelle – woher?**

Mit den Register-Sammelbezeichnungen d-n-s ergeben sich pauschale Befehls-Codes, in welche die Register nachträglich eingesetzt werden können.

Nachdem wir uns jetzt zu den Mikrocomputer-Spezialisten zählen dürfen, wollen wir das vorher programmierte 6-stellige Anzeigen-Programm so darstellen, wie es von Fachleuten „gelesen“ wird:

Adresse	Eingabe Befehls-Code	Befehls-Kürzel (MNEP = Abkürzung für Mnemonic)
00	F60	DISP 0
01	FF0	KIN 0
02	FF1	KIN 1
03	FF2	KIN 2
04	FF3	KIN 3
05	FF4	KIN 4
06	FF5	KIN 5
07	C00	GOTO 00

Würden wir bei der obigen Tabelle unter der Adresse 00 den Befehls-Code F02 DISOUT einsetzen und alle übrigen Programm-Schritte unverändert bestehen lassen, würde der Computer zwar alle eingegebenen Werte speichern, jedoch das Display bleibt dunkel (wir sehen nicht, daß der Computer arbeitet), weil wir mit DISOUT den Befehl gegeben haben: Display abschalten. Dieser Befehl wird beispielsweise bei größeren Rechenoperationen innerhalb des Programms verwendet, wenn wir nicht sehen wollen, wie der Computer rechnet, sondern wenn uns lediglich das Ergebnis interessiert.

## Hexadezimal ist keine Hexerei!

Die verschiedenen Zahlensysteme:

### Dezimal – Hexadezimal – Dual

Unser Computer kann natürlich nicht nur Zahlen anzeigen – er kann auch rechnen. Wir werden später feststellen, daß der Computer durch seine enorm hohe Arbeitsgeschwindigkeit ein wahrhaftiger Rechenkünstler ist.

Bevor wir ihm jedoch das Rechnen beibringen, müssen wir wissen, wie und mit welchen Zahlen der Computer seine Rechenaufgaben durchführt.

Wir sind es gewohnt, alle Rechenaufgaben mit dem dezimalen Zahlensystem, also den Ziffern 0 bis 9 durchzuführen. Logisch, daß die Addition der beiden 1-stelligen Ziffern  $9 + 1 = 10$  ergibt. Das Ergebnis ist 2-stellig geworden. Der Computer geht mit mehrstelligen Zahlen nicht so großzügig um wie wir das tun. Der Computer arbeitet bei der Zahl 10 immer noch mit einer Stelle weiter und verwendet hierfür den Buchstaben A. Die Zahl 11 wird B, 12 wird C, 13 wird D, 14 wird E, 15 wird F. Er kann also (einschl. der Zahl 0) 16 verschiedene Werte (16 Möglichkeiten) 1-stellig darstellen. Das ist das hexadezimale Zahlensystem.

Es gibt noch ein drittes Zahlensystem – das Dualsystem. Wir werden die duale Zahlendarstellung sehr schnell begreifen, wenn wir unseren Computer durch entsprechende Programmierung auffordern, uns das Dualsystem vorzuführen.

Wir bringen den Computer mit den bekannten Befehlen in seine Anfangsposition: **HALT – NEXT – 00** und wir nehmen folgende Befehls-Eingabe vor (bitte nicht vergessen nach jeder Befehls-Eingabe die Taste NEXT zu betätigen):

Adresse (auto-matisch)	Eingabe Befehls-Code	Befehls-Kürzel (Mnemonic)	Kurzerklärung
00	<b>F10</b>	DISP 1,0	Ausgabe auf Leuchtdiode
01	<b>FE0</b>	<b>DOT</b> 0	Daten an Ausgänge (neuer Befehl!)
02	<b>FF0</b>	KIN 0	Eingabe-Befehl
03	<b>C00</b>	GOTO 00	Sprung-Befehl

Wir starten das Programm wie bekannt:

**HALT – NEXT – 00 – RUN.**

Über die Tastatur können wir wieder einen Wert eingeben, welcher auf dem Display angezeigt wird. Zusätzlich leuchten an den Ausgängen der Computer-Platine eine oder mehrere (oder gar keine) Leuchtdioden auf. Durch den **DOT**-Befehl werden die eingegebenen Werte nicht nur auf dem Display angezeigt – der DOT-Befehl bringt den Inhalt des Speicher-Registers 0 an die 4 Ausgänge unseres Computers. An jedem Ausgang ist eine Leuchtdiode fest angeschlossen und sie leuchtet auf, wenn der Computer die notwendige Betriebsspannung zu einer Ausgangsleitung schaltet. DOT bedeutet **DATA OUT** – also Daten an Ausgänge bringen.

Wir betätigen die Zifferntaste 0 – das Display zeigt 0 – keine Leuchtdiode leuchtet. Taste 1 – Display zeigt 1 – eine Leuchtdiode leuchtet. Taste 2 – eine andere Leuchtdiode leuchtet. Taste 3 – zwei Leuchtdioden leuchten, usw.

Wenn wir die einzelnen Tasten-Werte gemäß nachfolgender Tabelle eingeben, stellen wir fest, daß eine oder mehrere Leuchtdioden nach einem ganz bestimmten System vom Computer geschaltet werden. In der Tabelle sind die vier Leuchtdioden an den Ausgängen 1 – 4 (der Computerplatine) durch Ziffern 0 und 1 dargestellt: 0 = Leuchtdiode aus, 1 = Leuchtdiode leuchtet.

Eingegebener Tasten-Wert		Dual-System dargestellt durch Leuchtdioden an den Ausgängen: 4 3 2 1
0	↑	0 0 0 0
1		0 0 0 1
2		0 0 1 0
3		0 0 1 1
4		0 1 0 0
5		0 1 0 1
6		0 1 1 0
7		0 1 1 1
8	↓	1 0 0 0
9	↓	1 0 0 1
A		1 0 1 0
B		1 0 1 1
C		1 1 0 0
D		1 1 0 1
E		1 1 1 0
F	↓	1 1 1 1

Wir erkennen daß wir die 16 verschiedenen Möglichkeiten des Hexadezimal-Systems auch durch das Dual-System darstellen können. Diese Tabelle finden wir für spätere Anwendungsebeispiele ebenfalls auf dem Buchlesezeichen.

Wir haben auch gesehen, daß ein direkter Zusammenhang zwischen dem eingegebenen Tasten-Wert und den am Ausgang leuchtenden LED's besteht. Ist an einem Ausgang ein Strom vorhanden: LED leuchtet. Ist kein Strom vorhanden: LED leuchtet nicht. Die LED's befinden sich in einem leuchtenden, oder einem nichtleuchtenden „Zustand“.

Die gesamte Computer-Technik arbeitet nur mit diesen beiden „Zuständen“. Wie in vorangegangener Tabelle dargestellt, wird der Zustand: Kein Strom vorhanden, mit der Ziffer 0 bezeichnet. Der Zustand: Strom vorhanden, wird mit der Ziffer 1 be-

zeichnet. Die Computer-Technik arbeitet also nur mit den beiden Ziffern 0 und 1. Um mit den Zahlen von 0 – 9 und den Buchstaben A – F arbeiten zu können, benötigt man eine 4-stellige „Computer-Zahl“. Diese wird durch das Dual-System erreicht.

Diesen speicherbaren kleinsten Computer-Wert (0 oder 1) bezeichnet man als ein **bit** (Abkürzung des englischen **binary digit**). Um die Zahlen 0 – 9 und Buchstaben A – F darstellen zu können, benötigt man 4 bit. Die 4 bit bringen dem Computer 16 Kombinationsmöglichkeiten. Diese 16 Möglichkeiten bietet auch das Hexadezimal-System durch die Zahlen 0 – 9 und Buchstaben A – F. Nach einigen Experimenten haben wir uns schnell mit dem hexadezimalen Zahlen-System vertraut gemacht.

Eine Frage steht noch offen: Beim Programmieren haben wir gesehen, daß der Computer für jeden neu eingegebenen Programm-Schritt (Adresse) automatisch um einen Zahlenwert erhöht. Die nächste Zahl nach 9 wäre 10 – der Computer verwendet hierfür A. B = 11 usw. bis die 15 zu F geworden ist. Was kommt dann? Oder können wir unserem Computer nur 16 Adressen-Möglichkeiten eingeben?

Wir müssen die Adressen-Numerierung der Programm-Schritte von der Numerierung der Register unterscheiden. Für die Adressen der Programm-Schritte hat unser Computer eine 2-stellige Speicher-Möglichkeit. Wenn mit F die Zahl 15 erreicht wurde, zählt der Computer 2-stellig im Hexadezimalen-Zahlen-System weiter. Die folgende Tabelle zeigt uns den Vergleich wie die Dezimal-Zahlen von 0 – 255 hexadezimal dargestellt aussehen. Von 0 – 255 ergeben sich 256 Möglichkeiten. Deshalb kann unser Computer 256 Programm-Schritte als zweistellige Adressen speichern.

Sollten wir noch nicht alles verstanden haben, so ist dies kein Problem. Bei den folgenden Experimenten wird uns vieles wesentlich verständlicher werden.

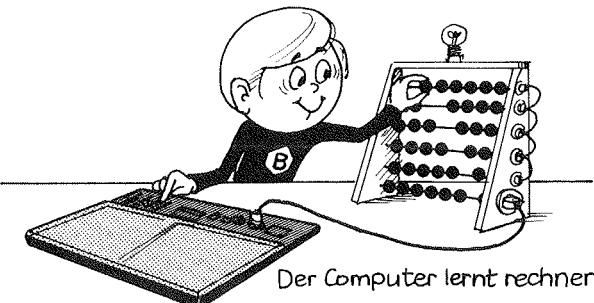
Gegenüberstellung dezimales / hexadezimales Zahlensystem						
Dez. (Dezimal) = Hex. (Hexadezimal)						
Dez. = Hex.	Dez. = Hex.	Dez. = Hex.	Dez. = Hex.	Dez. = Hex.	Dez. = Hex.	Dez. = Hex.
0 00	48 30	96 60	144 90	192 C0	240 F0	
1 01	49 31	97 61	145 91	193 C1	241 F1	
2 02	50 32	98 62	146 92	194 C2	242 F2	
3 03	51 33	99 63	147 93	195 C3	243 F3	
4 04	52 34	100 64	148 94	196 C4	244 F4	
5 05	53 35	101 65	149 95	197 C5	245 F5	
6 06	54 36	102 66	150 96	198 C6	246 F6	
7 07	55 37	103 67	151 97	199 C7	247 F7	
8 08	56 38	104 68	152 98	200 C8	248 F8	
9 09	57 39	105 69	153 99	201 C9	249 F9	
10 0A	58 3A	106 6A	154 9A	202 CA	250 FA	
11 0B	59 3B	107 6B	155 9B	203 CB	251 FB	
12 0C	60 3C	108 6C	156 9C	204 CC	252 FC	
13 0D	61 3D	109 6D	157 9D	205 CD	253 FD	
14 0E	62 3E	110 6E	158 9E	206 CE	254 FE	
15 0F	63 3F	111 6F	159 9F	207 CF	255 FF	
16 10	64 40	112 70	160 A0	208 D0		
17 11	65 41	113 71	161 A1	209 D1		
18 12	66 42	114 72	162 A2	210 D2		
19 13	67 43	115 73	163 A3	211 D3		
20 14	68 44	116 74	164 A4	212 D4		
21 15	69 45	117 75	165 A5	213 D5		
22 16	70 46	118 76	166 A6	214 D6		
23 17	71 47	119 77	167 A7	215 D7		
24 18	72 48	120 78	168 A8	216 D8		
25 19	73 49	121 79	169 A9	217 D9		
26 1A	74 4A	122 7A	170 AA	218 DA		
27 1B	75 4B	123 7B	171 AB	219 DB		
28 1C	76 4C	124 7C	172 AC	220 DC		
29 1D	77 4D	125 7D	173 AD	221 DD		
30 1E	78 4E	126 7E	174 AE	222 DE		
31 1F	79 4F	127 7F	175 AF	223 DF		
32 20	80 50	128 80	176 B0	224 E0		
33 21	81 51	129 81	177 B1	225 E1		
34 22	82 52	130 82	178 B2	226 E2		
35 23	83 53	131 83	179 B3	227 E3		
36 24	84 54	132 84	180 B4	228 E4		
37 25	85 55	133 85	181 B5	229 E5		
38 26	86 56	134 86	182 B6	230 E6		
39 27	87 57	135 87	183 B7	231 E7		
40 28	88 58	136 88	184 B8	232 E8		
41 29	89 59	137 89	185 B9	233 E9		
42 2A	90 5A	138 8A	186 BA	234 EA		
43 2B	91 5B	139 8B	187 BB	235 EB		
44 2C	92 5C	140 8C	188 BC	236 EC		
45 2D	93 5D	141 8D	189 BD	237 ED		
46 2E	94 5E	142 8E	190 BE	238 EE		
47 2F	95 5F	143 8F	191 BF	239 EF		

## Weiterlesen – oder experimentieren?

Wir haben bereits festgestellt, daß ein sinnloses Experimentieren vom Computer nicht akzeptiert wird. Nachdem wir die grundsätzliche Bedienung des Computers kennengelernt haben, ist es uns möglich, die im zweiten Teil des Anleitungsbuches enthaltenen Programme in den Computer einzugeben, um zu erleben, wie er für die verschiedenen Verwendungszwecke eingesetzt werden kann. Es werden uns allerdings verschiedene Zusammenhänge unklar bleiben, weshalb wir die Programmierung sehr sorgfältig vornehmen müssen. Wenn ein Programm nicht einwandfrei funktioniert, können wir mit der NEXT-Taste das Programm „durchblättern“, um einen evtl. falsch eingegebenen Befehl zu korrigieren.

Wenn wir die Funktionsweise des Computers und seiner Befehle genauer kennenlernen wollen, sollten wir das Anleitungsbuch Seite für Seite durcharbeiten. Nur wenn wir unseren Computer und alle seine Möglichkeiten richtig verstehen, werden wir in der Lage sein, eigene Programme zu entwickeln, um den Computer zu veranlassen, unsere eigenen Ideen auszuführen. Er bietet uns dann fast unbegrenzte Möglichkeiten.

## Der Computer lernt rechnen



Wir wollen unseren Computer dazu bringen, daß er nicht nur Zahlen anzeigt, sondern daß er zwei Zahlen addiert. Am Anfang soll uns eine 1-stellige Addition genügen, die uns neue Erkenntnisse der Computer-Technik bringen wird.

Wir starten die Programm-Eingabe wie bekannt: **HALT – NEXT – 00** als dann:

Adresse (macht Computer) automatisch)	Eingabe Befehls- Code	Befehlskürzel (Mnemonics)
00	<b>F10</b>	DISP 1,0
01	<b>FF1</b>	KIN 1
02	<b>F11</b>	DISP 1,1
03	<b>FF0</b>	KIN 0
04	<b>410</b>	<b>ADD 1,0</b>
05	<b>C00</b>	GOTO 00

\*Wir haben einen neuen Befehl (**ADD** für Addition) verwendet.

Programm-Eingabe beendet, also: **HALT – NEXT – 00**  
Programm starten: **RUN**

Wir geben nacheinander zwei Zahlen ein, z. B. zuerst 3 dann 4. Das Display zeigt die zuerst eingegebene Zahl 3 und sofort nach der eingegebenen Zahl 4 (wird nicht angezeigt) zeigt das Display das Ergebnis: 7.

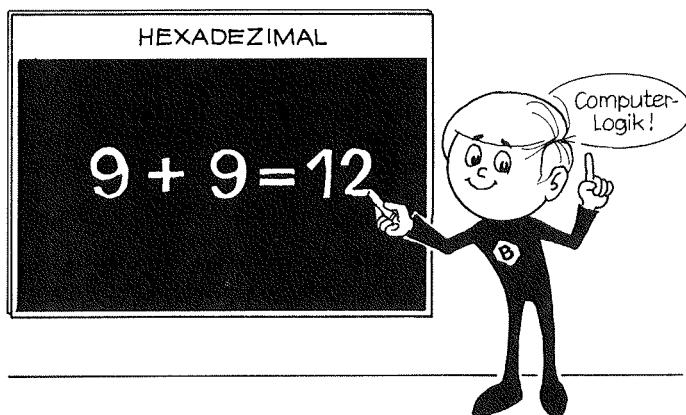
Wir geben zwei weitere Zahlen ein, z. B. 3 und 6 = 9.

Was passiert, wenn wir 5 und 5 eingeben? Das Ergebnis ist A. Der Computer ist vom Dezimalen- zum Hexadezimalen-Zahlenbereich gekommen. Das ist richtig, denn  $5 + 5 = 10$ , und wir können in der vorangegangenen Tabelle ablesen, daß die dezimale 10 ein hexadezimales A ergibt.

Für den Computer gibt es keinen Unterschied, ob er mit Zahlen oder Buchstaben rechnet. Probieren wir es aus und geben ein: B + 2 = D. Der hexadezimale Wert B ist eine dezimale 11 + 2 = hexadezimal D was der dezimalen Zahl 13 entspricht. (Vergleiche mit Tabelle).

Interessant wird jetzt die Addition von 9 + 9. Das Display zeigt als Ergebnis 2! Hat sich der Computer verrechnet?

Der Computer hat richtig gerechnet – unser Additions-Programm ist jedoch nicht vollständig – er kann mit diesem Programm nur 1-stellig rechnen. Sein hexadezimaler Zahlenaufang besitzt 16 Ziffern, der höchste hexadezimale Wert F entspricht dem dezimalen Wert 15. Zur Anzeige von 16 oder eines höheren Wertes, müßte der Computer eine zweite Stelle dazunehmen.



Dezimal gerechnet ergibt  $9 + 9 = 18$ . Hexadezimal lautet das Ergebnis 12 (siehe Tabelle). Da dem Computer gesagt wurde, daß er nur eine Stelle anzeigen soll, bringt er das Ergebnis der letzten Stelle, nämlich 2. Er hat sich jedoch gemerkt, daß ein „Übertrag“ stattgefunden hat und zeigt dies durch die aufleuchtende obere LED neben dem Display an.

Diese LED leuchtet immer dann auf, wenn der Computer einen Übertrag machen muß, d. h. wenn sich z. B. durch Addition ein höherer Wert als das hexadezimale F ergibt. Diese Leuchtdiode wird als **Carry-Flag** bezeichnet und bedeutet für den Computer eine **Übertrag-Markierung**. Der Computer bringt gewissermaßen ein Signal, mit welchem die zweite Stelle bei einem Übertrag angesteuert werden kann.

Wir wollen unser kleines Additions-Programm etwas genauer betrachten:

Adresse	Eingabe Befehls- Kürzel	Befehls- Kürzel	Erklärungen
00	F10	DISP 1,0	Anzeige-Befehl
01	FF1	KIN 1	Eingabe-Befehl: Eingegebener Wert in Register 1 abspeichern
02	F11	DISP 1,1	Ergebnis von Adresse 01 anzeigen
03	FF0	KIN 0	Eingabe-Befehl: Den zweiten eingegebenen Wert in Register 0 speichern
04	410	ADD 1,0	Additions-Befehl: Addiere den Inhalt von Register 1 + Register 0. Ergebnis in Register 0 speichern
05	C00	GOTO 00	Sprung-Befehl: Auf Programm Anfang springen und dort durch den Befehl DISP 1,0 Ergebnis anzeigen

Der **ADD-Befehl** hat den Befehls-Code **4sd. 4** ist für den Computer der Befehl zum Addieren. Für **s** erinnern wir uns an source-register = Quell-Register. Hierfür wird ein beliebiges Register verwendet. **d** ist das zweite Register, welches addiert werden soll. **d** steht wieder für destination, also Ziel-Register, weil in diesem Register das Ergebnis steht. Ein in diesem Register vorher eingegebener Wert wird durch das neu hinzugefügte Ergebnis gelöscht. Es wird also immer nur das letzte Ergebnis angezeigt. Im vorangegangenen Programm haben wir für **s** das Register 1 eingegeben und für das **d** das Register 0.

## Der rasende Automatik-Zähler

Wie üblich, bringen wir den Computer auf den Programm-Anfang: **HALT – NEXT – 00**.

Wir programmieren:

Adresse	Eingabe Befehls-Code	Befehlskürzel (Mnemonic)
00	<b>F10</b>	DISP 1,0
01	<b>510</b>	ADDI 1,0
02	<b>C00</b>	GOTO 00

Programm-Start: **HALT – NEXT – 00 – RUN**

Der Computer zählt so schnell, daß wir auf dem Display keine Zahlen erkennen. Die beiden LED's (Carry- und Zero-Flags) neben dem Display blinken.

Bevor wir den Zählvorgang künstlich verlangsamen, wollen wir das kleine Programm untersuchen. Neu ist der **ADDI-Befehl**. Er ist ebenfalls ein Additions-Befehl. Der Unterschied zum vorher gelernten ADD-Befehl, der zwei unterschiedliche Werte addiert, arbeitet der ADDI-Befehl mit einem konstanten Wert. Der **ADDI-Befehl** hat den Befehls-Code **5nd. 5** befiehlt die Addition. Für **n** wird ein konstanter Additionswert eingegeben (in unserem Programm-Beispiel die Ziffer 1). Dieser konstante Wert wird zum Register **d** (destination-register/Ziel-register) hinzugefügt. ADDI ist die Abkürzung für das englische **add immediate**.

Nun wollen wir einmal in „Zeitlupe“ sehen, wie unser Programm arbeitet. Wir stoppen das Programm mit HALT und gehen mit NEXT 00 an den Programm-Anfang. Anstelle der bisher üblichen RUN-Taste, betätigen wir jetzt nochmals die HALT- und dann die **STEP**-Taste. Das Programm wird jetzt schrittweise gestartet, d. h. daß wir mit der STEP-Taste jeden einzelnen Programm-Schritt bei der Arbeit verfolgen können.

Bei jeder STEP-Tasten Betätigung wird die Adresse und der Befehls-Code angezeigt, der als nächstes abgearbeitet wird. Wird erneut die STEP-Taste betätigt, wird der zuvor angezeigte Befehl ausgeführt, gleichzeitig wird der nächste Befehl angezeigt. Durch immer wieder erneute Betätigung der STEP-Taste können wir sehen, wie sich der Register-Inhalt jeweils um den Wert 1 erhöht. Wenn wir mit der STEP-Taste solange weiterzählen, bis wir über die Buchstaben A-B-C-D-E den Wert F überspringen (d. h. daß der Zählvorgang wieder bei 0 beginnt), leuchtet das Carry-Flag neben dem Display auf. Ein Übertrag hat stattgefunden. Das ebenfalls aufleuchtende Zero-Flag soll uns momentan noch nicht interessieren.

## Ein 2-stelliger Automatik-Zähler

Der jetzt folgende Automatik-Zähler addiert auf 2 Stellen. Nach Eingabe von HALT – NEXT – 00 nehmen wir folgende Neuprogrammierung vor:

Adresse	Eingabe Befehls-Code	Befehlskürzel (Mnemonic)	Kurzerklärung
00	<b>F20</b>	DISP 2,0	2 Stellen anzeigen
01	<b>510</b>	ADDI 1,0	konstanten Wert 1 dazuaddieren
02	<b>FB1</b>	ADC 1	Übertrag auf 2. Stelle übernehmen
03	<b>C00</b>	GOTO 00	zurück zu Adresse 00

Programm-Start: HALT – NEXT – 00 – RUN

Auf der letzten Display-Stelle ergibt sich derselbe schnelle Zählvorgang wie beim vorangegangenen Experiment. Auf der 2. Stelle können wir jedoch den hexadezimalen Zählvorgang (von 0 – F) genau verfolgen.

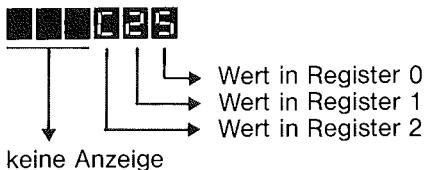
Das Programm ist mit dem vorangegangenen Programm fast identisch. Auf Adresse 02 haben wir einen neuen **ADC-Befehl (ADD CARRY)** eingefügt. Der neue Befehl „addiere-CARRY“ registriert den „Übertrag“, wodurch nun auch die 2. Stelle angezeigt wird. Wir erinnern uns, daß die neben dem Display mit Carry Flag bezeichnete Leuchtdiode immer dann aufleuchtet, wenn der Computer bei seinem hexadezimalen Zählvorgang die höchste 1-stellige „Ziffer“ F überspringt, d. h. wenn ein Übertrag zu einer zweiten Stelle notwendig wird.

Bisher haben wir den Computer beauftragt, jeweils nur die Ziffer 1 (oder besser gesagt den Wert 1) dazu zu addieren. Was müssen wir ändern, wenn der Computer bei jedem Zählvorgang den Wert 2 dazu addieren soll?

Der ADDI-Befehl ist dafür zuständig, immer einen konstanten Wert zu addieren. Der bisherige Befehls-Code hierfür war: **510**. Wenn wir das bisherige Programm bei Adresse 01 durch den Befehls-Code **520** abändern (wobei alle übrigen Befehle unverändert bestehen bleiben), wird der Zählvorgang erheblich beschleunigt, weil der Computer jetzt nicht mehr 1-2-3-4-5... sondern 2-4-6-8-A... hexadezimal in der rechten Anzeigestelle zählt. Würden wir bei der Adresse 01 den Befehls-Code **550** eingeben, wird der konstante Wert 5 addiert.

Für diese Programm-Änderung müssen wir nicht das ganze Programm neu eingeben, sondern wir gehen direkt auf die zu ändernde Adresse 01 durch: HALT – NEXT – 01. Der Computer zeigt uns die Adresse 01 und den bisherigen Befehls-Code **510**. Durch Eingabe des neuen Befehls-Codes, z. B. **520** wird der alte Befehl gelöscht und der neue Befehl durch NEXT im Programm-Speicher gespeichert. Programm-Änderung beendet, also: HALT – NEXT – 00 und Programm-Start mit RUN.

Unsere Computer-Kenntnisse haben nun einen Stand erreicht, welcher uns befähigen wird, in Kürze unser erstes Programm selbst zu entwickeln. Vermutlich haben wir jedoch den Zusammenhang zwischen dem Befehls-Code (der im großen Programm-Speicher abgespeichert wird) und den Registern (die in kleinen Register-Speichern abgespeichert werden) noch nicht so ganz richtig begriffen. Wir lernen die Zusammenhänge schnell, **wenn wir uns vorstellen**, wir würden den vorangegangenen 3-stelligen Zähler durch die Taste HALT an irgendeiner Stelle abbrechen. Auf dem Display **können** z. B. folgende Werte angezeigt sein:



## Ein 3-stelliger Automatik-Zähler

Wir wollen einen 3-stelligen Zähler programmieren (HALT NEXT 00), und wir nehmen folgende Eingabe vor:

Adresse	Eingabe Befehls-Code	Befehlskürzel (Mnemonic)	Kurzerklärung
00	<b>F30</b>	DISP 3,0	3 Stellen anzeigen
01	<b>510</b>	ADDI 1,0	konstanten Wert 1 dazuaddieren
02	<b>FB1</b>	ADC 1	Übertrag auf 2. Stelle übernehmen
03	<b>FB2</b>	ADC 2	Übertrag auf 3. Stelle übernehmen
04	<b>C00</b>	GOTO 00	zurück zu Adresse 00

Programm-Start: HALT – NEXT – 00 – RUN

Der Computer zählt auf 3 Stellen. Die rechte Stelle wird am schnellsten, die linke Stelle am langsamsten abgearbeitet. Das ist logisch, denn immer wenn die sehr schnell zählende rechte Stelle den höchsten dezimalen Wert F überspringt, gelangt der Übertrag auf die mittlere Stelle. Da auch die mittlere Stelle nach einiger Zeit den höchsten Wert F erreicht, erfolgt der Übertrag zur linken Display-Stelle.

Beim Vergleich mit dem vorangegangenen Programm sehen wir, daß wir lediglich bei der Adresse 00 den DISP-Befehl F20 durch F30 so geändert haben, (daß 3 Stellen angezeigt werden) und bei der Adresse 03 haben wir einen weiteren ADC-Befehl eingefügt, damit der Übertrag auch auf der 3. Stelle registriert wird.

Wie wir uns erinnern, stehen uns 16 verschiedene Register zur Verfügung. Damit wir und der Computer diese 16 Register unterscheiden können, ist die fortlaufende hexadezimale Numerierung von 0 – F möglich. Für unseren 3-stelligen Automatik-Zähler haben wir von den 16 Register-Möglichkeiten nur 3 benötigt, nämlich das Register Nr. 0, Nr. 1 und Nr. 2. Alle übrigen Register-Möglichkeiten wurden bei diesem Programm nicht verwendet. Die obige Darstellung zeigt als Beispiel, daß im Register Nr. 0 der Wert 5 angezeigt und demnach auch gespeichert ist. Im Register 1 sehen wir den Wert 2 und in Register 2 den Wert C.

Sehen wir uns doch noch einmal unsere letzte Programmierung an.

Adresse	Befehls-Code	Erklärung
00	F30	es sollen 3 Stellen angezeigt werden (F30) ab Register 0 (F30)
01	<b>510</b>	<b>konstanten Wert 1 (510)</b> dazuaddieren und das Ergebnis in Register 0 (510) speichern
02	<b>FB1</b>	<b>Übertrag (von Register 0) registrieren (FB) und in Register 1 (FB1) übernehmen</b>
03	FB2	Übertrag (von Register 1) registrieren und in Register 2 (FB2) übernehmen
04	C00	zu Adresse 00 (C00) zurückspringen, dort bisheriges Ergebnis anzeigen – weiterzählen – der Programm-Lauf wiederholt sich

Stellen wir uns vor, beim Programm-Start wird auf allen 3 Stellen jeweils die Ziffer 0 angezeigt. In Register Nr. 0 (rechte Stelle) ist der Wert 0 enthalten, ebenfalls in Register 1 (mittlere Stelle) und im Register 2 (linke Stelle). Bei der Adresse 01 wird

der konstant mit 1 eingegebene Wert dazuaddiert, bei den Adressen 02 und 03 ergibt sich noch kein Übertrag, bei 04 springen wir zum Programmanfang zurück, also wird bei Adresse 00 das bisherige Ergebnis: „1“ angezeigt. Das Spiel wiederholt sich: Bei Adresse 01 wird wieder der konstante Wert 1 dazuaddiert, bei 02 und 03 ergibt sich kein Übertrag, bei 04 Rücksprung zum Anfang und bei Adresse 00 wird der neue Wert „2“ angezeigt. Dieses Spiel wiederholt sich 15 Mal, bis in Register 0 (rechte Anzeigestelle) der Wert F erreicht wurde. Wird nun beim 16. Durchgang der Wert F durch den konstant hinzugefügten Wert 1 „übersprungen“, beginnt in Register 0 (rechte Anzeigestelle) der Zählvorgang wieder bei 0. Zuvor wurde jedoch das CARRY-Signal ausgelöst, was durch den Befehl FB1 (Adresse 02) registriert wurde, wodurch dieser erste Übertrag in Register 1 (mittlere Anzeigestelle) durch den sich zunächst ergebenden Wert 1 angezeigt wird. Bei jedem neuen Übertrag aus Register 0 (rechte Anzeigestelle) erhöht sich auch der Wert in Register 1 (mittlere Anzeigestelle) solange, bis auch dort der Wert F übersprungen wird. Dieser Übertrag wird vom Befehl FB2 (Adresse 03) registriert, wodurch jetzt auch im Register 2 (linke Anzeigestelle) ein erster Wert gespeichert und angezeigt wird.

Bei einem Befehls-Code wird also auch die jeweilige Register-Nummer (in diesem Fall Anzeigestelle) mit erwähnt, damit der Computer weiß, in welcher Anzeigestelle (d. h. in welcher Register-Nummer) die jeweilige Rechenoperation auszuführen ist.

## Unser erstes selbstprogrammiertes Programm

Auf Grund der soeben gewonnenen Erkenntnisse wollen wir unser erstes Programm „schreiben“.

**Aufgabenstellung:** Der Computer soll ein 6-stelliges Ergebnis anzeigen. Er soll automatisch zählen und bei jedem Zählvorgang soll der konstante Wert 2 dazuaddiert werden. Wird auf der rechten Anzeigestelle das höchste hexadezimale Ergebnis F übersprungen, sollen diese Übertrag-Werte ebenfalls gespeichert und angezeigt werden. Durch laufendes Zählen des Computers sollen die Ergebnisse letzten Endes auf allen 6 Stellen des Displays angezeigt werden.

**Unsere Überlegungen:** Der Anzeige-Befehl DISP soll 6 Stellen anzeigen, beginnend bei Register 0 (rechte Anzeigestelle) bis zum Register 5 (linke äußerste Anzeigestelle). Mit dem ADDI-Befehl soll ein konstanter Wert 2, in welches Register addiert werden? Sobald ein Register durch die höchste hexadezimale Ziffer F übersprungen wird, soll der Wert im nächsten Register gespeichert werden. Der Zählvorgang soll ständig wiederholt und die Ergebnisse angezeigt werden.

Wir tragen den jeweiligen Befehls-Code und den Befehlskürzel in unten stehende Tabelle ein.

Adresse	Eingabe Befehls-Code	Befehlskürzel	Erklärungen
00			
01			
02			
03			
04			
05			
06			
07			

Nun nehmen wir die Programmierung vor, indem wir den Befehls-Code eingeben. Im Eifer des Gefechts vergessen wir nicht: HALT – NEXT – 00.

Nun sind wir gespannt, ob wir unserem Computer die notwendigen logischen Befehle gegeben haben, damit er die Programm-Aufgabe durchführt. Wir starten mit RUN. Sollte unser selbsterstelltes Programm einen Fehler aufweisen, können wir mit der Lösung (siehe Tabelle unten) vergleichen und notfalls berichtigen. Sollten wir jedoch unserem Computer wider Erwarten einen totalen Unsinn eingegeben haben und er streikt (reagiert auf Tasten-Eingabe nicht mehr), erinnern wir uns an die grüne RESET-Taste auf der Computer-Platine, die uns zu einem neuen Start verhelfen wird.

Nachdem unser erstes Programm einwandfrei läuft, können wir uns selbst auf die Schulter klopfen – den schwierigsten Teil haben wir geschafft! Jetzt wird es mit jeder neuen Seite des Anleitungsbuches interessanter.

### Lösung unseres ersten selbstprogrammierten Programmes:

Adresse	Eingabe Befehls-Code	Befehlskürzel	Erklärungen
00	<b>F60</b>	DISP 6,0	6 Stellen anzeigen, ab Register Nr. 0
01	<b>520</b>	ADDI 2,0	konstanten Wert 2 zu Register Nr. 0 dazuaddieren
02	<b>FB1</b>	ADC 1	bei Übertrag den Wert in Register Nr. 1 übernehmen
03	<b>FB2</b>	ADC 2	bei Übertrag von Register Nr. 1 den Wert in Register Nr. 2 übernehmen
04	<b>FB3</b>	ADC 3	bei Übertrag von Register Nr. 2 den Wert in Register Nr. 3 übernehmen
05	<b>FB4</b>	ADC 4	bei Übertrag von Register Nr. 3 den Wert in Register Nr. 4 übernehmen
06	<b>FB5</b>	ADC 5	bei Übertrag von Register Nr. 4 den Wert in Register Nr. 5 übernehmen
07	<b>C00</b>	GOTO 00	Rücksprung zum Programm-Anfang und gespeicherte Ergebnisse anzeigen

## Zusammenfassung der ADD-Befehle

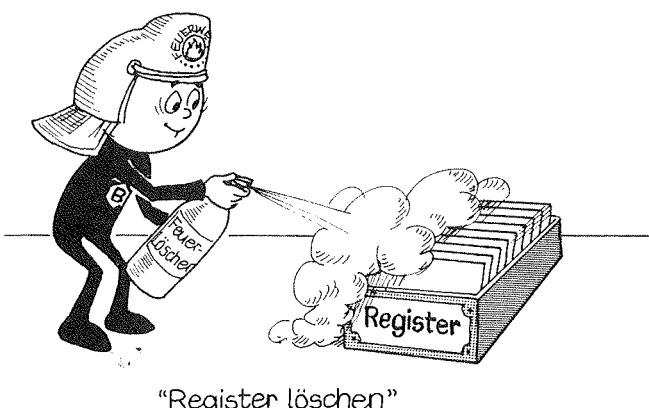
Der besseren Übersicht wegen, sollen die zuletzt gelernten Befehle noch einmal dargestellt werden:

Befehls- kürzel (Mnemonic)	Befehls- Code	Erklärungen
<b>ADD</b>	4sd	<b>Additionsbefehl</b> Der Befehls-Code ist <b>4</b> . Für <b>s</b> und <b>d</b> müssen 2 Register-Adressen (Register-Nr.) angegeben werden. Die Inhalte der beiden Register werden addiert und das Ergebnis in Register <b>d</b> (Ziel-Register/Destination-Register) gespeichert. Ist das gespeicherte Ergebnis größer als F, leuchtet das Carry-Flag auf.
<b>ADDI</b>	5nd	<b>Additionsbefehl mit einem konstanten Wert.</b> Der eigentliche ADDI-Befehl ergibt sich durch <b>5</b> . Für <b>n</b> muß ein konstanter Wert eingesetzt werden, welcher zum Register <b>d</b> hinzugefügt wird. Ist das Ergebnis größer als F, leuchtet das Carry-Flag auf.
<b>ADC</b>	FBd	<b>addiere Carry-Übertrag (ADD-Carry).</b> In das Register <b>d</b> wird immer der Wert <b>1</b> hinzugefügt, sobald ein Übertrag stattgefunden hat (wenn Carry-Flag leuchtet).

### Anmerkung zum Carry-Flag:

Immer wenn das Carry-Flag gesetzt wird, leuchtet die Carry-LED auf. Das Carry-Flag wird zurückgesetzt (Carry-LED erlischt) bei der nächsten Addition ohne Übertrag.

## Wir wollen Register-Inhalte „löschen“



Wir möchten nun gerne erreichen, daß wir die auf dem Display angezeigten Zahlen-Werte „löschen“ können, d. h., daß auf allen Textstellen des Displays die Ziffer 0 erscheint. Hierfür benötigen wir einen neuen Befehl: Den **CLEAR**-Befehl. Mit diesem „Löschen“-Befehl wird der Inhalt aller Register auf 0 gesetzt. Dies wird uns schnell deutlich, wenn wir eine neue Programmierung lt. nachfolgender Tabelle vornehmen. Zuerst gehen wir jedoch wieder auf den Programm-Anfang: HALT – NEXT – 00.

Adresse	Eingabe Befehls- Code	Befehls- kürzel	Erklärungen
00	<b>F60</b>	DISP 6,0	Inhalt ab Register 0 anzeigen
01	<b>FF0</b>	KIN 0	Eingabe einer Zahl (Wert) in Register 0
02	<b>FF1</b>	KIN 1	Eingabe einer Zahl (Wert) in Register 1
03	<b>FF2</b>	KIN 2	Eingabe einer Zahl (Wert) in Register 2
04	<b>FF3</b>	KIN 3	Eingabe einer Zahl (Wert) in Register 3
05	<b>FF4</b>	KIN 4	Eingabe einer Zahl (Wert) in Register 4
06	<b>FF5</b>	KIN 5	Eingabe einer Zahl (Wert) in Register 5
07	<b>FF6</b>	KIN 6	Eingabe einer Zahl (Wert) in Register 6
08	<b>F08</b>	CLEAR	alle Registerinhalte löschen (auf 0 setzen)
09	<b>C00</b>	GOTO 00	Rücksprung zur Adresse 00 (Programm-Anhang)

Programm-Start: HALT – NEXT – 00 – RUN

Wir geben nacheinander 6 verschiedene Zahlen-Werte ein, z. B. 5-1-4-3-9-6. Die zuerst eingegebene Zahl erscheint auf dem Display ganz rechts, die zuletzt eingegebene Zahl ganz links, also: 6-9-3-4-1-5. Nun geben wir noch eine 7. beliebige Zahl (oder Buchstaben) ein – das Display zeigt auf allen Stellen die Ziffern 0.

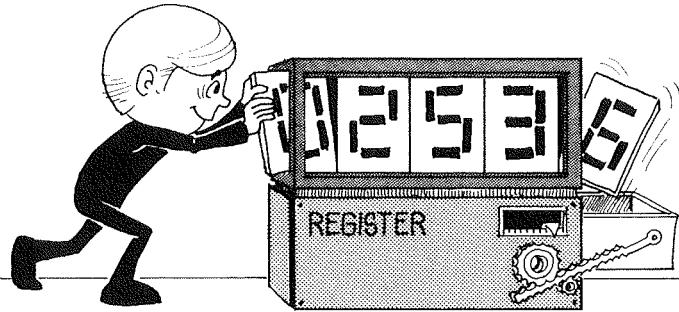
Der Computer hat befehlsgemäß reagiert: Mit dem DISP-Befehl (F60) wurde die Anzeige von 6 Stellen ab Register 0 befohlen. Mit den KIN-Befehlen, z. B. FF0 haben wir die Möglichkeit, in Register **0** einen Wert einzugeben und zu speichern, z. B. die Ziffer 5. Mit FF1 haben wir in Register **1** einen weiteren Wert, z. B. die Ziffer 1 gespeichert. FF3 hat die 3. eingegebene Zahl in Register **3** gespeichert usw. bis durch FF5 im Register 5 (also auf der 6. Anzeigenstelle) die letzte anzugezeigende Zahl gespeichert wurde. Mit FF6 haben wir eine weitere Zahl (oder Buchstaben) eingegeben, der ebenfalls gespeichert, jedoch nicht mehr angezeigt wurde. Wir können ja nur 6 Stellen anzeigen und haben hierfür die Register 0 – 5 verwendet. Logisch, daß dieser an 7. Stelle eingegebene Wert nicht mehr angezeigt werden kann. Dieser 7. eingegebene Wert hat jedoch den CLEAR-Befehl ausgelöst – alle Register-Inhalte wurden gelöscht (auf 0 gesetzt), weshalb auch das Display nur noch die Nullen anzeigt. Geben wir wieder 6 andere Werte ein, werden diese angezeigt bis durch die 7. Eingabe der CLEAR-Befehl wieder alle eingegebenen Register-Werte auf 0 setzt.

Wir haben nebenbei bemerkt, daß wir zwar nur 6 Stellen auf dem Display anzeigen können, daß wir jedoch in einem 7. Register ebenfalls einen Wert eingegeben haben, der uns jedoch unsichtbar geblieben ist. Wir erinnern uns, daß wir insgesamt Speichermöglichkeiten in 16 verschiedenen Registern haben, wobei im vorangegangenen Beispiel nur 7 Register belegt wurden.

### Der CLEAR-Befehl F08

Der Befehls-Code des CLEAR-Befehls lautet immer F08. Im Befehls-Code ergeben sich keine Variationsmöglichkeiten. Der Befehl hat grundsätzlich die Aufgabe, alle Register auf 0 zu setzen.

## Wir „verschieben“ Register-Werte



“Register-Werte verschieben”

Wenn wir auf dem Display die Zahl 589 anzeigen möchten, müßten wir beim vorangegangenen Versuch in der Reihenfolge 9-8-5 eingeben, weil die zuerst eingegebene Zahl rechts erschien und die zuletzt eingegebene Zahl links angezeigt wurde. Wie beim Taschenrechner sollen jetzt die Zahlen in der richtigen Reihenfolge auf dem Display erscheinen, d. h. die zuerst eingegebene Ziffer wird zunächst ganz rechts angezeigt. Geben wir eine zweite Zahl ein, muß die erste Zahl um eine Stelle nach links verschoben werden. Bei einer dritten eingegebenen Zahl müssen die beiden vorangegangenen ebenfalls wieder eine Stelle nach links geschoben werden. Hierfür verwenden wir einen Schiebe-Befehl: Den **MOV**-Befehl.

Der Computer soll uns zeigen, wie die Sache funktioniert. Wie üblich: HALT – NEXT – 00, dann Programm-Eingabe gemäß nachfolgender Tabelle:

Adresse	Eingabe Befehls- Code	Befehls- kürzel	Erklärungen
00	<b>F08</b>	CLEAR	alle Register-Inhalte löschen
01	<b>F31</b>	DISP 3,1	3 Stellen ab Register 1 anzeigen
02	<b>FF0</b>	KIN 0	eingegebenen Wert in Register 0 speichern
03	<b>023</b>	MOV 2,3	Inhalt von Register 2 nach Register 3 schieben
04	<b>012</b>	MOV 1,2	Inhalt von Register 1 nach Register 2 schieben
05	<b>001</b>	MOV 0,1	Inhalt von Register 0 nach Register 1 schieben
06	<b>C01</b>	GOTO 01	Rücksprung auf Adresse 01

Programm-Start: HALT – NEXT – 00 – RUN

Wir geben langsam und nacheinander 3 Ziffern ein, und wir sehen, wie mit jeder neu eingegebenen Zahl der vorangegangene Wert um eine Stelle nach links geschoben wird.

Wenn wir uns bei obiger Tabelle die Erklärungen ansehen, fällt uns auf, daß wir auf Adresse 00 den CLEAR-Befehl eingegeben haben, wodurch alle evtl. vorhandenen Register-Inhalte gelöscht wurden. Daher kann der bei Adresse 01 vorhandene Anzeige-Befehl DISP auch nur Nullen auf dem Display anzeigen. Beim DISP-Befehl (F31) fällt uns auf, daß wir nicht wie sonst üblich, die Werte ab Register 0, sondern die Anzeige ab Register 1 veranlaßen. Auf Adresse 02 konnten wir mit FF0 einen Wert in Register 0 eingeben und speichern, welcher jedoch nicht angezeigt wurde. Durch die MOV-Befehle (Adressen 03 – 05) haben wir die Register-Inhalte jeweils um eine Stelle nach links verschoben.

Wir werden uns fragen, warum die zuerst (in Register 0) eingegebene Zahl angezeigt wird, wenn der Anzeige-Befehl beauftragt ist, das Register 0 gar nicht anzusegn?

Durch den MOV-Befehl auf Adresse 05 haben wir die in Register 0 eingegebene Zahl (Wert) in das Register 1 geschoben und ab Register 1 werden die Ergebnisse angezeigt.

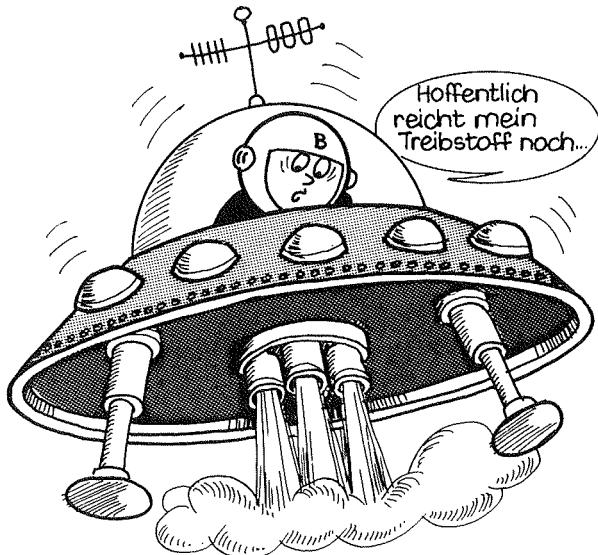
Der **MOV**-Befehl wird mit dem Befehls-Code **0** eingeleitet. Durch **s** wird der Inhalt des Source-(Quelle) Registers nach **d**, also zum Destination-(Ziel) Register übertragen. Der Inhalt des Source-Registers bleibt dabei erhalten. Beispiel: Wir hatten beim vorangegangenen Programm unter Adresse 04 den MOV-Befehls-Code **012** eingegeben. **0** bringt die Auslösung des MOV-Befehls. **1** hat den Wert von Register 1 zu **2**, also in Register 2 geschoben.

Vielleicht tun wir uns noch ein bißchen schwer, weil wir für einen scheinbar gleichen Befehl mehrere Bezeichnungen erfahren. So haben wir gesehen, daß der MOV-Befehl einerseits den Befehls-Code **0sd** hat, während wir beim vorangegangenen Programm, z. B. unter Adresse 03 den Befehls-Code **023** sehen, wobei unter „Befehlskürzel“ zu allem Überfluß auch noch **MOV 2,3** genannt wird. Der MOV-Befehl ist jedoch nicht immer der gleiche Befehl – es ist ein variabler Befehl. Der bei Adresse 03 verwendete Befehls-Code **023** sagt uns nicht viel. Der dazugehörige Befehlskürzel **MOV 2,3** sagt dagegen sehr deutlich: **MOV** = Schieben, **2** = Register 2, **3** = Register 3.

Unter Adresse 04 haben wir ebenfalls den Schiebe-Befehl: **MOV 1,2**, also einen Schiebe-Befehl, der den Inhalt der Register **1** und **2** betrifft. Nachdem wir unter Adresse 05 noch den Befehl **MOV 0,1** erkennen, ist es völlig klar, daß wir mit dem MOV-Befehl die verschiedensten Register ansprechen können. Das gleiche besagt der grundsätzliche Befehls-Code **0sd**, wobei wir für **s** und für **d** die Register-Nummern einsetzen, deren Inhalte zu verschieben sind. **s** geht nach **d**. Wenn wir also den Befehls-Code **023** verwenden, heißt dies nichts anderes als: **0** = schieben, **2** (Inhalt Register 2) nach **3** (Inhalt Register 3) verschieben. Es ist uns also verständlich, daß wir für **s** und **d** die anzusprechende Register-Nummern einsetzen.

**Das erste große Spiel-Programm mit physikalischen Rechenoperationen:**

## Die Mondlandung



Genug der Theorie – unser Computer gibt uns nun eine erste Kostprobe seines Könnens. Wir simulieren die Landung einer Raumfahrt auf dem Mond. Damit der Computer die schwierigen Berechnungen vornehmen kann, geben wir ihm das umfangreiche aus 133 Adressen bestehende Programm gemäß nachfolgender Tabelle ein.

### Spielverlauf:

Die Mondfahrt befindet sich im Landeanflug. Im Bord-Computer sind alle Daten (durch die vorher auszuführende Programm-Eingabe) gespeichert, wie z. B. momentane Geschwindigkeit, Gewicht der Raumfahrt, Anziehungskraft des Mondes, Treibstoff-Vorrat und Entfernung bis zur Mondoberfläche.

Wir betätigen die **Taste A** – auf dem Display erscheint die Meldung **A0 500**. Die Meldung besagt, daß sich das Raumschiff in diesem Augenblick 500 Meter über der Mondoberfläche befindet.

Durch erneute Betätigung der Taste A erscheint: **B0 050**: Die Raumfahrt fliegt momentan mit einer Geschwindigkeit von 50 Metern pro Sekunde auf die Mondoberfläche zu.

Erneute Betätigung der Taste A zeigt: **C0 120**: Für unser Bremsmanöver stehen uns momentan noch 120 kg Treibstoff zur Verfügung.

Betätigung der Taste A zeigt die Meldung: **00**. Der Computer erwartet unseren Befehl, wieviel kg Treibstoff für ein erstes Bremsmanöver verwendet werden sollen. Wir geben über die Zahlentasten z. B. „20“ ein.

Durch erneute Betätigung der Taste A wird die Eingabe abgeschlossen. Der Computer meldet die neue inzwischen erreichte Höhe: **A0 465**. Unter **A0** erhalten wir also jeweils die Höhenangabe, z. B. **465** Meter.

Wird wieder die Taste A betätigt, meldet der Computer die neue Sinkgeschwindigkeit, z. B. **B0 035**, d. h. die Geschwindigkeit beträgt nur noch **35** Meter pro Sekunde.

Nochmals A betätigen, der Computer meldet den verbleibenden Treibstoff-Vorrat, z. B.: **C0 100**, es sind also noch **100** kg Treibstoff vorhanden.

Wieder Taste A drücken. Meldung: **00** – der Computer erwartet eine erneute Eingabe für den weiter erforderlichen Bremsvorgang. Wir entscheiden uns nochmals 30 kg Treibstoff zu opfern. Nach Tastendruck A meldet der Computer: **A0 455**, d. h. wir haben kräftig gebremst und befinden uns noch **455** Meter über der Mondoberfläche. Wieder Taste A, Meldung: **B0**

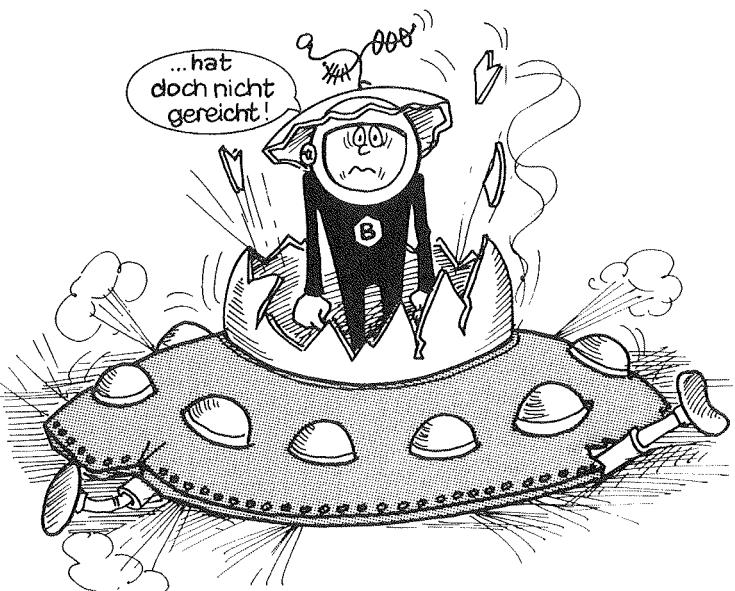
010, d. h. unsere Sinkgeschwindigkeit beträgt nur noch 10 Meter pro Sekunde. Taste A bringt die Meldung: **C0 070** – wir haben noch **70** kg Treibstoff-Vorrat. Taste A – Meldung **00** – Computer erwartet weitere Eingaben.

Für unsere weiteren Entscheidungen müssen wir beachten, daß der Computer tatsächlich physikalische Gegebenheiten berücksichtigt. Bremsen wir am Anfang des Landeanflugs zu stark, wird u. U. die Anziehungskraft des Mondes aufgehoben, das Raumschiff entfernt sich und eine weiche Mondlandung ist nicht mehr möglich. Bremsen wir zu wenig, wird die Sinkgeschwindigkeit nicht genügend gemindert – im Gegenteil, durch die Mondanziehungskraft wird sich die Fallgeschwindigkeit sogar erhöhen. Verbrauchen wir am Anfang zu viel Treibstoff, können wir u. U. bei der sich zum Schluß steigernen Mondanziehungskraft nicht mehr genügend abbremsen – das Raumschiff zerschellt auf der Mondoberfläche. Wir können aber auch im Anfangsstadium gar nicht bremsen (bei Meldung **00** wird dann keine Treibstoffmenge eingegeben, sondern direkt wieder die Taste A betätigt) und bewahren uns die Treibstoff-Vorräte für die Schlußphase auf. Ziel des simulierten Landeanfluges ist es, die Treibstoff-Vorräte so einzusetzen, daß die Landefähre mit einer möglichst geringen Geschwindigkeit auf der Mondoberfläche aufsetzt. Am Spielende gibt es 4 verschiedene Anzeigmöglichkeiten auf dem Display:

1. **EEEEEE mit Dauerton** – das Raumschiff wurde weich gelandet.
2. **AAAA mit unterbrochenem Ton** = die Fähre ist mit zu großer Geschwindigkeit auf dem Mond aufgeschlagen und wurde zerstört.
3. **AEAE mit unterbrochenem Ton** = der Treibstoff-Vorrat wurde vorzeitig verbraucht, es kann nicht mehr gebremst werden, ein Aufschlag auf der Mondoberfläche wird erfolgen.
4. **AFAF mit unterbrochenem Ton** = es wurde zu stark gebremst, die Raumfahrt entfernt sich wieder vom Mond, eine weiche Landung ist nicht mehr möglich.

Nach Anzeige des Spielendes kann durch erneute Betätigung der Taste A ein neuer Landeanflug begonnen werden.

Wir werden feststellen, daß eine weiche Mondlandung gar nicht so einfach ist. Der Computer errechnet uns zwar die momentanen Daten, wir müssen ihm jedoch die richtigen Entscheidungen eingeben. Interessant und wichtig zu wissen ist es, daß zum Ausgleich der Mondanziehungskraft 5 kg Treibstoff notwendig sind, d. h., daß bei Eingabe der Ziffer 5 die Sinkgeschwindigkeit konstant bleibt. Wird mehr als 5 kg Treibstoff eingegeben, ergibt sich eine Bremswirkung. Ist die Sinkgeschwindigkeit bereits bei 0 angekommen, kann sich eine Beschleunigung ergeben, wodurch eine Landung nicht mehr möglich ist.



## Programm-Eingabe „Mondlandung“

Mit HALT – NEXT – 00 zum Programm-Anfang gehen und Befehlscodes lt. Tabelle eingeben. Nach jeder Befehlseingabe Taste NEXT nicht vergessen.

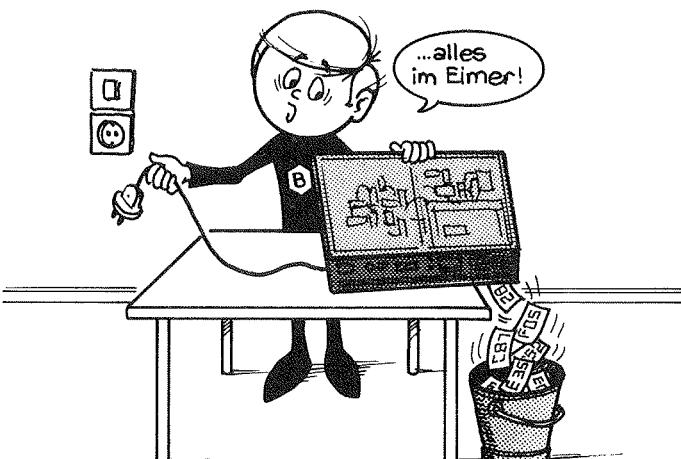
Adr.	Eingabe-Befehl	Befehlskürzel (Mnemonics)
00	F 0 2	DISOUT
01	F 0 8	CLEAR
02	F E 0	DOT 0
03	1 4 2	MOVI 4,2
04	1 F 3	MOVI F,3
05	1 1 4	MOVI 1,4
06	1 2 5	MOVI 2,5
07	1 3 6	MOVI 3,6
08	1 8 7	MOVI 8,7
09	1 7 8	MOVI 7,8
0A	1 A 1	MOVI A,1
0B	0 2 D	MOV 2,D
0C	0 3 E	MOV 3,E
0D	0 4 F	MOV 4,F
0E	F 0 3	HDXZ
0F	F 5 D	DISP 5,D
10	F F B	KIN B
11	F 0 2	DISOUT
12	1 B 1	MOVI B,1
13	1 0 F	MOVI 0,F
14	0 5 D	MOV 5,D
15	0 6 E	MOV 6,E
16	F 0 3	HDXZ
17	F 5 D	DISP 5,D
18	F F B	KIN B
19	F 0 2	DISOUT
1A	1 C 1	MOVI C,1
1B	0 7 D	MOV 7,D
1C	0 8 E	MOV 8,E
1D	F 0 3	HDXZ
1E	F 5 D	DISP 5,D
1F	F F B	KIN B
20	1 0 D	MOVI 0,D
21	1 0 E	MOVI 0,E
22	F 2 D	DISP 2,D
23	F F B	KIN B
24	9 9 B	CMPI 9,B
25	D 2 9	BRC 29
26	0 D E	MOV D,E
27	0 B D	MOV B,D
28	C 2 2	GOTO 22
29	F 0 2	DISOUT
2A	1 0 F	MOVI 0,F
2B	F 0 4	DZHX
2C	6 D 7	SUB D,7
2D	F C 8	SUBC 8
2E	D 6 9	BRC 69
2F	6 E 8	SUB E,8
30	D 6 9	BRC 69
31	7 5 D	SUBI 5,D
32	F C E	SUBC E
33	D 5 A	BRC 5A
34	4 D 2	ADD D,2
35	F B 3	ADC 3
36	F B 4	ADC 4

Adr.	Eingabe-Befehl	Befehlskürzel (Mnemonics)
37	4 E 3	ADD E,3
38	F B 4	ADC 4
39	6 5 2	SUB 5,2
3A	F C 3	SUBC 3
3B	F C 4	SUBC 4
3C	D 7 B	BRC 7B
3D	6 6 3	SUB 6,3
3E	F C 4	SUBC 4
3F	D 7 B	BRC 7B
40	6 D 5	SUB D,5
41	F C 6	SUBC 6
42	D 8 0	BRC 80
43	6 E 6	SUB E,6
44	D 8 0	BRC 80
45	9 0 4	CMPI 0,4
46	D 0 A	BRC 0A
47	9 0 3	CMPI 0,3
48	D 0 A	BRC 0A
49	9 5 2	CMPI 5,2
4A	D 0 A	BRC 0A
4B	9 0 6	CMPI 0,6
4C	D 0 A	BRC 0A
4D	9 5 5	CMPI 5,5
4E	D 0 A	BRC 0A
4F	1 E 0	MOVI E,0
50	1 E 1	MOVI E,1
51	1 E 2	MOVI E,2
52	1 E 3	MOVI E,3
53	1 E 4	MOVI E,4
54	1 E 5	MOVI E,5
55	1 F 6	MOVI F,6
56	F E 6	DOT 6
57	F 6 0	DISP 6,0
58	F F 0	KIN 0
59	C 0 0	GOTO 00
5A	6 D F	SUB D,F
5B	6 F 2	SUB F,2
5C	F C 3	SUBC 3
5D	F C 4	SUBC 4
5E	D 7 B	BRC 7B
5F	6 5 2	SUB 5,2
60	F C 3	SUBC 3
61	F C 4	SUBC 4
62	D 7 B	BRC 7B
63	6 6 3	SUB 6,3
64	F C 4	SUBC 4
65	D 7 B	BRC 7B
66	4 F 5	ADD F,5
67	F B 6	ADC 6
68	C 4 5	GOTO 45
69	1 E 0	MOVI E,0
6A	1 A 1	MOVI A,1
6B	1 E 2	MOVI E,2
6C	1 A 3	MOVI A,3
6D	1 F F	MOVI F,F

Adr.	Eingabe-Befehl	Befehlskürzel (Mnemonics)
6E	1 F E	MOVI F,E
6F	F E F	DOT F
70	7 1 E	SUBI 1,E
71	D 7 3	BRC 73
72	C 7 0	GOTO 70
73	F 4 0	DISP 4,0
74	F E F	DOT F
75	1 0 D	MOVI 0,D
76	F E D	DOT D
77	7 1 E	SUBI 1,E
78	D 5 8	BRC 58
79	F 0 2	DISOUT

Adr.	Eingabe-Befehl	Befehlskürzel (Mnemonics)
7A	C 7 3	GOTO 73
7B	1 A 0	MOVI A,0
7C	1 A 1	MOVI A,1
7D	1 A 2	MOVI A,2
7E	1 A 3	MOVI A,3
7F	C 6 D	GOTO 6D
80	1 F 0	MOVI F,0
81	1 A 1	MOVI A,1
82	1 F 2	MOVI F,2
83	1 A 3	MOVI A,3
84	C 6 D	GOTO 6D

Damit das Spielende auch akustisch wahrgenommen werden kann, muß noch der Piezo-Summer (siehe auch Seite 7) angegeschlossen werden. Wenn wir nach der letzten Befehlseingabe nochmals die Taste NEXT betätigt haben, gehen wir wie üblich mit HALT – NEXT – 00 zum Programm-Anfang und starten mit RUN. Das Display muß jetzt die Meldung bringen: A0 500. Sollte der Computer diese Meldung nicht bringen (oder auf dem Display z. B. überhaupt nichts anzeigen) hat sich bei der Programm-Eingabe ein Fehler eingeschlichen. Wir erinnern uns, daß wir die Programm-Eingabe überprüfen können, und wir gehen nochmals mit HALT – NEXT – 00 zum Programm-Anfang und wir sehen unter Adresse 00 den ersten eingegebenen Befehl F02. Immer wenn wir die NEXT-Taste betätigen, wird uns der nächste Befehl angezeigt. Entdecken wir einen Fehler, betätigen wir zweimal die Taste C/CE, wodurch der falsche Befehl gelöscht und der richtige Befehl eingegeben werden kann. Mit HALT – NEXT – 00 und RUN wird das Programm neu gestartet. Sollte immer noch nicht die richtige Meldung kommen, muß das ganze Programm wie beschrieben kontrolliert und richtiggestellt werden.



### Wichtige Information bezüglich Netzstromanschluß!

Wir haben erstmals mit einem gewissen Zeitaufwand ein längeres Programm in unserem Computer gespeichert. Wenn wir unsere „Mondlandung“ einem Bekannten vorführen möchten, darf **die Zuleitung zur Netzsteckdose nicht unterbrochen werden**. Der Computer speichert das eingegebene Programm nur solange, bis die Stromzufuhr unterbrochen wird. Ohne Strom kann der Speicher des Computers die Informationen nicht behalten.

Wir können unseren Computer jedoch unbesorgt im Dauerbetrieb an der Netzsteckdose angeschlossen halten, ohne mit einer Beschädigung oder Abnutzung rechnen zu müssen. Der Stromverbrauch ist minimal (ca. 4 Watt) d. h., daß der Computer in 24 Stunden weniger Strom verbraucht, als eine 100 Watt

Haushalts-Glühlampe pro Stunde. Wir können den Computer in den Leerlauf-Zeiten, z. B. als Digital-Leuchtuhr weiterlaufen lassen, in dem wir uns mit den Tasten HALT – PGM – 4 die Uhrzeit anzeigen lassen. Wir erinnern uns, daß für die Uhrzeit der „Eingang 4“ mit „Takt/Clock“ auf der Computer-Platine verbunden sein muß. Eine Verbindungsleitung zum Piezosummer klemmen wir ab, weil sonst im Sekundentakt ein Pfeifton entsteht.

Übrigens, wenn wir das Gerät bereits einige Stunden im Betrieb haben, ohne daß die Uhrzeit eingestellt wurde (jedoch unter der Voraussetzung, daß die Verbindungsleitung zwischen „Eingang 4“ und Takt/Clock vorhanden war) zeigt uns der Computer mit PGM 4 an, wie lange das Gerät seit der letzten Netzunterbrechung in Betrieb gewesen ist. Wir haben einen automatischen Betriebsstundenzähler!

Wir erinnern uns auch, daß wir mit HALT – PGM – 3 die Uhrzeit neu einstellen können (und diese mit HALT – PGM – 4 anzeigen). Mit HALT – NEXT – 00 – RUN können wir jederzeit (wenn die Netzteitung inzwischen nicht unterbrochen wurde) wieder unsere Mondlandung aufrufen.

Auf Seite 58 werden wir erfahren, wie wir durch Batteriebetrieb ein eingegebenes Programm auch bei einer Netzunterbrechung im Speicher abrufbereit halten können.

## Probieren und experimentieren?!

Wir haben inzwischen so viel gelernt, daß wir ohne Angst (etwas kaputt zu machen) experimentieren können. Wir sollten ruhig versuchen, mit den bis jetzt bekannten Befehlen, die ersten eigenen Programm-Ideen zu verwirklichen. Wir können auch eines der im zweiten Teil des Anleitungsbuches enthaltenen größeren Programme ausprobieren.

Wenn wir ein neues Kapitel durchgearbeitet haben, sollten wir unbedingt Variationen der vorgegebenen Programm-Vorschläge ausprobieren. Auf diese Weise werden wir schnell mit unserem Computer vertraut. Wir wissen, daß wir mit HALT – NEXT – 00 jeweils zu der zuerst eingegebenen Programm-Adresse kommen, und daß wir das Programm mit RUN starten. Mit HALT kann das Programm jederzeit abgebrochen werden. Wenn wir z. B. unser zuvor eingegebenes Programm „Mondlandung“ im Computer-Speicher weiter halten möchten (keine Netzzstrom-Unterbrechung vorausgesetzt) sollten wir allerdings für ein neu einzugebendes Programm nicht mit HALT – NEXT – 00 fortfahren, weil wir ja mit 00 auf die erste eingegebene Programm-Adresse zurück springen würden. Wenn wir bei der Adresse 00 einen neuen Befehl eingeben, wird der dort vorhandene Befehl gelöscht und unser Programm „Mondlandung“ funktioniert nicht mehr.

Beim Programm „Mondlandung“ haben wir den letzten Befehl unter Adresse 84 mit C6D eingegeben. Wenn wir die Tabelle „Gegenüberstellung dezimales/hexadezimales Zahlen-System“ auf Seite 17 betrachten, erkennen wir, daß mit dem Programm „Mondlandung“, welches mit der hexadezimalen Ziffer 84 endet (= dezimal Nr. 132) erst etwa die Hälfte der Speicherkapazität unseres Computers belegt wurde. In unserem Speicher ist also noch genügend Platz, um weitere Programme einzugeben. Es ist zweckmäßig, einige Adressen-Nummern freizulassen und das folgende Programm mit der Adressen-Nummer 90 zu beginnen. In diesem Fall sagen wir dem Computer nicht wie bisher HALT – NEXT – 00, sondern HALT – NEXT – 90. Das nächste Programm beginnt also bei Adresse 90 und wenn wir dieses Programm aufrufen möchten, lautet logischerweise der Befehl HALT – NEXT – 90. Die Inbetriebnahme wie üblich mit RUN.

Für das im nächsten Kapitel beschriebene Experiment benötigen wir 14 Programm-Schritte, d. h., wenn wir mit der hexadezimalen Adresse 90 beginnen, wird die letzte belegte Adresse 9D sein. Wenn wir wieder unser „Mondlandespiel“ aufrufen möchten, erreichen wir dieses mit HALT – NEXT – 00 – RUN, während wir das neu eingegebene Programm mit HALT – NEXT – 90 aufrufen.

## Zwei wichtige Informationen sollten wir uns noch merken:

Gemäß Tabelle „dezimales/hexadezimales Zahlen-System“ können wir bis zu 255 Programm-Schritte (Adressen) speichern, d. h. bis zur hexadezimalen Adresse FF. Würden wir ein sehr langes Programm eingeben, wodurch die zur Verfügung stehenden 255 dezimalen Programm-Schritte nicht ausreichen, springt unser Computer bei der automatischen Adressen-Vergabe nach der hexadezimalen Ziffer FF automatisch zum Programm-Anfang 00 zurück. Ein dort bereits eingegebenes Programm wird gelöscht.

Wir können die Speicherkapazität des Computers bis zur Adresse FF ausnutzen. Wenn wir uns merken, bei welcher Adresse die verschiedenen Programme beginnen, können wir alle Adressen zwischen 00 und FF belegen. Außerdem steht uns jederzeit mit HALT – PGM – 4 die Uhrzeit zur Verfügung. Rufen wir jedoch mit PGM – 7 das Nimm-Spiel auf, so wird der Programm-Ablauf für dieses Spiel automatisch in unserem Programm-Speicher beginnend mit Adresse 00 eingespielt. Mit HALT – NEXT – 00 – RUN, können wir jetzt wieder das Nimm-Spiel durchführen. Da wir jedoch für das Nimm-Spiel die Speicherkapazität bis zur hexadezimalen Adresse 44 belegen, wird ein Teil unseres Programms „Mondlandung“ gelöscht. Würden wir mit HALT – NEXT – 45 den verbliebenen Rest des Programms „Mondlandung“ anwählen, wird unser Computer mit dem Programm-Rest immer noch weiterrechnen und Ergebnisse anzeigen, die jedoch durch den fehlenden Programm-Anfang sinnlos geworden sind.

Abschließend soll noch erwähnt werden, daß zur Übersichtlichkeit bei allen Programmen grundsätzlich die Start-Adresse 00 genannt wird. Es bleibt uns vorbehalten, evtl. mit einer anderen Adressen-Nummer (z. B. 90) zu starten, so daß wir dieses neue Programm mit HALT – NEXT – 90 – RUN aufrufen können.

## Vergleichen – ein wichtiger Befehl!

Bei den bisherigen Demonstrations-Programmen hat es uns vermutlich immer wieder gestört, daß der Computer mit dem ungewohnten hexadezimalen Zahlen-System arbeitet.

Mit dem folgenden automatischen Zähler möchten wir erreichen, daß dieser mit Dezimalzahlen arbeitet, d. h., daß nach der Ziffer 9 nicht wie bisher üblich mit A, B, C, D..., sondern mit 10, 11, 12, 13... weitergezählt wird.

Da Computer jedoch grundsätzlich immer hexadezimal arbeiten, müssen wir ihn durch ein Programm-Trick dazu bringen, nur noch die dezimalen Ergebnisse anzuzeigen: Mit HALT – NEXT – 00 geben wir folgendes Programm ein:

Adresse	Eingabe-Befehl	Mnemonic
00	<b>F08</b>	CLEAR
01	<b>F21</b>	DISP 2,1
02	<b>510</b>	ADDI 1,0
03	<b>FB1</b>	ADC 1
04	<b>991</b>	CMPI 9,1
05	<b>D07</b>	BRC 07
06	<b>C02</b>	GOTO 02
07	<b>561</b>	ADDI 6,1
08	<b>512</b>	ADDI 1,2
09	<b>992</b>	CMPI 9,2
0A	<b>DOC</b>	BRC 0C
0B	<b>C02</b>	GOTO 02
0C	<b>562</b>	ADDI 6,2
0D	<b>C02</b>	GOTO 02

Programm-Start: HALT – NEXT – 00 (oder 90) – RUN

Wir sehen, unser Computer zählt nun treu und brav von 0-99, um dann wieder von vorne zu beginnen. Die hexadezimalen Werte A-B-C-D-E-F sind nicht mehr vorhanden.

**Wichtiger Hinweis für alle, welche das Programm „Mondlandung“ nicht gelöscht und deshalb die neue Befehls-Eingabe mit Adresse Nr. 90 begonnen haben.**

Wir wundern uns, daß das Programm nicht wie vorgesehen funktioniert. Wenn wir bei Adresse 90 begonnen haben, kann der neue automatische Dezimalzähler nicht laufen – warum wohl?

Der Computer setzt unser logisches Denken voraus. Sehen wir uns doch nochmals das neu eingegebene Programm an. Gemäß Eingabe-Tabelle beginnt der erste Befehl mit Adresse 00 und der zuletzt eingegebene Befehl endet mit 0D. Da wir das Programm „Mondlandung“ nicht löschen wollten beginnt unser neues Programm nicht mit Adresse 00, sondern mit 90. In diesem Programm sind jedoch bei Adresse 05, 06, 0A, OB und 0D Sprung-Befehle enthalten, welche darauf abgestimmt sind, daß das Programm bei 00 beginnt. Unter Adresse 00 haben wir jedoch unsere „Mondlandung“.

Logik: Wenn wir die Adressen-Nummern ändern, müssen wir dies auch bei den Sprung-Befehlen berücksichtigen. In der Tabelle unter Adresse 05 lautet der Befehl D07, d. h., Sprung zur Adresse 07. Dieser Befehl muß in diesem Fall D97 lauten. Ebenso muß bei Adresse 06 der Befehl C02 geändert werden in C92 und bei Adresse 0A von D0C in D9C. Der Befehl C02 (Adresse OB) muß in C92 geändert werden. Auch der letzte Befehl unter Adresse 0D ändert sich von C02 in C92. Wenn wir jetzt wie üblich mit HALT – NEXT – 90 – RUN starten, funktioniert auch dieses Programm einwandfrei.

Wenn wir zukünftig ähnliche Experimente durchführen, sollten wir in der Programm-Tabelle die vorgedruckte Adressen-Nummer mit den tatsächlich verwendeten Adressen korrigieren und darauf achten, daß z. B. beim Sprung-Befehl die richtigen Adressen-Nummern verwendet werden.

Wir haben bei der Eingabe bereits festgestellt, daß in unserem Programm 3 neue Befehle enthalten sind. Bei der Adresse 00 finden wir den **CLEAR-Befehl F08**. Mit CLEAR werden alle Registerinhalte auf 0 gesetzt. Der DISP-Befehl unter Adresse 01 und der ADDI-Befehl unter Adresse 02 sind uns bekannt. Bei ADDI 1,0 wird im Register 0 wie bisher hexadezimal von 0-F gezählt. Dieses Register wird jedoch nicht angezeigt. Durch den Befehl (bei Adresse 03) ADC 1 wird jedoch jedesmal, wenn im Register 0 (bei ADDI 1,0) ein Zählvorgang die hexadezimale F nach 0 überspringt, der konstante Wert 1 beim Register Nr. 1 hinzugefügt. Bis hierher ist uns das Programm auf Grund der vorangegangenen Automatik-Zähler bekannt.

Neu ist der Vergleichs-Befehl **CMPI 9,1** bei Adresse 04. Dieser Befehl vergleicht den Inhalt (die Werte) von Register 1 mit der Ziffer 9. Ergibt sich im Register 1 ein Wert der größer als 9 ist (also z. B.: AB C...) wird das Carry-Flag gesetzt. Wir wissen, daß mit dem Carry-Flag ein Übertrag-Signal ausgelöst wird. Der unter Adresse 05 eingefügte neue Befehl **BRC 07** (aus dem englischen **branch if carry**) beobachtet das Carry-Flag. Wird das Carry-Flag gesetzt (die obere LED neben dem Display leuchtet) springt das Programm zur Adresse 07. Wenn kein Carry-Flag vorhanden ist, wird der nächste Befehl (Adresse 06 GOTO 02) bearbeitet, d. h., ohne Carry-Flag springt das Programm zur Adresse 02 zurück und setzt dort durch den Befehl ADDI 1,0 den Zählvorgang fort.

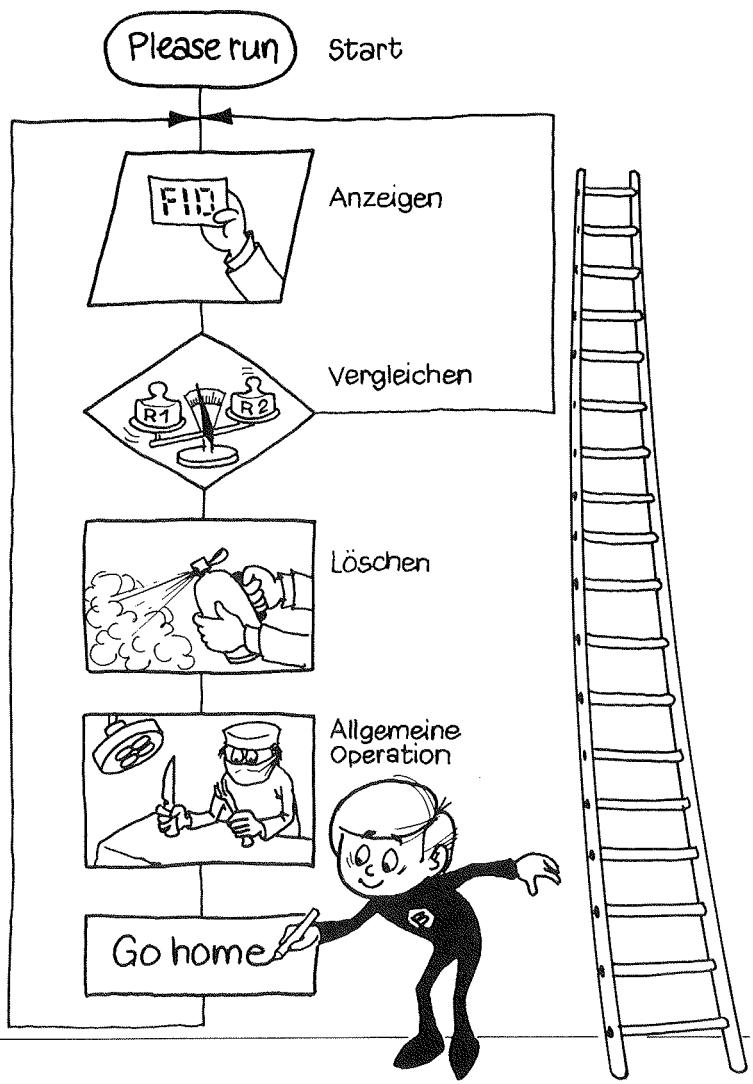
Bei ADDI wird im Register Nr. 1 fortlaufend von 1-9 gezählt. Nach der dezimalen 9 folgt das hexadezimale A. Sobald bei ADDI im Register 1 die A erscheint, wird durch den Befehl CMPI 9,1 das Carry-Flag gesetzt, weil A größer als 9 ist. Durch den nachfolgenden Befehl BRC 07 springt das Programm zur Adresse 07. Hier wird durch ADDI 6,1 der konstante Wert 6 zum Inhalt des Registers 1 hinzugefügt. Wir haben somit im Register 1 des ADDI-Befehls den Wert A, zu welchem der konstant eingegebene Wert 6 hinzugefügt wird. Wenn wir mit der Tabelle „dezimales/hexadezimales Zahlen-System“ (Seite 17 oder Buchleszeichen) vergleichen, stellen wir fest, daß hexadezimal gerechnet **A + 6** das hexadezimale Ergebnis **10** ergibt. Nun wissen wir jedoch, daß in einem Register nur einstellige Werte registriert werden. Bei ADDI 6,1 wird also von der zweistelligen Ziffer 10 nur der Wert 0 registriert. Die übrigbleibende 1 (von 10) wird durch den nächsten Befehl ADDI 1,2 in Register 2 hinzugefügt. Das Ergebnis 10 wird also in zwei verschiedenen Registern gespeichert. Der jetzt folgende Vergleichs-Befehl CMPI 9,2 vergleicht, ob der Inhalt von Register 2 größer als 9 ist und das Carry-Flag wird dementsprechend gesetzt. Ergibt sich ein Übertrag (Carry-Flag leuchtet) springt das Programm durch den Befehl BRC 0C zur Adresse 0C, dort wird der konstante Wert 6 zum Register 2 hinzugefügt.

Wie so manches Vorangegangene wird uns auch diese abenteuerliche Computer-Rechnerei zu einem späteren Zeitpunkt wesentlich einleuchtender werden, als die vorangegangene prinzipielle Darstellung.

## Zusammenstellung der neu gelernten Befehle

Befehls-kürzel (Mnemonic)	Befehls-Code	Erklärungen
<b>CLEAR</b>	<b>F08</b>	<b>Löschbefehl:</b> sämtliche Registerinhalte werden gelöscht (auf 0 gesetzt).
<b>CMPI</b>	<b>9nd</b>	<b>Vergleichsbefehl:</b> der Inhalt von Register <b>d</b> wird mit dem Wert <b>n</b> verglichen. Ist der Inhalt von Register d größer als n, wird das Carry-Flag gesetzt (LED leuchtet). Ist der Inhalt von Register d kleiner oder gleich groß als n, wird das Carry-Flag zurückgesetzt (LED leuchtet nicht). CMPI ist die Abkürzung für Compare immediate.
<b>BRC</b>	<b>Daa</b>	<b>Sprung-Befehl:</b> ( <b>branch if carry</b> ) der jedoch nur dann ausgeführt wird, wenn das Carry-Flag gesetzt wurde. Daher wird dieser Befehl auch „bedingter Sprung“ genannt. <b>aa</b> gibt in gleicher Weise (wie bei GOTO) die Sprung-Adresse an.

## „Programm-Ablaufplan“



# Der Programm-Ablaufplan – Das Flußdiagramm – Flow Chart

Drei Bezeichnungen für das gleiche Objekt – wir können uns aussuchen, was uns am besten gefällt.

Computer-Programme werden bei zunehmender Länge immer unübersichtlicher. Schon das letzte Programm ist nicht mehr ganz einfach zu übersehen. Damit auch sehr große Programme verständlich dargestellt werden können, wurden Sinnbilder entwickelt, die sich zu einem Programm-Ablaufplan zusammenfügen lassen.

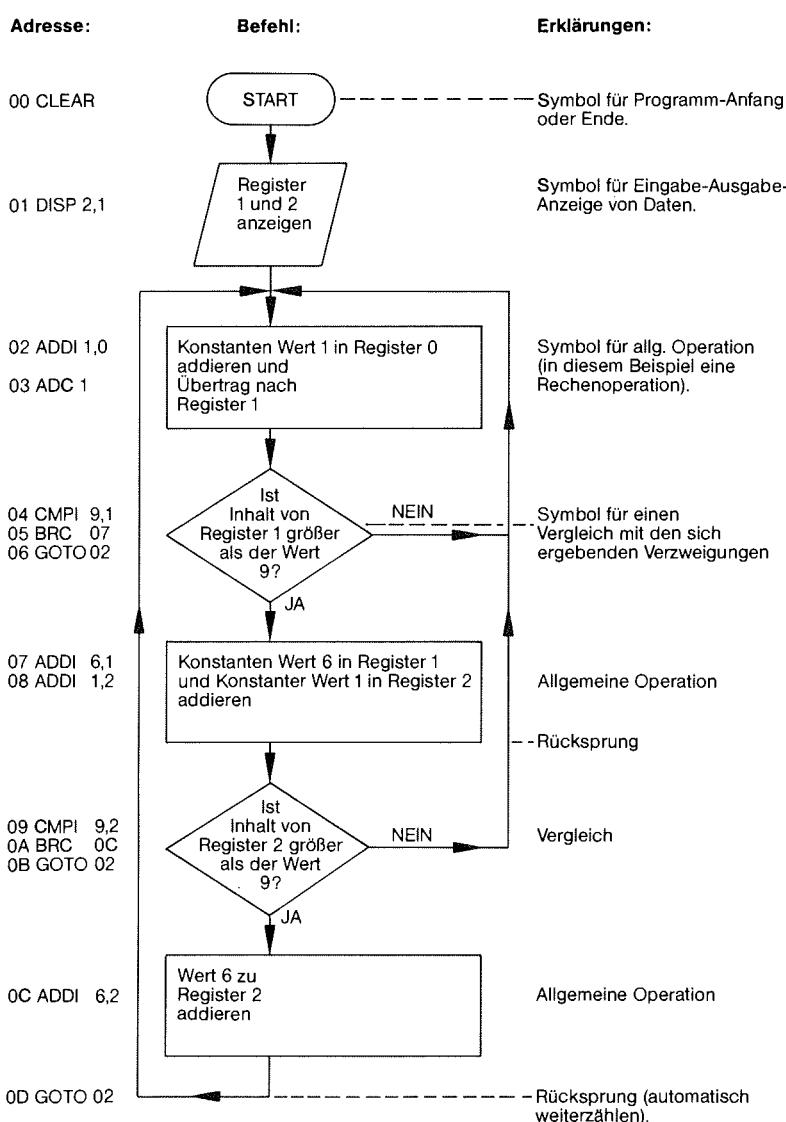
Nachstehend finden wir den Programm-Ablaufplan mit entsprechenden Erklärungen für den zuletzt programmierten Dezimal-Zähler.

Durch diesen Programm-Ablaufplan wird die Funktion des Programmes wesentlich übersichtlicher. Wir müssen jetzt lediglich den Sinnbildern von oben nach unten folgen und bei den Verzweigungen (bei den Programm-Sprüngen) den Linien in Pfeilrichtung nachgehen.

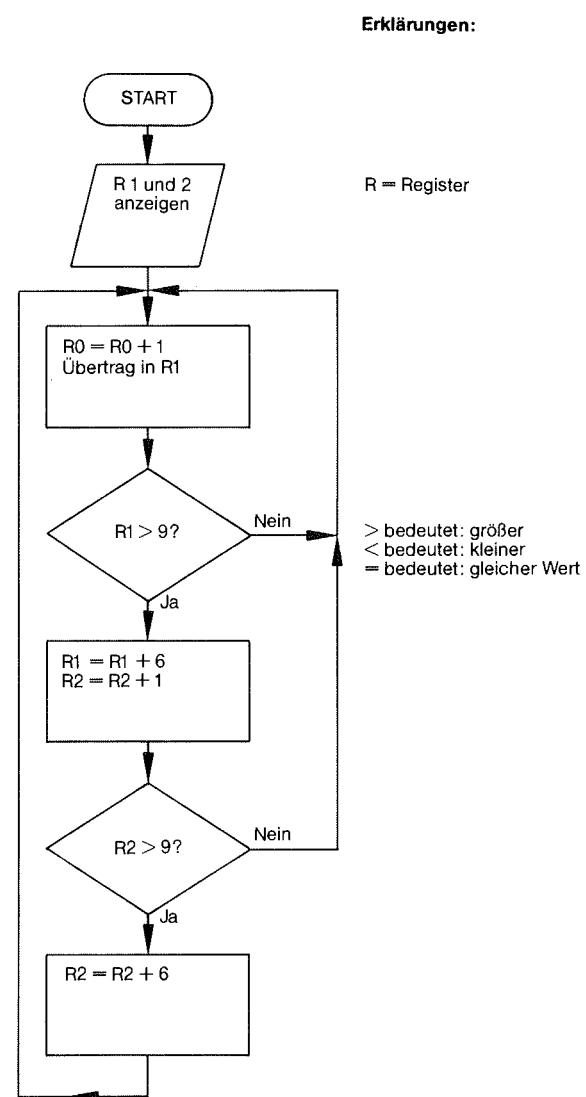
Dieser Programm-Ablaufplan kann nochmals vereinfacht dargestellt werden, ohne daß deshalb die Übersichtlichkeit beeinträchtigt wird. Wir müssen uns lediglich die Bedeutung der Symbole und einige Kurzbezeichnungen merken.

Dieses vereinfachte flow-chart oder Flußdiagramm oder Programm-Ablaufplan sieht dann folgendermaßen aus:

## Programm-Ablaufplan



## Programm-Ablaufplan, vereinfachte Darstellung



## Aufgabe für ein selbst zu entwickelndes Programm

Wir möchten erreichen, daß bei jedem Tastendruck einer beliebigen Zifferntaste der konstante Wert 1 dezimal addiert wird, wobei das Ergebnis auf allen 6 Stellen des Displays gleichzeitig angezeigt werden soll. Bei Programm-Beginn sollen alle Register gelöscht werden, damit sechsmal die Zahl 0 angezeigt wird. Bei Betätigung einer Zifferntaste wird der Wert 1 hinzugefügt, welcher dann auf allen 6 Display-Stellen angezeigt wird. Bei jedem weiteren Tastendruck erhöht sich das Ergebnis um den Wert 1. Sobald der Wert 9 erreicht wird, soll nicht hexadezimal mit A-B-C... , sondern es soll dezimal weitergezählt werden, also wieder mit 0 beginnend.

**Problem-Analyse:** Wir kennen alle notwendigen Befehle, um dieses Programm durchzuführen. Folgende Überlegungen sind notwendig:

1. Sämtliche Register löschen.
2. 6 Stellen auf dem Display anzeigen.
3. Auf Tastendruck (Zifferneingabe) warten.
4. Konstanten Wert 1 in Register 1 addieren.
5. Ist der Inhalt von Register 1 größer als 9? Wenn ja zum Programm-Anfang zurückspringen – wenn nein siehe 6.
6. Inhalt von Register 1 in die Register 2-6 schieben.
7. Rücksprung zu 2. (Ergebnis anzeigen).

Nachdem das Problem logisch aufgeteilt wurde, können wir das Programm in die entsprechenden Befehls-Codes umsetzen. Wir beachten, daß bei der Position „auf Tastendruck warten“ der KIN-Befehl notwendig ist, wobei der eingegebene Wert jedoch nicht weiter verarbeitet wird. Wir dürfen daher für KIN nicht die 6 Anzeige-Register verwenden. Aus der Problem-Analyse ersehen wir, daß die Register 1 bis 6 bereits belegt sind. Register 0 ist noch frei.

Wir schreiben das Programm in nachfolgende Tabelle:

Adresse	Eingabe-Befehl	Mnemonic
00		
01		
02		
03		
04		
05		
06		
07		
08		
09		
0A		
0B		

Wie üblich, nehmen wir das Programm mit HALT – NEXT – 00 – RUN in Betrieb. Auf dem Display muß jetzt sechsmal die Zahl 0 stehen und wenn wir eine Taste, z. B. „1“ betätigen, erscheint sechsmal die Ziffer 1. Bei jedem Tastendruck erhöht sich das Ergebnis um den Wert 1, so daß jeweils sechsmal die Ziffern 2, 3, 4, ... angezeigt werden.

Bringt der Computer nicht diese Ergebnisse, haben wir falsch programmiert. Also müssen wir überlegen, welche Befehle notwendig sind?

Hinweis: Wir benötigen die Befehle CLEAR – DISP – KIN – ADDI – CMPI – BRC – MOV – GOTO.

Falls das Programm immer noch nicht einwandfrei läuft, ergeben sich vielleicht noch Schwierigkeiten mit der Register-Beladung oder mit den Sprung-Befehlen. Die Programm-Lösung finden wir auf der folgenden Seite.

## Zweistelliger dezimaler Rechner

Nachdem wir inzwischen den Programm-Trick kennen, um unseren Computer zu überlisten seine Rechenoperationen zwar hexadezimal auszuführen, jedoch dezimal anzuseigen, wollen wir dies bei einem einfachen 2-stelligen Rechen-Programm ausprobieren.

Wir geben folgendes Programm ein:

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
00	<b>F08</b>	CLEAR	Alle Register löschen
01	<b>F10</b>	DISP 1,0	Eine Stelle (ab Register 0) anzeigen
02	<b>FF0</b>	KIN 0	Eingabe in Register 0
03	<b>FF1</b>	KIN 1	Eingabe in Register 1
04	<b>401</b>	ADD 0,1	Registerinhalt 0 und 1 addieren
05	<b>D09</b>	BRC 09	Wenn Ergebnis größer als F nach Adresse 09 springen
06	<b>991</b>	CMPI 9,1	Vergleichen ob Ergebnis größer als 9, dann
07	<b>D09</b>	BRC 09	Nach Adresse 09 springen
08	<b>C0B</b>	GOTO 0B	Sprung nach Adresse 0B
09	<b>561</b>	ADDI 6,1	In Register 1 konstanten Wert 6 addieren
0A	<b>512</b>	ADDI 1,2	In Register 2 konstanten Wert 1 addieren
0B	<b>F21</b>	DISP 2,1	2 Stellen ab Register 1 anzeigen
0C	<b>FF0</b>	KIN 0	Eingabe-Befehl als Warte-Befehl
0D	<b>C00</b>	GOTO 00	Rücksprung-Befehl zu Adresse 00

Programm-Start mit HALT – NEXT – 00 – RUN.

Das Display zeigt auf einer Anzeigestelle die Ziffer 0. Wenn wir jetzt z. B. „5“ eingeben, erscheint diese Zahl im Display. Wenn wir 7 hinzuaddieren (Tastendruck „7“) erscheint sofort das 2-stellige dezimale Ergebnis „12“. Wird erneut eine beliebige Zifferntaste betätigt, erscheint auf dem Display wieder die 0. Es können zwei andere Zahlen addiert werden.

Betrachten wir uns nochmals die Erklärungen zu den einzelnen Befehlen in der vorstehenden Tabelle und gehen jeden einzelnen Programm-Schritt mit dem Additions-Beispiel:  $5 + 7 = 12$  durch:

Bei Adresse 00 sind alle Register gelöscht.

Bei Adresse 01 wird eine Stelle angezeigt. Da alle Register gelöscht sind, zeigt das Display eine 0.

Bei Adresse 02 geben wir als Beispiel den Wert „5“ ein, welcher im Register 0 gespeichert und auf dem Display angezeigt wird.

Bei Adresse 03 geben wir als Beispiel die „7“ ein, welche in Register 1 gespeichert, jedoch nicht angezeigt wird, weil wir bei Adresse 02 befohlen haben, daß nur eine Stelle nämlich das Register 0 angezeigt werden soll.

Bei Adresse 04 wird der Register-Inhalt Nr. 0 (Wert „5“) und der Register-Inhalt Nr. 1 (Wert „7“) addiert.

Bei Adresse 05 prüfen, ob der sich ergebende Wert größer als das hexadezimale F ist. Da in unserem Beispiel das addierte Ergebnis ( $5 + 7 = 12$ ) ein hexadezimales C ergibt, wird bei

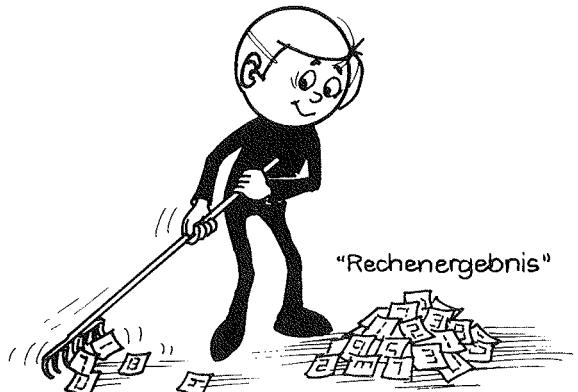
Adresse 06 verglichen, ob das Ergebnis größer als 9 ist. Nachdem dies zutrifft, erfolgt der Sprung zur Adresse 09.

Bei Adresse 09 wird in Register Nr. 1 (mit dem hexadezimalen Ergebnis C) der konstante Wert 6 dazuaddiert. Da hexadezimal C + 6 den hexadezimalen Wert 12 ergibt (vergleiche mit Tabelle „Dezimales/hexadezimales Zahlen-System“), im Register 1 jedoch nur ein 1-stelliger Wert gespeichert werden kann, verbleibt in Register 1 nur die letzte Stelle des 2-stelligen Ergebnisses 12, nämlich die 2. Bei der Addition von C + 6 wurde der hexadezimale Wert F übersprungen, wodurch kurz das Carry-Flag aufleuchtet.

Bei Adresse 0A wurde das Register 2 angesprochen, in welchem bisher (durch den CLEAR-Befehl bei Adresse 00) der Wert 0 gespeichert war. Da jedoch bei Adresse 0A durch den Befehl ADDI 1,2 der konstante Wert 1 im Register 2 hinzugefügt wird, steht im Register 2 nunmehr der Wert 1.

Bei Adresse 0B werden durch den DISP-Befehl zwei Stellen ab Register 1 angezeigt. Im Register 2 ist der Wert 1 gespeichert, im Register 1 ist der Wert 2 übriggeblieben – das Display zeigt das Ergebnis 12.

Hätten wir bei Adresse 0C nicht den KIN-Befehl, könnten wir das Ergebnis überhaupt nicht sehen, weil der unter Adresse 0D folgende Rücksprung-Befehl zum Programm-Anfang alle Register (und damit auch das Ergebnis) löschen würde. Bei Adresse 0C wirkt der KIN-Befehl als Warte-Befehl, nämlich solange, bis wir durch Betätigung einer Ziffern-Taste eine neue Eingabe machen. Erst jetzt erfolgt durch Adresse 0D der Rücksprung zum Programm-Anfang – alle Register werden gelöscht und eine Stelle, nämlich die Ziffer 0 wird angezeigt – das Spiel kann von vorne beginnen.



### Lösung von Seite ...

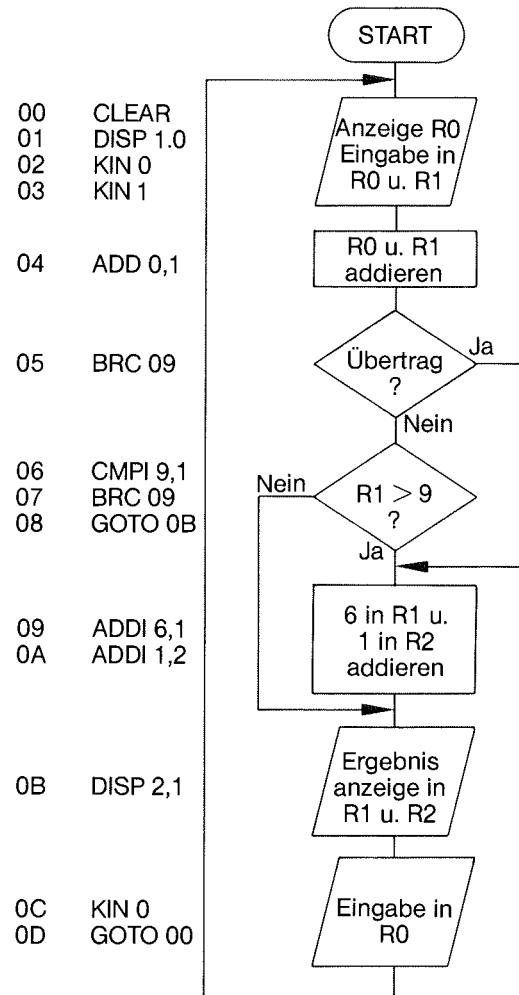
Adresse	Eingabe-Befehl	Mnemonic
00	<b>F08</b>	CLEAR
01	<b>F61</b>	DISP 6,1
02	<b>FF0</b>	KIN 0
03	<b>511</b>	ADDI 1,1
04	<b>991</b>	CMPI 9,1
05	<b>D00</b>	BRC 00
06	<b>012</b>	MOV 1,2
07	<b>013</b>	MOV 1,3
08	<b>014</b>	MOV 1,4
09	<b>015</b>	MOV 1,5
0A	<b>016</b>	MOV 1,6
0B	<b>C01</b>	GOTO 01

Damit wir nun endgültig begreifen, was in den Registern geschieht, betrachten wir uns zu dem Beispiel der Addition  $5 + 7 = 12$  die nachfolgende Darstellung:

Da wir jetzt wissen, was sich in den Registern abspielt, sollte für das ebenfalls noch dargestellte Fluß-Diagramm zu unserem 2-stelligen dezimalen Rechner keine besonderen Erklärungen notwendig sein.

Adr. Nr.	Eingabe-Befehl	Was passiert in den Registern			Erklärungen
		Nr. 2	Nr. 1	Nr. 0	
00	CLEAR	0	0	0	Alle Register löschen = Inhalt Wert 0
01	DISP 1,0	0	0	5	Eine Stelle – Register 0 (Wert 0) anzeigen
02	KIN 0	0	0	5	Eingabe und Anzeigen Wert 5 in Register 0
03	KIN 1	0	7	5	Eingabe (ohne Anzeige) Wert 7 in Register 1
04	ADD 0,1	0	7 +	5	Inhalt von Register 0 und 1 addieren
		0	= C	5	ergibt in Register 1 hexadezimales „C“
05	BRC 09	0	C	5	Da Ergebnis „C“ nicht größer als „F“ weiter nach 06
06	CMPI 9,1	0	C	5	Vergleichen ob Ergebnis „C“ größer als „9“ – ja / nein?
07	BRC 09	0	C	5	Weil „ja“ nach Adresse 09 springen
08	GOTO 0B	0	C	5	Wird übersprungen
09	ADDI 6,1	0	+ 6	5	konstanten Wert „6“ zu „C“ in Register 1 addieren
		0	(1) 2	5	Zwischenergebnis, Carry-Flag durch Übertrag
0A	ADDI 1,2	1	2	5	in Register 2 wird der konstante Wert 1 (zu bisher 0) addiert
0B	DISP 2,1	1	2	5	2 Stellen ab Register 1 (also Register 1 und 2) anzeigen = Ergebnis = 12
0C	KIN 0	1	2	5	Warten – Ergebnis bleibt sichtbar, bis neue Eingabe erfolgt
0D	GOTO 00	1	2	5	Rücksprung zu Adresse 00
00	CLEAR	0	0	0	Siehe Programm-Anfang

Die Zahlen auf schwarzem Grund werden auf dem Display angezeigt. Wir wundern uns vielleicht, daß beim Register 0 der eingegebene Wert 5 sehr lange in dieser Tabelle als angezeigter Wert dargestellt wird. In Wirklichkeit läuft das ganze Spiel zwischen Adresse 04 und 0C im Bruchteil einer Sekunde ab.



## Wir löschen den gesamten Programm-Speicher!

Wir wissen, daß die Programm-Speicher und Register unseres Computers alle Informationen verlieren, sobald die Stromzufuhr unterbrochen, d. h. wenn das Netzgerät aus der Steckdose entfernt wird. Außer den festinstallierten Programmen (Prüfprogramm, Nimm-Spiel, Uhren-Programm usw.) kann unser Computer ohne Stromzufuhr die eingegebenen Daten nicht behalten.

Wird das Gerät wieder eingeschaltet, stehen wahllos und zufällig sich ergebende Daten in den Programm-Speichern und Registern.

Falls wir unseren Computer nach der letzten Programm-Eingabe des 2-stelligen dezimalen Addierers noch nicht abgeschaltet haben, ist unter den Adressen 00 bis 0D dieses Programm auch weiterhin gespeichert. Ziehen wir jetzt einmal das Netzgerät aus der Steckdose. Bitte nicht sofort wieder einstecken – sondern ca. 10 Sekunden warten. Wenn wir das Gerät jetzt wieder in Betrieb nehmen, zeigt das Display wie bekannt 00 000. Wir gehen mit HALT – NEXT – 00 an den Programm-Anfang und sehen, daß unter der Adresse 00 beliebige Zahlen oder Buchstaben angezeigt werden. Betätigen wir jetzt noch RUN wird der Computer vermutlich mit irgendeiner Tätigkeit beginnen, ohne zu einem Ergebnis zu kommen. Er wird durch ein zufällig vorhandenes Programm gesteuert, was entweder dazu führt, daß auf dem Display immer wieder sich wiederholende Informationen erscheinen und die LEDs links neben dem Display blinken. Es kann auch passieren, daß sich das Programm „tot läuft“, d. h., daß der Computer durch ein unsinniges Programm den Betrieb einstellt.

Wir können mehrmals das Netzgerät aus der Steckdose ziehen – einige Sekunden warten – neu einschalten – mit HALT – NEXT – 00 – RUN wird sich immer wieder ein anderer Programm-Lauf ergeben.

Brechen wir doch ein derartiges, durch Einschalten zufällig vorhandenes Programm ab mit HALT – NEXT – 00. Durch laufendes Betätigen der NEXT-Taste stellen wir fest, daß tatsächlich auf sämtlichen Programm-Adressen zufällige Befehle vorhanden sind. Wenn wir bei Adresse 00 beginnen ein neues Programm einzugeben, ist dies nicht störend, weil durch Neueingabe von Befehlen die zufällig im Programm-Speicher stehenden Befehle gelöscht werden.

Wir können jedoch auch den gesamten Programm-Speicher löschen durch Betätigung der Tasten **HALT – PGM – 5**. Das Display wird für einen kurzen Moment abgeschaltet und zeigt dann 00 000. Bei HALT – NEXT – 00 ist auf Adresse 00 kein Befehl mehr vorhanden. Nun betätigen wir die NEXT-Taste und sehen, daß auch unter der Adresse 01 alle Eingaben auf 0 stehen. Wir können mit NEXT den gesamten Programm-Speicher „durchblättern“ – im Programm-Speicher sind keine Befehle vorhanden.

Es ist sinnvoll, wenn wir vor einer neuen Programm-Eingabe unseren Programm-Speicher durch HALT – PGM – 5 säubern.

## 6-stellige Addition – wie beim Taschenrechner!

Wir wollen nun eine 6-stellige Addition programmieren, damit unser Computer ähnlich wie ein Taschenrechner mehrstellige Zahlen addiert und die richtigen Ergebnisse anzeigt.

Zuvor sollten wir uns einige Gedanken über diese Addition machen, weil sich die Funktionsabläufe sowohl bei Taschenrechnern als auch bei Groß-Computern in gleicher Weise ergeben.

Aus der nachfolgenden Tabelle ergibt sich ein größerer Aufwand. Ein recht umfangreiches Programm ist notwendig. Vielleicht fragen wir uns, weshalb für ein derartiges Additions-Programm so viele Programm-Schritte einzugeben sind, während wir bei einem preiswerten Taschenrechner sofort mit der Eingabe von Rechenaufgaben beginnen können?

In einem Taschenrechner ist ebenfalls ein kleiner Mikroprozessor enthalten, welchem durch ein fest installiertes Programm alle für die Rechenoperationen notwendigen Programm-Schritte eingegeben wurden. Auch unser Computer hat fest installierte Programme wie z. B. Uhren-Programm, Nimm-Spiel usw. Ein preiswerter Taschenrechner kann sofort nach Inbetriebnahme alle Rechenoperationen ausführen. Außer rechnen kann er jedoch nichts! Es sei denn, daß wir beim Kauf etwas mehr angelegt haben, weshalb vielleicht noch eine Uhr integriert ist. Dann kann er rechnen – und die Uhrzeit anzeigen – mehr kann er nicht!

Gegenüber dem Taschenrechner ist unser Computer frei programmierbar. Zusätzlich hat er fest installierte Programme (Uhr, Nimm-Spiel usw.). Wir können ihn durch Befehls-Eingabe dazu bringen, daß er uns die gesamten Geheimnisse der Computer-Technik preisgibt und daß er außerdem noch für tausendfache Verwendungsmöglichkeiten einsetzbar wird. Wenn wir im 2. Teil unseres Anleitungsbuches blättern, erkennen wir einen Teil seiner Möglichkeiten. Weil wir uns jedoch intensiv mit der Materie beschäftigen, erlernen wir Schritt für Schritt seine Programmierung und wir werden am Ende des Anleitungsbuches feststellen, daß wir uns erst am Anfang der sich ergebenden Möglichkeiten unseres Computers befinden.

Zurück zur 6-stelligen Addition. Wir wissen inzwischen, daß alle Computer grundsätzlich immer nur zwei Register, also zwei einzelne Zahlen, auf einmal addieren können. Bei einer Addition von zwei 6-stelligen Zahlen müssen  $2 \times 6$ , also insgesamt 12 Register miteinander addiert werden. Hierfür benutzt man einen einfachen Rechentrick, den wir schon in der Schule kennengelernt haben.

Beispiel:

$$\begin{array}{r} + 5. 7 3 6 \\ + 5. 8 9 . 4 \\ \hline = 6. 6 3 0 \end{array}$$

Wir schreiben die zu addierenden Zahlen untereinander und beginnen die Addition bei den letzten (also rechts stehenden) Stellen. Ergibt sich ein Übertrag (wenn das Additionsergebnis der beiden Stellen größer als 9 ist) wird dieser Übertrag eine Stelle weiter links festgehalten, (siehe oben): Die beiden letzten Stellen  $6 + 4$  addiert ergeben 10 – wir schreiben als Ergebnis 0 und die übrigbleibende 1 schieben wir eine Stelle weiter nach links. Addieren wir die nächsten Stellen  $3 + 9 = 12$  (+ die übriggebliebene 1) = 13. Als Ergebnis schreiben wir 3 und schieben die übrigbleibende 1 eine Stelle weiter usw.

Unser Computer-Programm arbeitet nach der gleichen Methode: Es werden zwei Register addiert – der Übertrag wird beim nächsten Register dazuaddiert usw. Wie das im einzelnen funktioniert und durch verschiedene Programm-Schritte dargestellt werden kann, haben wir beim letzten Programm bereits kennengelernt.

Frisch an's Werk – wir geben das nachfolgende Programm gemäß Tabelle ein. Wir sollten allerdings aufpassen, daß uns keine „Zahlen-Dreher“ oder sonstigen falschen Angaben unterlaufen, weil dann der Rechner nicht einwandfrei funktioniert oder falsche Ergebnisse errechnet. Wir erinnern uns, daß wir mit der Taste C/CE falsche Eingaben löschen und durch Neueingabe berichtigen können. Ist uns z. B. bei Adresse 20 eine falsche Eingabe unterlaufen, die wir jedoch erst bemerkt haben, nachdem wir bereits die NEXT-Taste betätigten, können wir sofort korrigieren. Durch NEXT zeigt uns das Display zwar schon die Adresse 21 an – wir gehen jedoch einfach mit HALT – NEXT – 20 zu der zu korrigierenden Adresse zurück – geben den richtigen Befehl ein und programmieren nach NEXT weiter.

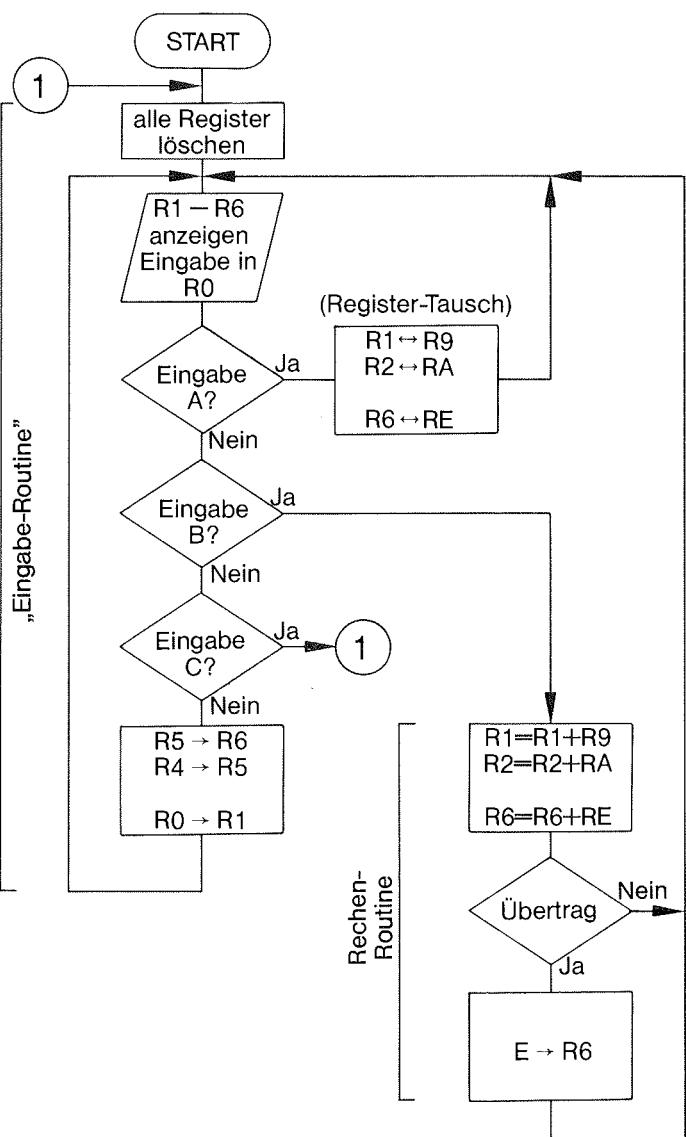
## Programm: 6-stellige Addition

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
00	<b>F08</b>	CLEAR	alle Register löschen
01	<b>F61</b>	DISP 6,1	Reg. 1 bis Reg. 6 anzeigen
02	<b>FF0</b>	KIN 0	Eingabe in Reg. 0
03	<b>9A0</b>	CMPI A,0	eingegebener Wert A?
04	<b>E10</b>	BRZ 10	dann Sprung zu Zahlen umspeichern
05	<b>9B0</b>	CMPI B,0	eingegebener Wert B?
06	<b>E12</b>	BRZ 12	dann Sprung zu Addition
07	<b>9C0</b>	CMPI C,0	eingegebener Wert C?
08	<b>E00</b>	BRZ 00	dann Sprung zu „alle Register löschen“
09	<b>056</b>	MOV 5,6	alle Register um
0A	<b>045</b>	MOV 4,5	eine Stelle
0B	<b>034</b>	MOV 3,4	nach links
0C	<b>023</b>	MOV 2,3	schieben
0D	<b>012</b>	MOV 1,2	
0E	<b>001</b>	MOV 0,1	
0F	<b>C02</b>	GOTO 02	Rücksprung zu Eingabe
10	<b>F0F</b>	EXRA	Registertausch
11	<b>C02</b>	GOTO 02	
12	<b>491</b>	ADD 9,1	1. Stelle von rechts addieren
13	<b>D17</b>	BRC 17	
14	<b>991</b>	CMPI 9,1	
15	<b>D17</b>	BRC 17	
16	<b>C19</b>	GOTO 19	
17	<b>561</b>	Addi 6,1	
18	<b>512</b>	Addi 1,2	
19	<b>4A2</b>	ADD A,2	2. Stelle von rechts addieren
1A	<b>D1E</b>	BRC 1E	
1B	<b>992</b>	CMPI 9,2	
1C	<b>D1E</b>	BRC 1E	
1D	<b>C20</b>	GOTO 20	
1E	<b>562</b>	ADDI 6,2	
1F	<b>513</b>	ADDI 1,3	
20	<b>4B3</b>	ADD B,3	3. Stelle von rechts addieren
21	<b>D25</b>	BRC 25	
22	<b>993</b>	CMPI 9,3	
23	<b>D25</b>	BRC 25	
24	<b>C27</b>	GOTO 27	
25	<b>563</b>	Addi 6,3	
26	<b>514</b>	Addi 1,4	
27	<b>4C4</b>	ADD C,4	4. Stelle von rechts addieren
28	<b>D2C</b>	BRC 2C	
29	<b>994</b>	CMPI 9,4	
2A	<b>D2C</b>	BRC 2C	
2B	<b>C2E</b>	GOTO 2E	
2C	<b>564</b>	Addi 6,4	
2D	<b>515</b>	Addi 1,5	
2E	<b>4D5</b>	Add D,5	5. Stelle von rechts addieren
2F	<b>D33</b>	BRC 33	
30	<b>995</b>	CMPI 9,5	
31	<b>D33</b>	BRC 33	
32	<b>C35</b>	GOTO 35	
33	<b>565</b>	ADDI 6,5	
34	<b>516</b>	ADDI 1,6	

35	<b>4E6</b>	ADD E,6	6. Stelle von rechts addieren
36	<b>D3A</b>	BRC 3A	
37	<b>996</b>	CMPI 9,6	
38	<b>D3A</b>	BRC 3A	
39	<b>C01</b>	GOTO 01	
3A	<b>1E6</b>	MOVI E,6	
3B	<b>C01</b>	GOTO 01	

Programm-Eingabe beendet: HALT – NEXT – 00 – RUN. Das Display zeigt 000000. Mit den Zahlen-Tasten geben wir beispielsweise ein: 5 7 3 6. Taste **A** betätigen (die wir als  $\oplus$ -Taste verwenden). Jetzt die zweite zu addierende Zahl eingeben, z. B. 8 9 4, dann die Taste **B** betätigen (die wir als  $\ominus$ -Taste verwenden), der Computer zeigt das Ergebnis: 6630. Mit der Taste **C** werden alle Register und damit auch das Ergebnis gelöscht – eine neue Rechnung kann eingegeben werden. Wird das Ergebnis einer Addition zu groß, d. h., daß die vorhandenen sechs Anzeigestellen nicht mehr ausreichen, wird durch ein E der sogenannte „Überlauf“, also ein nicht korrektes Ergebnis kenntlich gemacht. Unser Computer arbeitet bei diesem Programm-Beispiel noch etwas langsam, wir sollten also die zu addierenden Zahlen nicht zu schnell hintereinander eintippen.

Arbeitet der Computer nicht einwandfrei, gehen wir mit HALT – NEXT – 00 auf den Programm-Anfang und überprüfen durch immer wieder erneute Betätigung der NEXT-Taste, ob alle Programm-Schritte richtig eingegeben worden sind.



## Nach welchen Kriterien arbeitet unser Programm?

Um länger Programme zu erklären, ist es zweckmäßig, daß Gesamt-Programm in mehrere kleine Einzel-Programme aufzuteilen.

Bei unserer 6-stelligen Addition stellen die Programm-Schritte der Adressen 00 bis 11 ein Unterprogramm dar – wir bezeichnen es als „**Eingabe-Routine**“. Das eigentliche Rechen-Programm ist in den Befehlen der Adressen 12 bis 3B enthalten. Bei den nun folgenden Erklärungen sollten wir zweckmäßigerweise die Programm-Tabelle (mit den einzelnen Befehlen) und den Programm-Ablaufplan (Fluß-Diagramm) vergleichen.

Bei der Betrachtung des Programm-Ablaufplanes fallen uns zunächst die eingekreisten Zahlen ① auf. Diese Kreise machen den Programm-Ablaufplan übersichtlicher. Bei „**START**“ sehen wir die eingekreiste Ziffer ① mit einem Pfeil und im Programm-Ablaufplan weiter unten sehen wir einen Pfeil, der auf die eingekreiste Ziffer ① zeigt. Hierdurch werden sich überkreuzende Linien beim Programm-Ablaufplan vermieden, denn die weiter unten stehende ① bedeutet lediglich, daß sich an dieser Programm-Stelle ein Sprung zu der am Programm-Anfang stehenden ① ergibt. Außerdem ist es beachtenswert, daß zur Abkürzung des Wortes Register lediglich ein **R** geschrieben wird. (R1 Inhalt des Registers Nr. 1). Die kleinen Pfeile (→) bedeuten, daß Zahlen bzw. Daten, von einem Register in ein anderes Register zu bewegen sind. R5 → R6 bedeutet, daß der Inhalt vom Register Nr. 5 im Register Nr. 6 gespeichert wird.

Nun wollen wir das Programm untersuchen, zunächst die bei Adresse 00 beginnende „**Eingabe-Routine**“. Durch den CLEAR-Befehl werden alle Register gelöscht. Der Befehl DISP 6,1 bewirkt, daß die Register Nr. 1 bis 6 auf dem Display angezeigt werden.

Mit KIN 0 kann eine (zu addierende) Zahl in das Register 0 eingegeben werden. Wenn wir weder die Tasten A, B oder C betätigen (die wir als  $\oplus$ -Taste,  $\ominus$ -Taste oder Lösch-Taste verwenden) werden die Befehle bei den Adressen 03 bis einschl. 08 übersprungen, d. h., der eingegebene Zahlenwert erreicht die MOV-Befehle bei den Adressen 09 bis 0E. Der im Register 0 zunächst eingegebene Zahlenwert wird also z. B. bei Adresse 0E von Register 0 in Register Nr. 1 geschoben. Da wir sechs MOV-Befehle haben, können wir bis zu maximal sechs (zu addierende) Zahlen eingegeben, welche durch die MOV-Befehle nacheinander von Register 0 in Register 1 von dort nach Register 2 usw. weitergeschoben werden. Haben wir z. B. drei verschiedene Zahlen eingegeben, wandern diese von Register 0 in die Register Nr. 1, 2 und 3. Sie werden durch den DISP-Befehl angezeigt. Betätigen wir nun die Taste A, steht als letzte Eingabe der Wert A in Register 0. Bei Adresse 03 haben wir den Befehl CMPI A,0. Dieser Vergleichs-Befehl registriert, ob im Register Nr. 0 der Wert A vorhanden ist. Wir erinnern uns, daß wir mit CMPI einen Übertrag veranlassen können, wodurch das Carry-Signal ausgelöst wird. Der CMPI hat eine weitere Eigenschaft: Er kann auch vergleichen, ob zwei gleiche Werte vorhanden sind. CMPI A,0 sagt: vergleiche, ob in Register Nr. 0 der Wert A vorhanden ist. Nachdem wir die Taste A betätigt haben, steht der Wert A in Register 0, was vom Vergleichs-Befehl

registriert wird. Hierdurch wird das **Zero-Flag** ausgelöst, was die aufleuchtende untere LED links neben dem Display bestätigt. Wird also bei Adresse 03 festgestellt, daß im Register 0 der Wert A vorhanden ist, wird durch das nun gesetzte Zero-Flag der bei Adresse 04 vorhandene neue Befehl **BRZ10** angesprochen. Der Befehl BRZ (aus dem englischen „branch if zero“) ist ein Sprung-Befehl, welcher durch das Zero-Flag ausgelöst wird. BRZ 10 bedeutet Sprung zur Adresse 10. Hier erkennen wir den ebenfalls neuen Befehl EXRA.

Der Befehl **EXRA** bewirkt einen „**Register-Tausch**“. Nehmen wir an, daß wir vor Betätigung der Taste A drei zu addierende Zahlen eingegeben haben, z. B. die Ziffern 3 – 5 – 2. Durch die MOV-Befehle stehen diese drei Zahlen wie folgt in den Registern:

- 2 in Register Nr. 1
- 5 in Register Nr. 2
- 3 in Register Nr. 3

Auf dem Display steht 352. Durch den EXRA-Befehl haben wir den Register-Tausch veranlaßt, wodurch der Inhalt von Register 1 in das Register Nr. 9 wandert, der Inhalt von Register 2 in Register Nr. A, der Inhalt von Register Nr. 3 in das Register Nr. B. Der Register-Tausch bewirkt noch eine zweite Besonderheit. Bevor die Zahlenwerte 253 in die Register Nr. 9, A und B gebracht wurden, war in diesen Registern der Wert 0 vorhanden (weil durch den bei Adresse 00 vorhandenen CLEAR-Befehl beim Programm-Start alle Register-Werte auf 0 gesetzt wurden). Der Register-Tausch bewirkt nicht nur, daß die in den Registern Nr. 1, 2 und 3 vorhandenen Zahlenwerte „253“ nach den Registern Nr. 9, A und B gebracht werden – die in diesen drei Registern vorhandenen Werte 0 werden gleichzeitig in die Register Nr. 1, 2 und 3 gebracht (getauscht). Ganz einfach: Die Werte, die wir in die Register 1, 2 und 3 eingegeben haben, stehen jetzt in den Registern 9, A und B und die in diesen Registern vorhandenen Werte 0 stehen jetzt in den Registern 1, 2 und 3.

Bei Adresse 11 folgt jetzt der Befehl GOTO 02 – also ein Rückprung zum Eingabe-Befehl bei Adresse 02.

Wir können jetzt weitere Zahlen für die Addition eingeben, die wiederum durch die MOV-Befehle auf dem Display angezeigt werden. Da eine Addition durchgeführt werden soll, werden wir nun die Taste **B** betätigen. Dies wird bei Adresse 05 durch den Befehl CMPI B,0 registriert. Durch das ausgelöste ZERO-Signal wird die Adresse 06 mit dem Sprung-Befehl BRZ 12 angesprochen. Somit wird das Programm bei Adresse 12 fortgesetzt. Hier beginnt das eigentliche Additions-Programm, welches wir als „**Rechen-Routine**“ bezeichnen. Bei Adresse 12 erkennen wir den Befehl ADD 9,1, d. h. der Inhalt von Register 9 wird mit dem Inhalt von Register 1 addiert. Prinzipiell läuft das gleiche Additions-Programm ab, wie wir dies bei unserem 2-stelligen Addierer vorher bereits kennengelernt haben. Durch die Programm-Erweiterung können wir jetzt nicht nur zwei, sondern insgesamt sechs Stellen addieren.

Wir haben noch eine besondere Funktion in unserem Programm eingebaut: Wurde die sechste Stelle bei der Adresse Nr. 36 addiert, wird abgefragt, ob sich bei dieser sechsten Stelle ein Übertrag ergibt. Würde sich ein solcher Übertrag ergeben, wäre eine siebte zusätzliche Anzeigestelle erforderlich. Da dies nicht möglich ist, bewirkt das durch einen Übertrag ausgelöste Carry-Signal bei der Adresse Nr. 3A, daß in das Register 6 durch den **MOVI**-Befehl ein E geschoben wird. Dieses E wird auf der sechsten Stelle (im Display ganz links) angezeigt, wodurch der „Überlauf“ erkannt wird.

Wenn wir das angezeigte Ergebnis löschen möchten, wird die Taste **C** betätigt. Dies wird unter Adresse 07 vom Befehl CMPI C,0 erkannt, wodurch sich bei Adresse 08 mit dem Befehl BRZ 00 ein Sprung zum Programm-Anfang Adresse 00 ergibt. Durch den CLEAR-Befehl werden alle Register gelöscht – eine neue Eingabe ist möglich.

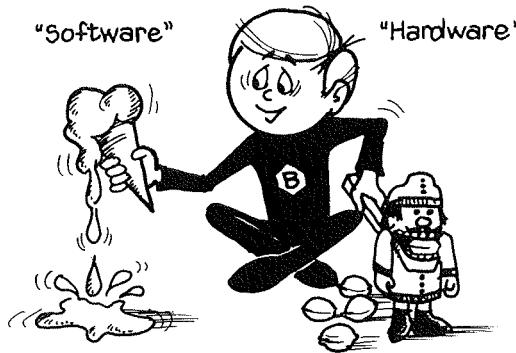
Alles, was wir in den vorstehenden Abschnitten gelesen haben, wird vom Computer innerhalb eines kurzen Augenblickes erledigt.

## Beschreibung der Microtronic-Befehlsmöglichkeiten.

Der gesamte Microtronic-Befehls-Satz ist ab Seite 73 mit allen wissenswerten Details beschrieben. Eine Kurzbeschreibung der Befehle finden wir auf dem beigegebenen Buch-Lesezeichen.



## Software und Hardware.



Nachdem wir im letzten Kapitel mit dem 6-stelligen Additions-Programm den schwierigsten Teil unserer Einführung in die Computer-Technik überwunden haben, wollen wir uns zur Abwechslung einmal mit unproblematischeren Dingen befassen.

Bis jetzt hatten wir in erster Linie „Software-Probleme“ zu bewältigen. Also sollten wir uns jetzt einmal mit der „Hardware“ unseres Computers befassen.

Schon wieder Computer chinesisch? Das englische „software“ könnte man mit „Weich-Material“ übersetzen. Unter Software versteht man alles, was man bei einem Computer nicht sehen und nicht greifen kann: Die einzugebenden Daten und Programme und auch das im Mikroprozessor enthaltene „Monitor-Programm“, durch welches unser Computer funktionsfähig wird.

Unter Hardware verstehen wir die greif- und sichtbaren Teile des Computers: Die Tastatur, das Display, die Platine mit den Chips usw.

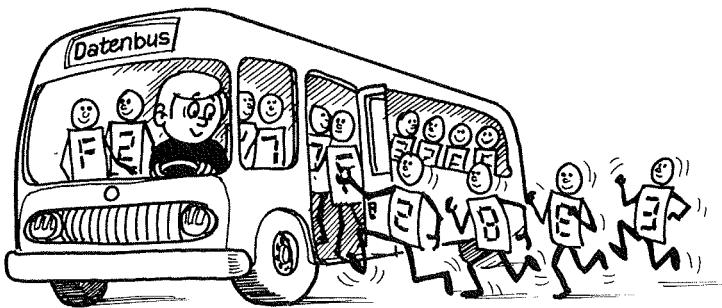
Mittelpunkt und Gehirn unseres Computers ist der größere schwarze Chip mit seinen 40 Anschlußbeinchen: Der Mikroprozessor.

Mit der Erfindung des Mikroprozessors wurde der höchste technische Entwicklungsstand eingeleitet, den die Menschheit bisher erreicht hat. So unglaublich es klingen mag – der Mikroprozessor wird unser Weltbild verändern. Seine Einsatzmöglichkeiten sind unübersehbar und durch den Mikroprozessor werden zukünftig Dinge möglich sein, die noch vor wenigen Jahren von den meisten Menschen als utopisch angesehen wurden. Unsere intensive Beschäftigung mit dem Mikrocomputer gibt uns die einmalige Chance, die technische Weiterentwicklung der nächsten Jahre besser zu verstehen.

Der Mikroprozessor alleine ist noch kein Computer. Um einen „Zugriff“ zum Mikroprozessor zu haben, benötigen wir die Tastatur und die Daten-Eingangsleitungen. Damit sich der Mikroprozessor „mitteilen“ kann, benötigt er ein Display (in unserem Fall die Leuchtanzeige – bei größeren Geräten einen Bildschirm), die Daten-Ausgangsleitungen und eine „Drum-Herum-Elektronik“, die als Peripherie-Elektronik bezeichnet wird.

Der Mikroprozessor als Zentraleinheit des Computers wird als **CPU** bezeichnete (sprich: „ZEH-PEH-UH“ oder englisch ausgesprochen „si-pi-juh“). Dies ist die Abkürzung für **central processing unit**. Die CPU verwaltet die Ein- und Ausgabe der Daten und den Zugriff zu den Speichereinheiten. Außerdem ist die CPU die eigentliche Recheneinheit des Computers.

## Bus-Verbindungen für den Datentransport?

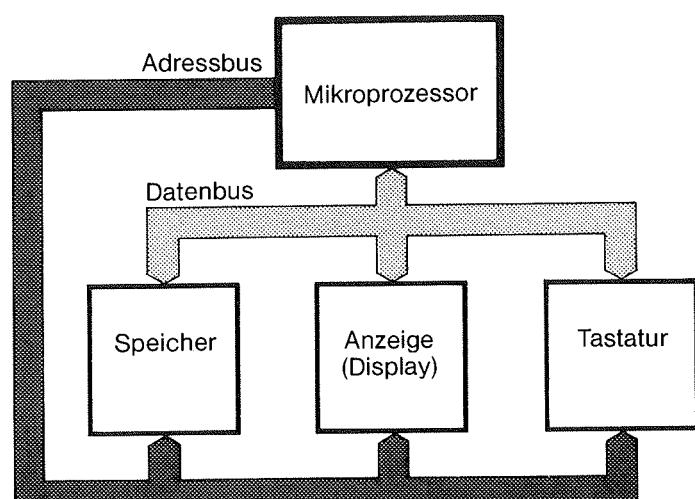


Kaum vorstellbar, doch zwischen den Einheiten des Computers besteht eine Bus-Verbindung für den Datentransport. Mit Bus werden die Verbindungsleitungen (Leiterbahnen auf der Platine usw.) bezeichnet. Wie wir gleich sehen werden, benötigt ein Computer ungewöhnlich viele Verbindungsleitungen, weil der Mikroprozessor gleichzeitig (oder besser gesagt nacheinander im Bruchteil einer tausendstel Sekunde) die verschiedensten Aufgaben bearbeitet.

Wir wissen, daß Computer nur mit den beiden Zuständen „0“ (keine Spannung) und „1“ (Spannung vorhanden) arbeiten. Die Verbindungsleitungen sind für die Übertragung dieser Signale notwendig. Wir erinnern uns, daß wir zur Darstellung einer dezimalen Zahl (durch das hierfür erforderliche Dual-System) bereits 4 Verbindungsleitungen benötigen. Nun muß der Mikroprozessor jedoch dem Speicher-Baustein mitteilen können, aus welcher Speicher-Adresse er seine Befehle haben möchte. Nachdem jede Programm-Adresse aus 2 Ziffern besteht, sind auch hierfür  $2 \times 4 = 8$  Verbindungsleitungen notwendig. Der eigentliche Befehls-Code besteht aus 3 hexadezimalen Ziffern, d. h., daß weitere 12 Verbindungsleitungen notwendig sind, so daß wir bereits jetzt 20 Leitungen benötigen würden.

Der Mikroprozessor muß jedes einzelne Leuchtsegment des Displays ansteuern können. Jede auf dem Display dargestellte Stelle besteht aus 7 Leuchtsegmenten und nachdem wir auf dem Display 6 Stellen (mit jeweils 7 Leuchtsegmenten) haben, wären hierfür weitere 42 Verbindungsleitungen notwendig. Unsere Tastatur mit den 24 Einzeltasten würde ebenfalls 24 Leitungen benötigen und, wenn wir noch die 4 Eingänge und Ausgänge hinzunehmen, müßten bereits 94 Leitungen zu unserem Mikroprozessor führen. Da wir auch noch eine RESET-Taste, die Takt-Frequenz, die Stromzufuhr, Carry- und ZERO-Flag usw. haben, wären ganze Kabelwülste notwendig, um die Zentraleinheit Mikroprozessor mit den übrigen Computer-Elementen zu verbinden.

Wir sehen bereits – so geht das nicht. Man hat sich daher überlegt, wie man mit wesentlich weniger Leitungen auskommen kann. Das Resultat sind die sogenannten Bus-Verbindungen, wodurch mit den gleichen Verbindungsleitungen nicht nur einzelne, sondern eine ganze Reihe Bauelemente miteinander verbunden sind.



Über den sogenannten Daten-Bus (je nach Mikroprozessor-Typ sind 4 bis 16 Leitungen erforderlich) werden die Daten vom Mikroprozessor zum Speicher, zur Anzeige usw. in schnellstem Wechsel hin und her transportiert. Durch den Adressen-Bus (weitere ca. 4 bis 16 Leitungen) teilt der Mikroprozessor den einzelnen Komponenten mit, von welcher Stelle er Daten haben möchte, bzw. zu welcher Stelle die Daten transportiert werden sollen. Der Mikroprozessor ist also nicht nur ein Rechenwerk, sondern eine Steuereinheit, die ganz nebenbei und ohne daß wir etwas davon bemerken, sämtliche angeschlossenen System-Komponenten steuert.

Nehmen wir als Beispiel unser Display mit der 6-stelligen Leuchtanzeige. Wenn auf dem Display 6 verschiedene Ziffern leuchten, so ist dies (wie wir gleich erfahren) eine optische Täuschung. Der Mikroprozessor steuert über den Adress-Bus zunächst nur eine Leuchtanzeige an und übermittelt die darzustellende Ziffer. Diese Ziffer leuchtet nur kurz auf und wird sofort wieder abgeschaltet, weil der Mikroprozessor inzwischen die zweite Leuchtanzeige angesteuert hat und dort eine andere Ziffer anzeigt. In gleicher Weise werden alle 6 Leuchtziffern nacheinander angesteuert – das Ergebnis angezeigt – und wieder abgeschaltet. Diese Schaltvorgänge werden derartig schnell ausgeführt, weshalb sie vom menschlichen Auge nicht wahrgenommen werden. Es entsteht der Eindruck, daß alle 6 Stellen gleichzeitig sichtbar sind.

Während der Mikroprozessor die Leuchtanzeigen steuert, kontrolliert er gleichzeitig durch ständiges Abfragen, ob eine der 24 Tasten betätigt wurde. Auch diese Kontrollarbeit erfolgt nur scheinbar gleichzeitig. Tatsächlich arbeitet der Mikroprozessor jede Aufgabe einzeln nacheinander ab, wie z. B. Ziffern anzeigen, Daten aus dem Speicher holen und anzeigen, Tastatur kontrollieren und eingegebene Werte anzeigen, einen Programm-Schritt durchführen und anzeigen, Daten aus dem Speicher holen oder in den Speicher zurückstellen, Rechenoperationen durchführen usw. Nachdem wir dies alles nicht bemerken, können wir uns vorstellen, mit welcher ungeheuerlichen Geschwindigkeit der Mikroprozessor arbeitet. Die wenigste Zeit ist er mit der Bearbeitung eines Programmes beschäftigt. Ein wesentlich größerer Zeitaufwand ist für die Verwaltung der umliegenden Komponenten notwendig. Wir werden später sehen, daß der Mikroprozessor fast dreimal schneller ein Programm ausführen kann, wenn z. B. die Leuchtanzeige durch den Befehl DISOUT abgeschaltet ist. Deshalb wird im zweiten Teil des Anleitungsbuches bei längeren Programmen das Display abgeschaltet, solange der Computer rechnen muß. Erst wenn das Ergebnis vorliegt, wird die Leuchtanzeige wieder in Betrieb genommen.

Beim Programmieren haben wir vielleicht bemerkt, daß die 4 Leuchtdioden an den Ausgängen in unregelmäßiger Folge aufleuchten. Für die 4 Ausgänge und für den Programm-Speicher wird der gleiche Daten-Bus benutzt. Wird ein Befehl vom Mikroprozessor in den Speicherbaustein gebracht, werden durch Umwandlung in das Dual-System die LED's an den Ausgängen verändert. Da auch der Piezo-Summer über eine der Ausgangsleitungen angesteuert wird, sollten wir diesen beim Programmieren abschalten, damit uns sein „ständiges Piepen“ nicht stört.

## Nach Addition folgt Subtraktion

Nachdem wir einen Einblick in die inneren Vorgänge unseres Computers erhalten haben, sollten wir noch einige wichtige Befehle kennenlernen.

Nicht nur für Rechenaufgaben, sondern auch für mancherlei andere Programm-Schritte werden wir den Subtraktions-Befehl benötigen. Er ist ähnlich wie der Additions-Befehl aufgebaut: Wir lernen seine Funktion durch die folgende kurze Programmeingabe (nach HALT – NEXT – 00):

Das Display zeigt 0. Wir geben nacheinander zwei Zahlen ein, z. B. 7 und 2. In der Anzeige erscheint das Ergebnis 5. Der Computer hat die Subtraktion durchgeführt.

Auch das Subtrahieren erfolgt hexadezimal. Dies wird uns bestätigt, wenn wir eingeben B – 3 = 8.

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
00	<b>F08</b>	CLEAR	Alle Register löschen
01	<b>F10</b>	DISP 1,0	Register 0 anzeigen
02	<b>FF0</b>	KIN 1	Eingabe in Register 0
03	<b>FF1</b>	KIN 1	Eingabe in Register 1
04	<b>610</b>	SUB 1,0	Inhalt von Register 1 von Register 0 abziehen
05	<b>C01</b>	GOTO 01	Sprung z. Progr.-Anfang

Programm-Start: HALT – NEXT – 00 – RUN.

Interessant ist die Eingabe 7 – 8. Als Ergebnis erscheint nicht wie von uns erwartet das negative Ergebnis – 1, sondern das hexadezimale F. Daß dieses Ergebnis negativ gerechnet wurde, erkennen wir durch das aufleuchtende Carry-Flag.

Während bei der Addition beim Überspringen der hexadezimalen Ziffer F das Carry-Flag den erforderlich werdenden Übertrag signalisiert, wird es bei der Subtraktion (also beim Rückwärtszählen) nach dem Überschreiten der Ziffer 0 gesetzt.

Der neue **SUB**-Befehl hat den pauschalen Befehls-Code: **6sd**. Der Inhalt von Register **s** wird von Register **d** abgezogen. Das Ergebnis steht dann wieder im Register **d** ( $d - s = d$ ). Während der ADD-Befehl hinzuzählt, wird mit dem SUB-Befehl das abziehen erreicht.

Ähnlich wie bei der Addition gibt es auch einen Befehl, um einen konstanten Wert von einem Register abzuziehen: **SUBI** (**sub immediate**). Der pauschale Befehls-Code ist **7nd** (ein konstanter Wert in Register **n** wird vom Inhalt des Registers **d** abgezogen).

Damit wir auch das Carry-Flag für einen Übertrag verwenden können, haben wir den Befehl: **SUBC** (**sub carry**). Der pauschale Befehls-Code lautet **FCd** (vom Inhalt des Registers **d** wird der Wert 1 abgezogen, wenn ein Carry-Flag gesetzt wurde).

Die bisher besprochenen Additions-Programme (Automatik-Zähler, 6-stellige Addition usw.) können leicht zu Rückwärts-Zählern umprogrammiert werden, indem wir die Befehle ADD durch SUB, ADDI durch SUBI, bzw. ADC durch SUBC austauschen. Wir sollten zunächst einmal das Zählerprogramm im Kapitel „Vergleichen – ein wichtiger Befehl“ zu einem Rückwärts-Zählerprogramm umschreiben, indem wir das geänderte Programm zunächst in die nachfolgende Tabelle eintragen und dann eingeben:

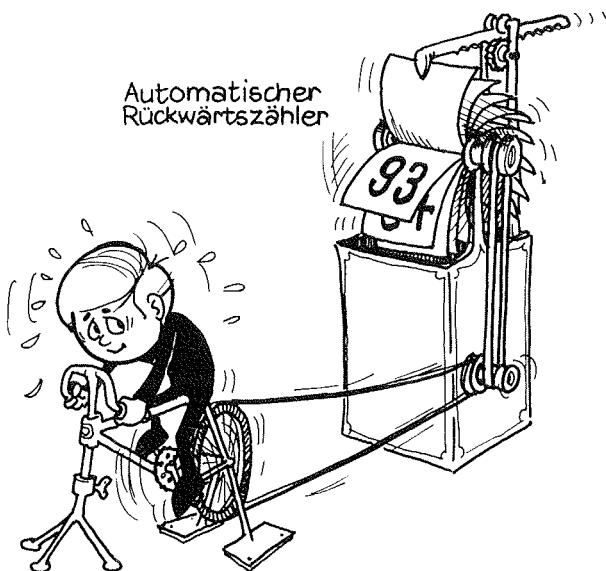
## Programmierung: Automatischer Rückwärts-Zähler

Adresse	Eingabe-Befehl	Mnemonic
00		
01		
02		
03		
04		
05		
06		
07		
08		
09		
0A		
0B		
0C		
0D		

Wir starten das Programm wie üblich mit HALT – NEXT – 00 – RUN. Unsere Rückwärts-Zähler beginnt bei 99 und zählt 98 – 97 – 96... bis er 00 erreicht hat und wieder bei 99 beginnt. Sollte dies nicht der Fall sein, können wir uns die Programm-Lösung auf der folgenden Seite ansehen.

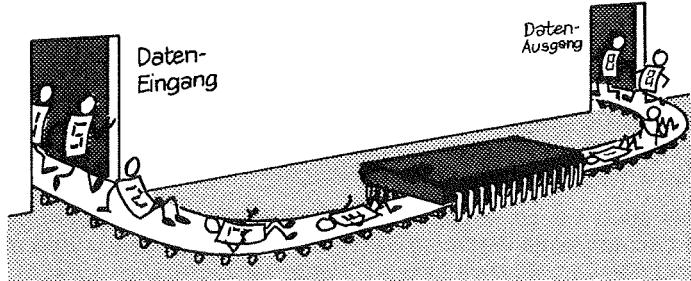
## Programm-Lösung: Automatischer Rückwärts-Zähler

Adresse	Eingabe-Befehl	Mnemonic
00	F08	CLEAR
01	F21	DISP 2,1
02	710	SUBI 1,0
03	FC1	SUBC 1
04	991	CMPI 9,1
05	D07	BRC 07
06	C02	GOTO 02
07	761	SUBI 6,1
08	712	SUBI 1,2
09	992	CMPI 9,2
0A	D0C	BRC 0C
0B	C02	GOTO 0,2
0C	762	SUBI 6,2
0D	C02	GOTO 02



Der Computer macht sich durch seine Ein- und Ausgänge bemerkbar!

## DIN und DOT



Im Kapitel „Dezimal – Hexadezimal – Dual“ haben wir den DOT-Befehl (data out) kennengelernt. Mit diesem Befehl haben wir die Möglichkeit, einen Register-Wert an die 4 Ausgänge zu bringen und den dualen Zahlenwert durch die LED's anzuziegen.

Hierfür geben wir folgendes Programm ein:

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
00	1F0	MOVI F0	F in Register 0 laden
01	FE0	DOT 0	R0 auf Ausgang legen
02	100	MOVI 00	0 in R0 laden
03	FE0	DOT 0	R0 auf Ausgang legen
04	C00	GOTO 00	Sprung zum Programm-Anfang

Bevor wir das Programm in Betrieb nehmen, schließen wir noch den Piezo-Summer an. (⊖ Leitung an die Platinen-Anschlußbuchse GND, ⊕ Leitung an den Ausgang Nr. 1). Programm-Start mit HALT – NEXT – 00 – RUN!

Alle 4 LED's an den Ausgangsleitungen blinken – der Piezo-Summer gibt einen unterbrochenen Ton.

Die Funktionen des Programmes: Zuerst wird durch den MOVI-Befehl ein konstanter Wert in Register 0 geladen. Wenn wir mit der Tabelle des dualen Zahlen-Systems vergleichen, stellen wir fest, daß der hexadezimale Wert F durch 1111 dual dargestellt wird. Mit dem nächsten Befehl wird der Inhalt des Registers 0 auf die Ausgänge gebracht. Da im Register 0 der Wert F (dual 1111) steht leuchten alle 4 LED's und der Signaltongeber summt.

Durch den folgenden Befehl MOVI 00 wird in das Register 0 der Wert 0 geladen, was der dualen Darstellung 0000 entspricht. Durch den nächsten DOT-Befehl wird dieses duale Ergebnis auf die Ausgänge gebracht, d. h. die LED's werden abgeschaltet – der Signaltongeber verstummt. Durch den Sprung zum Programm-Anfang wiederholt sich dieses Spiel in schnellem Wechsel.

Nun sollten wir ein bißchen experimentieren. Anstelle von 4 Leuchtdioden soll nur eine LED blinken – was ist zu tun?

Betrachten wir uns doch noch einmal die nachstehende Tabelle, in welcher die hexadezimalen Werte 0 bis F und die dualen Werte gegenübergestellt sind.

Eingegebener Tasten-Wert	Dual-System dargestellt durch Leuchtdioden an den Ausgängen: 4 3 2 1
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
A	1 0 1 0
B	1 0 1 1
C	1 1 0 0
D	1 1 0 1
E	1 1 1 0
F	1 1 1 1

Wir sehen bei F den dualen Wert 1111. Wenn nur eine LED blinken soll, ergibt sich bei 8 der duale Wert 1000. Also verändern wir den Befehl MOVI F0 in MOVI 80, d. h. wir geben unter Adresse 00 (statt bisher 1F0) neu ein: 180.

Nach dem Programm-Start wird nur noch die rechte Leuchtdiode am Ausgang Nr. 4 der Computer-Platine blinken. Der am Ausgang 1 angeschlossene Piezo-Summer ist verstummt.

Dies ist logisch, weil ja der Ausgang Nr. 1 kein Signal erhält und somit auch der Piezo-Summer keine Betriebsspannung hat. Nun möchten wir erreichen, daß ebenfalls nur eine LED blinkt – und daß der Piezo-Summer wieder „piepst“. Was ist zu tun? Da wir hierfür ein Signal am Ausgang Nr. 1 benötigen, sehen wir aus der Tabelle, daß wir den Tastenwert 1 eingeben müssen. Also ändern wir bei Adresse 00 den bisher eingegebenen Befehl „180“ in „110“. Wir denken daran, daß wir nach einer neuen Befehls-Eingabe zuerst die NEXT-Taste betätigen und erst dann den Programm-Start wie üblich mit HALT – NEXT – 00 – RUN beginnen. Jetzt blinkt nur noch die linke LED, der Summer „piepst“.

Wir erkennen, daß wir durch entsprechende Programmierung des Computers die 4 Ausgangsleitungen ansteuern können, wobei wir später durch entsprechende Anschlüsse an diesen Ausgängen noch zu überraschenden Effekten kommen. An den Ausgangsbuchsen sollten wir jedoch zunächst nur Bauelemente anschließen, die innerhalb unserer Experimente aufgeführt werden.

Außer den 4 Ausgängen hat unser Microtronic-Computer noch 4 Eingänge, die mit dem DIN-Befehl (data in) angesprochen werden.

Wir verbinden die Buchse „Takt/Clock“ mit dem „Eingang 1“. Der Piezo-Summer kann angeschlossen bleiben.

Wir geben das folgende kurze Programm ein:

Adresse	Eingabe-Befehl	Befehl-Mnemonic
00	<b>FD0</b>	DIN 0
01	<b>FE0</b>	DOT 0
02	<b>C00</b>	GOTO 00

Vor dem Programm-Start sollte der Piezo-Sommer angeschlossen werden und es darf nur die Verbindungsleitung zwischen „Eingang 1“ und „Takt/Clock“ bestehen. Alle übrigen Aus- und Eingänge sind nicht miteinander verbunden.

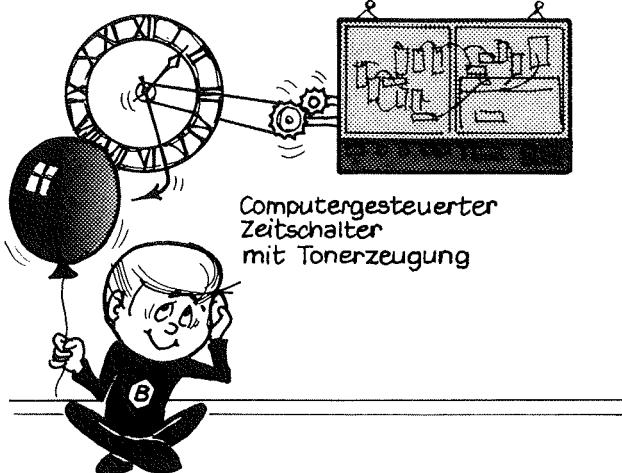
Nachdem Programm-Start wird die LED am Ausgang 1 im gleichen Takt wie die LED an „Takt/Clock“ blinken. Der Piezo-Summer bringt ebenfalls einen im Takt unterbrochenen Ton.

Durch den Befehl DIN 0 wird in das Register 0 der Wert geladen, welcher am Eingang als Spannung anliegt. Ist z. B. am „Takt/Clock“ eine Spannung vorhanden, dann erkennt der Computer an der Eingangsbuchse Nr. 1 die wechselnden „Zustände“: Spannung vorhanden – keine Spannung vorhanden – Spannung vorhanden . . . Im Zeitpunkt „Spannung vorhanden“ ergibt sich der duale Wert 0001, welcher in das Register 0 geladen wird. Durch den folgenden Befehl DOT 0 wird dieser Wert auf den Ausgang gebracht.

Wir können nun die Verbindungsleitung von der Buchse Takt/Clock an einen anderen Eingang legen oder sämtliche Eingangsbuchsen mit Takt/Clock verbinden – es werden immer die entsprechenden LED's an den Ausgängen mitblinken.

Wir erkennen, daß wir ein Eingangs-Signal durch eine entsprechende Programmierung an einen oder mehrere Ausgänge bringen können und es leuchtet uns ein, daß der Computer hierdurch Steuerungsaufgaben übernehmen kann. Diese Möglichkeiten werden wir bei späteren Experimenten noch weiter ausbauen.

## Timer – der Computer als Zeit-Schalter



Unter der Bezeichnung Timer verstehen wir Uhren die „rückwärts“ laufen, um nach einer bestimmten eingestellten Zeit z. B. einen Signalton auszulösen.

Für diesen Versuch bleibt der Piezo-Summer angeschlossen. Außerdem muß eine Verbindungsleitung zwischen Eingang 4 und Takt/Clock vorhanden sein. Zweckmäßigerverweise löschen wir vor Programm-Eingabe den gesamten Programm-Speicher mit: HALT – PGM – 5. Die einzelnen Befehle müssen sehr sorgfältig eingegeben werden, weil bei der Verwechslung einer Zahl oder eines Buchstabens das Programm nicht funktionieren wird. Wir achten auch darauf, daß wir pro Adresse jeweils nur 3 Zeichen eingeben und dann die NEXT-Taste betätigen.

### Programm: Timer als Zeitschalter

Sprungziel	Adr. Nr.	Eingabe-Befehl	Mnemonic	Sprung nach:	Erklärungen
„START“	00	<b>F08</b>	CLEAR	alle Register löschen	
	01	<b>FE0</b>	DOT 0		Ausgänge auf Null setzen
	02	<b>F41</b>	DISP 4,1		Reg. 1 – 4 anzeigen
	03	<b>FF0</b>	KIN 0		Eingabe in Reg. 0
	04	<b>990</b>	CMPI 9,0		Eingabe größer als 9?
	05	<b>D0B</b>	BRC 0B	„TIMER“	Ja, dann Sprung nach „TIMER“
	06	<b>034</b>	MOV 3,4	Zahlen eine Stelle weiterverschieben	
	07	<b>023</b>	MOV 2,3		
	08	<b>012</b>	MOV 1,2		
	09	<b>001</b>	MOV 0,1		
	0A	<b>C02</b>	GOTO 02		Sprung nach 02 für nächste Eingabe
„TIMER“	0B	<b>FD5</b>	DIN 5	Warte-schleife	Takt von Eingängen in Reg. 5
	0C	<b>856</b>	CMP 5,6		Reg. 5 und Reg. 6 gleich?
	0D	<b>E0B</b>	BRZ 0B		Ja, dann Sprung nach „TIMER“
	0E	<b>056</b>	MOV 5,6		R 5 in R 6 abspeichern
	0F	<b>905</b>	CMPI 0,5		Ist R 5 = 0?
	10	<b>E0B</b>	BRZ 0B		Ja, dann Sprung zu „TIMER“

Sprungziel	Adr. Nr.	Eingabe-Befehl	Mnemonic	Sprung nach:	Erklärungen
	11	<b>711</b>	SUBI 1,1		R 1 (Sekunden) 1 abziehen
	12	<b>D14</b>	BRC 14		Übertrag? Sprung nach 14
	13	<b>C1F</b>	GOTO 1F	„NULL“	Sprung nach „NULL“?
	14	<b>191</b>	MOVI 9,1		9 in R 1 speichern
	15	<b>712</b>	SUBI 1,2		R 2 (10 Sekunden) 1 abziehen
	16	<b>D18</b>	BRC 18		Übertrag? Sprung nach 18
	17	<b>C1F</b>	GOTO 1F	„NULL“	
	18	<b>152</b>	MOVI 5,2		5 in R 2 speichern
	19	<b>713</b>	SUBI 1,3		R 3 (Minuten) 1 abziehen
	1A	<b>D1C</b>	BRC 1C		Übertrag? Sprung nach 1C
	1B	<b>C1F</b>	GOTO 1F	„NULL“	
	1C	<b>193</b>	MOVI 9,3		9 in R 3 speichern
	1D	<b>714</b>	SUBI 1,4		R 4 (10 Minuten) 1 abziehen
	1E	<b>D2B</b>	BRC 2B	„ENDE“	Übertrag? Sprung nach „ENDE“
„NULL“	1F	<b>904</b>	CMPI 0,4		NULL?
	20	<b>E22</b>	BRZ 22		
	21	<b>C0B</b>	GOTO 0B		Vergleiche ob alle Registerinhalte 0 sind, wenn ja, dann „ENDE“, wenn nein, Sprung zu „TIMER“ und weitere Taktimpulse abwarten
	22	<b>903</b>	CMPI 0,3		
	23	<b>E25</b>	BRZ 25		
	24	<b>C0B</b>	GOTO 0B		
	25	<b>902</b>	CMPI 0,2		
	26	<b>E28</b>	BRZ 28		
	27	<b>C0B</b>	GOTO 0B		
	28	<b>901</b>	CMPI 0,1		
	29	<b>E2B</b>	BRZ 2B	„ENDE“	
	2A	<b>C0B</b>	GOTO 0B	„TIMER“	
„ENDE“	2B	<b>1F0</b>	MOVI F,0		
	2C	<b>FE0</b>	DOT 0		F in R 0 speichern und auf die Ausgänge gehen
	2D	<b>FF0</b>	KIN 0		Tastendruck abwarten
	2E	<b>C00</b>	GOTO 00		Sprung zum Programm-Anfang

Würden wir versehentlich 4 Zeichen eingegeben, erscheinen zwar nur die 3 zuletzt eingegebenen Zeichen im Display – bei der nächsten Befehls-Eingabe bleibt der Computer jedoch stehen, d. h., er nimmt keine weiteren Befehle an. In einem solchen Fall müssen wir bei der falschen Adresse die Programm-Eingabe wiederholen: Wir wählen mit HALT – NEXT und Angabe der Adresse an, betätigen zweimal die Taste C/CE, wodurch der falsche Befehl gelöscht wird und neu eingegeben werden kann. Mit NEXT können wir dann weiter programmieren.

Falls uns der Piezo-Summer beim Programmieren gestört hat, sollten wir ihn jetzt wieder anschließen. Wir starten das Programm wie üblich mit HALT – NEXT – 00 – RUN. Das Display zeigt 0000.

Auf den ersten beiden Stellen können wir mit den Zahlen-Tasten die Minuten eingeben – die letzten beiden Stellen sind für die Sekunden-Eingabe. Damit wir beim ersten Versuch nicht zu lange warten müssen geben wir ein: 0010 – also 10 Sekunden. Wird jetzt die Taste A betätigt, zählt unser Timer im Sekundentakt rückwärts und der Piezo-Summer ertönt, sobald die Sekunde 00 erreicht ist. Durch erneute Betätigung der Taste A wird der Summer abgeschaltet und es kann eine neue Zeit-Eingabe vorgenommen werden. Die höchstmögliche Timer-Zeit sind 99 Minuten und 59 Sekunden.

Wenn wir z. B. in einer halben Stunde ein wichtiges Telefongespräch führen möchten, geben wir unserem Computer die 30 Minuten ein und er wird uns durch seinen Summton pünktlich daran erinnern, daß wir unser Telefongespräch nicht vergessen.

Sollte das Programm nicht einwandfrei funktionieren, fehlt entweder die Verbindungsleitung von der Eingangsbuchse 1 zu Takt/Clock oder wir haben beim Programmieren einen Fehler gemacht. Wenn wir eine eingegebene Zeit nicht bis zum Ende abwarten möchten, gehen wir mit HALT – NEXT – 00 – RUN wieder an den Programm-Anfang und der Computer erwartet eine neue Eingabe.

### Programm-Beschreibung:

Zur besseren Übersicht wurde das Programm in mehrere Abschnitte unterteilt: **START** beginnt bei Adresse 00 und endet bei Adresse 0A. Wir bezeichnen diesen Teil als **Eingabe-Routine**, weil wir hier die Werte für Minuten und Sekunden eingeben, die dann vom Display angezeigt werden. Von Adresse 0B bis 1E haben wir das eigentliche **TIMER**-Programm. Den Programm-Teil **NULL?** haben wir bei den Adressen 1F bis 2A. Hierauf kommen wir noch zurück, ebenso auf das **ENDE**-Programm.

Nun sollten wir uns zweckmäßigerweise den Programm-Ablaufplan genau ansehen. Zum besseren Verständnis wurden bei den einzelnen Symbolen die Programm-Adressen angegeben. So finden wir (gleichlaufend mit der Programm-Tabelle) beim Symbol START die Adresse 00. Hier werden alle Register gelöscht. Beim nächsten Symbol (Adresse 01 und 02) werden die Ausgänge abgeschaltet, damit sich unser Piezo-Summer nicht vorzeitig bemerkbar macht. Außerdem werden die Register R1 bis R4 auf Display-Anzeige geschaltet, wobei durch die gelöschten Register zunächst 0000 angezeigt wird.

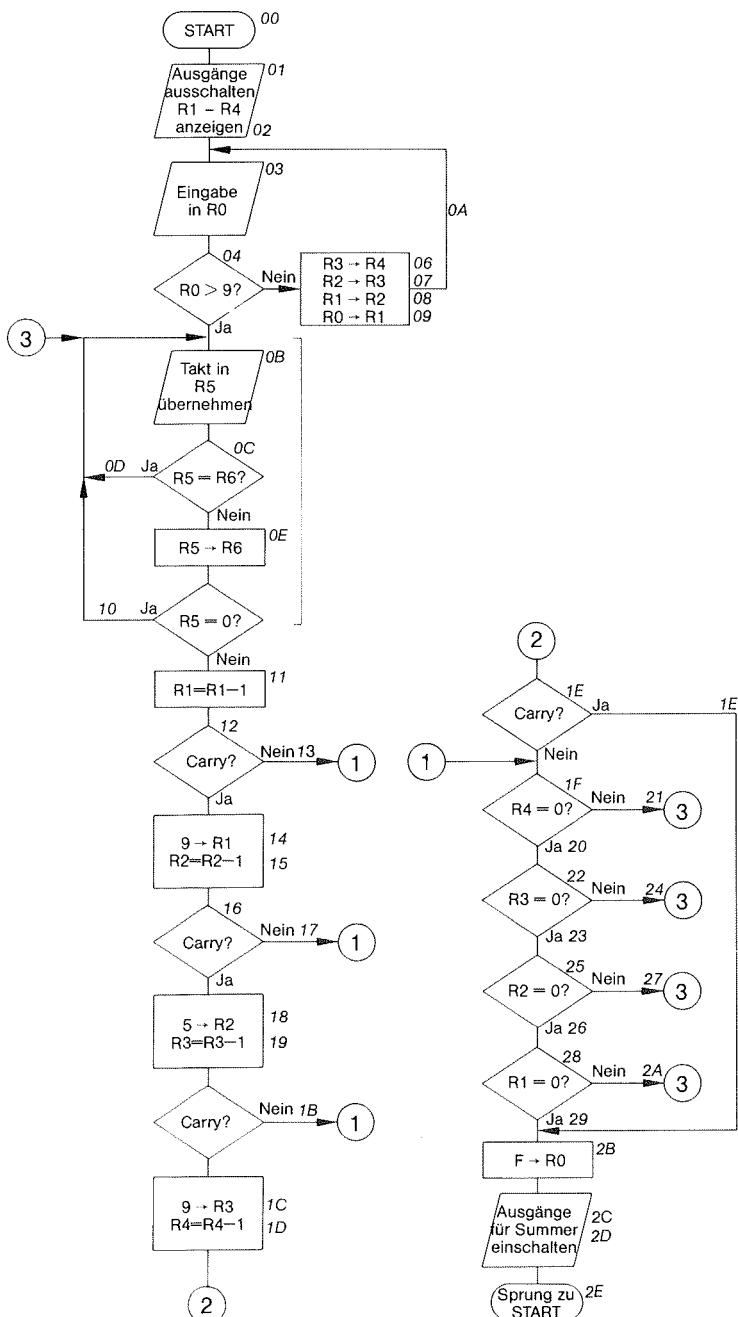
Beim folgenden Symbol (Adresse 03) kann die Eingabe vorgenommen werden (Minuten – Sekunden). Unter 04 erkennen wir das Symbol für einen Vergleichs-Befehl. Wurde eine Eingabe vorgenommen, so wandert der Wert zunächst in das Register 0 und der Vergleichs-Befehl fragt ab, ob ein Wert größer als 9 im Register 0 vorhanden ist? Gehen wir doch einmal davon aus, daß wir 12 Minuten und 34 Sekunden als Eingabe vorgesehen haben. Somit geben wir als erste Ziffer den Wert 1 ein. Die Frage des Vergleichs-Befehls, ob in Register 0 ein Wert größer als 9 eingegeben wurde, wird mit „nein“ beantwortet. Der nach rechts zeigende Pfeil demonstriert uns, daß hierdurch der Schiebe-Befehl angesprochen wurde, d. h. daß alle für die Display-Anzeige vorgesehenen Werte um eine Stelle verschoben werden. Somit wandert der eingegebene Wert 1 von Register 0 in das Register 1. Die im Register 1 (durch den CLEAR-Befehl) vorhandene 0 wandert ins Register 2 usw. Da wir über die Tastatur z. B. 12 Minuten und 34 Sekunden, also die Werte 1-2-3-4 nacheinander eingeben, wird dies jeweils vom Vergleichs-Befehl (Adresse 04) festgestellt. Durch den Schiebe-Befehl „füllen“ sich die Register 1 bis 4 und die Werte werden auf dem Display angezeigt.

Nun wird die Taste A betätigt. Auch dies wird vom Vergleichs-Befehl bei Adresse 04 registriert (die Frage, ob der in Register 0 eingegebene Wert größer als 9 ist, wird mit „ja“ beantwortet), wodurch bei unserem Ablaufplan die „Weiche“ nun nach unten weitergestellt wird.

Beim nächsten Symbol (0B) wird der Takt in das Register R5 übernommen. Welchen Takt? – Wir haben an der Computer-Platine eine Verbindungsleitung von TAKT/CLOCK zum Eingang Nr. 4 hergestellt. Die TAKT-LED blinkt im Sekunden-Takt, d. h., eine halbe Sekunde leuchtet die LED und eine halbe Sekunde ist sie ausgeschaltet. Wir erinnern uns „Spannung vorhanden“ bedeutet für den Computer den Wert 1 (LED leuchtet).

„Keine Spannung vorhanden“ bedeutet für den Computer den Wert 0 (LED leuchtet nicht). Ist also eine Spannung vorhanden, wird bei der Adresse 0B der Wert 1 in Register 5 (R5) übernommen.

Beim nächsten Symbol (Adresse 0C) folgt wieder ein Vergleichs-Befehl. Frage: Ist der Inhalt von Register 5 gleich groß wie in Register 6? Nachdem in Register 6 durch den anfänglichen CLEAR-Befehl der Wert 0 steht, wird die Frage mit „nein“ beantwortet. Damit kommen wir in unserem Ablaufplan zu dem unten folgenden Symbol (Adresse 0E) Register-Inhalt 5 in Re-



gister 6 übertragen. Somit stehen in den beiden Registern R5 und R6 die gleichen Werte, nämlich 1.

Der folgende Vergleichs-Befehl fragt: „Ist in Register 5 der Wert 0?“ Die Antwort lautet „nein“, somit weiter zur nächsten Adresse 11. Hier lautet der Befehl: Von Register 1 den konstanten Wert 1 abziehen. Wir haben die Werte 12 34 (12 Minuten, 34 Sekunden) eingegeben (die durch MOV in die Register 1 bis 4 „geschoben wurden“) Im angezeigten Register Nr. 1 stand somit bisher der Wert 4, welcher durch den Minus-Befehl zum Wert 3 wird.

Gehen wir weiter zum Symbol bei Adresse 12. Dort fragt der Vergleichs-Befehl, ob ein CARRY gesetzt wurde? Ein CARRY wird jedoch erst dann gesetzt, wenn beim Rückwärtszählen der Wert 0 erreicht wurde (also bei 0 Sekunden). Da in unserem Fall kein CARRY gesetzt wurde, heißt die Antwort „nein“ und der am Symbol nach rechts zeigende Pfeil zeigt, daß das Programm bei ① (also im rechten Teil des Ablauf-Plans) fortgesetzt wird.

Damit kommen wir zu den 4 untereinanderstehenden Vergleichs-Befehlen (Adresse 1F, 22, 25 und 28). Hier wird jeweils gefragt, ob in einem der 4 angezeigten Register R1 bis R4 der Wert 0 erreicht wurde.

Was hier Schritt für Schritt erläutert werden muß, wird vom Computer im Bruchteil einer Sekunde verarbeitet. Da wir bei unserer Beschreibung immer noch bei der letzten Sekundenstelle sind, welche jetzt mit 3 auf der Anzeige steht, werden alle Fragen der 4 Vergleichs-Befehle mit „nein“ beantwortet. Somit zeigen alle Pfeile des Ablaufplans nach rechts zur ③. Also kommen wir wieder zur ③ links oben im Ablaufplan. Da nur Sekundenbruchteile vergangen sind und die TAKT-LED immer noch leuchtet, wird bei Adresse 0B erneut der Wert 1 übernommen. Beim folgenden Vergleichs-Befehl (0C) ergibt sich wieder die Frage, ob im Register R5 der gleiche Wert wie in Register R6 steht. Nachdem wir zuvor festgestellt haben, daß R5 und R6 bei Adresse 0E gleichwertig gemacht wurden, lautet die Antwort „ja“, wodurch die (am nach links zeigenden Pfeil stehende) Adresse 0D angesprochen wird. Dem Ablaufplan folgend kommen wir wieder zu ③. Aus dieser Warteschleife kommen wir so lange nicht heraus, bis die TAKT-LED nicht mehr leuchtet. Jetzt ist „keine Spannung“ am Eingang Nr. 4 vorhanden, wodurch der Wert 0 erreicht wird. (Keine Spannung vorhanden = 0).

Der beschriebene Vorgang wiederholt sich mit geringen Abwandlungen: Bei Adresse 0B wird der neue TAKT-Wert 0 in das Register R5 übernommen. Bei 0C wird verglichen: „Ist Inhalt R5 und R6 gleich?“ Die Antwort lautet „nein“, weil ja in R6 noch der Wert 1 gespeichert ist. Damit folgt Adresse 0E: Wert von Register 5 wird in Register 6 übernommen, womit jetzt in beiden Registern wieder der gleiche Wert 0 steht.

Wir kommen zum nächsten Vergleichs-Befehl bei 0F: „Ist der Inhalt von Register R5 eine 0?“. Die Antwort lautet „ja“, womit wir erneut in der Warteschleife ankommen. Wir müssen wieder berücksichtigen, daß der Vorgang in Bruchteilen einer Sekunde abläuft, während sich unsere Beschreibung momentan in der Phase befindet, in welcher die TAKT-LED nicht leuchtet. In diesem Zeitraum wird das TIMER-Programm durch die „Weichenstellung“ beim Vergleichs-Befehl Adresse 0F nicht angesprochen.

Sobald die TAKT-LED wieder ein Signal erhält (Spannung vorhanden – Wert 1) wird der Wert 1 wieder in das Register R5 übernommen, die „Weichenstellung“ beim Vergleichs-Befehl Adresse 0F wird geändert und das sich anschließende TIMER-Programm läuft ab, wodurch bei der letzten Sekundenstelle der noch vorhandene Wert 3 auf 2 verringert wird. Wir waren ja davon ausgegangen, daß wir am Anfang unserer Beschreibung 12 Minuten 34 Sekunden eingegeben haben, und daß durch den inzwischen zweimal ausgeführten TIMER-Vorgang 12 Minuten 32 Sekunden übrig geblieben sind.

Nach zwei weiteren Durchläufen wird in Register 1 (bei der rechts auf dem Display angezeigten letzten Sekundenstelle) der Wert 0 erreicht sein. Wird jetzt nochmals der Wert 1 abgezogen, ergibt sich durch das rückwärts Zählen der hexadezimale Wert F. Hierdurch wird das CARRY-FLAG ausgelöst. Bei Adresse 12 wird gefragt, „CARRY-FLAG ausgelöst?“ Die Antwort lautet „ja“ – somit weiter zur Adresse 14: In das Register R1 wird der konstante Wert 9 übernommen (der bisher dort stehende Wert

F wird gelöscht) und bei Adresse 15 wird in Register 2 (die zweitletzte Sekundenstelle des Displays) der konstante Wert 1 abgezogen.

Bei Adresse 16 wird wieder gefragt, ob ein CARRY-FLAG gesetzt wurde – Antwort „nein“. Damit Fortsetzung des Programms bei ①. Bei den folgenden vier Vergleichs-Befehlen wird jeweils abgefragt, ob die angezeigten Display-Stellen (Register R1 bis R4) bereits den Wert 0 erreicht haben. Nachdem dies nicht der Fall ist (bei unserem simulierten Vorgang zeigt das Display an: 12 Minuten 29 Sekunden) wird das Programm wieder bei ③ fortgesetzt.

Dieser Vorgang wiederholt sich solange, bis (bei ①) alle Register-Werte bei 0 angelangt sind. Damit kommen wir zum Programmteil ENDE.

Bei Adresse 2B wird der Wert F in das Register R0 übernommen und die folgenden Befehle 2C und 2D schalten die Ausgänge des Computers ein – der Piezo-Summer gibt ein Signal. Das Programm springt zur START-Adresse 00. Alle Register werden gelöscht – das Display zeigt 0000. Neue Werte können eingegeben werden.

Es würde zuweit führen und sicherlich auch langweilig werden, wenn man alle noch folgenden Programme in dieser Ausführlichkeit darstellen wollte. Da es jedoch wichtig ist, die Programm-Vorgänge in allen Einzelheiten zu verstehen, ist es sinnvoll, wenn wir nochmals eine simulierte Eingabe vornehmen. Mit Farbstiften den Programm-Ablaufplan abfahren, um durch farbige Linien den sich immer wieder ändernden Ablauf kenntlich zu machen.

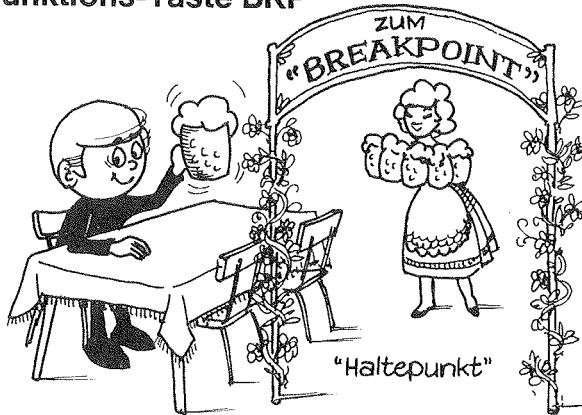
Wir sollten uns auch merken, wie wir die Register für dieses Programm eingeteilt (belegt) haben. Dies ersehen wir aus nachfolgender Tabelle, wobei wir bei „simulierte Werteingabe“ mit Bleistift eintragen können, welche Werte für Minuten und Sekunden für einen angenommenen Programm-Ablauf eingeben werden (z. B. 12 Minuten 34 Sekunden) was bedeuten würde, daß wir bei Register R1 den Wert 4, bei R2 den Wert 3, bei R3 den Wert 2, bei R4 den Wert 1 eintragen.

### Register-Belegung

	Reg.Nr. Verwendungszweck	simulierte Werteingabe
R6	Vergleichsregister	
R5	Eingangsregister für TAKT	
R4	2. Minutenstelle (Display linke Stelle)	<input type="checkbox"/>
R3	1. Minutenstelle (2. Display-Stelle)	<input type="checkbox"/>
R2	2. Sekundenstelle (3. Display-Stelle)	<input type="checkbox"/>
R1	1. Sekundenstelle (Display rechte Stelle)	<input type="checkbox"/>
R0	Eingabe-Register über Tastatur	

Eine wichtige Kontrollhilfe:

## Die Funktions-Taste BKP



**BKP** ist die Abkürzung für „breakpoint“. Man könnte den Begriff mit „Haltepunkt“ übersetzen. Der „breakpoint“ ist wichtig, wenn wir ein Programm austesten möchten um zu sehen, ob die Befehle vom Computer in der richtigen Reihenfolge abgearbeitet werden. Wir erinnern uns, daß dies auch mit der STEP-Taste möglich ist. Die STEP-Taste bringt jedoch die abzuarbeitenden Einzelschritte nur dann, wenn bei einem Programm keine zu verändernde Werte eingegeben werden, wie z. B. bei einem automatischen Zähler usw.

Wenn wir jedoch wie beim Programm TIMER die Minuten und Sekunden eingeben müssen, ist eine Durcharbeitung des Programms mit der STEP-Taste nicht möglich. Hier hilft uns die BKP-Taste weiter.

Das vorher eingegebene TIMER-Programm soll in Einzelschritten abgearbeitet werden. (Sollten wir bis zum Beginn dieses Kapitels den Computer durch Entfernen des Netzgerätes aus der Steckdose abgeschaltet haben, müßte das Programm neu eingegeben und auf richtige Funktion kontrolliert werden).

Wir möchten den „breakpoint“ bei Adresse **0B** haben, weil z. B. an dieser Stelle das eigentlich TIMER-(Rückwärtszähl-) Programm beginnt. Hierfür betätigen wir nacheinander die Tasten: HALT – BKP – **0B** – HALT – NEXT – 00. Das Programm wird wieder mit RUN gestartet, das Display zeigt 0000. Wir geben jetzt noch 10 Sekunden (10) ein und betätigen die Taste A. Das Programm läuft jetzt lediglich bis zur „breakpoint“ Adresse 0B und das Display zeigt: 0B Fd5. Nun sehen wir uns den Programm-Ablaufplan TIMER an und betätigen die Taste STEP: Die Adresse 0C (Vergleichs-Befehl) wird angezeigt. Der Vergleichs-Befehl (siehe Ablaufplan) fragt: Ist der Inhalt von R5 gleich groß wie R6? Sobald wir erneut die STEP-Taste betätigen sehen wir, was bei diesem Vergleich festgestellt wurde. Sind die Registerwerte gleich wird der nächste Programm-Schritt 0D angezeigt, wir kommen in die Warteschleife und springen zur Adresse 0B zurück. Wenn wir auch weiterhin langsam nacheinander die STEP-Taste betätigen, werden sich beim Vergleichs-Befehl auf Adresse 0C unterschiedliche Register-Werte ergeben (Spannung=1, keine Spannung=0). Wir kommen bei weiterer STEP-Tasten Betätigung aus der Warteschleife heraus. Das Programm wird wie im letzten Kapitel beschrieben, Schritt für Schritt abgearbeitet. Wir suchen uns im Ablaufplan die jeweils vom Computer angezeigten Adressen, so daß wir die beschriebenen Sprünge genau verfolgen können.

Alles, was wir vorher mühsam gelesen haben, wird vom Computer Schritt für Schritt dargestellt. Immer wenn wir bei der Adresse 0B ankommen, ist es entscheidend, ob in diesem Augenblick die TAKT-LED leuchtet (somit bei der Adresse 0B der Wert 1 festgestellt wird), oder ob die LED in diesem Augenblick nicht leuchtet (somit der Wert 0 festgestellt wird).

### Wie kann man den „breakpoint“ BKP wieder löschen?

Wir betätigen folgende Tasten: HALT – BKP – 00 – HALT. Damit haben wir den „Haltepunkt“ beseitigt.

Mit HALT – NEXT – 00 und RUN, können wir das TIMER-Programm wieder starten. Würden wir anstelle der START-Adresse 00 die Adresse 01 eingeben (HALT – NEXT – 01 – RUN), haben wir den bei Adresse 00 stehenden CLEAR-Befehl übersprungen und auf dem Display werden die restlichen Sekunden

angezeigt, die wir mit unserer STEP-Tastenbetätigung noch nicht abgearbeitet haben. Bei Druck auf Taste A werden die restlichen Sekunden vom Computer abgearbeitet bis zum Summtion am Ende.

**Bei Anwendung des „breakpoint“ und der BKP-Taste müssen wir unbedingt beachten:** Wenn wir BKP eingegeben haben dürfen wir nicht vergessen, diesen „breakpoint“ unbedingt wieder zurückzunehmen.

Vergessen wir die Rücknahme des „breakpoints“ und wir geben z. B. ein neues Programm ein, wird auch das neue Programm immer wieder auf der eingegebenen „breakpoint“-Adresse abgestoppt.

## Der Zufalls-Generator

Es ist schwierig einen Computer so zu programmieren, daß er zufällige Zahlen oder Buchstaben ermittelt. Für viele Spiele und auch für wissenschaftliche Untersuchungen werden jedoch Zufallszahlen benötigt, die vom Computer während des Programm-Ablaufs erzeugt werden müssen. Unter Zufallszahlen verstehen wir Zahlenwerte, deren Reihenfolge nicht im voraus bestimmbar ist (ähnlich wie beim Würfeln).

In Groß-Computern werden Zufallszahlen durch komplizierte Algorithmen erzeugt: Normalerweise soll der Computer nur logische Entscheidungen treffen. Für ihn ist es deshalb schwierig, unabhängig von anderen Berechnungen zu zufälligen Zahlenergebnissen zu kommen vor allem, wenn die verschiedenen Zahlen (z. B. 0-9) in einer gleichmäßigen Häufigkeit auftreten sollen.

Der Microtronic-Computer hat einen eingebauten Zufalls-Generator. Dies wird durch ein fest integriertes Programm erreicht, wodurch uns die schwierige Programmierung erspart bleibt.

Der Zufalls-Generator hat das Befehls-Kürzel (Mnemonic) **RND** was aus „random“ abgeleitet ist.

Der Zufalls-Generator wird durch den Eingabe-Befehls-Code **F05** aufgerufen. Durch F05 werden die zufällig erzeugten Zahlen in die Register D, E und F übernommen. Wir erinnern uns: Wir haben insgesamt 16 Register zur Verfügung (0 bis F), wobei der Zufalls-Generator die drei letzten Register (D, E, F) für seine Arbeit benötigt.

### Wie arbeitet der Zufalls-Generator?

Im Kapitel „Software – Hardware“ haben wir erfahren, daß der Mikroprozessor neben seinen vielen Tätigkeiten nebenher auch die Tastatur kontrolliert. In jeder Sekunde wird ca. 100 bis 200 Mal abgefragt, ob eine Taste betätigt worden ist. Immer wenn der Mikroprozessor die Tastatur kontrolliert hat, zählt er in seinem Betriebs-System (also intern) in einem 3-stelligen Zähler hexadezimal eine Stelle weiter, d. h., daß dieser interne Zähler pro Sekunde seinen Wert 100 bis 200 Mal verändert. Über den Befehl RND kann der Zählerstand in die Register D, E und F übernommen werden, wodurch wir diese Zufallszahlen bei einer entsprechenden Programmierung verwenden können.

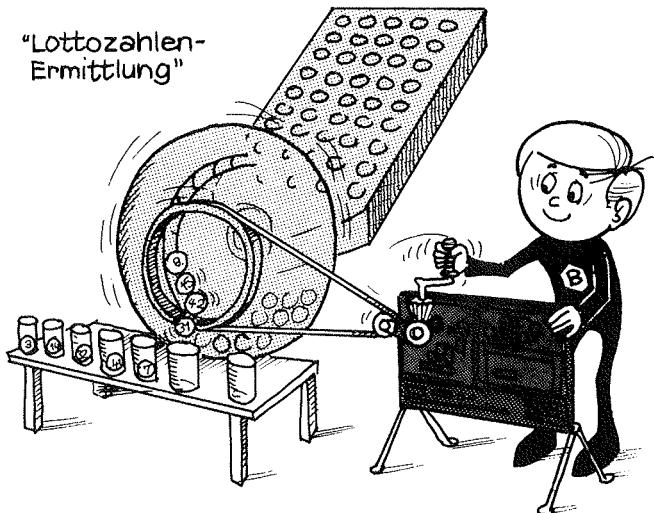
Wir möchten einmal sehen, wie der Zufalls-Generator arbeitet. Wir geben nach HALT – NEXT – 00 folgendes Kurz-Programm ein:

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
00	<b>F05</b>	RND	Zählerstand in Register D – E – F übernehmen
01	<b>F3D</b>	DISP 3,D	3 Stellen ab Register D anzeigen
02	<b>C00</b>	GOTO 00	Rücksprung, damit Zählerstand-Veränderung durchgeführt und anschließend angezeigt wird

HALT – NEXT – 00 und Programm-Start RUN.

Wir sehen, daß auf dem Display mit großer Geschwindigkeit auf 3 Stellen fortlaufend hexadezimal gezählt wird.

## Der Zufalls-Generator ermittelt Lotto-Zahlen



Lottoscheine werden oft nach den unterschiedlichsten Kriterien ausgefüllt. Teilweise werden Geburtstage oder Jahreszahlen verwendet. Andere tippen mit dem Kugelschreiber und geschlossenen Augen auf den Lottoschein und die so zufällig getroffene Zahl wird angekreuzt. Niemand ist jedoch in der Lage, die richtigen Glückszahlen im voraus zu ermitteln. Auch ein Computer kann dies nicht schaffen – aber er kann entsprechende Zufalls-Zahlen ermitteln, die wir dann auf den Lottoschein übernehmen können.

Wir geben folgendes Programm nach HALT – NEXT – 00 ein:

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
00	F02	DISOUT	Anzeige ausschalten
01	F05	RND	Zufallszahlen in Register D – E – F laden
02	99D	CMPI 9,D	Register D größer als 9?
03	D05	BRC 05	Ja, dann Sprung nach 05
04	C08	GOTO 08	Nein, Sprung nach 08
05	71F	SUBI 1,F	vom Register F konstanten Wert 1 abziehen
06	6FD	SUB F,D	vom Register D den Inhalt von Register F abziehen
07	C02	GOTO 02	Sprung nach 02
08	94E	CMPI 4,E	Register E größer als 4?
09	D0B	BRC 0B	Ja, Sprung nach 0B
0A	C0E	GOTO 0E	Nein, Sprung nach 0E
0B	71F	SUBI 1,F	Von Register F konstanten Wert 1 abziehen
0C	6FE	SUB F,E	Von Register E Register F abziehen
0D	C08	GOTO 08	Sprung nach 08
0E	90D	CMPI 0,D	Register D = 0?
0F	D12	BRC 12	Nein, dann Sprung nach 12
10	90E	CMPI 0,E	Register E = 0?
11	E00	BRZ 00	Sprung zum Programm-Anfang
12	F2D	DISP 2,D	Register D und E anzeigen
13	FF0	KIN 0	Tastatur-Eingabe abwarten
14	C00	GOTO 00	Sprung zum Programm-Anfang

## Programm-Beschreibung:

Um die Rechengeschwindigkeit zu erhöhen, wird beim Programm-Start die Anzeige auf dem Display abgeschaltet (Befehl DISOUT). Auf der Adresse 02 wird mit RND die 3-stellige Zufallszahl in die Register D, E und F übernommen. Da wir nur eine 2-stellige Zufallszahl benötigen, werden am Programm-Ende bei Adresse 12 (mit DISP 2,D) nur 2 Register, nämlich D und E angezeigt. Steht auf dem Display beispielsweise die Zahl **36**, wird die letzte Stelle, also **6** im Register D und **3** in Register E angezeigt.

Da wir nur Zahlen bis einschließlich 49 verwenden können, haben wir bei Adresse 02 einen Vergleichs-Befehl, bei welchem gefragt wird, ob der in Register D befindliche Wert größer als 9 (z. B. A B C ...) ist. Wenn „ja“ erfolgt ein Sprung zur Adresse 05. Hier bringen wir das eigentlich nicht benötigte Register F ins Spiel, indem vom Register F der konstante Wert 1 abgezogen wird. Bei Adresse 06 wird der sich nun im Register F ergebende Wert vom Register D abgezogen. Es erfolgt ein Rücksprung zur Adresse 02. Es erfolgt der gleiche beschriebene Ablauf, indem bei den Adressen 05 und 06 solange abgezogen wird, bis sich in Register D eine Zahl zwischen 0 und 9 ergibt. Erst jetzt erfolgt ein Sprung zur Adresse 08.

Der Vergleichs-Befehl prüft, ob in Register E ein Wert größer als 4 vorhanden ist. Falls „ja“ erfolgt bei den Adressen 0B und 0C mit dem Register E ähnliches wie es vorher bei Register D beschrieben wurde.

Um die Lottozahlen Ermittlung 00 zu vermeiden, haben wir unter Adresse 0E und 10 zwei Vergleichs-Befehle die prüfen, ob im Register D und E der Wert 0 vorhanden ist. Wenn „ja“ erfolgt ein Rücksprung zum Programm-Anfang. Das gesamte Spiel beginnt von vorne. Wenn „nein“ wird bei Adresse 12 die Anzeige eingeschaltet und die in den Registern D und E stehenden Ergebnisse angezeigt.

Der unter 05 vorhandene Subtraktions-Befehl SUBI 1,F ist besonders wichtig. Wäre er nicht vorhanden, würde sich das Programm in einer Endlos-Schleife „tot“ laufen. Warum wohl?

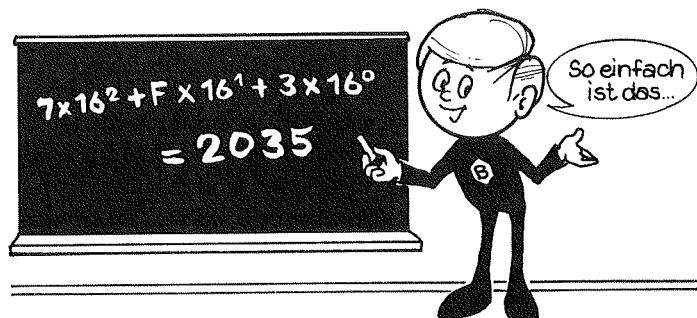
Bei Adresse 06 wird vom Register D der Wert des Registers F abgezogen. Nehmen wir an, in Register D würde der Wert B stehen und der Inhalt von Register F hätte den Wert 0. Wird vom Wert B der Wert 0 abgezogen, bleibt logischerweise der Wert B bestehen. Laut Programm-Adresse 07 erfolgt der Rücksprung zur Adresse 02. Nachdem der Wert B immer noch größer als 9 ist, erfolgt wieder der Sprung zur Adresse 05. Wäre unter Adresse 05 der SUBI-Befehl nicht vorhanden, könnte das Programm nicht fortgeführt werden, weil ständig Sprünge zwischen den Adressen 02 und 05 ausgeführt werden.

## Etwas für Mathematiker:

### Die Basis der Zahlen-Systeme

Nachdem der Computer hexadezimal rechnet, für uns jedoch Rechnungen im dezimalen Zahlen-System angenehmer sind, sollten einige Gedanken über die verschiedenen Zahlen-Systeme nicht uninteressant sein.

Wie bekannt, hat unser Dezimal-System die 10 Ziffern 0-1-2-3-4-5-6-7-8-9. Jede Zahl dieses Zahlen-Systems lässt sich durch ein Vielfaches einer Zehnerpotenz ausdrücken.



Programm-Start mit HALT – NEXT – 00 – RUN.

Auf dem Display werden 2 Stellen angezeigt. Wir können den angezeigten Wert auf dem Lottoschein eintragen. Sobald wir eine Zahlen- oder Buchstaben-Taste betätigen (egal welche), erscheint die nächste Zufallszahl.

Eine Zehnerpotenz ist z. B.  $10^2$  d. h.,  $10 \times 10 = 100$ . Oder  $10^3$  d. h.,  $10 \times 10 \times 10 = 1.000$ . Oder  $10^1 = 10$ . oder  $10^0 = 1$ .

So gesehen ist z. B. die Zahl **126** eine Abkürzung für:  
 $1 \times 10^2 + 2 \times 10^1 + 6 \times 10^0$  oder  $(1 \times 100 + 2 \times 10 + 6 \times 1)$ .

**Warum einfach, wenn es auch umständlich geht?**  
Wir sollten feststellen, daß **10** die Basis des dezimalen Zahlen-Systems darstellt.

Das hexadezimale System ist ähnlich aufgebaut, hat jedoch 16 Ziffern (0 bis F). Die **Basis des hexadezimalen Systems** ist **16**, weil jede hexadezimale Zahl durch ein Vielfaches von 16 dargestellt werden kann. Nehmen wir z. B. eine 3-stellige hexadezimale Zahl **7F3**.

$$7F3 = 7 \times 16^2 + F \times 16^1 + 3 \times 16^0$$

Durch diese Zerlegung läßt sich auch jede hexadezimale Zahl in das Dezimal-System umrechnen. Wie hoch ist der hexadezimale Wert **7F3** dezimal ausgedrückt?

$$7F3 = 7 \times 16^2 + 15 \times 16^1 + 3 \times 16^0 = 7 \times 16 \times 16 + 15 \times 16 + 3 \times 1 = 2035$$

↓  
(15 = F)

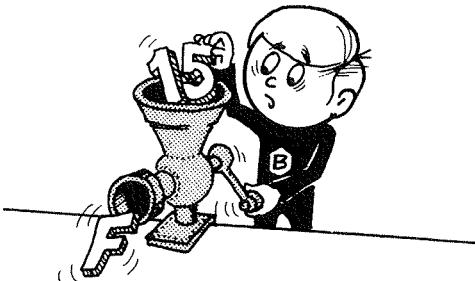
Ein anderes Beispiel:  $FE = 15 \times 16 + 14 \times 1 = 254$ .

Der Computer führt grundsätzlich intern alle Berechnungen hexadezimal durch, wie wir es bei den einleitenden Experimenten mit der Addition gesehen haben.

Für uns ist diese Arbeitsweise ungewohnt und so haben wir z. B. bei der 6-stelligen Addition die Ergebnisse sofort bei der Berechnung durch einen Programm-Trick in einen dezimalen Wert umgewandelt. Wir werden jedoch später feststellen, daß in vielen Programmen Berechnungen auszuführen sind, deren Ergebnisse nicht angezeigt werden, sondern als Grundlage für Schaltvorgänge oder für andere Berechnungen dienen. Deshalb ist es zweckmäßig, den Computer alle Berechnungen in seinem hexadezimalen System ausführen zu lassen und nur das letzte Ergebnis in unser Dezimal-System umzuwandeln.

Wie wir in folgendem Kapitel sehen, kann eine solche Umwandlung durch ein im Computer fest integriertes Programm durchgeführt werden.

## Umwandlung: Dezimal- in Hexadezimal-System



Auch für die Umwandlung einer **dezimalen Zahl** in eine **hexadezimale Zahl** hat der Microtronic-Computer einen Befehl **DZHX**. Der Befehls-Code ist **F04**.

Um diesen Befehl auszuprobieren, gehen wir mit HALT – NEXT – 05 zur Adresse 05 (im zuletzt eingegebenen Programm) und geben anstelle des dort stehenden Befehls F03 den neuen Befehl **F04** ein – Taste NEXT, alsdann das Programm wieder mit HALT – NEXT – 00 – RUN starten.

Das Display zeigt 000, wir geben ein 0-1-5. Der dezimale Wert 15 wird im Display mit F angezeigt. Die Eingabe der dezimalen 100 bringt das hexadezimale Ergebnis 64.

Logisch, daß wir bei dieser Programm-Konstellation nur dezimale Zahlen eingeben, um hexadezimale Ergebnisse zu erfahren. Würden wir versehentlich Buchstaben eingeben, führt der Computer ebenfalls eine Umwandlung durch, wobei dieses Ergebnis dann jedoch Unsinn ist.

Mitunter ist es schwierig, dezimale oder hexadezimale Ergebnisse auseinanderzuhalten. Zuvor haben wir ja gesehen, daß ein dezimales 100 ein hexadezimales 64 ergibt – wobei diese 64 genausogut ein dezimales Ergebnis sein könnte.

Es kann u. U. notwendig sein, die Basis eines Zahlen-Systems anzugeben. Dies würde dann wie folgt durchgeführt:

$$100_{10} = 64_{16}$$

(Dez.)      (Hex.)

## Umwandlung: Hexadezimal – in das Dezimal-System

Durch den neuen Befehl **HxDZ** wird eine **hexadezimale Zahl** in den Registern D, E und F in einen **dezimalen Wert** umgewandelt. Das Ergebnis steht nach der Umwandlung ebenfalls in den Registern D, E und F. Der Befehls-Code für **HxDZ** ist **F03**.

Nach HALT – NEXT – 00 geben wir folgendes Programm ein:

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
00	<b>F08</b>	CLEAR	alle Register löschen
01	<b>F3D</b>	DISP 3,D	Register D – Register F anzeigen
02	<b>FFF</b>	KIN F	Eingabe in Register F
03	<b>FFE</b>	KIN E	Eingabe in Register E
04	<b>FFD</b>	KIN D	Eingabe in Register D
05	<b>F03</b>	<b>HxDZ</b>	Hexadezimale Umwandlung
06	<b>C01</b>	GOTO 01	Sprung nach 01

Programm-Start: HALT – NEXT – 00 – RUN.

Das Display zeigt 000. Zuerst wollen wir den hexadezimalen Wert E umwandeln, und wir geben ein: 00E. Das umgewandelte Ergebnis = 14. Alsdann geben wir OFF ein = 255. Die höchste, von unserem Computer umwandelbare hexadezimale Zahl ist: 3E7 = 999.

Würden wir einen größeren Wert eingeben, reichen die 3 Stellen zur Umwandlung nicht mehr aus. In diesem Fall bringt der Computer das Ergebnis 000 und das ZERO-FLAG (untere LED neben dem Display) leuchtet auf als Signal für eine Bereichs-Überschreitung.

## Das Microtronic-Betriebs-System – ein unsichtbarer Helfer!

Im Kapitel „Software – Hardware“ haben wir von der enormen Arbeitsleistung des Mikroprozessors erfahren.

Der Mikroprozessor ergibt zusammen mit einem Speicher, einer Tastatur und einer Anzeige sowie verschiedenen weiteren elektronischen Bauelementen einen Mikrocomputer. Ein solcher Mikrocomputer kann jedoch keinerlei Aufgaben übernehmen, solange ihm noch ein Betriebs-System fehlt. Ein solches Betriebs-System muß dem Mikroprozessor in Form eines fest integrierten Programmes „mitgegeben“ werden.

Der im Microtronic-Computer-System verwendete Mikroprozessor mußte „vorprogrammiert“ werden, damit er als Computer arbeiten kann. Erst durch seine Vorprogrammierung kann er unsere Eingabe-Befehle verstehen und bearbeiten.

Der gleiche Mikroprozessor-Typ, kann auch für viele andere Anwendungsmöglichkeiten eingesetzt werden. Er benötigt dann jedoch eine völlig andere Vorprogrammierung als in unserem Fall.

So kann z. B. ein Mikroprozessor auch eine Waschmaschine steuern. Mit einer entsprechenden Vorprogrammierung wird er für seine zukünftige Aufgabe vorbereitet: Durch die Bedienungs-Tasten der Waschmaschine erfährt er, ob Bunt- oder Kochwäsche gewaschen werden soll. Er muß z. B. kontrollieren, ob das Wasser auf die richtige Temperatur aufgeheizt wurde. Er muß den Motor steuern – die Waschtrommel dreht sich langsam nach links – anschließend nach rechts. Er muß wissen, wann Waschpulver oder Weichspüler zugeführt wird und am Ende muß die Wäsche geschleudert werden.

Auch in vielen Autos ist neuerdings ein Bord-Computer vorhanden. Dieser arbeitet mit einer völlig anderen Programmie-

rung. Er erfährt durch entsprechende „Fühler“ wieviel Benzin im Tank enthalten ist, bei welchem Kilometerstand die Reise angetreten wurde, welche Geschwindigkeit das Tachometer anzeigt usw. Während der Fahrt kann der Computer Auskunft darüber geben, wie hoch bei der momentanen Geschwindigkeit der Kraftstoff-Verbrauch ist, wie lange die Reise bereits dauert und wieviel Kilometer inzwischen gefahren wurden und errechnet die Zeit aus, die bei der bisherigen Durchschnittsgeschwindigkeit notwendig ist, bis das voraussichtliche Ziel erreicht wird.

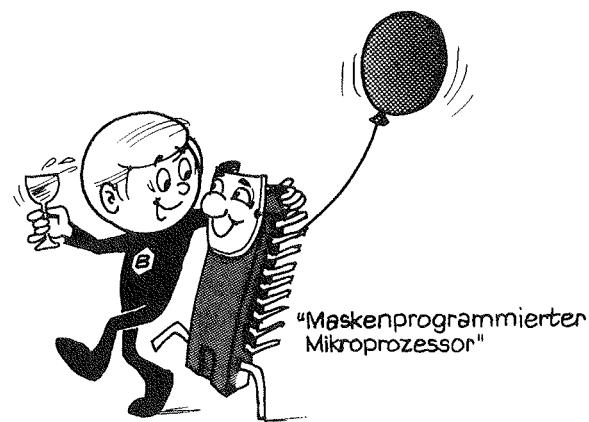
Beim Einsatz für das Microtronic-Computer-System hat der Mikroprozessor völlig andere Aufgaben.

Trotzdem haben alle diese Mikroprozessoren den gleichen inneren Aufbau. Sie haben eine sogenannte Rechenwerk, eine Anzahl Register, einen Programm-Schrittzähler, eine zentrale Ablaufsteuerung sowie Ein- und Ausgabekanäle. Außerdem hat er einen sogenannten **ROM-Speicher**, was aus dem englischen „read only memory“ abgeleitet mit „nur aus dem Speicher lesen“ übersetzt werden könnte. Der Mikroprozessor kann somit aus diesem ROM-Speicher nur Daten herausnehmen (herauslesen), er kann jedoch keine Daten „hineinschreiben“.

Dieser ROM-Speicher wird vom Hersteller des Mikroprozessors mit Daten vollgestopft, d. h., es werden ihm feste Programme eingegeben, die auch dann nicht verloren gehen, wenn z. B. die Spannungsversorgung unterbrochen wird.

Der ROM-Speicher enthält das „Betriebs-System“ des Mikroprozessors, welches auch als **Monitor-Programm** bezeichnet wird. Wir sehen, daß das Betriebs-Programm die speziellen Eigenschaften eines Mikroprozessors beinhaltet.

großer Rechenanlagen entwickelt, getestet, verbessert und immer wieder neu getestet. Wenn wir später selbst längere Programme entwickeln, werden wir feststellen, daß eine kleine Verbesserung an irgend einer Programm-Stelle oftmals größere Fehler an einer anderen Programm-Stelle mit sich bringen.



Arbeitet endlich das Monitor-Programm einwandfrei, wird die Vorprogrammierung des Mikroprozessors vorbereitet. Hierfür wird eine sogenannte „Maske“ erstellt – ein großer Film, in welchem alle Halbleiter-Strukturen des späteren ROM-Speichers innerhalb des Mikroprozessors enthalten sind. Durch die Maske werden die im ROM-Speicher stehenden Befehle festgelegt. Mit der Maske werden die Halbleiter-Stoffe (aus welchen das Innere des Mikroprozessors besteht) durch ein langwieriges physikalisches Verfahren (Diffusion) behandelt. Alle Eigenschaften unseres Mikroprozessors sind in einem winzigen ca. 5 x 5 mm großen Halbleiter-Chip enthalten.

Von der Idee, einen Experimentier- und Lern-Computer herzustellen, bis zum funktionsfertigen Endprodukt ist es ein langer Weg. Der Entwicklungszeitraum für ein solches Gerät beträgt ca. 2 Jahre. Vielleicht ahnen wir, wieviel Arbeit und Kosten in diesem unscheinbaren schwarzen Mikroprozessor stecken.

## Was bewirkt das Monitor-Programm unseres Mikroprozessors?

Durch das Monitor-Programm wird unser Mikroprozessor so gesteuert, daß er z. B. die 6-stellige Leuchtanzeige ansteuert und dort Ergebnisse anzeigt. Oder, daß er ständig die Tastatur abfragt, ob eine Taste betätigt wurde. Wird z. B. die NEXT-Taste betätigt, bewirkt das Betriebs-System, daß der Mikroprozessor in einem anderen Programm-Teil die eingegebenen Befehle abspeichert und den nächsten Befehl aus dem Speicher holt. Auch die mit unserem Computer abrufbaren Fest-Programme wie z. B. Prüf-Programm, Uhrzeit, Nimm-Spiel usw., sind ein Teil des Betriebs-Systems. Wurde z. B. die Uhrzeit einmal eingestellt (durch PGM 3), dann wird diese Uhrzeit durch das Betriebs-System gesteuert – auch wenn sie nicht angezeigt wird.

Im Microtronic Betriebs-System sind auch die Fest-Programme enthalten mit denen z. B. die vorher beschriebene Umwandlung der Zahlen-Systeme vorgenommen werden. Jeden Befehl, den wir bisher kennengelernt haben wie z. B. CLEAR, ADDI, CMPI usw. oder auch die durch Funktions-Tasten ausgeführten Befehle wie HALT, NEXT, BKP usw., lösen jeweils die im Betriebs-System enthaltenen Fest-Programme aus. Wenn wir z. B. durch den Befehl GOTO 00 den Befehls-Code C00 eingegeben haben, stellen diese drei eingegebenen Werte C00 für den Computer nicht etwa drei Einzel-Befehle dar, sondern es wird hierdurch ein vorprogrammierter Befehls-Satz ausgelöst. Hierdurch weiß der Computer, daß er z. B. mit C00 an den Programm-Anfang zurückspringen soll. Wir werden noch weitere Befehle zum Eingriff in das Betriebs-System kennenlernen.

Die Entwicklung eines derartigen Betriebs-Systems ist eine komplizierte und langwierige Angelegenheit. Zunächst müssen alle Einzelheiten festgelegt werden, welche der Mikroprozessor später ausführen soll. Alsdann werden die Aufgaben festgelegt, wie der Mikroprozessor die Befehle ausführen soll. Dieses zunächst theoretische Betriebs-System wird mit Hilfe

## TIME – der Zeit-Befehl!

Wir wissen, daß die (mit PGM 3) eingestellte Uhrzeit im Innern des Computers ständig weiterläuft – auch wenn sie nicht angezeigt wird. Die Uhrzeit können wir jederzeit durch den TIME-Befehl in die 6 Register A bis F übernehmen und in einem anderen Programm weiterverarbeiten.

Bevor wir das nachstehende Programm eingeben, sollten wir prüfen, ob die Verbindungsleitung TAKT/CLOCK zum EINGANG 4 auf der Computer-Platine noch besteht. Alsdann geben wir mit HALT – PGM – 3 die Uhrzeit ein (Stunden und Minuten). Dann mit HALT – NEXT – 00 an den Programm-Anfang und die Befehle lt. folgender Tabelle eingeben:

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
00	F06	TIME	Uhrzeit in Register A – F übernehmen
01	F6A	DISP 6,A	Register A – F anzeigen
02	C00	GOTO 00	Rücksprung

Das Display zeigt uns jetzt die laufende Uhrzeit an.

Durch den TIME-Befehl auf Adresse 00 haben wir das gleiche erreicht, was wir bisher mit PGM 4 erreichen konnten.

Wir sollten beachten, daß die Uhrzeit nur dann übernommen wird, wenn unser Programm zum TIME-Befehl kommt. Würden wir den Rücksprung-Befehl GOTO 00 (Befehls-Code C00) ändern (z. B. C01) wird die Uhrzeit nur ein einziges Mal beim

Programm-Start übernommen – eine Zeit-Berichtigung könnte nicht mehr erfolgen. Logisch: Nachdem das Programm nicht mehr zur Adresse 00 kommt, werden auch die sich jede Sekunde ändernden Zeit-Befehle nicht mehr in die Register A bis F übernommen – die Zeit bleibt stehen.

Probieren wir es aus. Mit HALT – NEXT – 02 sind wir auf der Rücksprung-Adresse. Der dort befindliche Rücksprung-Befehl C00 wird geändert in C01. Dann NEXT-Taste (damit der geänderte Befehl zum Programm-Speicher übertragen wird) und neuer Programm-Start mit HALT – NEXT – 00 – RUN. Es wird nur noch der Zeitpunkt der Programm-Änderung angezeigt, obwohl die Uhr intern weiterläuft.

Das vorstehende Programm soll erweitert werden. Wir möchten erreichen, daß der (bei GND und AUSGANG 1) angeschlossene Piezo-Summer zu jeder vollen Minute eine Sekunde lang summt. Hierfür müssen wir wissen, wie die Zeitwerte in den einzelnen Registern aufgeteilt sind:

Register:	F 10-er Stunden	E Stunden	D 10-er Minuten	C Minuten	B 10-er Sekunden	A Sekunden
-----------	-----------------------	--------------	-----------------------	--------------	------------------------	---------------

Wenn der Summer jede Minute ausgelöst werden soll, ist es zweckmäßig, die Auslösung immer dann vorzunehmen, wenn die Sekundenstelle (Register A) und die 10-er Sekundenstelle (Register B) den Wert 0 erreichen. Außerdem soll der Summtion nur solange anhalten (1 Sekunde), wenn in den Registern B und A gleichzeitig der Zeitwert 0 vorhanden ist.

Nach HALT – NEXT – 00 folgendes Programm eingeben:

	Eingabe-Befehl	Mnemonic	Erklärungen
00	<b>100</b>	MOVI 0,0	Wert 0 nach Register 0 schieben
01	<b>1F1</b>	MOVI F,1	Wert F nach Register 1 schieben
02	<b>F6A</b>	DISP 6,A	Register A bis F anzeigen
03	<b>F06</b>	TIME	Uhrzeit in Register A – F laden
04	<b>90A</b>	CMPI 0,A	Register A = 0? (Sekunden)
05	<b>E08</b>	BRZ 08	Ja, Sprung nach 08
06	<b>FE0</b>	DOT 0	Nein, dann Ausgänge abschalten
07	<b>C03</b>	GOTO 03	Rücksprung nach 03
08	<b>90B</b>	CMPI 0,B	Register B = 0? (10 Sekunden)
09	<b>D06</b>	BRC 0	Nein, dann Sprung nach 06
0A	<b>FE1</b>	DOT 1	Ja, Ausgänge einschalten
0B	<b>C03</b>	GOTO 03	Rücksprung nach 03

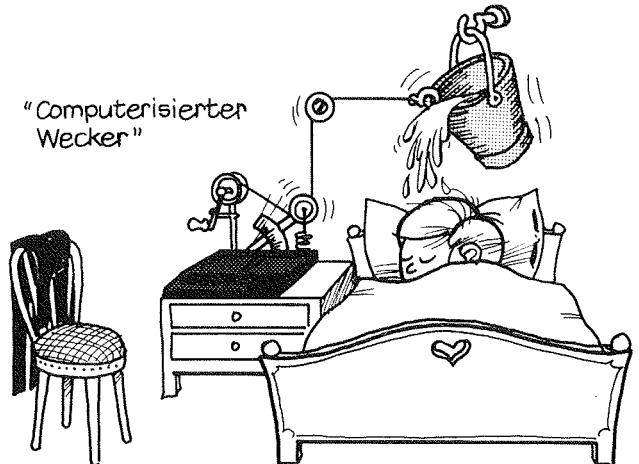
Falls wir den Piezo-Summer während des Programmierens abgestellt haben, sollte er jetzt wieder angeschlossen sein. Programm-Start: HALT – NEXT – 00 – RUN.

Das Display zeigt uns die genaue Uhrzeit mit laufenden Sekunden. CARRY-FLAG und ZERO-FLAG blinken in schnellem Wechsel. Sobald bei der Sekunden-Anzeige der Wert 59 nach 00 übersprungen wird, bringt der Summer ein kurzes Signal.

#### Programm-Beschreibung:

Wenn wir uns die Programm-Tabelle ansehen, müßten wir eigentlich die notwendigen Erklärungen selbst finden. Interessant ist die Funktionsweise der Vergleichs-Befehle und der DOT-Befehle. Bei Adresse 04 wird gefragt, ob in Register A (Sekunden) der Wert 0 erreicht wurde. Wenn ja, erfolgt der Sprung zur Adresse 08, wobei dort verglichen wird, ob auch in Register B der Wert 0 vorhanden ist. Ist dies nicht der Fall sorgt der DOT-Befehl dafür, daß die Ausgänge (und damit auch der Summer) abgeschaltet sind. Erst wenn bei Adresse 08 auch im Register B der Wert 0 vorhanden ist, wird über die Adresse 0A der Befehl DOT 1 ausgeführt. Die Ausgänge und damit auch der Summer schalten sich ein.

#### Der Computer als Weck-Uhr



Bei diesem Experiment werden wir wieder den TIME-Befehl in ein neu einzugebendes Programm integrieren, um die intern im Computer mitlaufende Uhrzeit in unseren Programm-Ablauf einzubeziehen.

Vorsorglich soll noch einmal darauf hingewiesen werden, daß TAKT/CLOCK mit EINGANG 4 auf der Computer-Platine verbunden sein muß.

Falls die Uhrzeit noch nicht eingegeben wurde, bitte mit HALT – PGM – 3 eingeben. Alsdann HALT und das folgende Programm wie üblich nach HALT – NEXT – 00 eingeben.

	Eingabe-Code	Mnemonic	Erklärungen
00	<b>F08</b>	CLEAR	Alle Register löschen
01	<b>FE5</b>	DOT 5	Ausgänge abschalten, da Register 5 durch „CLEAR“ 0 ist
02	<b>F41</b>	DISP 4,1	Reg. 1 – Reg. 4 anzeigen
03	<b>FF0</b>	KIN 0	Eingabe in Reg. 0
04	<b>990</b>	CMPI 9,0	Reg. 0 größer als 9?
05	<b>D0B</b>	BRC 0B	Ja, Sprung nach 0B
06	<b>034</b>	MOV 3,4	Andernfalls die Zahlen weiterschieben (siehe auch Programm „6-stellige Addition“)
07	<b>023</b>	MOV 2,3	
08	<b>012</b>	MOV 1,2	
09	<b>001</b>	MOV 0,1	
0A	<b>C03</b>	GOTO 03	
0B	<b>F6A</b>	DISP 6,A	Reg. A – Reg. F anzeigen
0C	<b>1F6</b>	MOVI F,6	F in Reg. 6 speichern
0D	<b>F06</b>	TIME	Uhrzeit in Reg. A – Reg. F laden
0E	<b>84F</b>	CMP 4,F	Reg. 4 = Reg. F?
0F	<b>E11</b>	BRZ 11	Ja, Sprung nach 11
10	<b>C19</b>	GOTO 19	Nein, Sprung nach 19
11	<b>83E</b>	CMP 3,E	
12	<b>E14</b>	BRZ 14	
13	<b>C19</b>	GOTO 19	
14	<b>82D</b>	CMP 2,D	
15	<b>E17</b>	BRZ 17	
16	<b>C19</b>	GOTO 19	
17	<b>81C</b>	CMP 1,C	
18	<b>E1B</b>	BRZ 1B	
19	<b>FE5</b>	DOT 5	Ausgänge abschalten
1A	<b>C0D</b>	GOTO 0D	(Reg. 5 = 0 durch CLEAR)
1B	<b>FE6</b>	DOT 6	Ausgänge einschalten
1C	<b>C0D</b>	GOTO 0D	

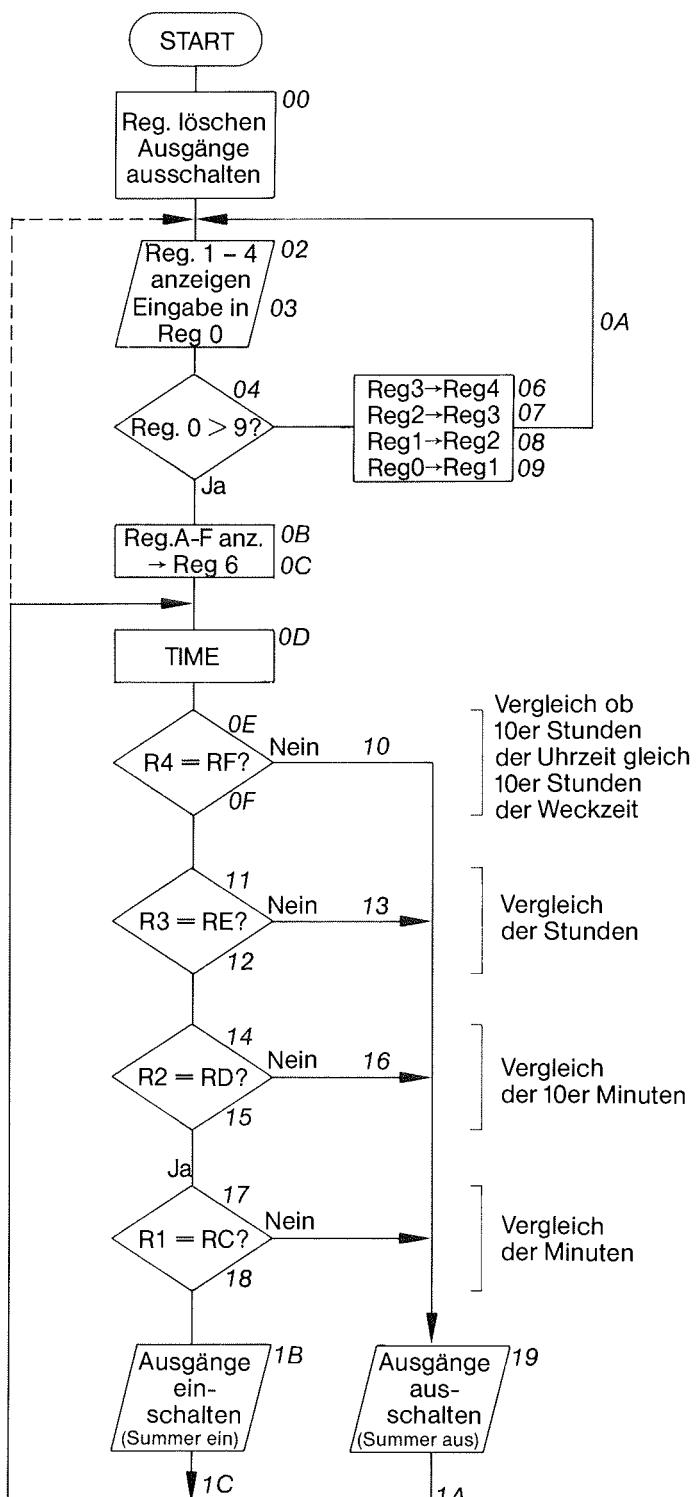
Nicht vergessen, den Piezo-Summer anzuschließen.

Programm-Start: HALT – NEXT – 00 – RUN. Das Display zeigt 0000. Wir geben jetzt Stunden und Minuten als Weck-Zeit ein.

Damit sich keine zu lange Wartezeit ergibt, nur wenig später als momentane Zeit eingeben. Beispiel: Es ist jetzt 9.45 Uhr, also geben wir ein 9.48 Uhr (0948). Taste A – der Computer zeigt die genaue Uhrzeit einschließlich Sekunden. Sobald die eingegebene Weck-Zeit erreicht ist, ertönt der Summer, welcher sich nach einer Minute automatisch wieder abschaltet.

Wenn wir an diesem Programm nichts ändern, wird sich der Computer täglich um die gleiche Zeit wieder melden. Wir können jedoch auch mit HALT – NEXT – 00 – RUN das Programm neu starten und eine andere Zeit eingeben.

Eine zweite interessante Programm-Variation ergibt sich, wenn wir den letzten Befehl unter Adresse 1C, welcher mit dem Befehls-Code C0D (also Rücksprung zu Adresse 0D eingeben war) in C02 (Rücksprung zur Programm-Adresse 02) ändern. Wir probieren es aus: HALT – NEXT – 1C, den geänderten Befehl C02 eingeben, NEXT-Taste betätigen und mit HALT – NEXT – 00 zum Programm-Anfang zurückgehen. Starten mit RUN – neue Weck-Zeit eingeben. Taste A betätigen.



Der Computer zeigt wieder die genaue Uhrzeit einschließlich Sekunden. Sobald die eingestellte Weck-Zeit erreicht wird, meldet sich der Summer. Das Display zeigt jetzt nur noch die eingegebenen Weck-Zeit mit Stunden und Minuten. Dieser Zustand bleibt solange erhalten, bis wir eine neue Weck-Zeit eingeben (wobei die bisher angezeigte Zeit gelöscht wird). Dann Taste A betätigen. Der Summer wird abgeschaltet und die laufende Uhrzeit einschließlich Sekunden wieder angezeigt.

Soll die eingestellte Zeit nicht geändert werden, müssen wir eine Minute warten. Sobald wir dann Taste A betätigen, wird wieder die laufende Uhrzeit einschließlich Sekunden angezeigt und der Computer meldet sich am nächsten Tag zur gleichen Zeit.

Wenn wir die Programm-Tabelle und den Programm-Ablaufplan (bei welchem wieder die Adressen jeweils rechts neben den Symbolen angegeben sind) vergleichen, sollten uns die Funktionen dieses Programmes keine Schwierigkeiten bereiten.

Wenn wir unseren Computer im Dauerbetrieb weiterlaufen lassen möchten, können wir das Weck-Uhren-Programm z. B. auch auf die letzten Adressen des Programm-Speichers bringen. Hierdurch hätten wir die Möglichkeit bei Adresse 00 beginnend, weitere Programme einzugeben und nebenher den Computer als Weck-Uhr einzusetzen. Auf welcher Adresse müßten wir in diesem Fall mit der Programm-Eingabe beginnen?

Wir vergleichen mit der Tabelle „Gegenüberstellung dezimales/hexadezimales Zahlen-System“ auf Seite 17, bzw. auf dem Buch-Lesezeichen. Bei der letzten Dezimal-Zahl 255 steht hexadezimal FF. FF ist die letzte belegbare Programm-Adresse. 29 Schritte rückwärts gezählt, kämen wir auf die E3. Zweckmäßigerweise beginnen wir jedoch bei E0. Können wir das gleiche Programm statt auf 00 beginnend auch bei der Adresse E0 beginnend eingeben?

Wenn wir uns die Programm-Tabelle ansehen, sollten wir eigentlich schnell zu der Erkenntnis kommen, daß durch die vielen Sprung-Befehle eine Programm-Änderung notwendig wird.

Den ersten Sprung-Befehl haben wir auf der bisherigen Adresse 05. Durch den Befehls-Code D0B sind wir bisher zur Adresse 0B gesprungen. Würden wir diesen Befehls-Code belassen, „springen wir weit aus unserem Programm heraus“. Welche Überlegungen sind notwendig? Welche Befehls-Codes müssen geändert werden?

Das gesamte Programm ist neu zu überdenken. Hierfür hilft nachstehende Tabelle mit Angabe der bisherigen Adressen-Nummern und der zukünftigen neuen Adressen, beginnend bei E0 bis FC. Bei einigen Adressen ist ein \* angegeben. Bei diesen Adressen sind Überlegungen notwendig, ob der bisherige Befehls-Code unverändert übernommen werden kann?

Bevor wir mit der Programmierung beginnen, schreiben wir das geänderte Programm in die Tabelle. Bei den neuen Adressen E0 bis einschließlich E4 können wir die unveränderten Eingabe-Befehle übernehmen (E0 = F08 usw.). Bei der Adresse E5 kommt ein Sprung-Befehl, welcher in unserem bisherigen Programm mit D0B stand, (da ein Sprung nach Adresse 0B auszuführen war). Die alte Adresse 0B heißt für unsere neue Programmierung EB. Logischerweise muß der bisherige Befehls-Code D0B in DEB geändert werden. Die nächste Überlegungsbedürftige neue Adresse ist EA. Was ist zu ändern?

## Programm-Änderung: Computer-Weck-Uhr

Bisherige Adresse	Neue Adresse	Befehls-Eingabe	Mnemonic	Erklärungen
00	E0			
01	E1			
02	E2			
03	E3			
04	E4			
05	*E5			
06	E6			
07	E7			
08	E8			
09	E9			
0A	*EA			
0B	EB			
0C	EC			
0D	ED			
0E	EE			
0F	*EF			
10	*F0			
11	F1			
12	*F2			
13	*F3			
14	F4			
15	*F5			
16	*F6			
17	F7			
18	*F8			
19	F9			
1A	*FA			
1B	FB			
1C	*FC			

Nachdem wir unser Programm und alle Sprung-Befehle unter Berücksichtigung der neuen Adressen umgeschrieben haben, wollen wir die Eingabe vornehmen: HALT – NEXT – **E0** (anstelle 00 wie sonst üblich)!

Nach Programm-Eingabe starten wir mit: HALT – NEXT – E0 – RUN. Das Display zeigt jetzt wieder 0000. Weck-Zeit eingeben – Taste A – der Computer wird sich nach der eingestellten Zeit melden. Wir haben jetzt die zuerst beschriebene Weck-Uhr Variation. Auch hier können wir wieder bei der letzten Adresse den Befehls-Code (der jetzt mit CED vorhanden ist) ändern auf CE2, um die beschriebene zweite Weck-Uhr Variation zu haben.

Sollte das Programm nicht einwandfrei laufen, haben wir beim Programmieren einen Fehler gemacht. Wir springen zum Programm-Anfang mit HALT – NEXT – E0 und vergleichen durch Betätigung der NEXT-Taste, ob die eingegebenen Befehle mit der nachfolgenden Tabelle übereinstimmen. Bei evtl. Fehlern den geänderten Befehl eingeben – nicht vergessen die NEXT-Taste zu betätigen.

## Berichtigtes Programm: Computer-Weck-Uhr

Neue Adresse	Eingabe Code	Mnemonic	Erklärungen
E0	<b>F08</b>	CLEAR	Alle Register löschen
E1	<b>FE5</b>	DOT 5	Ausgänge abschalten, (da Register 5 durch „CLEAR“ 0 ist)
E2	<b>F41</b>	DISP 4,1	Reg. 1 – Reg. 4 anzeigen
E3	<b>FF0</b>	KIN 0	Eingabe in Reg. 0
E4	<b>990</b>	CMPI 9,0	Reg. 0 größer als 9?
*E5	<b>DEB</b>	BRC EB	Ja, Sprung nach EB
E6	<b>034</b>	MOV 3,4	Andernfalls die Zahlen weiterschieben (siehe auch Programm „6-stellige Addition“)
E7	<b>023</b>	MOV 2,3	
E8	<b>012</b>	MOV 1,2	
E9	<b>001</b>	MOV 0,1	
*EA	<b>CE3</b>	GOTO E3	
EB	<b>F6A</b>	DISP 6,A	Reg. A – Reg. F anzeigen
EC	<b>1F6</b>	MOVI F,6	F in Reg. 6 speichern
ED	<b>F06</b>	TIME	Uhrzeit in Reg. A – Reg. F laden
EE	<b>84F</b>	CMP 4,F	Reg. 4 = Reg. F?
*EF	<b>EF1</b>	BRZ F1	Ja, Sprung nach F1
*F0	<b>CF9</b>	GOTO F9	Nein, Sprung nach F9
F1	<b>83E</b>	CMP 3,E	
*F2	<b>EF4</b>	BRZ F4	
*F3	<b>CF9</b>	GOTO F9	
F4	<b>82D</b>	CMP 2,D	
*F5	<b>EF7</b>	BRZ F7	
*F6	<b>CF9</b>	GOTO F9	
F7	<b>81C</b>	CMP 1,C	
*F8	<b>EFB</b>	BRZ FB	
F9	<b>FE5</b>	DOT 5	Ausgänge abschalten (Reg. 5 = 0 durch CLEAR)
*FA	<b>CED</b>	GOTO ED	
FB	<b>FE6</b>	DOT 6	Ausgänge einschalten
*FC	<b>CED</b>	GOTO ED	(oder bei 2. Variation: GOTO E2)

## Immer neue Möglichkeiten mit 0 und 1

Es muß noch einmal erwähnt werden, daß alle Computer nur mit den beiden Zuständen „Spannung vorhanden“ = Wert 1 oder „keine Spannung vorhanden“ = Wert 0 arbeiten.

Um bei der vorangegangenen Weck-Uhr den Piezo-Summer zum Tönen zu bringen war es notwendig, die Ausgänge zu „aktivieren“. Hierfür haben wir bei Adresse 0C den Wert F in einem Register gespeichert (mit MOVI F,6 wurde der Wert F in das Register 6 geschoben) und wir haben diesen Wert F am Programm-Ende (bei Adresse 1B) mit dem DOT-Befehl auf die Ausgänge geschaltet. Der an einem der Ausgänge angeschlossene Piezo-Summer wurde zum Tönen gebracht.

Später möchten wir eine Schalt-Uhr programmieren, die gleichzeitig oder nacheinander verschiedenartige Geräte ansteuert. Dazu ist es notwendig, daß wir nicht alle Ausgänge gleichzeitig, sondern daß wir die 4 Ausgänge unabhängig von einander ansteuern (aktivieren) können. Hierfür gibt es verschiedene Befehle, die wir kennen und verstehen lernen sollten.

Mit folgendem Programm werden die 4 Ausgänge nacheinander angesteuert, wodurch die an jedem Ausgang vorhandenen LEDs nacheinander aufleuchten und sofort wieder verlöschen. Es entsteht der Eindruck eines sich bewegenden Lichtpunktes, weshalb man hierzu auch „Lauflicht“ sagt.

Nach HALT – NEXT – 00 geben wir folgendes Programm ein:  
**Programm Lauflicht**

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
00	<b>110</b>	MOVI 1,0	In Register 0 konstanten Wert 1 schieben
01	<b>F10</b>	DISP 1,0	Eine Stelle ab Register 0 anzeigen
02	<b>FE0</b>	DOT 0	Register 0 auf Ausgänge schalten
03	<b>FA0</b>	SHL 0	(dualen) Wert von Register 0 verdoppeln
04	<b>FB0</b>	ADC 0	Carry in Register 0 addieren
05	<b>C02</b>	GOTO 02	Sprung zu Adresse 02

Programm-Start: HALT – NEXT – 00 – RUN. Auf dem Display ergibt sich auf der rechten Stelle eine Anzeige, die jedoch so schnell erfolgt, daß die angezeigten Zahlen nicht erkennbar sind. Die 4 Leuchtdioden an den Ausgängen auf der Computer-Platine bringen den erwähnten Lauflicht-Effekt. Das Lauflicht ist jedoch lediglich ein Nebeneffekt.

Viel wichtiger ist die Tatsache, daß die 4 Ausgänge in schneller Folge nacheinander ein- und ausgeschaltet werden. Bei der vorangegangenen Weck-Uhr hatten jeweils alle 4 LED's geleuchtet, wenn der Piezo-Summer in Betrieb war. Wie können wir die 4 Ausgänge getrennt ansteuern?

Betrachten wir das Programm: Mit dem MOVI-Befehl schieben wir den Wert 1 in das Register 0. Mit DISP wird das Register 0 angezeigt. Mit DOT bringen wir den Wert 1 auf die Ausgänge. Nun erinnern wir uns nochmals an das Dual-System (siehe Tabelle „Zahlen-Systeme“ Buchlesezeichen). Der Wert 1 ergibt dual dargestellt 0001. Bei der zuvor programmierten Weck-Uhr haben wir den Wert F auf die Ausgänge gebracht, das ist dual 1111. Beim Computer bedeutet der Wert 1, daß eine Spannung vorhanden ist. Durch F(1111) haben wir viermal eine Spannung an die 4 Ausgänge gebracht – alle 4 LED's leuchten. Mit dem Wert 1 (0001) ist jedoch nur an einem Ausgang eine Spannung vorhanden, also kann nur eine LED z. B. am Ausgang 1 leuchten. Wie bringen wir die Spannung (also den Wert 1) zum nächsten Ausgang?

Hier hilft uns der neue **SHL**-Befehl (aus dem englischen „shift left“). Dieser Befehl verschiebt an den nach dem Dual-System arbeitenden Ausgängen den Wert 1 (Spannung vorhanden) um



eine Stelle nach links (von 0001 nach 0010). Die bisher am Ausgang 1 leuchtende LED wird abgeschaltet (Wert 0 = keine Spannung vorhanden), der Ausgang 2 hat jetzt den Wert 1 (Spannung vorhanden) – die zweite LED leuchtet.

Dieses nach links Verschieben (dualer Wert 0001 wird 0010) bedeutet, daß im Register 0 der bisherige Wert 1 durch den SHL-Befehl zum Wert 2 geworden ist (siehe Tabelle Dual-System). Durch diesen Vorgang wird kein CARRY-FLAG ausgelöst, somit wird der in Adresse 04 vorhandene ADC-Befehl übersprungen und bei Adresse 05 erfolgt der Rücksprung zur Adresse 02. Das Spiel beginnt von neuem, weil nach dem DOT-Befehl der SHL-Befehl den Wert im Register 0 von bisher 2 auf den Wert 4 verdoppelt. Dual ergibt dies 0100, d. h. an den Ausgängen ist die Spannung (1) wieder eine Stelle weiter nach links gewandert – die dritte LED am Ausgang 3 leuchtet. Beim nächsten Durchlauf entsteht durch den SHL-Befehl in Register 0 der Wert 8 = dual 1000. Die vierte LED wird aufleuchten, weil jetzt die Spannung am vierten Ausgang vorhanden ist.

Immer wenn in unserem Programm der SHL-Befehl durchlaufen wurde, hat sich der Wert in Register 0 verdoppelt. Dies ergibt sich auch beim nächsten Durchlauf, wodurch der bisherige Wert 8 verdoppelt wird. Nachdem  $2 \times 8 = 16$  ergibt, unser Computer jedoch hexadezimal arbeitet, wird der höchste hexadezimale Wert F (15) übersprungen. Das hexadezimale F + 1 ergibt den hexadezimalen Wert 10. Da wir in jedem Register nur einen 1-stelligen hexadezimalen Wert unterbringen, bleibt vom Wert 10 die 0 übrig und die nicht übertragbare 1 löst das CARRY-FLAG aus. Dieser Vorgang wird auf Adresse 04 vom ADC-Befehl registriert. Der noch verbleibende Übertrag Wert 1 wird in das Register 0 addiert, was ein duales 0001 ergibt.

Beim folgenden Programm-Durchlauf schiebt der SHL-Befehl wieder alle Stellen nach links.

## Die Funktions-Tasten REG und STEP

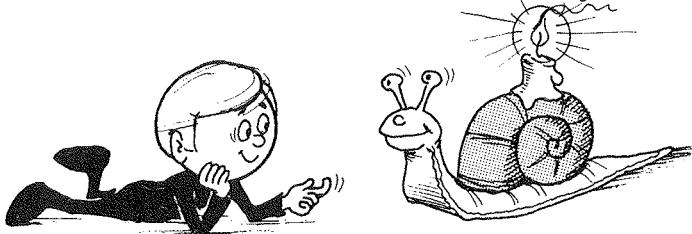
Die STEP-Taste haben wir bereits kennengelernt. Sie ist eine „Einzelschritt-Taste“, weil mit jeder Betätigung nur ein Programm-Schritt ausgeführt wird. Wir möchten STEP und die neue **REG**-Taste bei dem vorher programmierten Lauflicht ausprobieren.

Display-Anzeige	
Mit HALT – NEXT – 00 zeigt das Display:	01 110
STEP-Taste betätigen, der angezeigte Befehl wird ausgeführt und der nächste Befehl angezeigt:	01 F10
Betätigung STEP, der Befehl den Wert in Register 0 anzeigen, wird ausgeführt:	1
Der nächste Befehl DOT wird nicht angezeigt. Erneute STEP-Tasten Betätigung zeigt den nächsten Befehl:	03 FA0
Hierdurch wurde der DOT-Befehl ausgeführt, die LED am Ausgang Nr. 1 leuchtet. STEP bringt den nächsten Befehl:	04 FB0
Der SHL-Befehl wurde ausgeführt. In Register 0 müßte sich jetzt der Wert verdoppelt haben, d.h., es müßte der Wert 2 vorhanden sein? <b>Wir können dies mit der Reg-Taste kontrollieren.</b> Mit den Tasten HALT – REG – 0 – das Display zeigt: Die erste Ziffer ist die eingegebene Register Nr. 0, die Zweite der Register-Inhalt also 2.	0 2
Bevor wir mit STEP weitermachen zuerst HALT betätigen, der letzte Befehl wird wieder angezeigt:	04 FB0
Weitermachen mit STEP:	05 C02
Dieser letzte Befehl bedeutet Rücksprung zur Adresse 02 – also STEP-Taste betätigen:	02 FE0
STEP-Taste:	03 FA0
Soeben wurde der DOT-Befehl ausgeführt, die LED am Ausgang 1 ist aus und die LED am Ausgang 2 ist angegangen. Wieder STEP:	04 FB0
Hat der SHL-Befehl inzwischen den Register-Wert wieder verkoppelt? Wir kontrollieren mit HALT – REG – 0:	0 4
HALT – der letzte Befehl wird wieder angezeigt:	04 FB0
STEP-Taste:	05 C02

Mit der STEP-Taste können wir das Programm im Zeitlupentempo durcharbeiten. Mit der REG-Taste können wir uns den Register-Inhalt anzeigen lassen, wenn wir zu REG jeweils die Register Nr. (im letzten Beispiel 0) angeben.

Mit der REG-Taste können wir jedoch auch Register-Inhalte verändern. Mit HALT – REG – 03 können wir dies ausprobieren. Das Display zeigt: --0--3. Im Register 0 haben wir also jetzt den Wert 3 oder dual 0011. Mit HALT und mehrmaliger STEP-Tasten-Betätigung können wir jetzt sehen, daß an den Ausgängen jeweils 2 LED gleichzeitig leuchten, die nach der vierten STEP-Tasten-Betätigung um eine Stelle weitergeschoben werden.

## Selbst-Programmierung eines langsamen Lauflichtes



**Aufgabe:** Das vorangegangene Lauflicht-Programm soll so überarbeitet werden, daß die Laufgeschwindigkeit erheblich langsamer wird, damit auch die Anzeige auf dem Display verfolgt werden kann.

**Lösungsvorschläge:** Unsere Kenntnisse sind inzwischen so weit vorangeschritten, wodurch sich mehrere Möglichkeiten zur Realisierung der Aufgaben-Stellung anbieten wie z. B.

1. Wir könnten mit dem TAKT/CLOCK-Impuls in Verbindung mit einer Warteschleife die verminderte Geschwindigkeit erreichen (siehe auch TIMER-Programm).
2. Der Computer könnte intern zählen (von 0 bis F), wobei die Ausgangs-LED immer nur dann weitergeschaltet wird, wenn der Zähler einen Übertrag (CARRY-FLAG) ausführt.

Wir tragen unsere Programm-Vorstellungen in nachfolgende Tabelle ein:

Adresse	Befehls-Eingabe	Mnemonic	Erklärungen
00			
01			
02			
03			
04			
05			
06			
07			
08			
09			
0A			
0B			

Wenn das eingegebene Programm einwandfrei läuft, können wir mit den folgenden Lösungsvorschlägen vergleichen. Es kann ohne weiteres sein, daß ein einwandfrei funktionierendes Programm eingegeben wurde, welches mit unseren Lösungsvorschlägen nicht übereinstimmt, denn „viele Wege führen nach Rom“. Wir werden auch später noch feststellen, daß sich für manche Programm-Aufgabe mehrere Lösungen ergeben.

Falls wir uns für den Lösungsvorschlag 1 entschieden haben, muß wie üblich der Ausgang TAKT/CLOCK mit dem EINGANG 4 verbunden sein.

### Lösungsvorschlag Nr. 1

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
00	110	MOVI 1,0	
01	F10	DISP 1,0	
02	FD1	DIN 1	
03	812	CMP 1,2	
04	E02	BRZ 02	
05	012	MOV 1,2	Warteschleife, Erklärung siehe Timer (Synchronisation mit TAKT/CLOCK)
06	902	CMPI 0,2	
07	E02	BRZ 02	
08	FA0	SHL 0	
09	FB0	ADC 0	entspricht Programm „kleines Lauflicht“
0A	FE0	DOT 0	
0B	C02	GOTO 02	

### Lösungsvorschlag Nr. 2

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
00	110	MOVI 1,0	
01	F10	DISP 1,0	
02	511	ADDI 1,1	
03	D05	BRC 05	Zeitverzögerung durch Zählschleife
04	C01	GOTO 01	
05	FA0	SHL 0	
06	FB0	ADC 0	entspricht Programm „kleines Lauflicht“
07	FE0	DOT 0	
08	C01	GOTO 01	

### Durch Halbieren nach rechts verschieben?

Mit dem SHL-Befehl können wir einen Registerwert verdoppeln, wodurch die dualen Ausgänge beeinflußt (nach links verschoben) werden. Der Gegen-Befehl hierzu ist **SHR** (aus dem englischen *shift-right*), welcher die in einem Register vorhandenen Werte halbiert, wodurch ein duales nach rechts Verschieben erreicht wird.

Damit wir die Arbeitsweise des neuen Befehls auf dem Display verfolgen können, geben wir folgendes Programm ein:

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
00	F08	CLEAR	Alle Register löschen
01	F10	DISP 1,0	Register 0 anzeigen
02	FF0	KIN 0	Eingabe abwarten
03	F90	SHR 0	(dualen) Wert von Register 0 halbieren
04	C01	GOTO 01	Sprung zu Adresse 01

Wir starten das Programm. Das Display zeigt 0. Wir geben 8 ein, das Display zeigt den halbierten Wert 4. Bei der Eingabe 6 erscheint 3. Da wir nur mit einer Stelle (einem Register) arbeiten, können wir den 2-stelligen dezimalen Wert 14 nicht eingeben, dafür jedoch das hexadezimale E : Ergebnis 7.

Wenn wir 9 eingeben, müßte das Ergebnis 4,5 lauten. Dies ist jedoch mit einer Stellenanzeige nicht möglich, der Computer rundet ab und zeigt als Ergebnis 4.

Nachdem der SHR-Befehl genau entgegengesetzt wie der SHL-Befehl arbeitet, stellen wir fest: Geben wir 8 ein, haben wir wieder dual 1000. Durch die Halbierung ergibt sich 4, also dual 0100. Geben wir 5 ein (dual 0101) ergibt sich durch die Halbierung und Abrundung 2 (dual 0010). Vor der Halbierung waren beim Wert 5 (dual 0101) zweimal die Werte 1 vorhanden. Beim Halbieren und dem damit verbundenen dualen nach rechts schieben, geht die rechts stehende 1 „verloren“. Hierdurch wird das CARRY-FLAG gesetzt. Wir können dies bei allen nicht durch 2 teilbaren Zahleneingaben verfolgen.

Wenn wir 0 eingeben, leuchtet das ZERO-FLAG auf und bei der Eingabe 1 (was durch Halbierung ebenfalls 0 ergibt) leuchten CARRY- und ZERO-FLAG. Wir sollten dies zunächst einmal in unserem Gedächtnis speichern.

Vielelleicht wundern wir uns, daß die LED's an den Ausgängen auf die Eingaben nicht reagieren. Wundern wir uns wirklich?

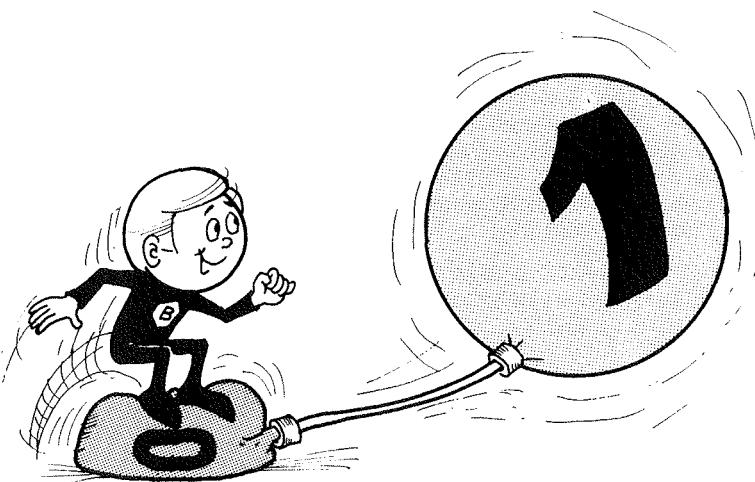
Da wir keinen DOT-Befehl in unserem kleinen Programm vorgesehen haben, werden die Ausgänge logischerweise auch nicht angesprochen. Wir können dies jedoch sehr leicht nachholen, indem wir mit HALT - NEXT - 04 dort den DOT-Befehl eingeben „FE0“. Nach NEXT geben wir den Rücksprung-Befehl jetzt bei Adresse 05 ein „C01“ und mit HALT - NEXT - 00 - RUN wieder starten: Bei Eingabe 2 erfolgt Halbierung = 1 (dual 0001) LED am Ausgang 1 leuchtet. Die Eingabe 4 bewirkt Halbierung = 2 (0010) LED Ausgang 2 leuchtet. Die Eingabe 6 bewirkt Halbierung = 3 (0011) die LED's an den Ausgängen 1 und 2 leuchten.

Am Ende des Anleitungsbuches finden wir eine Zusammenfassung aller Befehle. Dort wird auch beschrieben, wann das CARRY- und ZERO-FLAG gesetzt werden. Auch auf dem Buch-Sezeichen sind alle Befehle mit Kurzerklärung vorhanden, wodurch man für Programmiersversuche alle notwendigen Angaben zur Verfügung hat.

Übrigens, wenn wir das Weck-Uhren-Programm, welches wir (siehe Seite 46/47) unter Adresse E0 eingegeben haben, durch Herausziehen des Netzgerätes aus der Steckdose inzwischen noch nicht gelöscht haben, können wir uns jetzt mit HALT – NEXT – E0 – RUN die genaue Uhrzeit anzeigen lassen, bzw. eine neue Weck-Zeit eingeben.

Wir können dies mit allen Zahlen und Buchstaben (am besten der Reihe nach) durchprobieren und erhalten dann folgende Tabelle:

Eingegebener Wert	= Dualer Wert	Ausgegebener Wert	= Dualer Wert
0	0000	F	1111
1	0001	E	1110
2	0010	D	1101
3	0011	C	1100
4	0100	B	1011
5	0101	A	1010
6	0110	9	1001
7	0111	8	1000
8	1000	7	0111
9	1001	6	0110
A	1010	5	0101
0	1011	4	0100
C	1100	3	0011
D	1101	2	0010
E	1110	1	0001
F	1111	0	0000



## Aus 0 wird 1 – aus 1 wird 0!

Um langwierigen Erklärungen aus dem Wege zu gehen, geben wir nach HALT – NEXT – 00 folgendes Programm ein:

Adresse	Befehls-Eingabe	Mnemonic
00	<b>F08</b>	CLEAR
01	<b>F10</b>	DISP 1,0
02	<b>FF0</b>	KIN 0
03	<b>FE0</b>	DOT 0
04	<b>FF1</b>	KIN 1
05	<b>F80</b>	INV 0
06	<b>FE0</b>	DOT 0
07	<b>C02</b>	GOTO 02

Nach HALT – NEXT – 00 – RUN erscheint auf dem Display 0. Wir geben 2 ein. Dies wird auf dem Display angezeigt – die LED am Ausgang 2 leuchtet. Wir geben nochmals 2 ein – das Display zeigt D – die bisherige LED erlischt und die drei anderen LED's leuchten. Wir geben 6 ein – die zwei mittleren LED's leuchten. Wir wiederholen die Zifferneingabe 6 – die beiden äußeren LED's leuchten.

Nun geben wir 0 ein – alle LED's erlöschen – abermalige 0 Eingabe zeigt auf dem Display F – alle LED's leuchten.

Soeben geht uns ein Licht auf. Die Eingabe 0 entspricht dem dualen Wert 0000. Bei abermaliger 0 Eingabe wird F angezeigt = dual 1111. Wir haben also die dualen Werte umgekehrt.

In vorstehendem Programm kann bei Adresse 02 eine Zahl in das Register 0 eingegeben werden (z. B. 2). Durch den folgenden DOT-Befehl wird der eingegebene Wert an die Ausgänge weitergegeben (also 0010). Wiederum eine Taste betätigt folgt bei Adresse 05 der neue INV-Befehl. (Eine Abkürzung aus „inverse“, was soviel bedeutet wie „umgekehrt“). Durch den INV-Befehl werden die einzelnen Dual-Stellen im Register 0 umgekehrt.

Was bringt uns dieser Umkehr-Effekt? Wir erinnern uns noch an die ersten Subtraktions-Programme, die nicht zur vollen Befriedigung ausgeführt wurden, weil keine negativen Ergebnisse dargestellt wurden. Bei der Subtraktion 7-8 war das Ergebnis F und nicht -1 wie wir es gerne gehabt hätten.

Mit dem INV-Befehl lässt sich das F aber in 1 bzw. alle Zahlen in die entsprechenden negativen Zahlen umwandeln.

Bei der Rechnung 7-8 ist das Ergebnis F, wobei gleichzeitig das CARRY-FLAG gesetzt wird. Hierdurch können wir mit dem BRC-Befehl zu einem anderen Programm-Teil springen, in welchem F in 1 umgewandelt wird. Hierfür wird folgender Algorithmus angewandt:

F = dual 1111. Die Dual-Zahl wird zunächst invertiert, folglich ist das Ergebnis 0000. Wird hierzu der Wert 1 addiert, ergibt sich dual 0001. Gemäß Tabelle Dual-System ist das dezimale Ergebnis 1. Dies funktioniert auch mit allen anderen Zahlen, z. B.: 3-7 = C (hexadezimal gerechnet, wie dies vom Computer ausgeführt wird). Eigentlich sollte das Ergebnis -4 sein. Mit unserer neuen Rechenart ergibt sich folgendes:

C hat die Dual-Zahl	1100
invertiert ergibt sich	0011
+ 1 addiert =	0100 (oder dezimal 4)

## Subtraktion mit Angabe von negativen Ergebnissen

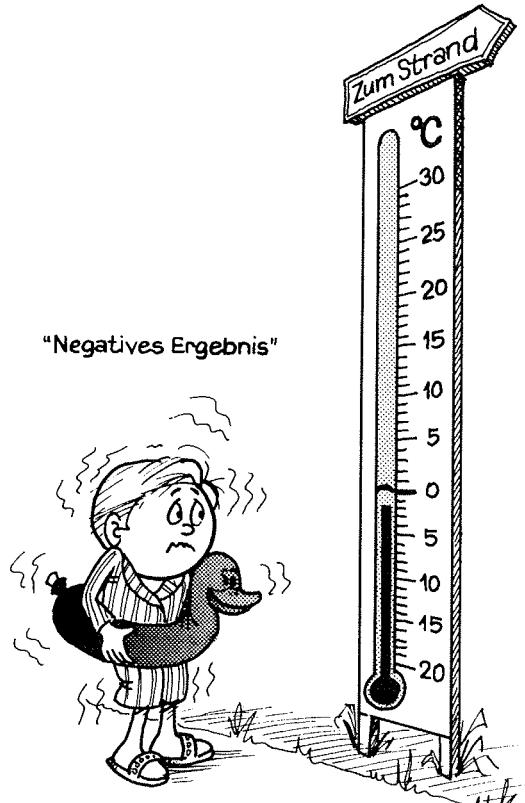
Wir geben folgendes Programm ein:

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
00	<b>F08</b>	CLEAR	
01	<b>F10</b>	DISP 1,0	
02	<b>FF0</b>	KIN 0	Eingabe in Register 0
03	<b>FF1</b>	KIN 1	Eingabe in Register 1
04	<b>610</b>	SUB 1,0	Reg. 0 – Reg. 1 = Reg. 0
05	<b>D07</b>	BRC 07	Wenn Übertrag Sprung nach 07
06	<b>C01</b>	GOTO 01	
07	<b>F80</b>	INV 0	Register 0 wird invertiert
08	<b>510</b>	ADDI 1,0	1 wird zu Register 0 hinzugezählt
09	<b>C01</b>	GOTO 01	

Mit HALT – NEXT – 00 – RUN in Betrieb nehmen. Display zeigt 0. Wir geben jetzt beispielsweise ein 8 und 5. Das Display zeigt 3. Der Computer hat gerechnet  $8 - 5 = 3$ .

Eingabe 7-8=-1 (das Minus-Zeichen wird momentan nicht gesetzt, weil in unserer Programm-Eingabe hierfür nichts vorgesehen war).

Die Programm-Funktion ist prinzipiell die gleiche wie im Kapitel „Subtrahieren“ beschrieben, lediglich, daß durch den neu eingefügten INV-Befehl nun auch „negative“ Dezimal-Ergebnisse erzielt werden.



## Dezimale Subtraktion mit Minus-Anzeige

Wir sehen bereits aus der folgenden Programm-Tabelle, daß sich ein erheblich größerer Programmierungs-Umfang ergibt, nur weil wir jetzt nicht mehr im hexadezimalen, sondern im 2-stelligen dezimalen Bereich rechnen wollen. Außerdem soll ein negatives Ergebnis kenntlich gemacht werden.

Nach HALT – NEXT – 00 geben wir das Programm ein:

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
00	<b>F08</b>	CLEAR	alle Register löschen
01	<b>F21</b>	DISP 2,1	Reg. 1 u. Reg. 2 anzeigen
02	<b>F0A</b>	RSC	Carry-Flag rücksetzen (LED aus)
03	<b>FF0</b>	KIN 0	Eingabe in Reg. 0
04	<b>9A0</b>	CMPI A,0	eingegebener Wert A?
05	<b>E0C</b>	BRZ OC	dann Sprung und Zahlen umspeichern
06	<b>9B0</b>	CMPI B,0	eingegebener Wert B?
07	<b>E0E</b>	BRZ 0E	dann Sprung zur Subtraktion
08	<b>D00</b>	BRC 00	größer als B, alle Register löschen (Sprung nach 00)
09	<b>012</b>	MOV 1,2	Register eine Stelle weiterverschieben
0A	<b>001</b>	MOV 0,1	
0B	<b>C03</b>	GOTO 03	
0C	<b>F0F</b>	EXRA	Registertausch
0D	<b>C03</b>	GOTO 03	
0E	<b>619</b>	SUB 1,9	die beiden rechten Stellen subtrahieren
0F	<b>D11</b>	BRC 11	Übertrag?
10	<b>C14</b>	GOTO 14	
11	<b>769</b>	SUBI 6,9	Dezimalausgleich
12	<b>71A</b>	SUBI 1,A	Übertrag von der linken Stelle (Reg. A) abziehen
13	<b>D18</b>	BRC 18	Übertrag? Sprung nach 18
14	<b>62A</b>	SUB 2,A	die beiden linken Stellen subtrahieren
15	<b>D19</b>	BRC 19	Übertrag? Ja, Sprung nach 19
16	<b>F0F</b>	EXRA	Registertausch, um Ergebnis in die Anzeige zu bringen
17	<b>C02</b>	GOTO 02	
18	<b>62A</b>	SUB 2,A	die beiden linken Stellen subtrahieren
19	<b>76A</b>	SUBI 6,A	Dezimalkorrektur
1A	<b>F89</b>	INV 9	Reg. 9 invertieren
1B	<b>769</b>	SUBI 6,9	Dezimalkorrektur
1C	<b>F8A</b>	INV A	Reg. A invertieren
1D	<b>76A</b>	SUBI 6,A	Dezimalkorrektur
1E	<b>519</b>	Addi 1,9	1 zu Reg. 9 addieren
1F	<b>999</b>	CMPI 9,9	Übertrag?
20	<b>D22</b>	BRC 22	Ja, Sprung nach 22
21	<b>C24</b>	GOTO 24	
22	<b>569</b>	Addi 6,9	Dezimalkorrektur
23	<b>51A</b>	Addi 1,A	
24	<b>F0F</b>	EXRA	Registertausch, um Ergebnis in die Anzeige zu bringen
25	<b>F09</b>	STC	Carry-Flag setzen (Minus-Anzeige)
26	<b>C03</b>	GOTO 03	Sprung nach 03

Programm-Start: HALT – NEXT – 00 – RUN. Das Display zeigt: 00.

Wir können 2-stellige Zahlen voneinander abziehen. Zum Beispiel: 81 – 23. Wir geben ein: 81. Dann Taste A als  $\ominus$  Taste betätigen, dann Eingabe 23 und Taste B als  $\ominus$  Taste betätigen. Wir erhalten das Ergebnis 58. Mit der Taste C kann das Ergebnis gelöscht und anschließend eine neue Aufgabe eingegeben werden.

Wenn wir von 15 z. B. 17 abziehen, ergibt sich das negative Ergebnis 2. Negative Ergebnisse werden durch Aufleuchten des CARRY-FLAGS kenntlich gemacht.

Zur Besseren Übersicht wurde dieses Programm zunächst nur 2-stellig ausgelegt. Beim später folgenden Taschenrechner-Programm mit allen Rechenarten, wird auch die Subtraktion auf 6 Stellen erweitert.

### Wie arbeitet dieses Programm?

Zum besseren Verständnis vergleichen wir wieder die Programm-Tabelle und den Programm-Ablaufplan (bei jedem Symbol wird auch die Adressen-Nummer angegeben).

Von Adresse 00 bis 0D werden die zu rechnenden Werte eingegeben (Eingabe-Routine). Bei Adresse 02 erkennen wir einen neuen Befehl **RSC** (von „reset carry“). Dieser Befehl sorgt dafür, daß das CARRY-FLAG in jedem Fall zurückgesetzt wird, es sei denn, daß sich ein negatives Ergebnis einstellt.

Folgendes Rechenbeispiel erklärt die Subtraktion mit positivem Ergebnis:

$$\begin{array}{r} 81 \\ - 23 \\ \hline = 58 \end{array}$$

Das Programm bringt eine dezimale Ergebnis-Anzeige, der Computer führt die Aufgabe jedoch hexadezimal durch. Wie bei der Addition, werden zuerst die beiden letzten Stellen subtrahiert:  $1 - 3 = E$ . Es ergibt sich also ein negatives Zwischenergebnis, wodurch das CARRY-FLAG gesetzt wird. Das hexadezimale E soll dezimal dargestellt werden. Wir erinnern uns an die Addition: Wurde beim Zusammenzählen die Ziffer 9 überschritten, kommen wir durch ABC... in den Hexadezimal-Bereich, das CARRY-FLAG wurde gesetzt, d. h. der konstante Wert 6 mußte hinzugezählt werden, um wieder die dezimale Darstellung zu erreichen. Ähnliches ergibt sich bei der Subtraktion.

Sobald beim Rückwärtszählen der Wert 0 unterschritten wird, rechnet der Computer hexadezimal rückwärts weiter und ermittelt bei vorstehender Aufgabe das negative Ergebnis E. Durch den Übertrag (CARRY-FLAG wird gesetzt) muß nun ebenfalls wieder der konstante Wert 6 abgezogen werden:  $E - 6 = 8$ , was auch auf dem Display angezeigt wird. Nun müssen noch die vorderen Stellen 8 – 2 subtrahiert werden. Der Computer hat sich jedoch durch das CARRY-FLAG den negativen Übertrag gemerkt, weshalb zunächst von 8 der negative Übertragswert 1 abgezogen wird: Zwischenergebnis 7. Jetzt erfolgt die eigentliche Subtraktion:  $7 - 2 = 5$ . Das jetzt auf dem Display angezeigte Gesamtergebnis 58 ist positiv, weil der bei Adresse 02 vorhandene neue RSC-Befehl dafür gesorgt hat, daß das durch das negative Zwischenergebnis (Subtraktion der beiden letzten Stellen) ausgelöste CARRY-FLAG sofort wieder zurückgesetzt wird.

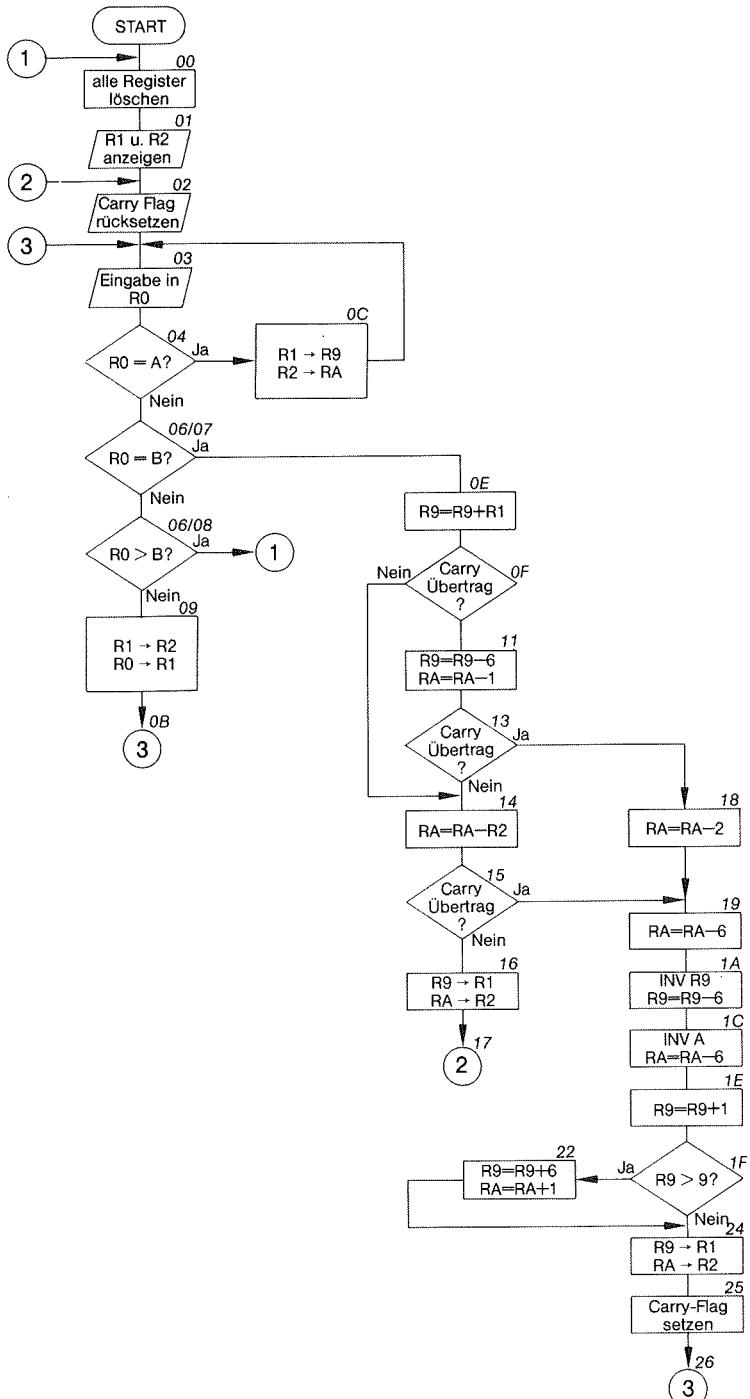
Hätten wir z. B. durch die Subtraktion  $16 - 28 = -12$  ein negatives Ergebnis erhalten, müßte dieses durch ein nichtverlöschendes CARRY-FLAG angezeigt werden. Hierfür ist im Programm unter Adresse 25 der Befehl **STC** enthalten („set carry“).

Das Programm ist auf dem Ablaufplan in 3 Stufen dargestellt. Auf der linken Seite ergibt sich die Eingabe-Routine. In der Mitte wird die eigentliche Subtraktion ausgeführt und auf der rechten Seite erfolgt die Umwandlung, wenn ein Ergebnis negativ wird.

Für die negative Umwandlung sehen wir im letzten Drittel der Programm-Tabelle, genauso wie im Ablaufplan, den INV-Befehl, mit welchem duale Werte umgekehrt werden können.

Der Computer führt eine Subtraktion mit negativem Ergebnis sehr umständlich aus. Die Rechenaufgabe soll lauten:

$$\begin{array}{r} 16 \\ - 28 \\ \hline = -12 \end{array}$$



Folgende Darstellung zeigt uns die einzelnen Rechenschritte wie sie vom Computer hexadezimal und gleichzeitig auch dual ausgeführt werden, bis das dezimale Ergebnis angezeigt wird:

	Dezimale Werte:			= Duale Werte:							
Eingabe über Tastatur		1	6		1	=	0001		6	=	0110
Eingabe über Tastatur	-	2	8		2	=	0010		8	=	1000
	Hexadezimale Rechenart			Duale Rechenart							
1. Schritt: Die beiden letzten Stellen voneinander abziehen – hexadezimales Zwischenergebnis	-		6 8 E					-	6 8 =	=	0110 1000 1110
2. Schritt: Übertrag (weil Ergebnis negativ), daher konstanten Wert 6 für Dezimalkorrektur abziehen	-	=	8	6				-	6 =	=	0110 1000
3. Schritt: Durch Carry-Flag konstanten Wert 1 als Übertrag von vorderen Stellen abziehen, ergibt Zwischenergebnis	-	=	0	1 1 0	-	1 1 =	=	0001 0001 0000			
4. Schritt: Verbleibende vorderen Stellenwerte voneinander abziehen ergibt negatives Zwischenergebnis	-		2 E		-	2 =	=	0000 0010 1110			
5. Schritt: Da Ergebnis negativ (Übertrag durch Carry-Flag) konstanten Wert 6 für Dezimal-Korrektur abziehen. Gesamtergebnis	-	=	8	8	-	6 =	=	0110 1000	8	-	1000
6. Schritt: Negatives Gesamtergebnis wird zunächst als positives Ergebnis dargestellt. Dieses Ergebnis muß invertiert werden.	=		7	7		7	=	0111	7	=	0111
7. Schritt: Da auch Invertierung hexadezimal erfolgte, muß konstanter Wert 6 auf beiden Stellen als Dezimal korrektur abgezogen werden.	-	=	1	1	-	6 =	=	0110 0001	-	6 1	- = 0110 0001
8. Schritt: Durch BRC-Befehl wird Korrektur (als konstanter Wert 1) auf der letzten Stelle dazuaddiert	+		0	1		0	=	0000	1	=	0001
Endergebnis:	=		1	2		1	=	0001	2	=	0010
Das Endergebnis ist negativ	-		1	2							
Der <b>STC</b> -Befehl ( <b>set carry</b> ) sorgt für ein beständiges Carry-Flag, wodurch das Ergebnis negativ dargestellt wird.											

Ist es ein Wunder, daß trotz dieser umständlichen Rechnungsweise das richtige Ergebnis zustande kommt?

Es ist reine Computer-Logik, die für uns schwer verständlich ist. Wir müssen deshalb solche Rechenkunststücke auch nicht im Kopf behalten. Aber es ist nicht uninteressant zu wissen, was z. B. in einem Taschenrechner in Sekundenbruchteilen vor sich geht.

Um die Computer-Logik zu verstehen, wollen wir die zuletzt gelernten und etwas schwierigen Rechenoperationen noch einmal zusammenfassen:

1. Mikroprozessoren arbeiten hexadezimal. Um ein korrektes dezimales Ergebnis zu erhalten, muß eine Dezimal-Korrektur durchgeführt werden, sobald der Computer hexadezimale Ergebnisse (A-B-C-D-E-F) erbringt. Bei der Addition ist die Dezimal-Korrektur notwendig, sobald ein Ergebnis größer als 9 ist. In diesem Fall wird der Wert 6 hinzugefügt. Bei der Subtraktion wird die Dezimal-Korrektur ausgeführt, wenn der Wert 0 unterschritten wird, wobei dann der Wert 6 abgezogen wird. Eine Dezimal-Korrektur bewirkt immer einen Übertrag zum nächsten Register (zur nächsten Stelle).

2. Ist das Gesamtergebnis einer Subtraktion negativ, muß dieses negative Ergebnis invertiert werden, um zu einem korrekten negativen Dezimal-Ergebnis zu kommen. Hierzu werden alle Register invertiert, dann wird sofort die vorerwähnte Dezimal-Korrektur durchgeführt (Wert 6 abziehen), wobei anschließend zu jedem Register der konstante Wert 1 hinzugefügt werden muß. Erst dann ist das korrekte negative Ergebnis erreicht.

Die Dezimal-Korrektur nach dem Invertieren ist notwendig, weil die Invertierung hexadezimal vorgenommen wird. So wird z. B. der vor dem Invertieren vorhandene dezimale Wert 3 durch die Invertierung zum hexadezimalen C. Durch die Dezimal-Korrektur wird von C der konstante Wert 6 abgezogen, d. h. C - 6 = 6. Der Wert 6 wiederum ist das dezimale Inverse von 3. Die nachfolgende Tabelle zeigt uns von jedem Wert den dezimal und hexadezimal invertierten Wert.

Inverse-Tabelle

Wert	dezimal invertiert	hexadezimal invertiert
0	9	F
1	8	E
2	7	D
3	6	C
4	5	B
5	4	A
6	3	9
7	2	8
8	1	7
9	0	6
A		5
B		4
C		3
D		2
E		1
F		0

Ein Wert addiert mit einem invertierten Wert muß beim Dezimal-System immer den Wert 10 ergeben. Da sich bei der Addition jedoch immer nur der Wert 9 ergibt, verstehen wir, warum für die Dezimal-Korrektur immer der konstante Wert 1 hinzugefügt werden muß.

Wert	+	Dezimale Inverse	=	+	Dezimale Korrektur	=		
0	+	9	=	9	+	1	=	10
1	+	8	=	9	+	1	=	10
2	+	7	=	9	+	1	=	10
3	+	6	=	9	+	1	=	10
4	+	5	=	9	+	1	=	10
5	+	4	=	9	+	1	=	10
6	+	3	=	9	+	1	=	10
7	+	2	=	9	+	1	=	10
8	+	1	=	9	+	1	=	10
9	+	0	=	9	+	1	=	10

Auch bei einer hexadezimalen Inverse ist die Korrektur in gleicher Weise erforderlich:

Wert	+	Hexadez. Inverse	=	+	Korrektur	=		
0	+	F	=	15	+	1	=	16
1	+	E	=	15	+	1	=	16
2	+	D	=	15	+	1	=	16
3	+	C	=	15	+	1	=	16
4	+	B	=	15	+	1	=	16
5	+	A	=	15	+	1	=	16
6	+	9	=	15	+	1	=	16
usw.								

Mit diesem Kapitel haben wir den schwierigsten Teil der Computer-Logik (für die Programmierung eines Computers notwendige Logik) hinter uns gebracht. Mit jedem der folgenden Kapitel nähern wir uns den größeren und interessanteren Programmen. Für uns ist nun der Computer nicht mehr das unbekannte Wesen, weil wir durch unsere Kenntnisse jetzt wissen, wie er arbeitet und wie er unsere Befehle ausführen kann.

## Mit „High“ und „Low“ an die Computer-Eingänge

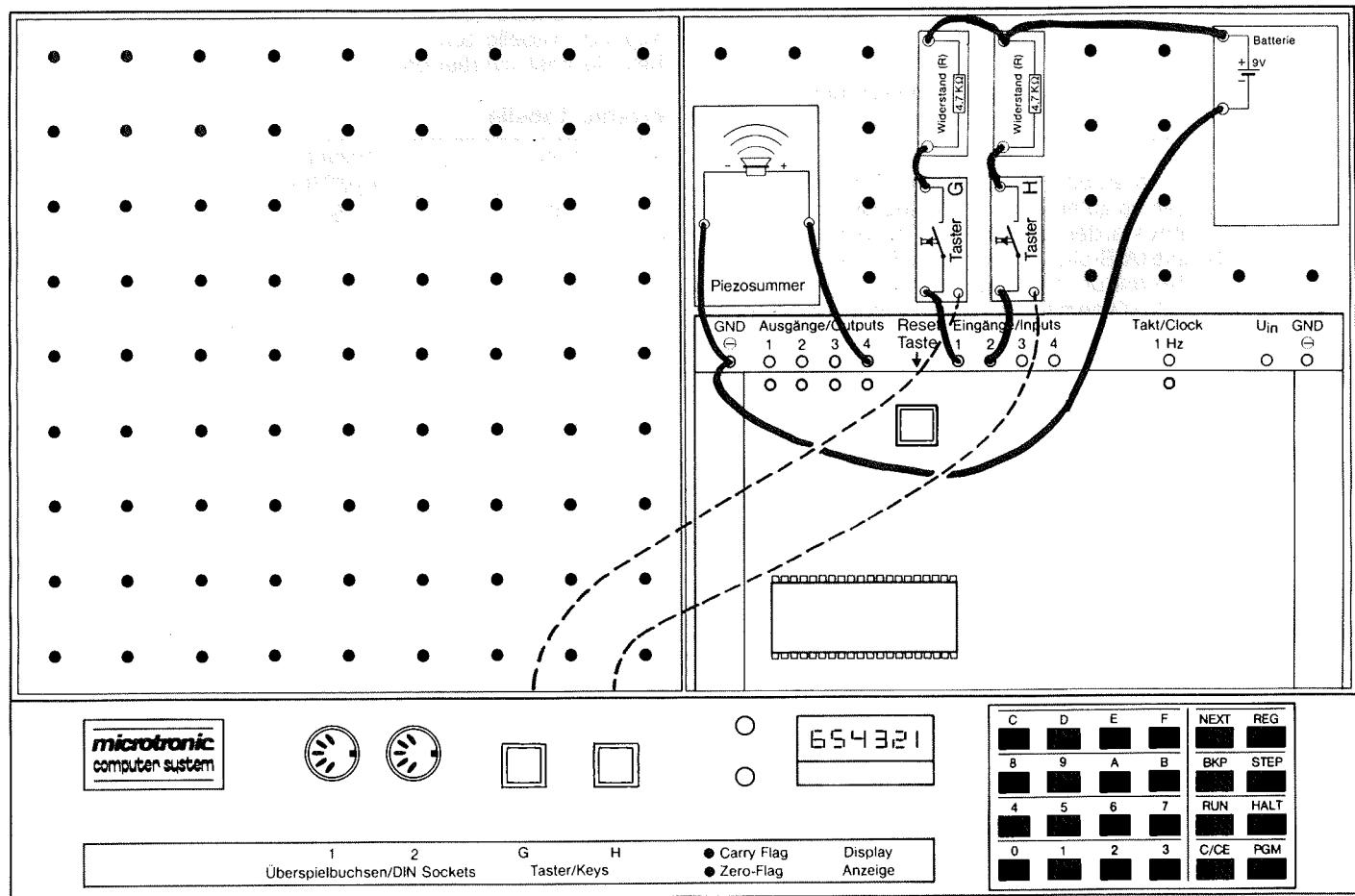
Mit den folgenden Experimenten sollen die Eingänge und Ausgänge der Computer-Platine in die Programmierung einbezogen werden. Ein Eingangs-Signal soll, über den Computer gesteuert, ein Ausgangs-Signal auslösen.

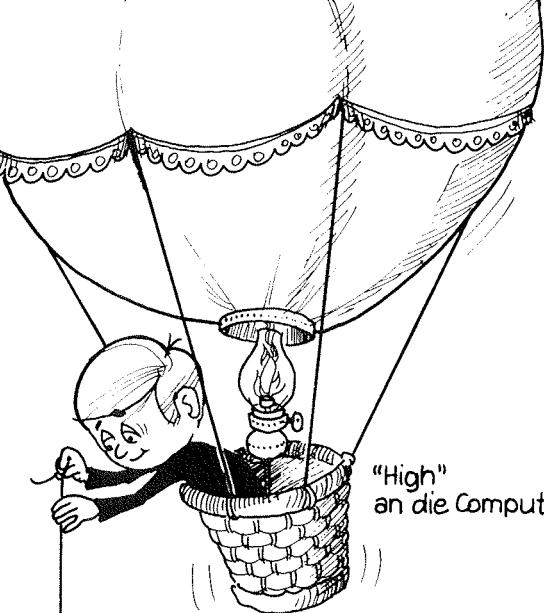
Um an einem Eingang eine Signalwirkung auszulösen, muß eine Spannung zu diesem Eingang herangeführt werden. Ist eine Spannung vorhanden, wird der Wert 1 registriert und man sagt auch, der Eingang ist „high“. Ohne Spannung (Wert 0) ist der Eingang „low“.

Um den Zustand „high“ an einen Eingang zu bringen, benötigen wir eine Spannung. Um diese Spannung zu erzeugen, ist eine zusätzliche Energie-Quelle (eine kleine 9 V Batterie) erforderlich. Hierfür verwenden wir die bei kleinen Transistor-Radios üblichen 9 V Blocks (Bezeichnung IEC 6 F22) z. B. von Varta Super 438 oder von Mallory MN 1604. Die Batterie wird in die Halterung des beigegebenen Batterie-Bausteins eingesetzt. Sobald der am Baustein vorhandene Anschlußclip auf die Batterienoppen aufgedrückt wird, ist der Batterie-Baustein einsatzfähig.

Der Computer arbeitet mit sehr geringen Spannungen. Die Batterie-Spannung 9 V darf niemals direkt an die Eingänge oder Ausgänge des Computers angeschlossen werden. Durch das Zwischenschalten entsprechender Widerstände wird die 9 V Spannung auf ein für den Computer erträgliches Maß reduziert.

Durch Tastendruck sollen kurze „high“-Signale an die Eingänge herangeführt werden. Hierfür sind am Armaturenboard die **roten Tasten G und H** vorhanden. Damit wir die Tasten (mit den Verbindungsleitungen und den gelben Plastiksteckern) leicht anschließen können, finden wir jeweils hinter den Tasten G und H einen Baustein mit der Aufschrift „Taster“. Die Bausteine werden hinter der Computer-Platine eingesteckt (siehe Abbildung unten), wobei zwischen den Bausteinen und den im Armaturenboard eingebauten Tasten eine Verbindungsleitung besteht. Der Baustein von Taster G wird links eingesteckt, der Baustein von Taster H wird rechts daneben platziert. Zur Reduzierung der Batterie-Spannung werden noch die beiden Bau-





## Wer ist schneller? Reaktionstest für 2 Personen

Die Anordnung der Taster und Widerstände aus dem vorangegangenen Experiment bleiben unverändert. Mit dem folgenden Programm kommen wir zu einem interessanten Reaktionstest, bei welchem der Computer ermittelt, welche der beiden Tasten G und H zuerst betätigt wurde.

Nach HALT – NEXT – 00 Programm-Eingabe vornehmen:

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
00	<b>F08</b>	CLEAR	alle Register löschen
01	<b>F02</b>	DISOUT	Anzeige ausschalten
02	<b>FE0</b>	DOT 0	Ausgangs-LED's dunkel schalten
03	<b>F05</b>	RND	Zufallsgenerator
04	<b>51C</b>	ADDI 1,C	
05	<b>FBD</b>	ADC D	Zeitschleife
06	<b>E08</b>	BRZ 08	
07	<b>C04</b>	GOTO 04	
08	<b>1F0</b>	MOVI F,0	F in Register 0
09	<b>FE0</b>	DOT 0	F auf die Ausgänge (4 LED's leuchten)
0A	<b>FD1</b>	DIN 1	Eingangswert in Register 1 speichern
0B	<b>E0A</b>	BRZ 0A	wenn Eingangswert 0, dann auf neuen Eingangs wert warten
0C	<b>FE1</b>	DOT 1	Eingangswert auf die Ausgänge legen
0D	<b>FF0</b>	KIN 0	Tastatur-Eingabe
0E	<b>C00</b>	GOTO 00	Sprung zum Programm-Anfang

Bevor wir das Programm starten, muß noch der Piezo-Summer an den Anschlüssen der Computer-Platine GND und AUSGANG 4 angeschlossen werden (siehe Abbildung).

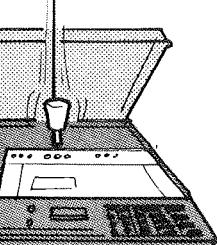
Programm-Start mit HALT – NEXT – 00 – RUN. Nach einer gewissen Zeit ertönt der Piezo-Summer. Gleichzeitig leuchten alle 4 LED's an den Ausgängen. Wird jetzt zuerst die Taste G betätigt, verstummt der Summer und es leuchtet nur noch die LED am Ausgang 1. Der gleiche Effekt ergibt sich, wenn zuerst die Taste H betätigt wird, es leuchtet dann jedoch die LED am Ausgang 2. Werden beide Tasten exakt zur gleichen Zeit betätigt, leuchten beide LED's. Durch Betätigen einer beliebigen Zahlen- oder Buchstaben-Taste, wird der beschriebene Vorgang erneut ausgelöst, wobei die Wartezeit bis zum Ertönen des Summers unterschiedlich kurz oder lang sein kann.

Für einen Reaktionstest wird jedem der beiden Mitspieler eine **rote Taste** zugeordnet. Alsdann den Zeitablauf durch Betätigen einer Buchstaben- oder Zahlen-Taste starten. Sobald der Summer ertönt, kann jeder der Mitspieler seine rote Taste betätigen. Die entsprechende Ausgangs-LED zeigt an, welche Taste zuerst betätigt wurde. Wenn beide Testpersonen exakt gleich schnell sind, leuchten beide LED's. Test-Wiederholung durch Betätigung einer Zahlen- oder Buchstaben-Taste.

### Programm-Beschreibung:

In diesem Programm wird bei Adresse 03 der Zufalls-Generator verwendet. Eine „Zeitschleife“ auf den Adressen 04 bis 07 schließt sich an. Hierdurch ergibt sich eine zufällige und unterschiedlich lange Wartezeit.

Nehmen wir an, daß durch den Befehl RND (Zufalls-Generator) z. B. der Wert E in das Register D übernommen (gespeichert) wird. Durch den folgenden ADDI-Befehl wird der Wert 1 in Register C addiert. Durch den CLEAR-Befehl am Programm-Anfang steht in Register C zunächst lediglich der Wert 0, weshalb in



diesem Register 16 Mal der Wert 1 addiert werden muß, bis der dann vorhandene Wert F übersprungen und der Übertrag zum Register D addiert wird. Wir sind davon ausgegangen, daß in Register D der Wert E vorhanden war, somit ergibt sich durch den Übertrag nun der Wert F. Jetzt wird erneut 16 Mal der Wert 1 in Register C hinzugefügt bis durch den sich ergebenden neuen Übertrag der Wert 0 in Register D steht. Durch den Befehl BRZ 08 kann jetzt die „Zeitschleife“ verlassen werden. Das Verlassen der Zeitschleife ist abhängig von der zufällig gefundenen Zahl in Register D. Da sich ein mehrmaliger und nicht voraberechenbarer Durchlauf der Zeitschleife ergibt, wird eine zufällige Wartezeit erreicht.

Bei Adresse 08 wird der Wert F (dual 1111) in das Register 0 gespeichert und durch den folgenden DOT-Befehl an die Ausgänge gebracht: Alle 4 Ausgänge sind „high“, die LED's leuchten und der Summer ertönt. Bei Adresse 0A wird der an den Eingängen vorhandene Wert 0 in das Register 1 gespeichert, denn so lange keine der roten Tasten betätigt wird, sind die Eingänge „low“, also 0 (dual 0000). Der bei Adresse 0B folgende BRZ-Befehl sorgt durch den Rücksprung zur Adresse 0A dafür, daß die Ausgänge zunächst „high“ bleiben.

Erst jetzt können wir durch Betätigung einer der roten Tasten in den Programm-Ablauf eingreifen. Mit dem Tastendruck auf G oder H wird einer der beiden Eingänge „high“, somit entsteht der Wert 1, welcher bei Adresse 0A im Register 1 übernommen wird. Hierdurch wird bei Adresse 0B der BRZ-Befehl übersprungen und der folgende DOT-Befehl gibt den Eingangs-Wert an die Ausgänge. Der Tastendruck G bringt den Wert 1 (dual 0001), LED am Ausgang 1 leuchtet. Der Tastendruck H bringt den Wert 2 (dual 0010), die LED am Ausgang 2 leuchtet. Werden zufällig beide Tasten absolut zur gleichen Zeit betätigt, ergibt sich der Wert 3 (dual 0011), beide LED's leuchten. Der KIN-Befehl auf Adresse 0D wartet, bis durch eine beliebige Tastenbetätigung der in Register 0 vorhandene Wert geändert wird. Durch den Rücksprung-Befehl zum Programm-Anfang wird das Spiel wiederholt.

## „Rucksack“-Programmierung verbessert den „Reaktionstest“

Ein Programm kann in Etappen entwickelt werden. Am Anfang stehen die Überlegungen, welche Bedingungen durch eine entsprechende Programmierung erreicht werden sollen. Ein Testlauf zeigt, ob die eingegebenen Befehle richtig ausgeführt werden.

Nachdem das im vorangegangenen Kapitel eingegebene Programm einwandfrei läuft, können wir uns überlegen, ob Verbesserungen möglich sind. Unser bisheriges Programm ist z. B. nicht „mogelsicher“, denn wenn eine der Testpersonen die rote Taste ständig gedrückt hat, mußte die LED des Mogelnden zuerst aufleuchten. Es ist davon auszugehen, daß jede der Testpersonen einen Finger auf einer roten Taste hat, um die schnelle Reaktion beim Er tönen des Summers zu beweisen. Deshalb ist es schwer feststellbar ob eine der beiden Personen die Taste zu früh betätigt. Daher ist es zweckmäßig, den Reaktionstest durch Einbau einer „Mogelsicherung“ zu verbessern.

Da es nicht möglich ist, bei einem eingegebenen Programm einige Programm-Schritte dazwischen zu schieben, müßte das ganze Programm neu überlegt und neu eingegeben werden. Bei längeren Programmen ist dies sehr zeitaufwendig. Da sich beim Programmieren auch immer wieder neue Ideen ergeben, müßte die Programmierung immer wieder von neuem begonnen werden.

Mit der sogenannten „Rucksack-Programmierung“ haben wir bei vielen Programmen die Möglichkeit, eine Ergänzung als „Rucksack“ anzuhängen. Dies wollen wir mit einer zusätzlichen „Mogelsicherung“ ausprobieren.

Die folgende Programm-Tabelle zeigt, daß das bisherige Programm von Adresse 00 bis 0E bestehen bleibt. Lediglich bei Adresse 06 muß der Sprung-Befehl BRZ geändert werden, damit wir in den bei Adresse 0F beginnenden neuen Programmteil springen können. Die bisherige Eingabe E08 (Sprung nach Adresse 08) wird geändert in EOF (Sprung nach Adresse 0F).

Das bei Adresse 0F angehängte „Rucksack-Programm“ muß neu eingegeben werden.

Was ist zu tun?

Mit HALT – NEXT – 06 geänderten Befehl eingeben (NEXT nicht vergessen) alsdann mit HALT – NEXT – 0F das anzu hängende „Rucksack-Programm“ eingeben.

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
00	F08	CLEAR	
01	F02	DISOUT	
02	FE0	DOT 0	
03	F05	RND	
04	51C	ADDI 1,C	
05	FBD	ADC D	
06	<b>EOF</b>	BRZ 0F	Änderung beachten
07	C04	GOTO 04	
08	1F0	MOVI F,0	
09	FE0	DOT 0	
0A	FD1	DIN 1	
0B	E0A	BRZ 0A	
0C	FE1	DOT 1	
0D	FF0	KIN 0	
0E	C00	GOTO 00	
0F	<b>FD1</b>	DIN 1	sind Eingänge (wenn keine Taster betätigt wurden) gleich „0“? Ja, dann Sprung zu 08
10	<b>E08</b>	BRZ 08	
11	<b>1F0</b>	MOVI F,0	F in Register 0 speichern
12	<b>FE0</b>	DOT 0	F zu Ausgänge geben
13	51C	ADDI 1,C	Zähler
14	<b>E00</b>	BRZ 00	
15	<b>FE2</b>	DOT 2	(Register 2 = 0) 0 an die Ausgänge geben u.
16	<b>C11</b>	GOTO 11	Sprung zu 11

Programm-Start mit HALT – NEXT – 00 – RUN.

Der Reaktionstest kann in gleicher Weise wie im letzten Kapitel durchgeführt werden. Wenn einer der Mitspieler mogelt, d. h. seine Taste ständig niedergedrückt hält, ertönt für kurze Zeit ein Intervallton. Anschließend wird direkt eine neue Zufallszeit erzeugt – der Test kann (ohne zu mogeln) fortgesetzt werden.

Bei Adresse 06 erfolgt der Sprung zum „Rucksack“ für die Mogelüberprüfung. Bei der ersten Rucksack-Adresse 0F wird überprüft, ob alle Eingänge „low“ (Wert 0) sind. Ist dies der Fall, wurde keine rote Taste betätigt und das Programm wird durch den Rücksprung auf Adresse 08 (Befehl BRZ 08) wie bisher abgearbeitet.

Wird einer der Eingänge „high“, erfolgt die Prüfung ob gemogelt wurde im „Rucksack-Programm“. Auf Adresse 11 wird der Wert F im Register 0 gespeichert und an die Ausgänge gebracht: Alle LED's leuchten, der Piezo-Summer ertönt. Bei Adresse 13 wird durch den ADDI-Befehl der konstante Wert 1 zum Register C hinzugefügt. Der DOT-Befehl bringt den Inhalt von Register 2 an die Ausgänge. Da in Register 2 der Wert 0 vorhanden ist, verlöschen die LED's und der Piezo-Summer verstummt. Sofort anschließend wird durch GOTO das Programm wieder bei Adresse 11 fortgesetzt – das Spiel wiederholt sich: LED's leuchten, Piezo-Summer tönt. Erneuter Rücksprung zu Adresse 11. LED's verlöschen, Piezo-Summer verstummt. Da der Programm-Ablauf sehr schnell erfolgt, sehen wir die flackernen LED's und der Summer erzeugt einen Intervallton. Dieses

Spiel wird solange wiederholt, bis bei jedem Programm-Durchgang auf Adresse 13 durch den konstant hinzugezählten Wert 1 beim Register 2 (nach Überspringen des hexadezimalen F, also nach dem 16. Durchgang) der Wert 0 erreicht wird. Dies wird auf Adresse 14 vom BRZ-Befehl registriert, wodurch der Sprung zum Programm-Anfang erfolgt.

Wenn wir später selbst Programme entwickeln und weiter verbessern, sollten wir uns den Trick der „Rucksack-Programmierung“ bei der Programm-Erstellung und Verbesserung merken. Werden mehrere „Rucksäcke“ angehängt, werden solche Programme allerdings schnell unübersichtlich. Es ist zweckmäßig, das Gesamt-Programm zu überdenken und die einzelnen „Rucksäcke“ in ein neues, logisch ablaufendes Gesamt-Programm einzurichten.

## 2. Programm-Verbesserung:

### Reaktionstest-Programm „de luxe“

Unser Programm Reaktionstest soll weiter ausgebaut werden. Der Computer soll selbstständig ein „Punkte-Konto“ führen, damit bei mehrmaliger Testwiederholung festgestellt werden kann, welcher Mitspieler als erster die 9 nacheinander durchzuführenden Reaktionsteste gewonnen hat. Um den Test noch interessanter zu gestalten, sollen einem mogelnden Spieler alle bisher erreichten Punkte abgezogen werden.

Für diese Programm-Erweiterung müßten mehrere „Rucksäcke“ angehängt werden, wodurch das Programm für unsere bisherigen Kenntnisse zu unübersichtlich wird. Da der Piezo-Summer beim Programmieren stört, klemmen wir eine Verbindungsleitung ab und nehmen dann nach HALT – NEXT – 00 die Neuprogrammierung vor.

#### Reaktionstest „de luxe“ (endgültige Ausführung)

Bevor wir das Programm starten, den Piezo-Summer wieder anschließen. Es soll nochmals darauf hingewiesen werden, daß die Anordnung der Bausteine Taster und Widerstände unverändert wie beim ersten Reaktionstest-Programm bestehen bleiben. Es dürfen also keine zusätzlichen Verbindungsleitungen angebracht werden.

Programm-Start mit HALT – NEXT – 00 – RUN.

Das Display zeigt: 00. Wie bisher ertönt nach einer zufälligen Wartezeit wieder der Piezo-Summer. Wer zuerst seinen roten Taster betätigt, erhält vom Computer einen Punkt gutgeschrieben. Dies ist aus dem Display ersichtlich: Taster G = linke Display-Stelle, Taster H = rechte Display-Stelle. Zusätzlich leuchten wieder die entsprechenden LED's an den Ausgängen. Nach Betätigung einer roten Taste wird das Spiel automatisch fortgesetzt, solange bis ein Spieler 9 Punkte erreicht hat. Auf dem Display erscheint dann links und rechts neben den angezeigten Punkten ein E für Ende. Ein neues 9 Punkte-Spiel kann nach Betätigung einer Zahlentaste gestartet werden.

#### Programm-Beschreibung

Die Programm-Schritte der Adressen 00 bis 08 sind identisch mit unserem ersten Reaktionstest-Programm. Bei den Adressen 09 bis 0F erkennen wir die „Mogelprüfung“, die wir im letzten Programm als „Rucksack“ angehängt hatten. Die folgenden Programm-Schritte überprüfen wer gemogelt hat, wodurch das entsprechende Register auf 0 gesetzt wird. Die Punktzahlen des Spielers mit der roten Taste G werden in Register 4 gezählt, Taster H in Register 3.

Ab Adresse 1A wird der eigentliche Reaktionstest ausgeführt. Durch die entsprechenden Vergleichs-Befehle wird jeweils 1 Punkt im Register des schnelleren Spielers hinzuaddiert.

Das gesamte Programm ist so übersichtlich und einfach aufgebaut, daß längere Erklärungen nicht notwendig sein sollten. Ein Hinweis zu den Adressen 1E bis 21: Durch DIN 1 wird ein Eingangswert in das Register 1 übernommen. Wurde der rote Taster G betätigt, ergibt sich dual der Eingangswert 0001 (also Wert 1). Wurde der Taster H betätigt, ergibt sich dual 0010 (also

#### Reaktionstest „de Luxe“

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
00	<b>F08</b>	CLEAR	
01	<b>F23</b>	DISP 2,3	
02	<b>FE0</b>	DOT 0	
03	<b>F05</b>	RND	
04	<b>51C</b>	ADDI 1,C	
05	<b>FBD</b>	ADC D	
06	<b>E08</b>	BRZ 08	
07	<b>C04</b>	GOTO 04	
08	<b>1F0</b>	MOVI F,0	
09	<b>FD1</b>	DIN 1	
0A	<b>E1A</b>	BRZ 1A	
0B	<b>FE0</b>	DOT 0	
0C	<b>51C</b>	ADDI 1,C	
0D	<b>E10</b>	BRZ 10	
0E	<b>FE2</b>	DOT 2	
0F	<b>C0B</b>	GOTO 0B	
10	<b>FE2</b>	DOT 2	
11	<b>911</b>	CMPI 1,1	Hat Taster G gemogelt?
12	<b>E16</b>	BRZ 16	Hat Taster H gemogelt?
13	<b>921</b>	CMPI 2,1	
14	<b>E18</b>	BRZ 18	
15	<b>C00</b>	GOTO 00	Beide haben gemogelt, Sprung zu 00 (CLEAR) Reg. 4 (Taster G) wird Null
16	<b>104</b>	MOVI 0,4	
17	<b>C03</b>	GOTO 03	
18	<b>103</b>	MOVI 0,3	Reg. 3 (Taster H) wird Null
19	<b>C03</b>	GOTO 03	
1A	<b>FE0</b>	DOT 0	Signalton
1B	<b>FD1</b>	DIN 1	Eingangswert in Reg. 1
1C	<b>E1B</b>	BRZ 1B	
1D	<b>FE1</b>	DOT 1	
1E	<b>911</b>	CMPI 1,1	Taster G betätigt?
1F	<b>E23</b>	BRZ 23	Taster H betätigt?
20	<b>921</b>	CMPI 2,1	
21	<b>E27</b>	BRZ 27	
22	<b>C03</b>	GOTO 03	
23	<b>514</b>	ADDI 1,4	Reg. 4 (Taster G) + 1
24	<b>994</b>	CMPI 9,4	Reg. 4 = 9?
25	<b>E2B</b>	BRZ 2,B	Ja, Sprung zu „Ende-Anzeige“
26	<b>C03</b>	GOTO 03	
27	<b>513</b>	ADDI 1,3	Reg. 3 (Taster H) + 1
28	<b>993</b>	CMPI 9,3	Reg. 3 = 9?
29	<b>E2B</b>	BRZ 2B	Ja, Sprung zu „Ende-Anzeige“
2A	<b>C03</b>	GOTO 03	
2B	<b>1E2</b>	MOVI E,2	E → Reg 2
2C	<b>1E5</b>	MOVI E,5	E → Reg 5
2D	<b>F42</b>	DISP 4,2	Reg. 2 bis Reg. 5 anzeigen
2E	<b>FF0</b>	KIN 0	
2F	<b>C00</b>	GOTO 00	

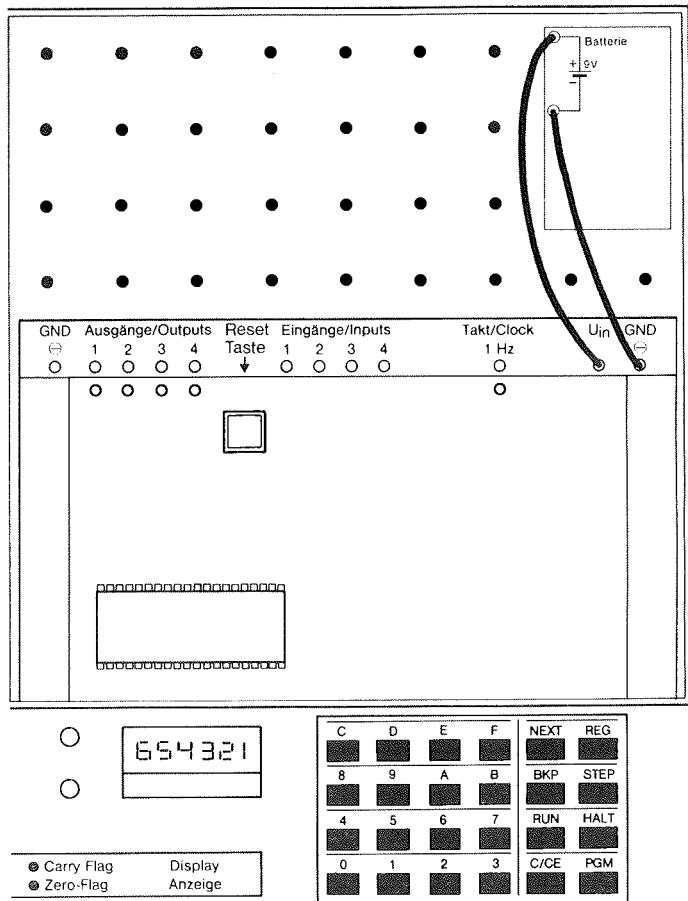
Wert 2). Durch die Befehle CMPI 1,1, bzw. CMPI 2,1 wird verglichen, welche der beiden roten Tasten betätigt wurde. Durch BRZ erfolgt ein Rücksprung und ein Punkt wird dem entsprechenden „Tasten-Register“ hinzugefügt. Wurden zufälligerweise beide Tasten zur gleichen Zeit betätigt, ergibt sich in Register 1 dual 0011 (also Wert 3). Durch den GOTO-Befehl auf Adresse 22 erfolgt ein Sprung zur Zeitschleife, d. h. daß keinem Register (und damit auch keinem Spieler) ein Punkt gutgeschrieben wird.

# Wie können wir bei kurzfristiger Netzstrom-Unterbrechung die eingegebenen Programme im Programm-Speicher halten?

Wie schon erwähnt, kann der Computer im Dauerbetrieb an der Netzstromsteckdose angeschlossen bleiben.

Nun kann es vorkommen, daß der Netzstrom aus irgendwelchen Gründen kurzfristig unterbrochen werden muß, weil der Computer beispielsweise von einem Raum in einen anderen Raum gebracht werden soll. Schade, wenn hierdurch ein größeres Programm verloren ginge und kurze Zeit später wieder neu eingegeben werden müßte.

Für Netzstromunterbrechungen bis zu maximal 1 Stunde, können wir die gespeicherten Programme halten, wenn die 9V-Batterie (siehe Abbildung) rechts hinten an der Computer-Platine angeschlossen wird. Dabei dürfen die Batterie-Pole in keinem Fall vertauscht werden: Der Minus-Pol der Batterie führt zur Anschlußbuchse GND (Ground), der Plus-Pol der Batterie führt zur Anschlußbuchse U<sub>in</sub>. Die Batterie wird erst kurz vor der Netzstrom-Unterbrechung angeschlossen.



**Vor dem Herausziehen des Netzgeräts aus der Steckdose** ist die grüne RESET-Taste auf dem Computer-Platine ca. 10 Sekunden niederzudrücken. Auch während und nach dem Herausziehen des Netzgeräts die RESET-Taste noch ca. 10 Sekunden in Druckstellung halten.

Der Programm-Speicher wird nun durch die Batterie-Spannung betriebsbereit gehalten, während alle übrigen Computer-Funktionen abgeschaltet sind. Es kann also mit Batteriebetrieb nicht weitergearbeitet werden.

Da sich für die Betriebsbereitschaft des Programm-Speichers ein Stromverbrauch von ca. 30 mA ergibt, sollte der Computer schnellstmöglichst (zur Schonung der Batterie) wieder an einer Steckdose angeschlossen werden.

**Vor dem Einsticken des Netzgeräts in die Steckdose** ist erneut die RESET-Taste ca. 10 Sekunden lang zu drücken, dann (bei gedrückter RESET-Taste) das Netzgerät einzustecken. Nach dem Einsticken die RESET-Taste noch ca. 10 Sekunden in Druckstellung halten.

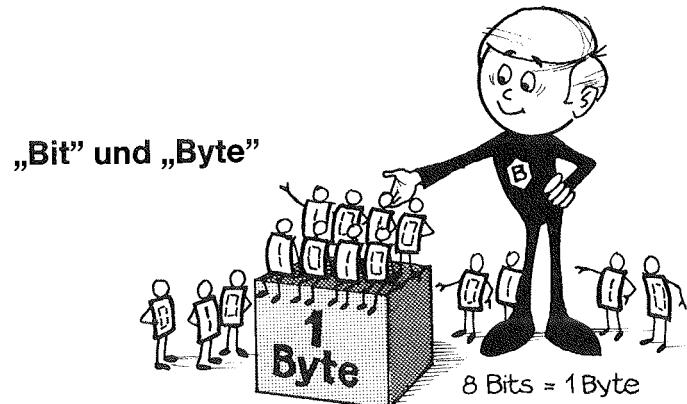
Batterie-Anschlüsse entfernen und der Computer ist wieder wie üblich einsatzbereit. Ein kurzer Programm-Test wird es uns beweisen, daß die eingegebenen Programme noch vorhanden sind.

Die Betätigung der RESET-Taste kurz vor und während der Netzstromunterbrechung (bzw. während der Netzstromeinschaltung) ist unbedingt erforderlich, damit keine Daten der eingegebenen Programme verloren gehen. Sollte durch falsche Betätigung der RESET-Taste ein Programm nicht einwandfrei funktionieren (was sich vor allem bei den ersten und letzten Adressen des Programm-Speichers auswirken könnte), müßten einzelne Korrekturen in der bekannten Weise durchgeführt werden.

## Ist auch eine Langzeit-Batterie-Versorgung des Programm-Speichers möglich?

In der Standard-Lieferausführung ist dies nicht möglich. Wir können jedoch auf der Computer-Platine den dort u. a. enthaltenen Speicher-Baustein gegen einen anderen Typ austauschen, welcher durch besonders geringe Stromaufnahme speziell für Batterie-Betrieb geeignet ist.

Bevor wir technische Eingriffe an der Platine vornehmen, sollten wir jedoch unseren Computer etwas besser kennenlernen. Hinweise zu diesem Thema finden wir im Kapitel „Speicher und Speicher-Möglichkeiten“.



Im Zusammenhang mit Computern wird häufig von Bits (Mehrzahl von Bit) und Bytes (Mehrzahl von Byte) gesprochen. Aus der technischen Beschreibung unseres Microtronic-Computers sehen wir z. B., daß ein 4-Bit-Mikroprozessor verwendet wird.

Aus unseren vorangegangenen Experimenten wissen wir, daß unser Mikrocomputer hexadezimal arbeitet, und daß die 16 hexadezimalen Werte (von 0 bis F) im Mikroprozessor durch die aus 0 und 1 gebildeten 4-stelligen Dualzahlen verarbeitet werden. Für jede Dual-Stelle benötigen wir 1 Bit. Da der Mikroprozessor mit 4 Dualstellen arbeitet, sind für seine Tätigkeit 4 Bit notwendig. Ein **Bit** (Abkürzung für „binary digit“) ist somit die kleinste Speicher-Einheit eines Mikroprozessors (und eines Computers).

Für jeden hexadezimalen Wert, den wir in einem unserer Register verarbeiten, benötigen wir 4 Bits. Diese 4 Bits müssen innerhalb des Mikroprozessors gleichzeitig, z. B. vom Programm-Speicher zu den Ausgängen gebracht werden. Hierfür sind 4 Leitungen erforderlich. Wir erinnern uns an den „Daten-Bus“. Ein 4-Bit-Mikroprozessor hat einen 4-Bit breiten Daten-Bus. Dies bedeutet z. B., daß immer nur eine 1-stellige hexadezimale Ziffer in einem Arbeitsgang bearbeitet (z. B. addiert) werden kann.

4-Bit-Mikroprozessoren haben weltweit die größte Verbreitung gefunden. Sie finden sich in fast allen Taschenrechnern, Auto-Computern, elektronischen Spielen, Haushaltsmaschinen, Steuerungen usw.

Außer den 4-Bit-Mikroprozessoren gibt es auch 8-Bit-Mikroprozessoren. Diese haben einen 8-Bit breiten Daten-Bus und können gleichzeitig z. B. 2-stellige hexadezimale Ziffern in einem Arbeitsgang verarbeiten. Hierdurch wird die Arbeitsgeschwindigkeit des Mikroprozessors verdoppelt. 8-Bit-Mikroprozessoren werden vor allem in EDV-Anlagen eingesetzt, weil dort durch den hohen Datenanfall eine höhere Arbeitsgeschwindigkeit erforderlich ist.

Außerdem gibt es noch 16-Bit-Mikroprozessoren, die in Groß-Rechen-Anlagen eingesetzt werden, um die dort anfallenden enormen Datensummen mit höchster Geschwindigkeit zu verarbeiten. 32-Bit-Mikroprozessoren für noch höhere Anforderungen sind in der Entwicklung.

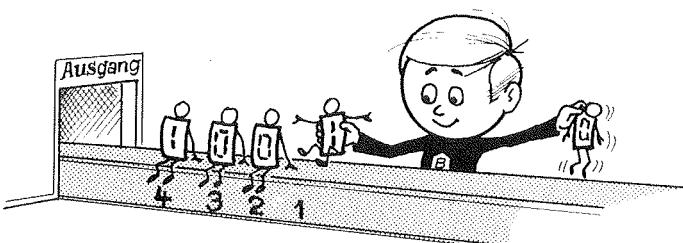
Man kann nicht sagen, daß ein 8-Bit-Mikroprozessor durch seine höhere Arbeitsgeschwindigkeit besser ist als ein 4-Bit-Prozessor. Es kommt letztlich darauf an, was durch den Mikroprozessor-Einsatz erreicht werden soll. Auch das Preis-/Leistungsverhältnis ist ausschlaggebend für die Auswahl eines entsprechenden Prozessors.

Der 4-Bit-Mikroprozessor hat für uns vor allem den Vorteil, daß seine Arbeitsweise noch halbwegs darstellbar und übersichtlich ist.

### Und was sind Bytes?

Ein Byte hat 8 Bits. 4 Bits reichen aus für die 16 einstelligen hexadezimalen Möglichkeiten. Acht Bits (also ein Byte) sind erforderlich, wenn das gesamte Alphabet mit Groß- und Kleinbuchstaben, allen Zahlen und Zeichen darzustellen sind. Mit einem Byte ergeben sich bereits 256 verschiedenartige Darstellungsmöglichkeiten.

Es ist auch üblich, Speicherkapazitäten nicht in Bit sondern in Byte anzugeben. In unserem Mikroprozessor z. B. stehen für das Monitor-Programm (Betriebssystem) 4096 Bytes (32.768 Bits) zur Verfügung. Außerdem ist im Mikroprozessor ein RAM-Speicher mit 64 Bytes (512 Bits) vorhanden. Über den Mikroprozessor wird ein externer RAM-Speicher angesteuert (zur Ein- und Ausgabe der Programm-Schritte) mit einer Kapazität von 512 Bytes (4.096 Bits.).



### Wir verändern einzelne Bits:

## Logische Operationen

Beim vorangegangenen Reaktionstest haben wir erfahren, wie die Ein- und Ausgänge des Computers verwendet werden können. Für jeden der 4 Eingänge benötigen wir 1 Bit, wobei dieses Bit entweder 0 oder 1 sein kann. Die „0-Bit“ oder „1-Bit“ können wir über den Mikroprozessor an die Ausgänge bringen. Mit einem „1-Bit“ ergibt sich am entsprechenden Ausgang eine Spannung: LED leuchtet oder Piezo-Summer tönt. Mit „0-Bit“ ist keine Spannung am Ausgang vorhanden: LED leuchtet nicht, Summer bleibt stumm. Wenn wir am Ausgang 4 eine LED zum leuchten bringen wollen, benötigen wir dual 1000, d. h. daß im entsprechenden Register der Wert 8 vorhanden sein muß. Soll die LED am Ausgang 1 leuchten, benötigen wir dual 0001, d. h. daß im entsprechenden Register der Wert 1 vorhanden sein muß.

In der Praxis kommt es häufig vor, daß wir von den 4 Bits, die z. B. an den 4 Ausgängen vorhanden sind, ein einzelnes Bit (also einen Ausgang) verändern möchten.

Zum besseren Verständnis werden wir ein kleines Programm eingeben. Die Taster und Widerstände bleiben wie beim Reaktionstest an der Computer-Platine angeschlossen. Die Zuleitung zum Piezo-Summer wird unterbrochen, damit sein fortwährendes Summen nicht stört.

Nach HALT – NEXT – 00 geben wir folgendes Programm ein:

Adresse	Befehls-Eingabe	Mnemonic	Erklärungen
00	F02	DISOUT	Anzeige ausschalten
01	181	MOVI 8,1	konstanten Wert 8 in Register 1 speichern
02	FD0	DIN 0	Data in
03	A10	OR 1,0	logisches „Oder“
04	FE0	DOT 0	Data out
05	C00	GOTO 00	Rücksprung

Programm-Start: HALT – NEXT – 00 – RUN.

Das ZERO-Flag neben dem Display flackert, die rechte LED am Ausgang 4 leuchtet. Betätigen wir die rote Taste G leuchtet zusätzlich die LED am Ausgang 1. Betätigen wir die rote Taste H leuchtet die LED am Ausgang 2 zusätzlich zur LED am Ausgang 4. Werden beide roten Tasten betätigt, leuchten zusätzlich zur LED am Ausgang 4 die beiden LED's an den Ausgängen 1 und 2.

Durch Betätigung einer roten Taste haben wir also jeweils nur ein einzelnes Bit am entsprechenden Ausgang geändert. Die Änderung einzelner Bits nennt man auch „logische Operationen“. Hierfür steht uns der OR-Befehl zur Verfügung.

Betrachten wir uns das Programm:

Mit dem DISOUT-Befehl wird das Display abgeschaltet. Mit dem MOVI-Befehl bringen wir den konstanten Wert 8 in Register 1. Der Wert 8 ergibt dual 1000. Der folgende OR-Befehl stellt fest, daß in Register 1 der Wert 8 vorhanden ist. Dieser Wert wird durch den OR-Befehl an das Register 0 weitergegeben und gelangt durch den DOT-Befehl an den Daten-Ausgang: Die LED am Ausgang 4 leuchtet.

Durch die rote Taste G bringen wir über Adresse 02 den konstanten Wert 1 (dual 0001) in das Register 0. Der wieder folgende OR-Befehl vergleicht nun die dualen Werte in Register 1 und in Register 0 und faßt alle Bits mit dem Wert 1 zusammen. Dies kann folgendermaßen dargestellt werden.

Ausgänge Nr.	4	3	2	1
Im Register 1 ergibt sich Wert 8 = dual	1	0	0	0
Im Register 0 durch Taste G Wert 1 = dual	0	0	0	1
OR-Befehl faßt alle duale 1 zusammen	1	0	0	1

Durch den OR-Befehl wurden also die „1-Bits“ in den beiden Registern zusammengefaßt, wodurch 2 Ausgänge „high“ wurden und die LED's an den Ausgängen 4 und 1 leuchten.

Durch die rote Taste H bringen wir den Wert 2 in Register 0 und es ergibt sich folgendes:

Ausgänge Nr.	4	3	2	1
Im Register 1 Wert 8	=	1	0	0
Im Register 0 durch Taste H Wert 2	=	0	0	1
OR-Befehl faßt zusammen	=	1	0	1

Jetzt leuchten die LED's an den Ausgängen 4 und 2.

Bei gleichzeitiger Betätigung der roten Tasten G und H:

Ausgänge Nr.	4	3	2	1
Im Register 1 Wert 8	=	1	0	0
Im Register 0 durch Taste G und H haben wir jetzt den Wert 3	=	0	0	1
OR-Befehl faßt zusammen	=	1	0	1

Die LED's an den Ausgängen 4, 2 und 1 leuchten. Der **OR-Befehl** ist also ebenfalls ein Vergleichs-Befehl, welcher die „1-Bit“ in 2 verschiedenen Registern vergleicht und die „1-Bits“ zusammenfaßt. Der pauschale Befehls-Code lautet **Asd**. Wir erinnern uns: **s** = Quell-Register und **d** = Ziel-Register. Im vorstehenden Programm-Beispiel haben wir den Code eingegeben: **A10**, was bedeutet, daß das Register **1** mit dem Register **0** ver-

glichen wird, und daß alle „1-Bits“ in Register 0 zusammengefaßt werden.

Der OR-Befehl ist ein „**ODER**-Befehl“ was besagt: Das Vergleichsergebnis ergibt „1-Bit“, wenn in einem Register ein „1-Bit“ **oder** in einem zweiten Register ein „1-Bit“ vorhanden ist.

### Auch mit dem AND-Befehl können einzelne Bits manipuliert werden

Ein zweiter Befehl für die „Bit-Manipulierung“ ist der AND-Befehl. Während der vorangegangene OR-Befehl die „1-Bits“ beeinflußt hat, können wir mit dem AND-Befehl die „0-Bits“ beeinflussen.

Zum besseren Verständnis geben wir nach HALT – NEXT – 00 folgendes Programm ein:

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
00	<b>F02</b>	DISOUT	Anzeige ausschalten
01	<b>111</b>	MOVI 1,1	konstanten Wert 1 in Register1 speichern
02	<b>FD0</b>	DIN 0	Data in
03	<b>210</b>	<b>AND 1,0</b>	logisches AND (UND)
04	<b>FE0</b>	DOT 0	Data out
05	<b>C00</b>	GOTO 00	Rücksprung Adresse 00

Widerstände, Taster und Batterie (ohne Piezo-Summer) sind wie beim letzten Versuch an der Computer-Platine angeschlossen.

Programm-Start: HALT – NEXT – 00 – RUN

Keine Anzeige auf dem Display. Keine LED leuchtet. Betätigen wir die rote Taste G leuchtet die LED am Ausgang 1. Betätigen wir die Taste H leuchtet keine LED.

Bei diesem Experiment scheint es sich um keinen interessanten Versuch zu handeln?

Der Schein trügt, denn wir haben wieder eine wichtige logische Operation ausgelöst, die wir verstehen sollten.

Wenn wir das Kurz-Programm mit dem vorangegangenen Experiment vergleichen, dann haben wir lediglich bei Adresse 01 den MOVI-Befehl geändert, indem wir jetzt den konstanten Wert 1 in das Register 1 speichern. Auf Adresse 03 haben wir den neuen AND-Befehl eingefügt, welcher ähnlich wie der vorangegangene OR-Befehl wieder die Register 1 und 0 vergleicht. Durch den MOVI-Befehl mit dem Wert 1 ergibt sich wieder dual 0001, weshalb eigentlich am Ausgang 1 eine LED leuchten müßte. Warum ist dies nicht der Fall?

Durch den AND-Befehl ergibt sich folgende Situation:

Ausgänge Nr.	4	3	2	1
In Register 1 steht durch MOVI der Wert 1	=	0	0	0 1
keine rote Taste bedeutet: In Register 0 Wert 0	=	0	0	0 0
Der AND-Befehl bewirkt im Register 0	=	0	0	0 0

Der AND-Befehl bewirkt in Register 0 den Wert 0 (dual 0000) somit ist an keinem Ausgang eine Spannung vorhanden, eine LED kann nicht leuchten.

Wenn wir die Taste G betätigten, ergibt sich folgendes:

Ausgänge Nr.	4	3	2	1
Im Register 1 steht durch MOVI der Wert 1	=	0	0	0 1
Rote Taste G bewirkt in Register 0 den Wert 1	=	0	0	0 1
Der AND-Befehl bewirkt in Register 0	=	0	0	0 1

Vorstehende Tabelle zeigt, daß unter Ausgang Nr. 1 zweimal ein „1-Bit“ untereinander steht und der AND-Befehl übernimmt den Wert 1 in das Register 0: Die LED am Ausgang Nr. 1 leuchtet.

Betätigen wir nun die rote Taste H wird uns deutlich, wie der AND-Befehl arbeitet:

Ausgänge Nr.	4	3	2	1
In Register 1 steht durch MOVI der Wert 1	=	0	0	0 1
Rote Taste H bewirkt in Register 0 den Wert 2	=	0	0	1 0
Der AND-Befehl bewirkt in Register 0	=	0	0	0 0

Der AND-Befehl vergleicht, ob die „1-Bits“ jeweils an der gleichen Stelle (in vorstehenden Tabellen untereinander) stehen. Beim letzten Beispiel stehen die „1-Bits“ versetzt (am Ausgang 2 und am Ausgang 1) und der AND-Befehl gibt in Register 0 den Wert 0. Alle Ausgänge bleiben „low“. Der Druck auf die rote Taste H bleibt ohne Auswirkung (er wird unterdrückt) und durch den Wert 0 (dual 0000) kann keine LED leuchten.

Der **AND-Befehl** ist ein „**UND-Befehl**“, denn er bewirkt: **1 und 1 = 1**. Der vorangegangene **OR-Befehl** bewirkt dagegen: **0 oder 1 = 1**.

Mit dem AND-Befehl können wir an einem Ausgang ein „1-Bit“ unterdrücken, bzw. ein Ausgang der normalerweise „high“ sein müßte, kann „low“ gemacht werden. Dies kann bei einem Programm von großer Bedeutung sein, wenn wir z. B. intern das Uhren-Programm mitverwenden. Z. B., wenn wir vom Ausgang TAKT/CLOCK ein Signal übernehmen, welches zu bestimmten Zeiten unterdrückt werden soll.

Wir können das ausprobieren, indem wir von TAKT/CLOCK eine Verbindungsleitung zum Eingang Nr. 2 herstellen. Normalerweise müßte jetzt am Ausgang Nr. 2 die LED im gleichen Takt blinken wie die TAKT-LED. Der AND-Befehl unterdrückt jedoch das von TAKT/CLOCK kommende Signal.

Ändern wir jetzt auf Adresse 03 den AND-Befehl, indem wir aus dem vorangegangenen Experiment den OR-Befehl eingeben (HALT – NEXT – 03, Befehls-Code A10 eingeben, Taste NEXT und mit HALT – NEXT – 00 – RUN das Programm wieder starten) ändert sich die Situation: Auf Ausgang 1 leuchtet die LED ständig. Auf den Ausgang 2 wird das TAKT-Signal übertragen – die LED blinkt.

### Hat ein Bit einen Wert?



Wir können diese Frage sicherlich mit ja beantworten, weil wir bei den letzten Experimenten die Unterschiede zwischen „0-Bit“ und „1-Bit“ kennengelernten. Ist ein „1-Bit“ mehr wert als ein „0-Bit“?

Entscheidend ist es, an welcher Stelle (an welchen Ausgängen) die „0-Bits“ und „1-Bits“ stehen. Die 4 Ausgänge haben nämlich unterschiedliche Werte:

Die Ausgänge Nr.	4	3	2	1
haben die Werte	8	4	2	1

Je nach dem wie die „1-Bits“ in dieser Tabelle stehen, ergeben sich unterschiedliche Werte:

Ausgänge Nr.				somit haben die „1-Bits“ folgende Werte:
4	3	2	1	
haben die Werte				
8	4	2	1	0
0	0	0	0	1
0	0	0	1	2
0	0	1	0	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	A(10)
1	0	1	1	B(11)
1	1	0	0	C(12)
1	1	0	1	D(13)
1	1	1	0	E(14)
1	1	1	1	F(15)

Die Tabelle zeigt uns nichts anderes, als das bekannte Dual-System mit der hexadezimalen Wertangabe.

Übrigens, es ist ganz einfach die dualen Werte in dezimale Werte umzuwandeln. Wir müssen lediglich die dualen „1“, die jeweils unter den „Werten“ stehen, addieren. Beispiel:

**Werte der Ausgänge:**

Dualzahl:

Werte bei dual „1“ addiert =

$$\begin{array}{r} 8 \quad 4 \quad 2 \quad 1 \\ 0 \quad 1 \quad 1 \quad 1 \\ \hline 4 + 2 + 1 = 7 \end{array}$$

Der Computer hat in vielen Dingen seine eigene Gedankenwelt. Man muß sich „eindenken“ und kommt dann zur Erkenntnis, daß die Computer-Logik auch für uns logisch werden kann.

### MAS mit dem pauschalen Befehls-Code F7d

MAS kann man mit „MOVE: Arbeitsregister in Speicherregister“ bezeichnen. Durch den MAS-Befehl wird der Inhalt eines Arbeitsregisters in einem Speicherregister gespeichert. Für **d** kann die Register-Nr. angegeben werden. Beispiel: MAS 5 (Befehls-Code F75) bedeutet, daß der Inhalt vom Arbeitsregister Nr. 5 im Speicherregister Nr. 5 gespeichert wird.



Für spezielle Berechnungen stehen uns somit insgesamt 32 Register zur Verfügung. Anwendungsbeispiele finden wir bei den nun folgenden neuen Rechen-Operationen.

## Die Multiplikation

Diesem Kapitel soll vorangestellt werden, daß unser Computer nicht „nur“ Rechen-Operationen durchführen kann, denn wir werden später noch sehen, daß alle „Computer-Spielereien“ an irgendeiner Stelle des Programms Rechen-Operationen erforderlich werden lassen. Ein Teil des Mikroprozessors ist ständig als „Rechner“ tätig, und wir werden feststellen, daß mit Rechen-Operationen noch ganz andere Dinge als „nur rechnen“ bewältigt werden können.

Im Monitor-Programm (Betriebs-System) stehen uns u.a. 2 Befehle zur Verfügung, die normalerweise für Mikroprozessoren ungewöhnlich sind: Ein Multiplikations- und ein Divisions-Befehl.

Zunächst befassen wir uns mit dem Multiplikations-Befehl. Er multipliziert 6 Arbeitsregister mit 6 Speicherregistern, d. h. daß wir 6-stellige Multiplikationen durchführen können. Diese Multiplikation wird außerdem nicht wie beim Mikroprozessor sonst üblich hexadezimal, sondern sofort dezimal ausgeführt.

Nach HALT – NEXT – 00 geben wir folgendes Programm ein:

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
00	<b>F08</b>	CLEAR	
01	<b>F60</b>	DISP 6,0	
02	<b>FF6</b>	KIN 6	
03	<b>9A6</b>	CMPI A,6	
04	<b>E0F</b>	BRZ 0F	
05	<b>9B6</b>	CMPI B,6	
06	<b>E11</b>	BRZ 11	
07	<b>D00</b>	BRC 00	
08	<b>045</b>	MOV 4,5	
09	<b>034</b>	MOV 3,4	
0A	<b>023</b>	MOV 2,3	
0B	<b>012</b>	MOV 1,2	
0C	<b>001</b>	MOV 0,1	
0D	<b>060</b>	MOV 6,0	
0E	<b>C02</b>	GOTO 02	
0F	<b>F0D</b>	EXRL	Arbeitsregister – Speicherregister tauschen
10	<b>C00</b>	GOTO 00	Rücksprung für 2. Eingabe
11	<b>F0B</b>	<b>MULT</b>	Multiplikation ausführen
12	<b>C02</b>	GOTO 02	Rücksprung für Ergebnisanzeige

Programm-Start: HALT – NEXT – 00 – RUN. Das Display zeigt 000000.

## Arbeitsregister – Speicherregister

Für unsere Tätigkeit mit dem Computer stehen uns 16 verschiedene Register (0, 1, 2 . . . bis F) zur Verfügung. In diese Register können wir Werte eingeben, und wir können uns z. B. die Werte anzeigen lassen. Da wir mit diesen Registern arbeiten, nennt man sie „Arbeitsregister“.

Neben diesen 16 Arbeitsregistern hat unser Microtronic-Computer auch noch 16 Speicherregister in denen wir Zahlen oder Werte speichern können, solange bis sie einmal gebraucht werden.

Die Inhalte der Speicherregister sind uns nicht direkt zugänglich, sondern wir müssen sie durch spezielle Befehle in die Arbeitsregister bringen, damit wir die Inhalte der Speicherregister aufrufen können. Hierfür stehen uns folgende 3 Befehle zur Verfügung:

### EXRL mit dem Befehls-Code F0D

Dieser Befehl bewirkt, daß die Inhalte der Arbeitsregister 0 bis 7 in die Speicherregister 0 bis 7 gebracht werden. Gleichzeitig werden die Inhalte der Speicherregister 0 bis 7 in die Arbeitsregister 0 bis 7 gebracht. Die Arbeits- und Speicherregister 0 bis 7 werden also ausgetauscht.

### EXRM mit dem Befehls-Code F0E

Gleiche Arbeitsweise wie der EXRL-Befehl, es werden jedoch die Inhalte der Arbeits- und Speicherregister 8 bis F ausgetauscht.

Wir möchten folgende Multiplikation ausführen: 27 x 426

Hierfür geben wir ein: Zahlen **27** dann Taste **A**, Zahlen **426** dann Taste **B**. Das Display zeigt das Ergebnis: 11502.

Die Taste A entspricht der  $\times$ -Taste beim Taschenrechner. Mit B haben wir die  $\ominus$ -Taste und mit C können wir eine Eingabe und die Ergebnisse löschen. Wir sollten die zu multiplizierenden Zahlen nicht zu schnell hintereinander eingeben, weil sonst die Gefahr besteht, daß einzelne Werte nicht im entsprechenden Register abgespeichert werden.

Wir geben jetzt eine Aufgabe ein, welche mit den 6 Stellen nicht mehr rechenbar ist, z.B.: 276 x 3726. Auf dem Display erscheint EEEEE für Error. (Error bedeutet Fehler, der Computer hat nicht richtig gerechnet, weil die vorhandenen 6 Stellen für das größere Ergebnis nicht ausreichen).

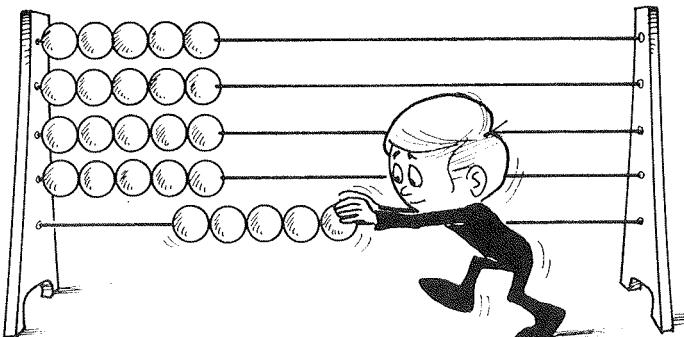
### Programm-Beschreibung

Die Befehle der Adressen 00 bis 0E stellen wieder eine Eingabe-Routine dar, wie diese bereits bei der 6-stelligen Addition besprochen wurde. Unterschiede zu diesem früheren Programm bestehen darin, daß die Eingabe der zu multiplizierenden Werte über das Register Nr. 6 (KIN 6) erfolgt, und daß diese eingegebenen Werte über die MOV-Befehle in die Register 0 bis 5 geschoben und dort auch angezeigt werden.

Bei Adresse 0F haben wir den EXRL-Befehl, mit welchem die in den Arbeitsregistern stehenden Werte in die Speicherregister gebracht werden. Nach Tastendruck A wird die nächste Eingabe ebenfalls in die Arbeitsregister 0 bis 5 übernommen. Der MULT-Befehl multipliziert dezimal die Arbeitsregister 0 bis 5 mit den Werten, die vorher in die Speicherregistern 0 bis 5 gebracht wurden. Das Ergebnis wird in die Arbeitsregister 0 bis 5 übernommen und angezeigt.

**Wie rechnet der Mikroprozessor mit dem MULT-Befehl?**  
Eine Multiplikation ist nichts anderes als eine wiederholte Addition.

Beispiel:  $5 \times 5 = 5 + 5 + 5 + 5 + 5 = 25$ .



Daß der Mikroprozessor für diese Rechen-Operation neben dem Arbeitsregister und dem Speicherregister auch noch ein Hilfsregister benötigt (welches ebenfalls durch das Betriebssystem gesteuert wird) soll uns nur am Rande interessieren, weil der Prozessor die Aufgabe automatisch ausführt. Wir erkennen jedoch, daß durch den MULT-Befehl eine Menge Programmierungsarbeit eingespart wird.

Der **MULT-Befehl** hat den Eingabe-Code **F0B**, wobei keine variablen Register anzugeben sind.

### Die Division

Das Gegenstück zur Multiplikation ist die Division. Auch hierfür hat das Betriebssystem unseres Mikroprozessors den speziellen **DIV-Befehl** mit dem Befehls-Code **F0C**. Auch für diesen Befehls-Code gibt es keine variablen Registerangaben.

Auch die Division wird sofort dezimal ausgeführt, sie ist jedoch aus internen Gründen auf vier Stellen begrenzt.

Nach HALT – NEXT – 00 geben wir das Divisions-Programm ein:

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
00	<b>F08</b>	CLEAR	
01	<b>F40</b>	DISP 4,0	
02	<b>FF6</b>	KIN 6	
03	<b>9A6</b>	CMPI A,6	
04	<b>E0D</b>	BRZ 0D	
05	<b>9B6</b>	CMPI B,6	
06	<b>E0F</b>	BRZ 0F	
07	<b>D00</b>	BRC 00	
08	<b>023</b>	MOV 2,3	
09	<b>012</b>	MOV 1,2	
0A	<b>001</b>	MOV 0,1	
0B	<b>060</b>	MOV 6,0	
0C	<b>C02</b>	GOTO 02	
0D	<b>F0D</b>	EXRL	Arbeitsregister – Speicherregister tauschen
0E	<b>C00</b>	GOTO 00	Rücksprung für 2. Eingabe
0F	<b>F0C</b>	<b>DIV</b>	Division ausführen
10	<b>E12</b>	BRZ 12	wenn Rest, dann Sprung nach 12
11	<b>C02</b>	GOTO 02	Rücksprung für Ergebnisanzeige
12	<b>FF6</b>	KIN 6	Warten f. Restwertanzeige
13	<b>F0D</b>	EXRL	Arbeitsregister – Speicherregister tauschen
14	<b>C02</b>	GOTO 02	Rücksprung für Restwertanzeige

Eingabe-Routine

Programm-Start: HALT – NEXT – 00 – RUN. Das Display zeigt: 0000.

Wir wollen zunächst folgende Aufgabe durchführen: 625 : 5.

Hierfür nehmen wir folgende Eingaben vor: Zahlen **625** dann Taste **A**, Zahl **5**, dann Taste **B**. Auf dem Display erscheint das Ergebnis: 125.

Die Taste A wird als Divisions-Taste verwendet. Taste-B ist wieder die  $\ominus$ -Taste. Mit Taste C haben wir die Lösch-Taste.

Als nächste Aufgabe rechnen wir: 19 : 5.

Das Display zeigt als Ergebnis: 0003. Das richtige Ergebnis müßte jedoch korrekt 3,8 sein. Rechnet unser Computer falsch?

Wir wollen das Ergebnis mit Taste C löschen, das Display zeigt jedoch an: 0004. Jetzt sehen wir, daß der Computer eigentlich richtig gerechnet hat:  $19 : 5 = 3$  Rest 4. Der Computer arbeitet nur mit ganzen Zahlen, Werte hinter dem Komma werden nicht weitergerechnet, sondern es wird der Restwert angezeigt. Bei erneuter Betätigung der Taste C wird der Restwert gelöscht – eine neue Aufgabe kann durchgeführt werden.

### Programm-Beschreibung

Bei den Adressen 00 bis 0C haben wir wieder eine Eingabe-Routine, die jedoch gegenüber der Multiplikation einige Programm-Schritte weniger hat, weil die Division nur 4-stellig ausgeführt wird und damit weniger MOV-Befehle notwendig sind.

Bei Adresse 0D werden durch den EXRL-Befehl die Werte der Arbeitsregister und Speicherregister ausgetauscht. Der Divident gelangt in die Speicherregister 0 bis 3, der Divisor steht in den Arbeitsregistern 0 bis 3 (Divident : Divisor = Ergebnis).

Zu beachten ist, daß bei der Division die Arbeits- und SpeicherregisterNr. 4 und 5 den Wert 0 haben müssen, was bei unserem vorstehenden Programm-Beispiel durch den CLEAR-Befehl am Programm-Anfang bewirkt wird.

Nach erfolgter Division steht das Ergebnis in den Arbeitsregistern 0 bis 3 und falls ein Rest vorhanden ist, steht dieser in den

Speicherregistern 0 bis 3. Ergibt sich ein Rest, leuchtet die ZERO-LED gleichzeitig mit der Ergebnis-Anzeige auf. Hierdurch ergibt sich ein Hinweis, daß ein Rest vorhanden ist. Bei einer Ergebnis-Anzeige ohne Rest, leuchtet die ZERO-LED nicht. Das ZERO-Flag kann dazu verwendet werden, das Programm abzufragen, ob ein Rest vorhanden ist.

Bei einigen Divisions-Aufgaben (z. B. 9999 : 1) beträgt die Rechenzeit mehrere Sekunden. Während dieser Zeit schaltet sich das Display aus, wir erkennen jedoch an der ständig leuchtenden ZERO-LED, daß der Computer arbeitet.

#### Wie führt der Mikroprozessor die Division aus?

Eine Division ist eine wiederholte Subtraktion, d. h. daß die Division schrittweise ausgeführt wird.

Beispiel:  $18 : 6 = 18 - 6 - 6 - 6 = 3 \times 6$

Kein Wunder, daß der Computer für die Aufgabe 9999 : 1 einige Sekunden benötigt. Trotzdem ist es erstaunlich, in welcher Geschwindigkeit 9999 -1 -1 -1 ... usw. gerechnet wird.

Auch bei der Division werden neben den Arbeitsregistern und Speicherregistern die im Betriebs-System zur Verfügung stehenden Hilfsregister herangezogen. Bei der vorerwähnten Aufgabe 9999 : 1 wird nicht nur 9999 mal der Wert 1 subtrahiert, sondern im Hilfsregister wird auch 9999 mal der Wert 1 hinzugefügt, damit wir am Ende durch verschiedene Manipulationen innerhalb des Mikroprozessors das Ergebnis 9999 ablesen können.

Wir müssen zwar eine zeitlang (ca. 8 Sekunden) auf das Ergebnis warten, der Mikroprozessor führt jedoch in dieser Zeit (durch das erforderliche Subtrahieren und Addieren) 19.998 Rechenvorgänge durch. Das sind 2.500 Rechenvorgänge pro Sekunde. Während der Rechenzeit steuert der Mikroprozessor das Display nicht an (daher trotz des DISP-Befehls keine Anzeige). Er fragt während der Rechenzeit auch die Tastatur nicht ab, weshalb wir die Taste HALT betätigen können, ohne daß der Computer hierauf reagiert.

Mit 2.500 Rechenvorgängen pro Sekunde setzt der Mikroprozessor mit allen übrigen Arbeiten vorübergehend aus, um möglichst schnell zum Ergebnis zu kommen. Dies führt auch dazu, daß z. B. die Uhrzeit während derartig langer Rechen-Operationen keinen Takt-Impuls erhält und somit einige Sekunden nachgeht. Es ist allerdings zu bedenken, daß die Division 9999 : 1 die längste Zeit benötigt. Bei der Division 9999 : 2 ist logischerweise nur die halbe Zeit (ca. 4 Sekunden) erforderlich und die Divisionszeit bei den üblichen normalen Aufgaben ist unerheblich.

Der DIV-Befehl bringt uns beim Programmieren ähnlich wie der MULT-Befehl eine große Erleichterung.

## Die letzten 4 Befehle – dann können wir alles!

#### Der HALT-Befehl, Befehls-Code F00

Neben der Funktionstaste HALT können wir auch direkt in einem Programm einen HALT-Befehl einbauen. Der Eingabe-Befehl ist F00. Ist dieser Befehl innerhalb eines Programms vorhanden, wird an dieser Stelle das Programm gestoppt. Auf dem Display wird die Adresse angezeigt, auf welcher der HALT-Befehl F00 eingegeben wurde. Für die Programm-Fortführung muß zuerst die STEP-Taste und dann die RUN-Taste betätigt werden.

Der HALT-Befehl kann z. B. als „Warte-Befehl“ in einem Programm vorhanden sein, er kann jedoch auch zum Austesten eines neuen Programmes verwendet werden.

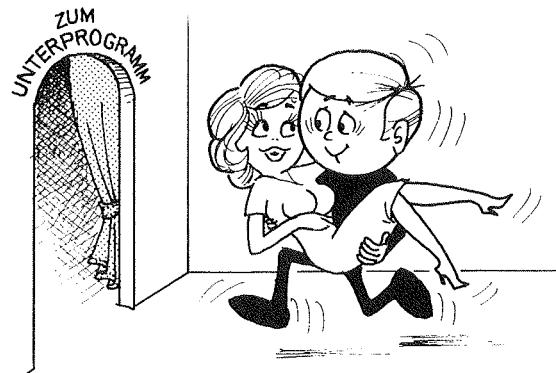
#### Der NOP-Befehl, Befehls-Code F01

Der NOP-Befehl (**no operation**) hat keinerlei Funktionen. Der NOP-Befehl ist ein Befehl, der keinen Befehl ausführt!

Was soll dann dieser Befehl?

Wenn wir längere Programme selbst erstellen und nicht sicher sind, ob evtl. nach einem ersten Programm-Test noch weitere Befehle eingefügt werden müssen, kann der NOP-Befehl eine gute Hilfe sein. Wir können nämlich beim Programmieren an uns kritisch erscheinenden Programm-Stellen einige NOP-Befehle eingeben. Wir haben dann die Möglichkeit später anstelle der NOP-Befehle die evtl. fehlenden Befehls-Eingaben vorzunehmen, ohne daß deshalb das gesamte Programm neu eingegeben werden müßte. Da die NOP-Befehle ohne Funktion sind, stört es bei einem späteren Programmlauf nicht, wenn einige unbenutzte NOP-Befehle dazwischen stehen, weil diese einfach übersprungen werden – so als ob sie nicht vorhanden wären.

Bei unserem Microtronic-Computersystem wurde an alles gedacht.



#### Mit CALL in ein Unterprogramm springen!

Bei längeren Programmen ist es möglich, daß gleiche Berechnungen an verschiedenen Programmstellen durchgeführt werden müssen. Damit diese gleichen Programmschritte nicht mehrmals an den verschiedenen Programmstellen einzufügen sind, verwenden wir für gleiche Programmschritte ein sogenanntes „Unter-Programm“:

Das Haupt-Programm wird vom Computer wie üblich abgearbeitet. Sobald die Adresse mit dem CALL-Befehl erscheint, erfolgt ein Sprung zum Unter-Programm, welches z.B. am Ende des Haupt-Programms im Programm-Speicher stehen kann.

CALL ist also ähnlich wie GOTO ein Sprung-Befehl, jedoch beim CALL-Befehl wird im Mikroprozessor dessen Adresse gespeichert.

Das Unter-Programm wird ebenfalls normal abgearbeitet, bis am Ende des Unter-Programms der Befehl RET (return) erscheint. Durch RET erfolgt ein Rücksprung zum Haupt-Programm. Da der Mikroprozessor die Adresse des CALL-Befehls registriert hat, wird im Haupt-Programm automatisch die nach dem CALL-Befehl folgende Adresse angesteuert und das Haupt-Programm kann weiter gearbeitet werden. Bei jedem CALL-Befehl wird die CALL-Adresse im Mikroprozessor neu gespeichert, wodurch es möglich ist, das Unter-Programm von verschiedenen Stellen des Haupt-Programmes aus anzuspringen. Genauso ist es möglich, vom Unter-Programm immer wieder an die verschiedenen Stellen des Haupt-Programms zurückzuspringen.

Mit CALL können auch verschiedene (mehrere) Unter-Programme angesteuert werden, wobei jedoch mit RET zuerst ein Rücksprung zum Haupt-Programm erforderlich ist. Ein Sprung von einem Unter-Programm direkt zu einem anderen ist nicht möglich.

Unter-Programme werden auch „Subroutine“ genannt. Durch eine derartige Subroutine kann ein Programm kürzer gehalten werden, weil die Wiederholung immer wiederkehrender gleicher Programmschritte nicht notwendig ist.

### **Der CALL-Befehl, Befehls-Code: Baa**

CALL kann aus dem englischen mit „rufen“ übersetzt werden. Es ist der Adressen-Aufruf zum Sprung in ein Unter-Programm. Der Befehls-Code ist **Baa**, wobei für **aa** die Start-Adresse des Unter-Programms eingegeben wird.

### **Der RET-Befehl, Befehls-Code F07**

RET kommt aus dem englischen **return** und kann mit „zurück“ übersetzt werden. Mit der Befehls-Eingabe F07 erfolgt der Rücksprung vom Unter-Programm automatisch zur nächsten nach dem CALL-Befehl folgenden Programm-Adresse.

Ein Unter-Programm wird uns im nächsten Kapitel gezeigt.

## **Wie wird ein Computer-Spiel programmiert?**

### **Beispiel: Das Nimm-Spiel**

Bei unseren ersten Computer-Erfahrungen haben wir das Nimm-Spiel kennengelernt. Da die Programmierung solcher Spiele sehr interessant ist, wollen wir das Nimm-Spiel-Programm einmal genauer untersuchen.

Wie bekannt, ist das Nimm-Spiel als Fest-Programm im Mikroprozessor gespeichert. Mit HALT – PGM – 7 können wir das Nimm-Spiel in den für uns zugänglichen Programm-Speicher einspielen.

**Achtung:** Durch Überspielen des Nimm-Spiels werden andere in unserem Programm-Speicher vorhandene Programme bis zur Adresse 44 gelöscht.

Mit HALT – NEXT – 00 gehen wir zum Programm-Anfang und wir können jetzt mit der NEXT-Taste das Programm ansehen und mit der Programm-Tabelle dieses Kapitels vergleichen.

Wenn wir selbst programmieren und ein derartiges Programm entwickeln möchten, müssen wir uns am Anfang mit der „Spiel-Theorie“ beschäftigen.

Das Nimm-Spiel ist ein zwar einfaches, aber trotzdem interessantes Spiel für 2 Personen. In der sogenannten „Spiel-Theorie“ wird dieses Spiel als „2-Personen-Nullsummen-Spiel mit vollständiger Information und optimaler Strategie“ bezeichnet.

„Nullsummen-Spiel“ bedeutet, daß es grundsätzlich einen Gewinner und einen Verlierer gibt. Ein Unentschieden ist nicht möglich.

„Vollständige Information“ bedeutet, daß es z. B. keine verdeckten Karten gibt, sondern beide Mitspieler haben während der gesamten Spielzeit die gleichen Informationen.

„Optimale Strategie“ bedeutet, daß es für den Spieler einen Weg gibt, der grundsätzlich zum Ziel führen muß. Hierdurch ist es relativ einfach möglich, für ein solches Spiel ein Programm zu entwickeln.

Die Hauptschwierigkeit ist es, die „optimale Strategie“ zu finden. Für dieses Nimm-Spiel ist die Strategie noch verhältnismäßig leicht zu finden. Bei dem im zweiten Teil des Anleitungsbuches folgenden Spiel „Nimm-2“ ist die Findung einer optimalen Strategie bereits recht schwierig.

### **Wie finden wir die „optimale Strategie“?**

Hierfür ist es notwendig, ein Spiel in einzelne Spielabschnitte zu zerlegen. Am besten ist es sogar, mit den Überlegungen am Spielende zu beginnen. Für unsere Überlegungen gehen wir davon aus, daß die beiden Mitspieler A und B gegeneinander spielen, und daß A gewinnen soll. Bei Spielbeginn soll ein Häufchen mit 15 Streichhölzern zur Verfügung stehen. Minimal 1 oder maximal 3 Hölzchen dürfen auf einmal weggenommen werden.

Nachdem der Spieler B verloren hat, muß B das letzte Hölzchen wegnehmen. Der Spieler A muß seine Strategie so einrichten, daß B bei seinem vorletzten Zug maximal 4 Hölzchen, mindestens jedoch 2 Hölzchen liegenlassen muß. Spieler A wird also versuchen, daß nach seinem Zug 5 Hölzchen übrigbleiben. Nimmt anschließend B 3 Hölzchen weg, bleiben 2

übrig, A nimmt 1 und B verliert mit dem letzten Hölzchen. Oder B nimmt von den 5 Hölzchen 1 weg, von den 4 übrigbleibenden nimmt A 3 und B verliert mit dem letzten Hölzchen.

Somit ist die Ziffer 5 eine Gewinnzahl. Der Spieler, welcher nach seinem Zug 5 Hölzchen übrig läßt, hat bereits gewonnen. Die nächst folgende Gewinnzahl ist 1, denn wer das letzte Hölzchen übergelassen kann gewinnt. Die Differenz aus den Gewinnzahlen 5 und 1 ist 4. Addieren wir zur Gewinnzahl 5 die Differenzzahl 4, ergibt sich 9 als nächste Gewinnzahl. Wer 9 Hölzchen übrig läßt und die Spiel-Strategie kennt – wird gewinnen. Die nächste Gewinnzahl ergibt sich aus 9 und 4 = 13.

Die Gewinnzahlen, die der Spieler A versuchen muß zu erreichen, sind also  $13 - 9 - 5 - 1$ . Man kann diese Gewinnzahlen zerlegen:

$$\begin{aligned} 05 &= 4 + 1 \\ 09 &= 2 \times 4 + 1 \\ 13 &= 3 \times 4 + 1 \end{aligned}$$

Auch die Zahl 4 hat einen Ursprung: Minimal muß 1 Hölzchen, maximal dürfen 3 Hölzchen weggenommen werden:  $3 + 1 = 4$ .

Man kann die Gewinnzahl auch mit Hilfe einer Formel berechnen.

Hierfür bezeichnen wir Gewinnzahl mit **GZ**, maximal mit **MAX** und minimal mit **MIN**:

$$GZ = 0 \times (\text{MAX} + \text{MIN}) + 1$$

Mit dieser Formel können auch andere Gewinnzahlen ermittelt werden, z. B. wenn maximal 6 Hölzchen und minimal 1 Hölzchen genommen werden können:

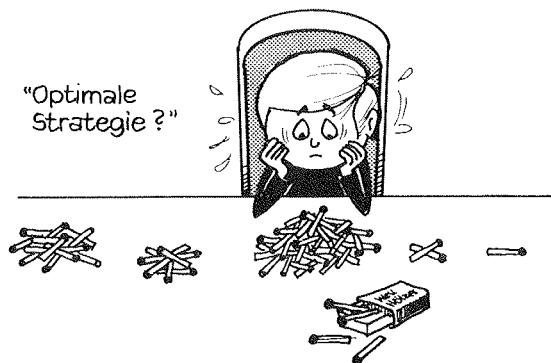
$$\begin{aligned} GZ &= 0 \times (6 + 1) + 1 = 1 \\ GZ &= 1 \times (6 + 1) + 1 = 8 \\ GZ &= 2 \times (6 + 1) + 1 = 15 \\ GZ &= 3 \times (6 + 1) + 1 = 22 \text{ usw.} \end{aligned}$$

Während eines derartigen Spiels, wird niemand solche Berechnungen ausführen. Soll jedoch ein derartiges Spiel gegen den Computer gespielt werden, muß ihm geholfen werden, die richtigen Berechnungen auszuführen.

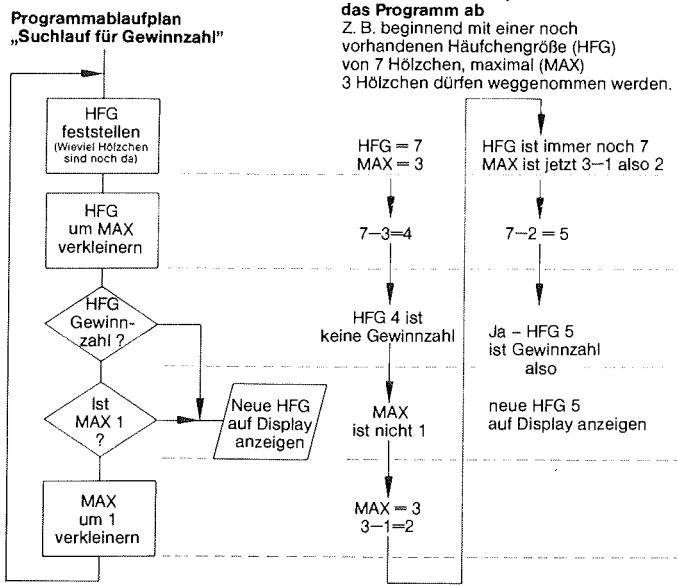
Für die Entwicklung eines solchen Programmes beginnt man zweckmäßig beim schwierigsten Programmteil, indem zunächst ein Programm-Ablaufplan skizziert wird. Wir müssen uns zunächst also einmal überlegen, wie der Computer eine Gewinnzahl finden kann.

Davon ausgehend, daß auf dem Häufchen maximal 15 Streichhölzer zur Verfügung stehen, und daß maximal 3, bzw. minimal 1 Hölzchen weggenommen werden darf, lassen wir den Computer zweckmäßigerweise nach folgendem Schema arbeiten:

Er nimmt (für seine internen schnellen Berechnungen) die maximal zulässige Zahl (3). Wird hierdurch keine Gewinnzahl erreicht, wird der maximale Wert um 1 Stück reduziert, d. h. der Computer rechnet, ob er mit 2 eine Gewinnzahl erreicht. Wenn nicht, wird abermals der Wert 1 reduziert und entweder wird jetzt eine Gewinnzahl gefunden, oder der minimale Wegnahmewert ist erreicht. Diese Methode hat den Vorteil, daß der Computer, falls er keine Gewinnzahl findet, vom zur Verfügung stehenden Häufchen immer nur 1 Hölzchen wegnimmt und damit das Spiel verzögert, um evtl. bei seinem nächsten Zug die Gewinnzahl zu erreichen.



Die Abbildung zeigt einen einfachen Programm-Ablaufplan wie die Gewinnzahl gesucht, und evtl. gefunden werden kann. Rechts daneben finden wir eine Darstellung, wie der Computer das Programm abarbeitet. Der Einfachheit halber gehen wir davon aus, daß zu Beginn dieses „Suchlaufs“ eine Häufchengröße (HFG) von 7 Hölzchen noch vorhanden ist.



Aus der rechten Darstellung ersehen wir, daß der Computer den Programm-Ablaufplan durchläuft, alsdann zurückspringt, wobei er MAX für seine interne Berechnung von ursprünglich 3 auf 2 Stück reduziert.

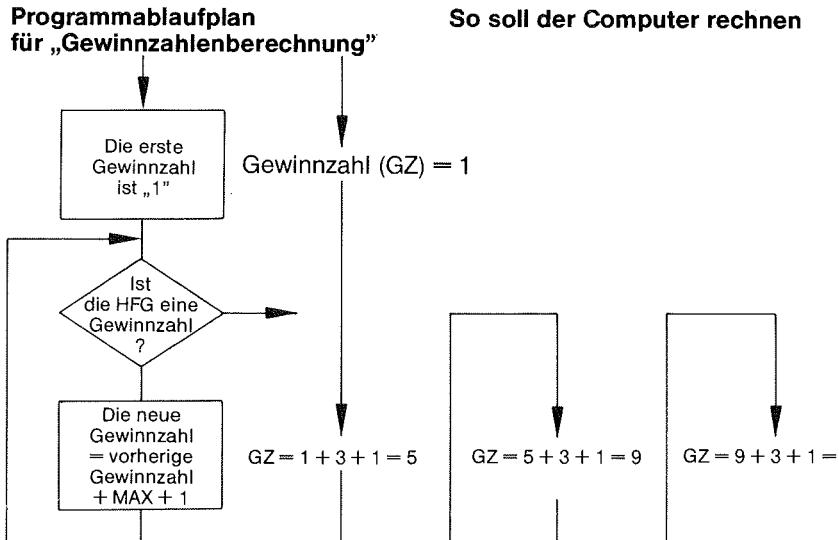
Der erste Teil unserer Überlegungen wie man ein solches Nimm-Spiel programmieren könnte ist damit erledigt.

Schön wär's – wir haben jedoch noch eine Kleinigkeit vergessen. Wir fragten nämlich im Programm-Ablaufplan, ob die HFG (Häufchengröße) eine Gewinnzahl ist. Kennt der Computer bereits die möglichen Gewinnzahlen?

Er kennt sich nicht – leider! Also müssen wir ihm helfen. Es bieten sich 2 Möglichkeiten an:

Wir könnten dem Computer eine Liste mit allen möglichen Gewinnzahlen eingeben. Hierfür wäre jedoch sehr viel Platz im Programm-Speicher erforderlich.

Obwohl es umständlich erscheint, ist es einfacher den Computer die Gewinnzahlen selbst berechnen zu lassen. Also machen wir uns Gedanken, wie ein weiterer Programm-Ablaufplan aussehen könnte: „Berechnung der Gewinnzahlen“ wobei ein Vergleich vorzusehen ist, ob die HFG (Häufchengröße) eine Gewinnzahl ist.



Wir wissen, daß die Gewinnzahlen 1, 5, 9 oder 13 sind. Wir wissen auch, wie die Gewinnzahlen errechnet werden können, wenn am Ende 1 Hölzchen für den Verlierer übrigbleiben soll, 3 maximal weggenommen werden dürfen und wenn man selbst (oder der Computer) ebenfalls 1 Hölzchen wegnimmt:  $1+3+1=5$  usw.

Also machen wir uns einen kleinen Programm-Ablaufplan und vermerken daneben wie der Computer rechnen soll:

Nebenstehende Darstellung zeigt uns, daß wir auch diese Aufgabe erfolgreich lösen können. Wenn wir uns die Darstellung allerdings etwas genauer betrachten, müssen wir feststellen, daß wir eine „Endlos-Programm-Schleife“ entwickelt haben, in welcher der Computer bis in alle Ewigkeit Gewinnzahlen ermitteln wird.

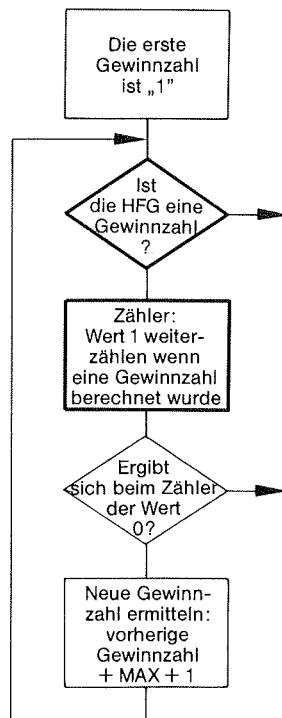
Was können wir tun, um aus dieser Endlos-Schleife wieder herauszukommen, wenn genügend Gewinnzahlen berechnet wurden?

Wir müssen dem Computer die Gelegenheit geben zu zählen, wie oft er die Gewinnzahlen schon berechnet hat. Wir bauen in das Programm einen „Zähler“ ein, bei welchem jedesmal der Wert 1 hinzugezählt wird, sobald der Computer wieder eine neue Gewinnzahl ermittelt hat.

Der vorangegangene Programm-Ablaufplan „Gewinnzahlen-Berechnung“ wird wie folgt durch einen zusätzlichen „Zähler“ erweitert.

Wir haben den „Zähler“ in den bisherigen Programm-Ablaufplan für die Gewinnzahlen-Berechnung dazwischen geschoben. Wenn die Häufchengröße (HFG) nicht gleichzeitig auch eine Gewinnzahl ist, kommt das Programm zum „Zähler“. Bei jeder neuen Gewinnzahl-Ermittlung wird im Zähler der Wert 1 solange hinzugezählt, bis der hexadezimale Wert F übersprungen wird. Hierdurch ergibt sich ein Übertrag des Zählers (CARRY-FLAG wird gesetzt) und der neue Zählerstand ist 0. Durch den Übertrag (CARRY-FLAG) können wir die „Zähl-Schleife“ durch einen BRZ-Befehl oder durch einen BRC-Befehl verlassen.

### Programmablaufplan „Gewinnzahnberechnung“ mit „Zähler“



**Der Computer stellt uns immer wieder vor neue Denksport-Aufgaben. Eine Aufforderung, unser logisches Denkvermögen unter Beweis zu stellen. Wir sollten erkennen, daß es genau diese Herausforderung ist, welche den Umgang mit einem Computer so interessant macht.**

Nachdem wir durch die Ablaufpläne und die dazugehörigen Überlegungen das Programm für ein Nimm-Spiel prinzipiell geklärt haben, müssen wir uns überlegen, wie und mit welchen Registern gezählt und gespeichert werden soll. Wir müssen z. B. auch berücksichtigen, daß am Anfang die Häufchengröße (HFG) 15 Hölzchen umfaßt, und daß hierfür eine 2-stellige Anzeige notwendig ist. Es bleibt uns freigestellt, welchen Registern die einzelnen Aufgaben zugeordnet werden, z. B.:

Reg. Nr.		Registerbelegung
0		Ausgabe an Ausgänge (z.B. Piezo-Summer)
1	Anzeige auf Display	HFG – Häufchengröße Einerstelle
2		HFG – Häufchengröße Zehnerstelle
3		– Frei –
4		Rechen-Register: wieviel Hölzchen nimmt Computer?
5		Speicher-Register für genommene Hölzchen
6		Interne Berechnung: MAX + 1
7		– Frei –
8		– Frei –
9		– Frei –
A		„Gewinnzahl-Berechnung“
B		„Zähl-Register“
C		Rechenregister für hexadezimale Berechnungen und Umwandlung Dez. in Hex. und Hex. in Dez.

Sobald die entsprechende Register-Einteilung feststeht, wird zuerst ein endgültiger Programm-Ablaufplan gezeichnet. Es wäre falsch, ohne Programm-Ablaufplan zu versuchen, bereits jetzt die einzelnen Programm-Schritte festzulegen, weil solche Programme sehr schnell unübersichtlich werden und bei evtl. Fehlern kaum noch korrigiert werden können. Es passiert auch zu leicht, daß wichtige Programm-Schritte vergessen werden, was zu einer längeren Fehlersuche führen kann.

Im vorangegangenen Kapitel haben wir erfahren, daß für mehrfach wiederkehrende Berechnungen innerhalb eines Programms ein „Unter-Programm“ verwendet wird. Auch bei unserem Nimm-Spiel haben wir an zwei Programm-Stellen die Anzahl der Hölzchen (welche weggenommen werden), auszurechnen. Hierfür ist in Register 1 und Register 2 eine Subtraktion notwendig. Also verwenden wir ein spezielles Subtraktions-Unter-Programm.

Wir sollten auch daran denken, daß alle Berechnung hexadezimal durchgeführt werden und erst für die Anzeige auf dem Display in Dezimalzahlen umzuwandeln sind.

Ist der endgültige Programm-Ablaufplan gezeichnet, wird er mit einem Zahlenbeispiel theoretisch „durchgespielt“. Oft findet man auf diese Weise schon die ersten Fehler und der Ablaufplan kann noch korrigiert werden.

Erst jetzt sollte man damit beginnen, die einzelnen Programm-Schritte festzulegen und in eine Programm-Tabelle einzutragen. (Solche Programm-Listen sind als Zubehör dem Microtronic-Computer beigelegt).

Für das „Schreiben“ des Programms geht man Schritt für Schritt vor, indem z. B. zuerst die Eingabe-Routine festgelegt und ausgetestet wird. Dann folgt der nächste Programm-Teil usw. bis das Programm fertig ist.

In gleicher Weise ist das Programm für unser Nimm-Spiel entwickelt worden und wie bekannt, können wir es mit dem Tasten-Aufruf HALT – PGM – 7 in unserem Programm-Speicher

automatisch einspielen. Ist dies geschehen sehen wir, daß der letzte Programm-Schritt die Adresse 44 angezeigt wird. Mit HALT – NEXT – 00 springen wir zum Programm-Anfang und starten wie üblich mit RUN. Wenn wir jetzt nochmals das Nimm-Spiel spielen, kommt uns zum Bewußtsein, welche Gedankenarbeit in einem solchen Programm-Ablauf enthalten ist.

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
00	<b>F08</b>	CLEAR	Alle Register löschen
01	<b>FE0</b>	DOT 0	Ausgänge auf Null setzen
02	<b>F41</b>	DISP 4,1	Reg. 1 – Reg 4 anzeigen
03	<b>FF2</b>	KIN 2	Eingabe „Haufengröße – Zehnerstelle“
04	<b>FF1</b>	KIN 1	Eingabe „Haufengröße – Einerstelle“
05	<b>FF4</b>	KIN 4	Eingabe wieviele Hölzchen max. weggenommen werden dürfen
06	<b>045</b>	MOV 4,5	R4 → R5
07	<b>046</b>	MOV 4,6	R4 → R6
08	<b>516</b>	ADDI 1,6	R6 + 1 = R6
09	<b>FF4</b>	KIN 4	Eingabe: Wieviel Hölzchen nehmen. (Abk.: „IN“)
0A	<b>854</b>	CMP 5,4	Ist „IN“ größer als „MAX“
0B	<b>D19</b>	BRC 19	Ja, Sprung nach 19
0C	<b>904</b>	CMPI 0,4	Ist „IN“ gleich Null?
0D	<b>E19</b>	BRZ 19	Ja, Sprung nach 19
0E	<b>B3F</b>	CALL 3F	Aufruf des Unterprogrammes (Adresse 3F)
0F	<b>F03</b>	HDXZ	HEXziffer → DEZziffer
01	<b>0D1</b>	MOV D,1	Reg D → 1 neu berechnete
11	<b>0E2</b>	MOV E,2	Reg E → 2 Haufengröße
12	<b>911</b>	CMPI 1,1	HFG – Einerstelle = 1?
13	<b>E15</b>	BRZ 15	Ja, Sprung nach 15
14	<b>C1A</b>	GOTO 1A	Nein, Sprung nach 1A
15	<b>902</b>	CMPI 0,2	HFG – Zehnerstelle = 0?
16	<b>D1A</b>	BRC 1A	Nein, Sprung nach 1A
17	<b>1F0</b>	MOVI F,0	Spieler hat gewonnen, F in Reg. 0
18	<b>FE0</b>	DOT 0	F (Reg. 0) auf Ausgänge (LED's leuchten)
19	<b>F00</b>	HALT	Programm stoppen
1A	<b>F02</b>	DISOUT	Anzeige ausschalten
1B	<b>064</b>	MOV 6,4	Reg. 6 → Reg. 4 (= MAX + 1)
1C	<b>10C</b>	MOVI 0,C	0 → Reg C
1D	<b>714</b>	SUBI 1,4	Reg. 4 - 1 = Reg. 4 (= MAX)
1E	<b>B3F</b>	CALL 3F	Aufruf des Unterprogramms „Subtraktion“
1F	<b>11A</b>	MOVI 1,A	1 in Reg. A
20	<b>10B</b>	MOVI 0,B	0 in Reg. B
21	<b>C24</b>	GOTO 24	Sprung nach 24
22	<b>46A</b>	ADD 6,A	Reg. 4 + Reg. A = Reg. 4
23	<b>FBB</b>	ADC B	Übertrag in Reg. B
24	<b>8AD</b>	CMP A,D	Reg. A = Reg. D?
25	<b>E27</b>	BRZ 27	Ja, Sprung nach 27
26	<b>C29</b>	GOTO 29	Nein, Sprung nach 29
27	<b>8BE</b>	CMP B,E	Reg. B = Reg. E?
28	<b>E2F</b>	BRZ 2F	Ja, Sprung nach 2F
29	<b>51C</b>	ADDI 1,C	Reg. C + 1 = Reg. C
2A	<b>E2C</b>	BRZ 2,C	Reg. C = 0? Ja, Sprung nach 2C
2B	<b>C22</b>	GOTO 22	Nein, Sprung nach 22
2C	<b>914</b>	CMPI 1,4	Reg. 4 = 1?
2D	<b>E2F</b>	BRZ 2F	Ja, Sprung nach 2F
2E	<b>C1C</b>	GOTO 1C	Nein, Sprung nach 1C
2F	<b>F03</b>	HDXZ	HEXziffer → DEZziffer Umwandlung

Adresse	Eingabe-Befehl	Mnemonic	Erklärungen
30	<b>0D1</b>	MOV D,1	Reg. D→Reg. 1 neu berechnete
31	<b>0E2</b>	MOV E,2	Reg. E→Reg. 2 Häufengröße
32	<b>F41</b>	DISP 4,1	Reg. 1 – Reg. 4 anzeigen
33	<b>902</b>	CMPI 0,2	Reg. 2 = 0?
34	<b>D09</b>	BRC 09	Nein, Sprung nach 09
35	<b>911</b>	CMPI 1,1	Reg. 1 = 1?
36	<b>E38</b>	BRZ 38	Ja, Sprung nach 38
37	<b>C09</b>	GOTO 09	Nein, Sprung nach 09
38	<b>1E2</b>	MOVI E,2	Spielende - Computer hat gewonnen
39	<b>1E3</b>	MOVI E,3	
3A	<b>1F5</b>	MOVI F, 5	
3B	<b>FE5</b>	DOT 5	
3C	<b>105</b>	MOVI 0,5	
3D	<b>FE5</b>	DOT 5	
3E	<b>C3A</b>	GOTO 3A	
3F	<b>01D</b>	MOV 1,D	Unterprogramm: Reg. 1 → Reg D
40	<b>02E</b>	MOV 2,E	Reg. 2 → Reg. E
41	<b>F04</b>	DZXH	DEZiffer→HEXziffer Umwandlung
42	<b>64D</b>	SUB 4,D	Reg. D – Reg. 4 = Reg. D
43	<b>FCE</b>	SUB CE	Übertrag von Reg. E abziehen
44	<b>F07</b>	RET	Rücksprung in das Hauptprogramm

Wenn wir uns mit den jetzt gewonnenen Erkenntnissen den Ablaufplan ansehen und mit der Programm-Tabelle vergleichen, wird uns vielleicht manches noch etwas schwierig erscheinen. Hierbei sollten wir bedenken, daß dieses Programm-Beispiel vom Entwickler sehr kritisch bearbeitet wurde. Nachdem es sich um ein Fest-Programm handelt, wurden die einzelnen Programmschritte auf das kleinstmögliche Maß gekürzt, um mit möglichst wenig Speicher-Kapazität auszukommen.

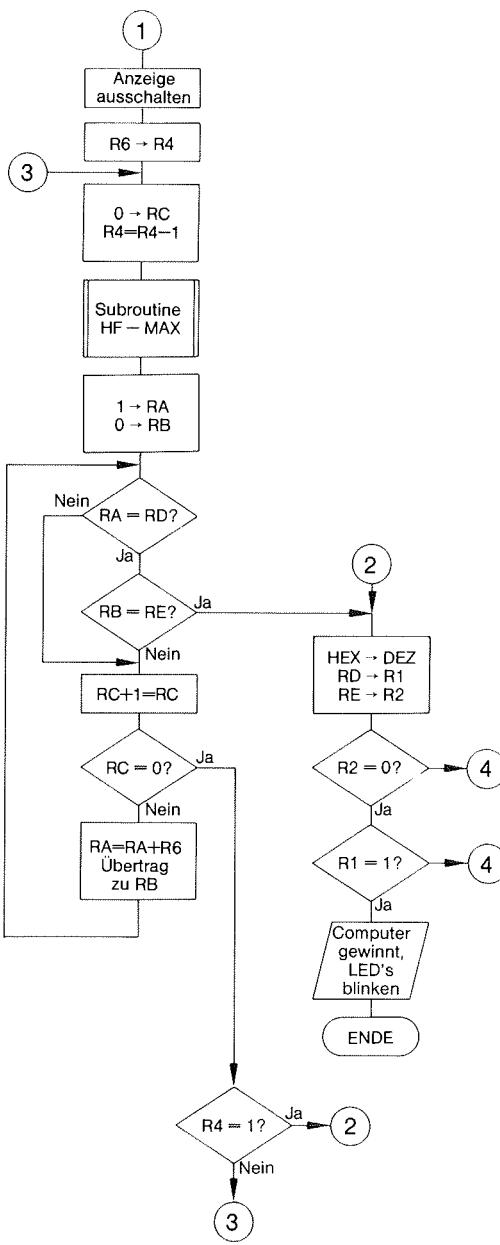
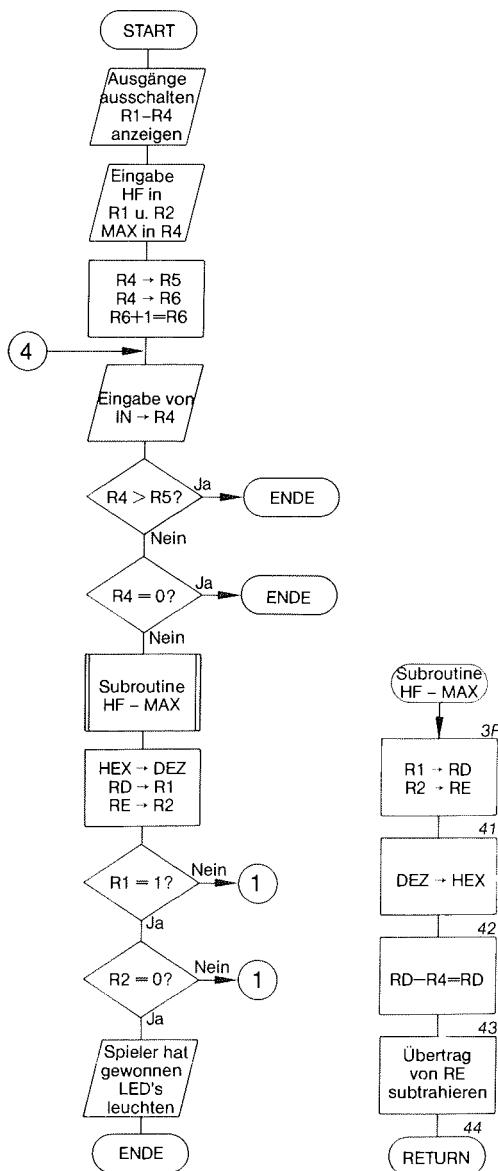
Wenn wir selbst programmieren, sollten wir auch wissen, daß z. B. ein derartiges Nimm-Spiel zu den fortgeschrittenen Programmierungs-Aufgaben gehört. Für das Selbst programmieren sollten wir mit möglichst einfachen Aufgaben beginnen, bis die ersten Erfahrungen mit der richtigen Verwendung der einzelnen Befehls-Codes gesammelt wurden.

Es ist oft schwierig Programm-Abläufe zu beschreiben. Einfacher ist es, selbsterdachte Programm-Aufgaben in die Computer-Logik umzusetzen.

### Programm-Beschreibung:

Bei Adresse 03 bis 05 ergibt sich die Eingabe der Häufchengröße (z. B. 15) und von MAX (wieviel Hölzchen maximal weggenommen werden dürfen: z. B. 3). MAX wird in den Registern 4 und 5 gespeichert. Außerdem wird der Wert MAX+1 in Register 6 gespeichert, weil dies für die Gewinnzahlen-Ermittlung notwendig ist.

In Register 4 wird die Anzahl der wegzunehmenden Hölzchen des Mitspielers übernommen.



Anschließend erfolgt die Kontrolle, ob der Spieler die zulässige Hölzchenzahl (1 bis 3) eingegeben hat. Mögelt der Spieler, wird das Programm durch HALT gestoppt. Andernfalls wird im Unter-Programm „Subtraktion“ (bei Adresse 3F) die eingegebene Zahl von der Häufchengröße abgezogen. Die Subtraktion erfolgt hexadezimal in den Registern D und E. Hier ergibt sich auch die Umwandlung der hexadezimalen Werte in dezimale Zahlen. Anschließend erfolgt der Vergleich, ob die Häufchengröße mittlerweile den Wert Ziffer 1 erreicht hat, wenn ja: Computer hat verloren, wenn nein: Der Computer ist am Zug und das Programm wird bei Adresse 1A fortgesetzt.

Während der Computer versucht das Ergebnis auszurechnen, bleibt die momentane Häufchengröße solange in den Registern 1 und 2 gespeichert, bis er sich entschlossen hat, wieviel Hölzchen er wegzunehmen gedenkt, womit die neue Häufchengröße ermittelt wird.

### **Jetzt wird es spannend: Wir programmieren selbst!**

Alle Befehle und Funktionen unseres Microtronic-Computersystems haben wir kennengelernt. Auch die wesentlichsten Programmiermöglichkeiten wurden behandelt. Der Zeitpunkt ist gekommen, um mit dem Selbst-Programmieren endgültig zu beginnen.

Hierbei wird das Buchzeichen unser wichtigster Helfer werden. Alle Befehle und Befehls-Möglichkeiten sind aufgeführt und wir finden dort auch die wichtigsten Umrechnungstabellen der Zahlen-Systeme.

Alle Einzel-Befehle und deren Funktionen sind am Ende des 1. Teils des Anleitungsbuches noch einmal ausführlich beschrieben. Diese Zusammenfassung sollten wir beim Selbst-Programmieren noch einmal durchlesen. Dabei werden wir noch viele interessante Hinweise (auch über CARRY- und ZERO-FLAG) erhalten.

Wenn wir an den Computer Ein- oder Ausgängen eine zusätzliche Elektronik anschließen möchten, sind hierfür die BUSCH-Electronic-Kästen 2060, 2065, 2070 und 2075 speziell geeignet. Für solche hochinteressanten Experimente sind elek-

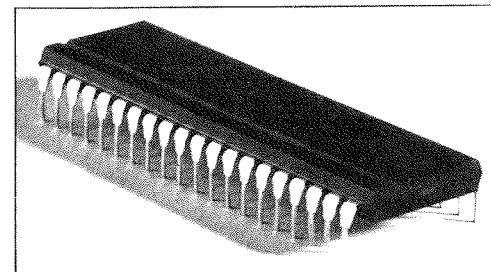
tronische Grundkenntnisse erforderlich, die uns mit den genannten BUSCH-Electronic-Studios vermittelt werden. Im zweiten Teil des Anleitungsbuches sind mehrere Programme in Verbindung mit zusätzlich aufgebauten elektronischen Schaltungen enthalten, wodurch wir viele Anregungen erhalten, wie eine Peripherie Elektronik eingesetzt werden kann. Wir sollten jedoch immer beachten, daß niemals die volle 9 V Batterie-Spannung an den Eingängen oder Ausgängen des Computers angeschlossen wird. Die Batterie-Spannung muß durch entsprechende Widerstände reduziert werden.

### **Das „Innenleben“ eines Computers**

Nachdem wir einen Teil der Möglichkeiten unseres Microtronic-Computersystems kennen, interessiert uns sicherlich auch das Innenleben eines Mikrocomputers.

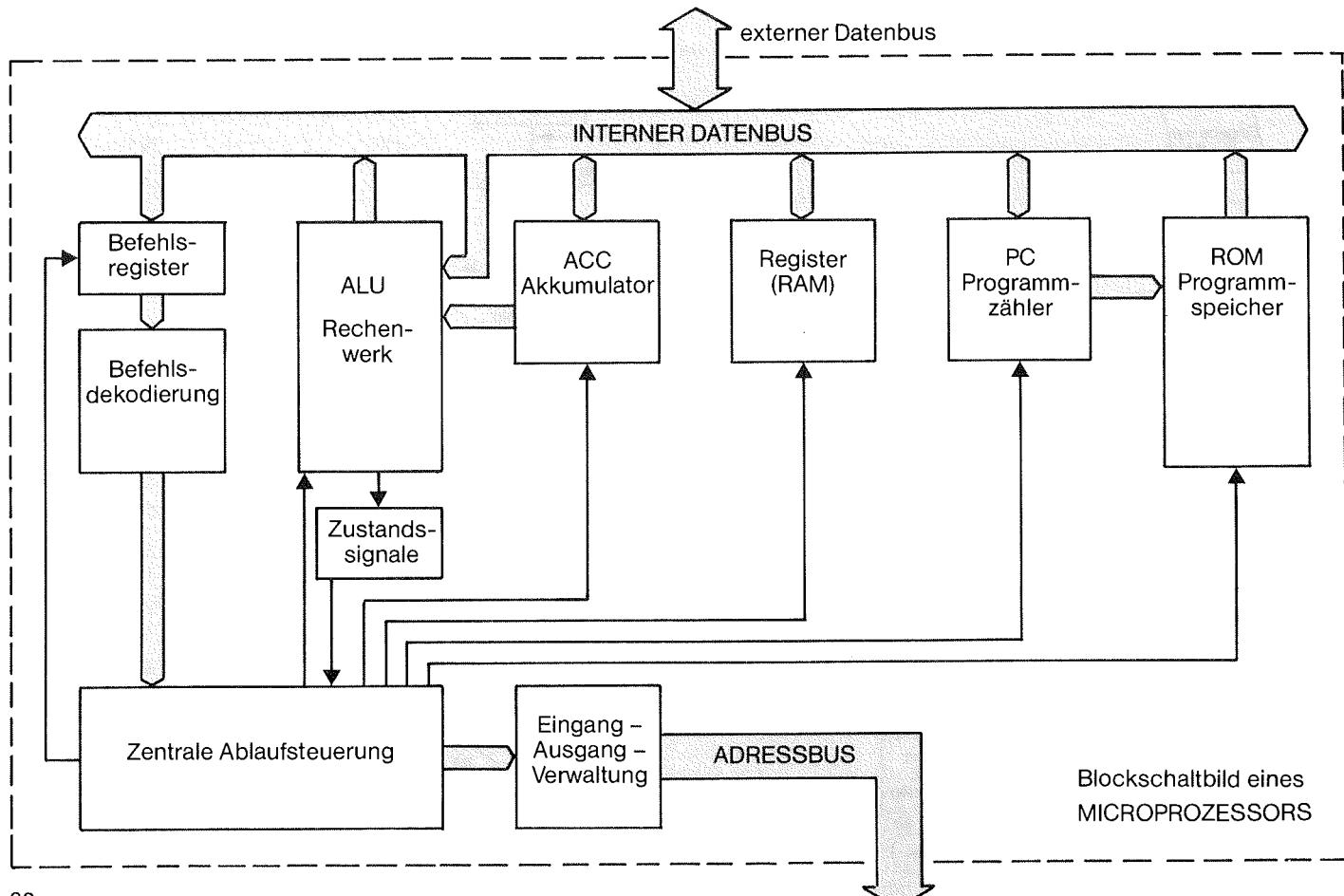
#### **Seele und Gehirn: Der Mikroprozessor**

Der Mikroprozessor ist ein hochkomplexes elektronisches Schalt- und Speicherwerk, mit mehreren Funktionseinheiten. Er hat eine ganze Reihe von Registern, um Daten zu speichern.



Unscheinbarer  
schwarzer IC-Block:  
Der Mikro-Prozessor

Unser Mikroprozessor hat 128 Register. Durch die verschiedenen Befehls-Codes können wir jedoch nur 32 dieser Register (16 Arbeits- und 16 Speicher-Register) verwenden. Die verbleibenden 96 Register sind für interne Berechnungen des Prozessors notwendig.



Alle 128 Register stellen einen **RAM-Speicher** dar. **RAM** (aus dem englischen „random access memory“) bedeutet einen „Schreib- und Lese-Speicher mit ständigem Zugriff“, weil wir jederzeit Daten in das RAM einspeichern oder herausholen (auslesen) können. Den RAM-Speicher kann man auch als RAM-Register bezeichnen.

Der „**Akkumulator**“ (Abkürzung **ACC**) ist ebenfalls ein spezielles Register, in welchem alle Berechnungen durchgeführt werden. Nach Ausführung einer Berechnung steht das Ergebnis zunächst im Akkumulator und wird von dort aus zu verschiedenen anderen Registern weitergegeben.

Das „**Rechenwerk**“ des Mikroprozessors ist die sogenannte **ALU** („arithmetic logic unit“), welches auch mit Arithmetik und Logikeinheit übersetzt werden kann. In der ALU werden Berechnungen, logische Verknüpfungen, Vergleiche usw. ausgeführt. An der ALU sind auch die „**Zustands-Signale**“ angeschlossen. Dort werden die „Zustände“ der CARRY- und ZERO-FLAGS gespeichert.

Der „**Programm-Zähler**“ **PC** (program counter) zählt die ausgeführten Befehle und liefert jeweils die Adresse für den nächsten auszuführenden Befehl.

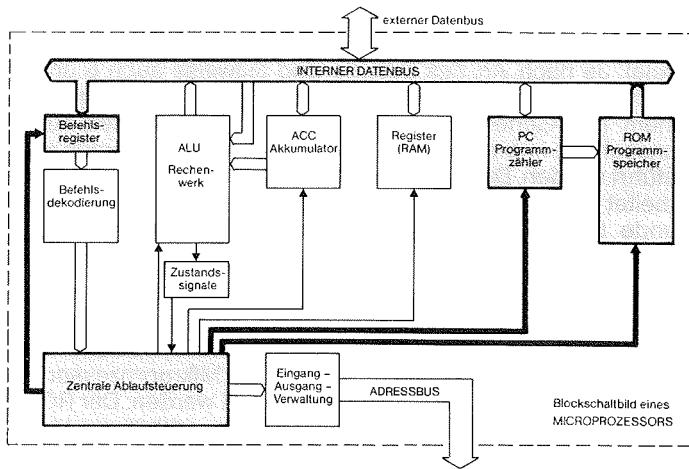
Die Befehle stehen in einem **ROM Speicher** (read only memory). Aus dem ROM-Speicher können Daten ausgelesen werden, es können jedoch keine neuen Daten eingespeichert werden. Der ROM-Speicher unseres Mikroprozessors enthält das Monitor-Programm (Betriebs-System) des Computers und alle hierfür erforderlichen Daten sind fest programmiert (nicht veränderbar).

Der Mikroprozessor enthält außerdem ein „**Befehls-Register**“ (instruction register) für eine Zwischenspeicherung eines momentan auszuführenden Befehls.

Das Befehls-Register arbeitet direkt mit der „**Befehls-Dekodierung**“ (instruction decoder) zusammen. Der jeweilige Befehl wird ausgewertet und die hieraus resultierenden Erkenntnisse an die „**zentrale Ablaufsteuerung**“ weitergeleitet. Die zentrale Ablaufsteuerung kontrolliert den gesamten Datenverkehr, die Befehlausführung usw. innerhalb des Mikroprozessors und gibt über die „**Eingabe-/Ausgabe-Verwaltung**“ die entsprechenden Kommandos, auch an die angeschlossenen Komponenten wie z. B. Display usw. weiter.

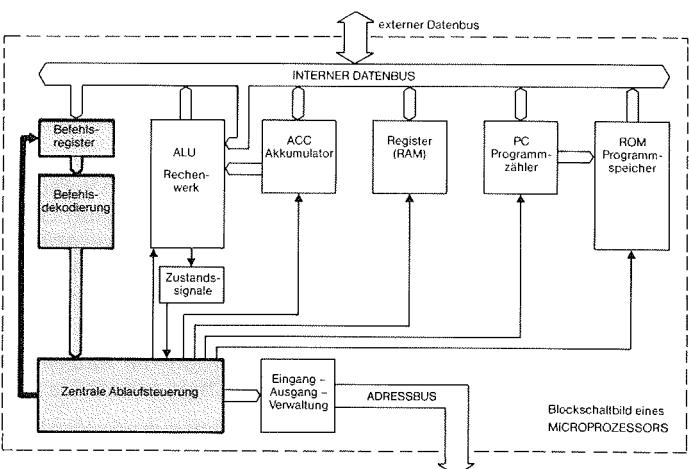
Alle Einheiten des Mikroprozessors sind durch einen **internen Daten-Bus** miteinander verbunden. Am internen Daten-Bus ist wiederum der **externe Daten-Bus** angeschlossen, durch welchen die übrigen Computer-Komponenten angesteuert werden. (Siehe auch Blockschaltbild eines Mikroprozessors).

Die folgenden 4 Darstellungen zeigen, was im Mikroprozessor vor sich geht, wenn z. B. innerhalb eines Programms durch einen Vergleichs-Befehl zwei Zahlen miteinander verglichen werden sollen.



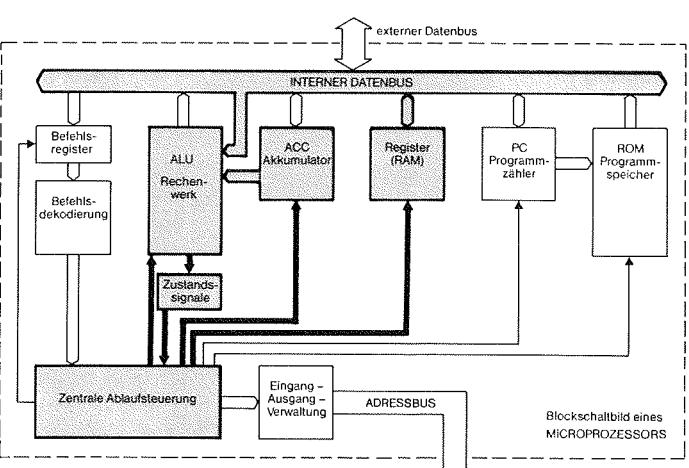
## 2. Der Befehl wird aus dem Programm-Speicher geholt:

Die zentrale Ablaufsteuerung beauftragt das ROM den nächsten Befehl auf den internen Daten-Bus zu legen. Der Befehl wird vom Befehls-Register übernommen und zwischengespeichert. Gleichzeitig wird im Programm-Zähler eine Stelle weitergezählt.



## 3. Der Befehl muß dekodiert werden:

Der Befehls-Dekodierer dekodiert den im Befehls-Register stehenden Befehl und übermittelt ihn an die zentrale Ablaufsteuerung.



## 4. Der Befehl muß ausgeführt werden:

Die zentrale Ablaufsteuerung beauftragt das Rechenwerk, die zwei Zahlen (Werte) zu vergleichen. Gleichzeitig wird der Akkumulator und ein entsprechendes Register im RAM aktiviert, um dem Rechenwerk die zu vergleichenden Zahlen bereit zu stellen. Das Rechenwerk (ALU) sorgt dafür, daß die entsprechenden Zustands-Signale (CARRY- oder ZERO-FLAGS) gesetzt werden.

Zur Ausführung eines einzigen Befehls waren 4 Funktions-schritte notwendig. Die meiste Zeit war der Mikroprozessor mit der Verwaltung seiner Einheiten und mit dem Datentransport beschäftigt.

### 1. Der eingegebene Befehl muß eine fortlaufende Adressen-Nr. erhalten:

Die zentrale Ablaufsteuerung aktiviert den Programm-Zähler, die für diesen Befehl gültige Adressen-Nr. zum Programm-Speicher zu übermitteln.

Die Anzahl der Schritte, die für die Ausführung eines Befehls benötigt werden, nennt man auch Befehls-Zyklus (instruction cycle). Je nach Art des Befehls sind teilweise auch 6 und 8 Funktionsschritte notwendig. Die Ablaufsteuerung wird von einem Takt-Generator gesteuert, damit die einzelnen Funktionsschritte immer in einem gleichbleibenden Rhythmus ausgeführt werden. Der Takt-Generator unseres Mikroprozessors arbeitet mit einer Frequenz von ca. 500 kHz, d. h. daß pro Sekunde 500.000 derartige Einzelschritte ausgeführt werden können.

Alle Funktionen, die der Mikroprozessor ausführt, sind von den Daten abhängig, die im Betriebs-System (ROM-Speicher) festgelegt wurden. Durch Änderung der im ROM-Speicher fest programmierten Daten, kann der gleiche Mikroprozessor für die verschiedensten Aufgaben vorbereitet werden. Der ROM-Speicher des Microtronic-Mikroprozessors enthält ein Monitor-Programm, welches dem Mikroprozessor die Fähigkeit gibt, als Mikrocomputer arbeiten zu können.

Die gesamten beschriebenen Funktionen des Mikroprozessors werden auf einem „Chip“, einem winzigen Plättchen mit ca. 5 x 5 mm ausgeführt. Warum ist der schwarze Mikroprozessor-Block dann 5 cm lang?

Der winzige Chip hat 40 Anschlußstellen, nämlich die 40 Anschlußbeinchen des Mikroprozessors. Die einzelnen Anschlußbeinchen erfordern beim Einlöten auf der Platine einen gewissen Mindestabstand. Deshalb sind die äußeren Abmessungen des Mikroprozessors um ein Vielfaches größer als der in seinem schwarzen Gehäuse eingegossene Mikroprozessor-Chip.

Wenn wir das schwarze Gehäuse öffnen könnten, (ein Versuch wird nicht empfohlen) und wir würden den freigelegten Chip unter einer starken Lupe betrachten, könnten wir die winzigen Goldfäden sehen, welche die einzelnen Funktions-Gruppen des Prozessors mit den außenherum angeordneten Anschlußpunkten verbinden (siehe Foto).

Bei einer nochmaligen Vergrößerung werden die einzelnen Funktions-Gruppen des Mikroprozessor-Chips erkennbar:

Unter einem Mikroskop sieht das Chip ähnlich aus wie eine Platinne, auf welcher die winzigen Bauelemente angeordnet sind. Ein gigantisches elektronisches Schaltwerk mit ca. 35.000 transistorähnlichen Funktionen.

## Speicher und Speicher-Möglichkeiten

Der ROM-Speicher unseres Mikroprozessors ist für das Betriebs-System (Monitor-Programm) zuständig.

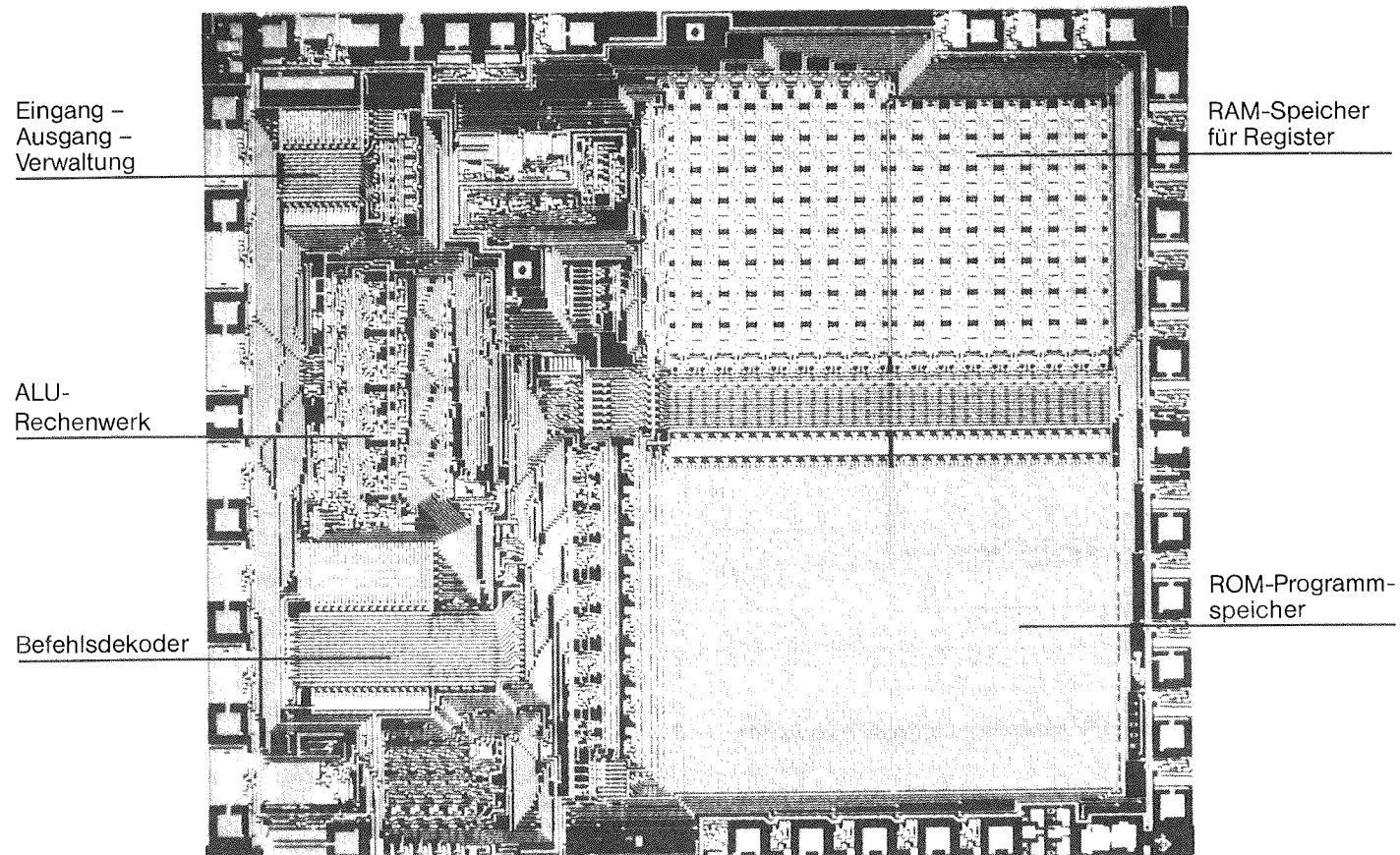
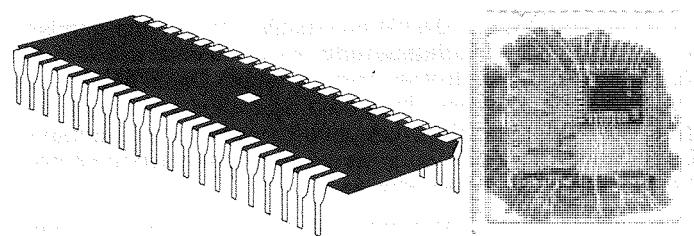
Außerdem ist ein RAM-Speicher für die Register-Inhalte integriert.

Unsere eingegebenen Programme werden in einem separaten RAM-Speicher Baustein (auf der Computer-Platine) gespeichert.

Außer RAM und ROM gibt es auch noch **PROM-Speicher** (programmable read only memory). Die in einem PROM gespeicherten Daten können nicht mehr gelöscht werden. PROM's werden dann eingesetzt, wenn für spezielle Einsatzmöglichkeiten Speicher mit Fest-Programmen in geringen Stückzahlen benötigt werden.

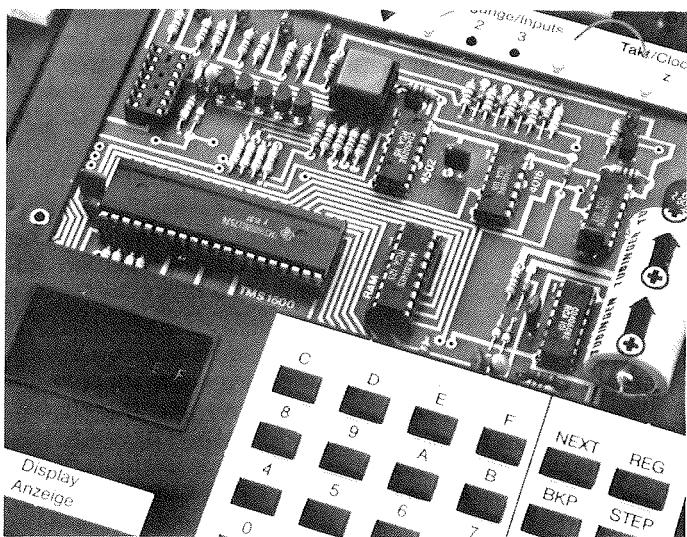
Mit einem **EPROM** (erasable – programmable – ROM) können ähnlich wie im PROM feste Daten oder Programme gespeichert werden. Beim EPROM können diese Daten jedoch mit UV-Licht wieder gelöscht werden und stehen dann für einen neuen Verwendungszweck zur Verfügung. Der kleine Chip innerhalb des EPROM-Bausteins ist durch ein „Fenster“ sichtbar, wodurch die UV-Lichtbestrahlung möglich ist.

RAM-Speicher (z. B. unser Programm-Speicher) verlieren die eingegebenen Daten, sobald die Betriebs-Spannung abgeschaltet wird. Deshalb wurden RAMs entwickelt, welche mit äußerst minimaler Spannung und sehr kleinen Strömen arbeiten. Diese sogenannten **C-MOS-RAMs** können deshalb z. B. von einer Batterie versorgt werden und die eingegebenen Daten über einen längeren Zeitraum speichern.



**Ein spezielles C-MOS-RAM ist auch für unseren Microtronic-Computer lieferbar**, damit die im Computer eingegebenen Programme auch nach dem Abschalten des Netzstroms nicht verloren gehen.

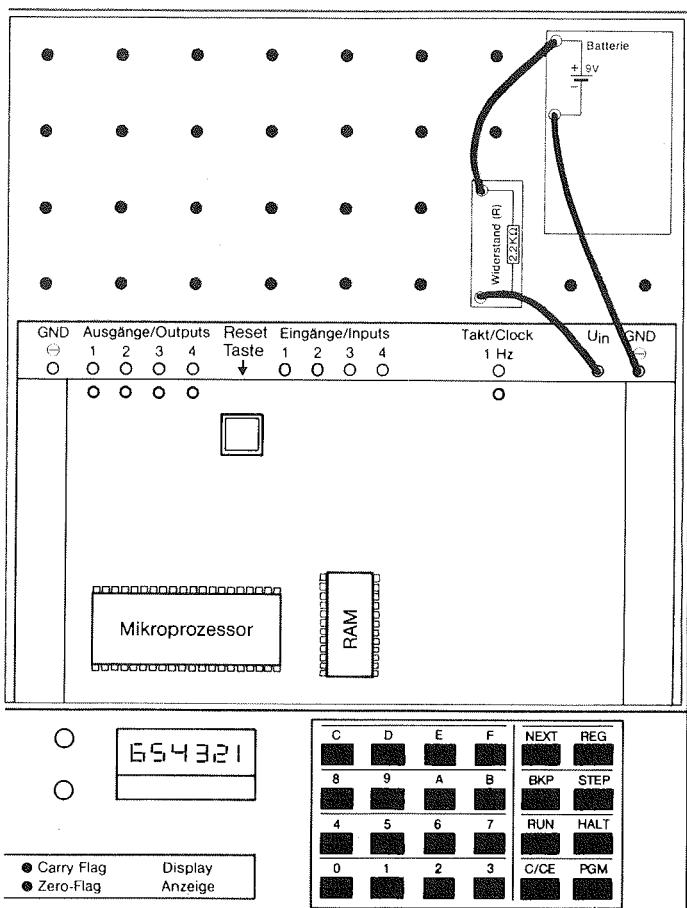
Das auf der Computer-Platine aufgesteckte RAM kann leicht gegen ein C-MOS-RAM ausgetauscht werden.



Aus der Abbildung ersehen wir, daß das RAM rechts neben dem Mikroprozessor zu finden ist. Das RAM ist auf einem mit der Platine verlötzten Sockel aufgesteckt. Es kann (durch Dazwischenschlieben eines kleinen Schraubenziehers) angehoben und gegen das C-MOS-RAM ausgetauscht werden.

Ein speziell für das Microtronic-Computersystem geeignetes C-MOS-RAM kann mit beigelegter Ersatzteil-Preisliste unter der Bestellnummer 20910 (gegen Einzahlung des Betrags auf Postscheck-Konto) angefordert werden.

Die 9V-Batterie kann dann die Spannungsversorgung des C-MOS-RAMS für viele Wochen übernehmen. Unter Zwischenschaltung eines  $2,2\text{ k}\Omega$  Widerstandes (siehe Abbildung) wird die Batterie angeschlossen. Die Verbindungsleitungen dürfen nicht vertauscht werden.



**Vor dem Herausziehen des Netzgeräts aus der Steckdose** ist die grüne RESET-Taste auf dem Computer-Platine ca. 10 Sekunden niederzudrücken. Auch während und nach dem Herausziehen des Netzgeräts die RESET-Taste noch ca. 10 Sekunden in Druckstellung halten.

Der Programm-Speicher wird nun durch die Batterie-Spannung betriebsbereit gehalten, während alle übrigen Computer-Funktionen abgeschaltet sind. Es kann also mit Batteriebetrieb nicht weitergearbeitet werden.

**Vor dem Einsticken des Netzgeräts in die Steckdose** ist erneut die RESET-Taste ca. 10 Sekunden lang zu drücken, dann (bei gedrückter RESET-Taste) das Netzgerät einstecken. Nach dem Einsticken die RESET-Taste noch ca. 10 Sekunden in Druckstellung halten.

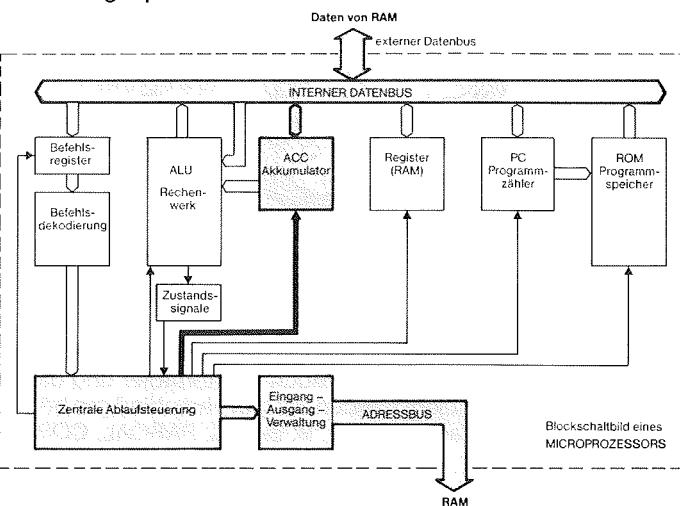
Batterie-Anschlüsse entfernen und der Computer ist wieder wie üblich einsatzbereit. Ein kurzer Programm-Test wird es uns beweisen, daß die eingegebenen Programme noch vorhanden sind.

Die Betätigung der RESET-Taste kurz vor und während der Netzstromunterbrechung (bzw. während der Netztromeinschaltung) ist unbedingt erforderlich, damit keine Daten der eingegebenen Programme verloren gehen. Sollte durch falsche Betätigung der RESET-Taste ein Programm nicht einwandfrei funktionieren (was sich vor allem bei den ersten und letzten Adressen des Programm-Speichers auswirken könnte), müssen einzelne Korrekturen in der bekannten Weise durchgeführt werden.

## Das ROM ist der Dolmetscher unseres Mikroprozessors

Der ROM-Speicher mit dem Betriebs-System und der RAM-Speicher mit den von uns eingegebenen Programmen arbeiten im Mikroprozessor „Hand in Hand“.

Wenn der Computer ein eingegebenes Programm abarbeitet, geschieht folgendes: Sobald wir die RUN-Taste betätigen, wird im ROM-Speicher dessen Programm gestartet. Der Mikroprozessor erhält den Befehl, den ersten Programm-Schritt (die unter Adresse 00 gespeicherte Eingabe) aus dem RAM-Speicher zu holen, der über den Daten-Bus im Akkumulator (ALU) zwischengespeichert wird.



Beginnt ein Befehl z.B. mit der Zahl 4, erkennt der Mikroprozessor durch Vergleichen, daß es sich hier um einen ADD-Befehl handelt und es erfolgt im ROM-Speicher ein Programm-Sprung um den ADD-Befehl auszuführen.

Wie alle Eingabe-Befehle beinhaltet auch z.B. der ADD-Befehl mehrere Einzel-Befehle an den Mikroprozessor:

1. Die beiden zu addierenden Register werden in den Akkumulator (ALU) und in ein zweites Hilfs-Register geladen.
2. Die Addition wird durchgeführt. Das Ergebnis steht im Akkumulator.
3. Das Ergebnis wird zum Ziel-Register (Destination-Register) weitergegeben.

Jeder Programm-Schritt wird durch das Betriebs-System gesteuert und vom Mikroprozessor bearbeitet, bzw. ausgewertet. Die von uns eingegebenen Befehls-Codes sind für den Prozessor nichts anderes, als auszuwertende Daten. Ein Eingabe-Befehl bewirkt, daß verschiedene Einzel-Befehle abgearbeitet werden. Das im ROM-Speicher vorhandene Betriebs-System ist ein „Übersetzungs-Programm“, welches die von uns eingegebenen Befehle in eine für den Mikroprozessor verständliche Sprache übersetzt und bearbeitet.

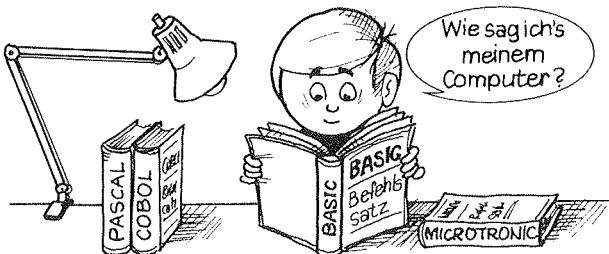
## Der Befehls-Satz ist eine Programmier-Sprache!

Bei einem Programmlauf holt sich der Mikroprozessor nacheinander unsere eingegebenen Befehle aus dem RAM-Speicher. Durch Vergleichen erkennt er den Befehls-Code, z. B. daß eine Addition durchzuführen ist. Im ROM-Speicher steht für jeden Befehl ein Unterprogramm zur Verfügung, welches beim Vergleichen abgearbeitet wird.

Dieses Unterprogramm arbeitet mit der eigentlichen „Mikroprozessor-Sprache“. Unsere Befehls-Codes, mit welchen wir gelernt haben zu addieren, multiplizieren, dividieren oder Werte auf dem Display anzuzeigen, benötigen zur Ausführung eine ganze Reihe einzelner Mikroprozessor-Befehle. Der Microtronic-Befehls-Satz erleichtert die Programmierung, weil durch wenige Eingabe-Werte teilweise sehr umfangreiche Unterprogramme oder ganze Programmteile innerhalb des Mikroprozessors angesprochen werden. Der Befehls-Satz ist eine Programmier-Sprache.

Der Microtronic-Befehls-Satz kann bereits als eine „höhere Programmier-Sprache“ bezeichnet werden. Trotzdem hat der Microtronic-Befehls-Satz eine Ähnlichkeit mit der eigentlichen „Mikroprozessor-Sprache“. Dies ist ein großer Vorteil, wenn z. B. ein Mikroprozessor zu programmieren ist, der nicht über ein so komfortables Betriebs-System verfügt wie in unserem Falle. Eine gedankliche Umstellung ist leicht möglich, weil wir die prinzipielle Logik und Arbeitsweise eines Mikroprozessors durch den Microtronic-Befehls-Satz bereits kennen.

## Es gibt verschiedene Programmier-Sprachen



Bei größeren Computern mit hoher Speicher-Kapazität kann auch die Programmier-Sprache noch komfortabler und damit leicht erlernbar ausgelegt werden. Es gibt verschiedene höhere Programmier-Sprachen wie z. B. BASIC, PASCAL, COBOL usw.

Eine Programmierung in BASIC würde folgendermaßen aussehen:

Eingabe-Befehle	Erklärungen
10 : INPUT A	Einen Wert in Speicher A übernehmen
20 : B = A * A	A x A = B
30 : PRINT B	B wird über Bildschirm oder Drucker ausgegeben
40 : END	Ende des Programmes

Ein Vorteil ergibt sich, weil bestimmte Schlüsselwörter (z. B. INPUT, PRINT etc.) direkt programmiert werden, während wir dem viel kleineren Mikrocomputer einen verschlüsselten Code eingeben.

Da auch Groß-Computer mit Mikroprozessoren arbeiten, muß die höhere Programmier-Sprache durch ein sehr umfangreiches Übersetzungs-Programm in die Mikroprozessor-Sprache zurückverwandelt werden. Der erforderliche Aufwand steigt beträchtlich. Es ist nicht nur eine vollständige Tastatur mit allen Buchstaben, Zahlen und Zeichen erforderlich. Die Überwachung der Tastatur, die Umsetzung aller Buchstaben und Zeichen in für Mikroprozessoren verständliche Daten und das Übersetzungs-Programm in Mikroprozessor-Befehle, benötigen eine hohe Speicher-Kapazität. Daß die umfangreichen Übersetzungen trotz rasanter Arbeitweise der Mikroprozessoren Zeit benötigen, ist einleuchtend. Deshalb werden mitunter auch größere EDV-Anlagen direkt in der Maschinen-Sprache (Mikroprozessor-Sprache) programmiert, wenn z. B. zeitkritische Aufgaben zu lösen sind.

Bei Groß-Computern, die mit der gleichen Programm-Sprache arbeiten, können die Programme ausgetauscht werden. Ein Computer, der z. B. PASCAL versteht, kann Programme übernehmen, die auf einem anderen Computer in PASCAL eingegeben worden sind.

Interessant ist es noch, daß es Alternativen gibt, eine höhere Programmier-Sprache in die Mikroprozessor-Maschinen-Sprache zu übersetzen. Ein „Interpreter“ ist ein Übersetzungs-Programm, um eine Programmier-Sprache in die Maschinen-Sprache zu übersetzen und die einzelnen übersetzten Befehle sofort auszuführen.

Der ROM-Speicher im Microtronic-Computersystem arbeitet ebenfalls als Interpreter. Ein Microtronic-Befehl wird aus dem Speicher geholt, übersetzt und ausgeführt. Erst dann wird der nächste Befehl in gleicher Weise abgearbeitet.

„Compiler“ ist ein Übersetzungs-Programm, mit welchem alle Befehle einer höheren Programmier-Sprache zunächst in die Maschinen-Sprache übersetzt werden. Die Maschinen-Sprachen-Übersetzung wird gespeichert und erst dann wird das gesamte übersetzte Programm ausgeführt.

## Beschreibung der Microtronic-Funktions-Tasten

### HALT

Ende einer Funktion oder Programm-Stop. Wird mit der HALT-Taste ein laufendes Programm angehalten, erscheint auf dem Display die Programm-Adresse und der Befehls-Code.

**Die HALT-Taste ist immer zu betätigen, wenn von einer Funktion auf eine andere Funktion gewechselt wird!** Die HALT-Taste schließt die vorangegangene Funktion ab – eine neue Funktion kann begonnen werden.

### NEXT

Wird zur Programmierung, bzw. Programm-Änderung benötigt. Durch Betätigung der NEXT-Taste wird der nächste Befehl aus dem Programm-Speicher geholt und angezeigt. Dieser angezeigte Befehl kann durch Neueingabe geändert werden. Damit ein Befehl im Programm-Speicher gespeichert wird, muß nach der Befehls-Eingabe die NEXT-Taste betätigt werden.

Wird zuerst die HALT-Taste und dann die NEXT-Taste betätigt, will der Computer wissen, welcher Befehl als nächstes angezeigt oder bearbeitet werden soll. Es ist die 2-stellige Adressen-Nr. einzugeben.

### BKP

(Break Point – Haltepunkt) wird zur Programm-Austestung benötigt. Mit HALT, BKP und der 2-stelligen Adresse wird die Haltepunkt-Adresse eingegeben. Wird das Programm (mit RUN) abgearbeitet, ergibt sich ein Stop bei der Haltepunkt-Adresse. Das Display zeigt die Stop-Adresse und den Befehls-Code. Mit der REG-Taste können nun Register-Werte geändert oder mit der STEP-Taste das Programm ab dieser Stelle in Einzel-Schritten weiterbearbeitet werden. Eine Programm-Fortsetzung ist mit RUN möglich.

Da sich der „Haltepunkt“ auf eine gespeicherte Adresse bezieht, ergibt sich auch bei Änderung der Programm-Eingabe ein Stop auf der Haltepunkt-Adresse. **Es sollte daher niemals vergessen werden, den Haltepunkt zu löschen mit: HALT – BKP – 00 – HALT.**

#### REG

Kontrolle und Änderung der 16 Arbeits-Register (0 bis F). Durch HALT – REG und der Eingabe einer Register-Adresse (0 bis F) erscheint auf dem Display die Register-Adresse und der Register-Inhalt. Wird jetzt eine Ziffern-Taste betätigt, wird dieser Wert in das Register übernommen und angezeigt. Für die Anzeige eines anderen Registers muß wieder die HALT-Taste betätigt werden.

#### C/CE

Bei falscher Eingabe eines Befehls, wird durch einmalige Betätigung die letzte Ziffer des eingegebenen Befehls-Codes gelöscht. Bei zweimaliger Betätigung wird der gesamte (3-stellige) Befehls-Code gelöscht.

#### RUN

Programm-Start-Taste. Der Programm-Ablauf beginnt mit der Adresse, welche momentan auf dem Display angezeigt wird. Soll mit einer anderen Adresse gestartet werden, wird die gewünschte Adresse z. B. 00 (Programm-Anfang) eingegeben.

#### STEP

Bei Betätigung der STEP-Taste wird der auf dem Display angezeigte Befehl ausgeführt und der nächstfolgende Befehl auf dem Display angezeigt. Mit STEP kann ein Programm schrittweise durchgearbeitet und dabei die Flags (CARRY und ZERO-FLAG) beobachtet, bzw. Register-Inhalte kontrolliert werden.

#### PGM

Abruf-Taste für die Fest-Programme, z. B.:

PGM 0 = Test-Programm

PGM 1 = Programme von einem Tonband oder Cassetten-Recorder in den Programm-Speicher überspielen.

PGM 2 = Programme vom Programm-Speicher auf Tonband oder Cassetten-Recorder überspielen.

PGM 3 = Uhrzeit eingeben

PGM 4 = Uhrzeit anzeigen

PGM 5 = Alle Programme im Programm-Speicher löschen (bei sämtlichen Adressen wird der Code 000 angezeigt)

PGM 6 = NOP in den Programm-Speicher laden (auf allen Adressen steht jetzt der Code F01). NOP ist ein Befehl ohne Funktion. Bei den NOP-Adressen können nachträglich andere Befehls-Codes eingegeben werden.

PGM 7 = Nimm-Spiel

**Achtung:** Für PGM 1 und PGM 2 ist ein zusätzliches Cassetten-Interface notwendig. Werden diese Programme ohne Interface aufgerufen, kann der Computer nur durch Betätigung der RESET-Taste wieder funktionsfähig gemacht werden.

#### RESET

Durch Betätigung der grünen RESET-Taste (auf der Computer-Platine) wird das Betriebs-System des Computers aktiviert. Hierdurch werden gespeicherte Register-Werte gelöscht, bzw. auf 0 gesetzt (es geht z. B. die eingegebene Uhrzeit verloren). Eingegebene Programme werden durch die RESET-Taste nicht gelöscht.

## Der Microtronic-Befehls-Satz

Alle Befehle, die ein Mikroprozessor verarbeiten kann, werden „Befehls-Satz“ genannt.

Der Microtronic-Befehls-Satz kann in 6 Gruppen unterteilt werden:

#### Zwei-Adress-Befehle:

Befehls-Code 0 bis A. Die erste Code-Ziffer gibt an, um welchen Befehl es sich handelt. Die beiden nächsten Ziffern geben an, welche Register (bzw. Konstanten) durch den Befehl beeinflußt werden. Diese Befehle haben das Schema:

1. Ziffer Befehl z. B. 6	2. Ziffer <b>s</b> = Quell-Register oder	3. Ziffer <b>d</b> = Ziel-Reg.	z. B. <b>6sd</b>
Befehl z. B. 5	<b>n</b> = Konstante	<b>d</b> = Ziel-Reg.	<b>5nd</b>

Für **s** und **d** werden die Register-Nummern, für **n** der konstante Wert eingegeben.

#### Sprung-Befehle (Verzweigungs-Befehle):

Befehls-Code B bis E. Die erste Code-Ziffer gibt an, um welchen Befehl es sich handelt, auf den beiden nächsten Stellen wird die hexadezimale Sprung-Adresse eingegeben.

1. Ziffer Befehl z. B. C	2. Ziffer Sprung-Adr. = <b>a</b>	3. Ziffer Sprung-Adr. = <b>a</b>	z. B. <b>Caa</b>
-----------------------------	-------------------------------------	-------------------------------------	---------------------

Für **aa** wird die Adressen-Nr. eingegeben.



#### Ein-Adress-Befehle:

Die erste Code-Ziffer ist grundsätzlich **F**. Die zweite Code-Ziffer (**7** bis **C**) gibt an, um welchen Befehl es sich handelt. Die dritte Code-Ziffer **d** betrifft das Register, welches verändert wird.

1. Ziffer Befehl: <b>F</b>	2. Ziffer Befehlsart z.B. 9	3. Ziffer Ziel-Reg. = <b>d</b>	Beispiel <b>F9d</b>
-------------------------------	--------------------------------	-----------------------------------	------------------------

#### Ein- und Ausgabe-Befehle:

Die erste Code-Ziffer ist grundsätzlich **F**. Die zweite Code-Ziffer (**D**-**E** oder **F**) gibt bei DIN, DOT und KIN an, um welchen Befehl es sich handelt. Bei **n** oder **s** wird eingegeben aus welchem oder in welches Register die Daten kommen sollen.

Beispiel: **FDd**, **FEs**, **FFd**

#### Anzeige-Befehl:

Die erste Code-Ziffer ist grundsätzlich **F**. Auf den beiden letzten Codeziffern **n** und **s** werden die auf dem Display anzugegenden Registerstellen eingegeben (siehe DISP-Befehl).

Beispiel: **Fns**

#### Sonder-Befehle:

Die erste und zweite Code-Ziffer sind grundsätzlich immer **F0**. Bei der dritten Code-Ziffer (Eingabe 0 bis F) wird angegeben, um welchen Befehl es sich handelt.

Beispiel: **F00** bis **F0F**

Der Microtronic-Befehls-Satz hat insgesamt 41 verschiedene Befehle:

- 11 Zwei-Adress-Befehle
- 4 Sprung-Befehle
- 6 Ein-Adress-Befehle
- 3 Ein- und Ausgabe-Befehle
- 1 Anzeige-Befehl
- 16 Sonder-Befehle

# Die Microtronic-Einzel-Befehle

Bei den nachfolgend beschriebenen Befehlen ergeben sich u. a. folgende Abkürzungen:

**d** = destination Register oder Ziel-Register. Für d kann eine hexadezimale Zahl als Register-Adresse eingesetzt werden. Das d-Register enthält nach Durchführung des Befehls das Ergebnis.

**s** = source Register oder Quell-Register. Für s kann eine hexadezimale Zahl als Register-Adresse eingesetzt werden. Das s-Register bleibt nach Ausführung des Befehls unverändert.

**n** = Konstante. Für n kann eine hexadezimale Zahl als konstanter Wert eingesetzt werden. Der konstante Wert wird durch Ausführung des Befehls nicht verändert.

**aa** = 2-stellige hexadezimale Adresse bei Sprung- und Verzweigungs-Befehlen.

Bei den Beschreibungen für das **CARRY**- bzw. **ZERO-FLAG** wird **0** angegeben, wenn das Flag zurückgesetzt wird (LED am Ausgang leuchtet nicht) und **1** wenn das Flag gesetzt wird (LED am Ausgang leuchtet). Die Flag-Angaben beziehen sich immer auf den Zeitpunkt sofort nach Durchführung des Befehls.

## Zwei-Adress-Befehle:

Die 2 letzten Stellen des Eingabe-Codes sind variabel.

### MOV = Os<sub>d</sub> (move)

Funktion: Der Inhalt von Register **s** wird in Register **d** geschoben (gespeichert). (Beispiel: **012** = Durch den Befehl **0** wird der Inhalt von register **1** in das Register **2** transportiert).

Carry-Flag: unverändert

Zero-Flag: 0, wenn Inhalt von d = 1 bis F  
1, wenn Inhalt von d = 0

### MOVI = 1n<sub>d</sub> (move immediate)

Funktion: Die Konstante **n** wird in **d** gespeichert.  
(Beispiel: **112** = Durch den Befehl **1** wird konstanter Wert **1** in Register **2** gespeichert)

Carry-Flag: unverändert

Zero-Flag: 0, wenn Inhalt von d = 1 bis F  
1, wenn Inhalt von d = 0

### AND = 2s<sub>d</sub> (and)

Funktion: logische UND-Verknüpfung des Inhaltes von **s** mit dem Inhalt von **d**.

Carry-Flag: 0, wird immer zurückgesetzt

Zero-Flag: 0, wenn Inhalt von d = 1 bis F  
1, wenn Inhalt von d = 0

### ANDI = 3n<sub>d</sub> (and immediate)

Funktion: logische UND-Verknüpfung der Konstante **n** mit dem Inhalt von **d**.

Carry-Flag: 0, wird immer zurückgesetzt.

Zero-Flag: 0, wenn Inhalt von d = 1 bis F  
1, wenn Inhalt von d = 0

### ADD = 4s<sub>d</sub> (Addition)

Funktion: **s + d = d**. Der Inhalt von **s** wird zum Inhalt von **d** addiert.

Carry-Flag: 0 wenn kein Übertrag vorhanden ist  
1 wenn ein Übertrag vorhanden ist (Überlauf)

Zero-Flag: 0, wenn Inhalt von d = 1 bis F  
1, wenn Inhalt von d = 0

### ADDI = 5n<sub>d</sub> (add immediate)

Funktion: **n + d = d**. Die Konstante **n** wird zum Inhalt von **d** addiert.

Carry-Flag: 0 wenn kein Übertrag vorhanden ist  
1 wenn ein Übertrag vorhanden ist (Überlauf)

Zero-Flag: 0, wenn Inhalt von d = 1 bis F  
1, wenn Inhalt von d = 0

### SUB = 6s<sub>d</sub> (Subtraktion)

Funktion: **d - n = d**. Der Inhalt von **s** wird vom Inhalt von **d** abgezogen.

Carry-Flag: 0 wenn kein Übertrag vorhanden ist  
1 wenn ein Übertrag vorhanden ist (Unterlauf)

Zero-Flag: 0, wenn Inhalt von d = 1 bis F  
1, wenn Inhalt von d = 0

### SUBI = 7n<sub>d</sub> (sub immediate)

Funktion: **d - n = d**. Die Konstante **n** wird vom Inhalt von **d** abgezogen.

Carry-Flag: 0 wenn kein Übertrag vorhanden ist  
1 wenn ein Übertrag vorhanden ist (Unterlauf)

Zero-Flag: 0, wenn Inhalt von d = 1 bis F  
1, wenn Inhalt von d = 0

### CMP = 8s<sub>d</sub> (compare)

Funktion: Die Inhalte der Register werden miteinander verglichen. Beide Registerinhalte bleiben unverändert. Die Flags werden entsprechend gesetzt. Für **s** und **d** sind die entsprechenden Register einzusetzen.

Carry-Flag: 0 wenn Inhalt von **s** größer oder gleich dem Inhalt von **d** ( $s \geq d$ )  
1 wenn Inhalt von **s** kleiner als Inhalt von **d** ( $s < d$ )

Zero-Flag: 0 wenn die Register ungleichen Inhalt haben ( $s \neq d$ )  
1 wenn die Register gleichen Inhalt haben ( $s = d$ )

### CMPI = 9n<sub>d</sub> (compare immediate)

Funktion: Die Konstante **n** wird mit dem Inhalt von Register **d** verglichen. Der Registerinhalt bleibt unverändert. Die Flags werden entsprechend gesetzt.

Carry-Flag: 0 wenn **n** größer oder gleich dem Inhalt von **d** ( $n \geq d$ )  
1 wenn **n** kleiner als Inhalt von **d** ( $n < d$ )

Zero-Flag: 0 wenn **n** ungleich dem Registerinhalt **d** ( $n \neq d$ )  
1 wenn **n** gleich dem Registerinhalt **d** ( $n = d$ )

### OR = As<sub>d</sub> (or)

Funktion: logische Oder-Verknüpfung des Inhaltes von **d** mit dem Inhalt von **s**.

Carry-Flag: 0 wird zurückgesetzt

Zero-Flag: 0 wenn Inhalt von d = 1 bis F  
1 wenn Inhalt von d = 0

## Sprung- und Verzweigungsbefehle

Auf den beiden letzten Stellen des Eingabe-Codes (aa) werden die anzuspringenden Adressen eingegeben.

### CALL = Ba<sub>a</sub>

Funktion: Sprung aus einem Hauptprogramm in ein Unterprogramm mit der Startadresse **aa**. Das Unterprogramm ist mit dem RET (F07)-Befehl abschließen.

Wichtig: In einem Unterprogramm darf kein CALL-Befehl vorhanden sein, weil sonst eine „Endlosschleife“ entsteht. Mit anderen Sprungbefehlen kann jedoch in ein anderes Unterprogramm gesprungen werden.

Carry-Flag: unverändert

Zero-Flag: unverändert

**GOTO = Caa**Funktion: Sprung zur Adresse **a a**

Carry-Flag: unverändert

Zero-Flag: unverändert

**BRC = Daa** (branch if carry, Verzweige wenn carry)Funktion: Sprung zur Adresse **aa aa**, wenn das Carry-Flag gesetzt (1) ist.

Carry-Flag: unverändert

Zero-Flag: unverändert

**BRZ = Eaa** (branch if zero, Verzweige wenn zero)Funktion: Sprung zur Adresse **aa aa**, wenn das Zero-Flag gesetzt (1) ist.

Carry-Flag: unverändert

Zero-Flag: unverändert

**Ein-Adress-Befehle:**

Die letzte Stelle des Eingabe-Codes ist variabel.

**MAS = F7d** (move Arbeitsregister in Speicherregister)Funktion: Der Inhalt des Arbeitsregisters **d** wird im Speicherregister innerhalb der gleichen Adresse gespeichert. Der Inhalt von **d** bleibt unverändert

Carry-Flag: unverändert

Zero-Flag: unverändert

**INV = F8d** (inverse)Funktion: Der Inhalt von **d** wird dual invertiert. Aus 1 wird 0, bzw. aus 0 wird 1.

Carry-Flag: 0, wird zurückgesetzt.

Zero-Flag: 0, wenn Inhalt von d = 1 bis F  
1, wenn Inhalt von d = 0**SHR = F9d** (shift right)Funktion: Der Inhalt von **d** wird (dual) nach rechts geschoben.

Carry-Flag: wird mit dem „herausgeschobenen“ Bit geladen

Zero-Flag: 0, wenn Inhalt von d = 1 bis F  
1, wenn Inhalt von d = 0**SHL = FAd** (shift left)Funktion: Der Inhalt von **d** wird (dual) nach links geschoben.

Carry-Flag: wird mit dem „herausgeschobenen“ Bit geladen

Zero-Flag: 0, wenn Inhalt von d = 1 bis F  
1, wenn Inhalt von d = 0**ADC = FBd** (add carry)Funktion: Der Inhalt des Carry-Flag (0 oder 1) wird zum Inhalt von **d** addiert.

Carry-Flag: 0 wenn kein Übertrag vorhanden ist

1 wenn ein Übertrag vorhanden ist (Überlauf)

Zero-Flag: 0, wenn Inhalt von d = 1 bis F  
1, wenn Inhalt von d = 0**SUBC = FCd** (sub carry)Funktion: Der Inhalt des Carry-Flag (0 oder 1) wird vom Inhalt des Registers **d** abgezogen.

Carry-Flag: 0 wenn kein Übertrag vorhanden ist

1 wenn ein Übertrag vorhanden ist (Überlauf)

Zero-Flag: 0, wenn Inhalt von d = 1 bis F  
1, wenn Inhalt von d = 0**Ein- und Ausgabe-Befehle**

Die letzte Stelle des Eingabe-Codes ist variabel.

**DIN = FDd** (data in)Funktion: Die an den vier Eingängen vorhandene Information wird im Register **d** gespeichert.

Wichtig: Auf richtige Beschaltung der Eingänge achten

Carry-Flag: 0 wird zurückgesetzt

Zero-Flag: 0, wenn Inhalt von d = 1 bis F  
1, wenn Inhalt von d = 0**DOT = FE<sub>s</sub>** (data out)Funktion: Der Inhalt von **s** wird an die Ausgänge gebracht und der Wert dual durch die 4 LEDs angezeigt. Die Information bleibt an den Ausgängen bis zum nächsten DOT-Befehl stehen.

Carry-Flag: 0, wird zurückgesetzt

Zero-Flag: 0, wenn Inhalt von d = 1 bis F  
1, wenn Inhalt von d = 0**KIN = FFd** (keyboard in)Funktion: KIN bewirkt eine Programmunterbrechung bis eine Ziffer über die Tastatur eingegeben wird. Der Wert dieser Ziffer (0 bis F) wird in **d** gespeichert.

Carry-Flag: 0, wird zurückgesetzt

Zero-Flag: 0, wenn Inhalt von d = 1 bis F  
1, wenn Inhalt von d = 0**Anzeige-Befehl**

Die beiden letzten Stellen des Eingabe-Codes sind variabel.

**DISP = Fn<sub>s</sub>** (display)Funktion: Der Inhalt von **s** wird in der rechten Anzeigestelle angezeigt. **n** gibt an, wieviele Register (Stellen) angezeigt werden sollen.Beispiel: DISP 3,2 ergibt Eingabe-Code: **F 3 2****F** = Befehl auf dem Display Werte anzuzeigen.  
**3** = 3 Register (oder 3 Stellen) auf dem Display anzeigen.**2** = Anzeige beginnt **ab Register 2**. Durch die vorgestellte **3** werden **3 Register** also Reg. Nr. 2-3-4 angezeigt.Wichtig: **n** muß eine Zahl zwischen 1 und 6 sein. **n + s** darf höchstens F (15) sein. Überschreitet **n** den Wert F, werden die überzähligen Register mit Reg. 0 beginnend mit angezeigt.

Carry-Flag: unverändert

Zero-Flag: unverändert

**Sonder-Befehle**

Nicht veränderbarer Befehls-Code. Keine variable Eingabe-Möglichkeit

**HALT = F00**

Funktion: Das Programm wird angehalten. In der Anzeige erscheint die Haltestelle und der Code F00. Ein erneuter Programmstart ist möglich, wenn zuerst die STEP und dann die RUN-Taste betätigt wird.

Carry-Flag: unverändert

Zero-Flag: unverändert

**NOP = F01** (no operation)

Funktion: Es wird lediglich die Adresse um „1“ erhöht. Der Befehl hat keine Funktion. Wird als „blinder Befehl“ eingesetzt, der nachträglich durch andere Befehle ersetzt werden kann.

Carry-Flag: unverändert

Zero-Flag: unverändert

**DISOUT = F02** (display out)

Funktion: Das Display wird abgeschaltet. Hierdurch wird die Geschwindigkeit der Befehlausführung erhöht.

Carry-Flag: unverändert

Zero-Flag: unverändert

**HDXZ = F03** (Hexadezimalzahl in Dezimalzahl)  
Funktion: Die in Reg. D, Reg. E und Reg. F vorhandenen dreistelligen hexadezimalen Ziffern werden in Dezimalzahlen umgewandelt. Das Ergebnis steht wieder in den drei Registern. In Reg. D steht die Einerstelle, in Reg. E die Zehnerstelle und in Reg. F die Hunderterstelle. Der Befehl arbeitet grundsätzlich immer mit den Registern D, E und F.  
Wichtig: Die Hexadezimalzahl darf nicht größer als 3E7 sein!  
Carry-Flag: 0 wird zurückgesetzt  
Zero-Flag: 0, wenn die Hex-Zahl nicht größer als 3E7 ist 1, wenn die Hex-Zahl größer als 3E7 ist (Bereichsüberschreitung).  
Es können sich Zeitfehler beim Uhrenprogramm ergeben.

**DZHX = F04** (Dezimalzahl in Hexadezimalzahl)  
Funktion: Die in Reg. D, Reg. E und Reg. F vorhandenen dreistelligen Dezimalzahlen werden in Hexadezimalzahlen umgewandelt. Das Ergebnis steht wieder in diesen drei Registern. In Reg. D steht die Einerstelle, in Reg. E die Zehnerstelle, in Reg. F die Hunderterstelle. Der Befehl arbeitet grundsätzlich immer mit den Registern D, E und F.  
Carry-Flag: 0, wird zurückgesetzt  
Zero-Flag: 0, wird zurückgesetzt  
Es können sich Zeitfehler beim Uhrenprogramm ergeben.

**RND = F05** („random“-Zufallsgenerator)  
Funktion: Durch den RND-Befehl wird der Wert eines internen Zählers als Zufallsgenerator in die Reg. D, Reg. E und Reg. F übernommen.  
Carry-Flag: unverändert  
Zero-Flag: unverändert

**TIME = F06** (Uhrzeit)  
Funktion: Die im Betriebssystem laufende Uhr wird in die Reg. A bis Reg. F geladen. Die Uhrzeit kann mit PGM 3 eingegeben werden. Der Takt-/Clock-Ausgang muß mit dem Eingang 4 verbunden sein. Nach Ausführung des TIME-Befehls steht die Zeit wie folgt in den Registern:  
Reg. A 1er-Sekunden  
Reg. B 10er-Sekunden  
Reg. C 1er-Minuten  
Reg. D 10er-Minuten  
Reg. E 1er-Stunden  
Reg. F 10er-Stunden

Wichtig: Die gültige Uhrzeit wird über den TIME-Befehl lediglich in die Reg. A bis Reg. F geladen. Die dort stehende Zeit ändert sich erst durch einen neuen TIME-Befehl.  
Carry-Flag: unverändert  
Zero-Flag: unverändert

**RET = F07** (return)  
Funktion: Dieser Befehl steht am Ende eines Unterprogrammes (siehe CALL). Durch RET erfolgt ein Rücksprung in das Hauptprogramm zur Adresse a + 1 des CALL-Befehls.  
Carry-Flag: unverändert  
Zero-Flag: unverändert

**CLEAR = F08**  
Funktion: Die Arbeitsregister werden auf 0 gesetzt (Löschenbefehl). Die Speicherregister sind davon nicht betroffen!  
Carry-Flag: 0, wird zurückgesetzt  
Zero-Flag: 1, wird gesetzt

**STC = F09** (set carry-flag)  
Funktion: Das Carry-Flag kann an jeder beliebigen Programm-Stelle gesetzt werden. (Carry-LED leuchtet).  
Carry-Flag: 1, wird gesetzt  
Zero-Flag: unverändert

**RSC = F0A** (reset carry-flag)  
Funktion: Ein gesetztes Carry-Flag kann an jeder beliebigen Programm-Stelle auf 0 zurückgesetzt werden.  
Carry-Flag: 0, wird zurückgesetzt  
Zero-Flag: unverändert

**MULT = F0B** (multiplizieren)  
Funktion: Es werden die Arbeits-Register 0 bis 5 (6-Stellen) mit den entsprechenden Speicherregistern dezimal multipliziert. Das Ergebnis steht alsdann wieder in den Arbeitsregistern. Die niedrigwertigste Zahl (Einerstelle) steht in Reg. 0, die höchswertige in Reg. 5 (unbedingt bei Eingabe beachten). Bei Überlauf steht „E“ in allen Registern 0 bis 5  
Wichtig: Sollen nur z. B. 3-stellige Zahlen multipliziert werden, müssen die höherwertigen Register (Arbeits- oder Speicherregister) bis Reg. 5 mit 0 aufgefüllt werden.  
Carry-Flag: 0, wenn Ergebnis korrekt  
1, wenn Überlauf (Ergebnis größer als 6-Stellen) oder bei Eingabe einer Hexadezimalen Zahl (A bis F)  
Zero-Flag: 0, wird zurückgesetzt.  
Es können sich Zeitfehler beim Uhrenprogramm ergeben.

**DIV = F0C** (Division)  
Funktion: Es werden die Speicherregister 0 bis 3 durch die Arbeitsregister 0-3 (4-Stellen) dezimal dividiert. Das Ergebnis steht im Arbeitsregister, evtl. Restwerte im Speicherregister. Besteht ein Rechenwert nur aus 0000 erfolgt Abbruch der Division und in Reg. 0 bis Reg. 5 steht E.  
Wichtig: In den Registern 4 und 5 muß im Arbeitsregister und Speicherregister der Wert 0 stehen, obwohl diese Register für die Division nicht benötigt werden.  
Carry-Flag: 0, wenn Ergebnis korrekt  
1, wenn ein Zahlenwert 0 ist (es erfolgt keine Division), oder bei Eingabe einer hexadezimalen Zahl (A bis F)  
Zero-Flag: 0, wenn kein Rest vorhanden ist  
1, wenn ein Rest vorhanden ist  
Es können sich Zeitfehler beim Uhrenprogramm ergeben.

**EXRL = F0D** (exchange least significant register)  
Funktion: Die Inhalte der Arbeitsregister 0 bis 7 werden mit den Speicherregistern 0 - 7 getauscht.  
Carry-Flag: unverändert  
Zero-Flag: unverändert

**EXRM = F0E** (exchange most significant register)  
Funktion: Die Inhalte der Arbeitsregister 8 bis F werden mit den Speicherregistern 8 bis F getauscht  
Carry-Flag: unverändert  
Zero-Flag: unverändert

**EXRA = F0F** (exchange Arbeitsregister)  
Funktion: Die Inhalte der Arbeits-Register 0 bis 7 werden mit den Inhalten der Arbeits-Register 8 bis F getauscht. Der Inhalt von Reg. 0 steht dann in Reg. 8, der Inhalt von Reg. 8 in Reg. 0. Die Inhalte der Register 1 und 9; 2 und A; usw. werden ebenfalls entsprechend getauscht.  
Carry-Flag: unverändert  
Zero-Flag: unverändert

Im 2. Teil des Anleitungsbuches ergeben sich teilweise sehr lange Programme, welche neu eingegeben werden müssen, wenn die Netzstrom-Versorgung zum Computer unterbrochen wird.

Da auch die Speicher-Kapazität auf 256 Programm-Schritte begrenzt ist, wird durch Eingabe eines sehr langen Programms ein anderes Programm gelöscht.

Durch das in Vorbereitung befindliche Cassetten-Interface kann der Inhalt des Programm-Speichers auf eine Cassette oder Tonband überspielt werden. Auf diese Weise ist es möglich ein umfangreiches Programm-Archiv anzulegen und außerhalb des Computers zu speichern. Diese „ausgelagerten“ Programme können jederzeit mit Hilfe des Cassetten-Interface wieder in den Programm-Speicher zurückgespielt werden und stehen damit „auf Abruf“ für eine erneute Programm-Abarbeitung zur Verfügung. Das Aus- oder Einspielen der Programme wird innerhalb weniger Minuten durchgeführt.

## **Das Microtronic-Computersystem wird zum Sprach-Computer**

Ein Zusatz-Modul, mit welchem ein Ausbau zum sprechenden Computer möglich wird, befindet sich ebenfalls in Vorbereitung. Ähnlich wie beim Cassetten-Interface werden solche Zusatz-Ausrüstungen durch leicht herstellbare Steckverbindungen durchgeführt.

Erfragen Sie bitte zu gegebener Zeit beim einschlägigen Fachhandel oder direkt bei uns die Liefermöglichkeiten.

Wenn sie uns die dem Computersystem beigegebene Registrierungs-Karte übersenden, erhalten Sie automatisch Informationen über alle Zusatz-Einrichtungen.

**Mikroprozessor:** 4-bit Prozessor TMS 1600

**Taktfrequenz:** 500 kHz

**Speicher:** im Mikroprozessor integriertes ROM mit 4096 Bytes Monitorprogramm, und RAM mit 64 Bytes sowie externes RAM mit 512 Bytes.

**Befehlssatz:** Speziell entwickelter microtronic-Befehlssatz mit über 40 Einzelbefehlen, den eigentlichen Mikroprozessorbefehlen ähnlich aufgebaut. Durch die komplexeren Einzelbefehle (z. B. Displaybefehl, Multiplikations-Divisionsbefehle usw.) wird das Programmieren wesentlich vereinfacht.

**Tastatur:** 16 Hexadezimal-Tasten  
8 Funktionstasten  
2 freie Sonderfunktionstasten

**Display-Anzeige:** 6-stellige Siebensegment-Multifunktionsanzeige. 2 LED's zur Anzeige von Carry- und Zero-Flags, 4 LED's für Zustandsanzeige der Datenausgänge. 1 LED zur Takt-Funktionsanzeige.

**Ausgang:** 4-Bit-Datenausgang zur direkten Ansteuerung von IC's, Transistoren, BUSCH-micro-electronic Schaltrelais 5964 usw. (kein zusätzliches Interface notwendig). Pegel 5 V, TTL – kompatibel

**Eingang:** 4-Bit-Dateneingang zur direkten Ansteuerung durch IC's, Transistoren usw. (kein zusätzliches Interface notwendig, Überlastgeschützt). Pegel 5 V (max. 9 V), TTL-kompatibel.

**Normbuchsen:** 2 Buchsen für handelsübliche Stereo-Überspielkabel für Cassettenrecorder und externe Geräte-Anschlüsse.

**Integrierter Taktgenerator:** 1 Hz Quarzgesteuerter Taktgenerator zur Steuerung von Uhren oder zeitabhängigen Schaltungen.

**Signaltongeber:** Piezo-Summer durch Programmierung ansteuerbar.

**Integrierte Festprogramme:** Digital-Uhr (mit Stunden-, Minuten- und Sekunden-Leuchtanzeige), Zufallsgenerator, Interface-Steuerung, Hexadezimal-/Dezimal-Converter, Nimm-Spiel u. a.

**Stromversorgung:** Eingebautes Netzgerät 220/10 Volt, Sicherheitstransformator mit Spannungsregelung nach VDE-Vorschriften geprüft.

**Im Preis enthaltenes Zubehör:** Umfangreiches Anleitungsbuch, Programmierformulare, sowie einige Elektronik-Elemente auf BUSCH-Steckbausteinen montiert, zum Aufbau einfacher Peripherie-Schaltungen.

**Zusätzliche Erweiterungsmöglichkeit:** microtronic-Cassetten-Interface Nr. 2095 für die direkte Programmspeicherung mit handelsüblichem Cassetten-Recorder oder Tonbandgerät.

Zum schnellen Aufbau elektronischer Peripherie-Geräte werden die BUSCH Electronic-Studios 2060, 2070 bzw. 2075 empfohlen. Bitte Prospekte anfordern.

# Verzeichnis der Einzelteile des Microtronic-Computersystems 2090

	Einzelbestell-Nummer
1 Anleitungsbuch, 1. Teil	20900
1 Anleitungsbuch, 2. Teil	20901
20 Programm-Tabellen (zum Selbstprogrammieren)	20904
1 Armaturenboard fertig bestückt mit Computer-Tastatur, 2 Sondertasten mit angeschlossenen Tasterbausteinen, 2 Überspiel-Buchsen mit angeschlossenen Überspiel-Bausteinen, rote Filter-Glasscheibe	20905
1 Computer-Platine mit sämtl. Bauelementen	20906
1 Steckernetzgerät 220/10 V, 250 mA	20907
1 Baustein: Piezo-Summer	20908
1 Baustein: NPN-Transistor	20611
1 Baustein: Widerstand 47 $\Omega$	20682
1 Baustein: Widerstand 1 k $\Omega$	20685
2 Bausteine: Widerstand 4,7 k $\Omega$	20687
1 Baustein: Batteriehalter	20780
2 schwarze Steckplatten 180 x 180 mm	20782
1 Gehäuse-Unterteil	20796
1 Abdeckhaube, rauchglasfarbig	20798
22 Kabel-Abschnitte: 7 rot – 4 cm lang 3 grün – 6 cm lang 6 gelb – 10 cm lang 1 braun – 18 cm lang 5 grau – 35 cm lang	
32 Stück gelbe Plastikstecker	

## Ergänzungsmöglichkeiten (nicht im Lieferumfang enthalten)

Niedervolt-Schaltrelais (für Experimente im Bereich 9-16 V, z. B. Modellbahn-Steuerungen usw.)	5964
Netzstrom-Schaltgerät: Mit dem Computer 220 V Netzstrom schalten (ca. Januar 1982 lieferbar)	2087
Cassetten-Interface für Programm-Überspielung auf Tonband oder Cassetten-Recorder (voraussichtlich ab März 82 lieferbar)	2095
C-MOS-RAM mit Vorwiderstand, für Programm-Speicherung mit Batterie-Betrieb	20910
Für evtl. Bestellungen fordern Sie bitte kostenloses Bestellformular (mit Preisangaben) an.	

# Sachwort-Verzeichnis

**Adresse:** Bezeichnung für eine Speichereinheit, in welcher bestimmte Daten abgelegt sind, z. B. Befehle im Programm-Speicher, Inhalte von Registern, usw.

**Akkumulator (ACC, accumulator):** Das „Haupt-Register“ eines Mikroprozessors. Im Akkumulator steht nach einer Rechen-Operation das Ergebnis.

**Algorithmus:** Festgelegte Reihenfolge von Rechen-Schritten, um ein bestimmtes Ergebnis zu erzielen, z. B. um bei Spielen eine Gewinnposition zu erreichen.

**ALU (arithmetic and logic unit):** Rechenwerk, Teil eines Mikroprozessors oder eines Computers, in welchem die Rechenvorgänge, Vergleiche usw. ausgeführt werden.

**Befehl:** Anweisung für den Computer eine entsprechende Operation auszuführen.

**Befehls-Satz (instruction set):** Summe aller Befehle, die von einem Computer ausgeführt werden können. Wichtige Befehlsarten eines Befehls-Satzes sind:

1. Arithmetische und logische Befehle (z. B. ADD, SUB, AND usw.)
2. Befehle zur Steuerung des Programm-Ablaufes (z. B. GOTO, HALT usw.)
3. Befehle zum Datentransport (z. B. MOV usw.)

**Betriebs-System (Monitor-Programm):** Steuer-Programm für die wichtigsten Grundaufgaben eines Computers, wie z. B. Tastatur-Eingaben überwachen, Werte anzeigen, Speicher-Verwaltung usw.

**Bit (binary digit):** Kleinste Speicher-Einheit eines Computers, bzw. einer EDV.

**Byte:** 8 Bit ergeben 1 Byte.

**Bus:** Verbindungsleitung, welche gleichzeitig (bzw. nacheinander) für mehrere angeschlossene Komponenten verwendet wird. Der „Adress-Bus“ dient der Übermittlung von Adressen (z. B. welche Adressen von welchem Speicher-Baustein an den Computer übermittelt werden sollen). Der Daten-Bus dient zur Daten-Übertragung.

**Carry-Flag:** Ein vom Mikroprozessor erzeugtes Signal, welches dann entsteht, wenn sich bei Rechen-Operationen ein Übertrag ergibt, oder wenn zwei zu vergleichende Ziffern ungleich sind.

**Chip:** Ein Silizium-Plättchen, nur wenige qmm groß, welches durch ein spezielles Verfahren (Diffusion) eine komplette elektronische Schaltung enthält.

**CPU (central processing unit):** Zentral-Einheit eines Computers. Die CPU umfaßt ein Rechenwerk (ALU), eine zentrale Ablaufsteuerung, den Programm-Zähler und verschiedene Register. Die CPU ist das eigentliche „Gehirn“ eines Computers. Die moderne Technik hat es ermöglicht, eine gesamte CPU auf einem winzigen Halbleiter-Chip zu integrieren. Ein CPU-Chip wird auch als Mikroprozessor bezeichnet.

**Destination-Register:** Ziel-Register, in welchem die Ergebnisse abgespeichert werden. (siehe auch Source-Register).

**Dezimal-Korrektor (dezimal adjustment):** Notwendige Programm-Schritte, um aus hexadezimalen Berechnungen ein dezimales Ergebnis zu erhalten.

**Dezimal-System:** Zahlen-System bestehend aus den Ziffern 0 bis 9. Übliches Zahlen-System für allgemeine Rechenaufgaben.

**Display:** Vorrichtung zur Anzeige von Zahlen, Buchstaben und Zeichen, z. B. Leuchtziffern-Anzeige, Flüssigkristall-Anzeige, Bildschirm usw.

**Dual-System:** Besteht lediglich aus den beiden Ziffern 0 und 1. Alle Computer-Systeme arbeiten grundsätzlich nur mit diesen beiden Ziffern, weil hierdurch die Darstellung der beiden Zustände „Spannung vorhanden“ (high) oder „keine Spannung vorhanden“ (low) möglich ist, (high = 1, low = 0).

**Hardware:** Bezeichnung für die elektronischen Komponenten und Bauteile eines Computers. Siehe auch Software.

**Hexadezimal-System:** Besteht aus 16 Ziffern (den Zahlen 0 bis 9 und den Buchstaben A bis F). Speziell in der Mikroprozessor-Technik und Datenverarbeitung übliches Zahlen-System.

**High:** Bezeichnung für den Zustand, daß an einer bestimmten Stelle einer elektronischen Schaltung „eine Spannung vorhanden“ ist. Der Zustand „high“ wird auch durch die Ziffer 1 angezeigt. (Siehe auch „low“ und Dual-System).

**Interface:** Elektronische Schnittstelle (Übergangsstelle), um zwei oder mehrere Geräte elektronisch miteinander zu verbinden, z. B. Computer mit Cassetten-Recorder. Durch die in einem Interface vorhandene elektronische Schaltung werden die Signale eines Geräts so umgewandelt, damit diese Signale von einem anderen Gerät übernommen werden können. (Z. B. Cassetten-Interface wandelt Daten in akustische Signale).

**Low:** Bezeichnung für den Zustand, daß z. B. an einer bestimmten Stelle einer elektronischen Schaltung „keine Spannung vorhanden“ ist. Der Zustand „low“ wird auch durch die Ziffer 0 angezeigt. (Siehe auch „high“ und Dual-System).

**Mikroprozessor:** Ein IC (integrated circuit), welcher die Zentraleinheit (CPU) eines Computers enthält.

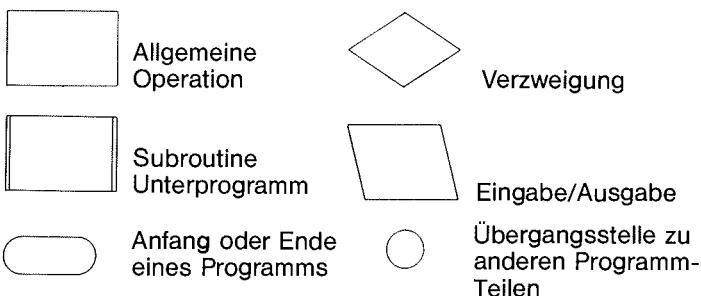
**Mnemonic:** Bezeichnung für eine Kurzform, welche einen Hinweis auf die Art eines Code-Wortes gibt (z. B. ADD für Addition, MULT für Multiplikation usw.)

**Monitor-Programm:** Siehe Betriebs-System.

**Programm:** Eine Folge von Befehlen, zur Lösung einer Aufgabe durch einen Computer.

**Programm-Ablaufplan (Fluß-Diagramm, flow-chart):** Stellt den Funktionsablauf eines Computer-Programmes dar. Die einzelnen Operationen werden in verschiedenartige Symbol-Bilder eingetragen und durch Ablauflinien miteinander verbunden. Programm-Ablaufpläne sind unentbehrliches Hilfsmittel zum Entwickeln und Verfolgen von Computer-Programmen.

Symbole für Programm-Ablaufpläne nach DIN 66001:



> bedeutet: größer

< bedeutet: kleiner

= bedeutet: gleicher Wert

**Programm-Schrittzähler:** Eine Einheit einer CPU. Der Programm-Schrittzähler enthält die Adresse des auszuführenden Befehls. Er wird daher auch als Befehls-Zähler (program-counter, instruction counter) bezeichnet.

**RAM (random access memory):** Ein sogenannter „Schreib- und Lese-Speicher“. In einem RAM-Speicher können Daten eingegeben und herausgelesen werden. Das RAM wird deshalb auch als „Arbeits-Speicher“ bezeichnet. (Siehe auch ROM).

**Register:** Speicher zur Aufnahme sehr kleiner Informationen, z. B. einer Ziffer oder eines Zeichens. Register werden zur Zwischen-Speicherung von Ziffern verwendet.

**ROM (read only memory):** Ein sogenannter „Nur-Lese-Speicher“. In einem ROM-Speicher sind Daten fest eingegeben (gespeichert) und können später nicht mehr verändert werden. Die fest eingegebenen Daten können jederzeit „ausgegeben“ werden, es ist jedoch nicht möglich, neue Daten einzugeben. (Siehe auch RAM).

**Software:** Bezeichnung für nicht „sicht- und greifbare“ Computer-Funktionen, wie z. B. Programme, Betriebs-System usw.

**Source-Register:** Quell-Register, aus welchem die Daten für eine weitere Bearbeitung übernommen werden. (Siehe auch Destination-Register).

**Speicher:** Elektronische Möglichkeit zur Aufbewahrung von Daten. So werden z. B. „high“- oder „low“-Signale in Halbleiter-Bausteinen (siehe z. B. RAM oder ROM) oder auf Magnetbändern, bzw. Magnetplatten gespeichert.

**Zero-Flag:** Ein vom Mikroprozessor erzeugtes Signal, welches auftritt, wenn zwei zu vergleichende Ziffern gleich sind, oder wenn eine Ziffer 0 ist.

# Inhaltsverzeichnis

## Anleitungsbuch: Erster Teil

### Einführung in die Mikroprozessor- und Computer-Technik

	Seite	
Der Computer – das unbekannte Wesen	3	Mit Batterie und C-MOS-RAM Programme speichern
Start frei zum ersten Probelauf	4	Das ROM ist der Dolmetscher unseres Mikroprozessors
Der Computer testet sich selbst – das Prüfprogramm	4	Der Befehls-Satz ist eine Programmier-Sprache!
Ein guter Vergleich: Computer – Mensch?	6	Es gibt verschiedene Programmier-Sprachen
Der Computer als Spiele-Partner: Das Nimm-Spiel	7	Beschreibung der Microtronic Funktions-Tasten
Der Computer wird zur Digital-Leucht-Uhr	8	Der Microtronic Befehls-Satz
Die ersten Computer-Kenntnisse	9	Die Microtronic Einzel-Befehle
Computer-Programm – was ist das eigentlich?	10	Mit Cassetten-Recorder Programme speichern
Unser erstes Programm: Ein elektronischer Würfel	10	Technische Beschreibung
Der Computer soll automatisch zählen	12	Bauelemente-Verzeichnis
Etwas über Speicher und Adressen	13	Sachwort-Verzeichnis
Wir lernen die ersten Befehle	13	
Was sind „Register“?	14	
Wie kann der Computer mehrere Stellen anzeigen	14	
Mnemonics – und anderes Computer-chinesisch	15	
Dezimal – hexadezimal – dual	16	
Der Computer lernt rechnen	17	
Der rasende Automatik-Zähler	18	
Ein 2-stelliger Automatik-Zähler	19	
Ein 3-stelliger Automatik-Zähler	19	
Unser erstes selbst programmiertes Programm	20	
Wir wollen Register-Inhalte löschen	21	
Wir „verschieben“ Register-Werte	22	
„Die Mondlandung“ – ein interessantes Computer-Spiel	23	
Probieren und experimentieren?!	25	
Vergleichen – ein wichtiger Befehl!	25	
Der Programm-Ablaufplan (Fluß-Diagramm, flow chart)	27	
Aufgabe für ein selbst zu entwickelndes Programm	28	
2-stelliger dezimaler Rechner	29	
Wir löschen den gesamten Programm-Speicher!	31	
6-stellige Addition – wie beim Taschenrechner!	31	
Software und Hardware	34	
Bus-Verbindungen für den Datentransport?	34	
Nach Addition folgt Subtraktion	35	
DIN und DOT	36	
Timer – der Computer als Zeit-Schalter	38	
Kontroll-Hilfe: Die Funktions-Taste BKP	41	<b>Anleitungsbuch: Zweiter Teil</b>
Der Zufalls-Generator	41	<b>Eine interessante Auswahl großer Computer-Programme</b>
Der Zufalls-Generator ermittelt Lottozahlen	42	(Siehe separates Anleitungsbuch)
Die Basis der Zahlen-Systeme	42	
Umwandlung: Hexadezimal in dezimal und umgekehrt	43	<b>Computer-Spiele</b>
Das Microtronic Betriebs-System	43	
Was bewirkt das Monitor-Programm des Mikro- prozessors?	43	Tic-Tac-Toe
Time – der Zeit-Befehl!	43	See-Schlacht
Der Computer als Weck-Uhr	44	Code-Brecher (Denk- und Geschicklichkeits-Spiel)
Immer neue Möglichkeiten mit 0 und 1	44	Nimm 2
Die Funktions-Tasten REG und STEP	45	
Selbst-Programmierung eines langsamen Lauflichts	47	<b>Computer-Funktionen mit Decoder-Aufgaben</b>
Durch Halbieren nach rechts verschieben?	48	
Aus 0 wird 1 – aus 1 wird 0!	49	Morse-Decoder
Subtraktion mit Angabe von negativen Ergebnissen	50	Widerstands-Code-Decoder
Dezimale Subtraktion mit Minus-Anzeige	51	
Mit „high“ und „low“ an die Computer-Eingänge	54	<b>Computer-Berechnungen</b>
Reaktions-Test für 2 Personen	54	
Die „Rucksack-Programmierung“	55	Taschenrechner-Programm mit 4 Grundrechenarten
Reaktionstest Programm „de luxe“	56	Sinus-Berechnung (etwas für Mathematiker)
Bei Netzstrom-Unterbrechung die eingegebenen Programme im Programm-Speicher halten	57	Berechnung der Tage zwischen zwei Daten
„Bit“ und „Byte“	58	Aus eingegebenen Daten einen Wochentag berechnen
Logische Operationen	58	Berechnung des persönlichen Bio-Rhythmus
Hat ein Bit einen Wert?	59	
Arbeits-Register – Speicher-Register	60	<b>Computer-Funktionen mit Peripherie-Elektronik</b>
Die Multiplikation	61	
Die Division	61	Für diese Experimente sind zusätzliche BUSCH-Schaltrelais, bzw. BUSCH-Electronic-Studios erforderlich.
Die letzten 4 Befehle – dann können wir alles!	62	
Mit CALL in ein Unterprogramm springen!	62	Anschluß von Relais
Wie wird ein Computer-Spiel programmiert?	63	Der Computer als Schaltuhr (viele Variationsmöglichkeiten)
Wir programmieren selbst!	63	Computer-Orgel
Das „Innenleben“ eines Computers	64	Der komponierende Computer
Speicher und Speicher-Möglichkeiten	68	Reaktionszeit-Meßgerät
	68	Der Computer zählt Personen und Gegenstände
	70	Daten-Übertragung
		Der Computer zählt Frequenzen
		Digital-Voltmeter
		Computergesteuerte Modelleisenbahn

# 2090

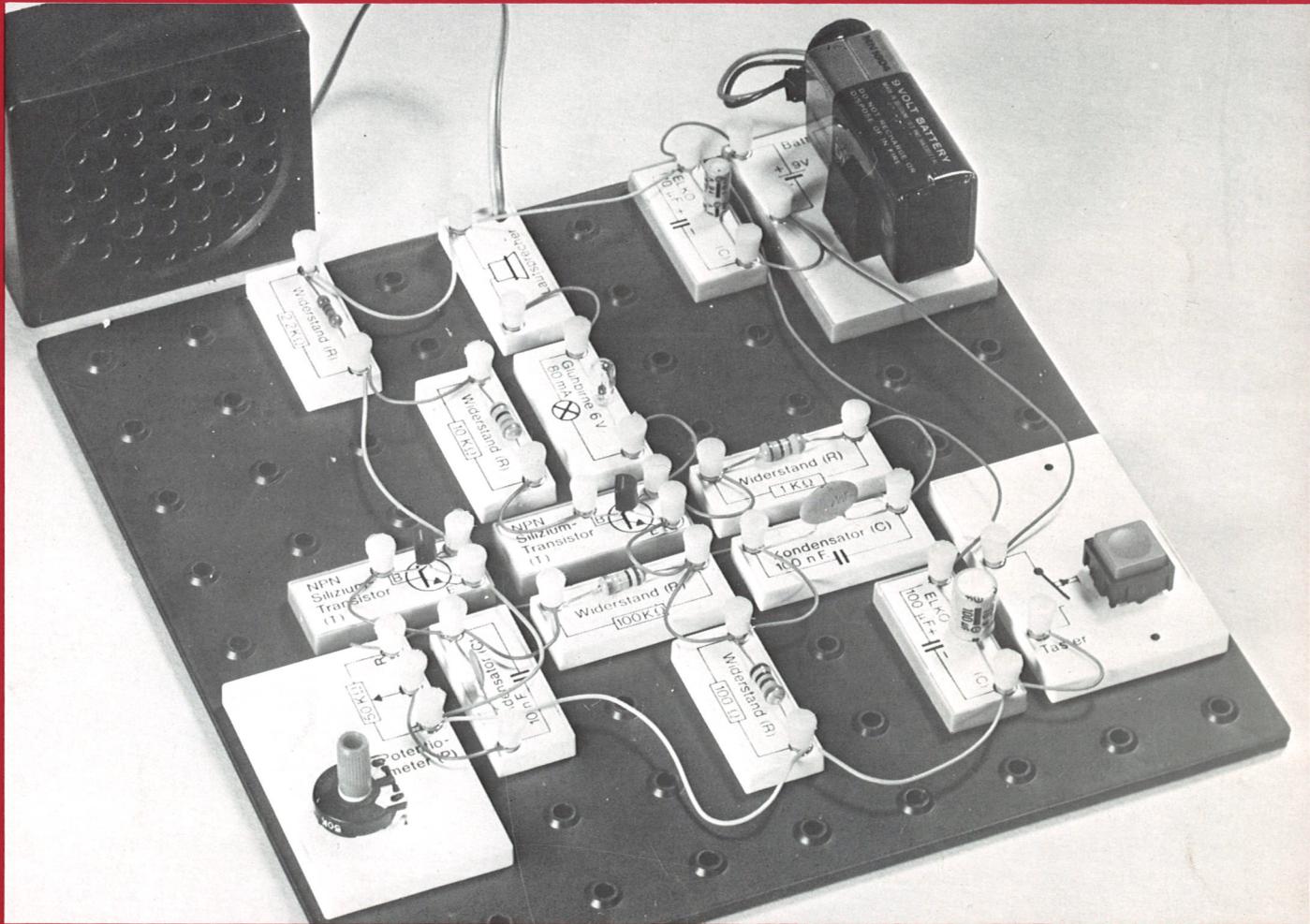


In Zusammenarbeit mit  
dem Elektronik-Magazin



**Mehr wissen als andere!**

BUSCH Electronic-Studios zeigen die gesamte Elektronik-Palette. Vom einfachen Stromkreis, über Blinkschaltungen bis zum Mikrocomputer.



Man kann klein beginnen, z. B.  
mit dem Compact-Studio 2060,  
mit ca. 40 interessanten Experi-  
menten und Geräteschaltungen.